Wireless M-Bus Host Controller Interface

Specification

Document ID: 4100/6404/0050

IMST GmbH

Carl-Friedrich-Gauß-Str. 2-4

47475 KAMP-LINTFORT

GERMANY

Document Information

File name	WMBus_HCI_Spec.docx
Created	2011-06-24
Total pages	57

Revision History

Version	Note
0.1	Created, Initial Version
0.2	Draft Version Created For Review
0.5	Preliminary Version
0.6	UART baudrate : 57600 bps
1.0	Reviewed and released
1.1	AES-128 messages added
1.2	AES Decryption Error : payload added
1.3	USB Stick characteristics included, default parameter changed
1.4	C-Mode added
1.5	Configuration parameter added
1.6	Added Example Code for Host Controller

Aim of this Document

This document describes the Host Controller Interface of the Wireless M-Bus Stack which is available for the entire Wireless M-Bus product family, including:

- iM871A Radio Module
- iM871A USB Stick
- iM170A Radio Module

Table of Contents

1. Introduction

1.1 Overview

The WM-Bus HCI Protocol is designed to expose the WM-Bus Stack Services to an external Host Controller. The communication between Host and WM-Bus Stack is based on so called HCI Messages which can be sent through a UART interface (see Fig.1). The WM-Bus Firmware provides many services for configuration, control and Radio Link access.

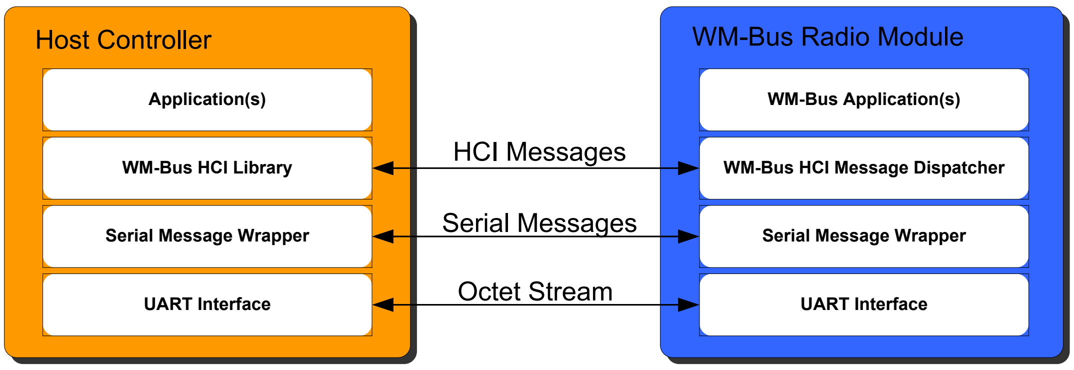

Fig. 1-1: Host Controller Communication

Document Guide

Chapter 2 explains the message flow between Host Controller and WM-Bus Module and describes the general message format.

Chapter 3 gives a detailed summary of the services which can be accessed via HCI.

2. HCI Communication

The communication between the WM-Bus Module and a Host Controller is message based. 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 WM-Bus Module:

1. Command Message: always sent from the Host Controller to the WM-Bus Module to trigger a function.

2. Response Message: sent from the WM-Bus Module to the Host Controller to answer a preceding HCI Command Message.

3. Event Message: can be sent from the WM-Bus Module to the Host Controller at any time to indicate an event or to pass messages which were received over the radio link.

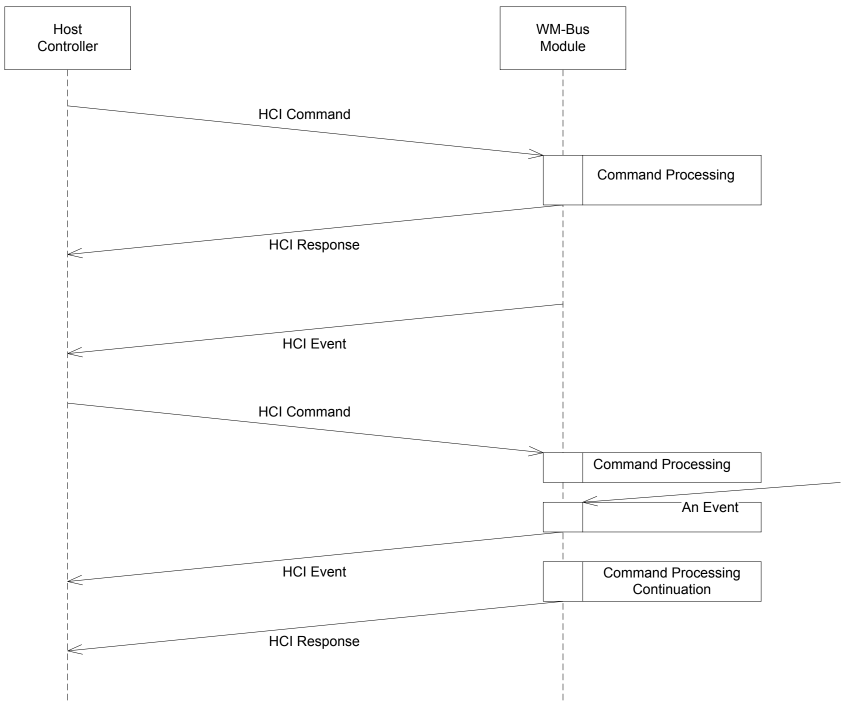

Fig. 2-1: HCI Message Flow

2.2 HCI Message Format

The communication between the WM-Bus module and a host controller is realized by means of the following message format.

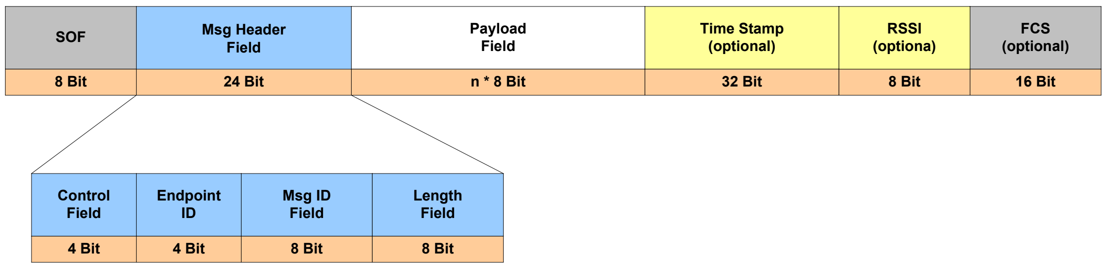

Fig. 2-2: HCI Message Format

This message format is used to call services and to exchange information over a UART interface. The message transmission starts with the SOF Field (**S**tart **O**f **F**rame = 0xA5) which is used to synchronize the octet stream. A message always contains a Header Field with fixed elements and fixed length. The following Payload Field, Time Stamp Field, RSSI Field and FCS Field are optional and will be indicated in the Header Field.

2.3 Elements of the HCI Message

This chapter describes the message format in detail.

2.3.1 Control Field (4 bits)

The Control Field is part of the Message Header and is used to indicate which of the optional fields are present in the current message. A bit which is set to 1 means, that the corresponding field is attached.

The Control Field coding is as follows:

Coding	Description
0000b	Reserved
0010b	Time Stamp Field attached
0100b	RSSI Field attached
1000b	CRC16 Field attached

2.3.2 Endpoint ID Field (4 Bit)

This field identifies a logical message endpoint which groups several messages.

2.3.3 Message ID Field (8 Bit)

This field identifies the message type itself.

2.3.4 Length Field (8 Bit)

The Length Field contains the number of octets which are present in the payload field. If the value is zero, the payload field is empty.

2.3.5 Payload Field (n Octets)

The Payload Field contains the message dependent data. The length of this field is variable and indicated by the Length Field.

2.3.6 Time Stamp Field (32 Bit)

The configurable Time Stamp Field contains a 32 Bit Time Stamp which is derived from the embedded Real Time Clock (RTC). The Time Stamp can be attached for every received radio link message which is passed to the Host Controller. The Time Stamp is optional and must be signaled in the Control Field.

2.3.7 RSSI Field (8 Bit)

The configurable RSSI Field includes an estimated Receive Signal Strength Indicator for every received radio link message which is sent to the Host. The RSSI Field is optional and must be signaled in the Control Field.

2.3.8 FCS Field (16 Bit)

The Frame Check Sequence Field (FCS) contains a 16-Bit CRC-CCITT cyclic redundancy check which enables the receiver to check a message for bit errors. The CRC computation starts from the Control Field and ends with the last octet of the Payload Field or Time Stamp Field or RSSI Field. The FCS Field is optional and must be indicated in the Control Field.

2.4 Physical Parameters

The standard HCI interface is a UART interface with the following settings:

57600 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
- Device Information
- Device Configuration
- Factory Reset
- System Operation Modes
- System Status
- Firmware Information
- Real Time Clock Support
- Host controlled Power Saving Support

3.1.1 Ping

This command is used to check if the connected WM-Bus Module is alive. The sender should expect a Ping Response within a certain time interval.

Message Flow

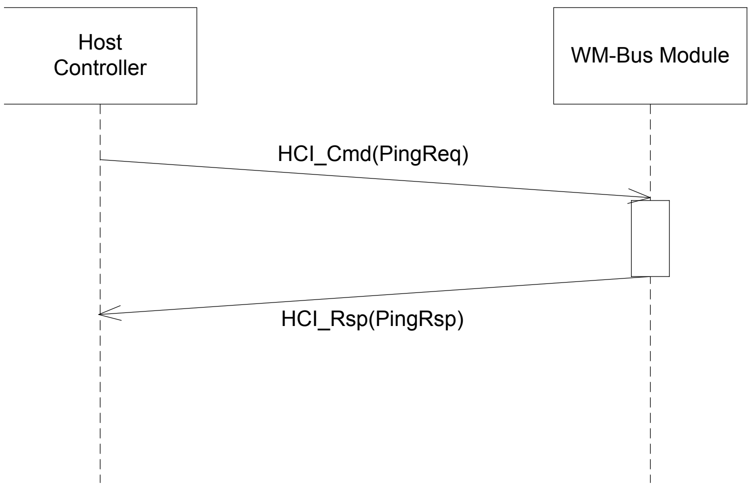

Fig. 3-1: Ping Request

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	0	No Payload

3.1.2 Reset

This message can be used to reset the WM-Bus Module. The reset will be performed after approx. 500ms.

Message Flow

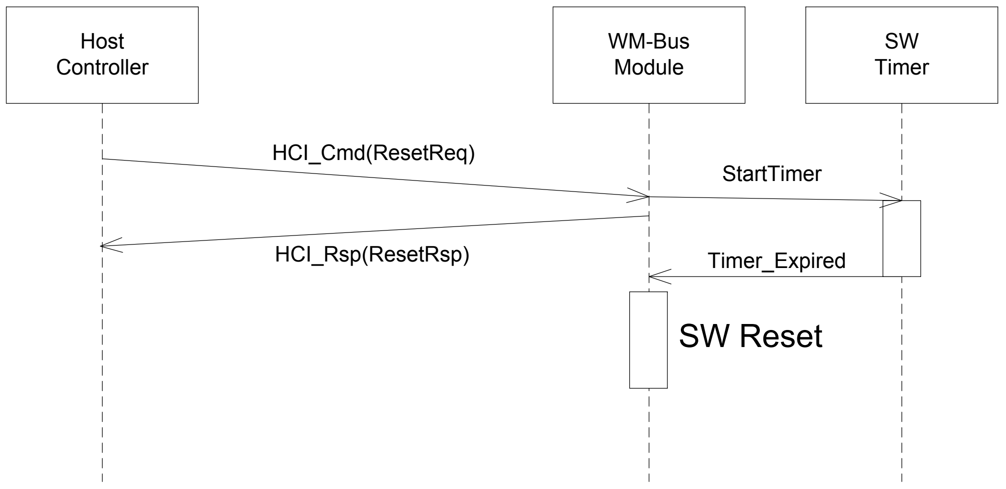

Fig. 3-2: Reset Request

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	0	No Payload

3.1.3 Device Information

The WM-Bus Firmware provides a service to readout some information elements for identification purposes.

3.1.3.1 Get Device Information

This service 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_DEVICEINFO_REQ	Get Device Info Request
Length	0	No Payload

Response Message

The response message contains the Device Information Field:

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_DEVICEINFO_RSP	Get Device Info Response
Length	8	8 Octets
Payload	Device Information Field	

3.1.3.2 Device Information Field

The Device Information Field contains the following elements:

Offset	Size	Name	Description
0	1	ModuleType	Identifies the Radio Module 0x33 = iM871A 0x36 = iM170A
1	1	Device Mode	Indicates the current Device Mode 0x00 = Other 0x01 = Meter
2	1	Firmware Version	Firmware Version 0x13 = V1.3
3	1	HCI Protcol Version	HCI Protocol Version 0x01
4 – 7	4	32 Bit Device ID	Unique Device ID

3.1.4 Device Configuration

The WM-Bus Firmware supports several kinds of configurable parameters which are stored in the non volatile flash memory. The configuration parameters are readout during startup and used to configure the firmware components and hardware units. The following items can be configured:

Item	Description
Device Mode	Determines if the module operates in Meter or Other Mode.
Link Mode	Determines one of the following radio link modes: S1, S1-m, S2, T1, T2, R2, C1, C2, N1A, N2A, N1B, N2B, N1C, N2C, N1D, N2D, N1E, N2E, N1F, N2F
WM-Bus Header Fields *	Fixed elements of the M-Bus Message Header, can be used from internal configuration memory for radio link access to reduce HCI communication.
Radio Channel	Selectable Radio Channel for R2 Mode
Radio Power Level	Radio Output Power
Automatic Power Saving	Enables the module to enter the low power mode as soon as possible without Host Controller interaction.
Radio Rx-Window	Defines a time interval for reception of radio messages in Meter Mode
Rx-Timestamp Attachment	Enables the firmware to generate an RTC timestamp for every received radio message. The timestamp will be attached to the HCI message when the radio message is passed to the Host Controller.
RSSI Attachment	Configures the firmware to attach the RSSI value for a received radio message when it is passed to the Host Controller.
LED Control	Enables the firmware to control several LEDs for internal events. LED1 – Alive Indicator: indicates if the module is in low power mode (off) or not (on) LED2 – Tx Indicator: this LED is toggled for every transmitted radio message. LED3 – Rx Indicator: this LED is toggled for every received radio message with valid CRC.
RTC Control	Controls the Real Time Clock which can be used to determine the operating hours or to generate Rx-Timestamps.

* these parameters are read-only in USB Stick variant

3.1.4.1 Get Device Configuration

This function can be used to readout the configuration parameters.

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_CONFIG_REQ	Get Config. Request
Length	0	No Payload

Response Message

The response message contains the Device Configuration Field which is described below.

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_CONFIG_RSP	Get Config. Response
Length	n	n octets
Payload	Device Parameter Field	List of configured Items

3.1.4.2 Set Device Configuration

This function can be used to change several system 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_CONFIG_REQ	Set Config Request
Length	n	variable length
Payload[0]	Store NVM Flag 0x00 : change configuration only temporary 0x01 : save configuration also in NVM	Non-Volatile Memory Flag
Payload [1..n-1]	variable Device Parameter List	List of Configuration Items

Response Message

This message acknowledges the Get Config Request message.

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_CONFIG_RSP	Get Config Response
Length	0	No Payload

3.1.4.3 Device Parameter List

The Device Parameter List contains the so called Information Indicator Flags which indicate, if a configuration parameter is present or not. A bit which is set to 1 means, that the corresponding parameter is included.

The device parameter list contains has the following layout:

Offset	Size	Name	Description	
0	1	IIFlag 1	Information Indicator Flag for first group of parameters: Bit 0 : Device Mode Bit 1 : Radio Mode Bit 2 : WM-Bus C Field Bit 3 : WM-Bus Man ID Bit 4 : WM-Bus Device ID Bit 5 : WM-Bus Version Bit 6 : WM-Bus Device Type Bit 7 : Radio Channel	
1	1	Device Mode	0x00 : Other 0x01 : Meter	
variable	1	Link Mode	iM871A: 0 : S1 1 : S1-m 2 : S2 3 : T1 4 : T2 5 : R2 6 : C1, Telegram Format A 7 : C1, Telegram Format B 8 : C2, Telegram Format A 9 : C2, Telegram Format B iM170A: 10 : N1A 11 : N2A 12 : N1B 13 : N2B 14 : N1C 15 : N2C 16 : N1D 17 : N2D 18 : N1E 19 : N2E 20 : N1F 21 : N2F	
variable	1	WM-Bus C Field	C Field, used in WM-Bus Radio Messages	
variable	2	WM-Bus Man ID	Manufacturer ID, used in WM-Bus Radio Messages	RO*
variable	4	WM-Bus Device ID	Device ID, used in WM-Bus Radio Messages	RO*
variable	1	WM-Bus Version	Version, used in WM-Bus Radio Messages	RO*
variable	1	WM-Bus Device Type	Device Type, used in WM-Bus Radio Messages	RO*

variable	1	Radio Channel (iM871A only)	RF Channel used in R2 Mode : 1 : 868.09 MHz (R-Mode) 2 : 868.15 MHz (R-Mode) 3 : 868.21 MHz (R-Mode) 4 : 868.27 MHz (R-Mode) 5 : 868.33 MHz (R-Mode) 6 : 868.39 MHz (R-Mode) 7 : 868.45 MHz (R-Mode) 8 : 868.51 MHz (R-Mode) 9 : 868.57 MHz (R-Mode) 10 : 868.30 MHz (S-Mode) 11 : 868.95 MHz (T-Mode)	
variable	1	IIFlag 2	Information Indicator Flag for second group of parameters: Bit 0 : Radio Power Level Bit 1 : Radio Data Rate Bit 2 : Radio Rx-Window Bit 3 : Auto Power Saving Bit 4 : Auto RSSI Attachment Bit 5: Auto Rx-Timestamp Attachment Bit 6: LED Control Bit 7: RTC Control	
variable	1	Radio Power Level	iM871A: 0 : -8 dBm 1 : -5 dBm 2 : -2 dBm 3 : 1 dBm 4 : 4 dBm 5 : 7 dBm 6 : 10 dBm 7 : 14 dBm iM170A: 0 : 1dBm 1 : 10 dBm 2 : 20 dBm	
variable	1	Radio Data Rate	Radio Data Rate, reserved for future use	
variable	1	Radio Rx-Window	Reception Window [ms] after Transmit: The module will listen for radio messages for the given time before it enters a power saving state. This parameter is useful especially for battery powered devices (Meters) which are configured for bidirectional Radio communication (S2, T2, R2, C2, N2x)	

variable	1	Auto Power Saving	Automatic Power Saving Management: 0 : off 1 : device enters power saving mode after message transmission (S1, S1-m, T1, C1, N1x), reception or when the Radio Rx Window terminates (S2, T2, R2, C2, N2x).	
variable	1	Auto RSSI Attachment	This flag controls the automatic RSSI output: 0 : no RSSI output 1 : RSSI output for each received Radio message	
variable	1	Auto Rx-Timestamp Attachment	This flag controls the automatic Rx-Timestamp output: 0 : no output 1 : Rx-Timestamp attached for each received Radio message	
variable	1	LED Control	Three LEDs can be selected independently by setting the corresponding bit. Bit 0 : LED1 - System Alive indicator Bit 1 : LED2 - Radio message transmitted Bit 2 : LED3 - Radio message received	
variable	1	RTC Control	0 : RTC off 1 : RTC enabled	

* this parameter is read-only in USB Stick variant

3.1.4.4 Default Configuration iM871A

The following table lists the default configuration for the iM871A.

Parameter	Value
Device Mode	Other
Link Mode	S2
WM-Bus C Field	0x00
WM-Bus Man ID	Starter Kit : 0x0CAE
	USB Stick : 0x25B3
WM-Bus Device ID	Starter Kit : 0x12345678
	USB Stick: <preconfigured address>
WM-Bus Version	0x01
WM-Bus Device Type	0x00
RF Power Level	7 : 14dBm
RF Channel	1 : 868.09 MHz (R-Mode)
Radio Rx Window	50 = 50ms
Auto Power Saving	0 : none
Auto RSSI Attachment	0 : not attached
Auto Rx Time Stamp Attachment	0 : not attached
LED Control	0 : disabled
RTC	0 : off

3.1.4.5 Default Configuration iM170A

The following table lists the default configuration for the iM170A.

Parameter	Value
Device Mode	Other
Link Mode	N2C
WM-Bus C Field	0x00
WM-Bus Man ID	Starter Kit : 0x0CAE
WM-Bus Device ID	Starter Kit : 0x12345678
WM-Bus Version	0x01
WM-Bus Device Type	0x00
RF Power Level	2 : 20dBm
RF Channel	n.a.
Radio Rx Window	0 = 0ms (off)
Auto Power Saving	0 : none
Auto RSSI Attachment	0 : not attached
Auto Rx Time Stamp Attachment	0 : not attached
LED Control	0 : disabled
RTC	0 : off

3.1.4.6 Factory Reset

This function can be used to reset the WM-Bus Module configuration to its default factory settings.

Note: The new configuration gets active after reboot.

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_FACTORY_RESET_REQ	Factory Reset Request
Length	1	1 Octet
Payload[0]	Reboot Flag 0x00 : no reboot 0x01 : reboot device	Reboot option

Response Message

This message acknowledges the Factory Reset Request message.

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_FACTORY_RESET_RSP	Factory Reset Response
Length	1	1 Octet
Payload[0]	Status 0x00 = Operation failed 0x01 = Operation successful	

3.1.5 System Operation Modes

The WM-Bus firmware can operate in different System Operation Modes. The operation modes enable the device to align its behaviour according to a given use case e.g. test mode, application mode. The System Operation Mode is determined during firmware start-up and requires a reset to get changed.

3.1.5.1 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	1	1 Octet
Payload[0]	Current System Operation Mode	

3.1.5.2 Set System Operation Mode

This message sets the next System Operation Mode and performs a firmware reset.

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	

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 0x00 = Operation failed 0x01 = Operation successful	

3.1.5.3 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

3.1.6 System Status

The firmware provides several status values. Some values are only determined during system startup while the others are updated continuously. All values are maintained in RAM and not stored in the non-volatile memory.

3.1.6.1 Get System Status

This message is used to read the system status.

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_SYSSTATUS_REQ	Get System Status Request
Length	0	No Payload

Response Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_SYSSTATUS_RSP	Get System Status Response
Length	38	38 Octets
Payload	System Status Field	

3.1.6.2 System Status Field Details

The System Status Field contains the following information elements:

Offset	**Size**	**Name**	**Description**
0	1	NVM Status	0 : no error else : NVM corrupt or not yet configured Bits representation: Bit 0: Configuration Data Bit 1: Production Data Bit 2: AES Data Bit 7: FFS Consistency Note : this item is only updated during system startup

1	1	Reserved	
2 – 5	4	System Tick	System Ticks with 10 ms resolution, updated continuously when the module is not in a power saving state.
6 – 9	4	Reserved	
10 – 13	4	Reserved	
14 – 17	4	NumTxFrames	Number of transmitted radio messages
18 – 21	4	NumTxErrors	Number of not transmitted radio messages
22 – 25	4	NumRxFrames	Number of received radio messages
26 – 29	4	NumRxCRCErrors	Number of received CRC errors (radio link)
30 – 33	4	NumRxPhyErrors	Number of received decoding errors
34 – 37	4	Reserved	

3.1.7 Firmware Information

The WM-Bus Module supports several information elements to identify the firmware. The single information elements can be readout message per message by sending an appropriated key.

3.1.7.1 Get Firmware Info

This message is used to readout one specific information element.

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_FWINFO_REQ	Set Operation Mode Request
Length	1	1 Octet
Payload[0]	Key	Key for information element

Response Message

This message contains the requested information element.

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_GET_FWINFO_RSP	Set Operation Mode Response
Length	variable	
Payload	Firmware Information Field	Information element

3.1.7.2 Firmware Information Elements

Firmware Version, Key = 0x00

Offset	Size	Name	Description
0	1	Key	Requested type of information
1	1	Length	Length of FW information
2	1	Version	Firmware Version e.g. 0x13 = V1.3
3 – 4	2	Build Version	Build Version e.g. 107

Firmware Name, Key = 0x01

Offset	Size	Name	Description
0	1	Key	Requested type of information
1	1	String Length	Length of Firmware Name
2 - N+1	N	String	Firmware Name

Date String, Key = 0x02

Offset	Size	Name	Description
0	1	Key	Requested type of information
1	1	String Length	Length of Date String
2 - N+1	N	String	Date of firmware build

Time String, Key = 0x03

Offset	Size	Name	Description
0	1	Key	Requested type of information
1	1	String Length	Length of Time String
2 - N+1	N	String	Time of firmware build

3.1.8 Real Time Clock (RTC) Support

The WM-Bus 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. The RTC time can be read and set at any time. The RTC is reset to zero during system startup. For usage the RTC needs to be enabled (see chapter ***Device Configuration***).

3.1.8.1 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	4	4 Octets
Payload	32 Bit time value with 32768Hz resolution	Information element

3.1.8.2 Set RTC Time

This message can be used to set 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 value request
Length	4	4 Octets
Payload	32 Bit time value with 32768Hz resolution	New time

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 value response
Length	1	1 Octet
Payload[0]	Status 0x00 Operation failed 0x01 Operation successful	

3.1.9 Host controlled Power Saving

In addition to the automatic power saving feature the firmware provides a command to enter the low power mode. The LPM mode will be left with every new HCI message.

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_ENTER_LPM_REQ	Enter LPM request
Length	1	1 Octet
Payload[0]	Mode = 0x00	Low Power Mode

Response Message

This message acknowledges the LPM request and is sent before the module enters the low power mode.

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_ENTER_LPM_RSP	Enter LPM response
Length	1	1 Octet
Payload[0]	Status 0x00 Operation failed 0x01 Operation successful	

3.1.10 AES-128 Encryption / Decryption

The firmware supports automatic AES-128 encryption and decryption of radio link messages. This service is optional and maybe not available in all firmware versions.

3.1.10.1 Set AES-128 Encryption Key

This function can be used to change the AES-128 encryption key which is used for packet transmission. The function allows to change the encryption key directly and to save it optionally in the non-volatile flash memory.

Note: this message doesn't enable the AES encryption service.

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_SET_AES_ENCKEY_REQ	Set AES Encryption Key request
Length	17	17 Octets
Payload[0]	Store NVM Flag 0x00 : change configuration only temporary 0x01 : save configuration also in NVM	Non-Volatile Memory Flag
Payload [1..16]	AES-128 bit key	AES-128 key

Response Message

This message acknowledges the Set AES-128 Key message.

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_SET_AES_ENCKEY_RSP	Set AES Encryption Key response
Length	1	1 Octet
Payload[0]	Status 0x00 Operation failed 0x01 Operation successful	

3.1.10.2 Enable / Disable AES-128 Encryption

This message can be used to enable the automatic AES-128 encryption. This message allows to change the AES encryption state directly and to save it optionally in the non-volatile flash memory.

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_ENABLE_AES_ENCKEY_RE Q	Enable AES Encryption
Length	2	2 Octets
Payload[0]	Store NVM Flag 0x00 : change configuration only temporary 0x01 : save configuration also in NVM	Non-Volatile Memory Flag
Payload[1]	Activation Flag 0x00 : disable AES 0x01 : enable AES	Activation Flag

Response Message

This message acknowledges the Enable AES-128 encryption message.

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_ENABLE_AES_ENCKEY_RSP	Enable AES Encryption response
Length	1	1 Octet
Payload[0]	Status 0x00 Operation failed 0x01 Operation successful	

3.1.10.3 Set AES-128 Decryption Key

This function can be used to change the AES-128 decryption key which is used for packet reception. The function sets the decryption key for multiple WM-Bus Devices in volatile memory. The keys and corresponding WM-Bus Device Address Filters are stored in a table in volatile memory (RAM). During packet reception the decryption key will be selected from that table according to the received WM-Bus Device Address. If the decryption process fails as a result of an invalid key, an error message will be sent to the host (see AES Decryption Error Indication).

Note: This message enables the AES decryption service for a given WM-Bus Device Address.

Command Message

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_SET_AES_DECKEY_REQ	Set AES Decryption Key request
Length	25	25 Octets
Payload[0]	Table index: 0 – max. table index	Table Index (1 Octet)
Payload [1..8]	WM-Bus Man ID (2 Octets) WM-Bus Device ID (4 Octets) WM–Bus Version (1 Octet) WM-Bus Device Type (1 Octet)	WM-Bus Device Filter (8 Octets)
Payload [9..24]	AES-128 bit decryption key	AES-128 decryption key (16 Octets)

Note: The maximum table index is firmware specific. The table can be cleared by means of a system reset.

Response Message

This message acknowledges the Set AES-128 Decryption Key message.

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_SET_AES_DECKEY_RSP	Set AES Decryption Key response
Length	1	1 Octet
Payload[0]	Status 0x00 Operation failed (index exceeds table) 0x01 Operation successful	

3.1.10.4 AES Decryption Error Indication

This message is sent to the host in case of a failed packet decryption. It indicates that the AES encryption key used on sender side and the AES decryption key used on receiver side are not the same.

Field	Content	Description
Endpoint ID	DEVMGMT_ID	Endpoint Identifier
Msg ID	DEVMGMT_MSG_AES_DEC_ERROR_IND	AES Decryption Error Indication
Length	9	9 Octets
Payload [0-8]	WM-Bus C-Field (1 Octet) WM-Bus Man ID (2 Octets) WM-Bus Device ID (4 Octets) WM-Bus Version (1 Octet) WM-Bus Device Type (1 Octet)	WM-Bus Header of received packet

3.2 Radio Link Services

The Radio Link endpoint provides services for transmission and reception of radio link messages according to EN 13757 part 4.

3.2.1 WM-Bus Message Request

This command can be used to send an M-Bus message containing header and payload via radio link. The first octet of the HCI payload is expected to be the C- Field of the M-Bus message. The CRC16 of each M-Bus Data Block and the M-Bus Length Field will be calculated and inserted by the firmware itself.

The following figure shows the relationship between an HCI message and the resulting M-Bus message which is sent via radio link. The message in this example consists of two M-Bus Data Blocks.

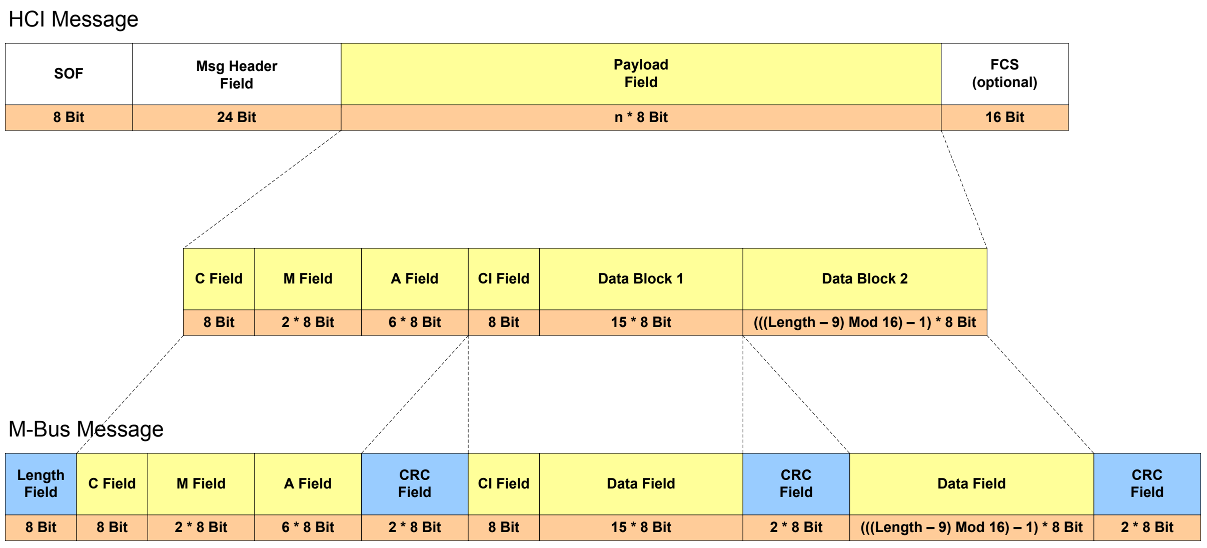

Fig. 3-3: HCI and M-Bus message (Telegram Format A)

Command Message

Field	Content	Description
Endpoint ID	RADIOLINK_ID	Endpoint Identifier
Msg ID	RADIOLINK_MSG_WMBUSMSG_REQ	Send WM-Bus message request
Length	n	n Octets
Payload	WM-Bus message, starting with C Field	

Response Message

Field	Content	Description
Endpoint ID	RADIOLINK_ID	Endpoint Identifier
Msg ID	RADIOLINK_MSG_WMBUSMSG_RSP	Send WM-Bus message response
Length	1	1 Octet
Payload[0]	Status 0x00 Operation failed 0x01 Operation successful	

3.2.2 WM-Bus Message Reception

Whenever the module receives an M-Bus message over the radio link, this message will be passed to the Host Controller. The included CRC Fields of the M-Bus message will be removed automatically and only the M-Bus Header and Data Blocks will be transmitted.

Event Message

Field	Content	Description
Endpoint ID	RADIOLINK_ID	Endpoint Identifier
Msg ID	RADIOLINK_MSG_WMBUSMSG_IND	WM-Bus message indication
Length	n	n octets
Payload	WM-Bus message, starting with C Field	

3.2.3 WM-Bus Data Request

This message can be used to send data as an M-Bus message via radio link. The first octet of the HCI payload is expected to be the CI- Field of the M-Bus message. The M-Bus Header Fields (C-Field, M-Field and A-Field) are taken from the configuration memory and can be modified via Device Configuration. The CRC16 of each M-Bus block and the M-Bus Length Field will be calculated and inserted by the firmware itself.

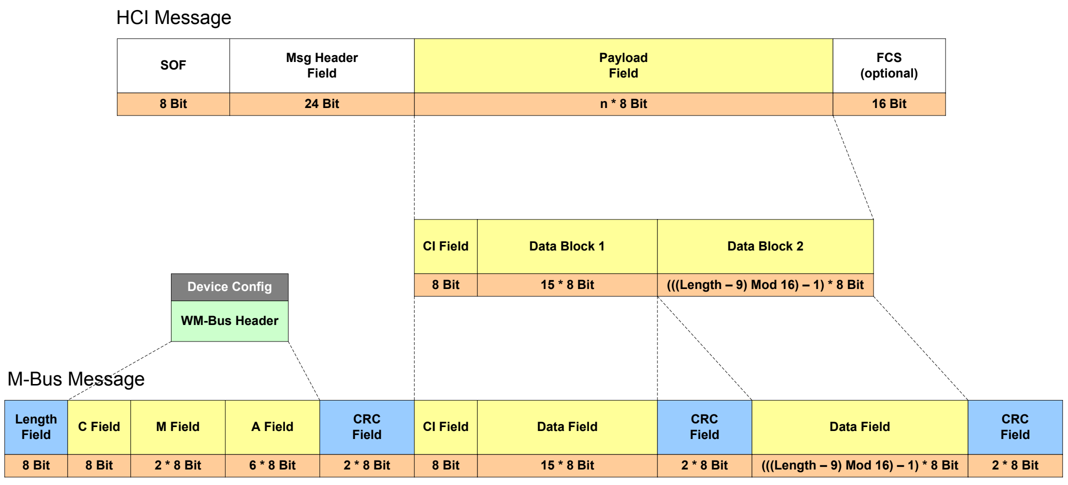

Fig. 3-4: WM-Bus Data Request Format (Telegram Format A)

Command Message

Field	Content	Description
Endpoint ID	RADIOLINK_ID	Endpoint Identifier
Msg ID	RADIOLINK_MSG_DATA_REQ	Send data as WM-Bus message
Length	n	n Octets
Payload	WM-Bus message starting with CI Field	

Response Message

Field	Content	Description
Endpoint ID	RADIOLINK_ID	Endpoint Identifier
Msg ID	RADIOLINK_MSG_DATA_RSP	Send data response
Length	1	1 Octet
Payload[0]	Status 0x00 Operation failed 0x01 Operation successful	

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 connected WM-Bus Module itself. A second WM-Bus Module in range is required, which is configured with same *Link Mode (S2, T2, R2, C2, N2x)* and which operates in *Other Mode*. The local connected module must be configured to *Meter Mode*.

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

Message Flow

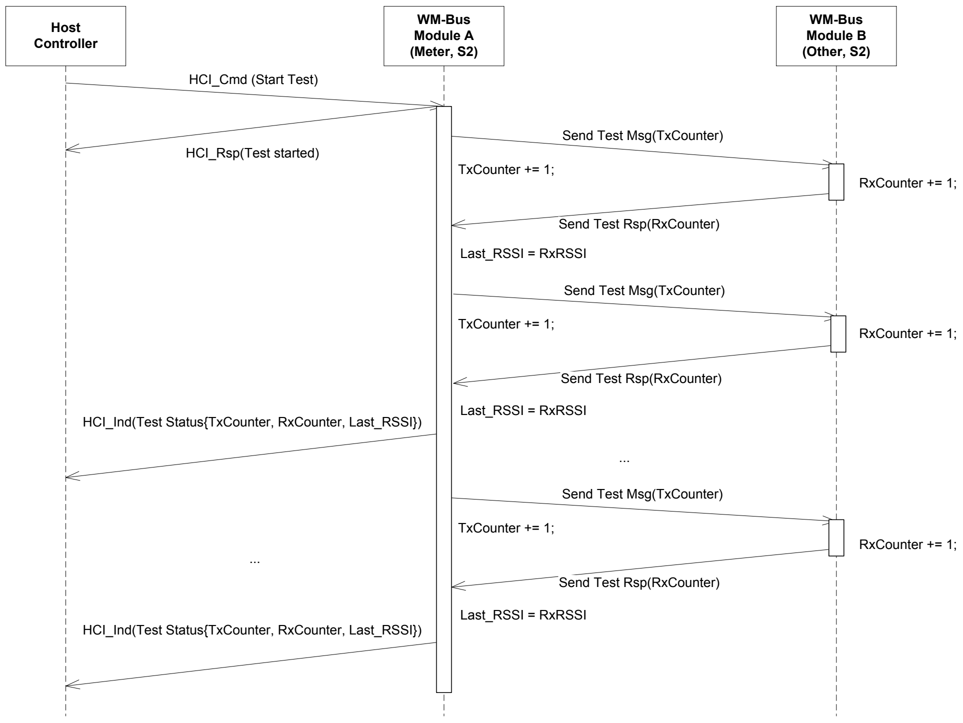

Fig. 3-5: Radio Link Test

During test operation the connected WM-Bus Module sends status messages to the Host Controller approximately every 500ms. The Status Message includes the following quality values:

- TxCounter - indicates the number of transmitted test messages
- RxCounter - indicates the number of received test messages
- estimated RSSI value from the last received radio message

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

$$PER[\%] = (1 - RxCounter/TxCounter) * 100$$

3.3.1 Start Radio Link Test

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

Command Message

Field	Content	Description
Endpoint ID	RADIOLINKTEST_ID	Endpoint Identifier
Msg ID	RADIOLINKTEST_MSG_START_REQ	Start Test Request
Length	N	n Octet
Payload	Radio Link Test Parameter Field	see below

Response Message

Field	Content	Description
Endpoint ID	RADIOLINKTEST_ID	Endpoint Identifier
Msg ID	RADIOLINKTEST_MSG_START_RSP	Start Test Response
Length	1	1 Octet
Payload[0]	Status 0x00 Operation failed 0x01 Operation successful	

3.3.2 Radio Link Test Status Message

This message is sent from the WM-Bus Module to the Host Controller during test operation.

Field	Content	Description
Endpoint ID	RADIOLINKTEST_ID	Endpoint Identifier
Msg ID	RADIOLINKTEST_MSG_STATUS_IND	Test Status Indication
Length	6	6 Octets
Payload[0]	Test Mode	Configured Test Mode
Payload[1-2]	TxCounter	Number of transmitted test messages
Payload[2-4]	RxCounter	Number of received test messages
Payload[5]	Last RSSI	Estimated RSSI of last received message

3.3.3 Radio Link Test Parameter Field

The following test parameter can be configured:

Offset	Size	Name	Description
0	2	Test Mode	0x00 = Single Test Run 0x01 = Repeated Test Runs (Note : repeated test runs must be stopped by Host Controller)
2	2	NumPackets	Number of Test Messages per Test Run
4	2	PacketSize	Number of Octets per Test Message
6	2	TxInterval	Time between two Test Messages

3.3.4 Stop Radio Link Test

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

Command Message

Field	Content	Description
Endpoint ID	RADIOLINKTEST_ID	Endpoint Identifier
Msg ID	RADIOLINKTEST_MSG_STOP_REQ	Stop Test Request
Length	0	No Payload

Response Message

Field	Content	Description
Endpoint ID	RADIOLINKTEST_ID	Endpoint Identifier
Msg ID	RADIOLINKTEST_MSG_STOP_RSP	Stop Test Response
Length	0	No Payload

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 Hardware Test Mode (see **System Operation Modes**).

Note: This feature is optional and maybe 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_RADIOTEST_REQ	Radio Test Request
Length	n	n Octets
Payload	Radio Test Parameter Field	

Response Message

Field	Content	Description
Endpoint ID	HWTEST_ID	Endpoint Identifier
Msg ID	HWTEST_MSG_RADIOTEST_RSP	Radio Test Response
Length	1	1 Octet
Payload[0]	Status 0x00 Operation failed 0x01 Operation successful	

3.4.2 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	Reserved	Reserved, must be set to 0x00	
2	1	Radio Channel	iM871A : 1 : 868.09 MHz (R-Mode) 2 : 868.15 MHz (R-Mode) 3 : 868.21 MHz (R-Mode) 4 : 868.27 MHz (R-Mode) 5 : 868.33 MHz (R-Mode) 6 : 868.39 MHz (R-Mode) 7 : 868.45 MHz (R-Mode) 8 : 868.51 MHz (R-Mode) 9 : 868.57 MHz (R-Mode) 10 : 868.30 MHz (S-Mode) 11 : 868.95 MHz (T-Mode)	iM170A : 0 : 169.40652 MHz (NA-Mode) 1 : 169.41875 MHz (NB-Mode) 2 : 169.43125 MHz (NC-Mode) 3 : 169.44375 MHz (ND-Mode) 4 : 169.45625 MHz (NE-Mode) 5 : 169.46875 MHz (NF-Mode) 6 : 169.43750 MHz (NG-Mode)
3	1	Radio Power Level	iM871A: 0 : -8 dBm 1 : -5 dBm 2 : -2 dBm 3 : 1 dBm 4 : 4 dBm 5 : 7 dBm 6 : 10 dBm 7 : 14 dBm	iM170A: 0 : 1dBm 1 : 10 dBm 2 : 20 dBm

4. Appendix

4.1 List of Constants

4.1.1 List of Endpoint Identifier

Name	Value
DEVMGMT_ID	0x01
RADIOLINK_ID	0x02
RADIOLINKTEST_ID	0x03
HWTEST_ID	0x04

4.1.2 Device Management Message Identifier

Name	Value
DEVMGMT_MSG_PING_REQ	0x01
DEVMGMT_MSG_PING_RSP	0x02
DEVMGMT_MSG_SET_CONFIG_REQ	0x03
DEVMGMT_MSG_SET_CONFIG_RSP	0x04
DEVMGMT_MSG_GET_CONFIG_REQ	0x05
DEVMGMT_MSG_GET_CONFIG_RSP	0x06
DEVMGMT_MSG_RESET_REQ	0x07
DEVMGMT_MSG_RESET_RSP	0x08
DEVMGMT_MSG_FACTORY_RESET_REQ	0x09
DEVMGMT_MSG_FACTORY_RESET_RSP	0x0A
DEVMGMT_MSG_GET_OPMODE_REQ	0x0B
DEVMGMT_MSG_GET_OPMODE_RSP	0x0C
DEVMGMT_MSG_SET_OPMODE_REQ	0x0D
DEVMGMT_MSG_SET_OPMODE_RSP	0x0E
DEVMGMT_MSG_GET_DEVICEINFO_REQ	0x0F
DEVMGMT_MSG_GET_DEVICEINFO_RSP	0x10
DEVMGMT_MSG_GET_SYSSTATUS_REQ	0x11
DEVMGMT_MSG_GET_SYSSTATUS_RSP	0x12
DEVMGMT_MSG_GET_FWINFO_REQ	0x13

DEVMGMT_MSG_GET_FWINFO_RSP	0x14
DEVMGMT_MSG_GET_RTC_REQ	0x19
DEVMGMT_MSG_GET_RTC_RSP	0x1A
DEVMGMT_MSG_SET_RTC_REQ	0x1B
DEVMGMT_MSG_SET_RTC_RSP	0x1C
DEVMGMT_MSG_ENTER_LPM_REQ	0x1D
DEVMGMT_MSG_ENTER_LPM_RSP	0x1E
DEVMGMT_MSG_SET_AES_ENCKEY_REQ	0x21
DEVMGMT_MSG_SET_AES_ENCKEY_RSP	0x22
DEVMGMT_MSG_ENABLE_AES_ENCKEY_REQ	0x23
DEVMGMT_MSG_ENABLE_AES_ENCKEY_RSP	0x24
DEVMGMT_MSG_SET_AES_DECKEY_RSP	0x25
DEVMGMT_MSG_SET_AES_DECKEY_RSP	0x26
DEVMGMT_MSG_AES_DEC_ERROR_IND	0x27

4.1.3 Radio Link Message Identifier

Name	Value
RADIOLINK_MSG_WMBUSMSG_REQ	0x01
RADIOLINK_MSG_WMBUSMSG_RSP	0x02
RADIOLINK_MSG_WMBUSMSG_IND	0x03
RADIOLINK_MSG_DATA_REQ	0x04
RADIOLINK_MSG_DATA_RSP	0x05

4.1.4 Radio Link Test Message Identifier

Name	Value
RADIOLINKTEST_MSG_START_REQ	0x01
RADIOLINKTEST_MSG_START_RSP	0x02
RADIOLINKTEST_MSG_STOP_REQ	0x03
RADIOLINKTEST_MSG_STOP_RSP	0x04
RADIOLINKTEST_MSG_STATUS_IND	0x07

4.1.5 Hardware Test Message Identifier

Name	Value
HWTEST_MSG_RADIOTEST_REQ	0x01
HWTEST_MSG_RADIOTEST_RSP	0x02

4.2 Example Code for Host Controller

4.2.1 Send HCI Message

```
//------------------------------------------------------------------------------
//
//  Section Defines
//
//------------------------------------------------------------------------------
#define WMBUS_SERIAL_PAYLOAD_SIZE           255
#define WMBUS_SERIAL_SOF                    0xA5
#define WMBUS_SERIAL_CRC_FLAG               0x80
#define WMBUS_SERIAL_RSSI_FLAG              0x40
#define WMBUS_SERIAL_TIME_FLAG              0x20
#define WMBUS_SERIAL_SAP_ID_MASK            0x0F

#define WMBUS_SERIAL_HEADER_SIZE            3
#define WMBUS_SERIAL_PAYLOAD_SIZE           255
#define WMBUS_SERIAL_SOF_SIZE               1
#define WMBUS_SERIAL_CRC_SIZE               2

//------------------------------------------------------------------------------
//
//  Section Typedefs
//
//------------------------------------------------------------------------------
// HCI Message
typedef struct
{
    // Start of Frame Character
    UINT8   SOF;
    // Control Field and Service Access Point Identifier
    UINT8   Ctrl;
    // Message Identifier
    UINT8   MsgID;
    // Payload Length Information
    UINT8   Length;
    // Payload Field
    UINT8   Payload[WMBUS_SERIAL_PAYLOAD_SIZE];
    // Frame Check Sequence Field
    UINT8   CRC16[2];
}TWMBusMessage;

typedef enum
{
    csw_RxSOF = 0,
    csw_RxHeader,
    csw_RxPayload
}TRxState;

typedef struct
{
    UINT8           SapID;
    UINT8           MsgID;
    TWMBusMessage   Msg;
    int             State;
    int             Count;
    UINT8*          Ptr;
}TReceiver;
```

```
//-------------------------------------------------------------------------------
//
//  Section Data
//
//-------------------------------------------------------------------------------

TReceiver           Rx;

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

//-------------------------------------------------------------------------------
//
//  SendHCIMessage
//
//-------------------------------------------------------------------------------
bool
TWMBusDevice_SendHCIMessage(UINT8    sapID,
                            UINT8    msgID,
                            UINT8*   payload,
                            UINT16   length)
{
    TWMBusMessage msg;

    // init header & SOF
    msg.SOF   =   WMBUS_SERIAL_SOF;
    msg.Ctrl  =   sapID | WMBUS_SERIAL_CRC_FLAG;
    msg.MsgID=    msgID;

    // truncate length, if necessary
    if(length > WMBUS_SERIAL_PAYLOAD_SIZE)
       length = WMBUS_SERIAL_PAYLOAD_SIZE;
    msg.Length = length;

    // copy payload
    UINT8* dstPtr = msg.Payload;
    while (length--)
        *dstPtr++ = *payload++;

    // calc length for CRC
    UINT16 txLength = msg.Length + WMBUS_SERIAL_HEADER_SIZE;

    // calc CRC16
    UINT16  crc16 = ~CRC16_Calc((UINT8*)&msg.Ctrl, txLength, CRC16_INIT_VALUE);

    // append CRC16
    *dstPtr++ = LOBYTE(crc16);
    *dstPtr++ = HIBYTE(crc16);

    // correct txLength
    txLength += (WMBUS_SERIAL_SOF_SIZE + WMBUS_SERIAL_CRC_SIZE);

    return SerialDevice_SendData(&msg.SOF, txLength);
}
```

4.2.2 Receive HCI Message

```
//------------------------------------------------------------------------------
//
//  ReceiverProcess
//
//------------------------------------------------------------------------------

void
TWMBusDevice_ReceiverProcess()
{
    UINT8 rxBuffer[256];

    // read chunk of octets from serial device
    int numRxBytes = SerialDevice_ReadData(rxBuffer, sizeof(rxBuffer));
    if (numRxBytes > 0)
    {
        TWMBusDevice_ProcessRxData (rxBuffer, numRxBytes);
    }
}




//------------------------------------------------------------------------------
//
//  ProcessRxData
//
//------------------------------------------------------------------------------

static void
TWMBusDevice_ProcessRxData(UINT8* rxBuffer, int length)
{
    // iterate over all received bytes
    while(length--)
    {
        // get rxByte
        UINT8 rxByte = *rxBuffer++;

        switch(Rx.State)
        {
        case    csw_RxSOF:
                // start of frame (SOF) received ?
                if(rxByte == WMBUS_SERIAL_SOF)
                {
                    // yes -> next state
                    Rx.State = csw_RxHeader;
                    // init counter
                    Rx.Count = WMBUS_SERIAL_HEADER_SIZE;
                    // init pointer
                    Rx.Ptr = (UINT8*)&Rx.Msg.Ctrl;
                }
                break;

        case    csw_RxHeader:
                // store rx byte
                *Rx.Ptr++ = rxByte;
                // decrement counter
                Rx.Count--;
                // last byte of header (length field) received ?
                if(!Rx.Count)
                {
                    // load payload counter
                    Rx.Count = rxByte;
                    if(Rx.Msg.Ctrl & WMBUS_SERIAL_CRC_FLAG)
                    {
                        Rx.Count += 2;
                    }
                    if(Rx.Msg.Ctrl & WMBUS_SERIAL_TIME_FLAG)
                    {
                        Rx.Count += 4;
                    }
                    if(Rx.Msg.Ctrl & WMBUS_SERIAL_RSSI_FLAG)
                    {
                        Rx.Count += 1;
                    }
                    // payload attached ?
                    if(Rx.Count != 0)
                    {
                        // next state
                        Rx.State = csw_RxPayload;
                    }
                    else
                    {
                        // handle received message
                        TWMBusDevice_ProcessRxMsg();
                        // next state: ready to receive
                        Rx.State = csw_RxSOF;

                    }
                }
                break;

        case    csw_RxPayload:
                // store rx byte
                *Rx.Ptr++ = rxByte;
                // decrement payload counter
                Rx.Count--;
                // check end of frame
                if(!Rx.Count)
                    // handle received message
                    TWMBusDevice_ProcessRxMsg();
                break;
        }
    }
}
```

```
//--------------------------------------------------------------------------------
//
//  ProcessRxMsg
//
//--------------------------------------------------------------------------------
void
TWMBusDevice_ProcessRxMsg()
{
    // CRC attached ?
    if(Rx.Msg.Ctrl & WMBUS_SERIAL_CRC_FLAG)
    {
        UINT16 length;

        length = Rx.Msg.Length + WMBUS_SERIAL_HEADER_SIZE +
                 WMBUS_SERIAL_CRC_SIZE;

        if(Rx.Msg.Ctrl & WMBUS_SERIAL_RSSI_FLAG)
            length++;

        if(Rx.Msg.Ctrl & WMBUS_SERIAL_TIME_FLAG)
            length += 4;

        // CRC error ?
        if(!CRC16_Check((UINT8*)&Rx.Msg.Ctrl, length, CRC16_INIT_VALUE))
        {
            return;
        }
   }
   // handle message
   DispatchRxMsg(Rx.Msg);
}
```

4.2.3 CRC6 Calculation

Example code for CRC16 calculation:

```
File: CRC16.h

#ifndef     __CRC16_H__
#define     __CRC16_H__
//------------------------------------------------------------------------------
//
//   Section Include Files
//
//------------------------------------------------------------------------------

#include <inttypes.h>

typedef uint8_t      UINT8;
typedef uint16_t     UINT16;
//------------------------------------------------------------------------------
//
//   Section Defines
//
//------------------------------------------------------------------------------
#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
//------------------------------------------------------------------------------
// C++ Extensions
//------------------------------------------------------------------------------
#ifdef    __cplusplus
extern  "C" {
#endif
//------------------------------------------------------------------------------
//
//   Section Prototypes
//
//------------------------------------------------------------------------------

//------------------------------------------------------------------------------
//! Calc CRC16
UINT16
CRC16_Calc  (UINT8*     data,
             UINT16     length,
             UINT16     initVal);
//------------------------------------------------------------------------------
//! Calc & Check CRC16
bool
CRC16_Check (UINT8*     data,
             UINT16     length,
             UINT16     initVal);

//------------------------------------------------------------------------------
// C++ Extensions
//------------------------------------------------------------------------------
#ifdef     __cplusplus
}
#endif
//------------------------------------------------------------------------------

#endif // __CRC16_H__
```

```
File: CRC16.c

//------------------------------------------------------------------------------
//
//  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 CRC16 value according to f the standard
//! 16-BIT CRC CCITT polynomial G(x) = 1 + x^5 + x^12 + x^16
//!
//! <!-------------------------------------------------------------------------->
//! @param[in]      data        pointer to data block
//! @param[in]      length      number of bytes
//! @param[in]      initVal     CRC16 initial value
//! <!-------------------------------------------------------------------------->
//! @retVal         crc16       crc
//------------------------------------------------------------------------------
#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

// calculate CRC16 without table
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
//!
//! <!------------------------------------------------------------------------->
//! @param[in]      data        pointer to data block
//! @param[in]      length      number of bytes (including CRC16)
//! @param[in]      initVal     CRC16 initial value
//! <!------------------------------------------------------------------------->
//! @retVal         true        CRC16 ok -> data block ok
//! @retVal         false       CRC16 failed -> data block corrupt
//------------------------------------------------------------------------------

bool
CRC16_Check     (UINT8*                     data,
                 UINT16                     length,
                 UINT16                     initVal)
{
    // calculate ones complement of CRC16
    UINT16 crc = ~CRC16_Calc(data, length, initVal);

    if( crc == CRC16_GOOD_VALUE)
        return true;

    return false;
}
//------------------------------------------------------------------------------
// end of file
//------------------------------------------------------------------------------
```

4.3 List of Abbreviations

AES	Advanced Encryption Standard
CRC	Cyclic Redundancy Check
FW	Firmware
HCI	Host Controller Interface
HW	Hardware
RAM	Random Access Memory
RF	Radio Frequency
RSSI	Received Signal Strength Indicator
RTC	Real Time Clock
SW	Software
UART	Universal Asynchronous Receiver/Transmitter
WM-Bus	Wireless M-Bus

4.4 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

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