Wireless M-Bus Host Controller Interface DLL

Specification

Document ID: 4100/6404/0051

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

Document Information

File name	WMBus_HCIDLL_Spec.docx
Created	2011-06-24
Total pages	48

Revision History

Version	Note
0.1	Created, Initial Version
0.2	Draft Version Created For Review
0.5	Preliminary Version
1.0	Reviewed and released
1.1	AES functions added
1.2	C-Mode added
1.3	Rework
1.4	AES Decryption Configuration changed

Aim of this Document

This document describes the Wireless M-Bus HCI DLL Application Interface.

Table of Contents

1. Introduction

1.1 Overview

The WM-Bus HCI library enables Windows applications to configure and control several kinds of Radio Modules which support the IMST WM-Bus Host Controller Interface (HCI). Such modules are referred to as WM-Bus Modules hereafter.

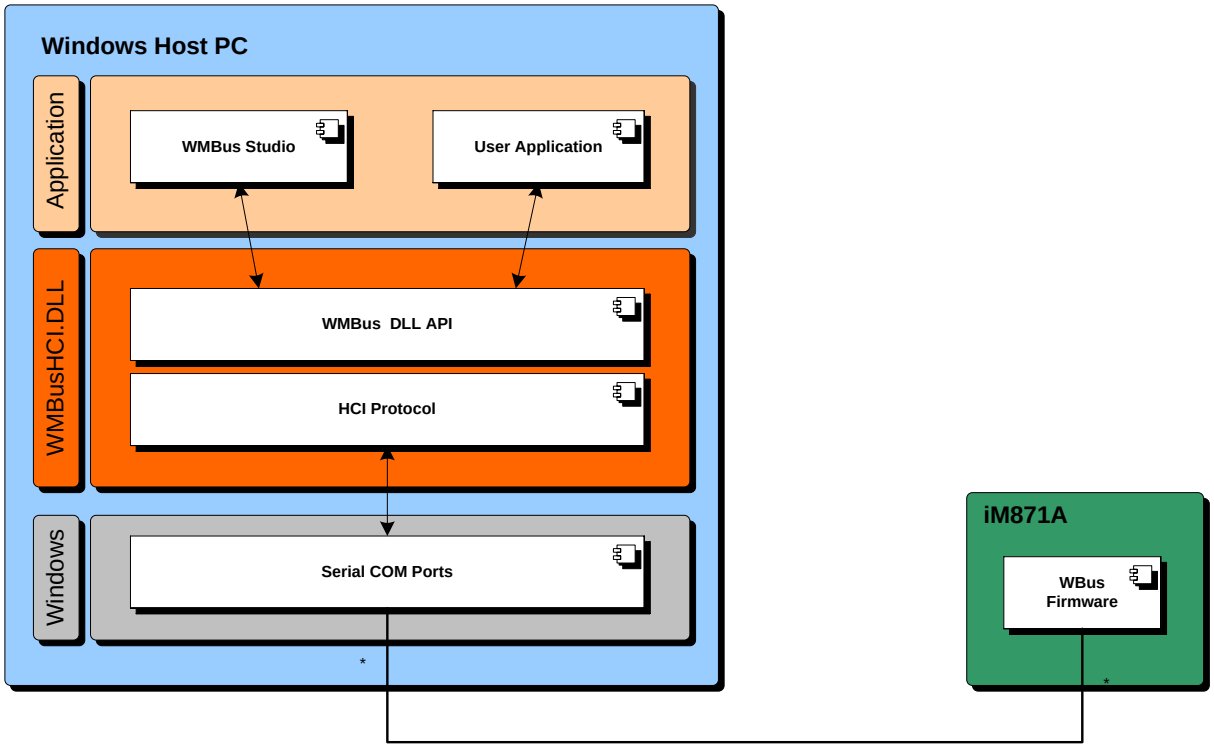

Fig. 1-1: Overview

The library supports several service functions which use the WM-Bus HCI Protocol (see [1]) to communicate with an attached WM-Bus Module (e.g. iM871A). These HCI messages will be sent through a Serial Comport (COM1…X).

2. Services

This chapter outlines the supported DLL functions which are subdivided into the following functional groups:

- General DLL functions which are not related to a connected radio module
- Device Management functions for configuration & control
- Data Link functions for data exchange
- Radio Link Test function for radio performance measurements
- Hardware Test functions

2.1 General DLL Functions

This chapter covers functions which are only related to the WM-Bus HCI DLL itself.

2.1.1 WM-Bus Connection Handle

The key element for communication with a dedicated connected WM-Bus device is a so called WM-Bus Connection Handle (TWMBusHandle). A handle can be obtained by opening a dedicated Serial Comport. In general the WM-Bus DLL supports access to multiple connected WM-Bus Modules in parallel.

2.1.1.1 Open Device

This function opens a Serial Communication Port to a connected WM-Bus Module. On success the function returns a WM-Bus Connection Handle which can be used for communication purposes. Note: This function doesn't check if a device is really connected, it only opens the Serial Port.

Prototype

- TWMBusHandle WMBus_OpenDevice(const char *comPort)

Parameter

- comPort pointer to Comport name e.g. "COM1"

Return Value

- 0 invalid handle
- else valid handle

Note: TWMBusHandle is defined as ***unsigned long***.

2.1.1.2 Close Device

This function closes a Serial Communication Port.

Prototype

- bool WMBus_CloseDevice(TWMBusHandle handle)

Parameter

- handle communication handle

Return Value

- true comport closed
- false error

2.1.2 Error Handling

Most of the provided DLL functions simply return the boolean value **false** in case of an error or **true** in case of success. The detailed error reason is stored within the DLL and can be readout and translated into a human readable error string.

2.1.2.1 Get Last Error

This function returns the error code of the last triggered function.

Prototype

- UINT32 WMBus_GetLastError(TWMBusHandle handle)

Parameter

- handle communication handle

Return Value

- error code

2.1.2.2 Get Error String

Error codes can be translated in human readable error strings by means of the following function.

Prototype

- void WMBus_GetErrorString (UINT32 error,
char* string,
int size)

Parameter

- error an error code returned by WMBus_GetLastError
- string pointer to string
- size size in bytes of the character array

Note: A size of at least 64 byte is recommended.

Return Value

- an updated '0' terminated string

2.1.2.3 Error Codes and Strings

The following table shows the implemented error codes and strings:

Error Codes	Error Strings
0	"WMBUS_NO_ERROR"
1	"WMBUS_INVALID_COMPORT"
2	"WMBUS_CONNECTION_BUSY"
3	"WMBUS_INVALID_HANDLE"
4	"WMBUS_NO_RESPONSE"
5	"WMBUS_TRANSMIT_ERROR"
6	"WMBUS_UNEXPECTED_FRAMELENGTH"
7	"WMBUS_COMMAND_NOT_SUPPORTED"
8	"WMBUS_COMMAND_FAILED"
9	"WMBUS_UNEXPECTED_STATUS"
10	"WMBUS_UNEXPECTED_RESPONSE_CODE"
11	"WMBUS_UNEXPECTED_COMMAND_ERROR"
12	"WMBUS_INVALID_PARAMETER"

2.1.3 Get DLL Version

This function returns the DLL Version string.

Prototype

- bool WMBus_GetDLLVersion (char* resultString,
  int size)

Parameter

- resultString pointer to an character array
- size size in byte of the character array

Note: A size of at least 32 byte is recommended.

Return Value

- true an updated resultString
- false resultString size to small

2.1.4 Set Timeout

After sending a HCI Command message to the connected device the DLL is waiting for a response. The maximum wait time in milliseconds can be configured by means of this function.

Prototype

- bool WMBus_SetTimeout(TWMBusHandle handle,
  UINT32 timeout)

Parameter

- handle communication handle
- timeout timeout in milliseconds

Note: the default value is 1000 ms

Return Value

- true parameter changed
- false error

2.1.5 Register Message Handler

Most of the service functions are processed in a synchronous way i.e. a function doesn't return until a WM-Bus device response was received or a configurable timeout expires.

For asynchronous communication an application can register a message handler to listen for several kinds of messages which are generated within the DLL. The message handler will receive 2 different parameters when called, the message type (msg) and a generic parameter (param) with different meaning depending on the given message type.

Prototype

- void WMBus_RegisterMsgHandler(TWMBus_CbMsgHandler cbMsgHandler)

Parameter

- cbMsgHandler pointer to message handler procedure

The function pointer type is defined as follows:

typedef void (*TWBus_CbMsgHandler)(UINT32 msg, UINT32 param)

Return Value

- none

The following messages are implemented:

Message	Parameter	Description
WMBUS_MSG_HCI_MESSAGE_IND (0x00000004)	Handle	a HCI message is available and can be read by means of WMBus_GetHCIMessage

2.1.6 Get HCI Message

This function can be used to read buffered HCI Messages.

Prototype

- bool WMBus_GetHCIMessage (TWBusHandle handle,
UINT8* buffer,
UINT16 bufferSize)

Parameter

- handle communication handle
- buffer pointer to message buffer (256 bytes recommended)
- bufferSize size of message buffer

Return Value

- true message readout successfully
- false error, no message/frame available

Message Buffer Layout

Message Header				Message Payload	Time Stamp (optional)	RSSI (optional)
Ctrl Field	Endpoint	Message ID	Payload Length (n)			
4 Bit	4 Bit	8 Bit	8 Bit	n * 8 Bit	32 Bit	8 Bit

Fig. 2-1: Overview

Offset	Size	Name	Description
0	1	Ctrl Field & Endpoint ID	Endpoint Identifier (Bits 0-3), indicates to which logical endpoint this message belongs. The following endpoint are available: 0x01 : Device Management Services 0x02 : Radio Data Link Services 0x03 : Radio Link Test 0x04 : Hardware Test Services Bit 4 : reserved Bit 5 : (0x20) indicates if a 32-Bit Timestamp is attached Bit 6 : (0x40) indicates if an 8-Bit RSSI is attached Bit 7 : (0x80) reserved
1	1	Message ID	Message Identifier, indicates the type of this message
2	1	Payload Length	Length = Size of Payload Field in bytes
3 … N+2	N	Payload	Message dependend Payload Field
N + 3	4	Time Stamp	optinal Time Stamp (indicates time of received RF message e.g.)
N + 4	1	RSSI	optional RSSI value (see device configuration)

2.1.7 Shutdown

An application should call the Shutdown function when finished so that the DLL can close all connections and release all allocated system resources.

Prototype

- bool WMBus_DLLShutdown()

Parameter

- none

Return Value

- always true

2.2 Device Management Functions

This chapter describes functions which are mapped to the so called Device Management endpoint of the firmware. This 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

2.2.1 Ping Request

This function can be used to check the wired communication interface and if the connected device is alive. A "Ping Command" message is send to the local connected device which answers with a "Ping Response" message.

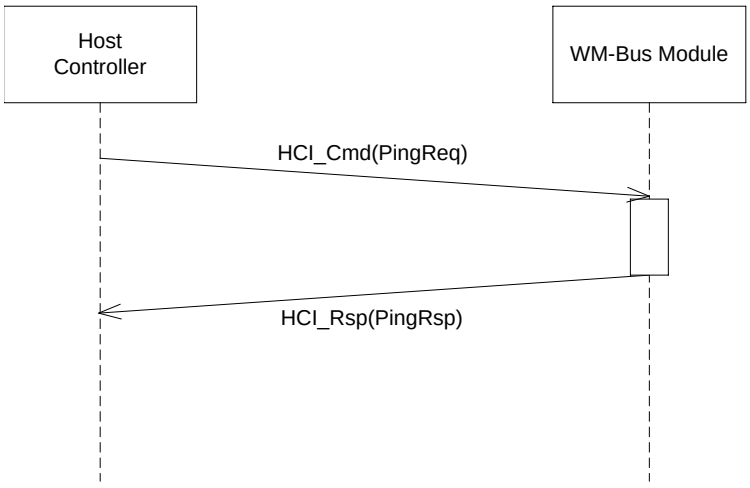

Fig. 2-2: Ping Request

Prototype

- bool WMBus_PingRequest(TWMBusHandle handle)

Parameter

- handle communication handle

Return Value

- true device connected and alive
- false error

2.2.2 Reset Request

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

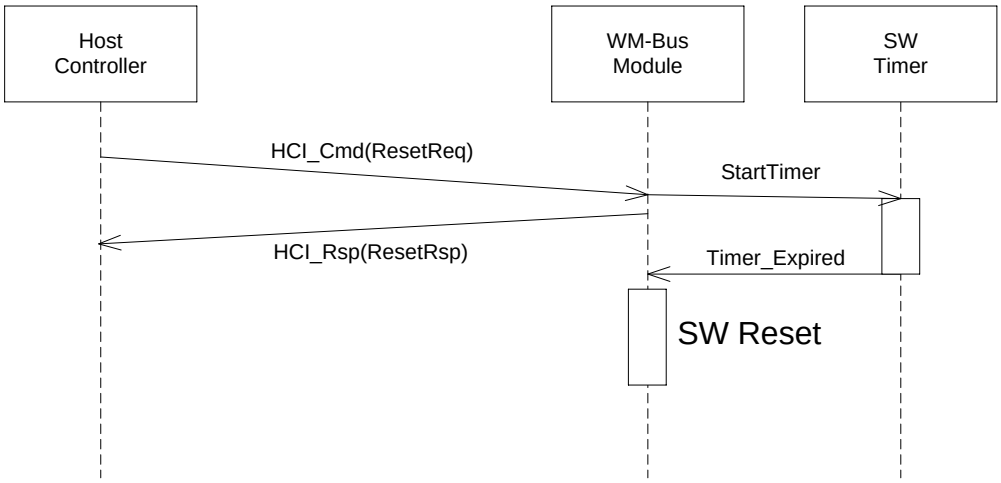

Fig. 2-3: Reset Request

Prototype

- bool WMBus_ResetRequest(TWMBusHandle handle)

Parameter

- handle communication handle

Return Value

- true device will be reset
- false error

2.2.3 Device Information

For identification purpose the WM-Bus Firmware provides a service to readout some information elements e.g. Module Type, Firmware Version.

2.2.3.1 Get Device Info

This command returns the basic device information block.

Prototype

```
- bool WMBus_GetDeviceInfo(TWMBusHandle    handle,
                           UINT8*          buffer,
                           UINT16          bufferSize)
```

Parameter

- handle communication handle
- buffer pointer to buffer
- bufferSize size of buffer in bytes

Return Value

- true updated buffer
- false error

2.2.3.2 Device Information

The updated buffer has the following layout:

Offset	Size	Name	Description
0	1	Result Length	Number of bytes in buffer
1	1	ModuleType	Identifies the Radio Module e.g. iM871A
2	1	Device Mode	Indicates the current Device Mode e.g. Meter, Concentrator
3	1	Firmware Version	Firmware Version, e.g. 0x13 = V1.3
4	1	HCI Protcol Version	HCI Protocol Version, e.g. 0x01
5 – 8	4	Device ID	Device ID

2.2.4 Device Configuration

The WM-Bus Firmware supports several kinds of configurable system parameters which are stored in a non volatile parameter memory e.g. Flash or EEPROM depending on the module type. The configuration parameters are readout during start-up and used to configure the firmware components and the hardware units. The following items can be configured:

Item	Description
Device Mode	Determines if the module operates in Meter or Other Mode.
Radio Link Mode	Determines one of the following radio link modes: S1, S1-m, S2, T1, T2, R2, C1, C2
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 from -8dBm to +13dBm
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 tor 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.

2.2.4.1 Default Configuration

The following table lists the default configuration.

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 : 13dBm
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

2.2.4.2 Get Device Configuration

This function can be used to readout the configuration parameters.

Prototype

```
- bool WMBus_GetDeviceConfig(TWMBusHandle    handle,
                             UINT8*          buffer,
                             UINT16          bufferSize)
```

Parameter

- handle — communication handle
- buffer — pointer to buffer
- bufferSize — size of buffer in bytes

Return Value

- true — updated buffer
- false — error

Device Configuration

The updated buffer has the following layout:

Offset	Size	Name	Description
0	1	Result Length	Number of bytes in buffer
1 – N	N	Device Parameter List	see Device Parameter List

2.2.4.3 Set Device Configuration

This function can be used to change several system parameters. The function allows to change parameter directly and to save them in a non-volatile memory (e.g. Flash or EEPROM).

Prototype

```
- bool WMBus_SetDeviceConfig(TWMBusHandle    handle,
                             UINT8*          configList,
                             UINT8           configLength,
                             bool            storeInNVM)
```

Parameter

- handle — communication handle
- configList — pointer to device parameter list
- configLength — length in bytes in parameter list

- storeInNVM	flag, if true the parameter will be stored in non-volatile memory

Return Value

- true	device parameter set
- false	error

2.2.4.4 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 Link 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	Radio Link Mode	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
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
variable	4	WM-Bus Device ID	Device ID, used in WM-Bus Radio Messages
variable	1	WM-Bus Version	Version, used in WM-Bus Radio Messages
variable	1	WM-Bus Device Type	Device Type, used in WM-Bus Radio Messages

variable	1	Radio Channel	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	RF Power Level: 0 : -8 dBm 1 : -5 dBm 2 : -2 dBm 3 : 1 dBm 4 : 4 dBm 5 : 7 dBm 6 : 10 dBm 7 : 13 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)
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), reception or when the Radio Rx Window terminates (S2, T2, R2, C2).
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

2.2.4.5 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.

Prototype

- bool WMBus_FactoryResetRequest(TWMBusHandle handle,
bool rebootFlag)

Parameter

- handle communication handle
- rebootFlag indicates if the device should perform reboot

Return Value

- true device configuration reset
- false error

2.2.5 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 and requires a reset to get changed.

2.2.5.1 Get System Operation Mode

The following function returns the current System Operation Mode.

Prototype

- bool WMBus_GetOperationMode(TWMBusHandle handle,
UINT8* mode)

Parameter

- handle communication handle
- mode pointer to return value

Return Value

- true operation mode read
- false error

2.2.5.2 System Operation Modes

The following System Operation Modes are supported:

Value	Description
0	Standard Application Mode / Default Mode
1	Hardware Test Mode

2.2.5.3 Set Application Mode

This function sets the system into Application Mode and performs a firmware reset.

Prototype

- bool WMBus_SetApplicationMode(TWMBusHandle handle)

Parameter

- handle communication handle

Return Value

- true Application mode set
- false error

2.2.5.4 Set Hardware Test Mode

This function sets the system into Hardware Test Mode and performs a firmware reset.

Prototype

- bool WMBus_SetHWTestMode(TWMBusHandle handle)

Parameter

- handle communication handle

Return Value

- true HW Test mode set
- false error

2.2.6 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 are not stored in the non-volatile memory.

2.2.6.1 Get System Status

This function can be used to read the current System Status:

Prototype

```
- bool WMBus_GetSystemStatus(TWMBusHandle    handle,
                             UINT8*          buffer,
                             UINT16          bufferSize)
```

Parameter

- handle communication handle
- buffer pointer to buffer
- bufferSize size of buffer in bytes

Return Value

- true updated buffer
- false error

2.2.6.2 System Status Details

The updated buffer contains the following information:

Offset	Size	Name	Description
0	1	Result Length	Number of bytes in buffer
1	1	NVM Status	0x00 : no error else : NVM contains corrupt data
2	1	Defects	Number of stored Defects
3 – 6	4	SystemTick	System Ticks with 10 ms resolution
7 – 10	4	Reserved	
11 – 14	4	Reserved	
15 – 18	4	NumTxFrames	Number of transmitted messages
19 – 22	4	NumTxErrors	Number of not transmitted messages
23 – 26	4	NumRxFrames	Number of received messages
27 – 30	4	NumRxCRC Errors	Number of received CRC errors

31 – 34	4	NumRxPHY Errors	Number of Rx PHY errors
35 – 38	4	Reserved	

2.2.7 Firmware Information

This function can be used to read information about the firmware itself.

Prototype

```
- bool WMBus_GetFWBuildInfo(TWMBusHandle    handle,
                            UINT8           key,
                            UINT8*          buffer,
                            UINT16          bufferSize)
```

Parameter

- handle — communication handle
- key — key for requested type of information
- buffer — pointer to buffer
- bufferSize — size of buffer in bytes

Return Value

- true — updated buffer
- false — error

Firmware Info

The updated buffer contains the following data depending on the given key:

Firmware Version, Key = 0x00

Offset	Size	Name	Description
0	1	Result Length	Number of bytes in buffer
1	1	Key	Requested type of information
2	1	Length	Length of FW information
3	1	Version	Firmware Version
4 – 5	2	Build Version	Build Version

Firmware Name, Key = 0x01

Offset	Size	Name	Description
0	1	Result Length	Number of bytes in buffer
1	1	Key	Requested type of information
2	1	String Length	Length of Firmware Name
3 - N+2	N	String	Firmware Name

Date String, Key = 0x02

Offset	Size	Name	Description
0	1	Result Length	Number of bytes in buffer
1	1	Key	Requested type of information
2	1	String Length	Length of Date String
3 - N+2	N	String	Date String

Time String, Key = 0x04

Offset	Size	Name	Description
0	1	Result Length	Number of bytes in buffer
1	1	Key	Requested type of information
2	1	String Length	Length of Time String
3 - N+2	N	String	Time String

2.2.9 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 startup. For usage the RTC needs to be enabled (see chapter ***Device Configuration***).

2.2.9.1 Get RTC Time

This function can be used to read the RTC time:

Prototype

- bool WMBus_GetRTCTime(TWMBusHandle handle,
UINT32* time)

Parameter

- handle communication handle
- time pointer to buffer for RTC time

Return Value

- true updated buffer with RTC time (32768Hz resolution)
- false error

2.2.9.2 Set RTC Time

This function can be used to set the RTC time:

Prototype

- bool WMBus_SetRTCTime(TWMBusHandle handle,
UINT32 newTime)

Parameter

- handle communication handle
- newTime new RTC time (32768Hz resolution)

Return Value

- true RTC updated with new time
- false error

2.2.10 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.

Prototype

- bool WMBus_EnterLowPowerMode(TWMBusHandle handle,
  UINT8 mode)

Parameter

- handle communication handle
- mode must be set to 0x00

Return Value

- true low power mode will be entered
- false error

2.2.11 AES-128 Encryption / Decryption

The firmware supports automatic AES-128 encryption and decryption of radio link messages. The following functions for key configuration and activation are provided:

2.2.11.1 AES-128 Encryption Key Configuration

The following function can be used to configure the AES encryption key.

Prototype

```
- bool WMBus_ConfigureAESKey(TWMBusHandle    handle,
                             UINT8*          key,
                             bool            storeInNVM)
```

Parameter

- handle	communication handle
- key	pointer to AES-128 encryption key (16 bytes)
- storeInNVM	flag, if true the key will be stored in non-volatile memory

Return Value

- true	AES key configured
- false	error

2.2.11.2 AES-128 Encryption Service Activation

The following function can be used to enable or to disable the AES encryption service.

Prototype

```
- bool WMBus_EnableAESKey(TWMBusHandle    handle,
                          bool            enableFlag,
                          bool            storeInNVM)
```

Parameter

- handle	communication handle
- enableFlag	flag, if true enables the AES-128 encryption service
- storeInNVM	flag, if true the key will be stored in non-volatile memory

Return Value

- true	AES encryption enabled / disabled
- false	error

2.2.11.3 AES-128 Decryption Key Configuration

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). The following function can be used to configure the AES decryption key.

Prototype

- bool

```
WMBus_ConfigureAESDecryptionKey(TWMBusHandle    handle,
                                UINT8           slotIndex,
                                UINT8*          addressFilter,
                                UINT8*          key)
```

Parameter

- handle communication handle
- addressFilter pointer to WM-BUS Device Filter (see WMBus_HCI_Spec.pdf)
- key pointer to AES-128 decryption key (16 bytes)

Return Value

- true AES key configured
- false error

2.2.12 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.

The message layout looks as follows:

Offset	Size	Name	Description
0	1	Endpoint ID	0x01 : Device Management Services
1	1	Message ID	0x27 : AES Decryption Error Indication
2	1	Payload Length	Length = **9 bytes**

3	9	Payload	WM-Bus Header of received packet : 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)

2.3 Data Link Services

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

2.3.1 Send Message

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.

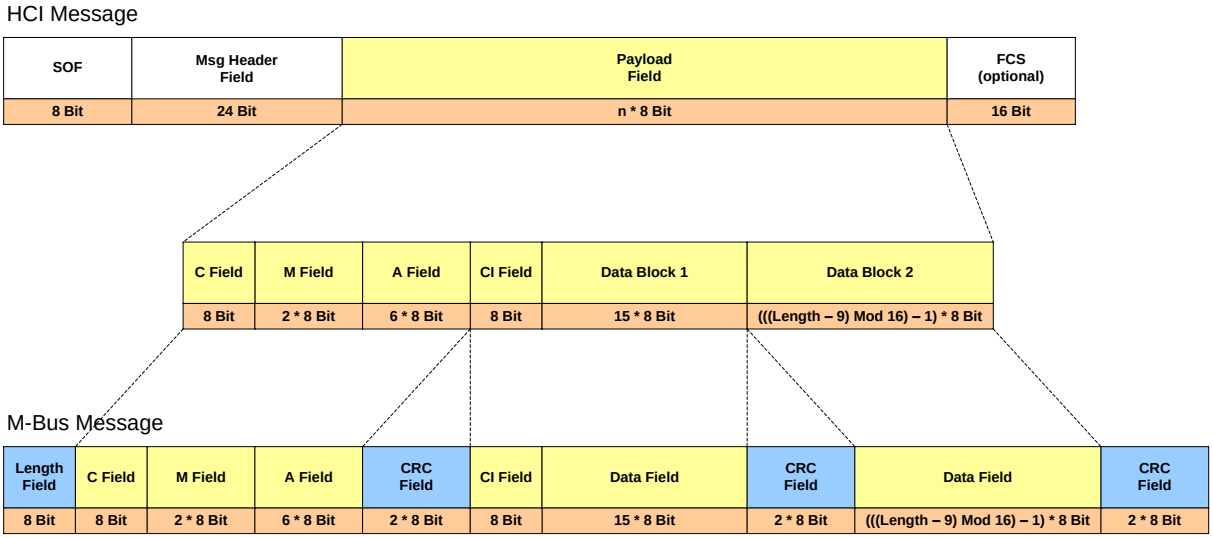

Fig. 2-4: HCI and M-Bus message (Telegram Format A)

Prototype

```
- bool WMBus_SendMessage(TWMBusHandle    handle,
                         UINT8*          payload,
                         UINT8           length)
```

Parameter

- handle communication handle
- payload point to M-Bus message
- length number of message bytes

Return Value

- true message transmitted
- false error

2.3.2 Message Reception

The received Radio Link Message can be read by means of the function WMBus_GetHCIMessage. The messages have an Endpoint Identifier value 0x02 = Radio Link Services.

The radio message layout looks as follows:

Offset	Size	Name	Description
0	1	Endpoint ID	0x02 : Radio Link Services
1	1	Message ID	0x03 : Radio Message Indication
2	1	Payload Length	Length = Size of Payload Field in bytes
3	N	Payload	Radio Message starting with C-Field
N + 3	1	[RSSI]	optional RSSI value (see device configuration)

2.3.3 WM-Bus Data Request

This message can be used to send data as 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.

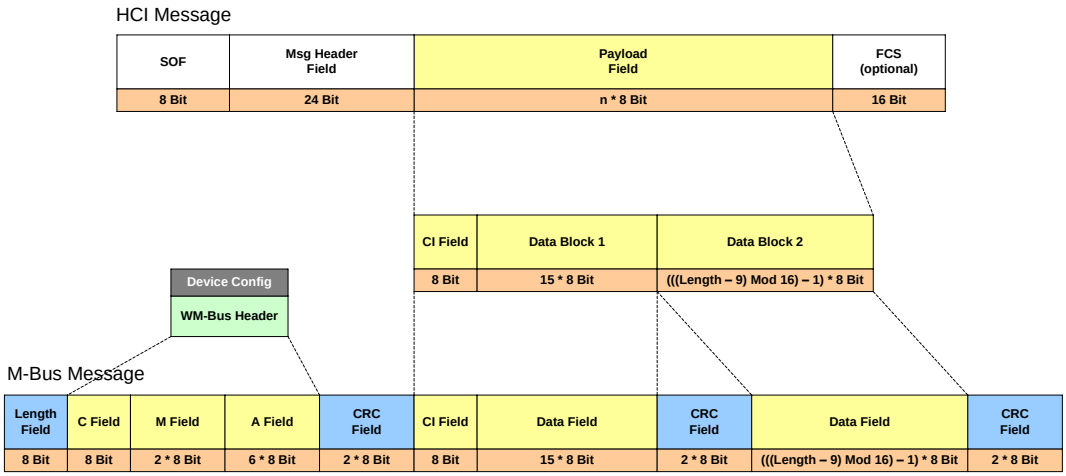

Fig. 2-5: WM-Bus Data Request Format (Telegram Format A)

Prototype

```
- bool WMBus_SendData(TWMBusHandle    handle,
                      UINT8*          payload,
                      UINT8           length)
```

Parameter

- handle	communication handle
- payload	point to M-Bus message data block
- length	number of message bytes

Return Value

- true	message transmitted
- false	error

2.4 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)*** and which operates in ***Other Mode***. The local connected module must be configured to ***Meter Mode***.

Message Flow

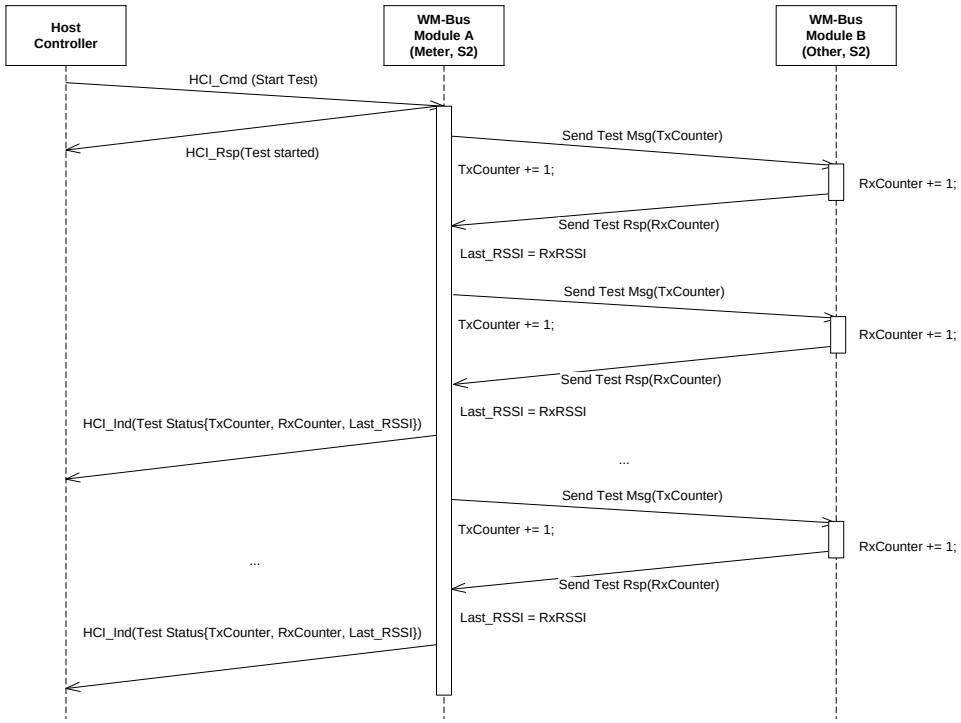

Fig. 2-6: Radio Link Test

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:

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

2.4.1 Start Radio Link Test

This function starts the packet error rate test.

Prototype

- bool WMBus_StartRadioLinkTest(TWMBusHandle handle,
UINT16 mode,
UINT16 numPackets,
UINT16 packetSize,
UINT16 txPeriod)

Parameter

- handle communication handle
- mode measurement mode:
0 = single test run
1 = repetitive test runs
- numPackets number of test packets to transmit
- packetSize packet length in bytes
- txPeriod time in milliseconds between two packets

Return Value

- true test started
- false error

2.4.2 Radio Link Test Status Message

The Radio Link Test Status Messages which can be read by means of the function WMBus_GetHCIMessage, have an Endpoint Identifier value 0x03 = Radio Link Test.

The status message layout looks as follows:

Offset	Size	Name	Description
0	1	Endpoint ID	0x03 : Radio Link Test
1	1	Message ID	0x07 : Radio Linkt Test Status Indication
2	1	Payload Length	Length = Size of Payload Field in bytes
3	1	Mode	Requested Test Mode
4	2	TxCounter	Number of transmitted test packets
6	2	RxCounter	Number of received test packets from peer device
7	1	RSSI	Last received RSSI

2.4.3 Stop Radio Link Test

This function can be used to stop the packet error rate test.

Prototype

- bool WMBus_StopRadioLinkTest(TWMBusHandle handle)

Parameter

- handle communication handle

Return Value

- true test stopped
- false error

2.5 Hardware Test Functions

Hardware test functions are available within a specific Hardware Test Firmware. Some test functions are also supported by the Standard Firmware but they can only be executed when the device is set to the System Operation Mode "Hardware Test" (see Set System Operation Mode).

2.5.1 Radio Test Functions

The firmware supports different radio test services which are described below:

2.5.1.1 Radio Test Request

This function activates/deactivates a certain Radio Test on the local connected device.

Prototype

```
- bool WMBus_RFTestRequest(TWMBUSHandle    handle,
                           UINT8*          parameter,
                           UINT8           length)
```

Parameter

- handle — communication handle
- parameter — pointer to Radio Test parameter block
- length — length of parameter block in bytes

Return Value

- true — test started
- false — error

2.5.1.2 Radio Test Parameter Block

The parameter block looks as follows:

Offset	Size	Name	Description
0	1	Test Mode	Type of RF Test: 0 : Stop test 1 : Start CW test
1	1	Reserved	must be set to 0x00
2	1	Radio Channel	Radio Channel (see device configuration)
3	1	Radio Power Level	Radio Power Level (see device configuration)

3. Appendix

3.1 List of Abbreviations

DIO	Digital Input/Output
DLL	Dynamic Link Library
FW	Firmware
GPIO	General Purpose Input/Output
HCI	Host Controller Interface
HW	Hardware
LPM	Low Power Mode
RAM	Random Access Memory
RF	Radio Frequency
RSSI	Received Signal Strength Indicator
RTC	Real Time Clock
SPI	Serial Peripheral Interface
SW	Software
UART	Universal Asynchronous Receiver/Transmitter
WM-Bus	Wireless M-Bus

3.2 List of Figures

3.3 References

[1] WMBus_HCI_Spec_Vx_y.pdf

4. 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.

5. Important Notice

5.1 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.

5.2 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