WiMOD DLL

Specification (V1.3, 2010-10-28)

IMST GmbH
Carl-Friedrich-Gauss-Str. 2-4
D-47475 Kamp-Lintfort

Document History

Date	Version	Chapter	Comment
2009-06-10	0.1	All	First version (KvW)
2010-01-07	0.2	All	Document format and title changed (KvW)
2010-03-22	0.3	General WiMOD Device Functions	Get Device Info added
2010-08-18	1.0	Data Exchange Services	Minor changes
2010-10-13	1.1	Chapter 1.4 Operating System	Limitations for Windows XP added
2010-10-21	1.2	Chapter 2.1.1 Chapter 2.2.6	New functions of DLL Release 1.24.0 added - Auto USB Detection enable function added - WiMOD_GetHCIMessage added
2010-10-28	1.3	Chapter 2.2.2 Reset	WiMOD_Reset corrected to WiMOD_ResetRequest

Table of Contents

Abbreviations

ADC	Analog-to-Digital Converter
DIO	Digital Input/Output
DLL	Dynamic Link Library
FW	Firmware
GPIO	General Purpose Input/Output
HCI	Host Controller Interface
HW	Hardware
RF	Radio Frequency
SPI	Serial Peripheral Interface
SW	Software
UART	Universal Asynchronous Receiver/Transmitter
WiMOD	Wireless Module

1 Introduction

1.1 Purpose

This document specifies the WiMOD Dynamic Link Library Application Interface.

1.2 Overview

This DLL enables Windows applications to configure and control several kinds of WiMOD devices in an easy and common way without knowing the underlaying communication protocol in detail.

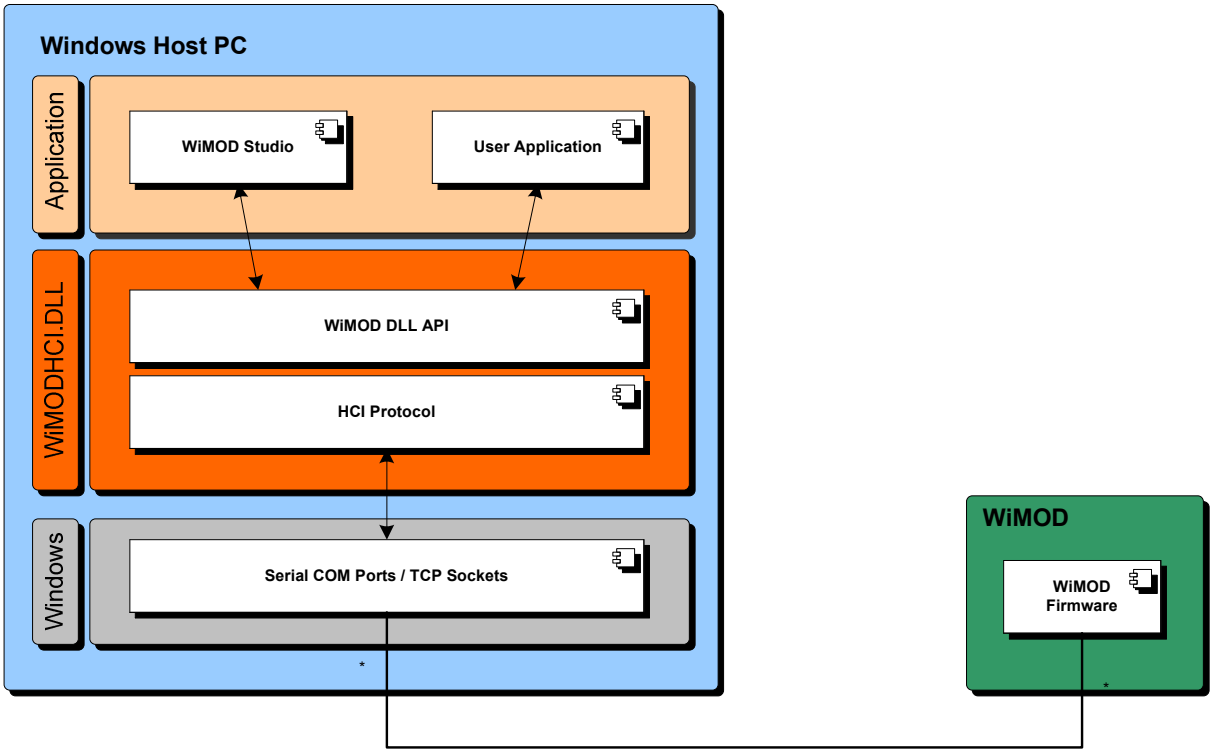

Fig. 1-1: Overview

Several service functions are implemented which create the required Host Controller Interface (HCI) messages and use the WiMOD HCI Protocol to communicated with an attached WiMOD device. These HCI messages will be send through an open communication channel which could be a Serial Comport (COM1…X) or an TCP Socket.

1.3 Applicable Documents

[1]	WiMODHCI_Spec.pdf	Specification of the WiMOD Host Controller Interface.
[2]	iMxxx_Data_Sheet.pdf	Data sheet for the corresponding device xxx
[3]	iMxxx_Settings.pdf	Parameter Settings and Examples for the corresponding device xxx
[4]	WiMODDLL_Spec.pdf	Specification of the WiMOD DLL

1.4 Operating System

The WiMODHCI.DLL is developed and tested for Microsoft Windows XP (SP3).

2 Functions

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

- General DLL Functions which are not related to a WiMOD device
- General WiMOD Device Functions which are common to all WiMOD devices

2.1 General DLL Functions

This chapter covers functions which are only related to the WiMOD DLL itself.

2.1.1 WiMOD Connection Handle

The basic mean for communication with a dedicated connected WiMOD device is a WiMOD Connection Handle (TWiMODHandle). A handle can be obtained by different services which are listed below. In general the WiMOD DLL supports access to multiple connected WiMOD devices.

2.1.1.1 Open Comport

This function opens a Serial Communication Port. On success the function returns a WiMOD Connection Handle which can be used to communicate with the connected WiMOD.

Prototype

- TWiMODHandle WiMOD_Open(const char *comPort)

Parameter

- comPort pointer to Comport name e.g. “COM1”

Return Value

- 0 invalid handle
- else valid handle

Note: TWiMODHandle is defined as unsigned long

2.1.1.2 Close Comport

This function closes an open Communication Port.

Prototype

- bool WiMOD_Close(TWiMODHandle handle)

Parameter

- handle communication handle

Return Value

- true comport closed
- false error

2.1.1.3 Discover Devices

This function starts a device discovery procedure which checks for connected WiMOD devices. The procedure is implemented as a background task and iterates over all available Serial Comports which are listed in the Windows Registry. Each available Serial Comport will be opened and a Ping command will be send to test if a WiMOD is connected. After finishing the procedure sends a message (WIMOD_MSG_DEVICE_CHANGE_IND) to a registered message handler to indicate that the result can be read now by means of the function WiMOD_GetDiscoveredDevices.

Prototype

bool WiMOD_DiscoverDevices()

Parameter

- none

Return Value

- true procedure started
- false procedure not started

Note: The Device Discovery procedure can also start automatically (see WiMOD_EnableAutoUSBDiscovery) if a USB device is connected or disconnected to the Windows-PC.

2.1.1.4 Get Discovered Devices

This function returns a list with WiMOD Connection Handles of currently connected WiMOD devices. The list is automatically updated due to one of the following events:

- WiMOD_DiscoverDevices called by user
- USB Device Change Windows messages
- WiMOD TCP Socket connection / disconnection if Server Mode is enabled

Prototype

- UINT32 WiMOD_GetDiscoveredDevices(TWiMODHandle* handles,
UINT32 numHandles)

Parameter

- handles pointer to an array for handles
- numHandles maximum number of array entries

Return Value

- number of discovered devices
- updated handle array

2.1.1.5 Enable/Disable Automatic USB Discovery

This function enables/disables the automatic Device Discovery procedure. In case of a connecting or disconnecting USB device the DLL starts the discovery procedure automatically. After finishing a message (WIMOD_MSG_DEVICE_CHANGE_IND) is sent to a registered message handler to indicate that the result can be read now by means of the function WiMOD_GetDiscoveredDevices.

Prototype

bool WiMOD_EnableAutoUSBDiscovery(bool enable)

Parameter

- enable true: feature enabled
false: feature disabled

Return Value

- true feature enabled/disabled
- false error

Note: This function is available in DLL Release 1.24.0

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

- TWiMODHCIError WiMOD_GetLastError(TWiMODHandle 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 WiMOD_GetErrorString (TWiMODHCIError error,
  char* resultString,
  int size)

Parameter

- error an error code
- resultString pointer to an character array
- size size in bytes of the character array

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

Return Value

- an updated resultString

2.1.2.3 Error Codes and Strings

The following table shows the implemented error codes and strings:

Error Codes	Error Strings
0	“WIMOD_NO_ERROR”
1	"WIMOD_INVALID_COMPORT"
2	"WIMOD_CONNECTION_BUSY"
3	"WIMOD_INVALID_HANDLE"
4	“WIMOD_NO_RESPONSE"
5	"WIMOD_TRANSMIT_ERROR”
6	"WIMOD_UNEXPECTED_FRAMELENGTH"
7	"WIMOD_COMMAND_NOT_SUPPORTED"
8	"WIMOD_COMMAND_FAILED"
9	"WIMOD_UNEXPECTED_STATUS"
10	"WIMOD_UNEXPECTED_RESPONSE_CODE"
11	"WIMOD_FWDL_PAGE_CRC_ERROR"
12	"WIMOD_FWDL_PAGE_ADDRESS_ERROR"
13	"WIMOD_FWDL_FIRMWARE_CRC_ERROR"
14	"WIMOD_FWDL_UNKNOWN_ERROR"
15	"WIMOD_FWDL_TIMEOUT_ERROR"
16	"WIMOD_FWDL_INPUT_FILE_ERROR"
17	"WIMOD_UNEXPECTED_COMMAND_ERROR"
18	"WIMOD_INVALID_PARAMETER"
19	"WIMOD_HOST_PROTOCOL_OUTDATED"
20	"WIMOD_FIRMWARE_PROTOCOL_OUTDATED"
21	"WIMOD_FWDL_UNSUPPORTED_WIMOD"

2.1.3 Get DLL Version

This function returns the DLL Version string.

Prototype

- bool WiMOD_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 the DLL is waiting for a device response. The maximum wait time in milliseconds can be configured by means of this function.

Prototype

- bool WiMOD_SetTimeout(TWiMODHandle handle,
  UINT32 timeout)

Parameter

- handle communication handle
- timeout timeout in milliseconds

Note: the default value is 500 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 WiMOD 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 WiMOD_RegisterMsgHandler (TWiMOD_CbMsgHandler cbMsgHandler)

Parameter

- cbMsgHandler pointer to message handler procedure

The function pointer type is defined as follows:

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

Return Value

- none

The following messages are implemented:

Message	Value	Parameter	Description
WIMOD_MSG_DEBUG_IND	0x00000001	none	a debug message is available
WIMOD_MSG_DEVICE_CHANGE_IND	0x00000002	None	a WiMOD device has connected / disconnected
WIMOD_MSG_HCI_MESSAGE_IND	0x00000004	Handle	a HCI message is available and should be read by means of function WiMOD_GetHCIMessage

2.1.6 Shutdown

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

Prototype

- bool WiMOD_DLLShutdown()

Parameter

- none

Return Value

- always true

2.2 General WiMOD Device Functions

2.2.1 Ping

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

Prototype

- bool WiMOD_Ping(TWiMODHandle handle)

Parameter

- handle communication handle

Return Value

- true device connected and alive
- false error

2.2.2 Reset Request

This function can be used to reset a WiMOD device.

Prototype

- bool WiMOD_ResetRequest(TWiMODHandle* handle)

Parameter

- handle communication handle

Return Value

- true device will be reset in approx. 500ms
- false error

2.2.3 Device Information

For identification purpose the WiMOD 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 WiMOD_GetDeviceInfo(TWiMODHandle* handle,
  TWiMODDeviceInfo* deviceInfo)

Parameter

- handle communication handle
- deviceInfo pointer to device info structure

Return Value

- true updated device info structure
- false error

2.2.3.2 Device Information Block

The Device Info structure contains the following information elements:

```
typedef struct
{
  UINT16    DeviceAddress;      //    16-Bit Device Address
  UINT8     ModuleType;         //    Module Identifier
  UINT8     DeviceMode;         //    Device Mode Identifier
  UINT8     FWVersion;          //    Firmware Version
  UINT8     HCIProtocolVersion; //    HCI Protocol Version
}TWiMODDeviceInfo;
```

Parameter	Description
Device Address	Unique 16-Bit value for addressing purpose
Module Type	8-Bit value which identifies the connected WiMOD device: The following Modules are supported: iM820A : 1 iM240A : 3 iM860A : 7
Device Mode	8-Bit value which identifies the configured device mode (see Device Parameter Block for details)

FWVersion	8-Bit value which identifies the programmed firmware: Bits 7..4 -> major FW Version number Bits 3..0 -> minor FW Version number Example: 0x14 -> FW Version = 1.4
HCIProtocolVersion	This value will be incremented on either DLL side or Firmware side to indicate an incompatibility within the underlying HCI protocol (see WiMOD Error Codes). An error code will be returned if the DLL detects a mismatch to the protocol of the connected WiMOD (see Error Codes).

2.2.4 Device Configuration

The WiMOD 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 WiMOD start-up and used to configure the firmware components and the hardware units.

2.2.4.1 Get Device Parameter

This function can be used to readout the configuration parameters.

Prototype

- bool WiMOD_GetDeviceParam(TWiMODHandle* handle,
  TWiMODDeviceParam* deviceParam)

Parameter

- handle communication handle
- deviceParam pointer to device parameter structure

Return Value

- true updated device parameter structure
- false error

2.2.4.2 Set Device Parameter

This function can be used to change several system parameters which are stored in a none volatile memory (e.g. Flash or EEPROM). The new values are used after a system reset.

Prototype

- bool WiMOD_SetDeviceParam(TWiMODHandle* handle,
  TWiMODDeviceParam* deviceParam)

Parameter

- handle communication handle
- deviceParam pointer to device parameter structure

Return Value

- true device parameter stored in memory
- false error

Note: To activate a new parameter set a Reset Command must be executed.

2.2.4.3 Device Parameter Block

The device parameter structure contains the following information elements:

```
typedef struct
{
   UINT8   IIFlag;              // Information Indicator Flag
   UINT8   Reserved_1;          // reserved: must be set to zero (0)
   UINT8   NetworkAddress;      // Network Address (0x01 .. 0xFE)
   UINT16  DeviceAddress;       // Device Address  (0x0001 .. 0xFFFE)
   UINT8   RF_DataRate;         // RF Datarate
   UINT8   RF_PowerLevel;       // RF PowerLevel
   UINT8   RF_Channel;          // RF Channel
   UINT8   DeviceMode;          // Device Mode
   UINT8   AckNumRetries;       // Number of retries for acknowledged data exchange
   UINT8   AckTimeoutTicks;     // Acknowledge Timeout in 10ms ticks
   UINT8   Reserved_2;          // reserved: must be set to zero (0)
   UINT8   Reserved_3;          // reserved: must be set to zero (0)
   UINT8   Reserved_4;          // reserved: must be set to zero (0)
}TWiMODDeviceParam;
```

The following parameters are device specific and can be found in [3]:

Parameter	Description
RF_DataRate	Index for RF Datarate
RF_PowerLevel	Index for RF Powerlevel
RF_Channel	Index for RF Channel

The following parameters are common to all WiMODs:

Parameter	Description
IIFlag	Information Identifier Flag (Bits 7..0): This bit field defines which of the parameters within the structure are valid. Only those parameters are recognized whose corresponding IIFlag-Bit is set to 1. A parameter with IIFlag-Bit cleared will not be transmitted to the connected WiMOD device. Bit 0: Network Address Bit 1: Device Address Bit 2: RF DataRate Bit 3: RF PowerLevel Bit 4: RF Channel Bit 5: DeviceMode Bit 6: AckNumRetries Bit 7: AckTimeoutTicks
NetworkAddress	A Network Address is used to separate groups of WiMODs from each other. A device accepts RF messages which contain its own Network Address or the BROADCAST_NETWORK_ADDRESS 0xFF. The values 0 and 0xFF are reserved. Note: Sniffer devices perform no Network Address filtering.
DeviceAddress	The Device Address is used to address a certain device within a group of devices with same RF settings. Therefore the device address must be set to a unique value to ensure proper operation. A device accepts RF messages which contain its own Device Address or the BROADCAST_DEVICE_ADDRESS 0xFFFF. The values 0 and 0xFFFF are reserved. Note: Repeater devices and Sniffer devices perform no Device Address filtering
DeviceMode	The firmware provides different RF operation modes: 0 = End Device (Default Configuration) 1 = Reserved 2 = Repeater: device which simply retransmits every received message 3 = Sniffer: sends every received message to its connected host
AckNumRetries	Defines the maximum number of retransmissions for RF messages send by means of WiMOD_DataRequest with ackFlag = true.
AckTimeoutTicks	Defines the timeout window (in 10ms ticks) the sender is waiting for an ACK message before a possible retransmission is initiated

2.2.4.4 Factory Reset

This function can be used to reset the WiMOD device configuration to its default factory settings. A configured device address will not be changed by this function.

Note: The new configuration gets active after reboot.

Prototype

- bool WiMOD_FactoryReset(TWiMODHandle* handle,
  bool rebootFlag)

Parameter

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

Return Value

- true device configuration reset
- false error

Note: The default values are module specific and can be found in [3].

2.2.5 Operation Modes

The WiMOD firmware can operate in different kind of 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 WiMOD start-up 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 WiMOD_GetOpMode(TWiMODHandle   handle,
                       UINT8*         opMode);
```

Parameter

- handle communication handle
- opMode pointer to return value

Return Value

- true operation mode read
- false error

2.2.5.2 Set Next Operation Mode

This function sets the next System Operation Mode which gets active after reset.

Prototype

```
- bool WiMOD_SetNextOpMode(TWiMODHandle   handle,
                           UINT8          nextOpMode,
                           const char*    password,
                           bool           rebootFlag);
```

Parameter

- handle communication handle
- nextOpMode next System Operation Mode
- password password string
- rebootFlag indicates if system should reboot or not

Return Value

- true operation mode set
- false error

Note: This function requires a password. A reboot takes place after approx. 500ms.

2.2.5.3 System Operation Modes

The following System Operation Modes are supported:

Value	Description
0	Standard Application Mode / Default Mode
1	Hardware Test Mode
2	Production Mode
3	Self test Mode

2.2.6 Data Exchange Services

The following functions are available to send RF messages from one host device to another.

2.2.6.1 Data Request

This function can be used to send user defined messages to a given destination (peer) device.

Prototype

```
- bool WiMOD_DataRequest(TWiMODHandle    handle,
                         UINT16          dstDevice,
                         UINT8           length,
                         UINT8*          payload,
                         bool            ackFlag);
```

Parameter

- handle — communication handle
- dstDevice — address of destination (WiMOD) device
- length — number of bytes in attached payload
- payload — pointer to payload
- ackFlag — switch for data service type:
  - false: data packet will be sent without acknowledgement / retransmissions
  - true: data packet will be sent with acknowledgement / retransmissions

Note: The number of retransmissions is configurable (see Set Device Parameter).

Return Value

- true — transmission started
- false — error

2.2.6.2 Get HCI Frame

A WiMOD device is allowed to send HCI event messages to its host at any time. Those messages which are not a response to a preceding command message are buffered within the DLL. For each buffered message a notification is send to the registered message handler (see Register Message Handler). This client should readout the buffered HCI message by means of the following function. RF messages witch are send by means of WiMOD_DataRequest from another Host PC will be handled in the same way.

Prototype

- bool WiMOD_GetHCIFrame(TWiMODHandle* handle,
TWiMODHCIFrame* hciFrame)

Parameter

- handle communication handle
- hciFrame pointer to hciFrame

Return Value

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

Note: This function might be called periodically to poll for buffered messages if message handler registration can not be realized.

2.2.6.3 TWiMODHCIFrame

The WiMOD HCI message format includes the following elements

```
typedef struct
{
        UINT8  Res1;          // Reserved
        UINT8  Res2;          // Reserved
        UINT8  DstID;         // Destination Endpoint Identifier
        UINT8  SrcID;         // Source Endpoint Identifier
        UINT8  MsgID;         // Message Identifier
        UINT8  Length;        // Size of Payload
        UINT8  Payload[256];  // Payload Field (max. 256 Bytes)
        UINT16 Res3;          // Reserved
        UINT8  Res4;          // Reserved
}TWiMODHCIFrame;
```

Parameter	Description
DstID	Destination Endpoint Identifier: Identifies the logical receiver endpoint of a given HCI message
SrcID	Source Endpoint identifier Identifies the logical sender endpoint of a given HCI message
MsgID	Identifies the type of a HCI message

Length	Defines the number of valid bytes within the reserved payload field
Payload	Field for message dependend data

Note: DstID and SrcID are used to implement independent service access points

2.2.6.4 Get HCI Message

This function can be used as an alternative to GetHCIFrame. The user can pass a pointer to a character array instead of a TWiMODHCIFrame structure.

This function is availabe in DLL Release 1.24.0

Prototype

- bool WiMOD_GetHCIMessage (TWiMODHandle* handle,
  UINT8* msgBuffer,
  UINT16 bufferSize)

Parameter

- handle communication handle
- msgBuffer pointer to message buffer
- bufferSize size of message buffer

Return Value

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

Message Buffer Layout

The msgBuffer looks as follows after successful receiption:

Offset	Content
0x00	DstID = Destination Endpoint Identifier
0x01	SrcID = Source Endpoint Identifier
0x02	MsgID = Message Identifier
0x03	Length = Size of Payload Field in bytes
0x04 …	Message dependend Payload Field

Note: A bufferSize of 256 Byte is recommended.

2.2.6.5 Receiving User Defined Messages

HCI messages which are transmitted by means of WiMOD_DataRequest from one Host PC can be readout by means of WiMOD_GetHCIMessage on the receiver side. These messages carry the DstID = 0x91 (DATAEXCHANGE_ID) and MsgID = 0x03 (UDATA_INDICIATION) or 0x06 (RDATA_INDICATION). Furthermore the first two bytes of the Payload field contain the device address of the sender device.

3 Important Notice

3.1 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.

Contact us to get information about the Declaration of Conformity.

3.2 Contact Information

IMST GmbH

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

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