WiMOD-LR_DevTool User Guide

Getting started: WiMOD-LR_DevTool

Document ID: 4100/40140/0107

Category: confidential

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

Document Information

File name	UserGuide_WiMOD_LR_Dev-Tool.docx
Created	2015-04-19
Total pages	18

Revision History

Version	Description
1.0	Initial version.

Aim of this Document

Aim of this document is to give some quick start instructions how to start working with the WiMOD-LR_DevTool.

Confidentiality Note

This document has to be treated confidentially. Its content must not be published, duplicated or passed to third parties without our express permission.

1 Introduction

The WiMOD-LR_DevTool is a tool for helping the user to understand the low level details of the Host Controller Interface (HCI) implemented in the WiMOD LR[1] wireless modules.

1.1 Target Audience

This tool is targeted at users who want to implement the host side of the communication channel to a WiMOD LR module.

Users who want to explore and use all the features of the WiMOD LR modules should use the user friendly PC tools, like the WiMOD_LoRaWAN_EndNode_Studio or WiMODLR_Studio provided by IMST. All tools are available on the web site https://wireless-solutions.de/.

2 Starting the WiMOD-LR_DevTool

The WiMOD-LR_DevTool is a single `.exe` file that does not need to be installed. There are only two requirements to be med in order to run the software.

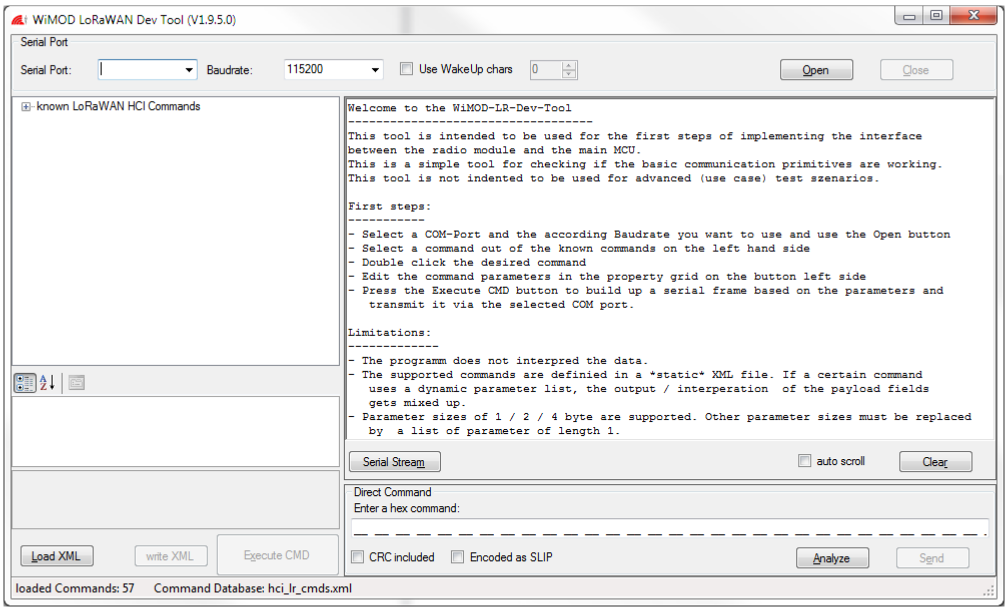
The first requirement is to have a preinstalled Microsoft .Net framework (Version 3.5 or newer) installed on a MS Windows PC[2]. Modern versions of MS Windows usually contain the .Net frameware already. Therefore the installation and configuration of the framework is out of the scope of this document. As second requirement there is a configuration file called `hci_lr_cmds.xml`. This file must be placed within the same folder as the .exe file.

If all requirements are met the user can start the tool by just double clicking on the `.exe` file.

2.1 Main Window

The main window looks like that:

[1] LR = Long Range. This family of WiMOD modules (currently) includes the firmware for LoRaWAN and WiMOD LR Base wireless protocols.
[2] The tool is based on the .Net framework. For Linux PCs there is a project called "mono" that allows the user to run .Net applications, too. However there are a few limitations running this tool on Linux. Most of the limitations are related to the placement of the GUI-elements in the main window. In some environments the elements are being placed on wrong positions. Besides that the application core can run on Linux, too. But has not been tested very well.



```
Welcome to the WiMOD-LR-Dev-Tool
-----------------------------------
This tool is intended to be used for the first steps of implementing the interface
between the radio module and the main MCU.
This is a simple tool for checking if the basic communication primitives are working.
This tool is not indented to be used for advanced (use case) test szenarios.

First steps:
-----------
- Select a COM-Port and the according Baudrate you want to use and use the Open button
- Select a command out of the known commands on the left hand side
- Double click the desired command
- Edit the command parameters in the property grid on the button left side
- Press the Execute CMD button to build up a serial frame based on the parameters and
   transmit it via the selected COM port.

Limitations:
------------
- The programm does not interpred the data.
- The supported commands are definied in a *static* XML file. If a certain command
   uses a dynamic parameter list, the output / interperation  of the payload fields
   gets mixed up.
- Parameter sizes of 1 / 2 / 4 byte are supported. Other parameter sizes must be replaced
   by  a list of parameter of length 1.
```

Figure 2-1: Screenshot of the main window

In the upper part of the main window the user can choose the serial port to which the WiMOD LR module is connected. The default baud rate is already setup for the vanilla firmware developed by IMST. The "Use WakeUp chars" option is only needed if the WiMOD LR module is configured to use automatic power saving[3]. The "Open" and "Close" buttons enable or disable the serial connection the radio module.

The left part of the window shows a list of available commands which can be used to send to the radio module. To select one command, navigate through the command tree and double click on one command. The details of the command will be displayed within the lower left part of the window. There the user can select individual parts of the payload and setup the desired values. In order to send the command to the module the "Execute CMD" button must be used.

The "Load XML" button can be used to load another XML based command definition file. (see Chapter 5.)

In the main field of the window shows all messages that have been exchanged with the radio module. Messages send by the PC are printed in a red color, while received messages (module to PC) are printed in blue color.

In the bottom part of the window there a group box named "Direct Command". The user can use the controls of this box to enter self defined hex values that should form valid commands for the radio module. The "Analyze" button tries to analyze the user data on the PC. The result will be printed in a green color. The "Send" button will transfer the user data (in form of an HCI frame) to the module.

[3] In automatic power saving mode the module needs a few edges on the UART interface to wake up before it is ready to receive (HCI) commands. A setting of 50 wakeup chars is recommended to be used.

3 HCI Basics

3.1 Overview

The WiMOD LR Base / LoRaWAN HCI protocol is designed to expose the radio module services to an external host controller. The communication between host and the radio (WiMOD LR) is based on so called HCI messages which can be sent through a UART interface (see Figure 3-1). The Application issues a command which is to be transferred to the peer device. The command plus its payload will be secured by a 16bit CRC[4] sum. (Sometimes the CRC Sum is called Frame Check Sequence (FCS)) After the CRC sum has been added the complete frame will be SLIP[5] encoded before it will be send out to the peer. On the peer device the processing is done the other way round.

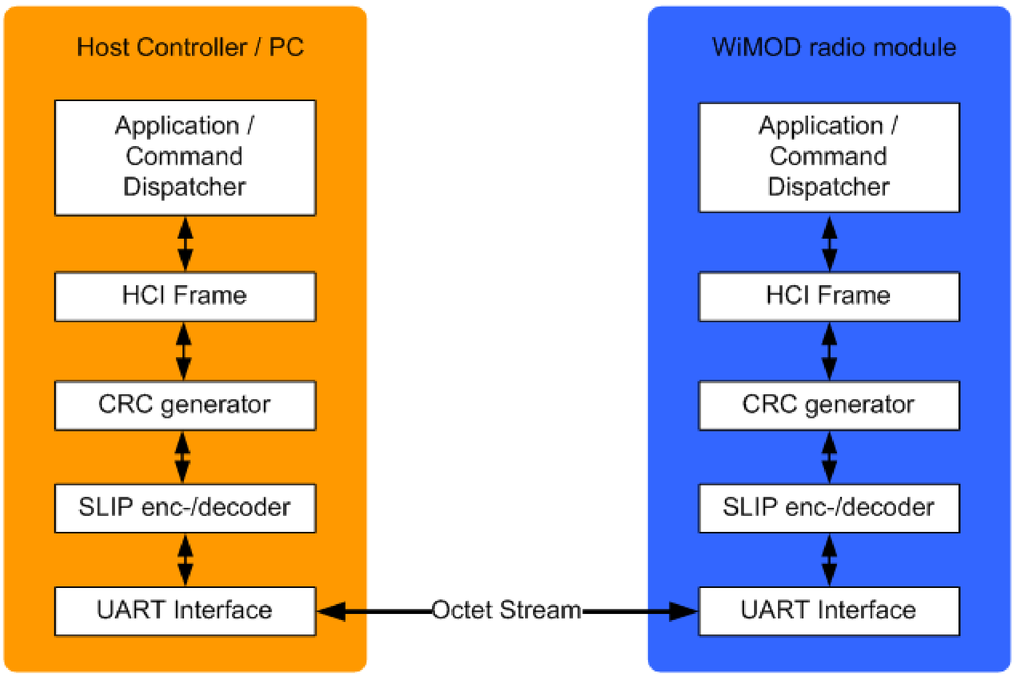


Figure 3-1: Host Controller Communication

3.2 HCI Communication

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

[4] See: https://en.wikipedia.org/wiki/Cyclic_redundancy_check
[5] See: https://en.wikipedia.org/wiki/Serial_Line_Internet_Protocol , https://tools.ietf.org/html/rfc1055

3.3 Message Flow

The HCI protocol defines three different types of messages which are exchanged between the host controller and the radio module:

1. Command Messages ("REQ"): always sent from the host controller to the WiMOD LR module to trigger a function.

2. Response Messages ("RSP"): sent from the radio module to the host controller to answer a preceding HCI request message.

3. Event Messages ("IND"): can be sent from the radio module to the host controller at any time to indicate an event or to pass data which was received over the radio link from a peer device (via radio link).

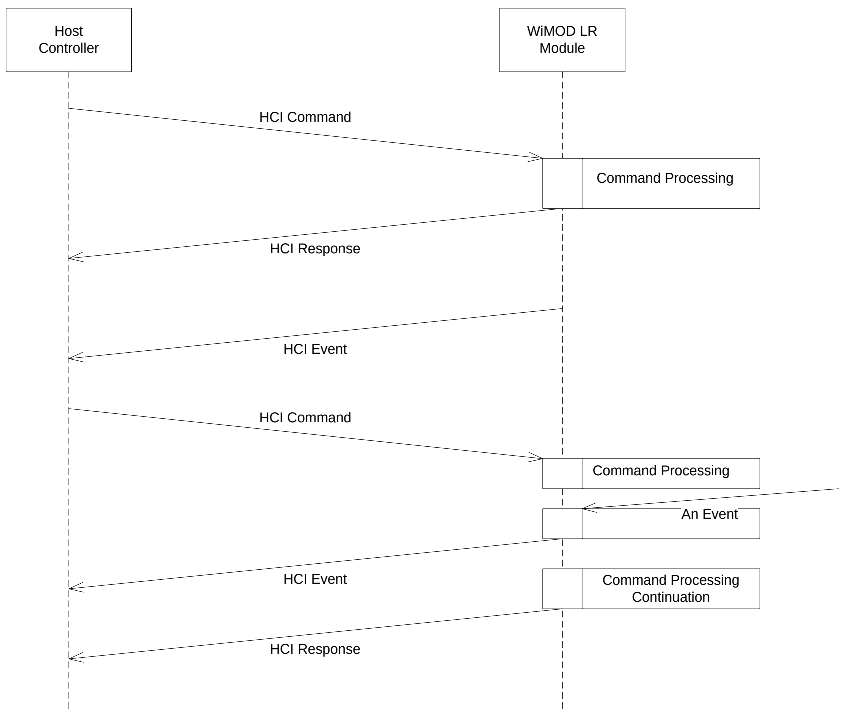


Figure 3-2: HCI Message Flow

3.4 HCI Message Format

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

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

Figure 3-3: HCI Message Format

3.4.1 Destination Endpoint Identifier (DstID)

This field identifies a logical service access point (endpoint) within a device. A service access point handles several messages of same kind.

3.4.2 Message Identifier (MsgID)

This field identifies a specific type of message and is used to trigger a corresponding service function or to indicate a service response when sent to the host controller.

3.4.3 Payload Field

The Payload Field has variable length and transports message dependent parameters. The length of this field is in the range $0 \leq n \leq 300$[6] Bytes.

3.4.4 Communication over UART

The standard host controller communication interface is a UART interface. The WiMOD LR HCI Protocol uses a SLIP wrapping layer when transmitted over asynchronous serial interfaces (UART).

3.4.4.1 SLIP Wrapper

The SLIP layer provides a mean to transmit and receive complete data packets over a serial communication interface. The SLIP coding is according to RFC 1055 (see http://www.faqs.org/rfcs/rfc1055.html for details)

The basic idea of the SLIP Wrapper goes like this:

[6] This is an approximation. The exact value is defined by the running firmware.

- Use a special character as start-of-frame and end-of-frame tag. (This tag is called “SLIP END”)
- Escape all occurrences of this SLIP END character in the data to send with a special escape sequence.

Using this simple scheme enables the receiver to synchronize itself to the start of a frame.

The next figure explains how a HCI message is embedded in a SLIP packet.

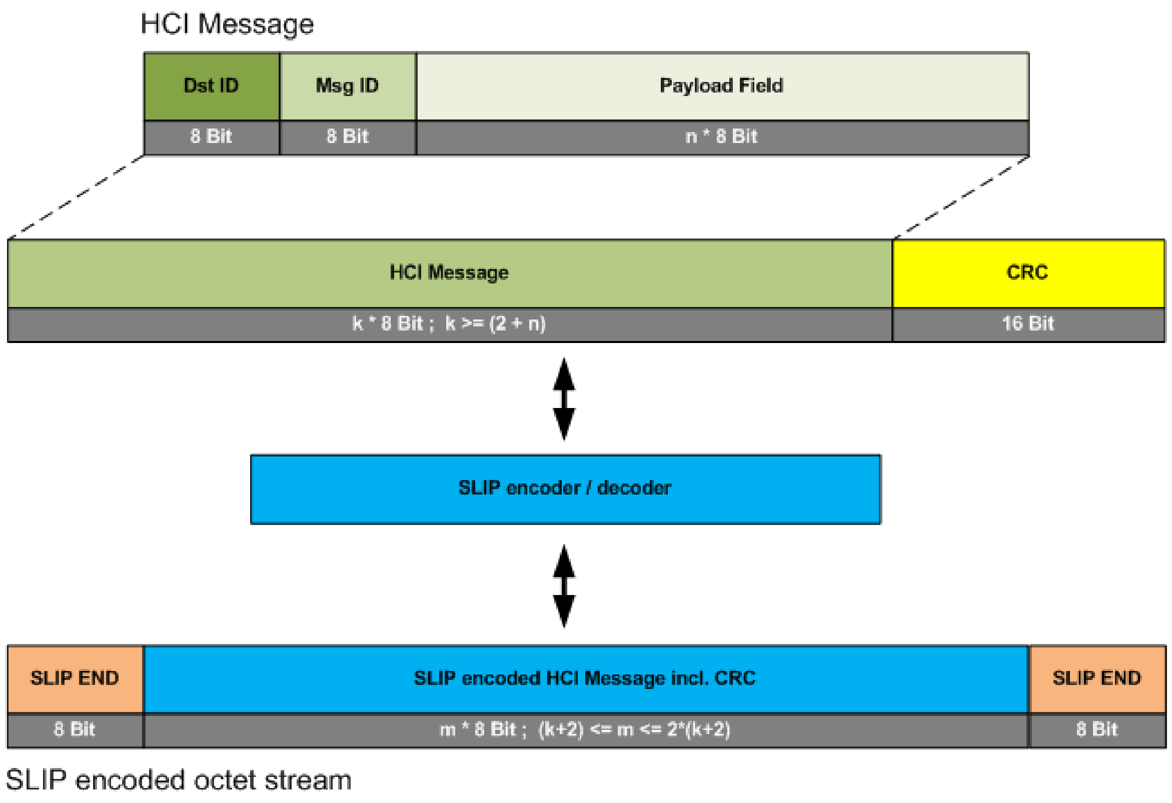


Figure 3-4:Communication over UART

Note: The variable payload length is not explicitly transmitted over the UART communication link. Indeed it can be derived from the SLIP wrappers receiver unit.

3.4.5 Frame Check Sequence Field : CRC Sum

Following the HCI message a 16-Bit Frame Check Sequence (FCS) is added to support a reliable packet transmission. The FCS contains a 16-Bit CRC-CCITT cyclic redundancy check which enables the receiver to check a received packet for bit errors. The CRC computation starts from the Destination Endpoint Identifier Field and ends with the last byte of the Payload Field. The CRC is added before SLIP encoding.

3.4.6 Physical Parameters of the UART Interface

The default UART settings for the iM880A/B, iU880A/B, iM881A are:

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

4 Using the WiMOD-LR_DevTool

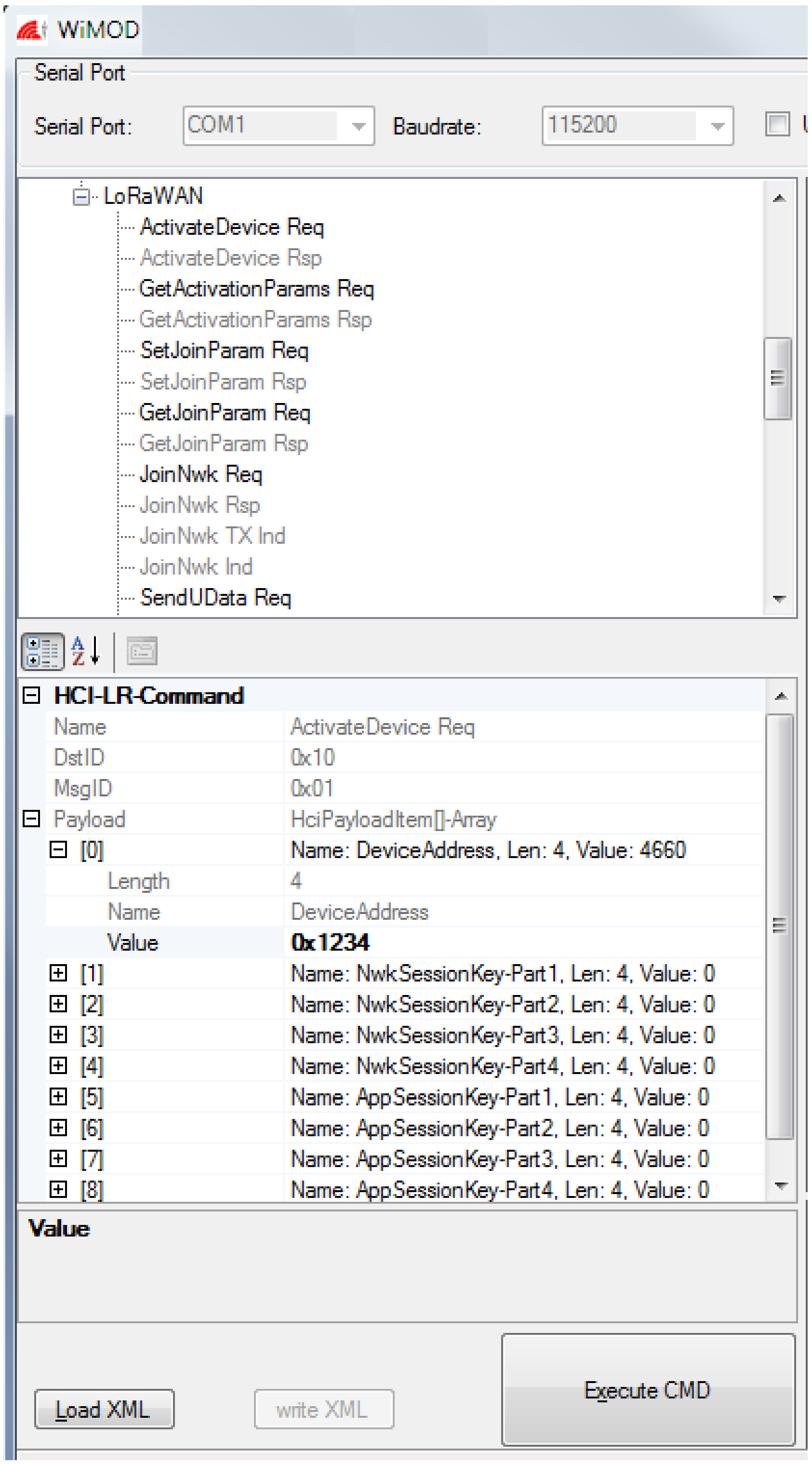
This chapter gives a short introduction how to use the WiMOD-LR_DevTool.

4.1 Selecting the UART Interface

In order to communicate with the WiMOD LR radio module a serial interface must be selected and opened. By clicking on the drop-down list for serial ports, a list of available ports is being displayed. Select the one connected the radio module and click on the “Open” button.

4.2 Using the pre-defined Commands

A List of available Commands has been loaded at program start. By navigating through the command on the left the user can choose one command to be executed. By double clicking on the selected item the current command will be displayed in the property grid on the left bottom of the main window. The user can navigate within the property grid throughout the payload items (if any) of the command and manipulate the values. Figure 4-1 depicts an example command. The parameter called “Device Address” has been set to 0x1234 by the user.

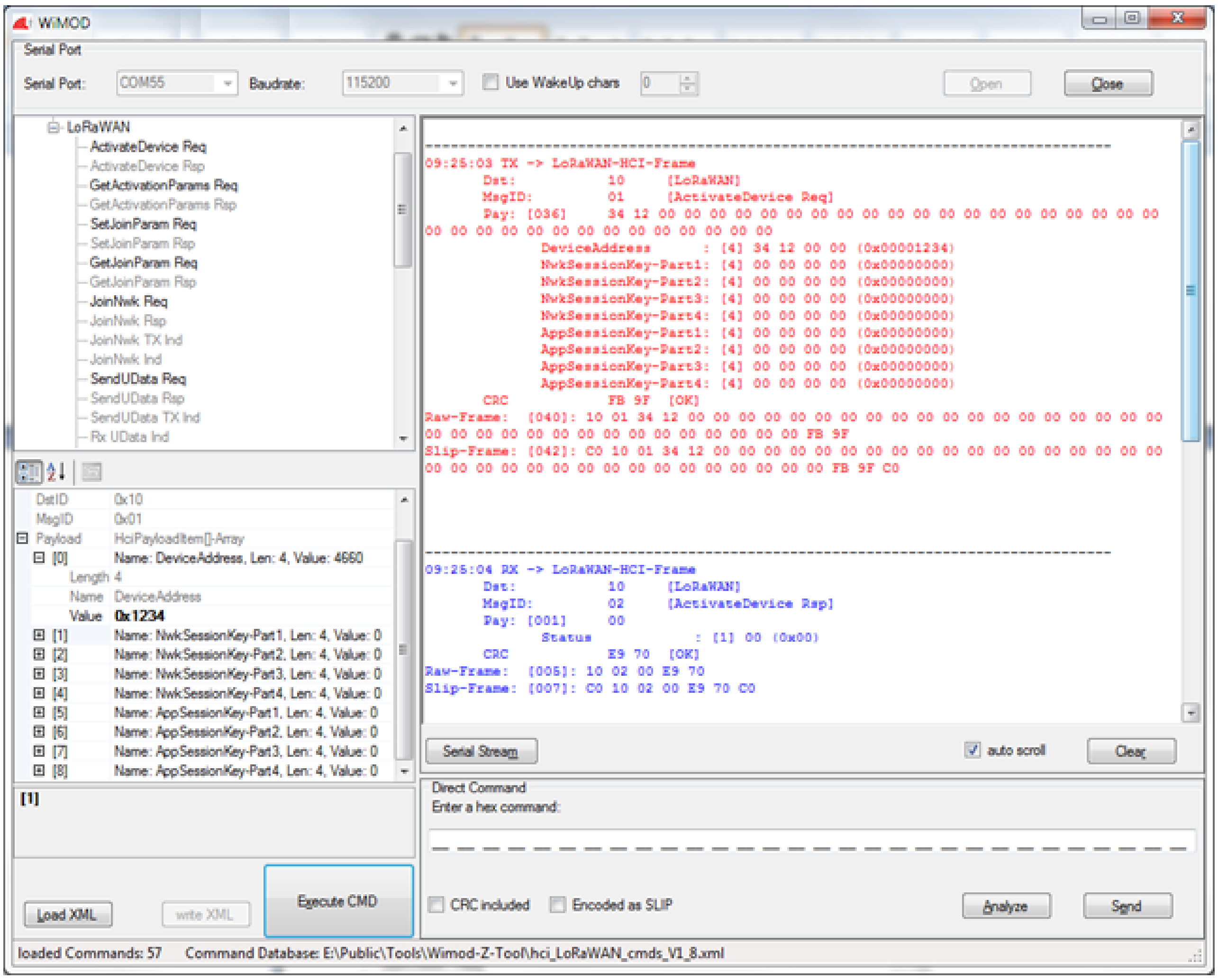


HCI-LR-Command	
Name	ActivateDevice Req
DstID	0x10
MsgID	0x01
Payload	HciPayloadItem[]-Array
[0]	Name: DeviceAddress, Len: 4, Value: 4660
Length	4
Name	DeviceAddress
Value	0x1234
[1]	Name: NwkSessionKey-Part1, Len: 4, Value: 0
[2]	Name: NwkSessionKey-Part2, Len: 4, Value: 0
[3]	Name: NwkSessionKey-Part3, Len: 4, Value: 0
[4]	Name: NwkSessionKey-Part4, Len: 4, Value: 0
[5]	Name: AppSessionKey-Part1, Len: 4, Value: 0
[6]	Name: AppSessionKey-Part2, Len: 4, Value: 0
[7]	Name: AppSessionKey-Part3, Len: 4, Value: 0
[8]	Name: AppSessionKey-Part4, Len: 4, Value: 0

Figure 4-1: Screenshot of the left part of the main window

If all payload items have been setup, the user can click on the "Execute CMD" button. The command will be send to the radio module.

The following screen shot will show the message exchange in the main part of the main window:



DstID	0x10
MsgID	0x01
Payload	HciPayloadItem[]-Array
[0]	Name: DeviceAddress, Len: 4, Value: 4660
Length	4
Name	DeviceAddress
Value	0x1234
[1]	Name: NwkSessionKey-Part1, Len: 4, Value: 0
[2]	Name: NwkSessionKey-Part2, Len: 4, Value: 0
[3]	Name: NwkSessionKey-Part3, Len: 4, Value: 0
[4]	Name: NwkSessionKey-Part4, Len: 4, Value: 0
[5]	Name: AppSessionKey-Part1, Len: 4, Value: 0
[6]	Name: AppSessionKey-Part2, Len: 4, Value: 0
[7]	Name: AppSessionKey-Part3, Len: 4, Value: 0
[8]	Name: AppSessionKey-Part4, Len: 4, Value: 0

[1]

```
-------------------------------------------------------------------------------
09:25:03 TX -> LoRaWAN-HCI-Frame
       Dst:           10      [LoRaWAN]
       MsgID:         01      [ActivateDevice Req]
       Pay: [036]     34 12 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00
              DeviceAddress      : [4] 34 12 00 00 (0x00001234)
              NwkSessionKey-Part1: [4] 00 00 00 00 (0x00000000)
              NwkSessionKey-Part2: [4] 00 00 00 00 (0x00000000)
              NwkSessionKey-Part3: [4] 00 00 00 00 (0x00000000)
              NwkSessionKey-Part4: [4] 00 00 00 00 (0x00000000)
              AppSessionKey-Part1: [4] 00 00 00 00 (0x00000000)
              AppSessionKey-Part2: [4] 00 00 00 00 (0x00000000)
              AppSessionKey-Part3: [4] 00 00 00 00 (0x00000000)
              AppSessionKey-Part4: [4] 00 00 00 00 (0x00000000)
       CRC            FB 9F   [OK]
Raw-Frame:  [040]: 10 01 34 12 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FB 9F
Slip-Frame: [042]: C0 10 01 34 12 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FB 9F C0


-------------------------------------------------------------------------------
09:25:04 RX -> LoRaWAN-HCI-Frame
       Dst:           10      [LoRaWAN]
       MsgID:         02      [ActivateDevice Rsp]
       Pay: [001]     00
              Status             : [1] 00 (0x00)
       CRC            E9 70   [OK]
Raw-Frame:  [005]: 10 02 00 E9 70
Slip-Frame: [007]: C0 10 02 00 E9 70 C0
```

loaded Commands: 57 Command Database: E:\Public\Tools\Wimod-Z-Tool\hci_LoRaWAN_cmds_V1_8.xml

Figure 4-2: Screenshot after sending a command

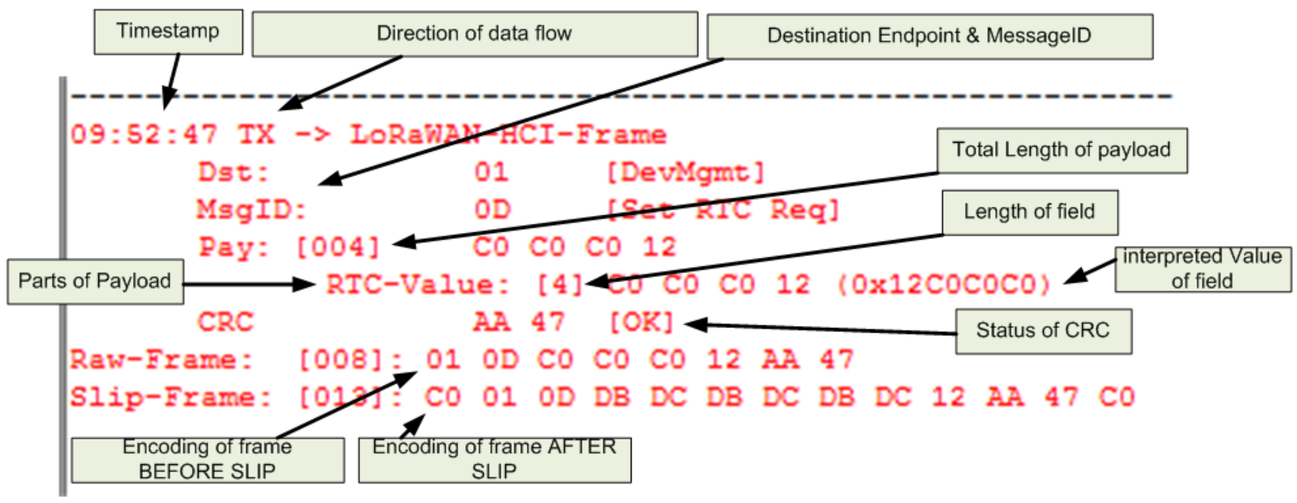
In the main part of the main window the transmitted HCI frame is printed in red. The parts of the frame (DstID, MsgID, Payload field and the CRC/FCS) are shown. Furthermore the complete frame is shown in raw format and in SLIP encoded format.

Please note: All numerical values are transferred in network order. The WiMOD-LR_DevTool shows the order of the bytes being transferred as well as the interpretation as numerical value. (See “DeviceAddress” in Figure 4-2)

After issuing the command the radio module will send back a response[7], which is printed in blue. In case of Figure 4-2 the response payload just contains a “status” field. The concrete meaning of the value of the status field is given in the HCI specification document. The WiMOD-LR_DevTool does not interpret the meaning of status or “status and format” fields.

The following picture explains the output in detail:

[7] A response frame is only generated if the command is known to the radio module. If an unknown command has been send no response will be generated.



```
------------------------------------------------------------
09:52:47 TX -> LoRaWAN-HCI-Frame
      Dst:          01      [DevMgmt]
      MsgID:        0D      [Set RTC Req]
      Pay: [004]    C0 C0 C0 12
         RTC-Value:  [4] C0 C0 C0 12 (0x12C0C0C0)
      CRC           AA 47   [OK]
Raw-Frame:  [008]: 01 0D C0 C0 C0 12 AA 47
Slip-Frame: [013]: C0 01 0D DB DC DB DC DB DC 12 AA 47 C0
```

Timestamp
Direction of data flow
Destination Endpoint & MessageID
Total Length of payload
Length of field
interpreted Value of field
Parts of Payload
Status of CRC
Encoding of frame BEFORE SLIP
Encoding of frame AFTER SLIP

Figure 4-3: Description of the printed values

The command shown in Figure 4-3 is only used as example. The RTC-value of the payload field contains some 0xC0 (= SLIP END) chars. Within the “raw-frame” the 0xC0 chars are still present and unchanged. After the SLIP encoding stage the 0xC0 bytes have been replaced (“escaped”) by a special sequence. That is why the frame transferred to the device contains more bytes than the raw frame.

The concrete algorithm for SLIP encoding / decoding is given in the official RFC 1055 document. The algorithm for the CRC calculation is given in the HCI specification document referring to the corresponding firmware running on the module. Both algorithms are out of scope of this document.

4.3 Generating Commands via Direct Input

Besides clicking on one command in the command tree on the left hand side of the main window, the WiMOD-LR_DevTool offers the possibility to construct a message frame by entering hex-values into the edit field in the lower part of the main window.

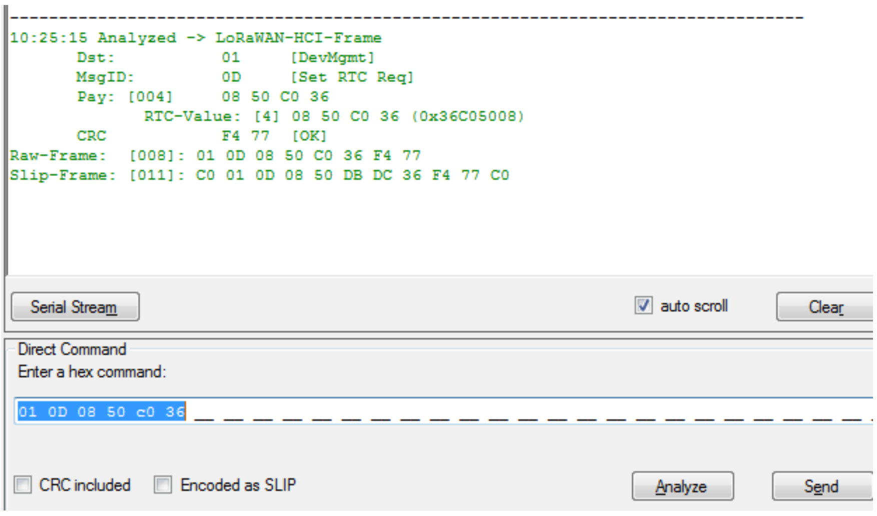
There the user can enter all parts of a message directly. For example entering the bytes

`“01 0D 08 50 c0 36”`

will construct a “Set RTC Req” command using a payload of “`08 50 c0 36`” which is used as RTC timestamp.

If the checkboxes “CRC included” and “Encoded as SLIP” are off, the WiMOD-LR_DevTool will automatically calculate the right CRC sum and add it to the message. After that the SLIP encoding stage is done automatically, too.

A click on the “Analyze” button will show the result (see Figure 4-4). A click on the “Send” button will transfer the message to the module.



```
--------------------------------------------------------------------------------
10:25:15 Analyzed -> LoRaWAN-HCI-Frame
       Dst:           01      [DevMgmt]
       MsgID:         0D      [Set RTC Req]
       Pay: [004]     08 50 C0 36
              RTC-Value: [4] 08 50 C0 36 (0x36C05008)
       CRC            F4 77   [OK]
Raw-Frame:  [008]: 01 0D 08 50 C0 36 F4 77
Slip-Frame: [011]: C0 01 0D 08 50 DB DC 36 F4 77 C0
```

Figure 4-4: Screenshot showing the direct command feature

5 Customizing the Command Description File

By default a command definition file called `hci_lr_cmds.xml` will be loaded by the WiMOD-LR_DevTool at startup.

The user can define a new command definition file by creating a new file based on the default one. The file is based on a simple XML syntax.

As simple example listing is given below:

```xml
<?xml version="1.0" encoding="utf-8"?>

<HCI_CMDs xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Endpoints>

    <WmCommandEndpoint Name="DevMgmt" ID="1">

        <WmCommands>
            <WmCommand Name="Ping Req" ID="1">
                <Payload>
                </Payload>
            </WmCommand>

            <WmCommand Name="Ping Rsp" ID="2">
                <Payload>
                    <WmPayload Name="Status    " Length="1" />
                </Payload>
            </WmCommand>
        </WmCommands>
    </WmCommandEndpoint>
  </Endpoints>
</HCI_CMDs>
```

After the xml root node "HCI_CMDS" the WiMOD-LR_DevTool expects a List of endpoint objects. Each endpoint object can contain any number of specified commands. A command is enclosed by the "WmCommand" tag. This tag must contain a "Name" and a "ID" attribute. The name is a human readable string. The ID is the decimal number of the HCI command ID defined in the corresponding HCI specification.

Each command can have any number of defined payload items. Each payload item is identified by a "WmPayload" XML tag. Each tag must contain a name and a length attribute. The name represents a human readable identifier for the corresponding item. The length is the number of octets/bytes used for the item. If the length is 1,2 or 4 byte the WiMOD-LR_DevTool will try to treat the item as numerical number which is transferred to or from the radio module in network byte order[8].

5.1 Limitations

Due to the simple design of the application core, there are some limitations to consider.

5.1.1 Parameters and Payload Values are set to 0x00 by Default

After the user selected a command (by double clicking on one command in the command tree) all values and parameters are set to 0x00 as default. Even if certain parameter values are defined to have another valid range (e.g. 1…5) the default value is set to 0x00. The core of the WiMOD-LR_DevTool has no knowledge of the meaning of the concrete values.

It is up to the user to read the corresponding HCI specification and enter valid parameters values. This applies to the "status" or "status / format" fields of the HCI response messages, too. The user must consult the HCI specification to interpret the given values.

5.1.2 Dynamic Payload Length

The tool is not able to process dynamic payload (length) very well. That means that the WiMOD-LR_DevTool needs to know the exact number of payload bytes in order to do the interpretation of the bytes. For most of the defined commands of the HCI specification this is sufficient. But for some commands which are designed for transferring a variable number of user defined payload bytes, this restriction is a problem. (A possible workaround is to adopt the command XML file for each length that is known to be used.)

5.1.3 Size of Single Payload Fields

The WiMOD-LR_DevTool is not able to process numerical values > 32 bit (4 byte). If a command expects bigger fields (e.g. 64 or 128 bit), the payload field has to be split up in to smaller pieces. A 128 bit field can be split up to 4 32 bit fields (, or 8 16 bit fields or 16 8 bit fields). An example is given in the default XML command file.

[8] See https://en.wikipedia.org/wiki/Endianness

As a general comment on the limitations: The main design goal of the tool was to show to principle work flow: plain data ↔ plain data + CRC ↔ SLIP coding.

The more convenient user software capable of handling this kind of commands is given by the other PC software provided by IMST (e.g. WiMOD_LoRaWAN_EndNode_Studio or WiMODLR_Studio)

6 Troubleshooting

The WiMOD_LoRaWAN_EndNode_Studio core logic does not interpret the values in a logical way. It is up to the user to interpret the data and react accordingly. The purpose of this tool is to encode / decode HCI frames as well as assisting the user to generate the right CRC fields.

However there are a few common pitfalls:

6.1 No Response Message from Module.

There is a bunch of reasons why a module does not send a response message back to the PC. Please consider checking this list:

- A wrong UART and or wrong USART parameters have been selected.
- The command send by PC is unknown to the module
- The automatic power saving feature is ative on the module. In this case try to activate the transmission of the wakeup chars. Select a value of 50.

6.2 There are no Commands shown in the Command Tree

The WiMOD_LoRaWAN_EndNode_Studio requires the command file to be in valid XML format. If there is an error in the XML coding, the tool will not load and parse the command description file. It is the task of the user to make sure that the file is in valid XML syntax.

6.3 “Unknown command” is displayed within the Tool

The WiMOD_LoRaWAN_EndNode_Studio displays the string “unknown” if a command is not contained within the command xml file. Make sure that all commands listed in the corresponding HCI specification is mentioned in the xml file.

6.4 Problems related to Variable Payload (Length)

As mentioned before the application core is kept very simple. Therefore the tool lacks some kind of smart support concerning variable payload length.

6.4.1 The Tool displays "ADDITIONAL DATA"

The string "ADDITIONAL DATA" is displayed if the module sent more data (within the payload field) than it was defined in the command definition file. For demo purposes the given commands define 4 bytes as user payload. If the module sends 10 byte, the WiMOD_LoRaWAN_EndNode_Studio does not find any information about the additional bytes and therefore the 5 bytes are just displayed with the tag "additional data". This is a limitation if of the tool. There is nothing wrong about the HCI message itself.

6.4.2 The Tool displays "Payload error"

The string "Payload error" is displayed if there is less data present as it is defined in the xml command definition file. For demo purposes the given commands define 4 bytes as user payload. If there are e.g. only 3 byte present, the WiMOD_LoRaWAN_EndNode_Studio will indicate this mismatch by showing a "payload error". It is dependent of the command if this is a real error, or only caused by the tool limitations about variable payload length.

7 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.

This Development Kit/Starter Kit does not fall within the scope of the European Union directives regarding electromagnetic compatibility, restricted substances (RoHS), recycling (WEEE), FCC, CE or UL, and therefore may not meet the technical requirements of these directives or other related directives.

8 Important Notice

8.1 Disclaimer

8.2 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