ANR009 PROTEUS-III / PROTEUS-III-SPI

ADVANCED DEVELOPER GUIDE

VERSION 1.3

AUGUST 18, 2021

The content of this document is valid for Proteus-III and Proteus-III-SPI. For reasons of simplicity, we only talk about Proteus-III in the following. Nevertheless, the same holds for Proteus-III-SPI.

Revision history

Manual version	FW version	HW version	Notes	Date
1.0	1.0	1.2	• Initial version	January 2020
1.1	1.0	1.2	• Added information on Bluetooth® stack version	January 2021
1.2	1.0	1.2	• Added the note that this document is also valid for Proteus-III-SPI	February 2021
1.3	1.3	1.2	• Added information about the new remote GPIO function (PWM and disconnected) • Better description of the RX and TX characteristic of the SPP-like profile	August 2021

Abbreviations and abstract

Abbreviation	Name	Description
BTMAC		Bluetooth® conform MAC address of the module used on the RF-interface.
CS	Checksum	Byte wise XOR combination of the preceding fields.
DTM	Direct test mode	Mode to test Bluetooth® specific RF settings.
GAP	Generic Access Profile	The GAP provides a basic level of functionality that all Bluetooth® devices must implement.
I/O	Input/output	Pinout description.
LPM	Low power mode	Mode for efficient power consumption.
MAC		MAC address of the module.
MTU	Maximum transmission unit	Maximum packet size of the Bluetooth® connection.
Payload		The intended message in a frame / package.
RF	Radio frequency	Describes wireless transmission.
RSSI	Receive Signal Strength Indicator	The RSSI indicates the strength of the RF signal. Its value is always printed in two's complement notation.
Soft device		Operating system used by the nRF52 chip.
SPI	Serial peripheral interface	Allows the serial communication with the module.
UART	Universal Asynchronous Receiver Transmitter	Allows the serial communication with the module.
[HEX] 0xhh	Hexadecimal	All numbers beginning with 0x are hexadecimal numbers. All other numbers are decimal, unless stated otherwise.

Contents

1 Introduction 5

2 Prerequisites 5

3 Bluetooth profiles 5

4 AMBER SPP-like profile 6
4.1 Generic Access Protocol (GAP) . . . 6
4.2 Generic Attribute Profile (GATT) . . . 7
4.2.1 Data length extension . . . 7
4.2.2 Company identifier . . . 7
4.2.3 UUID . . . 7
4.2.4 Primary Service . . . 7
4.2.4.1 Characteristics . . . 7
4.3 Bluetooth LE packet content . . . 8
4.3.1 RF-Packet format . . . 8
4.3.2 Advertising packet content . . . 9
4.3.3 Scan response packet content . . . 9

5 App development 10
5.1 Connection setup message charts . . . 10
5.1.1 No security and authentication . . . 10
5.1.2 Just works pairing . . . 10
5.1.3 Static pass key pairing . . . 11
5.1.4 Lesc passkey pairing . . . 12
5.1.5 Lesc numeric comparison pairing . . . 13
5.2 Enable notifications . . . 15
5.3 Remote GPIO control . . . 16
5.3.1 CMD_GPIO_REMOTE_WRITECONFIG_REQ . . . 16
5.3.1.1 Example: Configure two GPIOs of the connected remote device to output high . . . 17
5.3.2 CMD_GPIO_REMOTE_READCONFIG_REQ . . . 19
5.3.2.1 Example: Read the current GPIO configuration of the connected remote device . . . 20
5.3.3 CMD_GPIO_REMOTE_WRITE_REQ . . . 21
5.3.3.1 Example: Set a remote output GPIO to low . . . 22
5.3.4 CMD_GPIO_REMOTE_READ_REQ . . . 23
5.3.4.1 Example: Read the values of remote GPIOs . . . 24
5.3.5 CMD_GPIO_LOCAL_WRITE_IND . . . 24
5.3.5.1 Example: GPIOs of the remote device have been written by its local host . . . 24
5.4 Bonding development hints . . . 25
5.5 Nordic Bluetooth LE UART example app as base . . . 25

6 Custom firmware development 27
6.1 Custom firmware services of Würth Elektronik eiSos . . . 27
6.2 Important information for custom firmware development . . . 28
6.2.1 How to adapt Nordic Semiconductor SDK examples to run on the Proteus-III hardware? . . . 32
6.2.2 Firmware development hints . . . 34

7 Important notes **36**
7.1 General customer responsibility . . . 36
7.2 Customer responsibility related to specific, in particular safety-relevant applications . . . 36
7.3 Best care and attention . . . 36
7.4 Customer support for product specifications . . . 36
7.5 Product improvements . . . 37
7.6 Product life cycle . . . 37
7.7 Property rights . . . 37
7.8 General terms and conditions . . . 37

8 Legal notice **38**
8.1 Exclusion of liability . . . 38
8.2 Suitability in customer applications . . . 38
8.3 Trademarks . . . 38
8.4 Usage restriction . . . 38

9 License terms **40**
9.1 Limited license . . . 40
9.2 Usage and obligations . . . 40
9.3 Ownership . . . 41
9.4 Firmware update(s) . . . 41
9.5 Disclaimer of warranty . . . 41
9.6 Limitation of liability . . . 41
9.7 Applicable law and jurisdiction . . . 42
9.8 Severability clause . . . 42
9.9 Miscellaneous . . . 42

1 Introduction

This document provides all the information necessary to integrate the Proteus-III Bluetooth® LE module into user application. The standard features available with the default firmware are described in detail. Further, key parameters of the Bluetooth® LE specifications necessary to ensure interoperability with Bluetooth® compliant third party devices are listed and described in detail.
Besides of this, valuable hints to start an app development as well as to start a custom firmware development on base of the Proteus-III hardware are given in the subsequent chapters.

2 Prerequisites

A basic understanding of the Bluetooth® LE standard as well as application development background on the desired platform is necessary to fully understand this document.
The manual of Proteus-III contains basic information of the standard firmware and the software interfaces provided by the application. Please read this manual carefully and completely before using its information.
Würth Elektronik eiSos does not provide general support towards the Bluetooth® standard or smart device app development (independent of the platform).

3 Bluetooth profiles

Bluetooth® specification uses so called "Profiles" to specify the general behavior of a Bluetooth® enabled device to communicate with other Bluetooth® devices. Profiles are built on the Bluetooth® standard to clearly define what kind of data is transmitted. The device's application determines which profiles it must support, from hands-free capabilities to heart rate sensors to alerts and more.
A device may support more than one profile. For two devices to be compatible, they must support the same Bluetooth® LE profile.
The Proteus-III module ships with the so called AMBER SPP-like (Serial Port Profile) profile created based on the Generic Attribute profile (GATT). This profile aims at providing a Bluetooth® LE based wireless replacement to a serial cable connection.

4 AMBER SPP-like profile

This section contains the key data of the AMBER SPP-like profile. Each device in the network must support this profile to communicate with a Proteus-III device with the default SPP-like firmware. Customer applications may support and/or provide other profiles, services or interfaces.

4.1 Generic Access Protocol (GAP)

The main purpose of this protocol is to describe the parameters of lower layers of the Bluetooth® stack including discovery, scanning and security capabilities. The Proteus-III GAP specifications are listed below:

- Appearance as specified in the user setting `RF_Appearance`.
- Device name as specified in the user setting `RF_DeviceName`.
- Device address (6 Byte MAC) of type "public", see user setting `FS_BTMAC` (0x0018DAxxxxxx)
- Timings:
  - See user settings `RF_ScanTiming` and `RF_ScanFlags` for scan and advertising related timing parameters like
    - Advertising interval
    - Scan window
    - Scan interval
    - Connection setup timeout
  - See user setting `RF_ConnectionTiming` for connection related timing parameters like
    - Minimum connection interval
    - Maximum connection interval
    - Connection supervision timeout
  - See user setting `RF_TXPower` for TX power value.
  - See user setting `RF_SecFlags` for security settings.
  - Slave latency: 0
  - Peripheral requests for connection parameters update if central has differing connection parameters
    - Connection parameters update (initial): 5s
    - Connection parameters update (periodic): 10s
    - Connection parameters update counter before connection shut down: 3

4.2 Generic Attribute Profile (GATT)

4.2.1 Data length extension

The Proteus-III supports up to 243 Byte of payload data. To use this feature the data length extension has to be requested by the central device. In this case, the GATT MTU size must be 243 Byte payload + 1 Byte AMBER header + 3 Byte NUS header, which is 247 Byte in total.
The PDU size should be 243 Byte payload + 1 Byte AMBER header + 3 Byte NUS header + 4 Byte Bluetooth® LE header, which is 251 Byte in total.

Check also the message charts in chapter 5 to see the MTU request in the connection setup process.

4.2.2 Company identifier

The Bluetooth® listed company identifier of Würth Elektronik eiSos (formerly Amber wireless GmbH) is 0x031A (794_{dec}).

4.2.3 UUID

The Proteus-III uses a 128Bit UUID of type "Vendor specific". The base UUID is adapted by the 16Bit UUIDs of the primary service and the corresponding characteristics.
These UUIDs are only allowed to be used when one of the two corresponding devices is a Proteus-III module or contains a Proteus-III module of Würth Elektronik eiSos which have pre-installed firmware.

Service	16Bit UUID	Full UUID
Proteus-III base		6E400000-C352-11E5-953D-0002A5D5C51B
Proteus-III primary service	0x0001	6E400001-C352-11E5-953D-0002A5D5C51B
`RX_CHARACTERISTIC`	0x0002	6E400002-C352-11E5-953D-0002A5D5C51B
`TX_CHARACTERISTIC`	0x0003	6E400003-C352-11E5-953D-0002A5D5C51B

By means of the user setting `RF_SPPBaseUUID` the base UUID can be adapted to generate a custom profile.

4.2.4 Primary Service

4.2.4.1 Characteristics

- The first characteristic of the Proteus-III primary service is `RX_CHARACTERISTIC`:
  - The data is sent from central/client to peripheral/server using a write command.

  - Server:
    - Has to allow a write command as well as a write without response command.
  - Client:
    - Use write command to send data to the server.

- The second characteristic of the Proteus-III primary service is `TX_CHARACTERISTIC`:
  - The data is sent from peripheral/server to central/client using a notification.
  - Server:
    - Has to allow/enable notifications. Notify client/central when sending data.
    - When the notification enable bit is written in the `CCCD` (Client Characteristic Configuration Descriptor) by the central, the peripheral prints a `CMD_CHANNELOPEN_RSP` on the UART to signalize that the peripheral can send data to the central now. The central can only write this bit, when the configured security level of the peripheral has been met.
  - Client:
    - Has to enable notifications.

The permissions to access the characteristics is determined by the security mode of the module.

Proteus-III security mode	CCCD read	CCCD write, RX attribute read/write, TX attribute read/write
No security	no protection, open link	no protection, open link
Just works	no protection, open link	require encryption, but no MITM protection (Mode 1, Level 2)
Static pass key	no protection, open link	require encryption and MITM protection (Mode 1, Level 3)
Lesc Pass key	no protection, open link	require encryption, MITM protection, lesc (Mode 1, Level 4)
Lesc numeric comparison	no protection, open link	require encryption, MITM protection, lesc (Mode 1, Level 4)

4.3 Bluetooth LE packet content

4.3.1 RF-Packet format

To identify the type of data transmitted via Bluetooth® LE, the data protocol on the radio contains a 1 Byte packet header. Thus, the standard Bluetooth® LE payload has to match the following format to be understood by the Proteus-III:

Bluetooth® LE Payload	
AMBER Header	Payload
1 Byte	Φ_{ST} Bytes

Table 1: RF-packet format

The maximum payload size Φ_{ST} is 243 Bytes.
The AMBER Header has to be one of the following types:

0x01: `RF_HEADER_TYPE_DATA`: The following bytes contain the user payload data.

0x02: `RF_HEADER_TYPE_CMD`: The following bytes contain command data. See chapter `5.3` for more details.

Other: Other headers are reserved for future use and packets with this header are currently discarded.

4.3.2 Advertising packet content

The standard Proteus-III advertising packet contains the following data:

- Advertising data flags
- The UUID (128 Bit Proteus-III primary service UUID) of the AMBER SPP-like profile
- TXPower level (1 Byte in two's complement notation, only in command mode)
- Proteus-III device name as Shortened Local Name (up to 5 Bytes in command mode, up to 8 Bytes in peripheral only mode)

4.3.3 Scan response packet content

The scan response packet is requested during scan if active scanning is enabled. The standard Proteus-III scan response packet contains the following data:

- Manufacturer data (up to 20 Bytes) in RF-packet format (see Table `1`) using the company identifier. This manufacturer data is used to realize the Beacon feature.

5 App development

The definition of the AMBER SPP-like profile (see section 4) in combination with the message charts of chapter 5 are sufficient to develop custom apps for mobile devices. To implement this profile from scratch fundamental knowledge of app development as well as of the Bluetooth® LE standard is required.

5.1 Connection setup message charts

The following message charts show which steps are run during the connection setup process between two Proteus-III modules. To implement the central role in an app to connect to the Proteus-III peripheral the steps of the central device shown below have to be reproduced.
More detailed information can be found in the message chart chapter of Nordic Semiconductor's documentation of the Softdevice S140 V7.0.1.

5.1.1 No security and authentication

If the Proteus-III peripheral does not use any security settings, we just have to connect to it. After connecting a MTU request is necessary to allow a higher payload size. After the discovery of the characteristics, the notification of the RX characteristic has to be enabled.

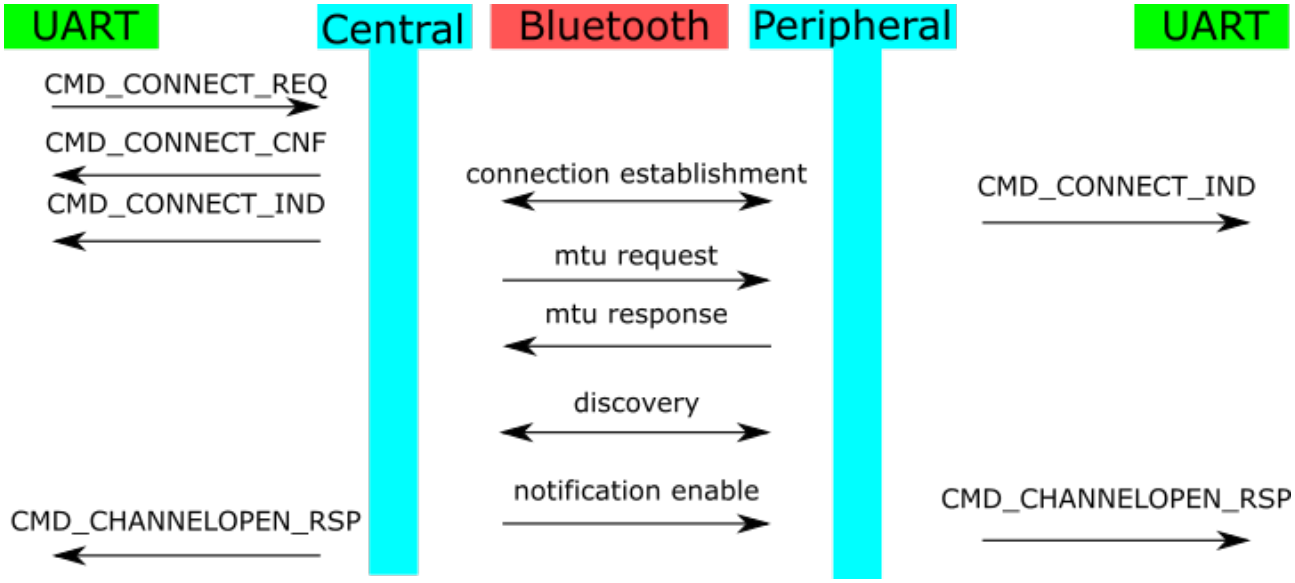

Figure 1: No security enabled

5.1.2 Just works pairing

If the Proteus-III peripheral needs the just works pairing security level, we just have to place a just works pairing request (no in/out capabilities, no mitm) after the connection step was run. Here a MTU request is necessary again to allow a higher payload size. After the discovery of the characteristics, the notification of the RX characteristic has to be enabled.

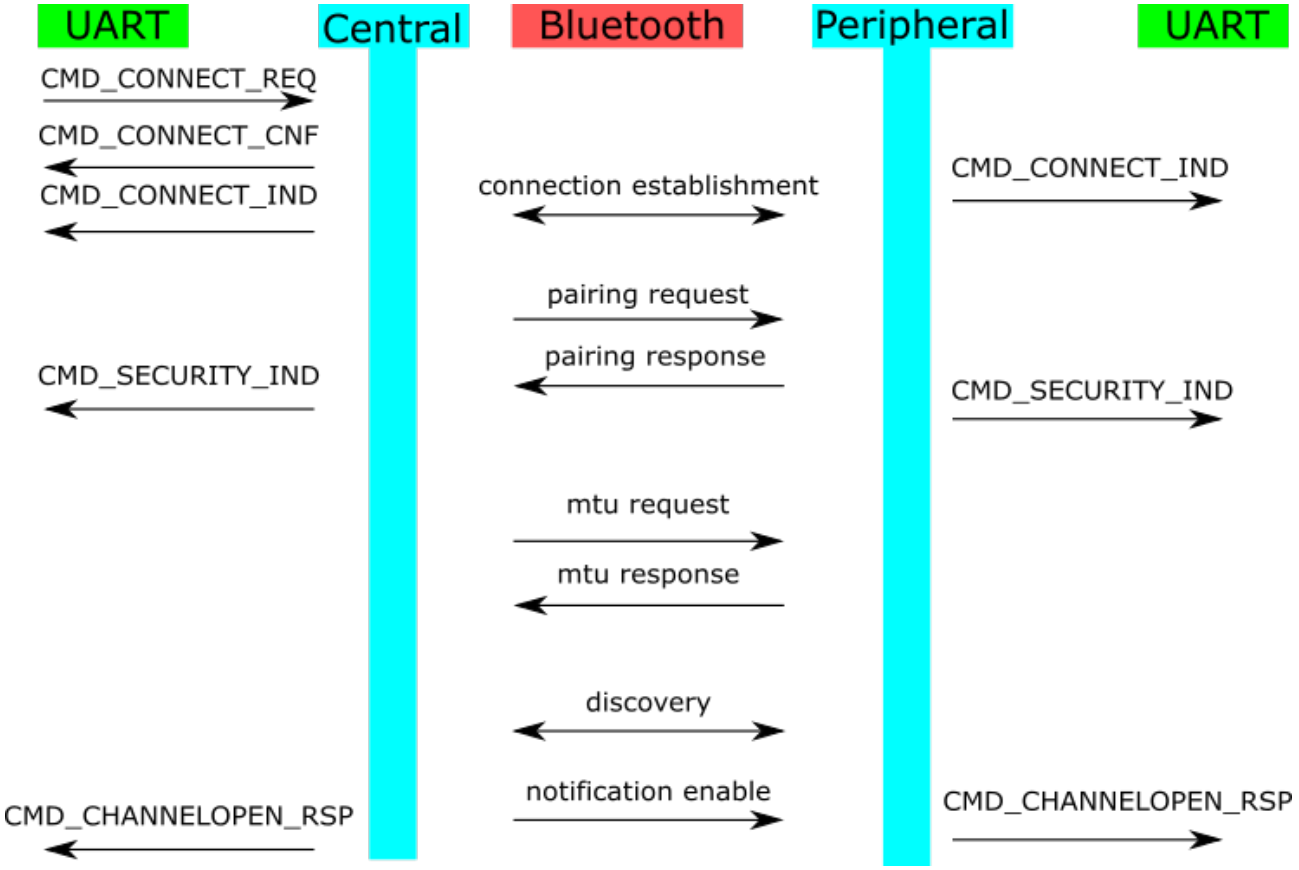

Figure 2: Just works pairing enabled

5.1.3 Static pass key pairing

If the Proteus-III peripheral needs the static pass key pairing security level, we just have to place a pairing request (keyboard only, mitm) after the connection step was run. The Proteus-III sends a pass key request, such that the static pass key of the Proteus-III peripheral has to be entered on the central side (app).
Afterwards a MTU request is necessary again to allow a higher payload size. After the discovery of the characteristics, the notification of the RX characteristic has to be enabled.

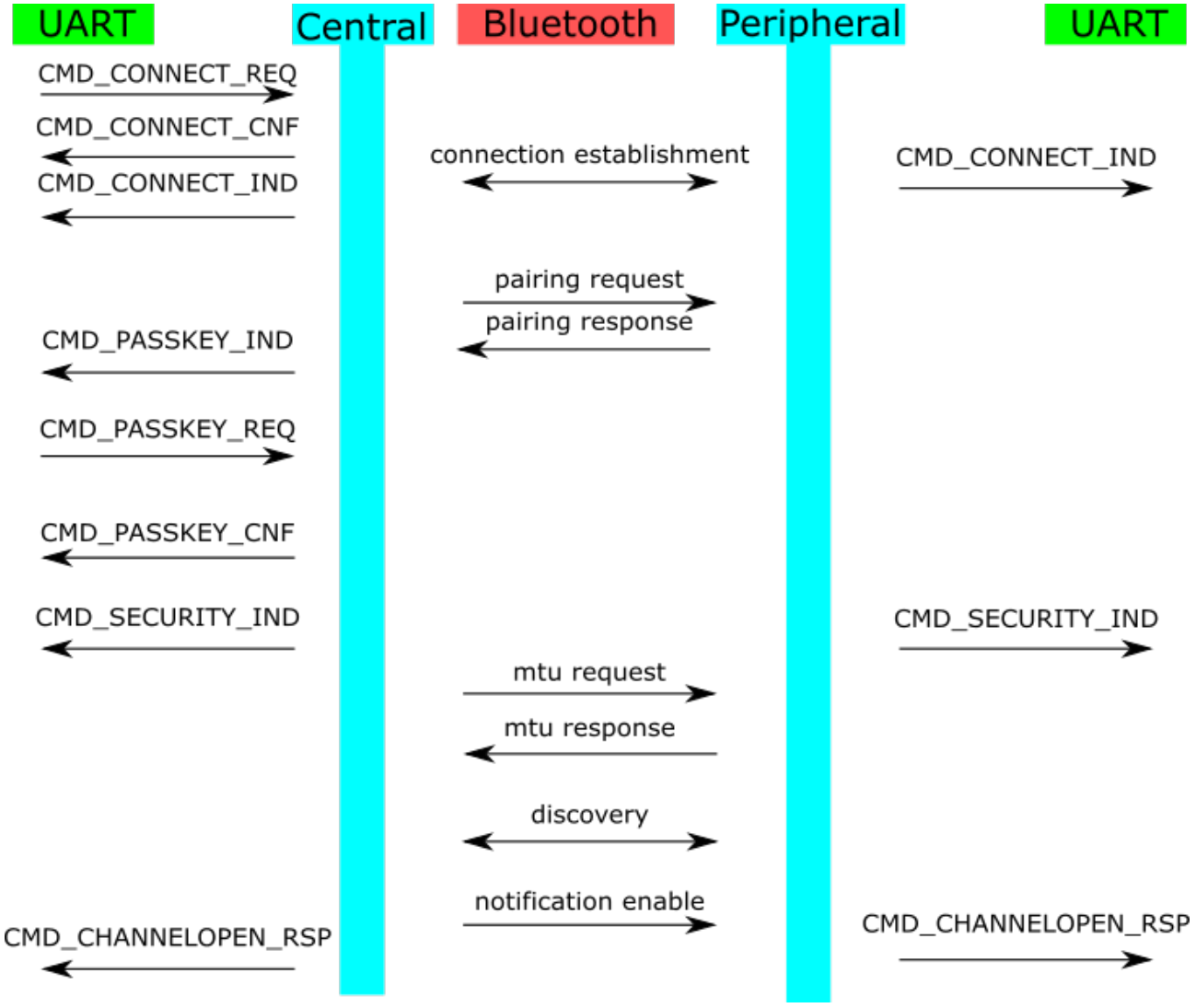

Figure 3: Static pass key pairing enabled

5.1.4 Lesc passkey pairing

If the Proteus-III peripheral needs the lesc pass key pairing security level, we just have to place a pairing request (keyboard only, mitm, lesc) after the connection step was run. The Proteus-III sends a lesc pass key request, such that the lesc pass key of the Proteus-III peripheral has to be entered on the central side (app). This key is not fix, but generated on each connection setup and output on the peripheral side by a `CMD_DISPLAYPASSKEY_IND` message.
Afterwards a MTU request is necessary again to allow a higher payload size. After the discovery of the characteristics, the notification of the RX characteristic has to be enabled.

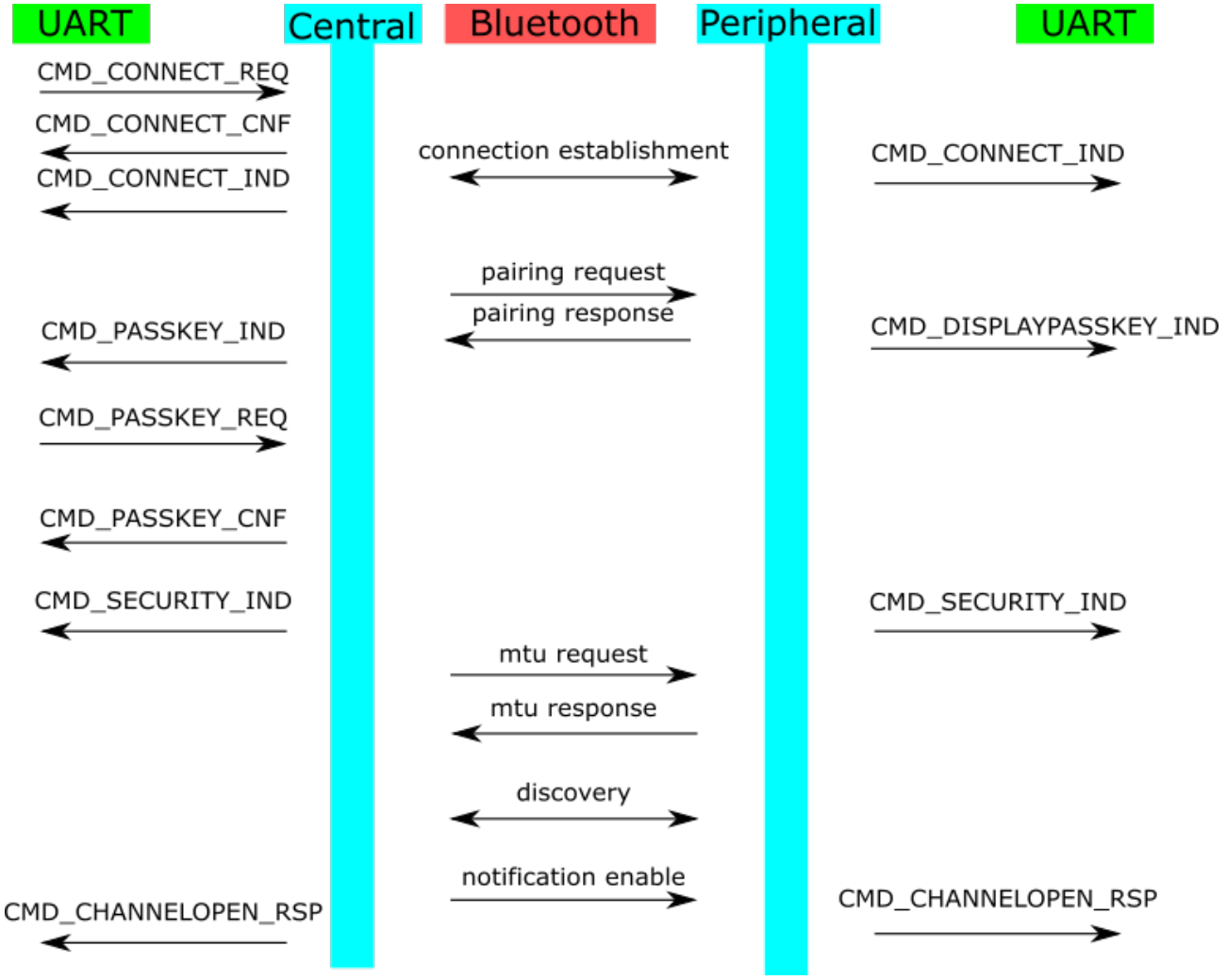

Figure 4: Lesc pass key pairing enabled

5.1.5 Lesc numeric comparison pairing

If the Proteus-III peripheral needs the lesc numeric comparison pairing security level, we just have to place a pairing request (display yes/no, mitm, lesc) after the connection step was run. The Proteus-III sends a lesc numeric comparison request, such that the lesc pass key is output on central and peripheral side. Both, the central and peripheral need to confirm that the displayed key on the central and peripheral device coincide.
Afterwards a MTU request is necessary again to allow a higher payload size. After the discovery of the characteristics, the notification of the RX characteristic has to be enabled.

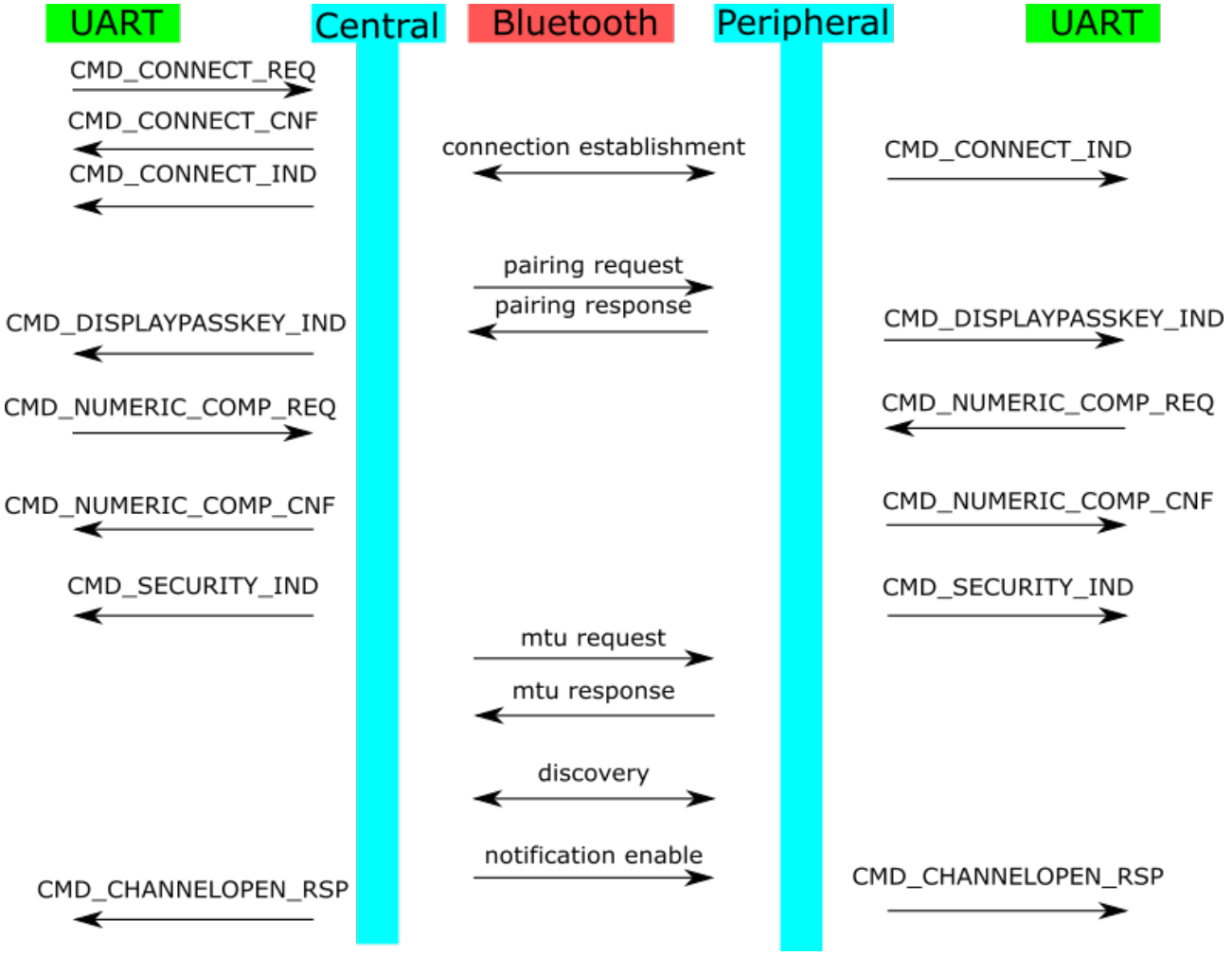

Figure 5: Lesc numeric comparison pairing enabled

.

5.2 Enable notifications

As described in the previous chapter 5.1 the final step for a successful connection set-up is the enabling of the notification of the TX_CHARACTERISTIC. To do so, the Android's Bluetooth® LE stack offers the following function, that has to be called with the TX_CHARACTERISTIC.

```
IBluetoothGatt mService;

 /**    TX NOTIFICATION
* Enable or disable  notifications / indications  for  a given  characteristic .
*
* <p>Once notifications are enabled for  a  characteristic ,  a
* {@link BluetoothGattCallback#onCharacteristicChanged} callback will be
* triggered  if  the  remote device indicates  that  the  given  characteristic
* has changed.
*
* <p>Requires {@link android.Manifest.permission#BLUETOOTH} permission.
*
* @param characteristic The characteristic  for  which to  enable  notifications
* @param enable Set to true to enable notifications / indications
* @return true,  if  the  requested  notification  status  was set successfully
*/
public boolean setCharacteristicNotification(BluetoothGattCharacteristic  characteristic ,
boolean enable) {
 if  (DBG) {
Log.d(TAG, " setCharacteristicNotification ()␣-␣uuid:␣" + characteristic.getUuid()
+ "␣enable:␣" + enable);
}
 if  (mService == null || mClientIf  == 0) return false;

BluetoothGattService service = characteristic .getService();
 if  (service  == null)  return false;

BluetoothDevice device = service.getDevice();
 if  (device == null)  return false;

try  {
mService.registerForNotification(mClientIf,  device.getAddress(),
 characteristic .getInstanceId(),  enable);
}  catch (RemoteException e) {
Log.e(TAG, "",  e);
return false;
}

return true;
}
```

Code 1: Example code to enable the TX characteristic notification

Please note that the iOS's Bluetooth® LE stack calls the corresponding function automatically. Thus calling a notification enable function from the app's application layer is not needed.

5.3 Remote GPIO control

The Proteus-III contains the feature to control its free GPIOs via remote access. To do so, first of all the local host must configure the GPIOs of interest. Then the GPIOs can be written or read by a remote device.
To use this feature via APP, the respective radio commands must be send via Bluetooth® LE. To do so, the data to be send has the following structure:

Bluetooth® LE Payload	
AMBER Header	Command Payload
0x02	x Bytes

Table 2: RF-packet command format

5.3.1 CMD_GPIO_REMOTE_WRITECONFIG_REQ

This command can be used to configure the free GPIOs of the remote device. Remote configuration can be blocked by writing the corresponding bit of the user setting `CFG_Flags`.
Format:

AMBER Header	Command	$Block_1$	...	$Block_n$
0x02	0x28	x Bytes		x Bytes

Response (`CMD_GPIO_REMOTE_WRITECONFIG_CNF`):

AMBER Header	Command | 0x40	$Block_1$	...	$Block_n$
0x02	0x68	x Bytes		x Bytes

CMD_GPIO_REMOTE_WRITECONFIG_REQ block structure

Each **Block** has the following format:

Length	GPIO_ID	Function	Value
1 Byte	1 Byte	1 Byte	(Length-2) Bytes

Length: Length of the subsequent bytes in this block

- if **Function** is disconnected, input or output: 3
- if **Function** is PWM: 5

GPIO_ID: ID of the GPIO, see Proteus-III manual

Function:

0x00: GPIO disconnected

0x01: GPIO works as input

0x02: GPIO works as output

0x03: GPIO works as PWM

Value:

- if **Function** is disconnected:
  Values field is not used in this Block
- if **Function** is input:
  0x00: GPIO has no pull resistor
  0x01: GPIO has pull down resistor
  0x02: GPIO has pull up resistor
- if **Function** is output:
  0x00: GPIO is output low
  0x01: GPIO is output high
- if **Function** is PWM:
  Byte 0 and 1: LSB first uint16 PWM period in ms (1 - 500 ms)
  Byte 2: uint8 ratio (0x00 = 0%,..., 0x7F = 50%,..., 0xFF=100%)

CMD_GPIO_REMOTE_WRITECONFIG_CNF block structure

Each **Block** has the following format:

Length	GPIO_ID	Status
0x02	1 Byte	1 Byte

Length: Length of the subsequent bytes in this block

GPIO_ID: ID of the GPIO, see Proteus-III manual

Status:

0x00: Success

0x01: Failed

0xFF: Remote configuration not allowed (blocked by the user setting `CFG_Flags` of the remote device)

5.3.1.1 Example: Configure two GPIOs of the connected remote device to output high

Configure the GPIOs with ID **0x01** and **0x02** to output high:

AMBER Header	Command	$Block_1$	$Block_2$
0x02	0x28	0x03 **0x01** 0x02 0x01	0x03 **0x02** 0x02 0x01

Response:

AMBER Header	Command | 0x40	$Block_1$	$Block_2$
0x02	0x68	0x02 **0x01** 0x00	0x02 **0x02** 0x00

Configured both GPIOs with success.

5.3.2 CMD_GPIO_REMOTE_READCONFIG_REQ

This command can be used to read the configuration of the free GPIOs of the remote device.
Format:

AMBER Header	Command
0x02	0x2C

Response (`CMD_GPIO_REMOTE_READCONFIG_CNF`):

AMBER Header	Command | 0x40	$Block_1$	...	$Block_n$
0x02	0x6C	x Bytes		x Bytes

CMD_GPIO_REMOTE_READCONFIG_CNF block structure

Each **Block** has the following format:

Length	GPIO_ID	Function	Value
1 Byte	1 Byte	1 Byte	(Length-2) Bytes

Length: Length of the subsequent bytes in this block

- if **Function** is disconnected, input or output: 3
- if **Function** is PWM: 5

GPIO_ID: ID of the GPIO, see Proteus-III manual

Function:

0x00: GPIO is not configured yet (Length is 0x02 and **Value** is empty)
0x01: GPIO works as input
0x02: GPIO works as output
0x03: GPIO works as PWM

Value:

- if **Function** is disconnected:
  Values field is not used in this Block
- if **Function** is input:
  0x00: GPIO has no pull resistor
  0x01: GPIO has pull down resistor
  0x02: GPIO has pull up resistor
- if **Function** is output:
  0x00: GPIO is output low
  0x01: GPIO is output high

- if **Function** is PWM:

  Byte 0 and 1: LSB first uint16 PWM period in ms (1 - 500 ms)

  Byte 2: uint8 ratio (0x00 = 0%,..., 0x7F = 50%,..., 0xFF=100%)

5.3.2.1 Example: Read the current GPIO configuration of the connected remote device

Read the current GPIO configuration of the connected remote device:

AMBER Header	Command
0x02	0x2C

Response:

AMBER Header	Command | 0x40	Blocks
0x02	0x6C	0x03 **0x01** 0x02 0x01 0x03 **0x02** 0x02 0x01 0x02 **0x03** 0x00 0x02 **0x04** 0x00 0x02 **0x05** 0x00 0x02 **0x06** 0x00

The GPIOs with GPIO_ID **0x01** and **0x02** are output high. The remaining GPIOs with GPIO_ID **0x03**,**0x04**,**0x05** and **0x06** are not configured.

5.3.3 CMD_GPIO_REMOTE_WRITE_REQ

This command can be used to write the free GPIOs of the remote device. This command can be only run successfully if the respective pins of the remote device are configured as output pins.

Perform a `CMD_GPIO_REMOTE_READCONFIG_REQ` before using the `CMD_GPIO_REMOTE_WRITE_REQ` command to ensure the pins are setup correctly, as the remote device will not be notified if the GPIO configuration changes.

Format:

AMBER Header	Command	$Block_1$	...	$Block_n$
0x02	0x29	x Bytes		x Bytes

Response (`CMD_GPIO_REMOTE_WRITE_CNF`):

AMBER Header	Command | 0x40	$Block_1$	...	$Block_n$
0x02	0x69	x Bytes		x Bytes

CMD_GPIO_REMOTE_WRITE_REQ block structure

Each **Block** has the following format:

Length	GPIO_ID	Value
0x02	1 Byte	1 Byte

Length: Length of the subsequent bytes in this block

GPIO_ID: ID of the GPIO, see Proteus-III manual

Value:

- if **Function** is output
  0x00: Set GPIO to LOW
  0x01: Set GPIO to HIGH
- if **Function** is PWM
  Byte 0: uint8 ratio (0x00 = 0%,..., 0x7F = 50%,..., 0xFF=100%)

CMD_GPIO_REMOTE_WRITE_CNF block structure

Each **Block** has the following format:

Length	GPIO_ID	Status
0x02	1 Byte	1 Byte

Length: Length of the subsequent bytes in this block

GPIO_ID: ID of the GPIO, see Proteus-III manual

Status:

0x00: Success

0x01: Failed

5.3.3.1 Example: Set a remote output GPIO to low

Set the output GPIO (GPIO_ID **0x01**) of the connected remote device to low:

AMBER Header	Command	$Block_1$
0x02	0x29	0x02 **0x01** 0x00

Response:

AMBER Header	Command | 0x40	$Block_1$
0x02	0x69	0x02 **0x01** 0x00

Successfully set GPIO with GPIO_ID **0x01** to low.

5.3.4 CMD_GPIO_REMOTE_READ_REQ

This command can be used to read the free GPIOs of the remote device. This command can be only run successfully if the respective pins of the remote device are configured.

Perform a `CMD_GPIO_REMOTE_READCONFIG_REQ` before using the `CMD_GPIO_REMOTE_READ_REQ` command to ensure the pins are setup correctly, as the remote device will not be notified if the GPIO configuration changes.

Format:

Amber Header	Command	$Block_1$	...	$Block_n$
0x02	0x2A	x Bytes		x Bytes

Response (`CMD_GPIO_REMOTE_READ_CNF`)

Amber Header	Command | 0x40	$Block_1$	...	$Block_n$
0x02	0x6A	x Bytes		x Bytes

CMD_GPIO_REMOTE_READ_REQ block structure

Each **Block** has the following format:

Length	$GPIO\_ID_1$	...	$GPIO\_ID_n$
1 Bytes	1 Byte		1 Byte

Length: Length of the subsequent bytes in this block

GPIO_ID: ID of the GPIO, see Proteus-III manual

CMD_GPIO_REMOTE_READ_CNF block structure

Each **Block** has the following format:

Length	GPIO_ID	Value
0x02	1 Byte	1 Byte

Length: Length of the subsequent bytes in this block

GPIO_ID: ID of the GPIO, see Proteus-III manual

Value:

- if **Function** is output

  0x00: The GPIO is LOW

  0x01: The GPIO is HIGH

0xFF: Failed reading remote GPIO value.

- if **Function** is PWM

  Byte 0: uint8 ratio (0x00 = 0%,..., 0x7F = 50%,..., 0xFF=100%)

5.3.4.1 Example: Read the values of remote GPIOs

Read the value of the GPIOs with GPIO_ID **0x01** and **0x02** of the connected remote device:

Amber Header	Command	$Block_1$
0x02	0x2A	0x02 **0x01 0x02**

Response:

Amber Header	Command | 0x40	$Block_1$	$Block_2$
0x02	0x6A	0x02 **0x01** 0x00	0x02 **0x02** 0x01

Successfully read the values of the remote GPIOs with GPIO_ID **0x01** (GPIO is low) and **0x02** (GPIO is high).

5.3.5 CMD_GPIO_LOCAL_WRITE_IND

This message informs the connected remote device, that the radio module's local host has written the GPIOs.

Please note that only the GPIOs are part of this message, that have been updated successfully. Failed attempts of GPIO updates will not be indicated by this message.

Format:

Amber Header	Command	$Block_1$	...	$Block_n$
0x02	0xA8	x Bytes		x Bytes

Each **Block** has the format of `CMD_GPIO_REMOTE_READ_CNF block structure`.

5.3.5.1 Example: GPIOs of the remote device have been written by its local host

Amber Header	Command	$Block_1$	$Block_2$
0x02	0xA6	0x02 **0x01** 0x00	0x02 **0x02** 0x01

The GPIOs with GPIO_ID **0x01** (GPIO is low) and **0x02** (GPIO is high) of the remote device have been written by its local host.

5.4 Bonding development hints

The firmware of the Proteus-III provides the bonding feature that allows to re-pair without repeating the authentication step (e.g. entering the static passkey). Thus, in the initial connection all bonding data is stored in the devices' flash to be used during the setup of subsequent connections.
The function `CMD_DELETEBONDS_REQ` of the Proteus-III allows to remove not needed bonding data from the module's flash. Thus in case of missing bonding data on one of the two connection partners, a re-bonding has to be initiated by the central device! Otherwise, the security level is not met to send the "notification enable" and thus the channel for data transmission cannot be opened.

Please note that iOS devices do not run the re-bonding step by default, if bonding data is missing on one of the two connection partners.
In certain cases, the bonding data on the iOS device has to be cleared first, such that iOS starts the re-bonding step.

5.5 Nordic Bluetooth LE UART example app as base

Nordic Semiconductor provides source code to develop Android, iOS and Windows applications. To implement the AMBER SPP-like profile for your own app, these source codes can be taken as a base for your own app development.

Please note that this app does not implement any authentication and security features. Thus, the Proteus-III to connect to must have no security enabled when using this provided example.
Furthermore, the request for data length extension is not part of the provided source code.

The following few changes have to be applied to the Nordic UART-APP-example to implement the SPP-like profile:

- Replace the implemented UUIDs by the SPP-like profile UUID.
  Android example:

```
private final static UUID UART_SERVICE_UUID = UUID.fromString("6E400001-C352-11E5
    -953D-0002A5D5C51B");
private final static UUID UART_RX_CHARACTERISTIC_UUID = UUID.fromString("6E400002-
    C352-11E5-953D-0002A5D5C51B");
private final static UUID UART_TX_CHARACTERISTIC_UUID = UUID.fromString("6E400003-
    C352-11E5-953D-0002A5D5C51B");
```

Code 2: Update UUID

- When sending data, add the packet header in front of the payload.
  Android example:

```
public void send(final String text) {
// Are we connected?
if (mRXCharacteristic == null)
return;

if (! TextUtils .isEmpty(text) && mOutgoingBuffer == null) {
final char AMBER_RF_HEADER_TYPE_DATA = 0x01;
```

```
final byte[] buffer = mOutgoingBuffer = (AMBER_RF_HEADER_TYPE_DATA + text).getBytes();
mBufferOffset = 0;
 ...
}
```

Code 3: Add packet header on sender side

```
public void onDataSent(final String data) {
 if (AMBER_RF_HEADER_TYPE_DATA == data.charAt(0)) {
Logger.a(getLogSession(), "Valid␣data␣sent:\"" + data.substring(1) + "\"");
}
else {
Logger.w(getLogSession(), "Invalid␣data␣sent:\"" + data + "\"");
}
 ...
}
```

Code 4: Check packet header in sender callback

- When receiving data, first interpret the header to detect the data type before any other action (data output or command execution) is performed.
  Android Example:

```
public void onDataReceived(final String data) {
 if (AMBER_RF_HEADER_TYPE_DATA == data.charAt(0)) {
Logger.a(getLogSession(), "Valid␣data␣received:\"" + data.substring(1) + "\"");
}
else {
Logger.w(getLogSession(), "Invalid␣data␣received:\"" + data + "\"");
}
 ...
}
```

Code 5: Remove packet header on receiver side

6 Custom firmware development

Using the Proteus-III hardware a custom firmware can be developed to better fit the customer's needs. Based on the Nordic Semiconductor SDK and demo examples various Bluetooth® LE profiles and custom applications can be realized and flashed on the Proteus-III module. The versatile and well documented Nordic stack ensures quick and easy realization of various standard Bluetooth® LE profiles. Chapter 6.2 contains the information needed to run Nordic standard examples on the Proteus-III hardware.
On the other hand, Würth Elektronik eiSos provides firmware development services for customers that are not interested in writing their own firmware stack. Here, Würth Elektronik eiSos can quickly adapt the Proteus-III standard firmware to the customer's need or completely develop a new firmware from scratch (see chapter 6.1).

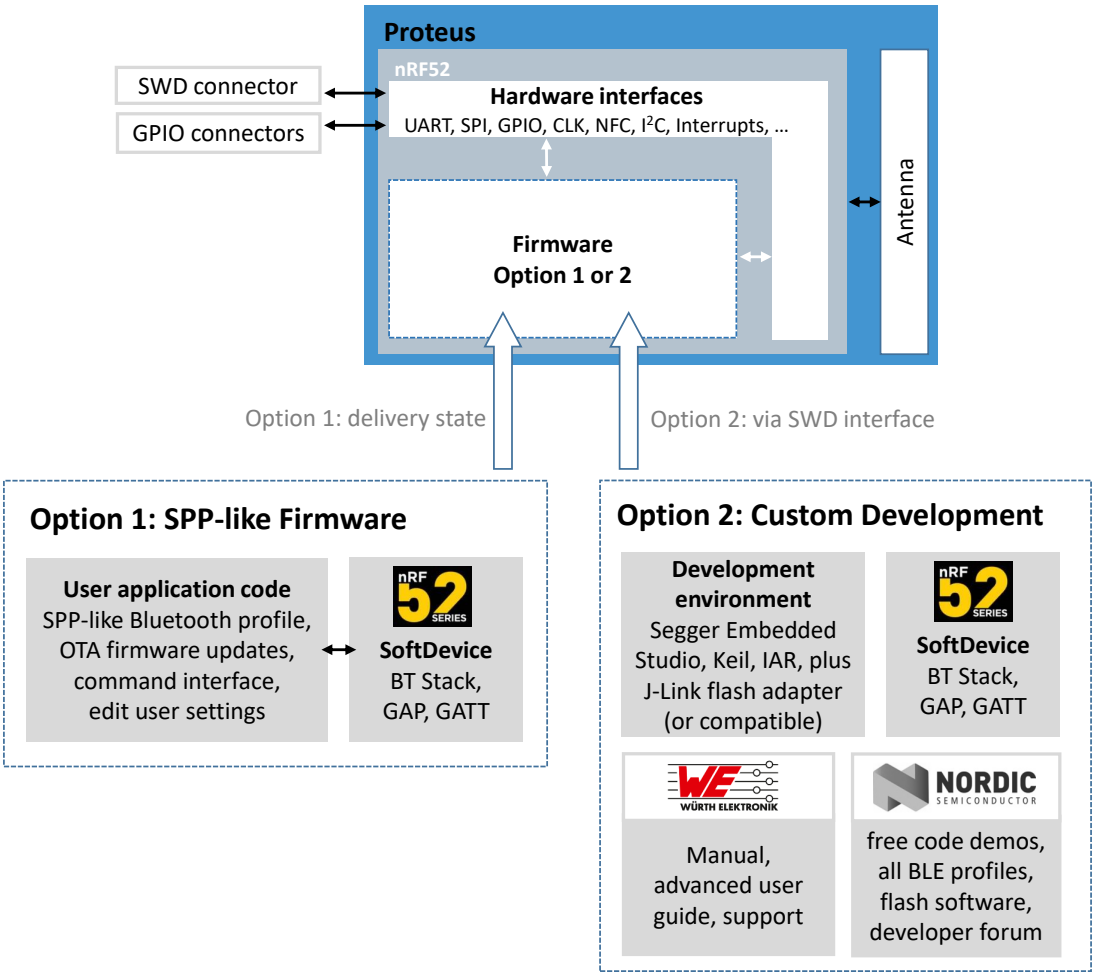

Figure 6: Options for running the Proteus-III with standard or custom firmware

6.1 Custom firmware services of Würth Elektronik eiSos

The Proteus-III firmware as described in the Proteus-III manual includes the Softdevice, a dual-bank bootloader and the application hosting the SPP-like protocol for RF communication. After flashing this firmware onto the chip, there are up to 150kB free memory for custom applications that can be included into the firmware on request.
If more memory is needed, the dual-bank bootloader can be replaced by a single-bank bootloader or even completely removed. In this case, more than 600kB of memory can be

reserved for custom application code. As an alternative external Flash/EEPROM IC(s) can be connected to the module (e.g. using SPI or I2C interface) on the customer PCB.

Source codes for the Proteus-III SPP-like firmware are property of Würth Elektronik eiSos and will not be provided to customers. Nonetheless, Würth Elektronik eiSos may consider different license models or exceptions for individual customers.

Besides of this, Würth Elektronik eiSos also provides custom firmware developments from scratch. Please contact your local field sales engineer (FSE) or wireless-sales@we-online.com to discuss further details.

6.2 Important information for custom firmware development

To start a custom firmware development on top of the Proteus-III hardware, the following information must be considered:

- Chip
  The Proteus-III contains the Nordic Semiconductor nRF52840-CKAA SoC. The CPU is a 64MHz ARM Cortex-M4F.

- Pinout
  The Proteus-III provides the following pins of the Nordic SoC with its pads. Only the *ANT*, *RF*, *GND*, *VDD*, *Reset*, *SWDCLK* and *SWDIO* pins are fixed. All other pins can be used for custom firmware development. For special functions like near field communication (NFC), external low frequency quartz crystal XL or analog input (AIN) the respective pins have to be used.

No.	Pad Name	No.	Pad Name
1	ANT	13	P1.08
2	RF	14	P1.09
3	GND	15	P0.11
4	SWDCLK	16	P0.12
5	SWDIO	17	P0.03/AIN1
6	P0.18/Reset	18	GND
7	P0.02/AIN0	B1	P0.09/NFC1
8	VDD	B2	P0.10/NFC2
9	P0.19	B3	P0.23
10	P0.22	B4	P1.00
11	P0.00/XL1	B5	P0.21
12	P0.01/XL2	B6	P0.07

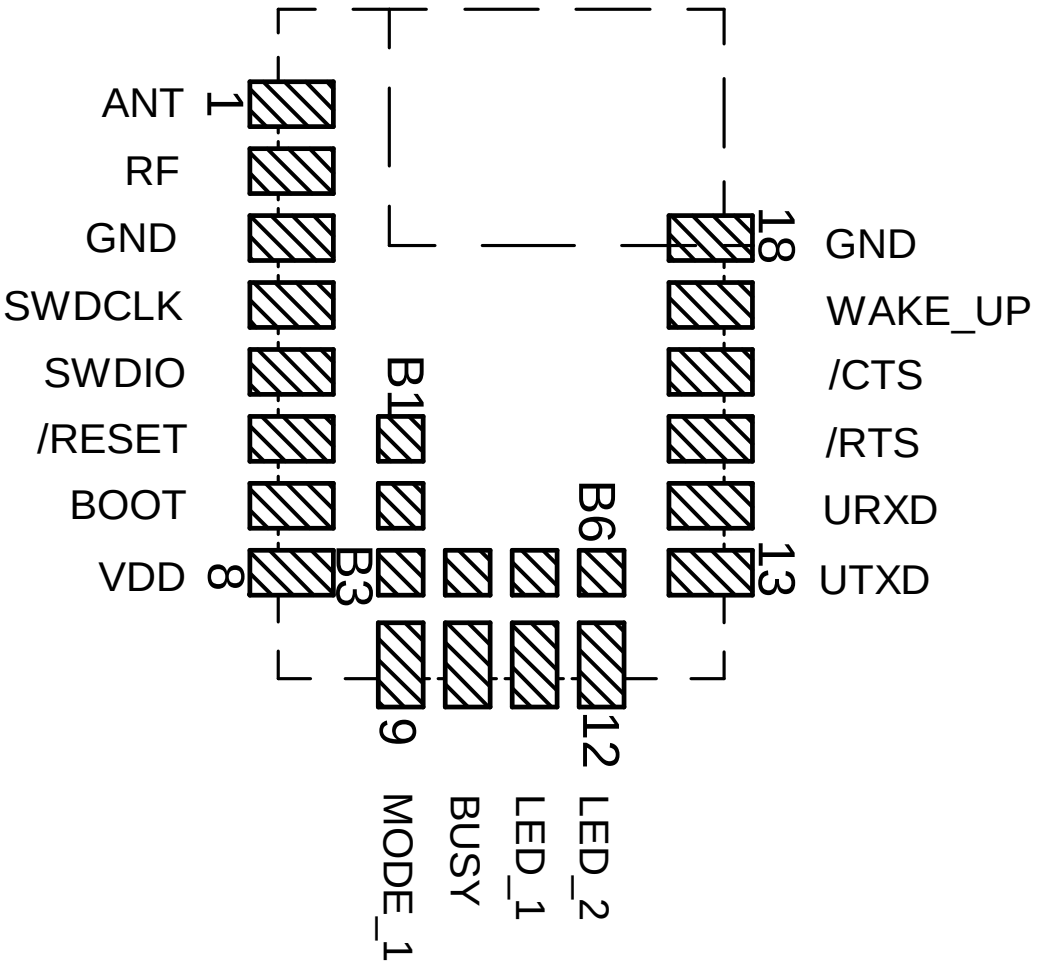

Figure 7: Pinout

- Hardware for development & debugging
  Using Segger J-Link flasher and the SWD interface is required for firmware development and debugging. Checkout the Proteus-III-EV board. It provides the easiest way to develop software based on Proteus-III module or apps for the SPP-like profile.

- Software development environment
  Nordic Semiconductor provides software packages for several compilers (KEIL, IAR, GCC, Segger Embedded).
  For example, the Keil SDK includes the required Bluetooth® LE stack ("Softdevice"), many demo examples for Bluetooth® LE profiles and services to conveniently develop a custom firmware on basis of the Nordic SoC. Further library's for hardware peripheral (such as ADC, I2C, SPI, UART etc.) are also include in the SDK and examples. More information and details about the chip and the operating system is bundled on the Nordic Semiconductor Infocenter:
  http://infocenter.nordicsemi.com/

  Please check the tab "nRF52 Series" to access the newest information about the nRF52 radio chip and the software environment.
  If available, use the examples for the Nordic evaluation platform (like PCA10040 or PCA10056) as a starting point. See also chapter 6.2.1 for more information how to run Nordic standard examples on top of the Proteus-III.

- Clock sources
  The Proteus-III module contains a dedicated RF clock (HFCLK). The Proteus-III does not contain a dedicated low frequency clock (LFCLK). Thus custom firmware must use the internal RC-oscillator as long as no external clock crystal is connected to the respective pins (*XL1*, *XL2*) on the customer PCB.
  Example for enabling the internal RC oscillator for SDK 15.3.0:

```
// <o> NRF_SDH_CLOCK_LF_SRC - SoftDevice clock source.
// <0=> NRF_CLOCK_LF_SRC_RC
// <1=> NRF_CLOCK_LF_SRC_XTAL
// <2=> NRF_CLOCK_LF_SRC_SYNTH
#ifndef NRF_SDH_CLOCK_LF_SRC
#define NRF_SDH_CLOCK_LF_SRC 0
#endif

// <o> NRF_SDH_CLOCK_LF_RC_CTIV - SoftDevice calibration timer interval.
#ifndef NRF_SDH_CLOCK_LF_RC_CTIV
#define NRF_SDH_CLOCK_LF_RC_CTIV 16
#endif

// <o> NRF_SDH_CLOCK_LF_RC_TEMP_CTIV - SoftDevice calibration timer interval under
    constant temperature.
// <i> How often (in number of calibration intervals) the RC oscillator shall be calibrated
// <i> if the temperature has not changed.
#ifndef NRF_SDH_CLOCK_LF_RC_TEMP_CTIV
#define NRF_SDH_CLOCK_LF_RC_TEMP_CTIV 2
#endif

// <o> NRF_SDH_CLOCK_LF_ACCURACY - External clock accuracy used in the LL to compute
    timing.
// <0=> NRF_CLOCK_LF_ACCURACY_250_PPM
// <1=> NRF_CLOCK_LF_ACCURACY_500_PPM
// <2=> NRF_CLOCK_LF_ACCURACY_150_PPM
// <3=> NRF_CLOCK_LF_ACCURACY_100_PPM
// <4=> NRF_CLOCK_LF_ACCURACY_75_PPM
// <5=> NRF_CLOCK_LF_ACCURACY_50_PPM
// <6=> NRF_CLOCK_LF_ACCURACY_30_PPM
// <7=> NRF_CLOCK_LF_ACCURACY_20_PPM
// <8=> NRF_CLOCK_LF_ACCURACY_10_PPM
// <9=> NRF_CLOCK_LF_ACCURACY_5_PPM
// <10=> NRF_CLOCK_LF_ACCURACY_2_PPM
// <11=> NRF_CLOCK_LF_ACCURACY_1_PPM
#ifndef NRF_SDH_CLOCK_LF_ACCURACY
#define NRF_SDH_CLOCK_LF_ACCURACY 1
#endif
```

Code 6: sdk_config.h

Code may differ when using different SDK version.

- Voltage regulator
  As internal voltage regulator, we recommend to use the DCDC instead of the LDO. The DCDC has to be switched on explicitly in application code. Example for SDK 15.3.0:

```
sd_power_dcdc_mode_set(NRF_POWER_DCDC_ENABLE);
```

Code may differ when using different SDK version.

Changing from LDO to DCDC reduces the current consumption of the module to meet lowest power specifications.

- Certification and Bluetooth®-Listing
  Custom firmware may require additional certification. Any (end-)device containing Bluetooth® IP must be listed by the Bluetooth® SIG which requires membership and qualification. Please contact the Bluetooth® SIG or your preferred Bluetooth® certification laboratory for question. Further information are available in the Proteus-III manual.

To make use of the existing certification and listing of the Proteus-III, it is mandatory to use the Bluetooth® stack Nordic Semiconductor S140 Version 7.0.1 .

- Serial number
  The unique serial number (used for tracing and the generation of the Proteus-III BT-MAC) is placed in the user information configuration register (UICR->Customer[0]) and will be removed by flashing a customer firmware onto the SoC.

6.2.1 How to adapt Nordic Semiconductor SDK examples to run on the Proteus-III hardware?

The following description is based on the SDK 15.3.0. Code may differ when using a different Softdevice and/or SDK version.

Please perform the following steps to run a Nordic standard example on the Proteus-III:

1. Open the example project of interest and compile.
2. In case of success[1], enable the DCDC by adding the following line at the end of the stack init function.

```
static void ble_stack_init(void){
.
.
.
// Enable DCDC
err_code = sd_power_dcdc_mode_set(NRF_POWER_DCDC_ENABLE);
APP_ERROR_CHECK(err_code);
}
```

3. If no external crystal has been connected to the radio module, enable the internal RC-oscillator as shown in code example 6.
4. Go to the file board.h and add the include for the Proteus-III.h board file.

```
#if defined(BOARD_PCA10040)
#include "pca10040.h"
#elif defined(BOARD_PROTEUSI)
#include "ProteusI.h"
#elif defined(BOARD_PROTEUSII)
#include "ProteusII.h"
#elif defined(BOARD_PROTEUSIII)
#include "ProteusIII.h"
#else
#error "Board␣is␣not␣defined"
#endif
```

[1] If you have a Nordic evaluation board available, please check that the original example without modifications runs successfully on the Nordic evaluation board.

5. Then create the Proteus-III board file. To do so, please copy the board file of the Nordic evaluation board (like PCA10040 or PCA10056) and add the pinout, led button numbering, button numbering and clock definition of the Proteus-III:

```
#ifndef PROTEUSIII_H
#define PROTEUSIII_H

#ifdef __cplusplus
extern "C" {
#endif

#include "nrf_gpio.h"


/* PINS of the nRF52840 */

#define NRF_PIN_LED_1 NRF_GPIO_PIN_MAP(0,0)
#define NRF_PIN_LED_2 NRF_GPIO_PIN_MAP(0,1)
#define NRF_PIN_BOOT NRF_GPIO_PIN_MAP(0,2)
#define NRF_PIN_SLEEP NRF_GPIO_PIN_MAP(0,3)
#define NRF_PIN_SPICS NRF_GPIO_PIN_MAP(0,7)
#define NRF_PIN_B6 NRF_PIN_SPICS
#define NRF_PIN_NFC1 NRF_GPIO_PIN_MAP(0,9)
#define NRF_PIN_B1 NRF_PIN_NFC1
#define NRF_PIN_NFC2 NRF_GPIO_PIN_MAP(0,10)
#define NRF_PIN_B2 NRF_PIN_NFC2
#define NRF_PIN_UARTRTS NRF_GPIO_PIN_MAP(0,11)
#define NRF_PIN_UARTCTS NRF_GPIO_PIN_MAP(0,12)
#define NRF_PIN_RESET NRF_GPIO_PIN_MAP(0,18)
#define NRF_PIN_SPICLK NRF_GPIO_PIN_MAP(0,19)
#define NRF_PIN_OPERATIONMODE NRF_PIN_SPICLK
#define NRF_PIN_SPI1 NRF_GPIO_PIN_MAP(0,21)
#define NRF_PIN_B5 NRF_PIN_SPI1
#define NRF_PIN_SPI2 NRF_GPIO_PIN_MAP(0,22)
#define NRF_PIN_BUSY NRF_PIN_SPI2
#define NRF_PIN_SPI3 NRF_GPIO_PIN_MAP(0,23)
#define NRF_PIN_B3 NRF_PIN_SPI3
#define NRF_PIN_SPI4 NRF_GPIO_PIN_MAP(1,0)
#define NRF_PIN_B4 NRF_PIN_SPI4
#define NRF_PIN_UARTTX NRF_GPIO_PIN_MAP(1,8)
#define NRF_PIN_UARTRX NRF_GPIO_PIN_MAP(1,9)


// LEDs definitions for PROTEUSIII
#define LEDS_NUMBER 0
#define LEDS_LIST { NRF_PIN_LED_1, NRF_PIN_LED_2 }
#define BSP_LED_0       NRF_PIN_LED_1
#define BSP_LED_1       NRF_PIN_LED_2
/* all LEDs are lit when GPIO is high */
#define LEDS_ACTIVE_STATE 1
#define LEDS_INV_MASK LEDS_MASK

// Buttons definitions for PROTEUSIII
#define BUTTONS_NUMBER 0
#define BUTTONS_LIST { NRF_PIN_SLEEP}
#define BSP_BUTTON_0  NRF_PIN_SLEEP
#define BUTTON_PULL NRF_GPIO_PIN_PULLUP
#define BUTTONS_ACTIVE_STATE 0

// UART definitions for PROTEUSIII
#define RX_PIN_NUMBER NRF_PIN_UARTRX
```

```
#define TX_PIN_NUMBER NRF_PIN_UARTTX
#define RTS_PIN_NUMBER NRF_PIN_UARTRTS
#define CTS_PIN_NUMBER NRF_PIN_UARTCTS


#ifdef __cplusplus
}
#endif

#endif // PROTEUSIII_H
```

Code 7: Content of the ProteusIII.h

6. In the project options, we need to link to the Proteus-III hardware instead to the Nordic evaluation board hardware. This can be done by adding "BOARD_PROTEUSIII" macro and by removing the respective macro of the Nordic platform in the precompiler options of the project.
7. Then check that the application code uses the pins names defined in the Proteus-III.h . Probably peripheral pins (UART, SPI,...), LED pins and/or button pins have to be adapted to fit the pin definition of the Proteus-III.h .

Please make sure that the selected pin number and its function matches the underlying hardware (e.g. evaluation board Proteus-III-EV).

8. Now all necessary changes have been done. Thus recompile the whole project and check for errors.
9. In case of success, erase the whole chip and flash ONLY the Softdevice onto the chip. The J-Flash tool can be used to do so.
10. After this, flash the compiled project code onto the chip using Keil (or the IDE of your choice) without erasing the flash area of the Softdevice.
11. Now, the whole code has been flashed and testing can start.

6.2.2 Firmware development hints

When creating a custom firmware the following hints may be useful during development:

- In standard Nordic examples, the *Reset* pin is hard coded. We recommend using the pin definition of the board-file to guarantee that changes in the layout take effect.
- After the chip was flashed or when a clock signal was applied to the *SWCLK* pin, the chip is in debug mode. In this case, all chip states are simulated. Please repower the chip to be in normal mode to test modes like the system off mode (especially when you want to measure currents of a low power mode).
- Reviewing the pin settings (direction, pull-up/-down resistors) of the firmware is the first option when experiencing leakage current.

- The *UART RX* pin is quite sensitive towards wrong levels during UART start-up. A floating *UART RX* pin of the SoC may result in unwanted behavior. In this case, an internal or external pull-up resistor can be installed to prevent floating. Be aware that this resistor will lead to leakage current.
- The *NFC* pins are optimized for NFC function and can lead to leakage current when not used properly in GPIO mode.
- Checkout the errata sheet of the nRF52 SoC to have an overview of known issues with the nRF52 SoC and possible software workarounds.
- Checkout the sections "Know issues" of the used SDK and soft device versions to be aware of potential issues.

7 Important notes

The following conditions apply to all goods within the wireless connectivity product range of Würth Elektronik eiSos GmbH & Co. KG:

7.1 General customer responsibility

Some goods within the product range of Würth Elektronik eiSos GmbH & Co. KG contain statements regarding general suitability for certain application areas. These statements about suitability are based on our knowledge and experience of typical requirements concerning the areas, serve as general guidance and cannot be estimated as binding statements about the suitability for a customer application. The responsibility for the applicability and use in a particular customer design is always solely within the authority of the customer. Due to this fact, it is up to the customer to evaluate, where appropriate to investigate and to decide whether the device with the specific product characteristics described in the product specification is valid and suitable for the respective customer application or not. Accordingly, the customer is cautioned to verify that the documentation is current before placing orders.

7.2 Customer responsibility related to specific, in particular safety-relevant applications

It has to be clearly pointed out that the possibility of a malfunction of electronic components or failure before the end of the usual lifetime cannot be completely eliminated in the current state of the art, even if the products are operated within the range of the specifications. The same statement is valid for all software sourcecode and firmware parts contained in or used with or for products in the wireless connectivity and sensor product range of Würth Elektronik eiSos GmbH & Co. KG. In certain customer applications requiring a high level of safety and especially in customer applications in which the malfunction or failure of an electronic component could endanger human life or health, it must be ensured by most advanced technological aid of suitable design of the customer application that no injury or damage is caused to third parties in the event of malfunction or failure of an electronic component.

7.3 Best care and attention

Any product-specific data sheets, manuals, application notes, PCN's, warnings and cautions must be strictly observed in the most recent versions and matching to the products firmware revisions. This documents can be downloaded from the product specific sections on the wireless connectivity homepage.

7.4 Customer support for product specifications

Some products within the product range may contain substances, which are subject to restrictions in certain jurisdictions in order to serve specific technical requirements. Necessary information is available on request. In this case, the field sales engineer or the internal sales person in charge should be contacted who will be happy to support in this matter.

7.5 Product improvements

Due to constant product improvement, product specifications may change from time to time. As a standard reporting procedure of the Product Change Notification (PCN) according to the JEDEC-Standard, we inform about major changes. In case of further queries regarding the PCN, the field sales engineer, the internal sales person or the technical support team in charge should be contacted. The basic responsibility of the customer as per section 7.1 and 7.2 remains unaffected. All wireless connectivity module driver software ¨wireless connectivity SDK¨ and it's source codes as well as all PC software tools are not subject to the Product Change Notification information process.

7.6 Product life cycle

Due to technical progress and economical evaluation we also reserve the right to discontinue production and delivery of products. As a standard reporting procedure of the Product Termination Notification (PTN) according to the JEDEC-Standard we will inform at an early stage about inevitable product discontinuance. According to this, we cannot ensure that all products within our product range will always be available. Therefore, it needs to be verified with the field sales engineer or the internal sales person in charge about the current product availability expectancy before or when the product for application design-in disposal is considered. The approach named above does not apply in the case of individual agreements deviating from the foregoing for customer-specific products.

7.7 Property rights

All the rights for contractual products produced by Würth Elektronik eiSos GmbH & Co. KG on the basis of ideas, development contracts as well as models or templates that are subject to copyright, patent or commercial protection supplied to the customer will remain with Würth Elektronik eiSos GmbH & Co. KG. Würth Elektronik eiSos GmbH & Co. KG does not warrant or represent that any license, either expressed or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, application, or process in which Würth Elektronik eiSos GmbH & Co. KG components or services are used.

7.8 General terms and conditions

Unless otherwise agreed in individual contracts, all orders are subject to the current version of the "General Terms and Conditions of Würth Elektronik eiSos Group", last version available at *www.we-online.com*.

8 Legal notice

8.1 Exclusion of liability

Würth Elektronik eiSos GmbH & Co. KG considers the information in this document to be correct at the time of publication. However, Würth Elektronik eiSos GmbH & Co. KG reserves the right to modify the information such as technical specifications or functions of its products or discontinue the production of these products or the support of one of these products without any written announcement or notification to customers. The customer must make sure that the information used corresponds to the latest published information. Würth Elektronik eiSos GmbH & Co. KG does not assume any liability for the use of its products. Würth Elektronik eiSos GmbH & Co. KG does not grant licenses for its patent rights or for any other of its intellectual property rights or third-party rights.

Notwithstanding anything above, Würth Elektronik eiSos GmbH & Co. KG makes no representations and/or warranties of any kind for the provided information related to their accuracy, correctness, completeness, usage of the products and/or usability for customer applications. Information published by Würth Elektronik eiSos GmbH & Co. KG regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof.

8.2 Suitability in customer applications

The customer bears the responsibility for compliance of systems or units, in which Würth Elektronik eiSos GmbH & Co. KG products are integrated, with applicable legal regulations. Customer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of Würth Elektronik eiSos GmbH & Co. KG components in its applications, notwithstanding any applications-related in-formation or support that may be provided by Würth Elektronik eiSos GmbH & Co. KG. Customer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences lessen the likelihood of failures that might cause harm and take appropriate remedial actions. The customer will fully indemnify Würth Elektronik eiSos GmbH & Co. KGand its representatives against any damages arising out of the use of any Würth Elektronik eiSos GmbH & Co. KG components in safety-critical applications.

8.3 Trademarks

AMBER wireless is a registered trademark of Würth Elektronik eiSos GmbH & Co. KG. All other trademarks, registered trademarks, and product names are the exclusive property of the respective owners.

8.4 Usage restriction

Würth Elektronik eiSos GmbH & Co. KG products have been designed and developed for usage in general electronic equipment only. This product is not authorized for use in equipment where a higher safety standard and reliability standard is especially required or where a failure of the product is reasonably expected to cause severe personal injury or death,

unless the parties have executed an agreement specifically governing such use. Moreover, Würth Elektronik eiSos GmbH & Co. KG products are neither designed nor intended for use in areas such as military, aerospace, aviation, nuclear control, submarine, transportation (automotive control, train control, ship control), transportation signal, disaster prevention, medical, public information network etc. Würth Elektronik eiSos GmbH & Co. KG must be informed about the intent of such usage before the design-in stage. In addition, sufficient reliability evaluation checks for safety must be performed on every electronic component, which is used in electrical circuits that require high safety and reliability function or performance. By using Würth Elektronik eiSos GmbH & Co. KG products, the customer agrees to these terms and conditions.

9 License terms

This License Terms will take effect upon the purchase and usage of the Würth Elektronik eiSos GmbH & Co. KG wireless connectivity products. You hereby agree that this license terms is applicable to the product and the incorporated software, firmware and source codes (collectively, "Software") made available by Würth Elektronik eiSos in any form, including but not limited to binary, executable or source code form.
The software included in any Würth Elektronik eiSos wireless connectivity product is purchased to you on the condition that you accept the terms and conditions of this license terms. You agree to comply with all provisions under this license terms.

9.1 Limited license

Würth Elektronik eiSos hereby grants you a limited, non-exclusive, non-transferable and royalty-free license to use the software and under the conditions that will be set forth in this license terms. You are free to use the provided Software only in connection with one of the products from Würth Elektronik eiSos to the extent described in this license terms. You are entitled to change or alter the source code for the sole purpose of creating an application embedding the Würth Elektronik eiSos wireless connectivity product. The transfer of the source code to third parties is allowed to the sole extent that the source code is used by such third parties in connection with our product or another hardware provided by Würth Elektronik eiSos under strict adherence of this license terms. Würth Elektronik eiSos will not assume any liability for the usage of the incorporated software and the source code. You are not entitled to transfer the source code in any form to third parties without prior written consent of Würth Elektronik eiSos.
You are not allowed to reproduce, translate, reverse engineer, decompile, disassemble or create derivative works of the incorporated Software and the source code in whole or in part. No more extensive rights to use and exploit the products are granted to you.

9.2 Usage and obligations

The responsibility for the applicability and use of the Würth Elektronik eiSos wireless connectivity product with the incorporated Firmware in a particular customer design is always solely within the authority of the customer. Due to this fact, it is up to you to evaluate and investigate, where appropriate, and to decide whether the device with the specific product characteristics described in the product specification is valid and suitable for your respective application or not.
You are responsible for using the Würth Elektronik eiSos wireless connectivity product with the incorporated Firmware in compliance with all applicable product liability and product safety laws. You acknowledge to minimize the risk of loss and harm to individuals and bear the risk for failure leading to personal injury or death due to your usage of the product.
Würth Elektronik eiSos' products with the incorporated Firmware are not authorized for use in safety-critical applications, or where a failure of the product is reasonably expected to cause severe personal injury or death. Moreover, Würth Elektronik eiSos' products with the incorporated Firmware are neither designed nor intended for use in areas such as military, aerospace, aviation, nuclear control, submarine, transportation (automotive control, train control, ship control), transportation signal, disaster prevention, medical, public information network etc. You shall inform Würth Elektronik eiSos about the intent of such usage before design-in stage. In certain customer applications requiring a very high level of safety and in which the malfunction or failure of an electronic component could endanger human life or

health, you must ensure to have all necessary expertise in the safety and regulatory ramifications of your applications. You acknowledge and agree that you are solely responsible for all legal, regulatory and safety-related requirements concerning your products and any use of Würth Elektronik eiSos' products with the incorporated Firmware in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by Würth Elektronik eiSos. YOU SHALL INDEMNIFY WÜRTH ELEKTRONIK EISOS AGAINST ANY DAMAGES ARISING OUT OF THE USE OF WÜRTH ELEKTRONIK EISOS' PRODUCTS WITH THE INCORPORATED FIRMWARE IN SUCH SAFETY-CRITICAL APPLICATIONS.

9.3 Ownership

The incorporated Firmware created by Würth Elektronik eiSos is and will remain the exclusive property of Würth Elektronik eiSos.

9.4 Firmware update(s)

You have the opportunity to request the current and actual Firmware for a bought wireless connectivity Product within the time of warranty. However, Würth Elektronik eiSos has no obligation to update a modules firmware in their production facilities, but can offer this as a service on request. The upload of firmware updates falls within your responsibility, e.g. via ACC or another software for firmware updates. Firmware updates will not be communicated automatically. It is within your responsibility to check the current version of a firmware in the latest version of the product manual on our website. The revision table in the product manual provides all necessary information about firmware updates. There is no right to be provided with binary files, so called "Firmware images", those could be flashed through JTAG, SWD, Spi-Bi-Wire, SPI or similar interfaces.

9.5 Disclaimer of warranty

THE FIRMWARE IS PROVIDED "AS IS". YOU ACKNOWLEDGE THAT WÜRTH ELEKTRONIK EISOS MAKES NO REPRESENTATIONS AND WARRANTIES OF ANY KIND RELATED TO, BUT NOT LIMITED TO THE NON-INFRINGEMENT OF THIRD PARTIES' INTELLECTUAL PROPERTY RIGHTS OR THE MERCHANTABILITY OR FITNESS FOR YOUR INTENDED PURPOSE OR USAGE. WÜRTH ELEKTRONIK EISOS DOES NOT WARRANT OR REPRESENT THAT ANY LICENSE, EITHER EXPRESS OR IMPLIED, IS GRANTED UNDER ANY PATENT RIGHT, COPYRIGHT, MASK WORK RIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT RELATING TO ANY COMBINATION, MACHINE, OR PROCESS IN WHICH THE WÜRTH ELEKTRONIK EISOS' PRODUCT WITH THE INCORPORATED FIRMWARE IS USED. INFORMATION PUBLISHED BY WÜRTH ELEKTRONIK EISOS REGARDING THIRD-PARTY PRODUCTS OR SERVICES DOES NOT CONSTITUTE A LICENSE FROM WÜRTH ELEKTRONIK EISOS TO USE SUCH PRODUCTS OR SERVICES OR A WARRANTY OR ENDORSEMENT THEREOF.

9.6 Limitation of liability

Any liability not expressly provided by Würth Elektronik eiSos shall be disclaimed.
You agree to hold us harmless from any third-party claims related to your usage of the Würth Elektronik eiSos' products with the incorporated Firmware, software and source code. Würth

Elektronik eiSos disclaims any liability for any alteration, development created by you or your customers as well as for any combination with other products.

9.7 Applicable law and jurisdiction

Applicable law to this license terms shall be the laws of the Federal Republic of Germany. Any dispute, claim or controversy arising out of or relating to this license terms shall be resolved and finally settled by the court competent for the location of Würth Elektronik eiSos' registered office.

9.8 Severability clause

If a provision of this license terms is or becomes invalid, unenforceable or null and void, this shall not affect the remaining provisions of the terms. The parties shall replace any such provisions with new valid provisions that most closely approximate the purpose of the terms.

9.9 Miscellaneous

Würth Elektronik eiSos reserves the right at any time to change this terms at its own discretion. It is your responsibility to check at Würth Elektronik eiSos homepage for any updates. Your continued usage of the products will be deemed as the acceptance of the change.
We recommend you to be updated about the status of new firmware and software, which is available on our website or in our data sheet and manual, and to implement new software in your device where appropriate.
By ordering a wireless connectivity product, you accept this license terms in all terms.

List of Figures

1 No security enabled 10
2 Just works pairing enabled 11
3 Static pass key pairing enabled 12
4 Lesc pass key pairing enabled 13
5 Lesc numeric comparison pairing enabled 14
6 Options for running the Proteus-III with standard or custom firmware 27
7 Pinout 29

List of Tables

1 RF-packet format 9
2 RF-packet command format 16

WÜRTH ELEKTRONIK

more than you expect

Internet of Things

Monitoring & Control

Automated Meter Reading

Contact:
Würth Elektronik eiSos GmbH & Co. KG
Division Wireless Connectivity & Sensors

Max-Eyth-Straße 1
74638 Waldenburg
Germany

Tel.: +49 651 99355-0
Fax.: +49 651 99355-69
www.we-online.com/wireless-connectivity