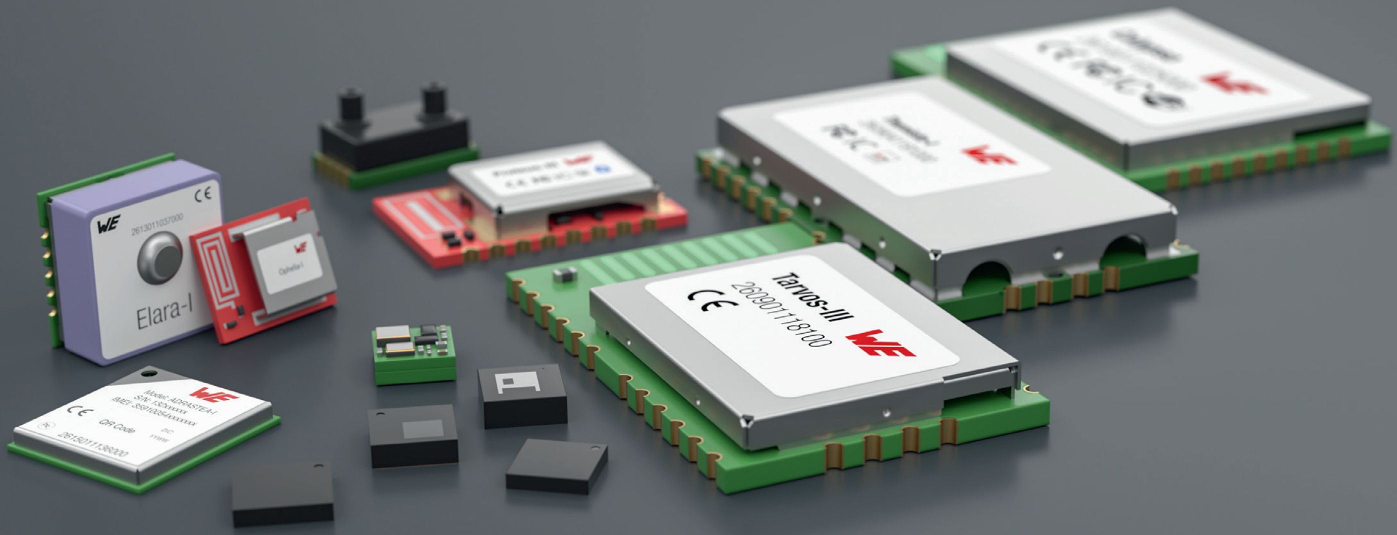

ANR009

Proteus-III / Proteus-III-SPI
Advanced developer guide

Version 1.5

December 4, 2023

WÜRTH ELEKTRONIK MORE THAN YOU EXPECT

Revision history

Manual version	FW version	HW version	Notes	Date
1.0	1.0.0	1.2	• Initial version	January 2020
1.1	1.0.0	1.2	• Added information on Bluetooth® stack version	January 2021
1.2	1.0.0	1.2	• Added the note that this document is also valid for Proteus-III-SPI	February 2021
1.3	1.3.0	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
1.4	1.4.0	1.2	• Added chapter `References` • Updated chapter `Remote GPIO control` concerning Proteus-III firmware version 1.4.0 • Updated `Important notes`, meta data and document style	July 2023
1.5	1.5.0	1.5	• Added new remote command `CMD_GETSTATE_REQ` • Moved chapter `Remote GPIO control` to `4.4.2`	December 2023

Abbreviations

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

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.

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 [4, 5, 6] 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 WE 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 WE SPP-like profile

This section contains the key data of the WE 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 (Φ_{ST}). 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 Header + 3 Byte NUS header, which is 247 Byte in total.
The PDU size should be 243 Byte payload + 1 Byte 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 packet header. Thus, the standard Bluetooth® LE payload has to match the following format to be understood by the Proteus-III. Other Bluetooth® LE frames will be discarded:

Bluetooth® LE Payload	
Header	Payload data
0x01	Φ_{ST} Bytes

Table 1: RF-packet format to transmit data

Bluetooth® LE Payload	
Header	Command data
0x02	Φ_{ST} Bytes

Table 2: RF-packet format to transmit commands, like remote GPIO commands (see chapter 4.4.2)

Bluetooth® LE Payload			
Header	Sequence number	Fragment ID	Payload data
0x04	1 Byte	1 Byte	$\Phi_{ST} - 2$ Bytes

Table 3: RF-packet format for fragmented data of the high throughput mode (see ANR006 [3])

The maximum payload value Φ_{ST} is negotiated during connection setup. The Proteus-III supports up to 243 Bytes.

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 WE 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 (1 byte header, up to 19 bytes payload) in RF-packet format (see Table 1) using the Würth Elektronik eiSos company identifier 0x031A. This manufacturer data is used to realize the Beacon feature (see command `CMD_SETBEACON_REQ` in the user manual [4, 5, 6]).

4.4 Remote commands

There are commands that can be send from the connected peer via radio to the Proteus-III, to run actions like GPIO control.

4.4.1 CMD_GETSTATE_REQ

Request the current state information such as the negotiated maximum payload size of the connection Φ_{ST}. Format:

Header	Command
0x02	0x01

Response (`CMD_GETSTATE_CNF`):

Header	Command | 0x40	Φ_{ST}
0x02	0x41	1 byte

4.4.2 Remote GPIO control

The Proteus-III contains a feature to control its free GPIOs via the Bluetooth® LE interface. To do so a connected remote device must send remote commands via the Bluetooth® LE interface to the Proteus-III.

In case the GPIOs of interest have not been configured by the locally connected host via UART commands, it must be configured first by the remote device via Bluetooth® LE using the `CMD_GPIO_REMOTE_WRITECONFIG_REQ` and `CMD_GPIO_REMOTE_READCONFIG_REQ` commands.

If this has been done, the GPIOs can be used as input and/or output pins (`CMD_GPIO_REMOTE_WRITE_REQ` / `CMD_GPIO_REMOTE_READ_REQ`).

4.4.2.1 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:

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

Response (`CMD_GPIO_REMOTE_WRITECONFIG_CNF`):

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 [4, 5, 6]

Function:

0x00: GPIO disconnected

0x01: GPIO works as input

0x02: GPIO works as output

0x03: GPIO works as PWM

Value:

- if **Function** is disconnected:

  0x00: value field must use 0x00
- 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: Ratio (0x00 = 0%,...,0xFE=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 [4, 5, 6]

Status:

- **0x00:** Success
- **0x01:** Failed
- **0xFF:** Remote configuration not allowed (blocked by the user setting `CFG_Flags` of the remote device)

Example: Configure two GPIOs of the connected remote device to output high Configure the GPIOs with ID **0x01** and **0x02** to output high:

Header	Command	Block$_1$	Block$_2$
0x02	0x28	0x03 **0x01** 0x02 0x01	0x03 **0x02** 0x02 0x01

Response:

Header	Command | 0x40	Block$_1$	Block$_2$
0x02	0x68	0x02 **0x01** 0x00	0x02 **0x02** 0x00

Configured both GPIOs with success.

4.4.2.2 CMD_GPIO_REMOTE_READCONFIG_REQ

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

Header	Command
0x02	0x2C

Response (`CMD_GPIO_REMOTE_READCONFIG_CNF`):

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: 2
- if **Function** is input or output: 3
- if **Function** is PWM: 5

GPIO_ID: ID of the GPIO, see Proteus-III manual [4, 5, 6]

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:
  Value 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: Ratio (0x00 = 0%,. . . ,0xFE=100%)

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

Header	Command
0x02	0x2C

Response:

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.

4.4.2.3 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:

Header	Command	Block$_1$	...	Block$_n$
0x02	0x29	x Bytes		x Bytes

Response (`CMD_GPIO_REMOTE_WRITE_CNF`):

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 [4, 5, 6]

Value:

- if **Function** is output

  0x00: Set GPIO to LOW

  0x01: Set GPIO to HIGH

- if **Function** is PWM

  Byte 0: Ratio (0x00 = 0%,. . . ,0xFE=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 [4, 5, 6]

Status:

> **0x00:** Success
>
> **0x01:** Failed

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

Header	Command	Block$_1$
0x02	0x29	0x02 **0x01** 0x00

Response:

Header	Command | 0x40	Block$_1$
0x02	0x69	0x02 **0x01** 0x00

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

4.4.2.4 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:

Header	Command	Block$_1$
0x02	0x2A	x Bytes

Response (`CMD_GPIO_REMOTE_READ_CNF`)

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 [4, 5, 6]

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 [4, 5, 6]

Value:

- if **Function** is input or output

  0x00: The GPIO is LOW

  0x01: The GPIO is HIGH

  0xFF: Failed reading remote GPIO value.
- if **Function** is PWM

  0xFF: Failed reading remote GPIO value.

  Other: Ratio (0x00=0%,. . . , 0xFE=100%)

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

Header	Command	Block$_1$
0x02	0x2A	0x02 **0x01 0x02**

Response:

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).

4.4.2.5 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:

Header	Command	Block$_1$	. . .	Block$_n$
0x02	0xA6	x Bytes		x Bytes

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

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

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 App development

The definition of the WE 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.

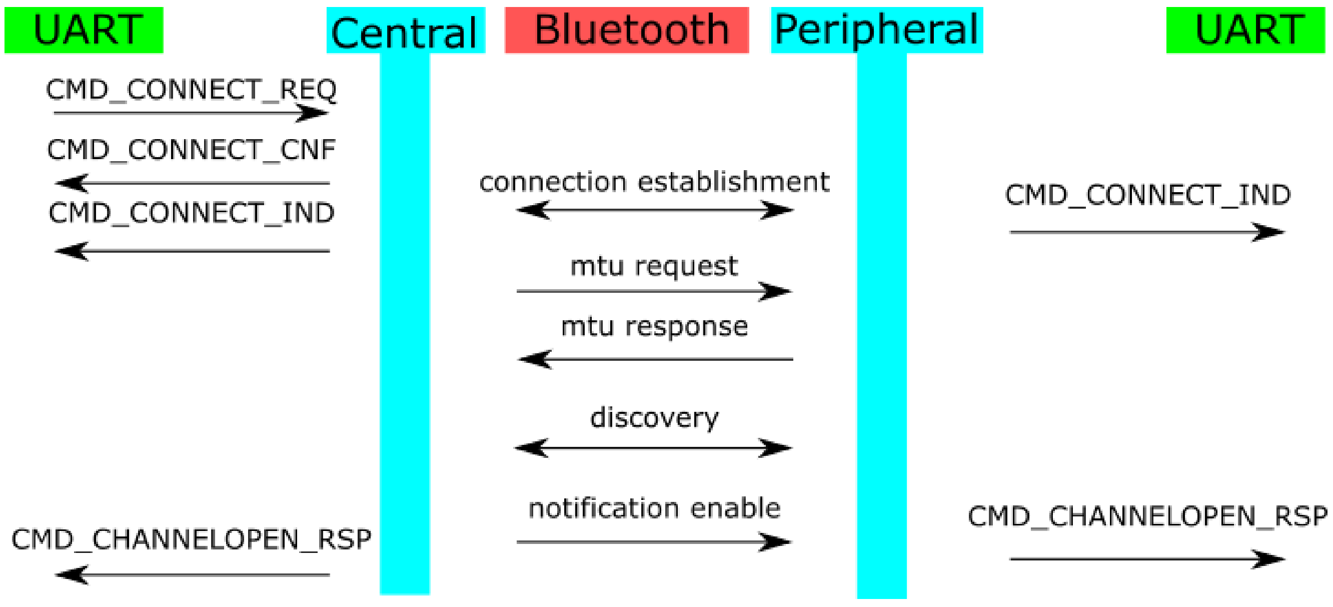

Figure 1: No security enabled

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.

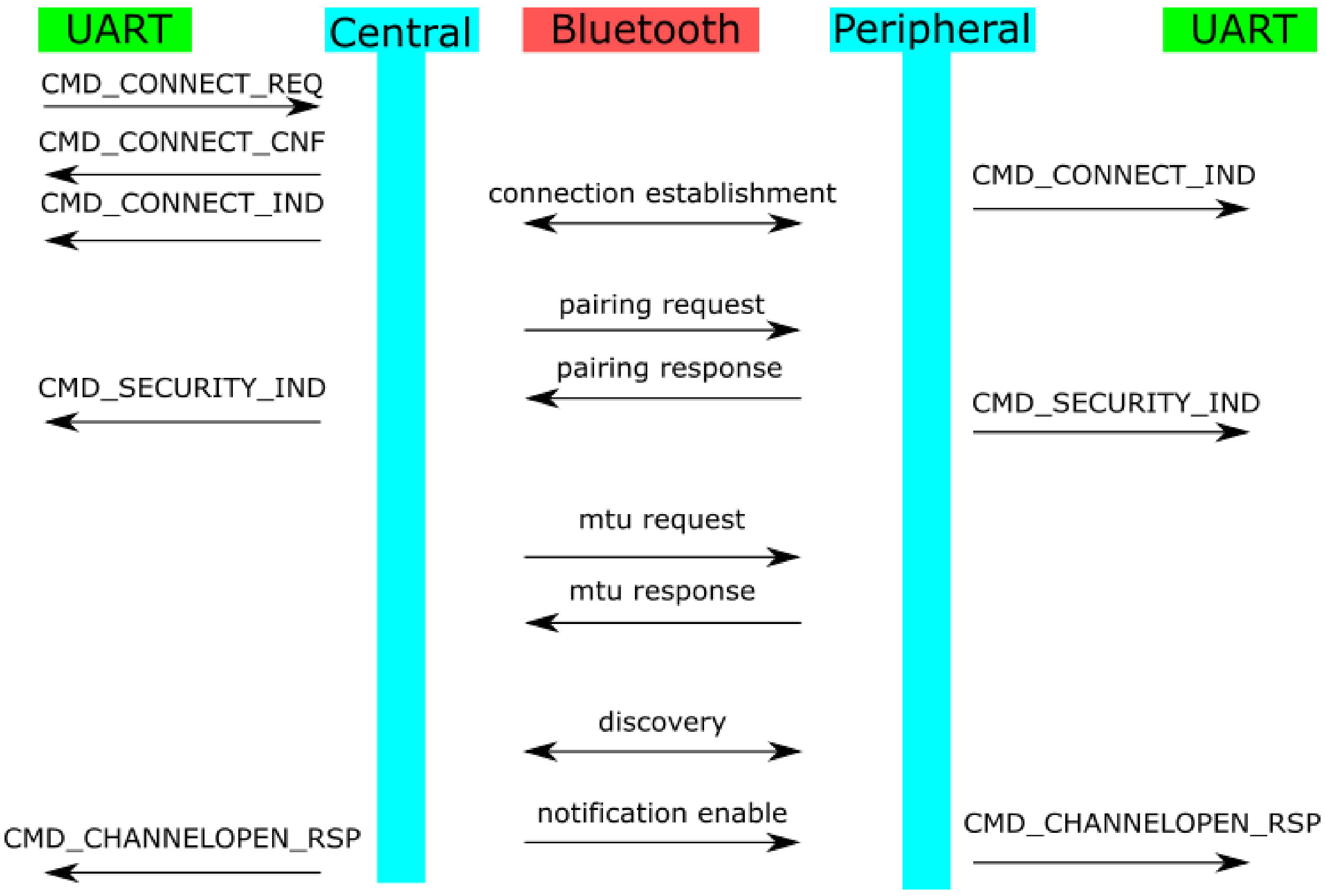

Figure 2: Just works pairing enabled

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.

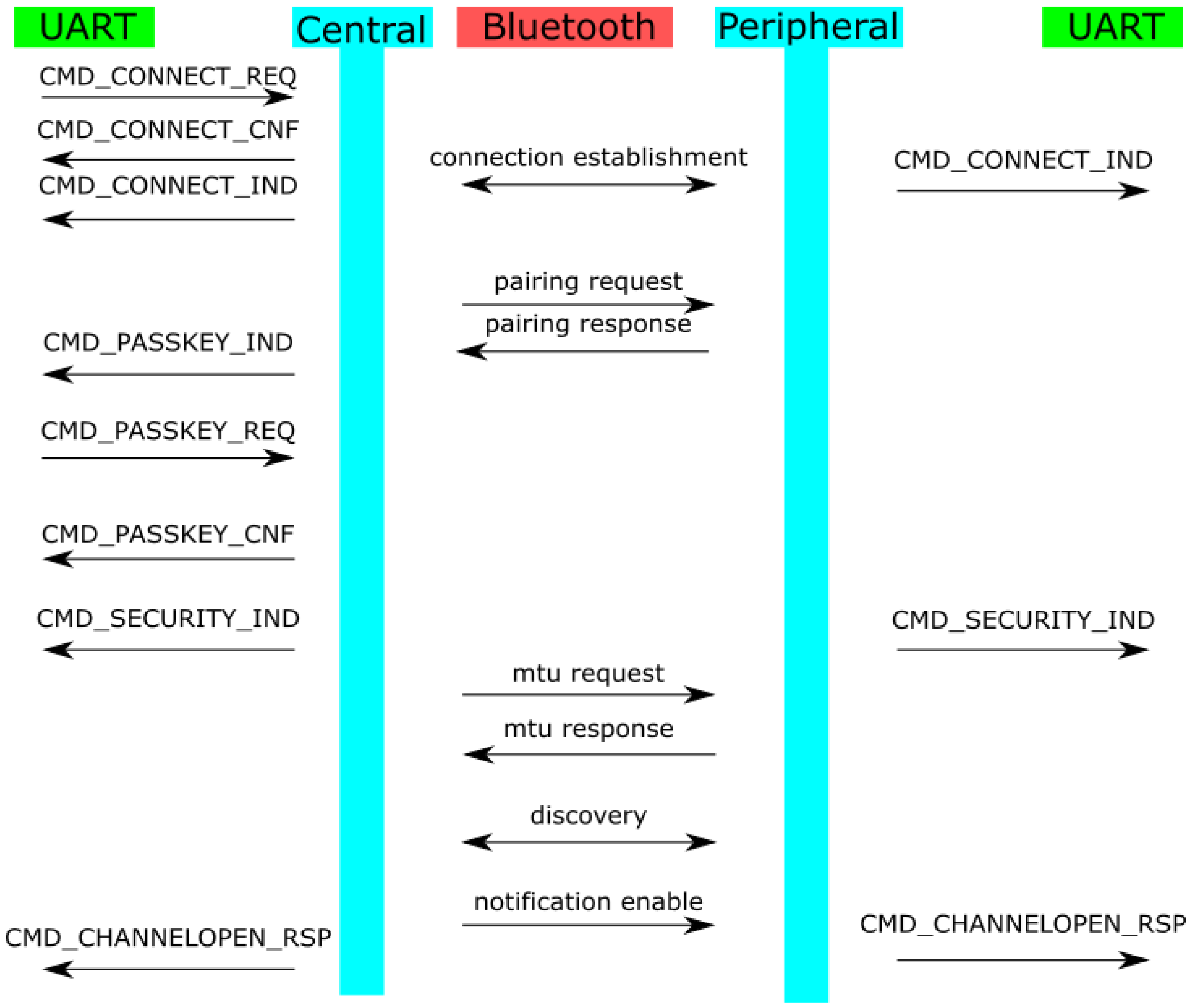

Figure 3: Static pass key pairing enabled

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.

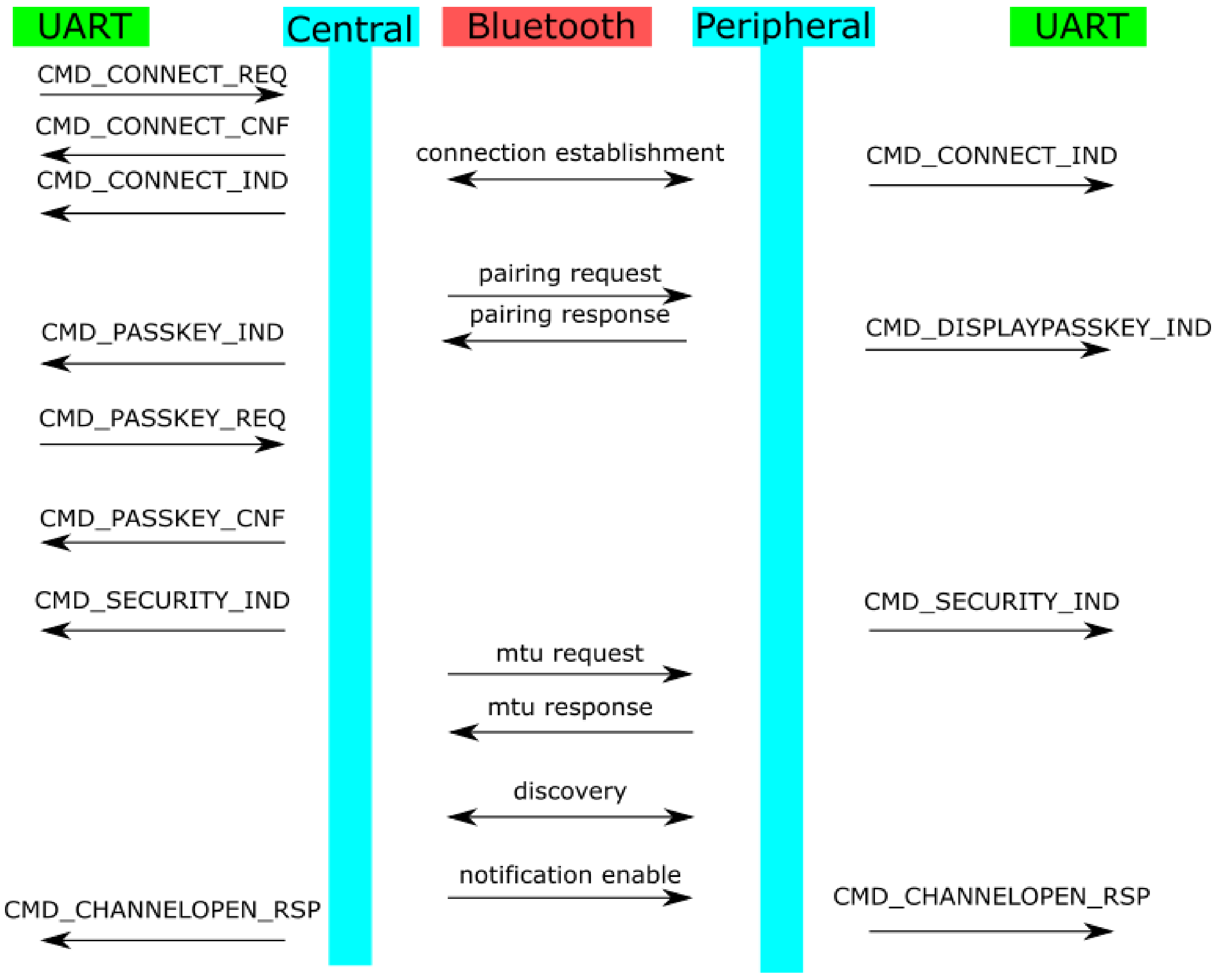

Figure 4: Lesc pass key pairing enabled

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.

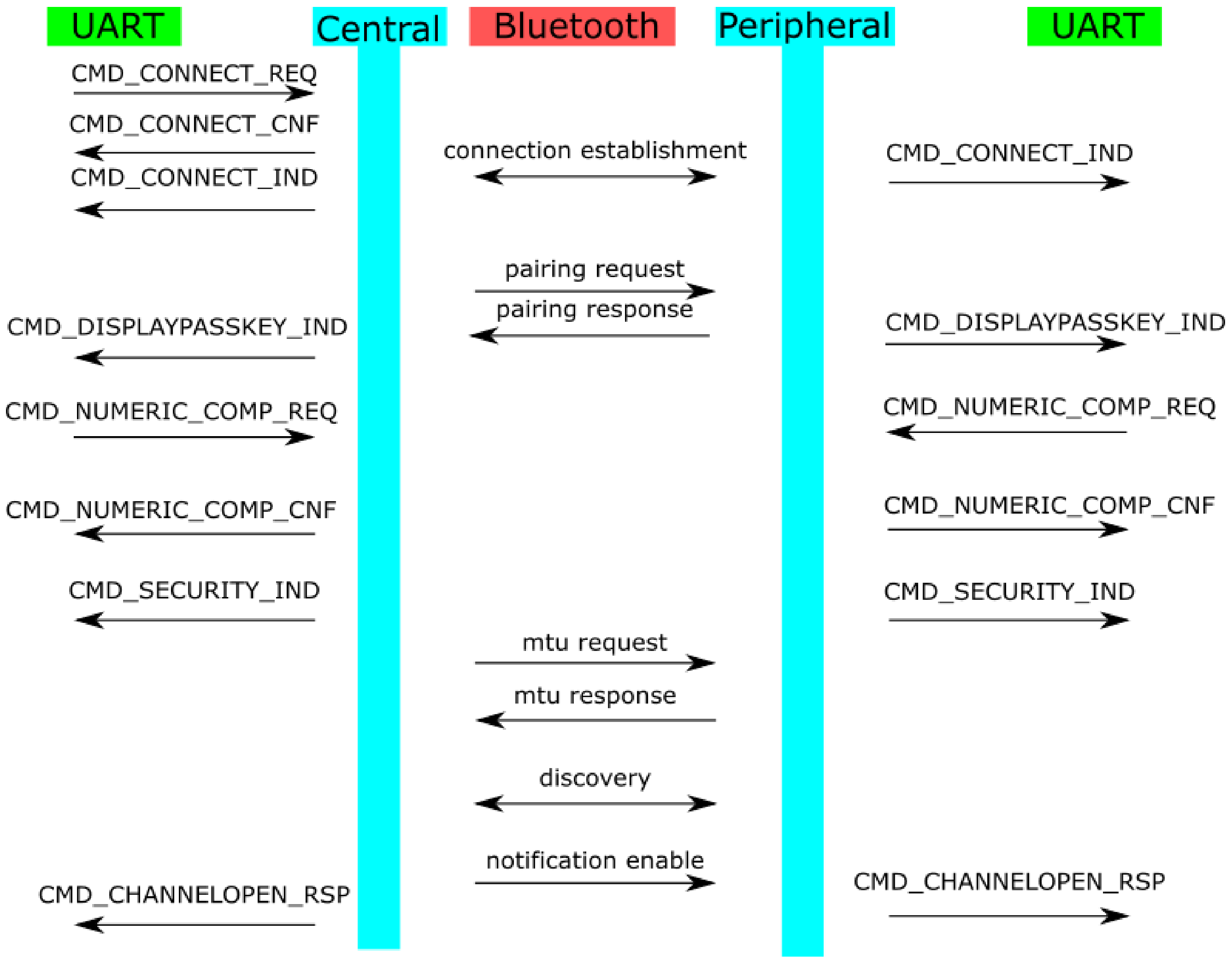

Figure 5: Lesc numeric comparison pairing enabled

.

5.2 Enable notifications

As described in the previous chapter 5.1 the final step for a successful connection setup 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 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.

5.4 Nordic Bluetooth LE UART example app as base

Nordic Semiconductor provides source code to develop Android, iOS and Windows applications. To implement the WE 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-C352-11E5-953
    D-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 RF_HEADER_TYPE_DATA = 0x01;
final byte[] buffer = mOutgoingBuffer = (RF_HEADER_TYPE_DATA + text).getBytes();
mBufferOffset = 0;
...
}
```

Code 3: Add packet header on sender side

```
public void onDataSent(final String data) {
if (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 (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).

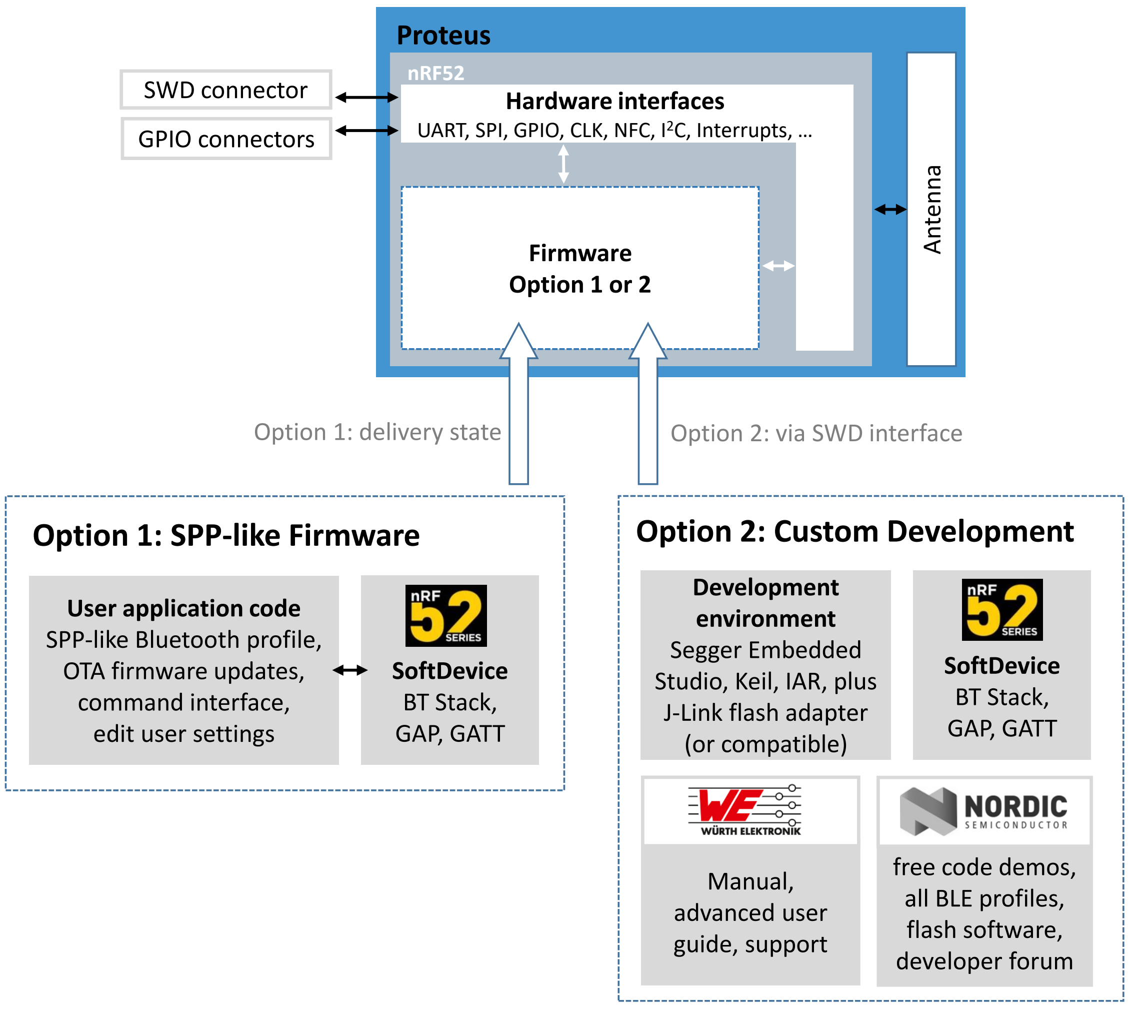

Figure 6: Options for running the Proteus-III with standard or custom firmware

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 *WCS@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. Please check the nRF52840 product specification [1] for all details regarding the pins, dependencies and possible functions of the Nordic chip set.

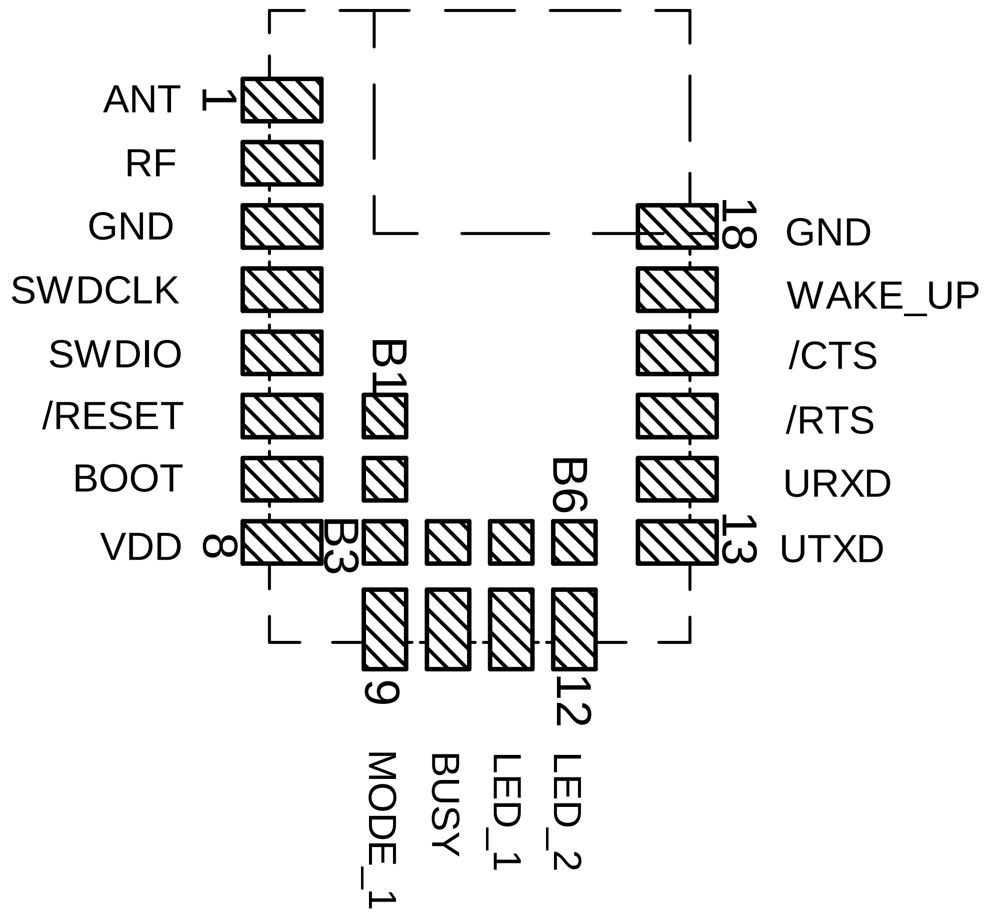

Figure 7: Pinout

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

- 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 [4, 5, 6] and in the application note ANR027 [2].

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 BTMAC) 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.
- To use module pins that support the NFC function as normal GPIOs, the NFC function must be disabled at compile time. To do so, the macro CONFIG_NFCT_PINS_AS_GPIOS has to be defined in the project preprocessor options. Otherwise, the respective pins are blocked for GPIO usage.
- 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 "Known issues" of the used SDK and soft device versions to be aware of potential issues.

6.2.3 Qualifying the Proteus-III with respect to Bluetooth® 5.2

The Proteus-III has been listed as end product with respect to Bluetooth® 5.1 specification. In case a listing with respect to a newer version of the Bluetooth® standard is desired another Bluetooth® LE stack must be used.

The end product listing (EPL) mainly has to be built up by adding two parts:

1. The listing of the controller subsystem, that contains the test of the module hardware to be Bluetooth® compliant. The hardware of the Proteus-III fulfills the specifications to be listed as a Bluetooth® 5.2 controller subsystem.
2. The listing of the host subsystem, that contains the test of the Bluetooth® stack used within the product. The Bluetooth® stack used in the Proteus-III is the S112 V7.3.0, that does not fulfill the specifications to be listed as a Bluetooth® 5.2 host subsystem. But another Bluetooth® 5.2 listed stack, like "Zephyr BLE Host" developed by Nordic, can be used to create a new Bluetooth® 5.2 listed end product listing.

7 References

[1] Nordic Semiconductor. Nordic nRF52840 resources. `https://www.nordicsemi.com/products/nrf52840`.

[2] Würth Elektronik. Application note 27 - Bluetooth listing guide. `http://www.we-online.com/ANR027`.

[3] Würth Elektronik. Application note 6 - Proteus high throughput mode. `http://www.we-online.com/ANR006`.

[4] Würth Elektronik. Proteus-I user manual. `https://www.we-online.de/katalog/de/manual/2608011024000`.

[5] Würth Elektronik. Proteus-II user manual. `https://www.we-online.de/katalog/de/manual/2608011024010`.

[6] Würth Elektronik. Proteus-III user manual. `https://www.we-online.de/katalog/de/manual/2611011024000`.

8 Important notes

The Application Note and its containing information ("Information") is based on Würth Elektronik eiSos GmbH & Co. KG and its subsidiaries and affiliates ("WE eiSos") knowledge and experience of typical requirements concerning these areas. It serves as general guidance and shall not be construed as a commitment for the suitability for customer applications by WE eiSos. While WE eiSos has used reasonable efforts to ensure the accuracy of the Information, WE eiSos does not guarantee that the Information is error-free, nor makes any other representation, warranty or guarantee that the Information is completely accurate or up-to-date. The Information is subject to change without notice. To the extent permitted by law, the Information shall not be reproduced or copied without WE eiSos' prior written permission. In any case, the Information, in full or in parts, may not be altered, falsified or distorted nor be used for any unauthorized purpose.

WE eiSos is not liable for application assistance of any kind. Customer may use WE eiSos' assistance and product recommendations for customer's applications and design. No oral or written Information given by WE eiSos or its distributors, agents or employees will operate to create any warranty or guarantee or vary any official documentation of the product e.g. data sheets and user manuals towards customer and customer shall not rely on any provided Information. THE INFORMATION IS PROVIDED "AS IS". CUSTOMER ACKNOWLEDGES THAT WE 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 A PURPOSE OR USAGE. WE 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 WE EISOS INFORMATION IS USED. INFORMATION PUBLISHED BY WE EISOS REGARDING THIRD-PARTY PRODUCTS OR SERVICES DOES NOT CONSTITUTE A LICENSE FROM WE eiSos TO USE SUCH PRODUCTS OR SERVICES OR A WARRANTY OR ENDORSEMENT THEREOF.

The responsibility for the applicability and use of WE eiSos' components 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 and investigate, where appropriate, and decide whether the device with the specific characteristics described in the specification is valid and suitable for the respective customer application or not. The technical specifications are stated in the current data sheet and user manual of the component. Therefore the customers shall use the data sheets and user manuals and are cautioned to verify that they are current. The data sheets and user manuals can be downloaded at *www.we-online.com*. Customers shall strictly observe any product-specific notes, cautions and warnings. WE eiSos reserves the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time without notice.

WE eiSos will in no case be liable for customer's use, or the results of the use, of the components or any accompanying written materials. IT IS CUSTOMER'S RESPONSIBILITY TO VERIFY THE RESULTS OF THE USE OF THIS INFORMATION IN IT'S OWN PARTICULAR ENGINEERING AND PRODUCT ENVIRONMENT AND CUSTOMER ASSUMES THE ENTIRE RISK OF DOING SO OR FAILING TO DO SO. IN NO CASE WILL WE EISOS BE LIABLE FOR

CUSTOMER'S USE, OR THE RESULTS OF IT'S USE OF THE COMPONENTS OR ANY ACCOMPANYING WRITTEN MATERIAL IF CUSTOMER TRANSLATES, ALTERS, ARRANGES, TRANSFORMS, OR OTHERWISE MODIFIES THE INFORMATION IN ANY WAY, SHAPE OR FORM.

If customer determines that the components are valid and suitable for a particular design and wants to order the corresponding components, customer acknowledges to minimize the risk of loss and harm to individuals and bears the risk for failure leading to personal injury or death due to customers usage of the components. The components have been designed and developed for usage in general electronic equipment only. The components are not authorized for use in equipment where a higher safety standard and reliability standard is especially required or where a failure of the components is reasonably expected to cause severe personal injury or death, unless WE eiSos and customer have executed an agreement specifically governing such use. Moreover WE eiSos components are neither designed nor intended for use in areas such as military, aerospace, aviation, nuclear control, submarine, transportation, transportation signal, disaster prevention, medical, public information network etc. WE eiSos 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 component which is used in electrical circuits that require high safety and reliability functions or performance. COSTUMER SHALL INDEMNIFY WE EISOS AGAINST ANY DAMAGES ARISING OUT OF THE USE OF THE COMPONENTS IN SUCH SAFETY-CRITICAL APPLICATIONS.

List of Figures

List of Tables

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

WÜRTH ELEKTRONIK MORE THAN YOU EXPECT