## Revision history

<table>
<thead>
<tr>
<th>Manual version</th>
<th>Version for STM32</th>
<th>Version for Raspb. Pi</th>
<th>Notes</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>-</td>
<td>3.0.0</td>
<td>• Initial version of this document</td>
<td>April 2019</td>
</tr>
<tr>
<td>1.2</td>
<td>-</td>
<td>3.0.0</td>
<td>• Updated file name to new AppNote name structure. Updated important notes, legal notice &amp; license terms chapters.</td>
<td>June 2019</td>
</tr>
<tr>
<td>1.3</td>
<td>-</td>
<td>3.1.0</td>
<td>• Updated supported modules</td>
<td>September 2019</td>
</tr>
<tr>
<td>1.4</td>
<td>-</td>
<td>3.2.0</td>
<td>• Updated supported modules</td>
<td>May 2020</td>
</tr>
<tr>
<td>1.5</td>
<td>-</td>
<td>3.3.0</td>
<td>• Updated supported modules</td>
<td>February 2021</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Updated supported libraries</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Updated installation instructions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Added description for SPI interfaces</td>
<td></td>
</tr>
<tr>
<td>1.6</td>
<td>-</td>
<td>3.4.0</td>
<td>• Fixed wrong links</td>
<td>August 2021</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Updated installation description of FTDI drivers</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Renamed USB radio sticks variants</td>
<td></td>
</tr>
</tbody>
</table>
### ANR008 - Wireless Connectivity SDK

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.7</td>
<td>August 2021</td>
<td>- Added new driver for STM32 microcontrollers (see chapter 2).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Restructured chapters for better overview</td>
</tr>
<tr>
<td>1.8</td>
<td>May 2022</td>
<td>- The STM32 SDK is now the primary version of the SDK. The Raspberry Pi</td>
</tr>
<tr>
<td></td>
<td></td>
<td>version has been archived.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Updated supported modules</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Updated chapter 2 (Wireless Connectivity SDK for STM32)</td>
</tr>
<tr>
<td>1.9</td>
<td>April 2023</td>
<td>- Updated SDK history of STM32 version (chapter 2.2)</td>
</tr>
<tr>
<td>1.10</td>
<td>July 2023</td>
<td>- Wireless Connectivity SDK for Raspberry has been archived</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Updated <strong>Important notes and meta data</strong></td>
</tr>
</tbody>
</table>

* For SDK version history see chapter 2.2 (for STM32) and chapter 3.2 (for Raspberry Pi).
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
<td>Checksum</td>
<td></td>
</tr>
<tr>
<td>DC</td>
<td>Duty cycle</td>
<td>Active transmission time per hour expressed as percentage. 1% means, channel is occupied for 36 seconds per hour.</td>
</tr>
<tr>
<td>FSE</td>
<td>Field Sales Engineer</td>
<td>Support and sales contact person responsible for limited sales area</td>
</tr>
<tr>
<td>0xhh [HEX]</td>
<td>Hexadecimal</td>
<td>The prefix 0x indicates hexadecimal values. All other numbers are decimal values.</td>
</tr>
<tr>
<td>HIGH</td>
<td>High signal level</td>
<td></td>
</tr>
<tr>
<td>LOW</td>
<td>Low signal level</td>
<td></td>
</tr>
<tr>
<td>LPM</td>
<td>Low power mode</td>
<td>Operation mode with reduced energy consumption.</td>
</tr>
<tr>
<td>LRM</td>
<td>Long range mode</td>
<td>Tx mode increasing the RX sensitivity by using spreading and forward error correction</td>
</tr>
<tr>
<td>LSB</td>
<td>Least significant bit</td>
<td></td>
</tr>
<tr>
<td>MSB</td>
<td>Most significant bit</td>
<td></td>
</tr>
<tr>
<td>PL</td>
<td>Payload</td>
<td>The real, non-redundant information in a frame/packet.</td>
</tr>
<tr>
<td>RF</td>
<td>Radio frequency</td>
<td>Describes everything relating to the wireless transmission.</td>
</tr>
<tr>
<td>SDK</td>
<td>Software development kit</td>
<td>Software code that implements the command interface of various Würth Elektronik eiSos products</td>
</tr>
<tr>
<td>UART</td>
<td>Universal Asynchronous Receiver Transmitter - a serial data transmission interface</td>
<td></td>
</tr>
<tr>
<td>VDD</td>
<td>Supply voltage</td>
<td></td>
</tr>
</tbody>
</table>
Contents

1 Introduction ................................. 5
   1.1 Motivation .............................. 7

2 Wireless Connectivity SDK for STM32 ..... 9
   2.1 Content .................................. 9
   2.2 Software history .......................... 10
   2.3 Host integration .......................... 11
   2.4 Running sample applications on the STM32 Nucleo board ................. 13
      2.4.1 Hardware connections ................. 13
      2.4.2 Run the project with STM32CubeIDE 14

3 Wireless Connectivity SDK for Raspberry Pi 16
   3.1 Content .................................. 16
   3.2 Software history .......................... 18
   3.3 Host integration .......................... 19
   3.4 Running sample applications on the Raspberry Pi .................... 21
      3.4.1 Hardware connections ................. 21
      3.4.2 Install the Raspberry Pi OS on the Raspberry Pi ................. 22
      3.4.3 Configuring the peripherals ............ 23
      3.4.4 Install the required libraries .......... 24
         3.4.4.1 Install the libgpiod library ........ 24
         3.4.4.2 Install the FTDI driver for USB (optional) ..... 25
      3.4.5 Install the Wireless Connectivity SDK .............. 26
      3.4.6 FAQ - Frequently asked questions .......... 28
         3.4.6.1 The initialization function fails, what can I do? 28
         3.4.6.2 ProteusIII-SPI: Pin Wake-up leads to "GetPin: Could not read pin level" 28

4 References ................................ 29

5 Important notes ......................... 30
1 Introduction

The Würth Elektronik eiSos wireless modules provide an easy to use radio interface to any embedded application. The module’s interface with the host processor of the embedded application via UART can be operated using a command interface.

The Wireless Connectivity SDK is a set of software tools that enable quick software integration of Würth Elektronik eiSos wireless modules into any of the most commonly used host processors. It consists of drivers and examples in C-code that use the UART, SPI or USB peripheral of the underlying platform to communicate with the attached radio device.

The Wireless Connectivity SDK has been developed for the STM32 (see chapter 2) and Raspberry Pi 3 (see chapter 3) platforms.

⚠️ The Wireless Connectivity SDK for Raspberry Pi 3 has been archived and will no longer be maintained.
Figure 1: Wireless connectivity SDK driver as part of the end product
1.1 Motivation

The aim of the Wireless Connectivity SDK is to minimize the effort required on customer side to enable his host MCU to communicate with Würth Elektronik eiSos radio modules. It contains the implementation of all available commands in pure C-code. In order to integrate any Würth Elektronik eiSos wireless module, the user has to simply port the corresponding C-code to his host processor. This significantly reduces the time needed for developing the software interface to the radio module.

Würth Elektronik eiSos products, like the 868MHz proprietary radio module Tarvos-III, use a so called command interface for configuration and operation tasks. This interface provides up to 30 commands that accomplish tasks like updating various device settings, transmit/receive data and putting the module into one of various low power modes.

There are Würth Elektronik eiSos wireless modules that can operate in transparent mode in addition to the standard command mode. When using the transparent mode, the device does not interpret the commands sent via UART. Please make sure that the connected radio device runs in command mode to use the Wireless Connectivity SDK.

The commands of such an interface can be divided into 3 categories:

1. Requests: The host requests the module to trigger an action, e.g. in case of the request CMD_RESET_REQ the host asks the module to perform a reset.

2. Confirmations: On each request the module answers with a confirmation message as a feedback on the requested operation status. In case of a CMD_RESET_REQ, the module answers with a CMD_RESET_CNF to tell the host whether the reset will be performed or not.

3. Indications and Responses: In case of special events, the module indicates the same spontaneously to the host. The CMD_DATAEX_IND indicates for example that data was received via radio.

The commands itself have the following format:

<table>
<thead>
<tr>
<th>Start byte</th>
<th>Command</th>
<th>Length</th>
<th>Payload</th>
<th>CS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x02</td>
<td>1 Byte</td>
<td>1 Byte</td>
<td>Length Bytes</td>
<td>1 Byte</td>
</tr>
</tbody>
</table>

Example: CMD_DATA_REQ of the Tarvos-III
The CMD_DATA_REQ has the command number 0x00. It provides a simple data transfer. The length field indicates the number of bytes to be transmitted via radio.

Format:

<table>
<thead>
<tr>
<th>Start byte</th>
<th>Command</th>
<th>Length</th>
<th>Payload</th>
<th>CS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x02</td>
<td>0x00</td>
<td>1 Byte</td>
<td>Length Bytes</td>
<td>1 Byte</td>
</tr>
</tbody>
</table>

Sending "Hello World!"
With the above command, we are sending 12 bytes (0x0C), corresponding to the ASCII string "Hello World!" (0x48 0x65 0x6C 0x6C 0x6F 0x20 0x57 0x6F 0x72 0x6C 0x64 0x21) and the resulting checksum is 0x0F.

To use the complete feature set of such a radio device, all available commands of the corresponding command interface have to be implemented on the custom host processor. This involves considerable effort for the user and this is exactly the reason why Würth Elektronik eiSos offers the Wireless Connectivity SDK.

The steps for porting are explained in more detail in chapter 2.3 and 3.3.
2 Wireless Connectivity SDK for STM32

2.1 Content

The radio modules supported by the latest version of the Wireless Connectivity SDK for STM32 are:

<table>
<thead>
<tr>
<th>SDK version</th>
<th>Radio standard</th>
<th>Radio module &amp; USB dongle</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.7.1</td>
<td>Bluetooth® LE</td>
<td>Proteus-III, Proteus-e</td>
</tr>
<tr>
<td></td>
<td>Cellular</td>
<td>Adrastea-I</td>
</tr>
<tr>
<td></td>
<td>Proprietary 868 MHz</td>
<td>Tarvos-III, Thebe-II</td>
</tr>
<tr>
<td></td>
<td>Proprietary 915 MHz</td>
<td>Telesto-III, Themisto-I</td>
</tr>
<tr>
<td></td>
<td>Proprietary 2.4 GHz</td>
<td>Thyone-I</td>
</tr>
<tr>
<td></td>
<td>Wi-Fi / WLAN</td>
<td>Calypso</td>
</tr>
<tr>
<td>Wireless M-BUS</td>
<td>Shared driver for Metis-II, Metis-I, Mimas-I</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Wireless Connectivity SDK for STM32

At the topmost level, the Wireless Connectivity SDK directory structure looks as follows.

```
/  
  `WCON_SDK` ................................. Root directory of the SDK
     `
     `STM32F4xx` ......................... STM32F4 driver (HAL, peripherals, ...) and project file
     `
     `STM32L0xx` ............................. STM32L0 driver (HAL, peripherals, ...) and project file
     `
     `WCON_Drivers` ......................... Wireless module drivers and example code
     `
     `Readme.md` ................................ Readme file
```

The root directory of the Wireless Connectivity SDK (`WCON_SDK`) contains driver directories for each supported family of STM32 microcontrollers (STM32F4xx and STM32L0xx) as well as the `WCON_Drivers` directory, which contains all code related to the Würth Elektronik eiSos wireless modules. Within this directory, each supported radio module has its own subdirectory containing the implementation of its command interface and a main file with a short example application.

Besides the module-specific directories, the subdirectory `global` contains all the shared functions as well as definitions related to the serial communication and GPIO interfaces of the underlying host.

The wired communication interface currently supported by the Wireless Connectivity SDK for STM32 is UART. The SPI interface (STM32 role Master) is currently not supported.
2.2 Software history

Version 1.0.0 "Release"
- Initial version of the SDK

Version 1.1.0 "Release"
- Added driver for the Calypso Wi-Fi radio module

Version 1.2.0 "Release"
- Added driver for the Proteus-e Bluetooth LE module
- Updated Proteus-III driver

Version 1.3.0 "Release"
- Improvement of the Thyone-I driver

Version 1.4.0 "Release"
- Improvement of the Calypso driver

Version 1.5.0 "Release"
- Improvement of the Proteus-III driver
- Added Proteus-II driver
- Restructured AT-command codes for coming radio modules

Version 1.7.0 "Release"
- Added driver for Adrastea-I
2.3 Host integration

As described in chapter 2.1, the Wireless Connectivity SDK provides implementations for the STM32L0 and STM32F4 platforms. In order to use all the features of the radio module with a different type of host microcontroller, the module’s drivers in the SDK have to be ported to the new custom platform.

In the following example, the steps required to port the drivers of the Bluetooth® LE 5.1 radio module Proteus-III to a custom platform are described.

- The directories WCON_Drivers/global and WCON_Drivers/ProteusIII have to be integrated into the custom project.
- The file global_types.h declares global types used by the drivers. It might be necessary to modify some parts of this file, such as port field of the WE_Pin_t structure.
- The file global.h declares the shared functions that deal with the serial interface as well as the usage of GPIOs for pin-related functions.

```c
/**
 * @brief Switch pin to output high/low
 * @param[in] pin Output pin to be set
 * @param[in] out Output level to be set
 * @return true if request succeeded, false otherwise
 */
extern bool WE_SetPin(WE_Pin_t pin, WE_Pin_Level_t out);

/**
 * @brief Initialize and start the UART.
 * @param[in] baudrate Baud rate of the serial interface
 * @param[in] flowControl Enable/disable flow control
 * @param[in] par Parity bit configuration
 * @param[in] dma Enables DMA for receiving data
 */
extern void WE_UART_Init(uint32_t baudrate, 
                         WE_FlowControl_t flowControl, 
                         WE_Parity_t par, 
                         bool dma);
```

Code 1: Code snippet of the file global.h

When integrating the Wireless Connectivity SDK into a custom host, the implementation of the functions defined in global.h must be done in the global.c file.

- Note that for reasons of clarity, part of the platform-specific functionality for the supported STM32 microcontroller families is implemented in the global_L0xx.c and global_F4xx.c files.
- The pins used by the module driver are defined in the driver’s main initialization function (here: ProteusIII_Init() in file ProteusIII.c). When integrating the Wireless Connectivity SDK into a custom host, this implementation may be adapted.

```c
ProteusIII_pins[ProteusIII_Pin_Reset].port = GPIOA;
ProteusIII_pins[ProteusIII_Pin_Reset].pin = GPIO_PIN_10;
```
ProteusIII_pins[ProteusIII_Pin_Reset].type = WE_Pin_Type_Output;
...
ProteusIII_pins[ProteusIII_Pin_Busy].port = GPIOB;
ProteusIII_pins[ProteusIII_Pin_Busy].pin = GPIO_PIN_8;
ProteusIII_pins[ProteusIII_Pin_Busy].type = WE_Pin_Type_Input;
...
if (false == WE_InitPins(ProteusIII_pins, ProteusIII_Pin_Count))
{
    /* error */
    return false;
}
WE_SetPin(ProteusIII_pins[ProteusIII_Pin_Boot], WE_Pin_Level_High);
WE_SetPin(ProteusIII_pins[ProteusIII_Pin_SleepWakeUp], WE_Pin_Level_High);

Code 2: Pin initialization in ProteusIII.c (snippet)

After dealing with these steps the driver is functional. The corresponding demo project can be considered as a basis for application development on the custom platform.
2.4 Running sample applications on the STM32 Nucleo board

2.4.1 Hardware connections

The Wireless Connectivity SDK for STM32 has been developed on the STM32 Nucleo-L073RZ [5] and Nucleo-F401RE [4] development boards using STM32CubeIDE [6]. To run the examples provided by the SDK on one of these boards, first connect the respective pins of the board to the radio module (typically mounted on the corresponding evaluation board). Besides the VCC, GND and the UART pins, other product-specific pins such as the Reset pin should be connected. Consult the Wireless Connectivity SDK readme file and/or the user manual of the used wireless module for additional information.

Please note that table 2 and figure 2 show a standard set of GPIOs used by most of the Würth Elektronik eiSos wireless modules. Depending on the used module and the application being developed, it may be necessary to make different connections.

<table>
<thead>
<tr>
<th>Function</th>
<th>Pin</th>
<th>I/O</th>
<th>Function</th>
<th>Pin</th>
<th>I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCC</td>
<td>+3V3</td>
<td>-</td>
<td>GND</td>
<td>GND</td>
<td>-</td>
</tr>
<tr>
<td>UART RX</td>
<td>PB7</td>
<td>Input</td>
<td>UART CTS</td>
<td>PA11</td>
<td>Input</td>
</tr>
<tr>
<td>UART TX</td>
<td>PB6</td>
<td>Output</td>
<td>UART RTS</td>
<td>PA12</td>
<td>Output</td>
</tr>
<tr>
<td>Reset</td>
<td>PA10</td>
<td>Output</td>
<td>Wake_up</td>
<td>PA9</td>
<td>Output</td>
</tr>
<tr>
<td>Boot</td>
<td>PA7</td>
<td>Output</td>
<td>Mode</td>
<td>PA8</td>
<td>Output</td>
</tr>
</tbody>
</table>

Table 2: Used pins of the STM32 Nucleo board

Figure 2: Layout for STM32 NUCLEO-L073RZ (source: UM1724 [3])
2.4.2 Run the project with STM32CubeIDE

First download the Wireless Connectivity SDK for STM32 [9] and save it on your computer. Then install and start the STM32CubeIDE [6] (recommended version 1.9.0 or higher). When starting, you will be asked for a workspace path. In case you already have a workspace from previous projects, select the desired path. In case you want to use a new workspace, create a new empty directory and select that directory.

![Figure 3: Choose workspace](image)

After the STM32CubeIDE started, go to "File→Open Projects from File System" and select the path where you saved the Wireless Connectivity SDK for STM32.

![Figure 4: Open project](image)

Select the project which you would like to import depending on the microcontroller family that you intend to use (e.g. STM32L0xx). Press the "Finish" button, so that the project is loaded into the "Project Explorer".

The Wireless Connectivity SDK for STM32 includes the examples and drivers for all supported products in one STM32CubeIDE project. Thus to evaluate a single product, the right radio module files have to be selected first.

In the example shown in figure 5 the radio module Tarvos-III is selected. In case Proteus-III shall be chosen instead of Tarvos-III, first right click on the Tarvos-III directory, then select "Properties" and check the box "Exclude resource from build". Then do the same for Proteus-III, but uncheck the "Exclude resource from build" box.
Then click on "Build" and check whether the project builds without errors. In case of success, click on "Debug" to run the project on the STM32 Nucleo board.
3 Wireless Connectivity SDK for Raspberry Pi

The Wireless Connectivity SDK for Raspberry Pi has been archived and will not be developed any further. Usage of the SDK for new products is discouraged. The final version of the SDK is 3.4.0.

3.1 Content

The radio modules supported by the latest version of the Wireless Connectivity SDK for Raspberry Pi are:

<table>
<thead>
<tr>
<th>SDK version</th>
<th>Radio standard</th>
<th>Radio module &amp; USB dongle</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4.0</td>
<td>Bluetooth® LE</td>
<td>Proteus-I, Proteus-II USB radio stick, Proteus-III, Proteus-III USB radio stick, Proteus-III-SPI</td>
</tr>
<tr>
<td></td>
<td>Proprietary 169 MHz</td>
<td>Titania</td>
</tr>
<tr>
<td></td>
<td>Proprietary 434 MHz</td>
<td>Thadeus</td>
</tr>
<tr>
<td></td>
<td>Proprietary 868 MHz</td>
<td>Tarvos-I, Tarvos-I USB radio stick, Tarvos-II, Tarvos-II USB radio stick, Tarvos-III, Tarvos-III USB radio stick, Thebe-I, Thebe-II</td>
</tr>
<tr>
<td></td>
<td>Proprietary 915 MHz</td>
<td>Telesto-I, Telesto-II, Telesto-III, Telesto-III USB radio stick, Themisto-I</td>
</tr>
<tr>
<td></td>
<td>Proprietary 2.4 GHz</td>
<td>Triton, Thalassa, Thalassa USB radio stick, Thyone-I, Thyone-I USB radio stick</td>
</tr>
<tr>
<td></td>
<td>Wi-Fi / WLAN</td>
<td>Calypso</td>
</tr>
<tr>
<td></td>
<td>Wireless M-BUS</td>
<td>Metis-II, Metis-II USB radio stick, Metis-I, Metis-I USB radio stick, Mimas-I, Mimas-I USB radio stick</td>
</tr>
</tbody>
</table>

Table 3: Wireless Connectivity SDK for Raspberry Pi

Besides the various sample projects, there is a directory named drivers. Within the driver directory, each supported radio module has its own directory containing the implementation of its command interface. Additionally to the definition of the commands, a thread is defined that checks for the confirmation and indication messages that are transmitted from the radio module to the host. Furthermore, functions that use specific radio module pins, like a pin reset or pin wake-up, are defined here.

Besides the module-specific directories, the subdirectory global contains all the shared functions as well as the definitions of the serial communication and GPIO interfaces of the underlying host.
The **global** directory contains various implementations of the serial and GPIO interface. Even though they are implemented with the Raspberry Pi platform in mind they can be used as a starting point for other platforms or even used on various Linux-based systems. Within the Code::Blocks projects the implementations can be selected by means of the build target.

<table>
<thead>
<tr>
<th>Build target</th>
<th>Interface</th>
<th>API/Library</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>UART</td>
<td>termios</td>
<td>For modules using UART</td>
</tr>
<tr>
<td>Release</td>
<td>GPIO</td>
<td>libgpiod</td>
<td>For modules using UART</td>
</tr>
<tr>
<td>DebugWiringPi</td>
<td>UART</td>
<td>wiringPi</td>
<td>For modules using UART</td>
</tr>
<tr>
<td>ReleaseWiringPi</td>
<td>GPIO</td>
<td>wiringPi</td>
<td>For modules using UART</td>
</tr>
<tr>
<td>Debug</td>
<td>SPI</td>
<td>spi_dev</td>
<td>For modules using SPI</td>
</tr>
<tr>
<td>Release</td>
<td>GPIO</td>
<td>libgpiod</td>
<td>For modules using SPI</td>
</tr>
<tr>
<td>Debug_WiringPi</td>
<td>SPI</td>
<td>spi_dev</td>
<td>For modules using SPI</td>
</tr>
<tr>
<td>Release_WiringPi</td>
<td>GPIO</td>
<td>wiringPi</td>
<td>For modules using SPI</td>
</tr>
<tr>
<td>Debug</td>
<td>UART</td>
<td>FTDI</td>
<td>For USB radio stick</td>
</tr>
<tr>
<td>Release</td>
<td>GPIO</td>
<td>None</td>
<td>For USB radio stick. Uses FTDI CBUS pins instead of GPIOs of the Raspberry Pi.</td>
</tr>
</tbody>
</table>

Table 4: Implementations of serial and GPIO interfaces in Wireless Connectivity SDK

WiringPi is deprecated by the author and no longer under development. WiringPi shall not be used anymore.

The definition of the serial and USB interfaces of the Raspberry Pi is included.

**3.2 Software history**

**Version 1.0.0** "Engineering"

- Initial version of the SDK

**Version 1.2.0** "Release"

- Added new products
- Updated driver structure to easily switch between serial and USB interface on Raspberry Pi
Version 1.6.3/2.0.0 "Release"
- Added new products

Version 3.0.0 "Release"
- Added driver for Wi-Fi module Calypso and proprietary high power radio module Thebe-I
- Replaced old module names by new module names

Version 3.1.0 "Release"
- Added driver for proprietary high power radio module Thebe-II and Themisto-I
- Bugfix in reset function of Proteus-* drivers
- Fixed typos in function names and resulting bug in Calypso driver

Version 3.2.0 "Release"
- Added driver for proprietary 2.4 GHz module Thyone-I and Thyone-I Plug
- Added driver for Bluetooth® LE module Proteus-III and Proteus-III Plug

Version 3.3.0 "Release"
- Added driver for Bluetooth® LE module Proteus-III-SPI
- Added implementation for SPI interfaces
- Added gpio implementation using libgpiod as wiringPi replacement

Version 3.4.0 "Release"
- Updated driver for proprietary 2.4 GHz module Thyone-I to add new features of latest module firmware
- Updated driver for Bluetooth® LE module Proteus-III add new features of latest module firmware
- This is the final version of the Wireless Connectivity SDK for Raspberry Pi. The SDK repository has been archived and the SDK will not be developed any further.

3.3 Host integration
As described in chapter 3.1, the Wireless Connectivity SDK provides an implementation for the Raspberry Pi platform. In order to use all the features of the radio module with a different type of host microcontroller, the module’s drivers in the SDK have to be ported to the new custom platform.

In the following example, the steps involved in porting the drivers of the Bluetooth® LE 4.2 radio module Proteus-I to a custom platform are described.

- The directories drivers/global and drivers/ProteusI as well as the file WE_common.h have to be integrated into the custom project.
In the function `InitDriver` of the file `ProteusI.c`, a thread is defined that listens to confirmation and indication messages that are transmitted from the attached radio module to the host processor.

Case 1: Threads are supported, the `rxthread` has to be ported to the host's thread system.
Case 2: Threads are not supported, a state machine has to be created that periodically checks for incoming UART bytes.

```c
void *rx_thread()
{
    while(1)
    {
        /* wait for 1ms, then check if new RX data is available */
        delay (1);
        while (BytesAvailable())
        {
            /* interpret received byte */
            if(ReadByte(&readBuffer))
            {
                ...
            }
        }
    }
}
```

Code 3: Code snippet of the rxthread

The file `global.h` declares the shared functions that deal with the serial interface as well as the usage of GPIOs for pin related functions.

```c
/* Switch pin to input/output high/low with/without pullup/pulldown
 * input:
 * - pin_number: number of pin
 * - inout: input or output
 * - pull: pullup, pulldown or no pull
 * - out: output level high or low
 * return: true, if success
 * false, otherwise
 */
extern bool SetPin( int pin_number, SetPin_InputOutput_t inout, SetPin_Pull_t pull, SetPin_Out_t out);

/*
 * Open the serial interface
 * input:
 * - baudrate: baudrate of the interface
 * return: true, if success
 * false, otherwise
 */
extern bool OpenSerial(int baudrate);
```

Code 4: Code snippet of the file global.h

Here the definition of these functions, depending on the custom host peripherals, has to be created by the user. The existing files `global_serial.c`, `global_ftdi.c` and `global.c` can be removed from the project as it contains the corresponding implementation for the Raspberry Pi.

After dealing with the `rxthread` and the definition of the functions declared in `global.h`, the driver is functional. The corresponding demo project can be considered as a basis for application development on the custom platform.
3.4 Running sample applications on the Raspberry Pi

3.4.1 Hardware connections

For creating custom applications on the basis of the Raspberry Pi, connect the pins of the module to corresponding pins on the Raspberry Pi (power supply, ground, serial/SPI interface and other pins like reset). If the SPI version of a module is used, the driver assumes that SPI0 CE0 is used as chip select pin. Please refer to figure 7 to get an overview of the pins of the Raspberry Pi used in the driver application examples.

![GPIO Raspberry Pi 3B](image)

Alternatively, Würth Elektronik eiSos USB dongles such as the Tarvos-II USB radio stick & Tarvos-III USB radio stick and evaluation boards can be directly connected to the USB-interface of the Raspberry Pi. If a USB port is to be used, the D2xx driver from FTDI has to be installed first (see section 3.4.4.2).
Some products, such as the Tarvos-II USB radio stick, operate in transparent mode by default. In that mode, the device does not interpret the commands sent by the Wireless Connectivity SDK and thus communication fails. In order to use the module with the SDK, first set the default operation mode to command mode, for example using the Würth Elektronik eiSos software tool ACC.

3.4.2 Install the Raspberry Pi OS on the Raspberry Pi

1. Install the Raspberry Pi OS to a SD card according to official documentation [2]. It is recommended to use the Raspberry Pi Imager.

2. After installing the image on the SD card, insert it into the Raspberry Pi’s SD card slot, connect your monitor, mouse and keyboard. Now the Raspberry Pi is ready to boot up. Please start it by powering it up.

3. After booting the Raspberry Pi, switch off the Bluetooth® interface by clicking on the Bluetooth® button on the right upper corner of the screen (see figure 8). Otherwise it is not possible to use the UART interface of the Raspberry Pi.

4. Then turn on the WiFi for connecting to the internet by clicking on the WiFi button on the right upper corner of the screen and selecting the WiFi of your choice.

![Figure 8: Switch off the Bluetooth® and connect to internet via WiFi](image)

5. After connecting to the internet make sure your Raspberry Pi is up to date with the latest versions of Raspberry Pi OS. To update the system open a terminal by clicking on the terminal symbol in the left upper corner (see figure 9).

6. Then upgrade the Raspberry Pi OS by typing in terminal:

```
sudo apt-get update
sudo apt-get upgrade
```
### 3.4.3 Configuring the peripherals

1. Next, the peripherals have to be enabled. To do so open the menu by clicking on the Raspberry Pi symbol on the left upper corner of the screen and open the Preferences → Raspberry Pi Configuration window (see figure 10). Enable the SPI or serial depending on the required interface.

![Raspberry Pi interface configuration](image1)

2. After enabling the interfaces a dialog should appear asking for a reboot to apply the changes. If no dialog appears reboot by clicking on the Raspberry symbol on the left upper corner of the screen and select **Shutdown**.

3. Now, after enabling the serial interface, the Raspbian OS claims it for console output. To disable this feature, please remove the string "console = serial0,115200" from the file `/boot/cmdline.txt` and save it. Root privilege is needed to change the file. To open the file accordingly type in terminal:

    ```bash
    sudo leafpad /boot/cmdline.txt
    ```
4. Please check whether the serial interface is still enabled by opening the file `/boot/config.txt` and check whether the string "enable_uart=1" is still included. If not, please add it and save the file. Root privilege is needed to change the file. To open the file accordingly type in terminal:

```
sudo leafpad /boot/config.txt
```

5. Please reboot as before and check whether the files `/boot/cmdline.txt` and `/boot/config.txt` are still as described in the previous two points. If not, adapt the two files again as described before and reboot. Otherwise the UART-interface to the module is not active and thus the module communication fails.

6. In order to use the peripherals as a non-root user, the local user has to be a member of the peripheral group. In order to check this, type in the following in the terminal,

```
groups
```

If the output contains GPIO and, if required, SPI, then skip the next step.

7. Add the current user to the groups by typing in the following commands in the terminal,

```
sudo adduser pi gpio
sudo adduser pi spi
```

Logout and login to update the user group settings.

### 3.4.4 Install the required libraries

By default termios is used to communicate with the UART interface of the Raspberry Pi. This is installed by default and no further action is required. The same is true for the SPI interface and the usage of the spidev API.

The library to control the GPIO pins has to be installed however.

#### 3.4.4.1 Install the libgpiod library

To access the GPIO pins `libgpiod` is used. It can be installed by typing the following in the terminal:

```
sudo apt-get update
sudo apt-get install libgpiod-dev libgpiod-dev libgpiod-doc
```

Note that as of now Raspberry Pi OS is based on Debian Buster and installs libgpiod version 1.2. This version of libgpiod does not support Pull-up and Pull-Down for GPIO pins yet. This feature was introduced with version 1.5. For the sample applications using modules with UART interface, libgpiod version 1.2 is sufficient.

For modules using the SPI interface however, Pull-up and Pull-Down is required and libgpiod has to be build and installed manually. See libgpiod documentation:

https://git.kernel.org/pub/scm/libs/libgpiod/libgpiod.git/about/
3.4.4.2 Install the FTDI driver for USB (optional)

To run a Würth Elektronik eiSos USB dongle such as the Tarvos-II USB radio stick on the Raspberry Pi, the FTDI library is used and has to be installed.

To do so, first download the latest D2xx Linux driver [1] for the ARM hard-float architecture. There are three versions available of the the FTDI D2xx driver which support the Raspberry Pi. To find out which architecture is used by your Raspberry Pi and Raspberry Pi OS, type the following in the terminal:

```
lsCPU
```

The command will print information about the CPU. For output "Architecture: armv71" use the v7 version of the D2xx driver.

To install the FTDI driver the following commands have to be run in the terminal. It is assumed that the driver is version 1.4.22 for armv7 and saved to the ~/Downloads directory.

1. Go to the directory where the driver is saved
   ```
   cd ~/Downloads
   ```

2. Unpack the gzip file
   ```
   tar -xvf libftd2xx-arm-v7-hf-1.4.22.tgz
   ```

3. Copy the needed header files to the system folder /usr/local/include
   ```
   sudo cp release/ftd2xx.h /usr/local/include
   sudo cp release/WinTypes.h /usr/local/include
   ```

4. Copy the libraries to the system folder /usr/local/lib
   ```
   sudo cp release/build/lib*/usr/local/lib
   ```

5. Create a symbolic link to the latest version
   ```
   sudo ln -sf /usr/local/lib/libftd2xx.so.1.4.22 /usr/local/lib/libftd2xx.so
   ```

6. Update the shared libraries
   ```
   sudo ldconfig
   ```

7. As the D2xx driver is incompatible with the FTDI VCP driver in the Linux kernel, the kernel modules "ftdi_sio" and "usbserial" have to be unloaded. To do so, please run:
   ```
   sudo modprobe -r ftdi_sio
   sudo modprobe -r usbserial
   ```

8. Now all supported USB dongles can be used.
3.4.5 Install the Wireless Connectivity SDK

The Wireless Connectivity SDK was developed in the Code::Blocks development environment.

1. First download and install the software Code::Blocks. Open a terminal and type:

   ```
   sudo apt-get install codeblocks
   ```


3. The file is going to be extracted to the folder ~/Projects. If the folder does not exist create it by typing in terminal:

   ```
   mkdir ~/Projects
   ```

4. Now extract the Wireless Connectivity SDK to ~/Projects by typing in terminal:

   ```
   unzip ~/Downloads/WirelessConnectivity-SDK.zip -d ~/Projects
   ```

5. Then start the desired project via Code::Blocks by typing in terminal. For example,

   ```
   codeblocks ~/Projects/WirelessConnectivity-SDK/Example_ProteusIII/ProteusIII.cbp &
   ```
6. Select the build target using the drop-down menu, e.g. Debug.

![Select build target](image1.png)

Figure 11: Select build target of the application

7. Then press **Build → Rebuild** to build the project (see figure 12).

![Rebuild application](image2.png)

Figure 12: Rebuild the application

8. If it builds without errors the setup succeeded.

9. In case of further question, please contact our technical support [7].
3.4.6 FAQ - Frequently asked questions

3.4.6.1 The initialization function fails, what can I do?

The initialization function usually sets up the serial interface, performs a pin reset and waits for the module’s response. In case this fails, there are several possibilities:

- The module is not powered up. Please check the VCC and GND connection.
- The RESET line is not connected, thus no pin reset was applied.
- The UART RX and UART TX lines are not connected, thus the module response was not transmitted.
- The UART interface does not run well. Please check the UART settings and initialization.
- The connected module or USB dongle does not run in command mode and thus does not respond to a pin reset and/or command request. In this case, set the device to command mode, for example by using the software tool ACC.
- The rxthread function, waiting for module response, does not work correctly.

3.4.6.2 ProteusIII-SPI: Pin Wake-up leads to "GetPin: Could not read pin level"

When using the ProteusIII-SPI, the same pin is used to wake-up the module and to check for new messages. So there is a chance that the pin is accessed at the same time by different functions within the application. Mostly this will have no effect and the wake-up will succeed. If it leads to errors, the problem can be fixed by using a mutex.
4 References


5 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

1 Wireless connectivity SDK driver as part of the end product ........................................... 6
2 Layout for STM32 NUCLEO-L073RZ (source: UM1724 [3]) ............................................. 13
3 Choose workspace .................................................................................................................. 14
4 Open project ........................................................................................................................... 14
5 Select module ........................................................................................................................ 15
6 Build and run .......................................................................................................................... 15
7 Extended connector from Raspberry Pi .................................................................................. 21
8 Switch off the Bluetooth® and connect to internet via WiFi .................................................. 22
9 Terminal button ....................................................................................................................... 23
10 Raspberry Pi interface configuration ..................................................................................... 23
11 Select build target of the application ...................................................................................... 27
12 Rebuild the application .......................................................................................................... 27

List of Tables

1 Wireless Connectivity SDK for STM32 ................................................................................. 9
2 Used pins of the STM32 Nucleo board .................................................................................. 13
3 Wireless Connectivity SDK for Raspberry Pi ....................................................................... 16
4 Implementations of serial and GPIO interfaces in Wireless Connectivity SDK .................. 18
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