Your Global Automation Partner

TBEN-L…-4RFID…-LNX
Compact RFID Interface

Instructions for Use

Contents

1 About these Instructions

These operating instructions describe the structure, functions and the use of the product and will help you to operate the product as intended. Read these instructions carefully before using the product. This is to avoid possible damage to persons, property or the device. Retain the instructions for future use during the service life of the product. If the product is passed on, pass on these instructions as well.

1.1 Target groups

These instructions are aimed a qualified personal and must be carefully read by anyone mounting, commissioning, operating, maintaining, dismantling or disposing of the device.

1.2 Explanation of symbols used

The following symbols are used in these instructions:

⚠	**DANGER** DANGER indicates a dangerous situation with high risk of death or severe injury if not avoided.
⚠	**WARNING** WARNING indicates a dangerous situation with medium risk of death or severe injury if not avoided.
⚠	**CAUTION** CAUTION indicates a dangerous situation of medium risk which may result in minor or moderate injury if not avoided.
!	**NOTICE** NOTICE indicates a situation which may lead to property damage if not avoided.
ℹ	**NOTE** NOTE indicates tips, recommendations and useful information on specific actions and facts. The notes simplify your work and help you to avoid additional work.
▶	**CALL TO ACTION** This symbol denotes actions that the user must carry out.
⇨	**RESULTS OF ACTION** This symbol denotes relevant results of actions.

1.3 Other documents

Besides this document the following material can be found on the Internet at www.turck.com:
- Data sheet
- EC Declaration of Conformity

1.4 Feedback about these instructions

We make every effort to ensure that these instructions are as informative and as clear as possible. If you have any suggestions for improving the design or if some information is missing in the document, please send your suggestions to techdoc@turck.com.

2 Notes on the Product

2.1 Product identification

These instructions apply to the following compact RFID interfaces:

- TBEN-L4-4RFID-8DXP-LNX
- TBEN-L5-4RFID-8DXP-LNX

2.2 Scope of delivery

- Compact RFID interface
- Closure caps for M12 connectors
- Quick Start Guide

2.3 Legal requirements

The device is subject to the following EC directives:

- 2014/30/EU (electromagnetic compatibility)
- 2011/65/EU (RoHS II Directive)

2.4 Manufacturer and service

Hans Turck GmbH & Co. KG
Witzlebenstraße 7
45472 Muelheim an der Ruhr
Germany

Turck supports you with your projects, from initial analysis to the commissioning of your application. The Turck product database contains software tools for programming, configuration or commissioning, data sheets and CAD files in numerous export formats. You can access the product database at the following address: **www.turck.en/products**
For further inquiries in Germany contact the Sales and Service Team on:

- Sales: +49 208 4952-380
- Technology: +49 208 4952-390

Outside Germany, please contact your local Turck representative.

3 For Your Safety

The product is designed according to state-of-the-art technology. However, residual risks still exist. Observe the following warnings and safety notices to prevent damage to persons and property. Turck accepts no liability for damage caused by failure to observe these warning and safety notices.

3.1 Intended use

The devices are intended for use in the industrial sector.
The TBEN-L…-4RFID-8DXP-LNX... block module is a programmable RFID interface for use in the Turck RFID system. The Turck RFID system is used for the contactless exchange of data between a tag and a read/write head in object or product identification applications. Four RFID channels are provided for connecting BL ident® read/write heads. Eight configurable digital channels are also provided. The interfaces communicate via TCP/IP with third party systems such as ERP systems.
The devices may only be used as described in these instructions. Any other use is not in accordance with the intended use; Turck accepts no liability for any resulting damage.

3.2 General Safety Notes

- The device may only be assembled, installed, operated, parameterized and maintained by professionally-trained personnel.
- The device may only be used in accordance with applicable national and international regulations, standards and laws.
- The device only meets the EMC requirements for industrial areas and is not suitable for use in residential areas.

4 Product Description

The devices are designed with a fully encapsulated housing with degree of protection IP67/ IP69K. Four RFID channels are provided for connecting read/write heads. It is also possible to connect sensors and actuators via eight digital I/O channels, which can be configured as inputs or outputs as required. The terminals for the read/write heads and for digital I/Os are M12 sockets. An M12 socket is provided for the Ethernet connection. The plug connectors are 4-pin (TBEN-L4) or 5-pin (TBEN-L5) 7/8″ female connectors.

4.1 Device overview

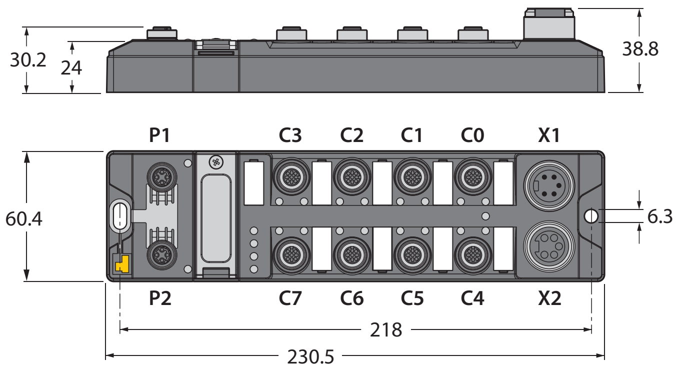


Fig. 1: Dimensions

4.1.1 Operating elements

The devices are provided with the following operating elements:

- Rotary coding switches and DIP switch for setting the IP address
- SET button for activating the write accesses of the USB Host port functions

4.2 Properties and features

- TCP/IP
- Freely programmable compact module based on Linux
- Programming languages C, C++, NodeJS, Python
- API and SDK available on request
- Implementation of the protocol for the read/write heads required
- 4 channels with M12 connector for RFID
- 8 configurable digital channels as 2 A pnp inputs and/or outputs
- Multiple LEDs for status display
- Integrated Ethernet switch enables line topology
- 10 Mbps/100 Mbps transfer rate
- Glass fiber reinforced housing
- Shock and vibration proof
- Fully encapsulated module electronics
- Degree of protection IP65/IP67/IP69K

4.3 Operating principle

The RFID interfaces connect the RFID system with other systems that communicate via TCP/IP (e.g. ERP systems). The interfaces are provide with an Ethernet interface and RFID interfaces.

The RFID system can be linked to a third-party system, such as an ERP system, via the TCP/IP interface. The read/write heads are connected to the interfaces via the RFID interfaces. The interfaces can also process sensor and actuator signals via 8 configurable digital channels.

4.4 Functions and operating modes

HF and UHF read/write heads can be connected to the RFID channels. Parallel operation of HF and UHF read/write heads on the same device is also possible.
Sensors and actuators can be connected to the configurable digital channels. Up to four 3-wire PNP sensors or four PNP DC actuators with a maximum output current of 2 A per output can be connected. The Linux operating system enables the device functions to be programmed with C, C++, NodeJS or Python. It is also possible to integrate middleware functions on the device.

4.4.1 Linux distribution – Software components

The Linux distribution of the device contains the following software components:

- SSH
- SFTP
- HTTP
- IBTP
- MTXP
- DHCP
- SNTP
- Node.js 6.9.5 (LTS)
- Python 3.x

4.5 USB Host Port

The device is provided with a USB Host port for connecting USB memory sticks. The USB Host port is a USB2.0 A socket. The firmware of the devices can be updated via the USB interface. Memory expansion via the USB Host port is not possible.

4.6 Technical Accessories

Accessories for mounting, connecting and parameterizing can be found in product database or the Accessories List for TBEN (D301367) under **www.turck.com**. The accessories are not part of the scope of delivery.

5 Mounting

The device must be attached to a level, pre-drilled and grounded mounting surface.

- ▶ Attach the module to the mounting surface with two M6 screws. The maximum tightening torque for the screws is 1.5 Nm.

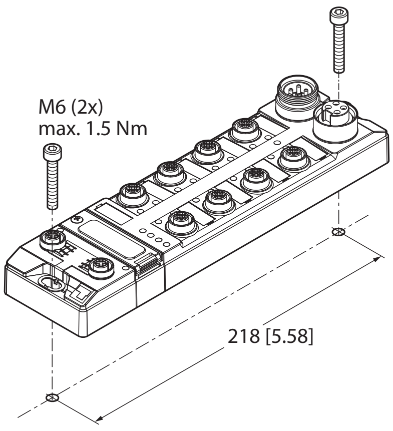


Fig. 2: Fixing the device to the mounting plate

5.1 Mounting the device outdoors

The device is UV-resistant according to DIN EN ISO 4892-2. Direct sunlight can cause material abrasion and color changes. The mechanical and electrical properties of the device are not affected.

- ▶ To avoid material abrasion and color changes: Protect the device from direct sunlight, e.g. by using protective shields.

5.2 Grounding the device

5.2.1 Grounding and shielding concept

The grounding and shielding concept of the TBEN-L modules enables the separate grounding of the fieldbus and I/O section.

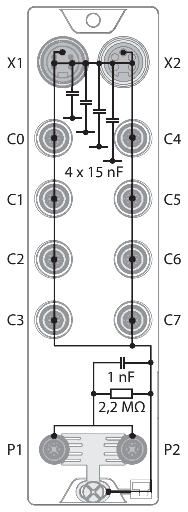


Fig. 3: Equivalent circuit, shielding concept

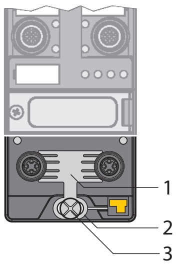


Fig. 4: Grounding components

The grounding strip (1) on the M12 plug connectors for the fieldbus connection (P1, P2) connects the shield of the fieldbus cables. The metal ring (2) is fitted underneath the grounding strip and connects the functional ground of the 7/8″ plug connectors (Pin 3) for the power supply with the functional ground of the M12 plug connectors (Pin 5) for connecting the read/write heads, sensors and actuators. A metal screw (3) connects the device with the reference potential of the system.

5.2.2 Grounding the device (FG)

Grounding strip and metal ring are connected to each other. A fixing screw through the bottom mounting hole of the module connects the shield of the fieldbus cables with the functional ground of the power supply and connected devices as well as the reference potential of the system. If a common reference potential is not required, remove the grounding clip to disconnect the fieldbus shield or fasten the module with a plastic screw.

Removing the grounding clip

▶ Lever up the grounding strip with a flat slot-head screwdriver and remove.

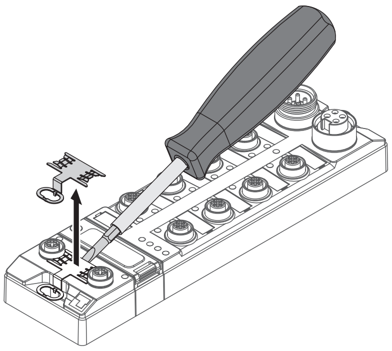


Fig. 5: Removing the grounding clip

Mounting the grounding clip

▶ Use a screwdriver to insert the grounding clip between the fieldbus connectors so that contact is made with the metal housing of the plug connectors.

➪ The shield of the fieldbus cables is connected to the grounding clip.

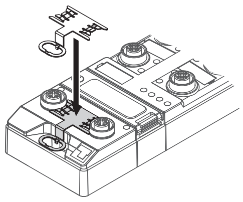


Fig. 6: Mounting the grounding clip

6 Connection

6.1 Connecting the modules to Ethernet

The connection to Ethernet is realized via the integrated auto-crossing switch is done using two 4-pole, D-coded M12 x 1-Ethernet-connectors. The maximum tightening torque is 0.6 Nm.

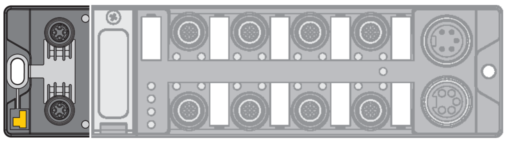
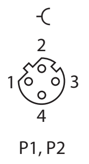


Fig. 7: M12 Ethernet connector for the connection to Ethernet

▶ Connect the device to Ethernet according to the pin assignment below.



1 = TX +
2 = RX +
3 = TX –
4 = RX –
flange = FE

Fig. 8: Pin assignment Ethernet connectors

6.2 Connecting the power supply

For the connection to the power supply, the device has two 5-pin 7/8" connectors. The power supply connectors are designed as 4-pole (TBEN-L4) or 5-pole (TBEN-L5) 7/8" connectors. V1 and V2 are galvanically isolated. The maximum tightening torque is 0.8 Nm.

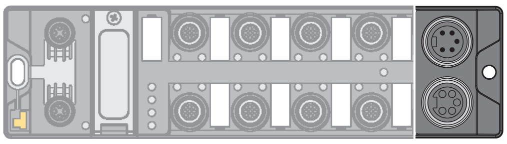


Fig. 9: 7/8'' connector for connecting the supply voltage

▶ Connect the device to the voltage supply according to the pin assignment below.

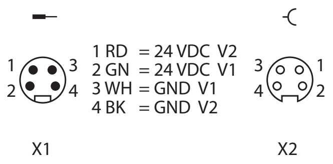


Fig. 10: TBEN-L4-… – Pin assignment power supply connectors

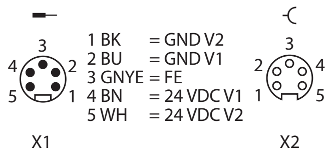


Fig. 11: TBEN-L5-… – Pin assignment power supply connectors

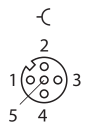
Connector	Function
X1	Power feed
X2	Continuation of the power to the next node
V1	System supply: Power supply 1 (incl. supply of electronics)
V2	Load voltage: Power supply 2

NOTE
The system voltage (V1) and the load voltage (V2) are supplied and monitored separately. In case of an undercut of the admissible voltage, the connectors are switched-off according to the module's supply concept. In case of an undervoltage at V2, the LED PWR changes from green to red. In case of an undervoltage at V1, the LED PWR is turned off.

6.3 Connecting RFID read/write heads

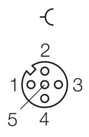
The device has four 5-pin M12 plug connectors for connecting RFID read/write heads. The maximum tightening torque is 0.8 Nm.

- ▶ Connect the read/write heads to the device as per the pin layout shown below.



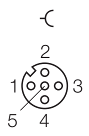
1 = $V_{aux}1$
2 = Data B
3 = GND V1
4 = Data A
5 = FE/Shield

Fig. 12: RS485 – Pin layout of the read/write head connections



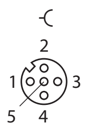
1 = BN (+)
2 = BK (Data)
3 = BU (GND)
4 = WH (Data)
5 = shield

Fig. 13: …/S2500 connection cables – Pin layout of the read/write head connections



1 = BN (+)
2 = WH (Data)
3 = BU (GND)
4 = BK (Data)
5 = shield

Fig. 14: …/S2501 connection cables – Pin layout of the read/write head connections



1 = RD (+)
2 = BU (Data)
3 = BK (GND)
4 = WH (Data)
5 = shield

Fig. 15: …/S2503 connection cables – Pin layout of the read/write head connections

6.4 Connecting digital sensors and actuators

The device has four 5-pin M12 plug connectors for connecting digital sensors and actuators. The maximum tightening torque is 0.8 Nm.

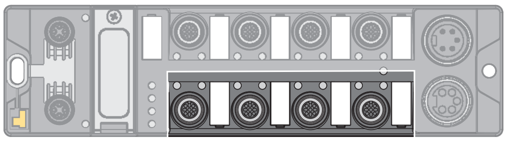
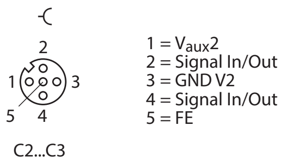


Fig. 16: M12 plug connector for connecting digital sensors and actuators

▶ Connect the sensors and actuators to the device as per the pin layout below.



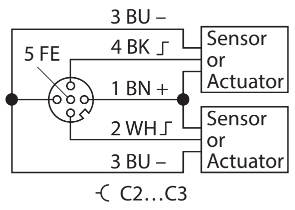


Fig. 17: Connections for digital sensors and actuators – Pin layout

Fig. 18: Connections for digital sensors and actuators – Wiring diagram

The channels are assigned to the slots as follows:

Channel	Slot	Pin
DXP8 Ch8	C4	4
DXP9 Ch9	C4	2
DXP10 Ch10	C5	4
DXP11 Ch11	C5	2
DXP12 Ch12	C6	4
DXP13 Ch13	C6	2
DXP14 Ch14	C7	4
DXP15 Ch15	C7	2

7 Commissioning

The Linux operating system enables the device functions to be programmed with C, C++, NodeJS or Python.
Additional software tools (e.g. PuTTY) are required to access the device via the console. A file exchange between the device and a PC can be used, e.g. via WinSCP. The following login data is stored on the device by default:
User: user
Password: password

NOTE
The read/write head protocol is not implemented by default. The protocol must be implemented by the user.

7.1 Setting the IP address

The IP address can be set via two decimal rotary coding switches and DIP switches on the device, via the web server or via the Turck Service Tool.

7.1.1 Setting the IP address via switches at the device

The IP address can be set via two decimal rotary coding switches and the DIP switch "Mode" on the device. The switches are located under a cover together with the USB ports and the SET button.

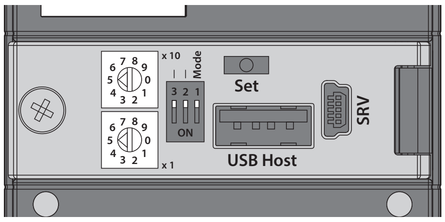


Fig. 19: Switches for setting the IP address

- ▶ Open the cover above the switches.
- ▶ Set the rotary coding switch to the desired position according to the table below.
- ▶ Set DIP switch "Mode" to the desired position according to the table below.
- ▶ Execute a power cycle.
- ▶ NOTICE! When the cover over the rotary coding switches is open, protection class IP67 or IP69K is not guaranteed. Damage to the device due to invasive foreign material or liquids Tightly close the cover above the switches.

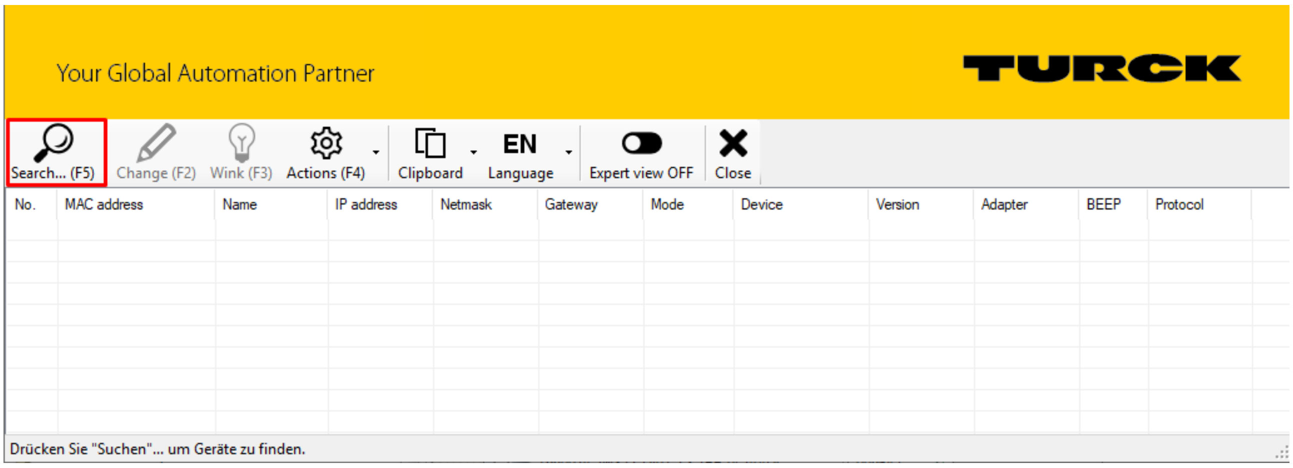
Addressing options

The IP address of the devices can be set in different ways. The following addressing options can be selected via the switches on the device. Changes to the settings become active after a voltage reset.

Setting option	**DIP switch "MODE"**	**Rotary coding switches**	**Description**
Default address	0	00	IP address: 192.168.1.100 Subnet mask: 255.255.255.0 Gateway: 192.168.1.1
Rotary mode	0	1…99	In rotary mode, the last byte of the IP address can be set manually at the gateway. The other network settings are stored in the non-volatile memory of the gateway and cannot be changed in rotary mode. Addresses from 1...99 can be set.
DHCP mode	1	40	In DHCP mode, the IP address is automatically assigned by a DHCP server in the network. The subnet mask assigned by the DHCP server and the default gateway address are stored non-volatile in the memory of the gateway DHCP supports 3 types of IP address assignment: ■ Automatic address assignment: The DHCP server assigns a permanent IP address to the client. ■ Dynamic address assignment: The IP address assigned by the server is only reserved for a certain period of time. After this time has elapsed or after the explicit release by a client, the IP address is reassigned. ■ Manual address assignment: A network administrator assigns an IP address to the client. In this case, DHCP is only used to transmit the assigned IP address to the client.
PGM Mode	1	50	In PGM mode, the complete IP address is assigned manually via the Turck Service Tool, FDT/DTM or via a web server. In PGM mode, the set IP address and the subnet mask are stored in the memory of the gateway. All network settings (IP address, subnet mask, default gateway) are taken from the internal EEPROM of the module.
PGM DHCP mode	1	60	In PGM-DHCP mode, the gateway sends DHCP requests until it is assigned a fixed IP address. The DHCP client is automatically deactivated if an IP address is assigned to the gateway via the DTM or a web server.
F_Reset	1	90	The F_Reset mode sets all device-settings back to the default values and deletes all data in the device's internal flash. The following values are reset or deleted: ■ IP address and subnet mask ■ Parameters
Restore	1	00	■ IP address: 192.168.1.100

7.1.2 Setting the IP address via the Turck Service Tool

- Connect the device to a PC via the Ethernet interface.
- Launch the Turck Service Tool.
- Click **Search** or press F5.

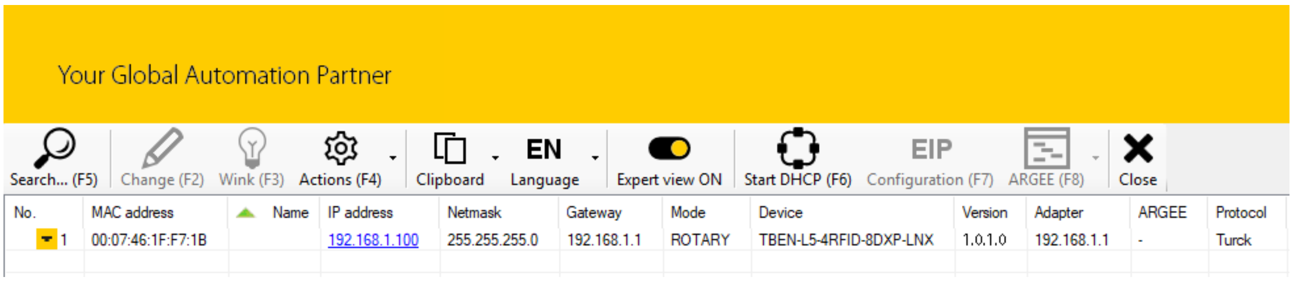


No.	MAC address	Name	IP address	Netmask	Gateway	Mode	Device	Version	Adapter	BEEP	Protocol

Drücken Sie "Suchen"... um Geräte zu finden.

Fig. 20: Turck Service Tool – Start screen

The Turck Service Tool displays the connected devices.



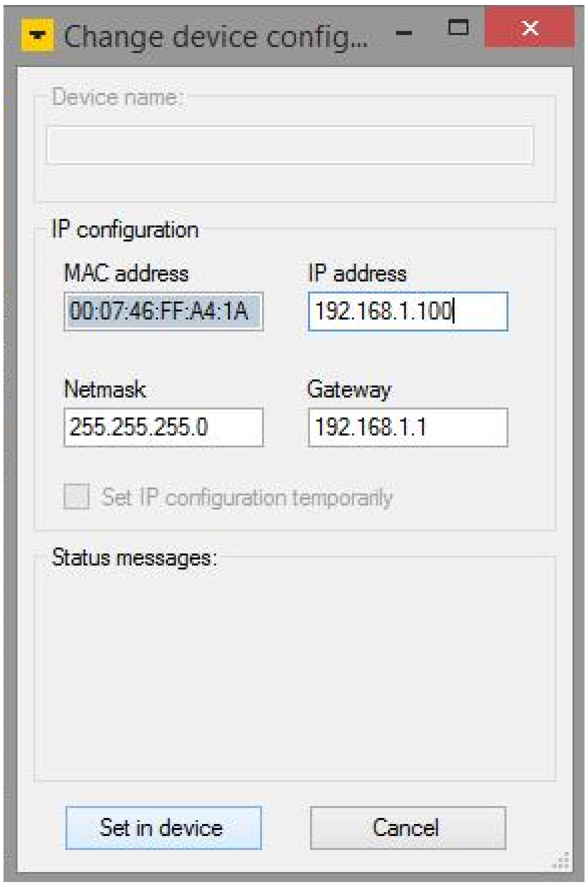
No.	MAC address	Name	IP address	Netmask	Gateway	Mode	Device	Version	Adapter	ARGEE	Protocol
1	00:07:46:1F:F7:1B		192.168.1.100	255.255.255.0	192.168.1.1	ROTARY	TBEN-L5-4RFID-8DXP-LNX	1.0.1.0	192.168.1.1	-	Turck

Fig. 21: Turck Service Tool – Found devices

- Click the required device.
- Click **Change** or press [F2].

NOTE
Clicking the IP address of the device opens the web server.

- Change the IP address and if necessary the network mask and gateway.
- Accept the changes by clicking **Set in device**.



Device name:

IP configuration

MAC address: 00:07:46:FF:A4:1A

IP address: 192.168.1.100

Netmask: 255.255.255.0

Gateway: 192.168.1.1

Set IP configuration temporarily

Status messages:

Fig. 22: Turck Service Tool – Changing the device configuration

7.2 Programming RFID channels

The RFID channels of the device are designed as RS485 serial interfaces. The interface operates in half-duplex mode. The direction of the control signal must be adjusted to switch between send and receive.

An external peripheral device is integrated in the RFID interface in order to communicate with connected devices. A separate slave controller is used for handling the messages exchanged between the interface and the read/write head. The slave controller must be adapted to the Linux settings via a script.

The serial interfaces are as follows:

COM interface	TTY	Channel
COM0	tty03	Ident 0
COM1	tty04	Ident 1
COM2	tty01	Ident 2
COM3	tty02	Ident 3

7.2.1 GPIOs of the RFID channels – Overview

The RFID channels can be programmed via the GPIOs. The GPIOs are located under the following path:
/sys/class/gpio/…

NOTE
When biasing is switched on, the polarity must also be set. For this use the GPIOs for the polarity change.
Polarity change and biasing must be permitted in the Linux settings.

Function	COM	GPIO	Possible values
Switch power/AUX	COM0	gpio493	0: Switch off 1: Switch on
	COM1	gpio440	
	COM2	gpio441	
	COM3	gpio442	
Biasing A-GND_B-5V_1_1 invertieren	COM0_I	gpio461	0: Switch off 1: Switch on
	COM1_I	gpio469	
	COM2_I	gpio450	
	COM3_I	gpio453	
Biasing A-5V_B-GND_1_1 in Normalmodus setzen	COM0_N	gpio462	0: Switch off 1: Switch on
	COM1_N	gpio470	
	COM2_N	gpio449	
	COM3_N	gpio454	
Polarity change (RxD → TxD)	COM0	gpio456	0: normal 1: Inverse
	COM1	gpio464	
	COM2	gpio447	
	COM3	gpio451	
Switch on RS485 bus terminating resistor	COM0	gpio460	0: Switch off 1: Switch on
	COM1	gpio468	
	COM2	gpio448	
	COM3	gpio452	

7.2.2 Adapt slave controller via script

A script is installed on the device for adapting the slave controller. The script is located under the following path:
/TURCK/scripts/serial.sh

The script can be used with the following syntax:
```
sh serial.sh device cmd [param]
```

Example of using the script:
```
sh serial.sh COM0 send "Hello World!"
```

The following parameter values can be used:

cmd	param
baud	Transfer rate, e.g. 9600
term	ON/OFF
bias	ON/OFF
swap	ON/OFF
send	Message
recv	
vaux	ON/OFF
sethw	
sets	

Script sh serial.sh – Overview of the commands

Command	Function
sh serial.sh COMx baud baudrate	Sets the transfer rate of the Linux system and the slave controller.
sh serial.sh COMx term ON/OFF	Switches the RS485 terminating resistor on or off.
sh serial.sh COMx bias ON/OFF	Switches the biasing on or off.
sh serial.sh COMx swap ON/OFF	Switches the swapping function on or off.
sh serial.sh COMx send "message"	Sends a data string to a connected read/write head.
sh serial.sh COMx recv	Receives messages from a connected read/ write head.
sh serial.sh COMx vaux ON/OFF	Switches the auxiliary voltage VAUX on or off.
sh serial.sh COMx sethw	Adapt slave controller to Linux settings
sh serial.sh COMx seths [baudrate]k[databits][parity][stopbits]	Adapt settings for slave controller and Linux settings

Example: Adapt settings for slave controller and Linux system via script

The settings for the slave controller and the Linux system can be adapted via the sets function of the script. The function can be used with the following syntax:

```
sh serial.sh COMx sets (Baudrate)k(databits)(parity)(stopbits)
```

Example: Set transfer rate of 115200 kbaud, 8 bit, without parity, 1 stop bit

```
serial.sh COM1 115.200k8n1
```

The possible values for the individual parameters are shown in the following table:

Transfer rate		**Bits**		**Parity**		**Stop bit**	
Parameter	**Meaning**	**Parameter**	**Meaning**	**Parameter**	**Meaning**	**Parameter**	**Meaning**
9600	9600 kBaud	5	cs5	n	None	1	1 stop bit
38400	38400 kBaud	6	cs6	e	Even	2	2 stop bits
115200	115200 kBaud	7	cs7	o	Odd		
		8	cs8				

7.2.3 Programming RFID channels with Python 3

The following examples illustrate the programming of the RFID interface with Python 3.

Example 1: Using the “pySerial” module

```
import serial # from module pySerial
from os import system as sh # for use of the sh-script

# open serial interface on port 0 and set a timout of 8 seconds
seri = serial.Serial("/dev/COM0", timeout=8)

# change settings
seri.baudrate = 115200 # set the baudrate of port COM0 to 115200
seri.parity ='N' # set no parity for port COM0
seri.bytesize = 7 # set the byte size for a sign to 7 for port
COM0
seri.stopbits = 1 # set stopbits to 1 for port COM0

sh('/TURCK/scripts/serial.sh COM0 sethw')

seri.write(bytearray.fromhex("aa 07 07 49 00 41 23")) # writes a
bytestream

print(seri.readline()) # reads incoming message as ascii
```

Example 2: Using the “periphery” module

```
from periphery import Serial
from os import system as sh # for use of the sh-script

# Open /dev/COM1 with baudrate 115200, and defaults of 8N1, no
flow control
serial = Serial("/dev/COM1", 115200)

# write a bytestream serial.write(bytearray.fromhex("aa 07 07 49
00 41 23"))

# Read up to 128 bytes with 500ms timeout
buf = serial.read(128, 0.5)

print(buf)
print("read %d bytes: _%s_" % (len(buf), buf))

serial.close()
```

7.2.4 Programming RFID channels with Node.js

The following examples illustrate the programming of the RFID interface with Node.js. Further information on Node.js and the Node.js packages is provided at:

- https://nodejs.org
- https://www.npmjs.com/

```
# initialize the serialport-v5 box
var SerialPort=require('serialport-v5');

# initialize the shelljs box
var shell = require('shelljs');

# initialize COM1
com1 = new SerialPort('/dev/COM1');

# adjust hardware to the System settings
shell.exec('sh /TURCK/scripts/serial.sh COM0 sethw');

# read up to 6 bytes
com1.on('readable', function () {
console.log('\nincomming Data com1:', com1.read(6));
});

# write buffer to COM1
const buf1 = new Buffer([0x01, 0x02, 0x03, 0x04, 0x05, 0x11]);
com1.write(buf1, console.log('message written from com1: ' +
buf1.toString('hex')));

# read line as ascii

const readline = SerialPort.parsers.Readline;

const parser = new readline();

com1.pipe(parser);

parser.on('data', console.log);

# write ascii com2.write('Let us talk together...\n');
```

7.2.5 Programming RFID channels with C or C++

The following examples illustrate the programming of the RFID interface with Ansi C or C++.

```
#include <stdio.h>
#include <stdlib.h>
#include <termios.h>
#include <fcntl.h>

// initialize function (use extern for C++)
ssize_t read (int __fd, void *__buf, size_t __nbytes) __wur;
ssize_t write (int __fd, const void *__buf, size_t __n) __wur;
int close (int __fd);

int main(void) {
     //choose Interface for connection
    const char *Path = "/dev/COM2";
    struct termios options;
    int fd, count, i ;
    unsigned char currentBuff[1];
    unsigned char InBuff[255];
    unsigned char *p_InBuff = InBuff;
    unsigned char Message[] = {0x48,0x65,0x6c,0x6c,0x6f};
    if((fd = open((Path), O_RDWR | O_NOCTTY)) != -1)
    {
         // Set serial Interface
        tcgetattr (fd, &options);

        cfsetspeed(&options, B9600);
        cfmakeraw (&options);
        options.c_cflag |= CS8;
        tcsetattr (fd, TCSANOW, &options);

        system("sh /TURCK/scripts/serial.sh COM2 sethw");
        // write to Interface COM2
        if ((write(fd, Message, sizeof(Message))) == -1)
                {
                         printf("not able to write...");
                 }

        // read from Interface COM2
        count = 0;
        do
        {
                 if ((count += read(fd, currentBuff, 1)) == -1)
                        printf("can not read...");
                *p_InBuff = currentBuff[0];
                p_InBuff++;
        }while(currentBuff[0] != 0xfe);

        // print:
         p_InBuff -= count;
        printf("\nData count: %i",count+1);
        printf("\nValues: \n");
         for(i = 0; i <= count ; i++)
        {
```

```
                 printf("%.02x ", *p_InBuff ); //"\nReceived value
%i: %.02x " p_InBuff++;
        }

        // close the Interface
        if((close(fd)) == -1)
        {
                 printf("\n can not close interface");
        }
    }
    else
        printf("can not open interface\n");
     return EXIT_SUCCESS;
}
```

7.3 Programming digital channels (DXP)

7.3.1 GPIOs of the DXP channels – Overview

The digital I/O channels (DXP) can be programmed as inputs or outputs via the GPIOs. The GPIOs are located under the following path:
/sys/class/gpio/…

Channel	Socket	Type	GPIO	Possible values
DXP8	C4	Input	110	0: Input off (0V) 1: Input on (24V)
		Output	12	0: Output off (0V) 1: Output on (24V)
DXP9		Input	111	0: Input off (0V) 1: Input on (24V)
		Output	13	0: Output off (0V) 1: Output on (24V)
DXP10	C5	Input	112	0: Input off (0V) 1: Input on (24V)
		Output	47	0: Output off (0V) 1: Output on (24V)
DXP11		Input	113	0: Input off (0V) 1: Input on (24V)
		Output	63	0: Output off (0V) 1: Output on (24V)
DXP12	C6	Input	114	0: Input off (0V) 1: Input on (24V)
		Output	86	0: Output off (0V) 1: Output on (24V)
DXP13		Input	116	0: Input off (0V) 1: Input on (24V)
		Output	87	0: Output off (0V) 1: Output on (24V)
DXP14	C7	Input	117	0: Input off (0V) 1: Input on (24V)
		Output	88	0: Output off (0V) 1: Output on (24V)
DXP15		Input	7	0: Input off (0V) 1: Input on (24V)
		Output	89	0: Output off (0V) 1: Output on (24V)

Setting the switchable VAUX power supply

Socket	Type	GPIO	Possible values
C4	Output	495	0: VAUX off 1: VAUX on
C5	Output	496	
C6	Output	497	
C7	Output	498	

Setting the switchable VAUX power supply – Diagnostics

Socket	Type	GPIO	Possible values
C4	Input	499	0: VAUX error-free 1: Error or overvoltage on VAUX
C5	Input	500	
C6	Input	501	
C7	Input	502	

7.3.2 Setting DXP functions via script

A script is installed on the device for setting the DXP channels. The script is located under the following path:
/TURCK/scripts/dxp.sh

The script can be used with the following syntax:

```
sh dxp.sh DXPx [value]
```

The following example sets the value for the DXP8 channel to ON.

```
sh dxp.sh DXP8 1
```

Parameter	Possible values
DXP8…DXP15	1: Switch on channel 0: Switch off channel

7.3.3 Programming DXP channels with Python 3

NOTE
The speed of the data transmission depends on the configured block size and the set transfer rate. The speed may possibly not be enough for time critical applications. To achieve faster data processing, the process can be set as a real-time process.

The following example shows the programming of the digital I/O channels with Python 3.

```
import sys
#GPIOs-> OUT: IN:
ports = ["47","112"]

# write GPIO:
try:
    # set direction to write DXP
    fo = open("/sys/class/gpio/gpio" + ports[0] +"/direction",
"w")
    fo.write("out")
    fo.close()
    # write DXP:
    f = open("/sys/class/gpio/gpio" + ports[0] +"/value", "w")
    f.write("1")
    f.close()
except:
    # export gpio if not done as yet
    f1 = open("/sys/class/gpio/export", "w")
    f1.write(ports[0]) f1.close()
    # set direction to write DXP
    fo = open("/sys/class/gpio/gpio" + ports[0] +"/direction",
"w")
    fo.write("out")
    fo.close()
    # write DXP:
    fw = open("/sys/class/gpio/gpio" + ports[0] +"/value", "w")
    fw.write("1")
    fw.close()

# read GPIO:
try:
    # set direction to read DXP
    fo = open("/sys/class/gpio/gpio" + ports[1] +"/direction",
"w")
    fo.write("in")
    fo.close()
    # set active low to get the right value...
    fal = open("/sys/class/gpio/gpio" + ports[1] +"/active_low",
"w")
    fal.write("1")
    fal.close()
    # read DXP: fr = open("/sys/class/gpio/gpio" + ports[1] +"/
value", "r")
    val=fr.read()
    fr.close()
    print(val)
except:
    # export gpio if not done as yet
    f1 = open("/sys/class/gpio/export", "w")
    f1.write(ports[1])
    f1.close()
    # set direction to read DXP
    fo = open("/sys/class/gpio/gpio" + ports[1] +"/direction", 
"w")
    fo.write("in")
    fo.close()
    # set active low to get the right value...
    fal = open("/sys/class/gpio/gpio" + ports[1] +"/active_low", 
"w")
    fal.write("1")
    fal.close()
    # read DXP:
    fr = open("/sys/class/gpio/gpio" + ports[1] +"/value", "r")
    val=fr.read()
    fr.close()
    print(val)
```

7.3.4 Programming DXP channels with Node.js

NOTE
The speed of the data transmission depends on the configured block size and the set transfer rate. The speed may possibly not be enough for time critical applications. To achieve faster data processing, the process can be set as a real-time process.

The following examples illustrate the programming of the digital I/O channels with Node.js. Further information on Node.js and the Node.js packages is provided at:
- **https://nodejs.org**
- **https://www.npmjs.com/**

```
// initialize the onoff box
const Gpio = require('onoff').Gpio;

function setGpioByInt(OUT, val) {
// switch from DXP to GPIO...
switch (OUT) {
    case 8:
        res = 12;
        break;
    case 9:
        res = 13;
        break;
    case 10:
        res = 47;
        break;
    case 11:
        res = 63;
        break;
    case 12:
        res = 86;
        break;
    case 13:
        res = 87;
        break;
    case 14:
        res = 88;
        break;
    case 15:
        res = 89;
}
        // initialize the GPIO just to write...
        const DXP_Write = new Gpio(res, "out");
        // write the GPIO / DXP...
        DXP_Write.writeSync(val);
        console.log('set Gpio '+ res + ' to ' + val);
}

function getGpio(IN) {
// switch from DXP to GPIO...
switch (IN) {
     case "8":
        res = 110;
```

```
            break;
        case "9":
            res = 111;
            break;
        case "10":
            res = 112;
            break;
        case "11":
            res = 113;
            break;
        case "12":
            res = 114;
            break;
        case "13":
            res = 116;
            break;
        case "14":
            res = 117;
            break;
        case "15":
            res = 7;
    }
        // initialize the GPIO just to read...
        const DXP_Read = new Gpio(res, "in");
        // set active low to get the right value...
        DXP_Read.setActiveLow('true');
        // read the GPIO / DXP...
        var res = DXP_Read.readSync();
        console.log('Gpio '+ r_Pin + ' is: ' + res);
        return res;
    }
```

7.3.5 Programming DXP channels with C or C++

The following example shows the programming of the digital I/O channels with Ansi C/C++.

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

// initialize function (use extern for C++)
int access(const char *pfad, int modus);

int main(void) {
     //choose DXP / GPIO for connection

    char GPIO_IN_FILE[] = "/sys/class/gpio/gpio114";
    char GPIO_OUT_FILE[] = "/sys/class/gpio/gpio86";
    char input[2]; FILE *fh;
/*
    ===============================================================
===============
    READ:

===================================================================
==========*/
    if( access( GPIO_IN_FILE, F_OK ) == -1 )
    {
        // file doesn't exist!
            // export gpio...
            if((fh = fopen("/sys/class/gpio/export", "w")) != 0)
            {
                fputs("114",fh);
                fclose(fh);
            }
            else
            {
                printf("failed on export to read...\n");
                printf("result: %i \n", (int)fh);
                return -1;
            }
    }
    // set direction to read...
    if((fh =fopen("/sys/class/gpio/gpio114/direction", "w")) != 0)
    {
        fputs("in",fh);
        fclose(fh);
    }
    else
    {
         printf("failed on setting direction to read...\n");
        return -1;
    }

    // set active low to read...
    if((fh = fopen("/sys/class/gpio/gpio114/active_low", "w")) !=
0)
    {
        fputs("1",fh);
        fclose(fh);
    }
    else
    {
        printf("failed on setting active low ...\n");
        return -1;
    }
    // read GPIO...
    if((fh = fopen("/sys/class/gpio/gpio114/value", "r")) != 0)
    {
        fgets(input,2,fh);
        fclose(fh);
        printf("Value: %c\n", input[0]);
    }
     else
    {
        printf("failed on reading ...\n");
        return -1;
    }
/*

===================================================================
==========
    WRITE:
    ===============================================================
==============*/
    if( access( GPIO_OUT_FILE, F_OK ) == -1 )
    {
        // file doesn't exist
            // export gpio...
            if((fh = fopen("/sys/class/gpio/export", "w")) != 0)
            {
                fputs("86",fh);
                fclose(fh);
            }
            else
            {
                printf("failed on export to write...\n");
                printf("result: %i \n", (int)fh);
                return -1;
        }
    }
    // set direction to read...
    if((fh = fopen("/sys/class/gpio/gpio86/direction", "w")) != 0)
    {
            fputs("out",fh);
            fclose(fh);
    }
    else
    {
         printf("failed on setting direction to write...\n");
        return -1;
    }
    // write GPIO...
    if((fh = fopen("/sys/class/gpio/gpio86/value", "w")) != 0)
    {
        fputs((const char*)"1",fh);
        fclose(fh);
    }
     else
    {
        printf("failed on writing ...\n");
        return -1;
    }
    return EXIT_SUCCESS;
}
```

7.4 Programming LED functions

7.4.1 LEDs – Overview

The device is provided with three freely programmable LEDs. The LEDs can be programmed individually via read and write commands. The LEDs are mapped on the system under the following path: "/sys/class/leds/..."

LED	Color	System name
APPL	Green	appl_green
	Red	appl_red
ERR	Green	err_green
	Red	err_red
RUN	Green	run_green
	Red	run_red

If the red and green of an LED are switched on at the same time, the LED is orange.

NOTE
During an ongoing firmware update the RUN-LED is used by the system.

7.4.2 Setting LED functions via a script

A script is installed on the device for setting the LEDs. The script is located under the following path:
/TURCK/scripts/led.sh

The script can be used with the following syntax:
```
sh led.h led color [value]
```

The following example switches on the red APPL LED.
```
sh led.sh appl red 1
```

LED	Possible color setting	Possible values
ERR	Green/red	1: Switch on LED 0: Switch off LED
RUN	Green/red	1: Switch on LED 0: Switch off LED
APPL	Green/red	1: Switch on LED 0: Switch off LED

7.4.3 Programming LED functions with Python 3

The following example shows the programming of the LED functions with Python 3:

```
import sys
import time
# write red LEDs:
fw = open("/sys/class/leds/run_red/brightness", "w")
fw.write("1")
fw.close()

fw = open("/sys/class/leds/appl_red/brightness", "w")
fw.write("1")
fw.close()

fw = open("/sys/class/leds/err_red/brightness", "w")
fw.write("1")
fw.close()

# Wait for 5 seconds
time.sleep(5)

# write green LEDs:
fw = open("/sys/class/leds/appl_green/brightness", "w")
fw.write("1")
fw.close()

fw = open("/sys/class/leds/err_green/brightness", "w")
fw.write("1")
fw.close()

fw = open("/sys/class/leds/run_green/brightness", "w")
fw.write("1")
fw.close()

# Wait for 5 seconds
time.sleep(5)

# clean red LEDs:
fw = open("/sys/class/leds/run_red/brightness", "w")
fw.write("0")
fw.close()

fw = open("/sys/class/leds/appl_red/brightness", "w")
fw.write("0")
fw.close()

fw = open("/sys/class/leds/err_red/brightness", "w")
fw.write("0")
fw.close()

# Wait for 5 seconds
time.sleep(5)

# clean green LEDs:
fw = open("/sys/class/leds/appl_green/brightness", "w")
```

```
fw.write("0")
fw.close()

fw = open("/sys/class/leds/err_green/brightness", "w")
fw.write("0")
fw.close()

fw = open("/sys/class/leds/run_green/brightness", "w")
fw.write("0")
fw.close()
```

7.4.4 Programming LED functions with Node.js

The following examples illustrate the programming of the LED functions with Node.js. Further information on Node.js and the Node.js packages is provided at:

- **https://nodejs.org**
- **https://www.npmjs.com/**

```
// initialize the onoff box
const Gpio = require('onoff').Gpio;

//initialize the leds which are free for the user

appl_green_led = new LED('appl_green');
appl_red_led = new LED('appl_red');
error_green_led = new LED('err_green');
error_red_led = new LED('err_red');
run_green_led = new LED('run_green');
run_red_led = new LED('run_red');
```

7.4.5 Programming LED functions with C or C++

The following example shows the programming of the LED functions with Ansi C/C++.

```
#include <stdio.h>
#include <stdlib.h>

// initialize function (use extern for C++)
char* strcpy(char* target, const char* source);
char* strcat(char* s1, const char* s2);

int main(void) {
   // LEDs for the customer:
   char *appl_green_led = "appl_green";
   char *appl_red_led = "appl_red";
   char *error_green_led = "err_green";
   char *error_red_led = "err_red";
   char *run_green_led = "run_green";
   char *run_red_led = "run_red";
   char *LED_FILE = "/sys/class/leds/";
   char *brightness = "/brightness";
   FILE *fh;

   char cur_Str[50] = {0};
   strcpy(cur_Str, LED_FILE);
   // take LED which will shine:
   strcat(cur_Str, run_red_led);
   strcat(cur_Str, brightness);

   //WRITE:
   printf("string to led: %s\n", cur_Str);

   // write LED...
   if((fh = fopen(cur_Str, "w")) != 0)
   {
      // write "1" to switch on and "0" to switch of LED
      fputs((const char*)"0",fh);
      fclose(fh);
   }
   else
   {
      printf("failed on writing ...\n");
      return -1;
   }
   return EXIT_SUCCESS;
}
```

7.5 Creating a C application

Requirements

The following components are required to create a C application:

- Toolchain for Cortex A8
- C program

Downloading a toolchain

The following toolchain is required for the Cortex A8 processor in order to cross compile a C program:

- OSELAS.toolchain-2014.12.0-arm-cortexa8-linux-gnueabihf-gcc-4.9.2-glibc-2.20-binutils-2.24-kernel-3.16-sanitized

The toolchain is available to download at **http://debian.pengutronix.de/debian**.

Example: Creating the C program

▶ Create the "hello.c" file.

▶ Copy the following text to the file:

```
// hello.c
#include <stdio.h>

int main() {
   printf("Hello World!\n");
   return 0;
}
```

▶ Create executable file with the following toolchain command:

```
/opt/OSELAS.Toolchain-2014.12.0/arm-cortexa8-linux-gnueabihf/
gcc-4.9.2-glibc-2.20-binutils-2.24-kernel-3.16-sanitized/bin/
arm-cortexa8-linux-gnueabihf-gcc -o helloExample hello.c
```

Example: Creating a C program via a make file

The “make” service program automates the creation of executable files from source code. C programs can be compiled via “make”. This uses a make file which contains the rules for creating executable files.

The following example shows a simple make file:

```
all: helloExample

helloExample: hello.o
    /opt/OSELAS.Toolchain-2014.12.0/arm-cortexa8-linux-gnueabihf/
gcc-4.9.2-glibc-2.20-binutils-2.24-kernel-3.16-sanitized/bin/arm-
cortexa8-linux-gnueabihf-gcc -o helloExample hello.o

hello.o: hello.c
   /opt/OSELAS.Toolchain-2014.12.0/arm-cortexa8-linux-gnueabihf/
gcc-4.9.2-glibc-2.20-binutils-2.24-kernel-3.16-sanitized/bin/arm-
cortexa8-linux-gnueabihf-gcc -c hello.c

clean:
   rm hello.o helloExample
}
```

- ▶ Create a make file.
- ▶ Save a make file in the same folder as the C application.
- ▶ Execute the make file with the “make” command.
- ⇨ The C program is installed.

7.6 Starting the application automatically (Autostart)

An application can be executed automatically with the Autostart function after the RFID interface is started. For this a configuration file (unit file) must be created, written to the device and activated

7.6.1 Autostart – Creating the configuration file (unit file)

- ▶ Create a unit file with the suffix “.service”.

Example: The “.setdxp.service” unit file starts a Node.js application, by which the DXP channels are triggered with every restart.

- ▶ Call up via “ExecStart” the application to be called every time the interface is restarted:
  `ExecStart=path_to_programm app/file`

A parameter can be transferred if required. Example: `ExecStart=path_to_programm app/file parameter`

Further information on unit configuration files is available at:
https://www.freedesktop.org/software/systemd/man/systemd.service.html

Example: Autostart of an application with a parameter transfer

```
ExecStart=/usr/bin/node /home/user/ hello_GPIO.js webactive
```

7.6.2 Example: Using the unit file

The following example downloads the Node.js file “hello_GPIO.js” and stores it at “/home/user”:

```
[Unit]
Description= trigger the DXPs
#After=Service_that_must_run_before.service

[Service] Type=simpleExecStart=/usr/bin/node /home/user/
hello_GPIO.js

[Install]
WantedBy=multi-user.target
```

- ▶ Creating an example file “./etc/systemd/system/”:
  `sudo touch /etc/systemd/system/setdxp.service`
- ▶ Open the created file:
  `sudo nano /etc/systemd/system/setdxp.service`
- ▶ Insert the source text shown above in the opened file.

7.6.3 Activating the unit file

After being created, the unit file must be activated via the systemctl command. Access rights to the root directory are required to activate. The .services file suffix is optional and can be omitted.

- ▶ Activate the unit file via the following command:
  `sudo systemctl enable setdxp.service`

The created symlink is:

```
/etc/systemd/system/multi-user.target.wants/setdxp.service â /etc/
systemd/system/setdxp.service
```

Deactivating the unit file

- ▶ Deactivate the unit file with the following command:
  `sudo systemctl disable setdxp.service`

7.7 Managing access rights

The device supports the standard Linux user management. The access rights can be managed with the following standard Linux tools:

- adduser
- addgroup
- passwd

User	Rights	Password
root	System administrator (all access rights)	turck
user	Restricted access rights and console rights	password
sftpuser	Access rights, SFTP rights in the directory /home	password

7.8 Installing Python packages

Modules, libraries and other software can be configured via the BSP (Board Support Package) with the PTXdist distribution tool and loaded on the device. If packages are to be integrated in an existing firmware, they must be created beforehand with PTXdist. PTXdist is available for download at **https://www.pengutronix.de/de/software/ptxdist.html**.

The ipkg package manager (Itsy Package Management System) is installed on the device for integrating software packages. The ipkg package manager makes it possible to also install Python modules at a later time.

7.8.1 Example: Installing the Python module

The following example explains the procedure for the installation of the Python sh module. The Python module is integrated at a later time in an existing firmware.

Requirements

- PTXdist is installed on the Linux host system.
- The required Python module was downloaded (example: **https://amoffat.github.io/sh/**).

Example: Installing the Python sh module

In order to create the Python sh module, a rule file must be created first.

▶ Create the rule file with the following command:

```
$ ptxdist newpackage target
```

▶ Create interactive information on the package:

```
Output:
ptxdist: creating a new 'target' package:
ptxdist: enter package name...........:
sh ptxdist: enter version number......:
1.12.13 ptxdist: enter URL of basedir.:https://github.com/
amoffat/sh/archive/
ptxdist: enter suffix.................: tar.gz
ptxdist: enter package author.........: Your Name <E-Mail>
ptxdist: enter package section........: Python3


generating rules/sh.make
generating rules/sh.in
```

The sh.make and sh.in files are created automatically.

▶ If known, enter the key of the package as a SH_MD5 parameter in the sh.make file.

▶ Set the SH_CONF_TOOL parameter in the sh.make file to the appropriate tool (in this case: Python 3).

```
SH_CONF_TOOL :=python3
```

▶ If a Python module has a separate subfolder: Create the subfolder in the target directory (in this case not required):

```
@$(call install_copy, module, 0, 0, 0755, $(PYTHON3_SITEPACK-
AGES)/foldername)
```

▶ In the #Target-Install area, specify the installation location of the Python module in the target system (Example: sh module):

```
@for file in `find $(SH_PKGDIR)/usr/lib/python$(PYTHON3_MA-
JORMINOR)/site-packages \
    ! -type d ! -name "*.py" -printf "%P\n"`; do \
  $(call install_copy, sh, 0, 0, 0644, -, \
    /usr/lib/python$(PYTHON3_MAJORMINOR)/site-packages/$
$file); \
done
```

Dependencies can be entered in the sh.in file. Python 3 must be available in the following example in order to install Python modules. The “setuptools” module must be available on the host system.

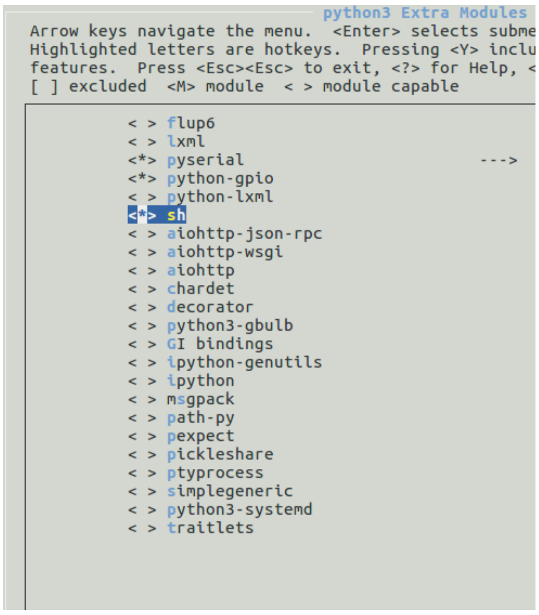
- ▶ Enter dependencies as follows:

```
## SECTION=python3
config PYTHON_SH
  tristate
  select PYTHON3                  # Python 3 must be installed
  select HOST_PYTHON3_SETUPTOOLS # Setuptools must be in-
stalled on the host
  prompt "sh"
  help
    FIXME
```

- ▶ Compile.

In order for the sh module to be created with the next build, the module must be selected in “menuconfig”:

- ▶ Open “menuconfig” with the following command:
  `ptxdist menuconfig`
- ▶ Navigate to the Python 3 modules via “Scripting Languages” → “python3 Extra Modules”.
- ▶ Select the sh module.
- ▶ Save the configuration.



```
                                        python3 Extra Modules
  Arrow keys navigate the menu.  <Enter> selects subme
  Highlighted letters are hotkeys.  Pressing <Y> inclu
  features.  Press <Esc><Esc> to exit, <?> for Help, <
  [ ] excluded  <M> module  < > module capable

            < > flup6
            < > lxml
            <*> pyserial                          --->
            <*> python-gpio
            < > python-lxml
            <*> sh
            < > aiohttp-json-rpc
            < > aiohttp-wsgi
            < > aiohttp
            < > chardet
            < > decorator
            < > python3-gbulb
            < > GI bindings
            < > ipython-genutils
            < > ipython
            < > msgpack
            < > path-py
            < > pexpect
            < > pickleshare
            < > ptyprocess
            < > simplegeneric
            < > python3-systemd
            < > traitlets
```

Fig. 23: PTXdist – “Python 3 Extra Modules”

▶ Generate ipkg packages with the following command:
```
ptxdist go
```
➪ If no errors have occurred, the package with the sh module can be found at "platform-tben-lx-linux/packages/":
```
$ ls platform-tben-lx-linux/packages/
...
python3_3.5.0_armhf.ipk
sh_1.12.14_armhf.ipk
...
```
▶ Copy the ipk file to the TBEN device (e.g. with scp):
```
scp ~/turck/TBEN-Lx-4RFID-8DXP-LNX/platform-tben-lx-linux/
packages/sh_1.12.14_armhf.ipk
root@Target-IP:/directory/of/your/choice/
```
▶ Log into the TBEN device in order to install the "sh_1.12.14_armhf.ipk" package.

▶ Call up the ipkg manager to install the Python module:
```
ipkg -force-depends install sh_1.12.14_armhf.ipk
```
➪ The module is available in the Python Interpreter.

8 Setting

The read/write heads must be assigned parameters via the read/write head protocol. For this the read/write head protocol must be implemented on the TBEN device.

9 Operation

9.1 LEDs

The devices are provided with three freely programmable multicolor LEDs.

DXP LEDs (digital channels, LEDs DXP0…3)		
LED green	**LED red**	**Meaning**
Off	Off	No I/O signal present
Lit	Off	I/O signal present
Off	Lit	Overload at output
Flashing	Flashing	Overload of the auxiliary voltage

APPL LED	**Meaning**
Flashing white	Wink command active

9.2 Reset device (Reset)

The device can be reset to the factory settings via the rotary coding switches and via the Turck Service Tool using the F_Reset function. The device can be reset via a reboot or the Reset command in the event of an error. The settings are retained if a restart was carried out or the device was reset with the Reset command.

10 Troubleshooting

If the device does not function as expected, first check whether ambient interference is present.
If there is no ambient interference present, check the connections of the device for faults.
If there are no faults, there is a device malfunction. In this case, decommission the device and replace it with a new device of the same type.

11 Maintenance

11.1 Executing the firmware update via the USB interface

▶ Create the “FW_UPDATE” folder on a USB stick.
▶ Save the firmware as a bin file in the “FW_UPDATE” folder.
▶ Insert the USB stick in the device.
➪ The RUN LED flashes green at 0.5 Hz.
▶ Hold down the Set button for at least 3 seconds within 30 segments.
➪ The RUN LED flashes in the sequence 3 x green - pause (1 Hz) - 3 × green - pause (1 Hz) - ….
➪ The data is loaded into the device.
➪ The firmware update is completed when the RUN LED flashes orange at 1 Hz.
▶ Remove the USB stick.
▶ Carry out a voltage reset.
➪ The device restarts.

11.2 Carrying out a firmware update via the console

The firmware update can either be transferred to the device with a suitable tool (e.g. WinSCP or FileZilla) or as a “Secure Copy”.

▶ Load the update file (e.g. TBEN-Lx-4RFID-8DXP-LNX_V1010.bin) to the device with a suitable tool (e.g. WinSCP) (see “Example: Carrying out a firmware update with WinSCP and PuTTY).
Alternatively transfer the update file to the device as a “Secure Copy”:
```
scp Path/To/Your/File/root.ubifs user@ip_of_your_board:/run/
Update:
```
▶ Carry out the update on the device with the following command:
```
sudo update system /path/To/The/Updatefile/updatefile
```
➪ If no error messages appear, the firmware was successfully installed.

11.2.1 Example: Carrying out a firmware update with WinSCP and PuTTY

The following example carries out a firmware update using the WinSCP and PuTTY tools.

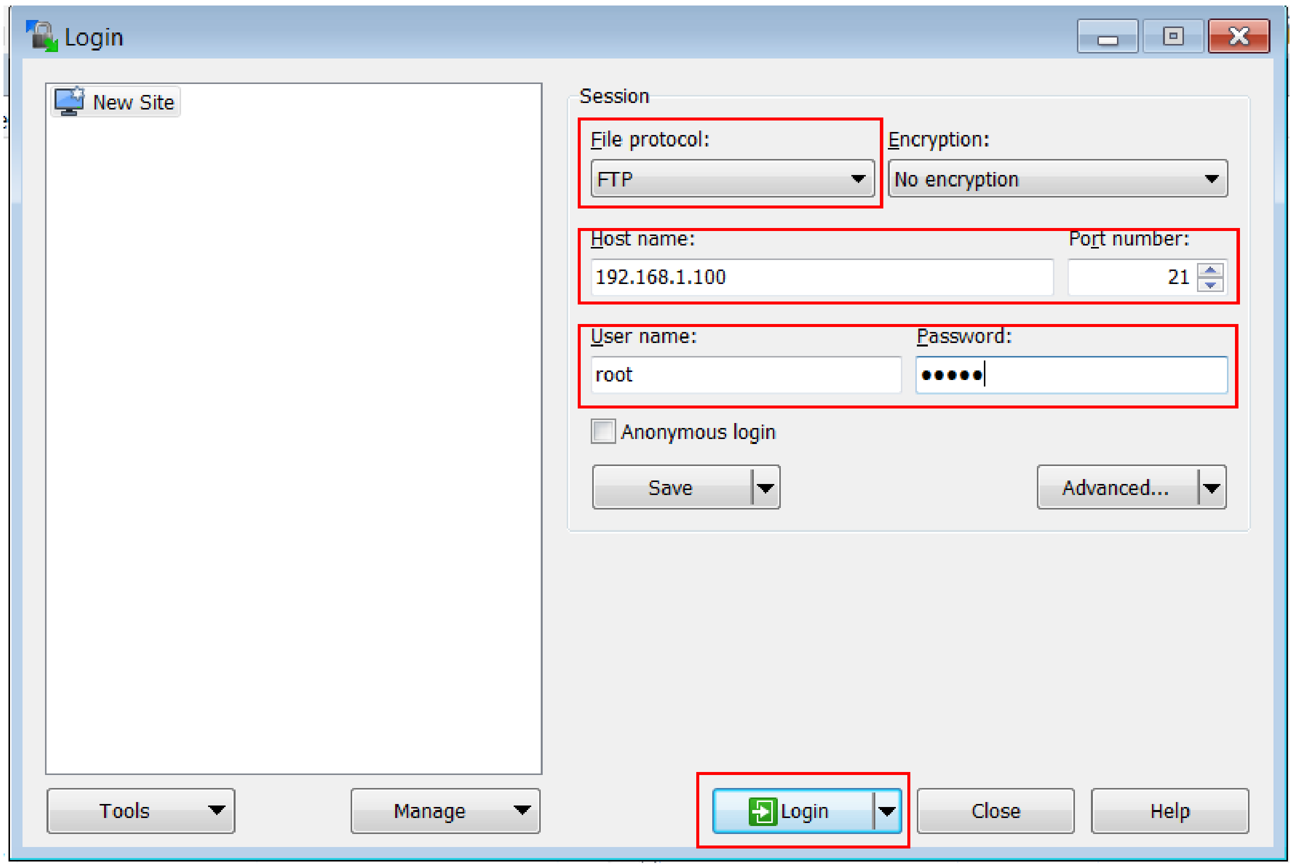
Requirements

- WinSCP is installed.
- PuTTY is installed.
- The update file is available as a .bin file on a local computer.

Transferring a firmware file with WinSCP

▶ Log into the device in WinSCP with the following entries:
Transmission protocol: FTP
Computer name: IP address of the device (in this case: 192.168.1.100)
Port: 21
User name: root
Password: turck

NOTE
The login via SFTP and Port 22 is also possible.



Login

Session

File protocol: FTP

Encryption: No encryption

Host name: 192.168.1.100

Port number: 21

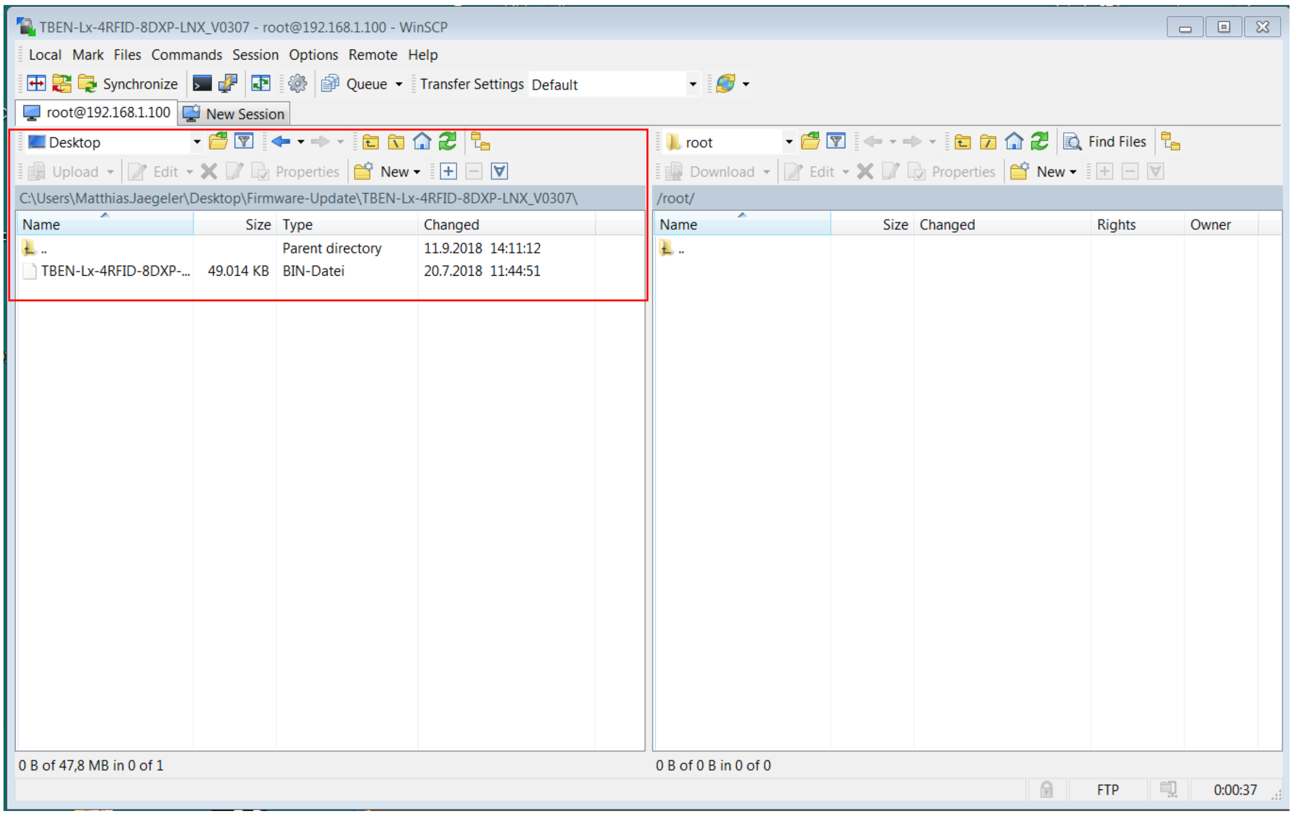
User name: root

Password: •••••

Anonymous login

Fig. 24: WinSCP – Login

▶ Navigate in WinSCP to the memory location of the update file on the host PC.


TBEN-Lx-4RFID-8DXP-LNX_V0307 - root@192.168.1.100 - WinSCP
Local Mark Files Commands Session Options Remote Help
Synchronize
Queue
Transfer Settings Default
root@192.168.1.100
New Session
Desktop
Upload
Edit
Properties
New
root
Find Files
Download
FTP
0:00:37

C:\Users\Matthias.Jaegeler\Desktop\Firmware-Update\TBEN-Lx-4RFID-8DXP-LNX_V0307\

Name	Size	Type	Changed
..		Parent directory	11.9.2018 14:11:12
TBEN-Lx-4RFID-8DXP-...	49.014 KB	BIN-Datei	20.7.2018 11:44:51

0 B of 47,8 MB in 0 of 1

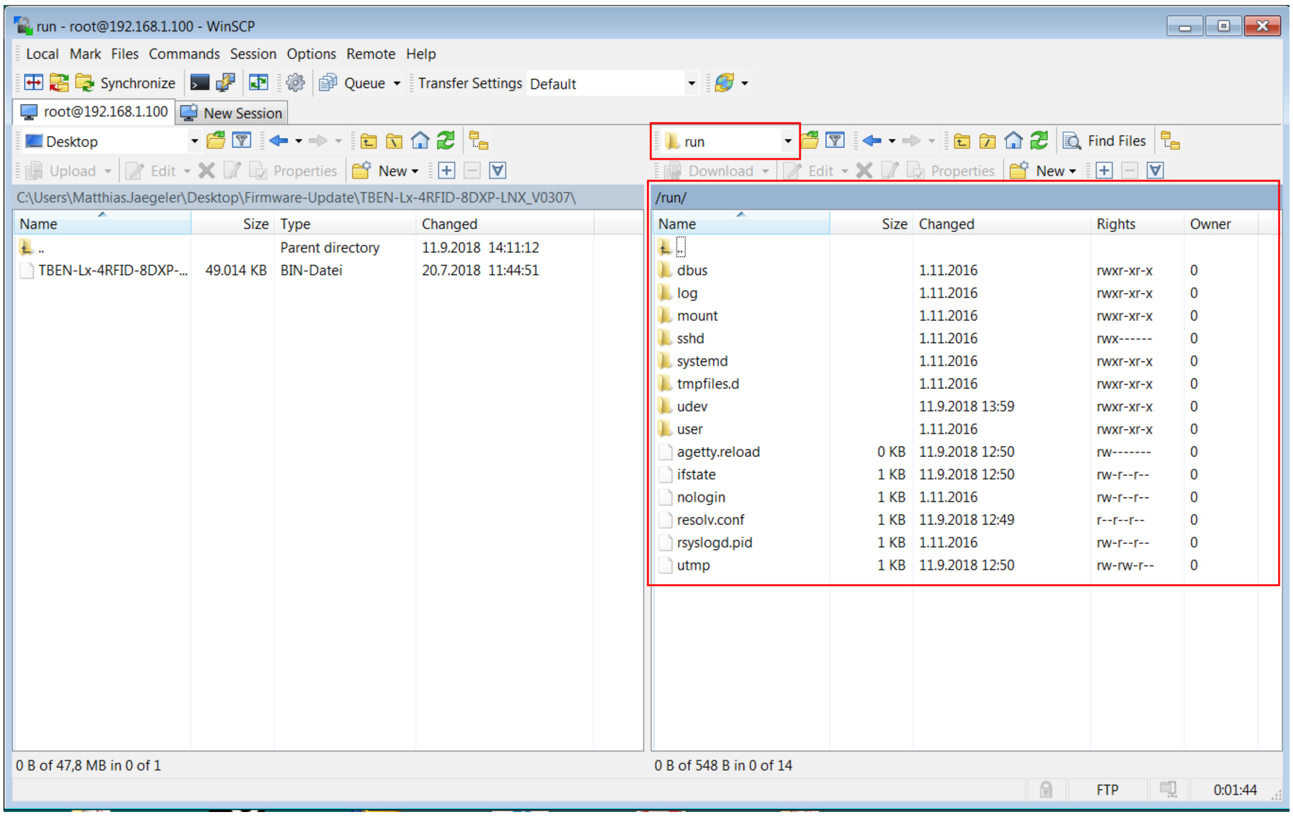
/root/

Name	Size	Changed	Rights	Owner
..				

0 B of 0 B in 0 of 0

Fig. 25: WinSCP – Memory location of the update file on the host PC

▶ Navigate to the Run directory on the device.


run - root@192.168.1.100 - WinSCP
Local Mark Files Commands Session Options Remote Help
Synchronize
Queue
Transfer Settings Default
root@192.168.1.100
New Session
Desktop
Upload
Edit
Properties
New
run
Find Files
Download
FTP
0:01:44

C:\Users\Matthias.Jaegeler\Desktop\Firmware-Update\TBEN-Lx-4RFID-8DXP-LNX_V0307\

Name	Size	Type	Changed
..		Parent directory	11.9.2018 14:11:12
TBEN-Lx-4RFID-8DXP-...	49.014 KB	BIN-Datei	20.7.2018 11:44:51

0 B of 47,8 MB in 0 of 1

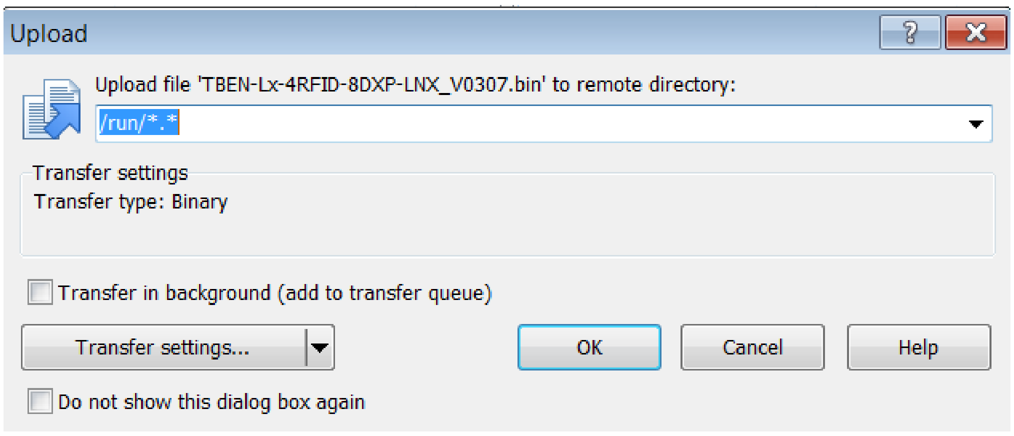
/run/

Name	Size	Changed	Rights	Owner
..				
dbus		1.11.2016	rwxr-xr-x	0
log		1.11.2016	rwxr-xr-x	0
mount		1.11.2016	rwxr-xr-x	0
sshd		1.11.2016	rwx------	0
systemd		1.11.2016	rwxr-xr-x	0
tmpfiles.d		1.11.2016	rwxr-xr-x	0
udev		11.9.2018 13:59	rwxr-xr-x	0
user		1.11.2016	rwxr-xr-x	0
agetty.reload	0 KB	11.9.2018 12:50	rw-------	0
ifstate	1 KB	11.9.2018 12:50	rw-r--r--	0
nologin	1 KB	1.11.2016	rw-r--r--	0
resolv.conf	1 KB	11.9.2018 12:49	r--r--r--	0
rsyslogd.pid	1 KB	1.11.2016	rw-r--r--	0
utmp	1 KB	11.9.2018 12:50	rw-rw-r--	0

0 B of 548 B in 0 of 14

Fig. 26: WinSCP – RUN directory on the device

- Save the update file in the run directory of the device by drag and drop or by clicking **Upload**.
- Confirm the following prompts with **OK**.


Upload

Upload file 'TBEN-Lx-4RFID-8DXP-LNX_V0307.bin' to remote directory:

/run/*.*

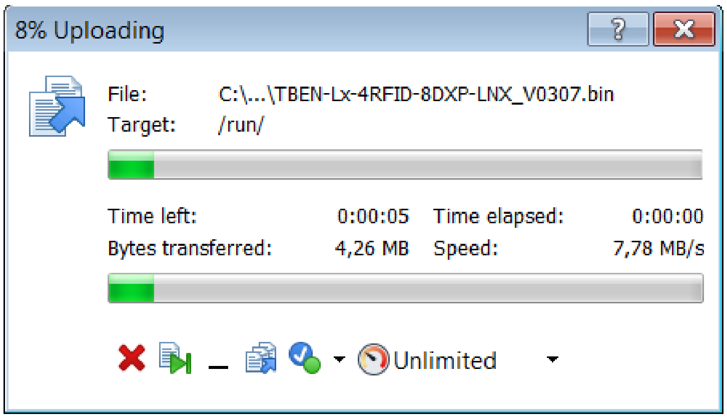
Transfer settings
Transfer type: Binary

☐ Transfer in background (add to transfer queue)

☐ Do not show this dialog box again

Fig. 27: WinSCP – Confirm the prompt for the transfer

The transfer of the update file is displayed by WinSCP as follows:


8% Uploading

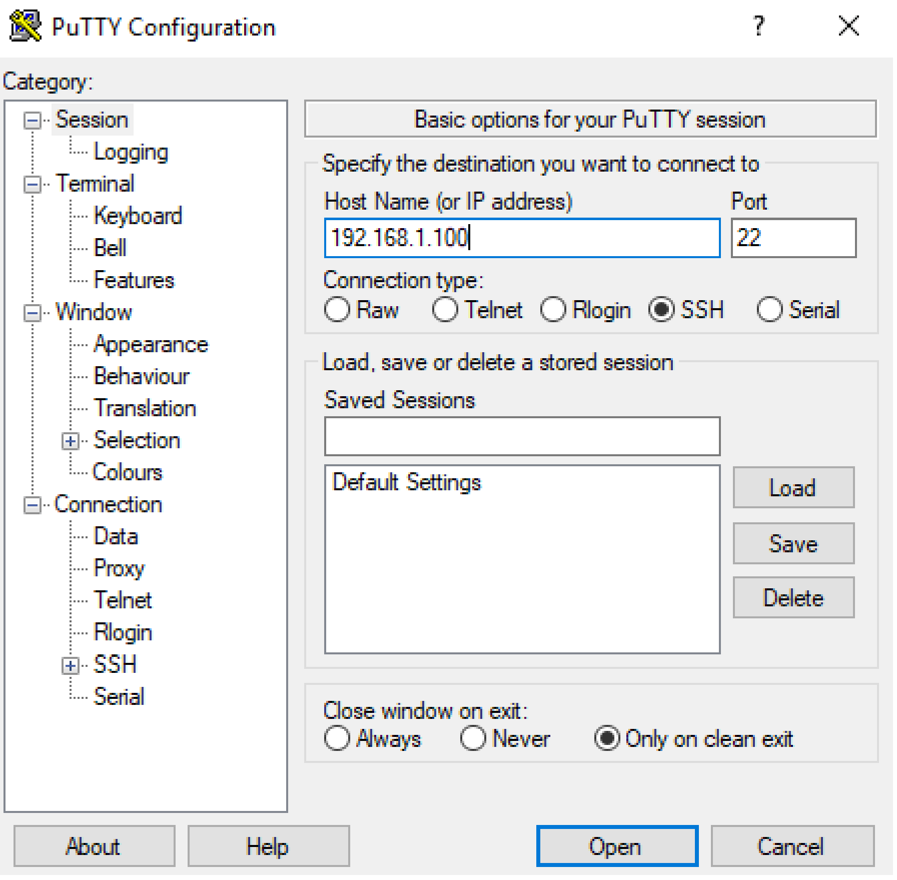
File: C:\...\TBEN-Lx-4RFID-8DXP-LNX_V0307.bin
Target: /run/

Time left: 0:00:05
Time elapsed: 0:00:00
Bytes transferred: 4,26 MB
Speed: 7,78 MB/s

Fig. 28: WinSCP – File transfer

Carrying out a firmware update with PuTTY

- ▶ Open PuTTY.
- ▶ Enter the following settings in PuTTY:
  - Host Name: IP address of the device
  - Port: 22
- ▶ Optional: Assign a name for the current session (here: TBEN-Lx_LNX). The session can be loaded via **Load** for later repetitions.
- ▶ With saved sessions: Select TBEN-Lx_LNX and confirm with **Load**.
- ▶ Click **Open**.



PuTTY Configuration

Category: Session > Logging; Terminal > Keyboard, Bell, Features; Window > Appearance, Behaviour, Translation, Selection, Colours; Connection > Data, Proxy, Telnet, Rlogin, SSH, Serial

Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): 192.168.1.100
Port: 22

Connection type: Raw Telnet Rlogin [✓ SSH] Serial

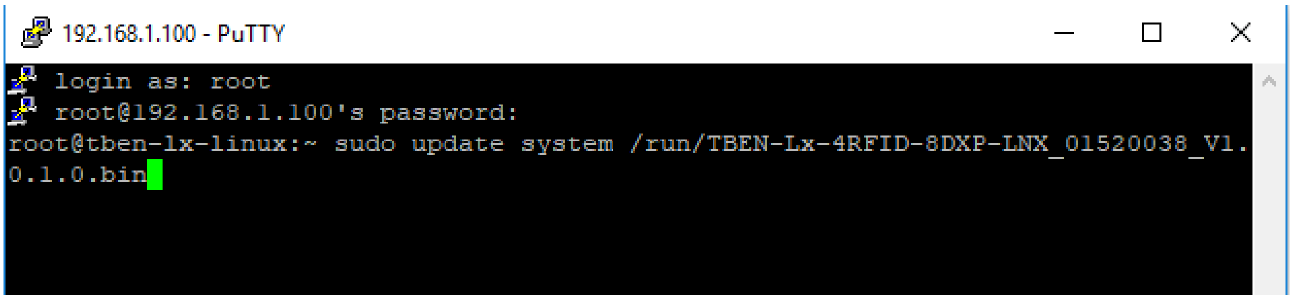
Load, save or delete a stored session

Saved Sessions: Default Settings

Close window on exit: Always Never [✓ Only on clean exit]

Fig. 29: PuTTY configuration

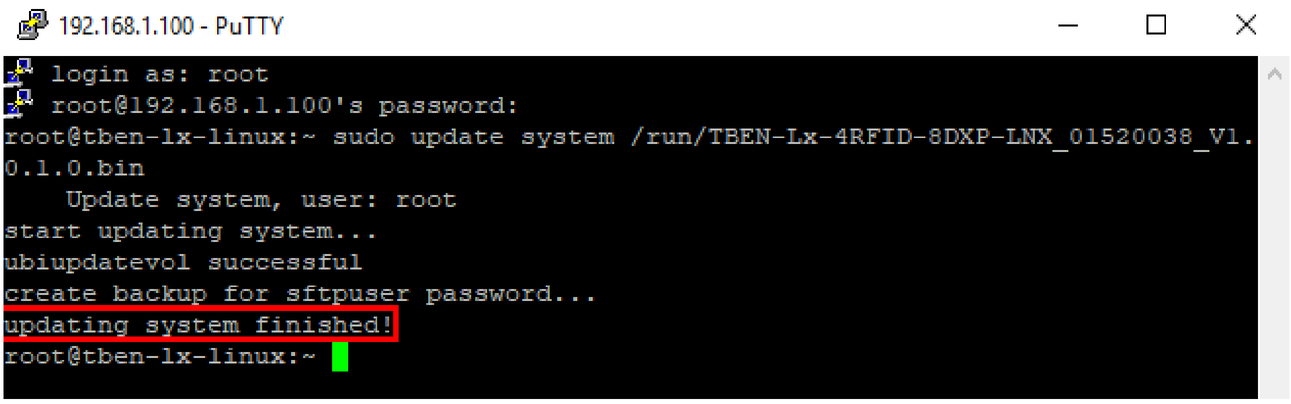
- Log in on the device with user name "root" (password: "turck"). The password entered is not displayed in PuTTY.
- Run the update with the command `sudo update system /run/[File name].bin`.
  Example: `sudo update system /run/TBEN-Lx-4RFID-8DXP-LNX_01520038_V1.0.1.0.bin`



```
192.168.1.100 - PuTTY
login as: root
root@192.168.1.100's password:
root@tben-lx-linux:~ sudo update system /run/TBEN-Lx-4RFID-8DXP-LNX_01520038_V1.
0.1.0.bin
```

Fig. 30: PuTTY – Starting the firmware update

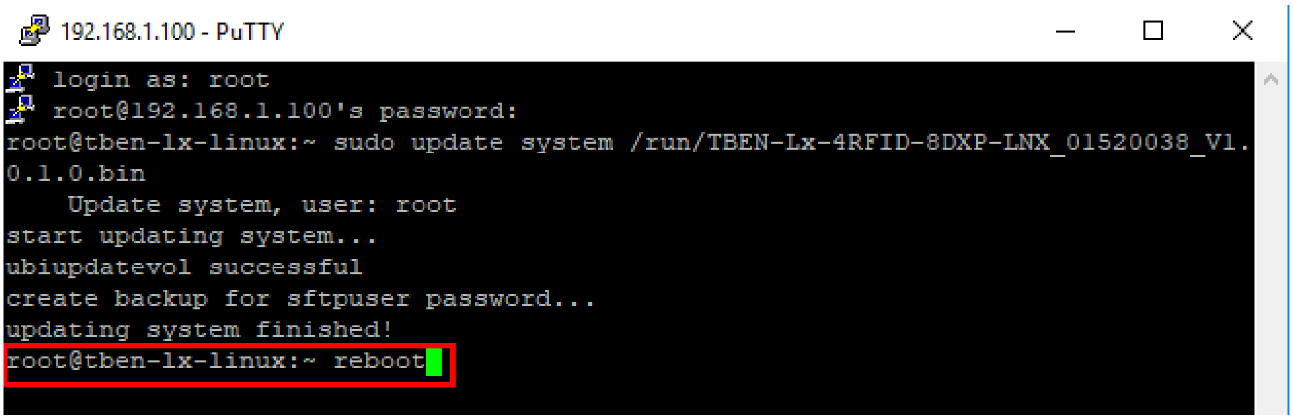
- Wait till the `updating system finished` message is displayed.



```
192.168.1.100 - PuTTY
login as: root
root@192.168.1.100's password:
root@tben-lx-linux:~ sudo update system /run/TBEN-Lx-4RFID-8DXP-LNX_01520038_V1.
0.1.0.bin
    Update system, user: root
start updating system...
ubiupdatevol successful
create backup for sftpuser password...
updating system finished!
root@tben-lx-linux:~
```

Fig. 31: PuTTY – Update successful

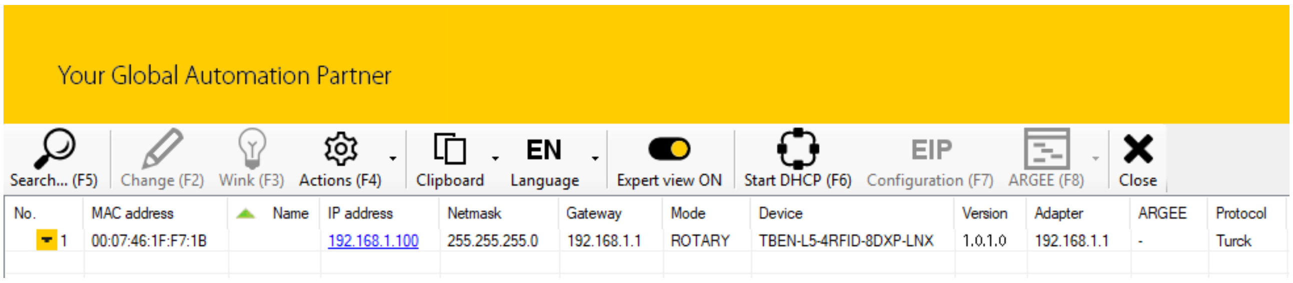
- Restart the device with the `reboot` command.



```
192.168.1.100 - PuTTY
login as: root
root@192.168.1.100's password:
root@tben-lx-linux:~ sudo update system /run/TBEN-Lx-4RFID-8DXP-LNX_01520038_V1.
0.1.0.bin
    Update system, user: root
start updating system...
ubiupdatevol successful
create backup for sftpuser password...
updating system finished!
root@tben-lx-linux:~ reboot
```

Fig. 32: PuTTY – Restarting the device

▶ Check the current firmware status, e.g. with the Turck Service Tool: The current firmware status is shown under **Version**.



No.	MAC address	Name	IP address	Netmask	Gateway	Mode	Device	Version	Adapter	ARGEE	Protocol
1	00:07:46:1F:F7:1B		192.168.1.100	255.255.255.0	192.168.1.1	ROTARY	TBEN-L5-4RFID-8DXP-LNX	1.0.1.0	192.168.1.1	-	Turck

Fig. 33: Turck Service Tool – Checking the firmware version

12 Repair

The device must not be repaired by the user. The device must be decommissioned if it is faulty. Observe our return acceptance conditions when returning the device to Turck.

12.1 Returning devices

Returns to Turck can only be accepted if the device has been equipped with a Decontamination declaration enclosed. The decontamination declaration can be downloaded from
http://www.turck.de/de/produkt-retoure-6079.php
and must be completely filled in, and affixed securely and weather-proof to the outside of the packaging.

13 Disposal

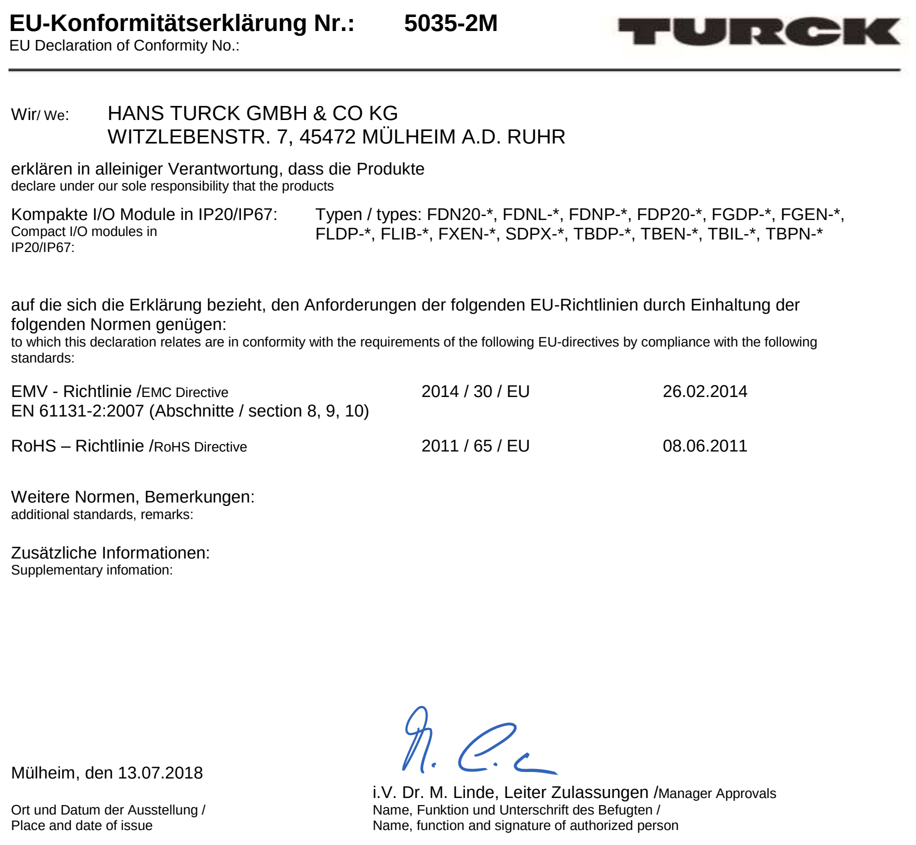
The devices must be disposed of correctly and must not be included in normal household garbage.

14 Technical Data

Technical data	
Power supply	
Power supply voltage	24 VDC
Permissible range	18…30 VDC
Total current	V1 max. 8 A, V2 max. 9 A at 70 °C per module
RFID power supply	2 A per channel at 70° C
Sensor/actuator supply	2 A per socket at 70°C
Potential isolation	Potential isolation of V1 and V2 voltage group
Dielectric strength	Up to 500 VDC V1 and V2 to Ethernet
Power dissipation	Typically ≤ 5 W
System description	
Processor	Cortex A8, 800 MHz
Memory	512 MB Flash ROM; 512 MB DDR3 RAM
Memory expansion	1 × USB Host port
Real-time clock	Yes
Operating system	Linux
System data	
Transfer rate	Ethernet 10 Mbit/s/100 Mbit/s
Connection technology	2 × M12, 4-pin, D-coded
RFID	
No. of channels	4
Connection technology	M12
Power supply	2 A per channel at 70 °C, short-circuit-proof
Cable length	max. 50 m
Digital inputs	
No. of channels	8
Connection technology	M12, 5-pin
Input type	PNP
Type of input diagnostics	Channel diagnostics
Switch threshold	EN 61131-2 type 3, pnp
Signal voltage Low signal	< 5 V
Signal voltage High signal	> 11 V
Signal current Low signal	<1.5 mA
Signal current High signal	> 2 mA
Potential isolation	Galvanic isolation at P1/P2
Dielectric strength	Up to 500 VDC (V1 and V1 compared to Ethernet)
Digital outputs	
No. of channels	8
Connection technology of outputs	M12, 5-pin
Output type	PNP
Type of output diagnostics	Channel diagnostics

Technical data	
Output voltage	24 VDC from potential group
Output current per channel	2.0 A, short-circuit proof, max. 4.0 A per socket
Utilization factor	0.56
Load type	EN 60947-5-1: DC-13
Short-circuit protection	Yes
Potential isolation	Galvanic isolation at P1/P2
Dielectric strength	Up to 500 VDC (V1 and V1 compared to Ethernet)
Conformity with standard/directive	
Vibration test	Acc. to EN 60068-2-6
Acceleration	Up to 20 g
Shock testing	Acc. to EN 60068-2-27
Drop and topple	Acc. to IEC 60068-2-31/IEC 60068-2-32
EMC (electromagnetic compatibility)	Acc. to EN 61131-2
Approvals and certificates	CE
UL cond.	cULus LISTED 21 W2, Encl.Type 1 IND.CONT.EQ., Encl. Type 1 -40…+55 °C
General information	
Dimensions (W × L × H)	60.4 × 230.4 × 39 mm
Operating temperature	-40…+70 °C
Storage temperature	-40…+85 °C
Operating height	max. 5000 m
Degree of protection	IP65/IP67/IP69K
MTTF	75 years to SN 29500 (Ed. 99) 20 °C
Housing material	PA6-GF30
Housing color	Black
Material of window	Lexan
Material of screw	303 stainless steel
Material of label	Polycarbonate
Halogen-free	Yes
Mounting	2 fixing holes, Ø 6.3 mm

15 Appendix: Declaration of Conformity



EU-Konformitätserklärung Nr.: **5035-2M**
EU Declaration of Conformity No.:

TURCK

Wir/ We: HANS TURCK GMBH & CO KG
WITZLEBENSTR. 7, 45472 MÜLHEIM A.D. RUHR

erklären in alleiniger Verantwortung, dass die Produkte
declare under our sole responsibility that the products

Kompakte I/O Module in IP20/IP67: Compact I/O modules in IP20/IP67:	Typen / types: FDN20-*, FDNL-*, FDNP-*, FDP20-*, FGDP-*, FGEN-*, FLDP-*, FLIB-*, FXEN-*, SDPX-*, TBDP-*, TBEN-*, TBIL-*, TBPN-*

auf die sich die Erklärung bezieht, den Anforderungen der folgenden EU-Richtlinien durch Einhaltung der folgenden Normen genügen:
to which this declaration relates are in conformity with the requirements of the following EU-directives by compliance with the following standards:

EMV - Richtlinie /EMC Directive EN 61131-2:2007 (Abschnitte / section 8, 9, 10)	2014 / 30 / EU	26.02.2014
RoHS – Richtlinie /RoHS Directive	2011 / 65 / EU	08.06.2011

Weitere Normen, Bemerkungen:
additional standards, remarks:

Zusätzliche Informationen:
Supplementary infomation:

[illegible]

Mülheim, den 13.07.2018	i.V. Dr. M. Linde, Leiter Zulassungen /Manager Approvals
Ort und Datum der Ausstellung / Place and date of issue	Name, Funktion und Unterschrift des Befugten / Name, function and signature of authorized person

16 Appendix: Example – “HelloGPIO” for Node.js

```
// This script show how to use the DXP's of the TBEN-4RFID-8DXP-LNX
// Start the script with "webactive" option will make the I/O useable via webbrowser
// Strat the script without options  will toggle the I/O's

const Gpio = require('onoff').Gpio;

if (process.argv[2] == 'webactive')
{
    runserver();
}
else
{
    //toggle the dxp's...
    time0 =new Date().getTime();
    for(var state = 1; state > -1; state--)
    {
        var index=8;
        while(index < 16 )
        {
            var timenow = new Date().getTime();
            if (timenow - time0 > 1000)
            {
                // func:
                setGpioByInt(index, state);
                index++;
                time0=timenow;
            }
        }
    }
}


// Make the dxp switchable via webbrowser
// http://ip_of_the_board:8080/?dxp=13&value=1
function runserver()
{
    var http = require('http');
    var url = require('url');
    http.createServer(function (req, res){
        res.writeHead(200, {'Contend-Type': 'text/plain'});
        var q = url.parse(req.url, true).query;

        dxp_port= q.dxp;
        dxp_value= q.value;
        //const DXP_Writer = new Gpio(12, "out");
        //DXP_Writer.writeSync(0);

        if (dxp_port > 7 && dxp_port < 16)
        {
            res.write('Would like read DXP' + dxp_port + ' or set to ' + dxp_value );
            if (dxp_value == '1')
            {
                setGpio(dxp_port, 1);
            }
            else if(dxp_value == '0'){
                setGpio(dxp_port, 0);
            }
            res.write('\nDXP'+ dxp_port +' ist: ' + getGpio(dxp_port) + ' now\n');

        }
        else{
            res.write('DXP ports are between 8 and 16 available only');
        }
        res.end('\nsee you...\n');
    }).listen(8080, '192.168.1.81');
    console.log('Server running at http://192.168.1.81:8080/');
}
```

```
function setGpio(w_Pin, val) {

    switch (w_Pin) {
    case "8":
        res = 12;
        break;
    case "9":
        res = 13;
        break;
    case "10":
        res = 47;
        break;
    case "11":
        res = 63;
        break;
    case "12":
        res = 86;
        break;
    case "13":
        res = 87;
        break;
    case "14":
        res = 88;
        break;
    case "15":
        res = 89;
    }
    const DXP_Write = new Gpio(res, "out");
    DXP_Write.writeSync(val);
    console.log('set Gpio '+ res + ' to ' + val);
}

function setGpioByInt(w_Pin, val) {

    switch (w_Pin) {
    case 8:
        res = 12;
        break;
    case 9:
        res = 13;
        break;
    case 10:
        res = 47;
        break;
    case 11:
        res = 63;
        break;
    case 12:
        res = 86;
        break;
    case 13:
        res = 87;
        break;
    case 14:
        res = 88;
        break;
    case 15:
        res = 89;
    }
    const DXP_Write = new Gpio(res, "out");
    DXP_Write.writeSync(val);
    console.log('set Gpio '+ res + ' to ' + val);
}

function getGpio(r_Pin) {

    switch (r_Pin) {
        case "8":
            res = 110;
            break;
        case "9":
            res = 111;
            break;
        case "10":
            res = 112;
            break;
        case "11":
            res = 113;
            break;
        case "12":
            res = 114;
            break;
        case "13":
            res = 116;
            break;
        case "14":
            res = 117;
            break;
        case "15":
            res = 7;
        }
        const DXP_Read = new Gpio(res, "in");
        DXP_Read.setActiveLow('true');
        var res = DXP_Read.readSync();
        console.log('Gpio '+ r_Pin + ' is: ' + res);
        return res;
    }
```

100002513 | 2019/05

www.turck.com