MANUAL

...feel the spirit of a new generation

RESI-KNX-MODBUS
RESI-KNX-ASCII

Great care has been taken in the creation of the text, illustrations and program examples in this manual. The editors and publishers accept no responsibility for any inadvertent omission of entries or for typographical or other errors herein. Nor can they be held responsible or liable for consequences arising from any errors herein.

1 History

Date	Editor	Description
13.01.10	DI HC Sigl	Initial release
05.05.15	DI HC Sigl	Correction to the query group function
22.09.15	DI HC Sigl	First version for KNX-MODBUS and KNX-ASCII

2 Content

RESI-KNX-MODBUS **1**

RESI-KNX-ASCII **1**

1 HISTORY **2**

2 CONTENT **3**

3 IMPORTANT SECURITY NOTES **4**

4 GENERAL INFORMATION **6**

5 MOUNTING AND CONNECTIONS **8**

5.1 ASSEMBLING 8
5.2 CLAMPS AND LEDS 9
5.3 DIP SWITCH SETTINGS 10
5.4 WIRING DIAGRAM 11

6 CONFIGURATION WITH RESI MODBUSCONFIGURATOR SOFTWARE **12**

6.1 ESTABLISH A CONNECTION 12
6.2 BASIC FUNCTIONS 12
6.3 THE CONFIGURATION TABLE 13
6.4 THE CONTEXT MENU 14
6.5 CONTEXT MENU: ADD ENTRY 14
6.6 CONTEXT MENU: DELETE SELECTED LINES 15
6.7 CONTEXT MENU: INSERT ENTRY 15
6.8 CONTEXT MENU: COPY ENTRY 16
6.9 KONTEXTMENÜ: CLEAR COMPLETE LIST 16
6.10 CONTEXT MENU: RENUMBER MODBUS REGISTERS 16
6.11 CONTEXT MENU: RENUMBER KNX GROUPS 17
6.12 CONTEXT MENU: SORT MODBUS REGISTER 17
6.13 CONTEXT MENU: SORT KNX GROUP 18
6.14 CONTEXT MENU: FIND MODBUS REGISTER 18
6.15 CONTEXT MENU: FIND KNX GROUP 19
6.16 CONTEXT MENU: FIND COMMENT 19
6.17 CONTEXT MENU: EDIT ENTRY 20
6.18 TESTING THE CONFIGURATION 25

7 SAMPLE CONFIGURATIONS **26**

7.1 READING THE STATUS OF A KNX SWITCH 26
7.2 WRITING TO A KNX ACTUATOR 26
7.3 READING ANALOGUE KNX VALUES 27

8 FUNCTIONAL DESCRIPTION **29**

8.1 ASCII PROTOCOL DESCRIPTION 29
8.1.1 Overview 29
8.1.2 Communication sequence 30
8.1.3 Request VERSION 30
8.1.4 Request module TYPE 31
8.1.5 Table of all ASCII commands 32
8.1.6 The Configuration Line 35
8.1.7 The Add Configuration Line 37
8.2 MODBUS – REGISTER DESCRIPTION 38
8.2.1 Table of holding registers 38
8.2.2 MODBUS datatype storage and common pitfalls 38

9 SPECIFICATIONS **41**

9.1 DIMENSIONS 41
9.2 3D DRAWING 42

3 IMPORTANT SECURITY NOTES

Danger to life through electrical current!
Only skilled personal trained in electro-engineering should perform the described steps in the following chapters. Please observe the country specific rules and standards. Do not perform any electrical work while the device is connected to power.

Pay attention to the following rules:

1. Disconnect the system from power
2. Secure the system against automatic power on
3. Check that the system is de-energized
4. Cover other energized parts of the system

IMPORTANT HINT: Before you start with the installation and the initial setup of the device, you have to read this document and the attached installation guide and the actual manual for the device very carefully. You have to follow all the herein given information very accurate!

- ❑ Only authorized and qualified personnel are allowed to install and setup the device!
- ❑ The connection of the device must be done in de-energized state!
- ❑ Do not perform any electrical work while the device is connected to power!
- ❑ Disable and secure the system against any automatic restart or power on procedure!
- ❑ The device must be operated with the defined voltage level!
- ❑ Supply voltage jitters must not exceed the technical specifications and tolerances given in the technical manuals for the product. If you do not obey this issue, the proper performance of the device cannot be guaranteed. This can lead to fail functions of the device and in worst case to a complete breakdown of the device!
- ❑ You have to obey the current EMC regulations for wiring!
- ❑ All signal, control and supply voltage cables must be wired in a way, that no inductive or capacitive interference or any other severe electrical noise disturbance may interfere with the device. Wrong wiring can lead to a malfunction of the device!
- ❑ For signal or sensor cables you have to use shielded cables, to avoid damages through induction!
- ❑ You have to obey and to apply the current safety regulations given by the ÖVE, VDE, the countries, their control authorities, the TÜV or the local energy supply company!
- ❑ Obey country-specific laws and standards!
- ❑ The device must be used for the intended purpose of the manufacturer!
- ❑ No warranties or liabilities will be accepted for defects and damages resulting from improper or incorrect usage of the device!
- ❑ Subsequent damages, which results from faults of this device, are excluded from warranty and liability!
- ❑ Only the technical data, wiring diagrams and operation instructions, which are part to the product shipment are valid!
- ❑ The information on our homepage, in our datasheets, in our manuals, in our catalogues or published by our partners can deviate from the product documentation and is not necessarily always actual, due to constant improvement of our products for technical progress!
- ❑ In case of modification of our devices made by the user, all warranty and liability claims are lost!
- ❑ The installation has to fulfil the technical conditions and specifications (e.g. operating temperatures, power supply, …) given in the devices documentation!
- ❑ Operating our device close to equipment, which do not comply with EMC directives, can influence the functionality of our device, leading to malfunction or in worst case to a breakdown of our device!

- Our devices must not be used for monitoring applications, which solely serve the purpose of protecting persons against hazards or injury, or as an emergency stop switch for systems or machinery, or for any other similar safety-relevant purposes!
- Dimensions of the enclosures or enclosures accessories may show slight tolerances on the specifications provided in these instructions!
- Modifications of this documentation is not allowed!
- In case of a complaint, only complete devices returned in original packing will be accepted!

4 General Information

With the RESI-KNX-MODBUS and RESI-KNX-ASCII gateways, the KNX can be integrated in almost every system with a RS232 or RS485 interface and a MODBUS/RTU master protocol or serial ASCII text based protocol. The gateway is a serial interface for connection to the KNX with an integrated 2 wire KNX bus-coupler. The time-critical KNX communication is done from the gateway itself. The gateway is configured with our MODBUSConfigurator software and maps the incoming KNX telegrams to MODBUS holding registers. When the host writes to a MODBUS holding register, the gateway generates the corresponding KNX telegram. When the gateway receives a KNX telegram, it maps and converts the incoming data to the specific MODBUS holding registers for readout through a host. For media control systems like Crestron®, AMX® or Control4® our gateway series RESI-KNX-ASCII offers a second protocol with simple ASCII commands.

- Easy integration of the KNX in any system
- MODBUS/RTU slave protocol
- Only RESI-KNX-ASCII: Additional commands with plain ASCII texts
- KNX and host interface are galvanically isolated
- Supports all 32768 group addresses
- Supports all DPT types
- Integrated KNX bus-coupler
- Host interfaces: RS232, 9600 to 57600 bps, 8 data bits, no or even parity, 1 stop bit
- Power supply with 24VDC
- Power consumption <0.5W
- Mountable onto a EN50022 DIN rail

Type	Description	Voltage	Power	Weight
RESI-KNX-MODBUS	KNX to MODBUS/RTU slave gateway with RS232 or RS485 interface for all 32768 KNX groups and max. 150 configuration entries	24 V=	<0.5W	55 g
RESI-KNX-ASCII	KNX to MODBUS/RTU slave gateway with additional ASCII text protocol and RS232 or RS485 interface for all 32768 KNX groups and max. 150 configuration entries	24 V=	<0.5W	55 g

Technical data			
Power supply			
Supply voltage	24 V= +/-10%	Storage temperature	-20...85 °C
Power LED indicator	Yes	Operation temperature	0...60°C
Power consumption	<0.5W	Humidity	25...90 % rH not condensing
		Protection class	IP20 (EN 60529)
		Dimensions LxWxH	17,5mm x 90mm x 58mm
		Weight	55g
		Mounting	on DIN EN50022 rail
MODBUS/RTU-KNX mapping			
Maximum entries	150		
MODBUS/RTU protocol			
Protocol	MODBUS/RTU slave		
Type	RS232 or RS485		
Baud rate	9600 to 57600/8/N or E/1		
Cable connection	Via clamps		
LED indicator	Yes		
Galvanic insulation to the KNX interface	Yes		
ASCII protocol interface			
Protocol	ASCII plain text		
Type	RS232 or RS485		
Baud rate	9600 to 57600/8/N or E/1		
Cable connection	Via clamps		
LED indicator	Yes		
Galvanic insulation to the KNX interface	Yes		
KNX bus interface			
Protocol	KNX		
Baud rate	9600Bits/s		
Cable connection	Via clamps		
Galvanic insulation to serial interface	Yes		
LED indicator	Yes		
Clamps			
Clamp wire cross section	Max. 1,5 mm²		
Tightening torque	Max. 0.5Nm	**CE conformity**	Yes

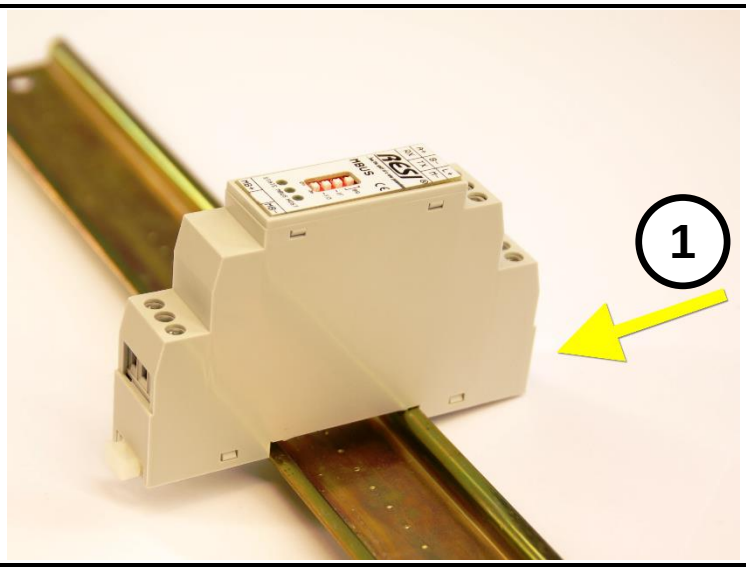
IT Accessories	
MODBUSConfigurator	Use our free software to configure and test the mapping between KNX and MODBUS register.

5 Mounting and Connections

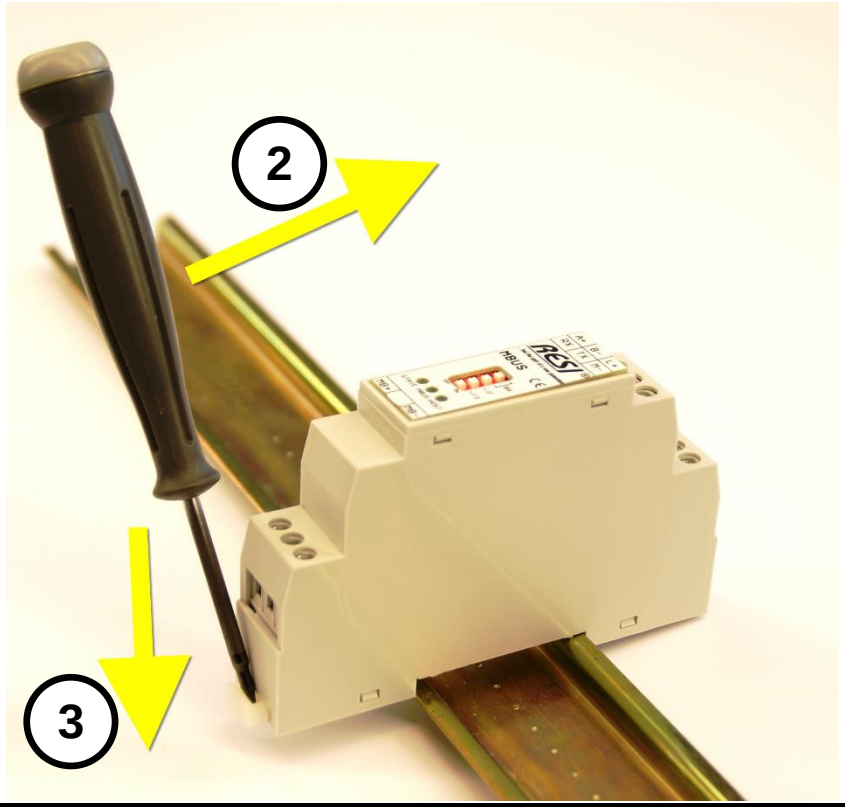
5.1 Assembling

Our RESI-KNX-MODBUS/ASCII gateways are designed for mounting on a 35mm DIN-EN50022 rail. Please note, that there are symbol photos used in the mounting pictures below.

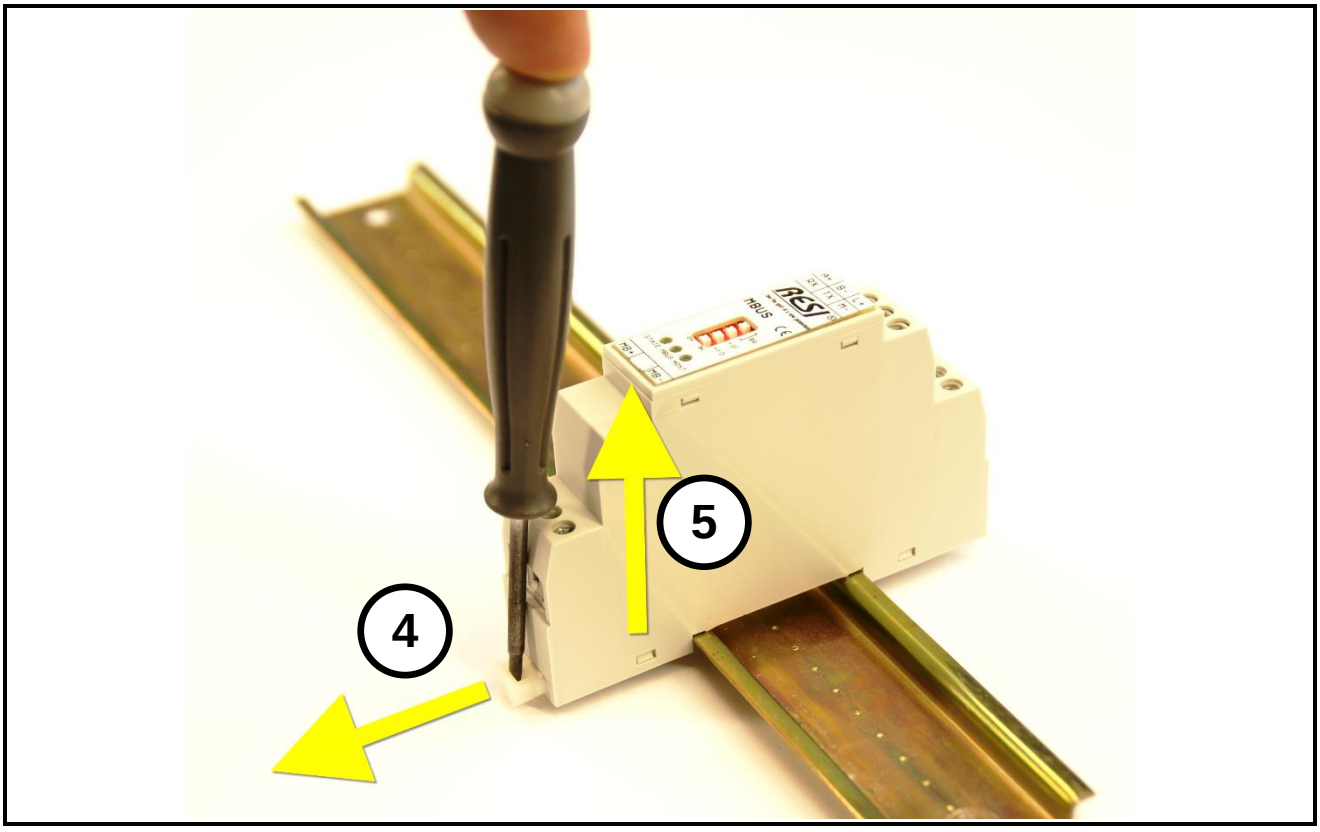
At first, put the gateway with the top side on the DIN rail (1).



Then open the clamp lever on the bottom side with a screw driver (2) and press the device on the DIN rail (3). Release the clamp lever. The module is now placed correctly on the DIN rail.



To dismount the module from the DIN rail first open the clamp lever with a screwdriver on the bottom side (4). Hold the clamp lever opened while you lift the module from the DIN rail (5). Then remove the gateway from the bar with while pulling it on the top side.



5.2 Clamps and LEDs

	RESI-KNX-MODBUS/ RESI-KNX-ASCII
L+ M-	Power supply L+: 24 V= M-: Ground
A B	RS485 Modbus/RTU slave or ASCII text interface A: DATA+ B: DATA-
RX TX M-	RS232 Modbus/RTU slave or ASCII text interface RX: serial receive TX: serial transmit M-: Ground for RS232
K+ K-	Interface to KNX bus system K+: KNX+ bus wire (red) K-: KNX- bus wire (black)
STATE	State-LED, flashes slowly, when gateway is ok and the KNX is connected. Flashes fast, if the gateway or the KNX connection has an error
KNX	KNX activity LED, this LED is on while the gateway send or receives KNX telegrams
HOST	HOST-LED, flashes, when host sends/receive telegrams

Table: Description of connectors and LEDs of the RESI-KNX-MODBUS/RESI-KNX-ASCII gateway

5.3 DIP switch settings

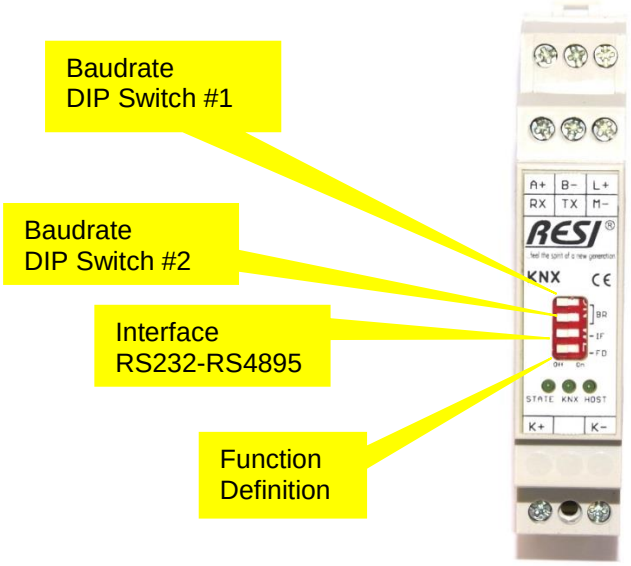


Illustration: Description of the DIP switch settings and LED status displays

DIP Switch	RESI-KNX-MODBUS/RESI-KNX-ASCII
Baudrate BR	Use DIP Switches 1+2 to select baud rate: OFF OFF: 9600Bd ON OFF: 19200Bd OFF ON: 38400Bd ON ON: 57600Bd HINT: The correct parity (NONE, EVEN or ODD) is configured with the PC software, not via DIP switches!
Interface IF	Select serial interface for the host communcation OFF=RS232 ON=RS485
Function Definition FD	Selects special Functions OFF=Use Modbus RTU slave bus address from FLASH memory ON=Use always Slave address 255

Table: Description of DIP switch functions

5.4 Wiring diagram

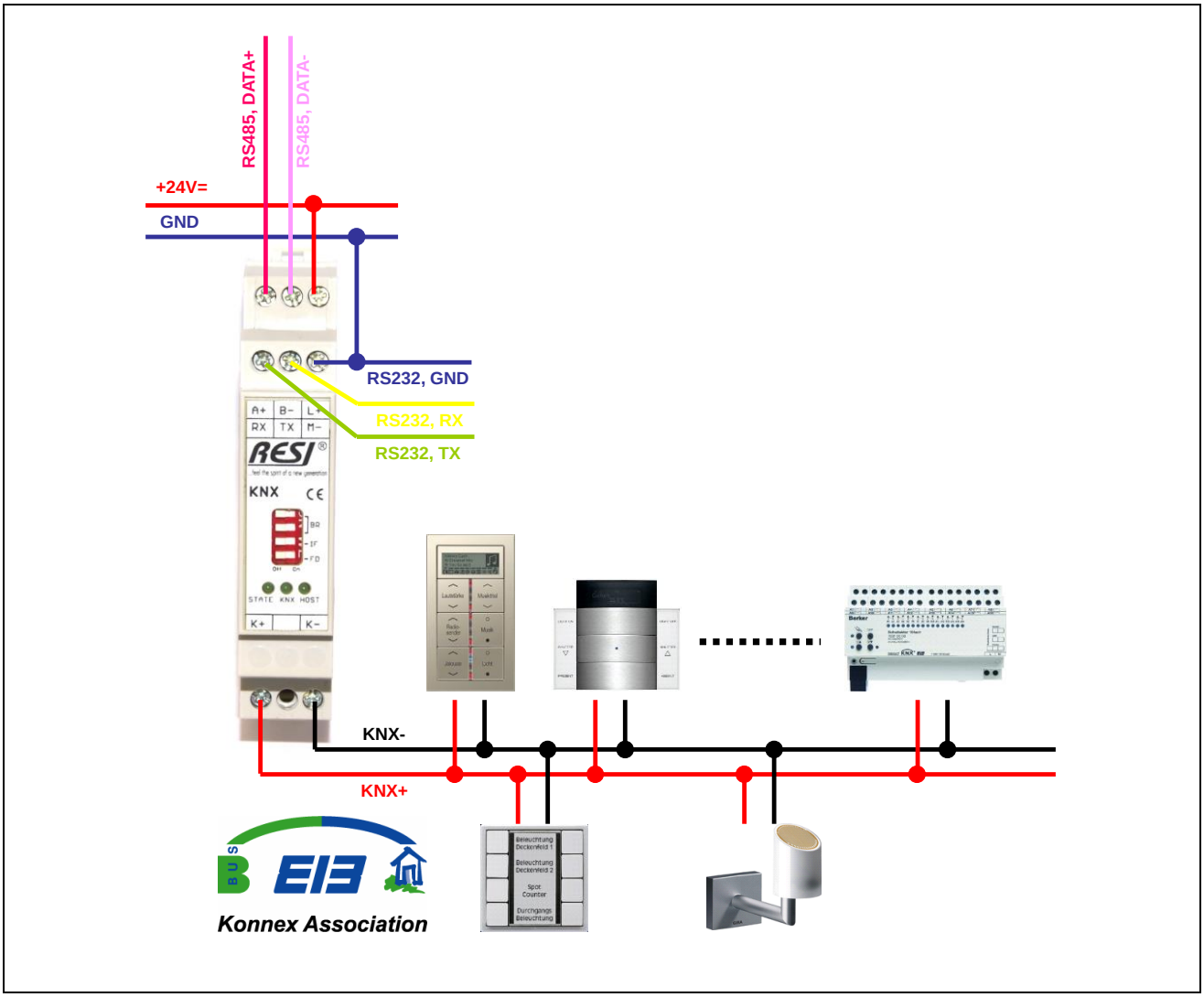


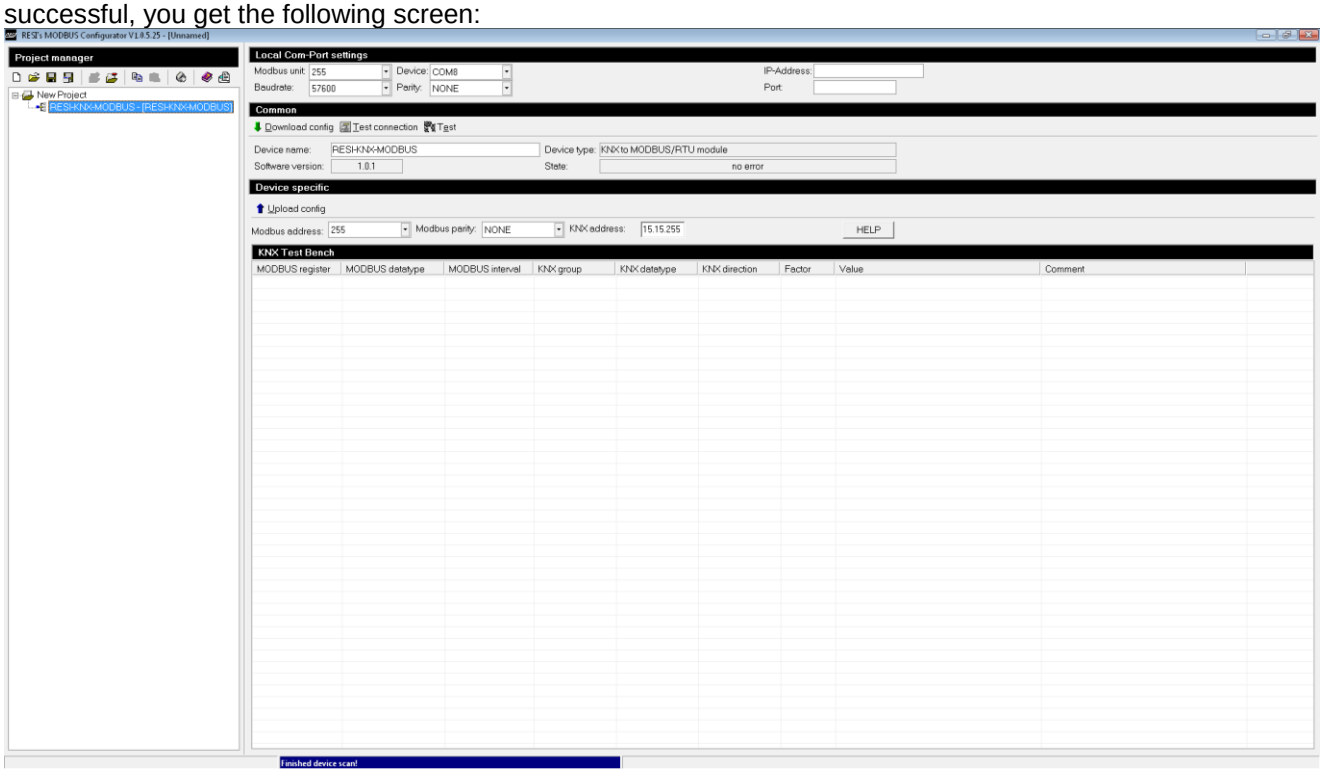
Illustration: wiring diagram of gateway

6 Configuration with RESI MODBUSConfigurator software

Download our free software from our homepage www.RESI.cc and install it on your computer.

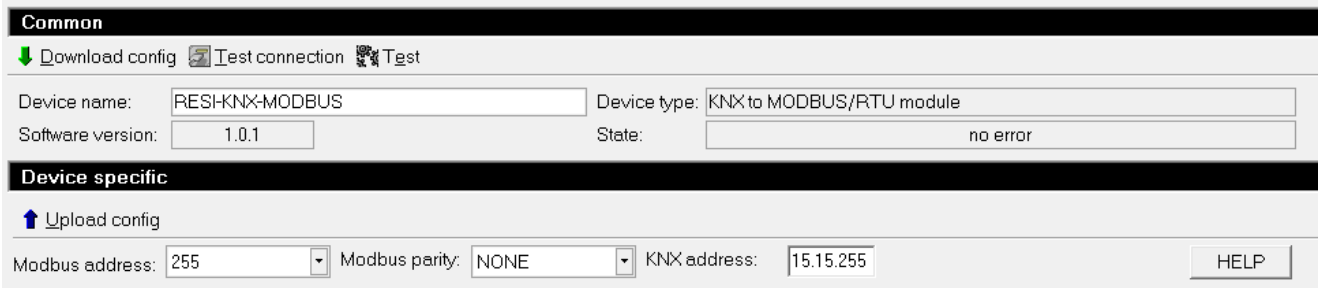
6.1 Establish a connection

Establish a connection between the module and our software tool RESI MODBUSConfigurator. If this is successful, you get the following screen:



6.2 Basic functions

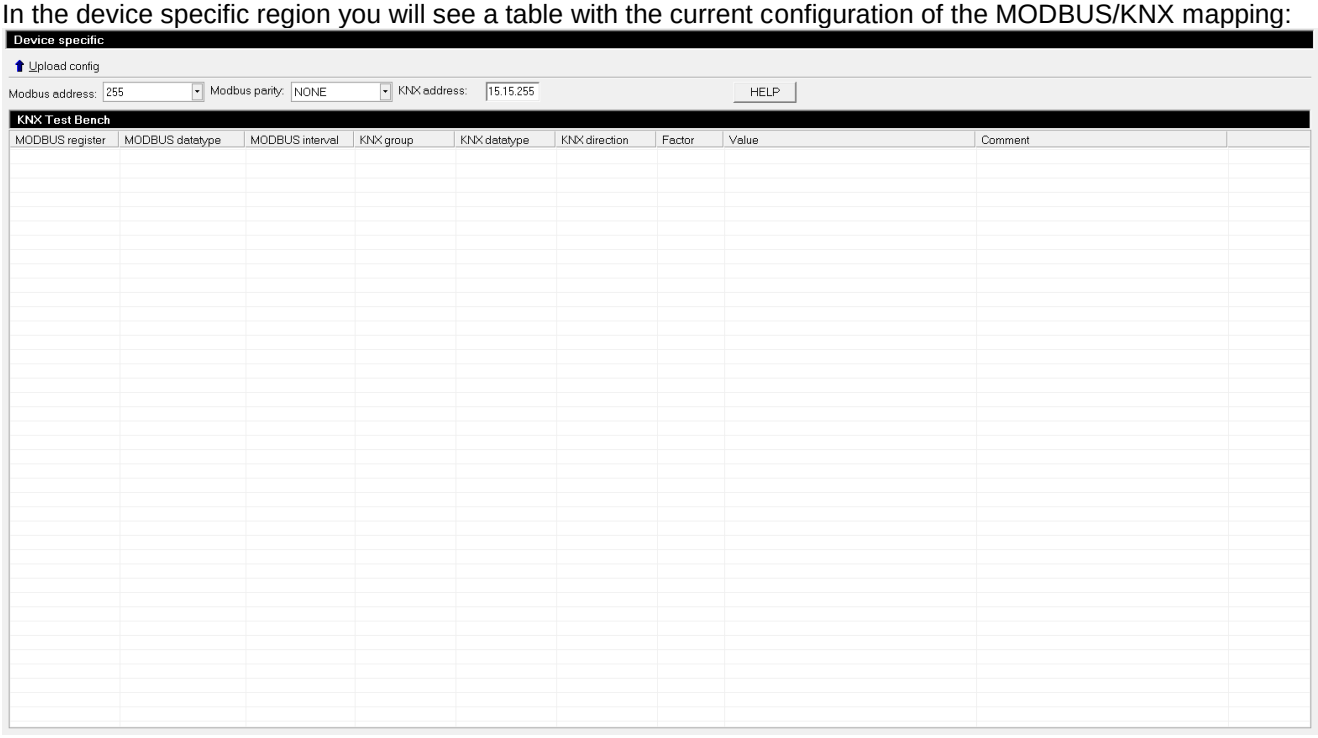
As you can see, you can do individual setup for the gateway in the above area of the screen:



- Button “Download config”: If you change the MODBUS/RTU slave address, the MODBUS parity or KNX address or if you change the KNX mapping, you have to download the new configuration to the gateway to activate the changes.
- Button “Upload config”: With this button you can upload the complete mapping of the converter into the software. But remember, the comments are not stored into the gateway, so this information is lost, if you upload the mapping from a gateway!
- Button “Test connection”: This button tests, if the software can communicate with the gateway or not.
- Button “Test”: This button activates/deactivates a test function, which will show all current contents of the configured MODBUS registers in the converter. In this test mode, you can also write to MODBUS/RTU holding registers and generate a KNX telegram on the KNX bus. The software polls every 5 seconds all configured MODBUS registers.

6.3 The configuration table

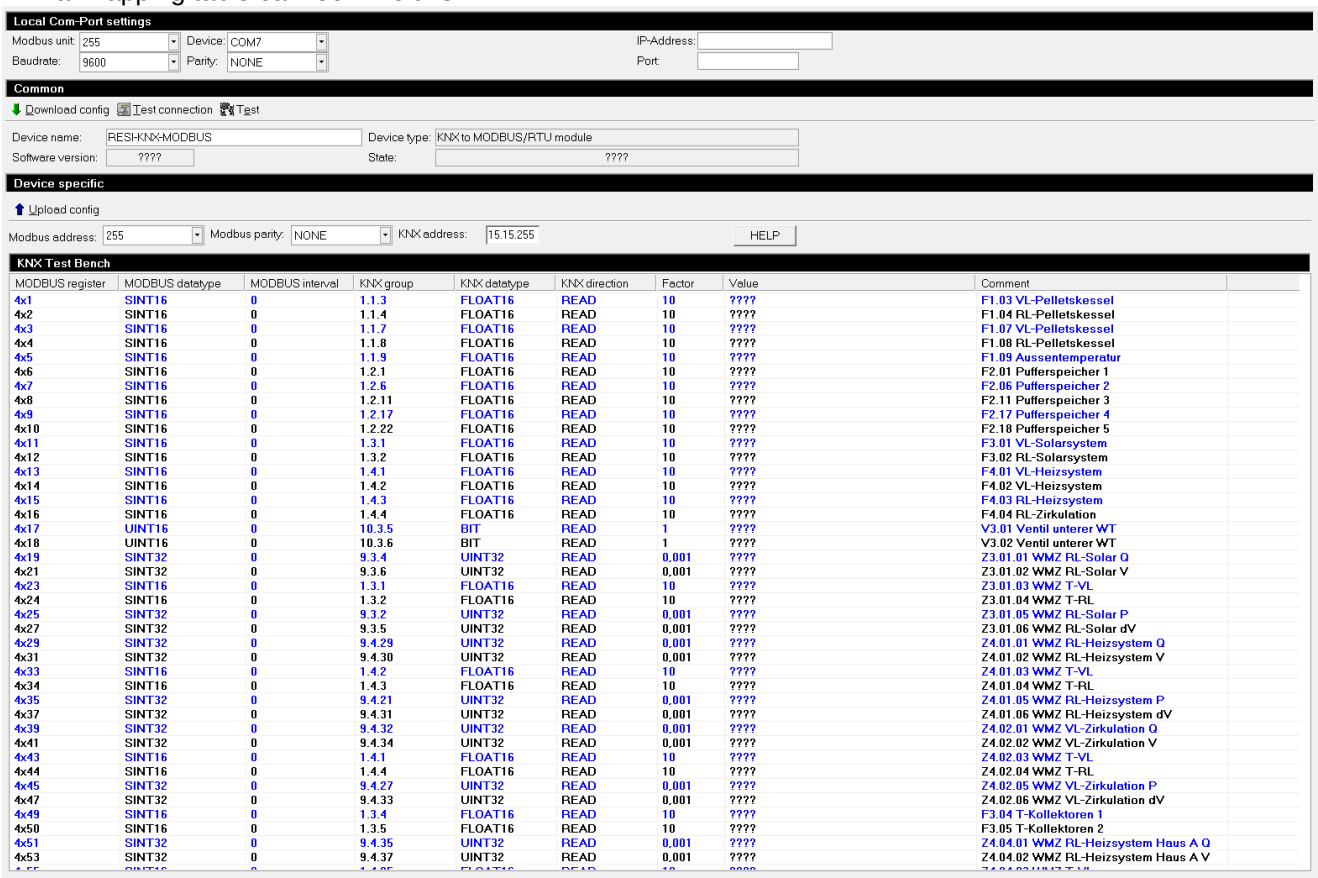
In the device specific region you will see a table with the current configuration of the MODBUS/KNX mapping:



A mapping entry consists out of the following entries:

- **MODBUS register**: The number of the single holding register or the start index of the holding registers, if more than one register is used, into which the incoming KNX data is stored or from which the outgoing KNX value is read before sending KNX data.
- **MODBUS datatype**: The data type for the MODBUS registers. It defines how the gateway converts KNX data into MODBUS data and how many MODBUS registers are used to store the KNX data.
- **MODBUS interval**: This is for future use and defines the time interval in seconds to request KNX data from the KNX bus automatically. At the moment this feature is not used.
- **KNX group**: this defines the KNX group address, which is used to send/receive KNX data for this MODBUS registers.
- **KNX datatype**: This defines the KNX datatype, which is used to send/receive data with the specific KNX group on the KNX bus.
- **KNX direction**: This defines in which direction the communication is done: You can read, write or read/write data from/to the KNX bus.
- **Factor**: This defines a factor which is used to multiply incoming KNX data before the data is stored into the MODBUS registers. For outgoing KNX telegrams the data of the MODBUS registers is divided by this factor, before sending to the KNX bus. A zero value defines unused.
- **Comment**: This defines a user specific comment for this mapping line. This is only for documentation reasons and is not stored into the gateway. It is only stored on the PC if you save your project. If you upload a configuration from a gateway into the software, this comment is lost!

A final mapping table can look like this:



Local Com-Port settings

Modbus unit: 255 Device: COM7 IP-Address:
Baudrate: 9600 Parity: NONE Port:

Common

Download config | Test connection | Test

Device name: RESI-KNX-MODBUS Device type: KNX to MODBUS/RTU module
Software version: ???? State: ????

Device specific

Upload config

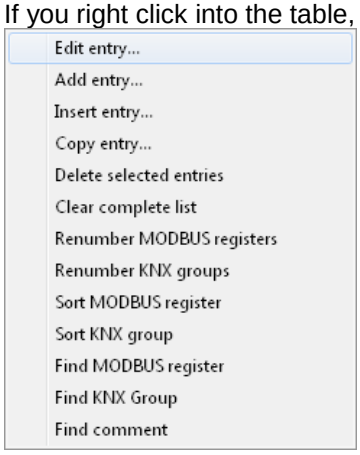
Modbus address: 255 Modbus parity: NONE KNX address: 15.15.255 HELP

KNX Test Bench

MODBUS register	MODBUS datatype	MODBUS interval	KNX group	KNX datatype	KNX direction	Factor	Value	Comment
4x1	SINT16	0	1.1.3	FLOAT16	READ	10	????	F1.03 VL-Pelletskessel
4x2	SINT16	0	1.1.4	FLOAT16	READ	10	????	F1.04 RL-Pelletskessel
4x3	SINT16	0	1.1.7	FLOAT16	READ	10	????	F1.07 VL-Pelletskessel
4x4	SINT16	0	1.1.8	FLOAT16	READ	10	????	F1.08 RL-Pelletskessel
4x5	SINT16	0	1.1.9	FLOAT16	READ	10	????	F1.09 Aussentemperatur
4x6	SINT16	0	1.2.1	FLOAT16	READ	10	????	F2.01 Pufferspeicher 1
4x7	SINT16	0	1.2.6	FLOAT16	READ	10	????	F2.06 Pufferspeicher 2
4x8	SINT16	0	1.2.11	FLOAT16	READ	10	????	F2.11 Pufferspeicher 3
4x9	SINT16	0	1.2.17	FLOAT16	READ	10	????	F2.17 Pufferspeicher 4
4x10	SINT16	0	1.2.22	FLOAT16	READ	10	????	F2.18 Pufferspeicher 5
4x11	SINT16	0	1.3.1	FLOAT16	READ	10	????	F3.01 VL-Solarsystem
4x12	SINT16	0	1.3.2	FLOAT16	READ	10	????	F3.02 RL-Solarsystem
4x13	SINT16	0	1.4.1	FLOAT16	READ	10	????	F4.01 VL-Heizsystem
4x14	SINT16	0	1.4.2	FLOAT16	READ	10	????	F4.02 VL-Heizsystem
4x15	SINT16	0	1.4.3	FLOAT16	READ	10	????	F4.03 RL-Heizsystem
4x16	SINT16	0	1.4.4	FLOAT16	READ	10	????	F4.04 RL-Zirkulation
4x17	UINT16	0	10.3.5	BIT	READ	1	????	V3.01 Ventil unterer WT
4x18	UINT16	0	10.3.6	BIT	READ	1	????	V3.02 Ventil unterer WT
4x19	SINT32	0	9.3.4	UINT32	READ	0,001	????	Z3.01.01 WMZ RL-Solar Q
4x21	SINT32	0	9.3.6	UINT32	READ	0,001	????	Z3.01.02 WMZ RL-Solar V
4x23	SINT16	0	1.3.1	FLOAT16	READ	10	????	Z3.01.03 WMZ T-VL
4x24	SINT16	0	1.3.2	FLOAT16	READ	10	????	Z3.01.04 WMZ T-RL
4x25	SINT32	0	9.3.2	UINT32	READ	0,001	????	Z3.01.05 WMZ RL-Solar P
4x27	SINT32	0	9.3.5	UINT32	READ	0,001	????	Z3.01.06 WMZ RL-Solar dV
4x29	SINT32	0	9.4.29	UINT32	READ	0,001	????	Z4.01.01 WMZ RL-Heizsystem Q
4x31	SINT32	0	9.4.30	UINT32	READ	0,001	????	Z4.01.02 WMZ RL-Heizsystem V
4x33	SINT16	0	1.4.2	FLOAT16	READ	10	????	Z4.01.03 WMZ T-VL
4x34	SINT16	0	1.4.3	FLOAT16	READ	10	????	Z4.01.04 WMZ T-RL
4x35	SINT32	0	9.4.21	UINT32	READ	0,001	????	Z4.01.05 WMZ RL-Heizsystem P
4x37	SINT32	0	9.4.31	UINT32	READ	0,001	????	Z4.01.06 WMZ RL-Heizsystem dV
4x39	SINT32	0	9.4.32	UINT32	READ	0,001	????	Z4.02.01 WMZ VL-Zirkulation Q
4x41	SINT32	0	9.4.34	UINT32	READ	0,001	????	Z4.02.02 WMZ VL-Zirkulation V
4x43	SINT16	0	1.4.1	FLOAT16	READ	10	????	Z4.02.03 WMZ T-VL
4x44	SINT16	0	1.4.4	FLOAT16	READ	10	????	Z4.02.04 WMZ T-RL
4x45	SINT32	0	9.4.27	UINT32	READ	0,001	????	Z4.02.05 WMZ VL-Zirkulation P
4x47	SINT32	0	9.4.33	UINT32	READ	0,001	????	Z4.02.06 WMZ VL-Zirkulation dV
4x49	SINT16	0	1.3.4	FLOAT16	READ	10	????	F3.04 T-Kollektoren 1
4x50	SINT16	0	1.3.5	FLOAT16	READ	10	????	F3.05 T-Kollektoren 2
4x51	SINT32	0	9.4.35	UINT32	READ	0,001	????	Z4.04.01 WMZ RL-Heizsystem Haus A Q
4x53	SINT32	0	9.4.37	UINT32	READ	0,001	????	Z4.04.02 WMZ RL-Heizsystem Haus A V

6.4 The context menu

If you right click into the table, a local context menu will appear with the following entries:



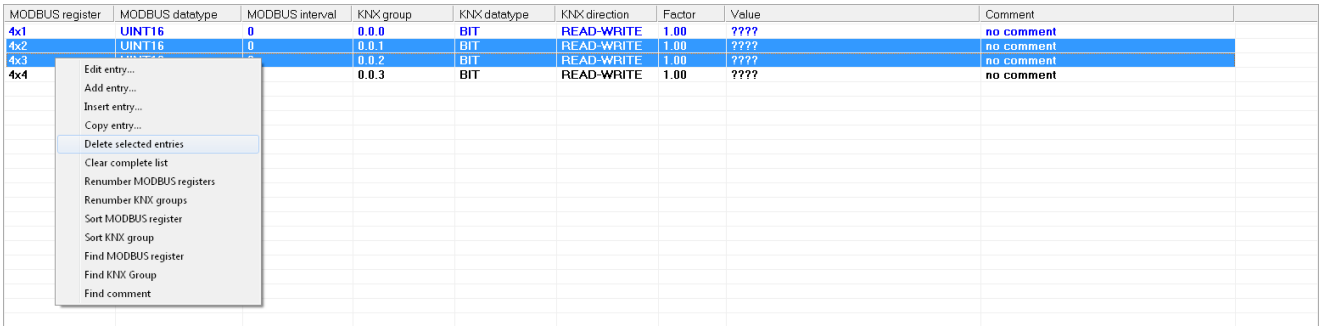
6.5 Context menu: Add entry

Entry “Add entry…”: If you select this item, a new empty configuration line is added to the configuration list.

MODBUS register	MODBUS datatype	MODBUS interval	KNX group	KNX datatype	KNX direction	Factor	Value	Comment
4x1	UINT16	0	0.0.0	BIT	READ-WRITE	1.00	????	no comment
4x2	UINT16	0	0.0.1	BIT	READ-WRITE	1.00	????	no comment

6.6 Context menu: Delete selected lines

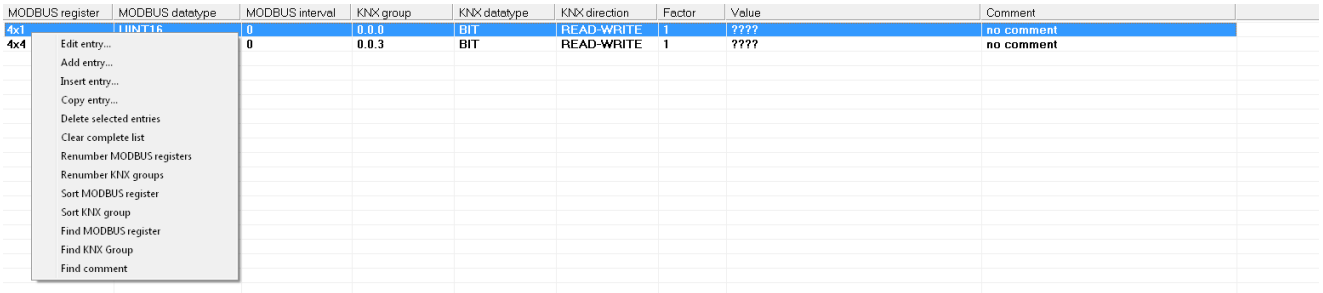
Entry “Delete selected lines…”: First select one or more lines in your configuration table. To select more than one line press and hold the STRG key or the SHIFT key, and then select other lines.



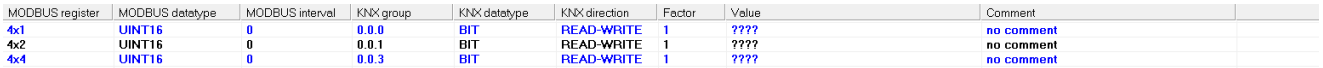
Then open the context menu and select the function “Delete selected lines…”. The system will delete the selected lines from the configuration list. The result will be like this:

6.7 Context menu: Insert entry

Entry “Insert entry…”: First select one or more lines. Then choose this function from the local context menu. The system now inserts a new configuration line directly after each selected line in the configuration.

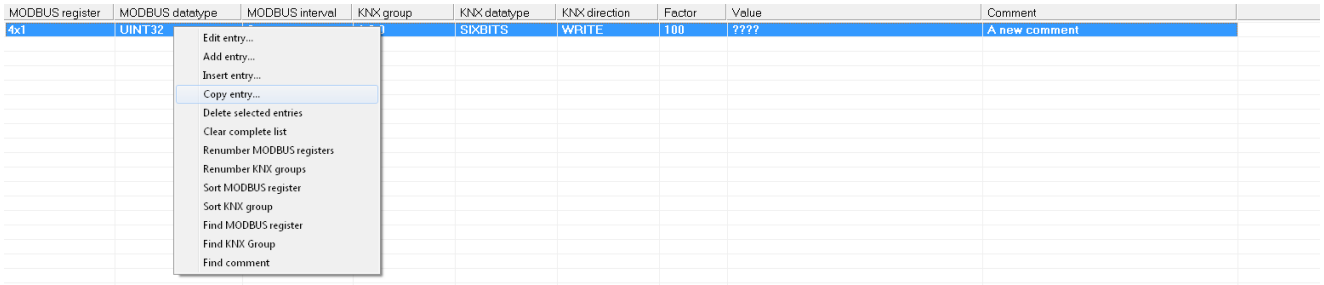


The result will be like this:



6.8 Context menu: Copy entry

Entry “Copy entry…”: First select one or more lines. Then choose this function from the local context menu:



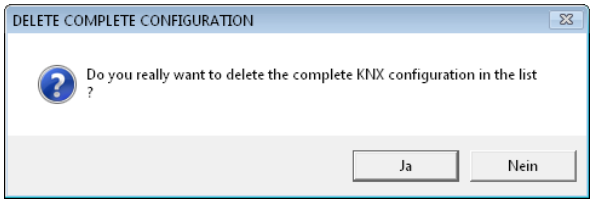
The system copies all selected lines and adds for each selected line a new entriy to the configuration. The result looks like this:

MODBUS register	MODBUS datatype	MODBUS interval	KNX group	KNX datatype	KNX direction	Factor	Value	Comment
4x1	UINT32	0	1.0.0	SIXBITS	WRITE	100	????	A new comment
4x3	UINT32	0	1.0.1	SIXBITS	WRITE	100	????	A new comment

As you will notice, the system auto increments the MODBUS/RTU register index depending to the configured MODBUS datatype. The same increment is done with the KNX group address.

6.9 Kontextmenü: Clear complete list

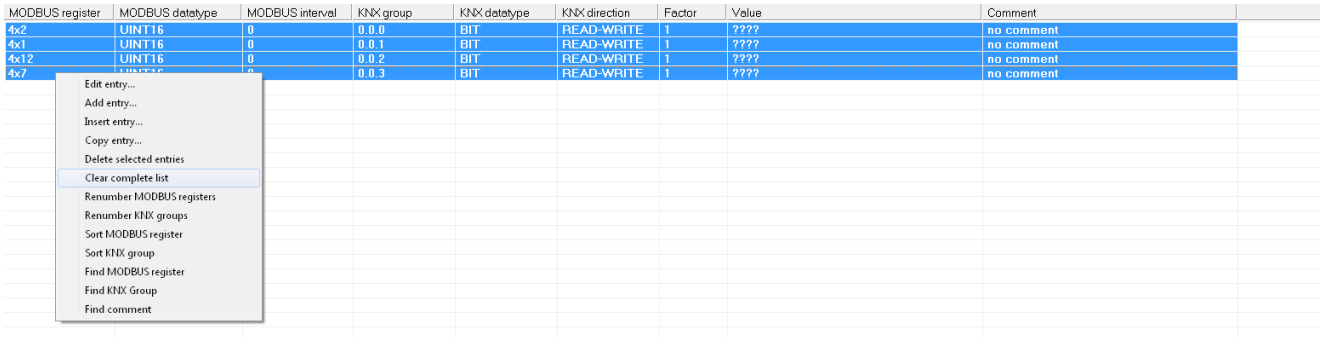
Entry “Clear complete list”: After selecting this function the system asks the following question:



If you answer with YES, all entries of your configuration will be deleted forever! The answer NO cancels this function.

6.10 Context menu: Renumber MODBUS registers

Entry “Renumber MODBUS registers”: First select the lines you want to renumber, then select this function from the local context menu:

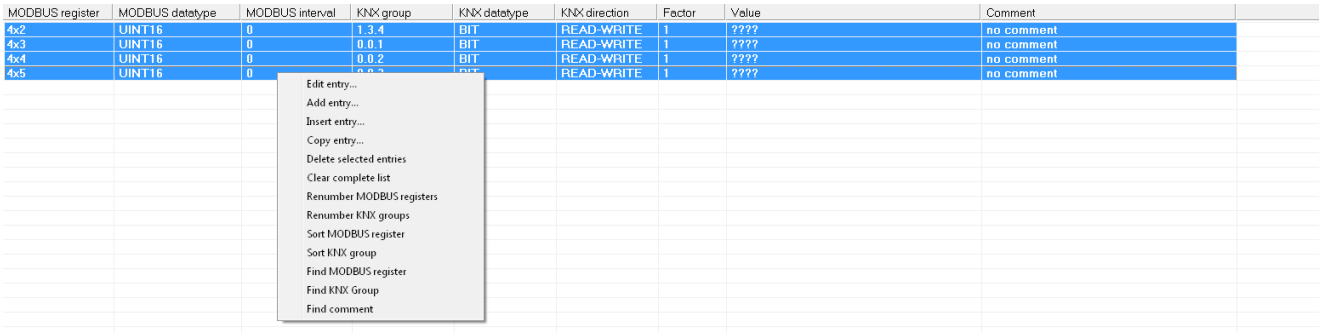


The starting index of the MODBUS register of the first selected line is used for the first entry. The next lines are renumbered depending on the MODBUS datatype of each line. The result will look like this:

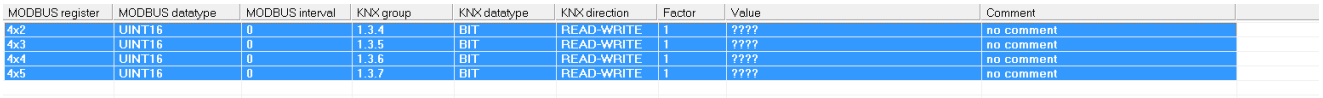
MODBUS register	MODBUS datatype	MODBUS interval	KNX group	KNX datatype	KNX direction	Factor	Value	Comment
4x2	UINT16	0	0.0.0	BIT	READ-WRITE	1	????	no comment
4x3	UINT16	0	0.0.1	BIT	READ-WRITE	1	????	no comment
4x4	UINT16	0	0.0.2	BIT	READ-WRITE	1	????	no comment
4x5	UINT16	0	0.0.3	BIT	READ-WRITE	1	????	no comment

6.11 Context menu: Renumber KNX groups

Entry “Renumber KNX groups”: First select the lines you want to renumber. Then select this function from the local context menu.

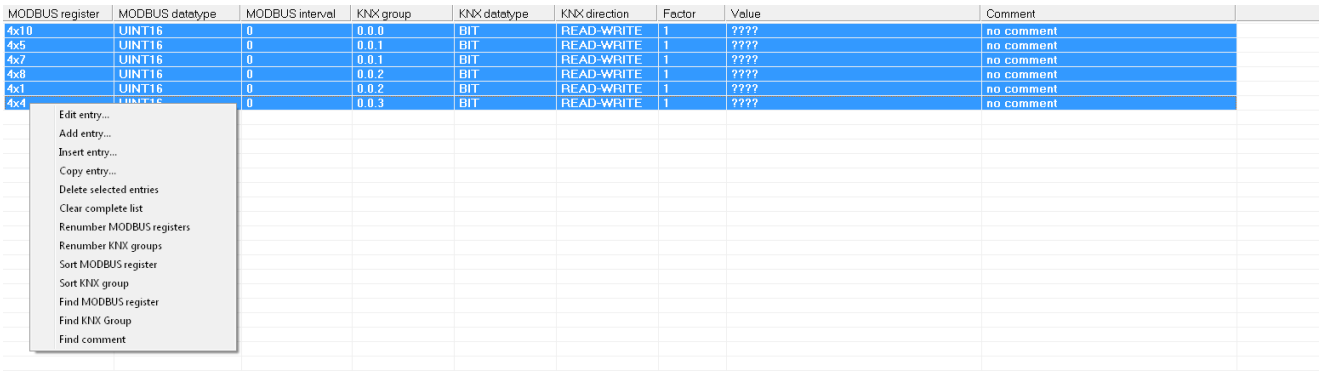


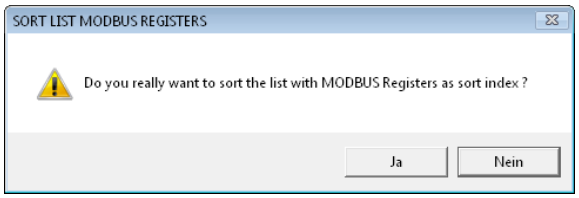
The KNX group address of the first selected line is used for the first entry. All selected lines are now renumbered with an ascending KNX group address. The result will look like this:



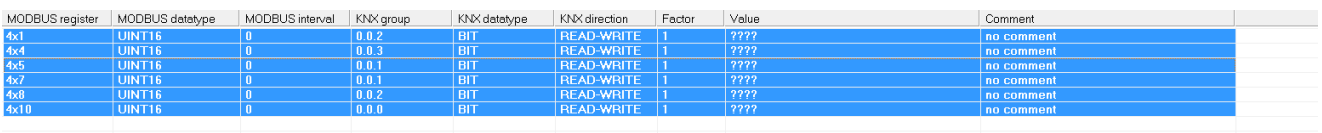
6.12 Context menu: Sort MODBUS register

Entry “Sort MODBUS register”: First select one or more lines you want to sort. Then select this function from the local context menu.



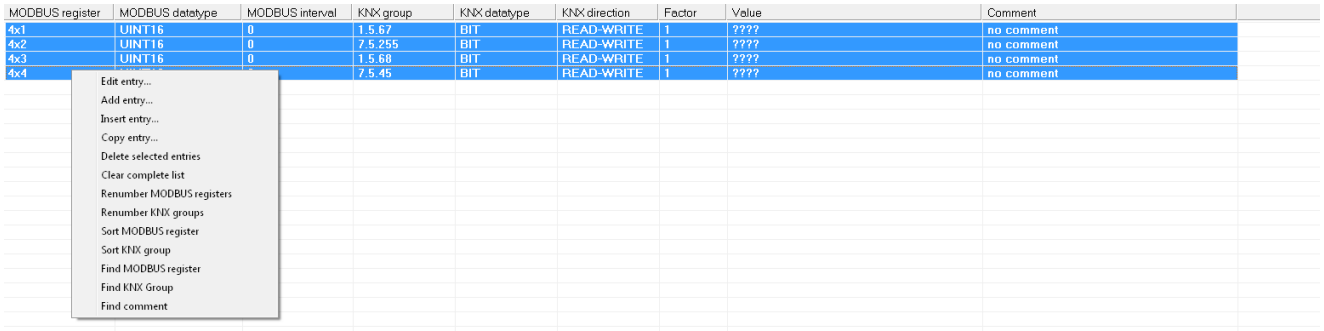


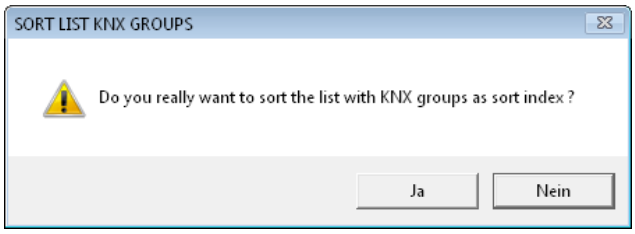
If you answer the above question with YES, the system sorts the select line using the MODBUS address as a sort key in ascending order. The result will look like this:



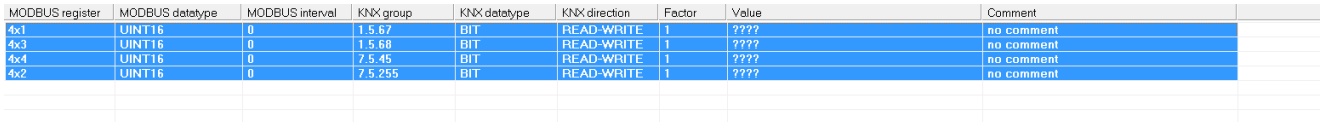
6.13 Context menu: Sort KNX group

Entry “Sort KNX group”: After selecting one or more lines for sorting, choose this function from the local context menu:



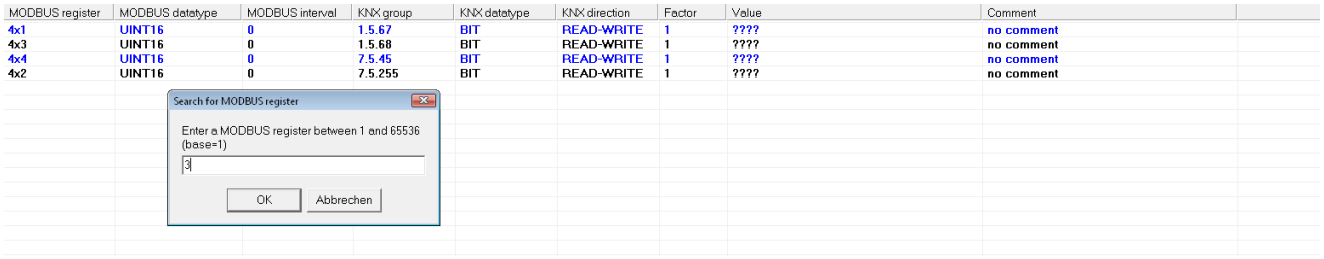


If you answer the above question with YES, the system sorts the selected lines using the KNX group address as a sort index in ascending order. The result will look like this:

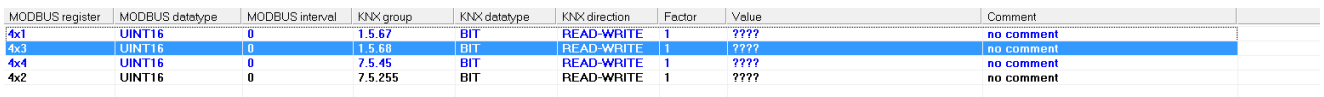


6.14 Context menu: Find MODBUS register

Entry “Find MODBUS register”: After selecting this function from the local context menu, an input window will appear. Enter a valid MODBUS register index and press the OK button.

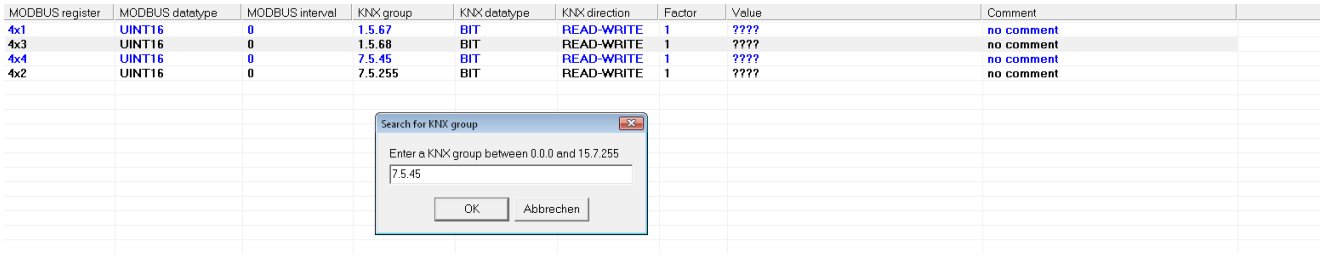


The system will now select all lines, in which the MODBUS register matches the entered number. The result will look like this:

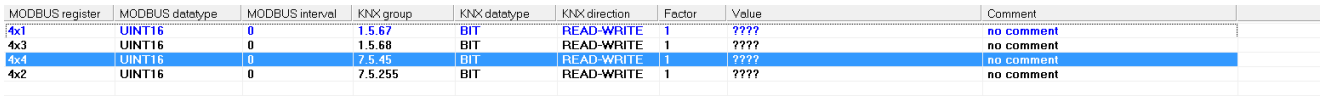


6.15 Context menu: Find KNX group

Entry “Find KNX group”: After selecting this function from the local context menu, an input window will appear:

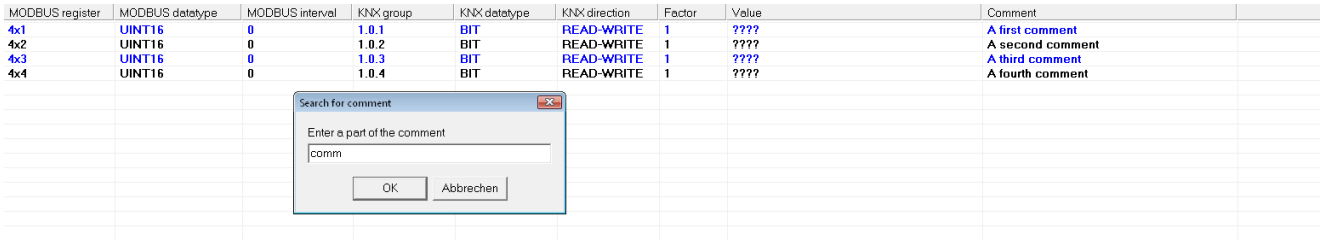


Enter a valid KNX group address. The system now selects all lines in the list, which matches with the entered KNX group address. The result will look like this:

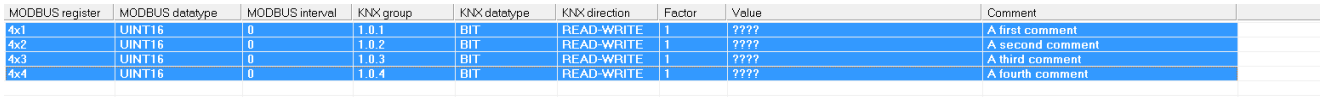


6.16 Context menu: Find comment

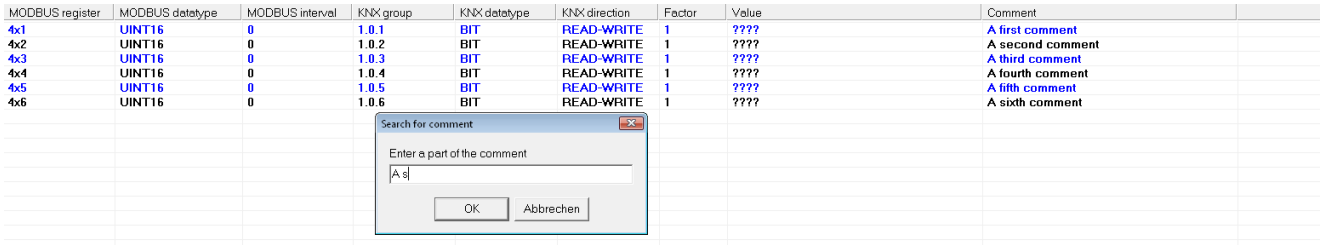
Entry “Find comment”: After selecting this function from the local context menu, an input window will appear. Enter a text part of the desired comment and select the OK button. The system will mark all lines, in which a text part of the comment matches to the entered text.



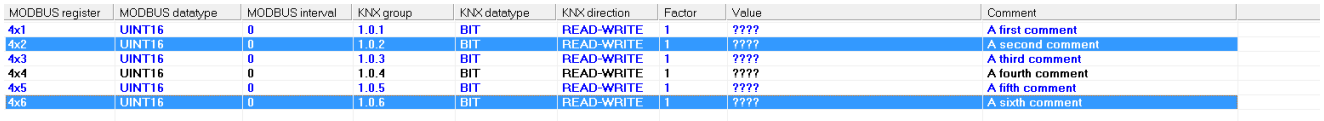
The result will look like this:



Another example:



The result looks like:

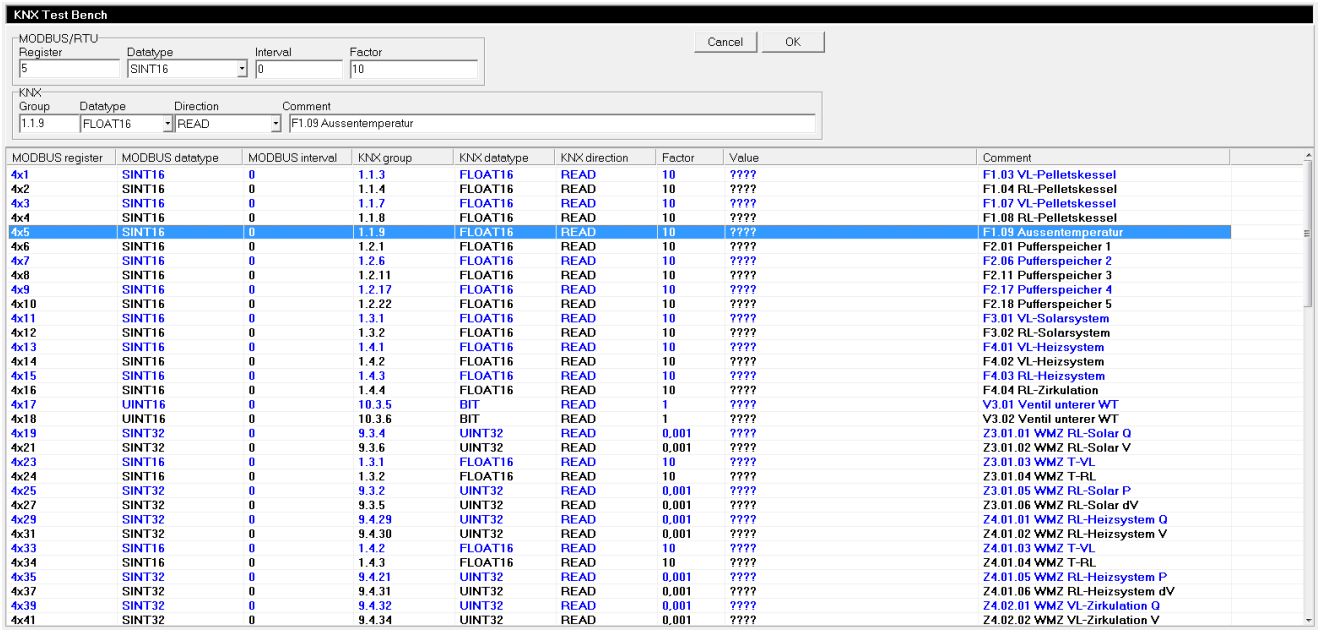


6.17 Context menu: Edit entry

Entry “Edit entry…”: After selecting a line and choosing this function from the local context menu or after a double click onto a line in the table, the below screen will appear. In the upper region an edit area is displayed with the current contents of the selected line.

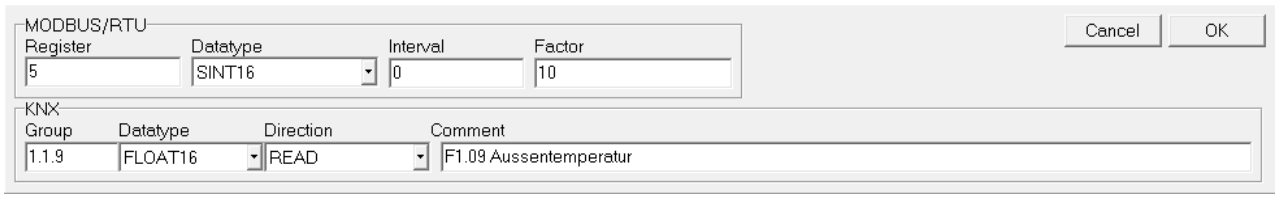
- Button “Cancel”: Selecting this function will close the edit operation, and no changes will be done in the configuration line. The edit area will disappear.
- Button “OK”: Selecting this function will update the selected configuration line with the altered data and close the edit function. The edit areas will disappear.

HINT: Don’t forget, that you must download the new configuration to the gateway, before the changes are used by the converter!



MODBUS register	MODBUS datatype	MODBUS interval	KNX group	KNX datatype	KNX direction	Factor	Value	Comment
4x1	SINT16	0	1.1.3	FLOAT16	READ	10	????	F1.03 VL-Pelletskessel
4x2	SINT16	0	1.1.4	FLOAT16	READ	10	????	F1.04 RL-Pelletskessel
4x3	SINT16	0	1.1.7	FLOAT16	READ	10	????	F1.07 VL-Pelletskessel
4x4	SINT16	0	1.1.8	FLOAT16	READ	10	????	F1.08 RL-Pelletskessel
4x5	SINT16	0	1.1.9	FLOAT16	READ	10	????	F1.09 Aussentemperatur
4x6	SINT16	0	1.2.1	FLOAT16	READ	10	????	F2.01 Pufferspeicher 1
4x7	SINT16	0	1.2.6	FLOAT16	READ	10	????	F2.06 Pufferspeicher 2
4x8	SINT16	0	1.2.11	FLOAT16	READ	10	????	F2.11 Pufferspeicher 3
4x9	SINT16	0	1.2.17	FLOAT16	READ	10	????	F2.17 Pufferspeicher 4
4x10	SINT16	0	1.2.22	FLOAT16	READ	10	????	F2.18 Pufferspeicher 5
4x11	SINT16	0	1.3.1	FLOAT16	READ	10	????	F3.01 VL-Solarsystem
4x12	SINT16	0	1.3.2	FLOAT16	READ	10	????	F3.02 RL-Solarsystem
4x13	SINT16	0	1.4.1	FLOAT16	READ	10	????	F4.01 VL-Heizsystem
4x14	SINT16	0	1.4.2	FLOAT16	READ	10	????	F4.02 VL-Heizsystem
4x15	SINT16	0	1.4.3	FLOAT16	READ	10	????	F4.03 RL-Heizsystem
4x16	SINT16	0	1.4.4	FLOAT16	READ	10	????	F4.04 RL-Zirkulation
4x17	UINT16	0	10.3.5	BIT	READ	1	????	V3.01 Ventil unterer WT
4x18	UINT16	0	10.3.6	BIT	READ	1	????	V3.02 Ventil unterer WT
4x19	SINT32	0	9.3.4	UINT32	READ	0,001	????	Z3.01.01 WMZ RL-Solar Q
4x21	SINT32	0	9.3.6	UINT32	READ	0,001	????	Z3.01.02 WMZ RL-Solar V
4x23	SINT16	0	1.3.1	FLOAT16	READ	10	????	Z3.01.03 WMZ T-VL
4x24	SINT16	0	1.3.2	FLOAT16	READ	10	????	Z3.01.04 WMZ T-RL
4x25	SINT32	0	9.3.2	UINT32	READ	0,001	????	Z3.01.05 WMZ RL-Solar P
4x27	SINT32	0	9.3.5	UINT32	READ	0,001	????	Z3.01.06 WMZ RL-Solar dV
4x29	SINT32	0	9.4.29	UINT32	READ	0,001	????	Z4.01.01 WMZ RL-Heizsystem Q
4x31	SINT32	0	9.4.30	UINT32	READ	0,001	????	Z4.01.02 WMZ RL-Heizsystem V
4x33	SINT16	0	1.4.2	FLOAT16	READ	10	????	Z4.01.03 WMZ T-VL
4x34	SINT16	0	1.4.3	FLOAT16	READ	10	????	Z4.01.04 WMZ T-RL
4x35	SINT32	0	9.4.21	UINT32	READ	0,001	????	Z4.01.05 WMZ RL-Heizsystem P
4x37	SINT32	0	9.4.31	UINT32	READ	0,001	????	Z4.01.06 WMZ RL-Heizsystem dV
4x39	SINT32	0	9.4.32	UINT32	READ	0,001	????	Z4.02.01 WMZ VL-Zirkulation Q
4x41	SINT32	0	9.4.34	UINT32	READ	0,001	????	Z4.02.02 WMZ VL-Zirkulation V

Here is a zoom into the edit area:



The edit area is divided into two areas:

Area “MODBUS/RTU”: Here you will find all edit fields corresponding to the MODBUS/RTU holding register setup.

- Field “Register”: Enter a valid MODBUS holding register start index in the range of 1 to 65535. How many MODBUS holding registers are used for this configuration entry is defined by the configured MODBUS/RTU datatype.

- Field “Datatype”: Choose one of the possible datatypes from the drop down list. This datatype defines one the one side, how many MODBUS registers are used for the mapping (e.g. datatype UINT16 needs one register, datatype FLOAT32 needs two consecutive MODBUS registers). And on the other hand this data type defines the data representation in this holding registers (e.g. datatype FLOAT32 stores the upper 16 bits of the 32 bit float value in the first holding register and the lower 16 bits are stored in the next consecutive holding register. FLOAT32R stores the two 16 bit words of the 32 bit value in reverse order: The low word in the first register, the high word in the next register).

MODBUS DATATYPE	SIZE	WORD ORDER	DESCRIPTION
ERR	none	none	Defines an invalid configuration entry and is ignored by the gateway
UINT16	16 bits 1 register	none	Defines a 16 bit unsigned integer value in the range of 0 to 65535 or 0x0000 to 0xFFFF
SINT16	16 bits 1 register	none	Defines a 16 bit signed integer value in the range of -32768 to +32767 or 0x8000 to 0x7FFF
UINT32	32 bits 2 register	0:High Word 1:Low Word	Defines a 32 bit unsigned integer value in the range of 0 to 4.294.967.295 or 0x00000000 to 0xFFFFFFFF
SINT32	32 bits 2 register	0:High Word 1:Low Word	Defines a 32 bit signed integer value in the range of −2.147.483.648 to +2.147.483.647or 0x80000000 to 0x7FFFFFFF
UINT32R	32 bits 2 register	0:Low Word 1:High Word	Defines a 32 bit unsigned integer value in the range of 0 to 4.294.967.295 or 0x00000000 to 0xFFFFFFFF with reveres word order
SINT32R	32 bits 2 register	0:Low Word 1:High Word	Defines a 32 bit signed integer value in the range of −2.147.483.648 to +2.147.483.647or 0x80000000 to 0x7FFFFFFF with reverse word order
FLOAT32	32 bits 2 register	0:High Word 1:Low Word	Defines a 32 bit float value in the range of $\pm 1.4 \cdot 10^{-45}$ to $\pm 3.403 \cdot 10^{38}$. A mantissa of 23 bits and an exponent of 8 bits are used. The value can store 7 to 8 digits after the comma.
FLOAT32R	32 bits 2 register	0:Low Word 1:High Word	Defines a 32 bit float value in the range of $\pm 1.4 \cdot 10^{-45}$ to $\pm 3.403 \cdot 10^{38}$. A mantissa of 23 bits and an exponent of 8 bits are used. The value can store 7 to 8 digits after the comma. The two 16 bit words are stored in reverse order.
DOUBLE64	64 bits 4 register	0:Highest Word 1:Higer Word 2:Lower Word 3:Lowest Word	Defines a 64 bit float value in the range of $\pm 4.24 \cdot 10^{-324}$ to $\pm 1{,}798 \cdot 10^{308}$. A mantissa of 52 bits and an exponent of 11 bits are used. The value can store 15 to 16 digits after the comma.
DOUBLE64R	64 bits 4 register	0:Lowest Word 1:Lower Word 2:Higher Word 3:Highest Word	Defines a 64 bit float value in the range of $\pm 4.24 \cdot 10^{-324}$ to $\pm 1{,}798 \cdot 10^{308}$. A mantissa of 52 bits and an exponent of 11 bits are used. The value can store 15 to 16 digits after the comma. The four 16 bit words are stored in reverse order.
GENERIC	64 bits 4 register	0: FIRST and SECOND byte 1: THIRD and FOURTH byte 2: FIFTH and SIXTH byte 3: SEVENTH and EIGHT byte	Currently unused
STRING	64 bits 4 register	0: FIRST and SECOND byte 1: THIRD and FOURTH byte 2: FIFTH and SIXTH byte 3: SEVENTH and EIGHT byte	Currently unused

- Field “Interval”: This field has no function at the moment. The intention is in future releases to define the time in seconds for an automatic polling for the configured KNX group on the KNX bus.

- Field “Factor”: This field defines a float value which is used to convert the KNX and MODBUS values after receiving from and before sending to the KNX bus.

  In case of receiving a KNX group the formula is:
  MODBUS value= KNX value multiplied by Factor

  In case of sending a KNX group to the KNX bus, the formula is:
  KNX value=MODBUS value divided by Factor

  In case the KNX data type is GENERIC or STRING, this factor defines the start index, from which the data bytes are readout of the KNX telegram. A KNX telegram can hold up to 14 data bytes.

  A Factor of 0 is ignored from the gateway.

Area “KNX”: Here you will find all edit fields corresponding to the KNX group address mapping.

- Field “Group”: Here you can define the KNX group address for this configuration entry in the range of 0.0.0 to 15.7.255.

- Field “Datatype”: Here you can define the KNX datatype of the incoming or outgoing KNX telegram.

KNX DATATYPE	SIZE	DESCRIPTION
ERR	none	Defines an invalid configuration entry and is ignored by the gateway
BIT	1 bit	Defines a bit value in the range from 0 to 1 or 0x0 to 0x1. Often interpreted as OFF and ON.
TWOBITS	2 bits	Defines an integer value consisting out of two bits in the range from 0 to 3 or 0x0 to 0x3
FOURBITS	4 bits	Defines an integer value consisting out of four bits in the range from 0 to 15 or 0x0 to 0xF.
SIXBITS	6 bits	Defines an integer value consisting out of six bits in the range from 0 to 63 or 0x00 to 0x3F.
CHARACTER	8 bits	Defines one text character consisting out of eight bits in the range from 0 to 255 or 0x00 to 0xFF. Please refer to the KNX documentation, how the encoding of the text character is done by the KNX standard. The encoding can be done for ASCII characters or for ISO 8859.1 characters.
UINT8	8 bits	Defines a 8 bit unsigned integer value in the range of 0 to 255 or 0x00 to 0xFF
SINT8	8 bits	Defines a 8 bit signed integer value in the range of -128 to +127 or 0x80 to 0x7F
UINT16	16 bits	Defines a 16 bit unsigned integer value in the range of 0 to 65535 or 0x0000 to 0xFFFF
SINT16	16 bits	Defines a 16 bit signed integer value in the range of -32768 to +32767 or 0x8000 to 0x7FFF
UINT32	32 bits	Defines a 32 bit unsigned integer value in the range of 0 to 4.294.967.295 or 0x00000000 to 0xFFFFFFFF
SINT32	32 bits	Defines a 32 bit signed integer value in the range of −2.147.483.648 to +2.147.483.647or 0x80000000 to 0x7FFFFFFF
FLOAT16	16 bits	Defines a 16 bit float value with a 4 bit exponent and a 12 bit mantissa. 2 MSB 1 LSB FloatValue M E E E E M M M M M M M M M M M FloatValue = $(0{,}01*M)*2^{(E)}$ E = [0 … 15] M = [-2 048 … 2 047], two’s complement notation For all Datapoint Types 9.xxx, the encoded value 7FFFh shall always be used to denote invalid data. [-671 088,64 … 670 760,96]

FLOAT32	32 bits	Defines a 32 bit float value in the range of $\pm 1.4 \cdot 10^{-45}$ to $\pm 3.403 \cdot 10^{38}$. A mantissa of 23 bits, and an exponent of 8 bits are used. The value can store 7 to 8 digits after the comma. 4 MSB 3 2 1 LSB S | Exponent | Fraction F F F F F F F F | F F F F F F F F | F F F F F F F F | F F F F F F F F The values are encoded in the IEEE floating point format according IEEE 754 single precision format. NOTE 7 This specifies that the exponent is biased. This allows negative exponent values. S (Sign) = {0,1} Exponent = [0 … 255] Fraction = [0 … 8 388 607] The resolution is given by the use of the IEEE 754 format and varies with the used exponent.
TIME	24 bits	Defines a 24 bit value encoded a time information in the following way: 3 MSB 2 1 LSB 0 0 0 Day | 0 0 0 0 Month | 0 Year r r r U U U U U | r r r r U U U U | r U U U U U U U Field: / Encoding: / Range: / Unit: / Resol.: Day: 1 = Monday … 7 = Sunday, 0 = no day / [0…7] / none / none Hour: binary encoded / [0…23] / hours / h Minutes: binary encoded / [0…59] / minutes / min Seconds: binary encoded / [0…59] / seconds / s
DATE	24 bits	Defines a 24 bit value encoded a date information in the following way: 3 MSB 2 1 LSB Day | Hour | 0 0 Minutes | 0 0 Seconds N N N U U U U U | r r U U U U U U | r r U U U U U U Field: / Range: / Unit: / Resol.: Day: [1…31] / Day of month / 1 day Month: [1…12] / Month / 1 month Year: [0…99] / Year / 1 year
DATETIME	64 bits	Defines a 64 bit value encoded a date and time information in the following way: 8 MSB 7 6 5 Year | 0 0 0 0 Month | 0 0 0 DayOfMonth | DayOf-Week | HourOfDay U U U U U U U U | r r r r U U U U | r r r U U U U U | U U U U U U U U 4 3 2 1 LSB 0 0 Minutes | 0 0 Seconds | F | WD | NWD | NY | ND | NDoW | NT | SUTI | CLQ | 0 0 0 0 0 0 0 r r U U U U U U | r r U U U U U U | B B B B B B B B | B r r r r r r r

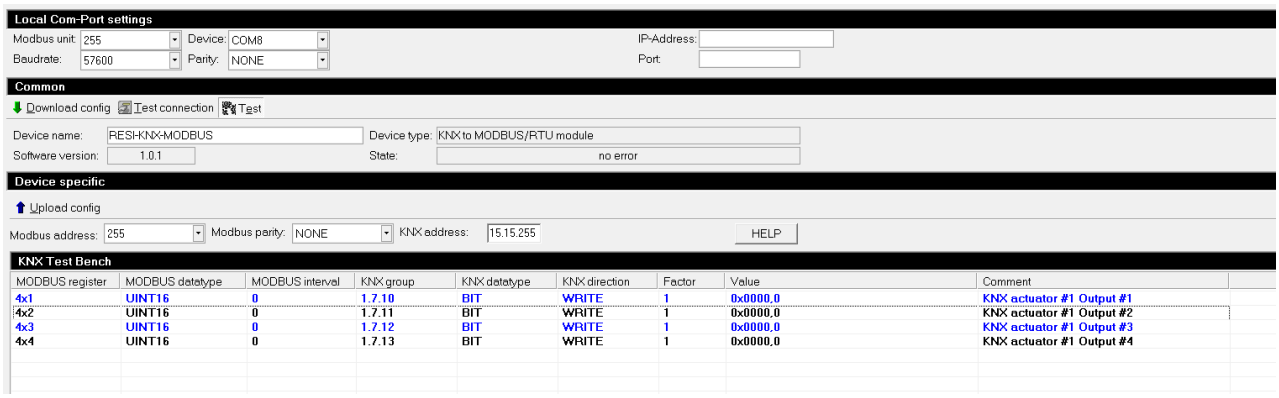
<table>
<tr>
<td></td>
<td></td>
<td>
<table>
<tr><th>Field</th><th>Description</th><th>Encoding</th><th>Range</th><th>Unit</th><th>Resol.:</th></tr>
<tr><td>Year</td><td>Year</td><td>Value binary encoded, offset 1900
0 = 1900
255 = 2155</td><td>[0…255]</td><td>year</td><td>1 year</td></tr>
<tr><td>Month</td><td>Month</td><td>Value binary encoded
1 = January
…
12 = December</td><td>[1…12]</td><td>Month</td><td>1 month</td></tr>
<tr><td>DayOfMonth</td><td>D</td><td>Value binary encoded
1 = 1st day
31 = 31st day</td><td>[1…31]</td><td>none</td><td>none</td></tr>
<tr><td>DayOfWeek</td><td>Day of week</td><td>Value binary encoded
0 = any day
1 = Monday
…
7 = Sunday</td><td>[0…7]</td><td>none</td><td>none</td></tr>
<tr><td>HourOfDay</td><td>Hour of day</td><td>Value binary encoded.</td><td>[0…24]</td><td>h</td><td>1 h</td></tr>
<tr><td>Minutes</td><td>Minutes</td><td>Value binary encoded.</td><td>[0…59]</td><td>min</td><td>1 min</td></tr>
<tr><td>Seconds</td><td>Seconds</td><td>Value binary encoded.</td><td>[0…59]</td><td>s</td><td>1 s</td></tr>
<tr><td>F</td><td>Fault</td><td>0 = Normal (No fault)
1 = Fault</td><td>{0,1}</td><td>none</td><td>none</td></tr>
<tr><td>WD</td><td>Working Day</td><td>0 = Bank day (No working day)
1 = Working day</td><td>{0,1}</td><td>none</td><td>none</td></tr>
<tr><td>NWD</td><td>No WD</td><td>0 = WD field valid
1 = WD field not valid</td><td>{0,1}</td><td>none</td><td>none</td></tr>
<tr><td>NY</td><td>No Year</td><td>0 = Year field valid
1 = Year field not valid</td><td>{0,1}</td><td>none</td><td>none</td></tr>
<tr><td>ND</td><td>No Date</td><td>0 = Month and Day of Month fields valid
1 = Month and Day of Month fields not valid</td><td>{0,1}</td><td>none</td><td>none</td></tr>
</table>
</td>
</tr>
<tr>
<td>STRING</td>
<td>max. 14 bytes</td>
<td>Defines up to 14 bytes of text data
14 MSB … 1 LSB
Character 1 … Character 14
AAAAAAAA … AAAAAAAA
These Datapoint Types are used to transmit strings of textual characters. The length is fixed to 14 octets. The contents are filled starting from the most significant octet. Each octet shall be encoded as specified for the chosen character set, as defined in clause 0. If the string to be transmitted is smaller then 14 octets, unused trailing octets in the character string shall be set to NULL (00h).
Example: ‘KNX is OK’ is encoded as follows :
4B 4E 58 20 69 73 20 4F 4B 00 00 00 00 00</td>
</tr>
<tr>
<td>GENERIC</td>
<td>64 bits</td>
<td>Defines a 64 bit value which represents up to 8 bytes from the data section of a generic KNX telegram. Due to the fact, that a generic KNX frame can hold up to 14 bytes, the field factor defines the start index for the 8 bytes in the range from 0 to 13. The system stores the first byte in the first 16 bit MODBUS register in the low 8 bits. The next byte is stored in the same register, but in the upper half of the word. The third byte is stored in the next Modbus register in the low half, and so on.</td>
</tr>
</table>

- Field “Direction”: Select the communication direction of the KNX group address on the KNX bus. Choose READ for only incoming KNX messages, WRITE for only outgoing KNX messages and READ_WRITE for incoming and outgoing messages. ERR defines an invalid configuration data and is ignored by the gateway.

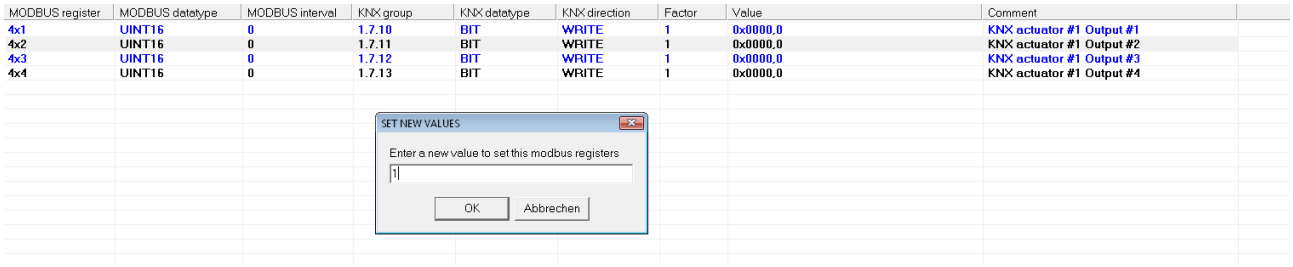
- Field “Comment”: Enter a comment to explain your KNX MODBUS mapping for documentation purpose. Note that the comment in only stored onto the PC, not in the gateway. So if you upload a configuration from the gateway, you will lose all comments.

6.18 Testing the configuration

After you download your new configuration into the converter and start the test mode with the button “Test”, you will see the following screen. The system automatically updates all MODUBS registers every 5 seconds.



To set a new value for a configuration line simple double click onto a configuration line. A window will open, in which you can enter the new value for the selected configuration line. The software will automatically create the correct MODBUS write command for all necessary registers, depending on the configured MODBUS datatype of the line.



Immediately after receiving all MODBUS registers, the converter will send out the corresponding KNX telegram onto the KNX bus, if KNX write is allowed.

7 Sample configurations

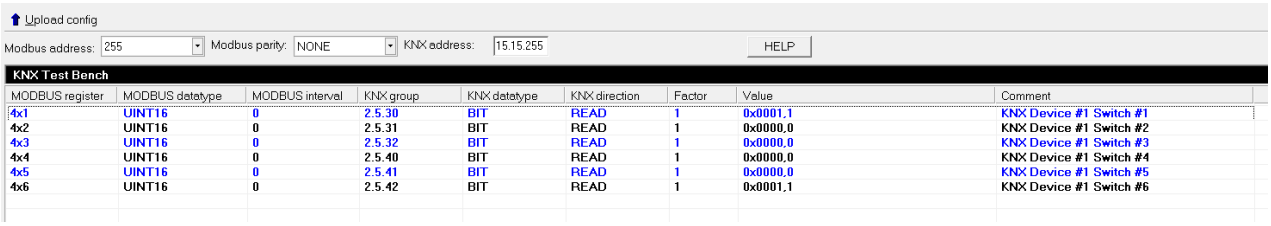
Here you will find some sample configurations to explain the configuration principles of our gateway.

7.1 Reading the status of a KNX switch

Assuming the following setup: 1 KNX switch e.g. GIRA with six switches, programmed with KNX group addresses in the following way:

All six KNX groups send as KNX data a BIT value defining the current state of the switch (0=OFF, 1=ON). If you press the left, top switch, the KNX device sends the KNX telegram 2.5.30=1 or 2.5.30=0 depending on the stored switch state in the KNX device.

So the configuration will look like this:


Upload config

Modbus address: 255 Modbus parity: NONE KNX address: 15.15.255 HELP

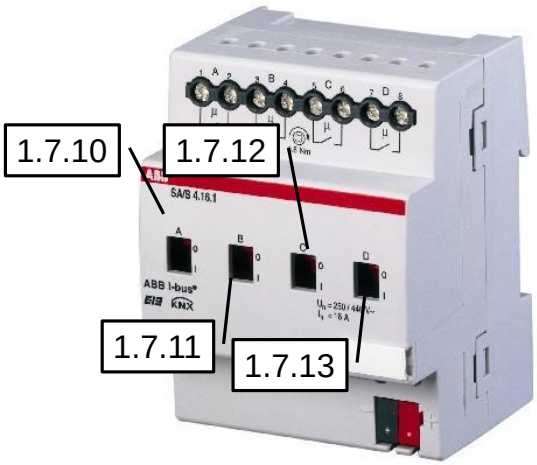
KNX Test Bench

MODBUS register	MODBUS datatype	MODBUS interval	KNX group	KNX datatype	KNX direction	Factor	Value	Comment
4x1	UINT16	0	2.5.30	BIT	READ	1	0x0001,1	KNX Device #1 Switch #1
4x2	UINT16	0	2.5.31	BIT	READ	1	0x0000,0	KNX Device #1 Switch #2
4x3	UINT16	0	2.5.32	BIT	READ	1	0x0000,0	KNX Device #1 Switch #3
4x4	UINT16	0	2.5.40	BIT	READ	1	0x0000,0	KNX Device #1 Switch #4
4x5	UINT16	0	2.5.41	BIT	READ	1	0x0000,0	KNX Device #1 Switch #5
4x6	UINT16	0	2.5.42	BIT	READ	1	0x0001,1	KNX Device #1 Switch #6

As you can see in the test mode, the switch state of Switch #1 and #6 is ON, all other switches are OFF. Press the six buttons and see, how the MODBUS registers are changed by the incoming KNX telegrams.

7.2 Writing to a KNX actuator

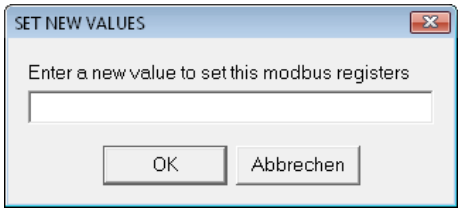
Using a KNX actuator with four outputs, e.g. an ABB KNX actuator, is also very simple. Assuming the following KNX group addresses for the four outputs. All of them expect a KNX telegram with bit data.



The correct configuration will look like this:

MODBUS register	MODBUS datatype	MODBUS interval	KNX group	KNX datatype	KNX direction	Factor	Value	Comment
4x1	UINT16	0	1.7.10	BIT	WRITE	1	0x0000,0	KNX actuator #1 Output #1
4x2	UINT16	0	1.7.11	BIT	WRITE	1	0x0000,0	KNX actuator #1 Output #2
4x3	UINT16	0	1.7.12	BIT	WRITE	1	0x0000,0	KNX actuator #1 Output #3
4x4	UINT16	0	1.7.13	BIT	WRITE	1	0x0000,0	KNX actuator #1 Output #4

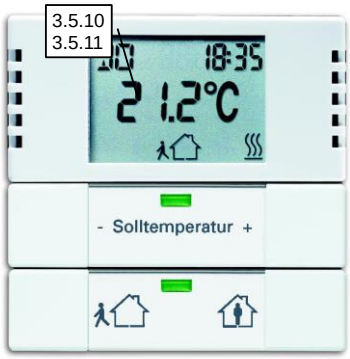
Download this configuration into the gateway and start the test mode. To change the state of the output #1, simple double click onto the first configuration line. The following window will be opened:



Enter a new value for the digital output, e.g. 1, and hit the OK button. Immediately the actuator on the KNX bus will be switched on. Double click again and enter the value 0, the output will be switched off. Try this for the remaining three outputs.

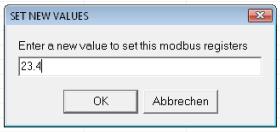
7.3 Reading analogue KNX values

More complicated is the mapping of analogue values from the KNX bus to MODBUS registers. We start with a simple KNX device, e.g. a room controller. We assume, that this device cyclically sends the current room temperature on the KNX group address 3.5.10. The set point can be send/received on the KNX group address 3.5.11. Both values are encoded with KNX datatype 9.001 temperature (°C).



Use the following configuration as a sample:

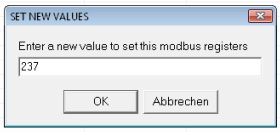
MODBUS register	MODBUS datatype	MODBUS interval	KNX group	KNX datatype	KNX direction	Factor	Value	Comment
4x1	FLOAT32	0	3.5.10	FLOAT16	READ	1	0x41BB851F,23.4400005340576	Actual temperature
4x3	FLOAT32	0	3.5.11	FLOAT16	READ-WRITE	1	0x41BC0000,23.5	New setpoint



As you can see, we map the float value FLOAT16 (This is the KNX representation of KNX Datatype 9.0001) from the KNX bus to FLOAT32 values in the MODBUS registers. A FLOAT32 value uses 2 consecutive registers. That's the reason, why the first value uses the index 4x1 and the second index uses the index 4x3. So this configurations uses four MODBUS registers with the indices 4x1, 4x2, 4x3 and 4x4.

But you can also map a float value from the KNX bus to an integer value on the MODBUS side. We extend the existing configuration with the following lines:

MODBUS register	MODBUS datatype	MODBUS interval	KNX group	KNX datatype	KNX direction	Factor	Value	Comment
4x1	FLOAT32	0	3.5.10	FLOAT16	READ	1	0x41BD47AE,23.6599998474121	Actual temperature
4x3	FLOAT32	0	3.5.11	FLOAT16	READ-WRITE	1	0x41BC0000,23.5	New setpoint
4x5	UINT16	0	3.5.10	FLOAT16	READ	10	0x00EC,236	Actual tempertaur multiplied by 10
4x6	UINT16	0	3.5.11	FLOAT16	READ-WRITE	10	0x00EB,235	New setpoint multiplied by 10



As you will notice, we use now UINT16 for the MODBUS datatype and a factor of 10 to preserve the first digit after the comma. So the MODBUS register 4x5 stores the value 236 if the current temperature 23.66 is received from the KNX bus.

You will also notice, that you can configure more than one line with the same KNX group address, but different MODBUS mappings.

8 Functional Description

The RESI-KNX-MODBUS and RESI-KNX-ASCII gateways communicate with a host system with the MODBUS/RTU slave protocol. The RESI-KNX-ASCII version of the module offer an additional protocol: An ASCII slave protocol with simple text string. The communication runs over a RS485 interface (half duplex) or over a RS232 interface (full duplex).

For the communication with ASCII texts, the host sends ASCII messages with a special start character # (0x23, 35dec) and a special end character (0x0d,13dec or CARRIAGE RETURN) to the module. The module uses also this special start and end characters to answer to the host request. Consult the below noted detailed command descriptions. In the ASCII protocol mode, the host can send messages with or without a bus number.

For communication with the MODBUS/RTU slave, the module offers the following MODBUS functions:

- READ HOLDING REGISTER (function code: 3)
- PRESET SINGLE REGISTER (function code: 6)
- PRESET MULTIPLE REGISTERS (function code: 16)

HINT:
The functions READ HOLDING REGISTER and PRESET MULTIPLE REGISTERS are restricted to max. 125 register per request!

8.1 ASCII protocol description

8.1.1 Overview

The IO module communicates with simple ASCII commands. The following special characters are used in this documentation:

stand for the **Hashtag** ASCII character 35dec or 0x23
: stand for the **colon** ASCII character 58dec or 0x3A
= stand for the **equal sign** with the ASCII code 61ec or 0x3D
- stand for the **minus sign** with the ASCII code 45dec or 0x2D
, stands for the **comma** with the ASCII code 44dec or 0x2C
<CR> stand for **CARRIAGE RETURN** ASCII character 13dec or 0x0D. In the following text we use the representation $_{CR}$.
<SP> stands for the **SPACE**. This is the space character with the ASCII code 32dec or 0x20. In the following text we will use the representation □.
<ADR> will be used as the current **bus address** of the module. The bus number can be transmitted decimal or hexadecimal and is separated with a comma (ASCII code 44dec or 0x2C) from the rest of the command. Hexadecimal numbers always start with 0x. Its only allowed to use the ASCII characters ‚0'-‚9' 48dec to 57dec, 0x30-0x39 and ‚A' to ‚F', 65dec to 70dec, 0x41-0x46. All modules react to the broadcast bus address 0 and to its own bus number. With a DIP switch, the user can easily change between the internal stored bus number in the FLASH and the fix bus number 255. Consult the DIP switch description for more details.

8.1.2 Communication sequence

In general the module sends no ASCII characters without a request from a host. So the host is the master of the communication and the module is always answering to host requests as a slave. If only one module is connected to a host (eg. Via RS232), you can dump the bus number in the protocol. If you use a RS485 interface, more than one module can be connected to the host. Therefore a bus number in the request frame of the host is always necessary.

The structure of the command look like this:

The host sends a command or a command with parameters without a bus address to the module:
#<Command><CR> or
#<Command>:<Parameter><CR>

The module answers, if it feels addressed, with the following answer telegram:
#<Answer><CR>

If the bus number is used, the module answers with:
#<ADR>,<Command><CR> or
#<ADR>,<Command>:<Parameter><CR>

The module answers with:
#<ADR>,<Answer><CR>

The bus address lies in the range of 1dec to 255dec or 0x00 to 0xFF hexadecimal. The setup is done with our free configuration software MODBUSConfigurator. Each module reacts also to the broadcast bus address 0.

For each command, we define two different writings. A long version and a short version, to avoid unnecessary traffic on the bus. For example to request the software version of the module you can use the command VERSION or the short command VER.

8.1.3 Request VERSION

This command returns the current software version of the module.

Host long version:
#VERSION<CR> or
#<ADR>,VERSION<CR>

Host short version:
#VER<CR> or
#<ADR>,VER<CR>

Answer:
#VERSION:<HIGH>.<MED>.<LOW><CR> or
#<ADR>,VERSION:<HIGH>,<MED>,<LOW><CR>

<HIGH>.<MED>.<LOW> represents the current software version e.g. 3.0.0

Samples:
→ **#VERSION$_{CR}$**
← **#VERSION:3.0.0$_{CR}$**

With broadcast address in decimal and long version:
→ **#0,VERSION$_{CR}$**
← **#0,VERSION:3.0.0$_{CR}$**

With broadcast address in hexadecimal and short version:
→ **#0x00,VER$_{CR}$**
← **#0x00,VERSION:3.0.0$_{CR}$**

With bus address 255 in decimal:
→ **#255,VER$_{CR}$**
← **#255,VERSION:3.0.0$_{CR}$**

With bus address 255 in hexadecimal
→ **#0xFF,VERSION$_{CR}$**
← **#0xFF,VERSION:3.0.0$_{CR}$**

With bus address 43 in decimal
→ **#43,VER$_{CR}$**
← **#43,VERSION:3.0.0$_{CR}$**

With bus address 43 in hexadecimal
→ **#0x2B,VER$_{CR}$**
← **#0x2B,VERSION:3.0.0$_{CR}$**

8.1.4 Request module TYPE

This command returns the current type of the module.

Host long version:
#TYPE<CR> or
#<ADR>,TYPE<CR>

Host short version:
#TYP<CR> or
#<ADR>,TYP<CR>

Answer:
#TYPE:<TYP><CR> or
#<ADR>,TYPE:<TYP><CR>

<TYP> defines the current type of the module. Currently RESI-KNX-ASCII

Samples:
→ **#TYPE$_{CR}$**
← **#TYPE:RESI-KNX-ASCII$_{CR}$**

→ **#255,TYP$_{CR}$**
← **#255,TYPE:RESI-KNX-ASCII$_{CR}$**

8.1.5 Table of all ASCII commands

Here you will find a possible ASCII commands of the module. We use here only the version with bus number. That you can avoid the bus number, we have discussed earlier in this document. If an argument as the extension Dec, it will be returned as a decimal number, If an argument has the extension Hex, then this argument is returned as a hexadecimal number. Many command returns the argument in decimal and hexadecimal representation. So the host can select, what kind of number conversion, it will handle in its software.

Direction	ASCII command
Host	**#<BusAdr>,VER$_{CR}$** **#<BusAdr>,VERSION$_{CR}$**
Answer	**#<BusAdr>,VERSION:<VersionHi>.< VersionMed>.<VersionLo>$_{CR}$**
	Returns the version number of the module VersionHi — Version number high (1..255) VersionMed — Version number medium (1..255) VersionLo — Version number low (1..255)
Host	**#<BusAdr>,TYP$_{CR}$** **#<BusAdr>,TYPE$_{CR}$**
Answer	**#<BusAdr>,TYPE:RESI-KNX-ASCII$_{CR}$**
	Returns the current type of the module
Host	**#<BusAdr>,OWN$_{CR}$** **#<BusAdr>,OWNER$_{CR}$**
Answer	**#<BusAdr>,OWNER:RESI$_{CR}$**
	Returns the owner of the module
Host	**#<BusAdr>,CRE$_{CR}$** **#<BusAdr>,CREATOR$_{CR}$**
Answer	**#<BusAdr>,CREATOR:DI HC SIGL,MSC$_{CR}$**
	Returns the creator of the module
Host	**#<BusAdr>,COPY$_{CR}$** **#<BusAdr>,COPYRIGHT$_{CR}$**
Answer	**#<BusAdr>,COPYRIGHT:2015 BY RESI AND DI HC SIGL,MSC WWW.RESI.CC$_{CR}$**
	Returns a copyright note about the module

Direction	ASCII command
Host	**#<BusAdr>,GDIP$_{CR}$** **#<BusAdr>,GET□DIP$_{CR}$**
Answer	**#<BusAdr>,GDIP:<DIPSwitchDec>,<DIPSwitchHex>$_{CR}$**
	Returns the current setting of the Dip switches as decimal number and as hexadecimal number. DIPSwitchDec DIPSwitchHex The current value of the DIP switches: Bit 0: DIP Switch 1 (=0:OFF, =1:ON) Bit 1: DIP Switch 2 (=0:OFF, =1:ON) Bit 2: DIP Switch 3 (=0:OFF, =1:ON) Bit 3: DIP Switch 4 (=0:OFF, =1:ON)
Host	**#<BusAdr>,CC$_{CR}$** **#<BusAdr>,CLEAR□CONFIG$_{CR}$**
Answer	**#<BusAdr>,OK$_{CR}$**
	This command clears the complete configuration table in the FLASH of the gateway.
Host	**#<BusAdr>,GCS$_{CR}$** **#<BusAdr>,GET□CONFIG□SIZE$_{CR}$**
Answer	**#<BusAdr>,GCS:<ConfigSize>$_{CR}$**
	Returns the actual number of configured lines in the FLASH of the gateway ConfigSize The current amount of configured lines in the gateway
Host	**#<BusAdr>,RC:<Index>$_{CR}$** **#<BusAdr>,READ□CONFIG:<Index>$_{CR}$**
Answer	**#<BusAdr>,KNX:I:<ConfigurationLine>$_{CR}$**
	Returns the requested configuration line <Index> Index The number of the requested configuration line in the range of 1..n, where n is the last line of the gateway table. ConfigurationLine See the explanation below for more details
Host	**#<BusAdr>,GC$_{CR}$** **#<BusAdr>,GET□CONFIG$_{CR}$**
Answer	For each configuration line the gateway answers with the line **#<BusAdr>,KNX:I:<ConfigurationLine>$_{CR}$** At the end of the configuration table the gateway send the line **#<BusAdr>,OK$_{CR}$**
	Returns the complete configuration table of the gateway ConfigurationLine See the explanation below for more details
Host	**#<BusAdr>,AC:<AddConfigurationLine>$_{CR}$** **#<BusAdr>,ADD□CONFIG:<AddConfigurationLine>$_{CR}$**
Answer	If the configuration line is correct, the gateway answers with **#<BusAdr>,OK:<Index>$_{CR}$** Otherwise the gateway throws an error answer **#<BusAdr>,ERR:<ErrorText>$_{CR}$**
	Adds a line to the configuration table AddConfigurationLine See the explanation below for more details Index The number of the new added configuration line in the range of 1..n. ErrorText A text which describes the error in detail

Direction	ASCII command
Host	**#<BusAdr>,RV:<Index>$_{CR}$** **#<BusAdr>,READ□VALUE:<Index>$_{CR}$**
Answer	**#<BusAdr>,RV:<FloatValue>$_{CR}$**
	Returns the current value of the requested configuration line <Index> Index — The number of the requested configuration line in the range of 1..n, where n is the last line of the current converter table. FloatValue — The current value in the MODBUS registers formatted as a float number.
Host	**#<BusAdr>,WV:<Index>=<FloatValue>$_{CR}$** **#<BusAdr>,WRITE□VALUE:<Index>=<FloatValue>$_{CR}$**
Answer	**#<BusAdr>,OK$_{CR}$**
	Writes a new value into the defined configuration line <Index>. If the configuration defines a write operation on the KNX bus, the corresponding KNX telegram is generated. Index — The number of the requested configuration line in the range of 1..n, where n is the last line of the current converter table. FloatValue — The new value for the MODBUS registers formatted as a float number

Direction	ASCII command
Host	**#<BusAdr>,SMBADR:<MBUnit>$_{CR}$** **#<BusAdr>,SET□MODBUS□ADDRESS:<MBUnit>$_{CR}$**
Answer	**#<BusAdr>,OK $_{CR}$**
	Redefines the unit ID of the module. This change will affect the MODBUS/RTU communication immediately. As a Unit IO you can use the values 0dec to 255dec.
Host	**#<BusAdr>,GMBADR$_{CR}$** **#<BusAdr>,GET□MODBUS□ADDRESS$_{CR}$**
Answer	**#<BusAdr>,GMBADR:<MBUnitDec>,<MBUnitHex> $_{CR}$**
	Returns the current MODBUS Unit IO of the module. MBUnitDec MBUnitHex — The current configured MODBUS/RTU Unit ID for the communication. Be aware, that the converter delivers 65535 or 0xFFFF, if no configuration is downloaded into the converter. This answer means also the bus address 255!
Host	**#<BusAdr>,SKNXADR:<KNXAddress>$_{CR}$** **#<BusAdr>,SET□KNX□ADDRESS:<KNXAddress>$_{CR}$**
Answer	**#<BusAdr>,OK $_{CR}$**
	Redefines the KNX address of the module for KNX bus communication. KNXAddress — The new KNX address for the communication on the KNX bus in the format <Hi>.<Med>.<Low> e.g. 15.15.255 Address range from 0.0.0 to 15.15.255
Host	**#<BusAdr>,GKNXADR$_{CR}$** **#<BusAdr>,GET□KNX□ADDRESS$_{CR}$**
Answer	**#<BusAdr>,GKNXADR:<KNXAdrDec>,<KNXAdrHex>,<KNXAdrKNX> $_{CR}$**
	Returns the current KNX address of the module. KNXAdrDec KNXAdrHex — The current configured KNX address for the communication on the KNX bus. KNXAdrKNX — The current configured KNX address for the communication on the KNX bus in the format <Hi>.<Med>.<Low> e.g. 15.15.255
Host	**#<BusAdr>,RST$_{CR}$** **#<BusAdr>,RESET$_{CR}$**
Answer	**None**
	Executes a software reset (Reboot) of the module.

8.1.6 The Configuration Line

The two ASCII commands READ CONFIG and GET CONFIG return all data for one configuration line. Here is a detailed description of each field of this line.

The common syntax for the line is the following:
#<BusAdr>,KNX:I:<Index>=<MBRegister>,<MBDataType>,<MBRegisterCount>,<Interval>, <KNXGroup>,<KNXDataType>,<KNXDirection>,<Factor>

<BusAdr> stands for the current MODBUS bus address of the module as a decimal number e.g. 255

<Index> stands for the Index in the configuration table starting with 1 and ending with n according to the actual size of the configuration table size as a decimal number.

<MBRegister> stands for the starting index of the holding registers, starting with 1 for the first holding register 4x00001 and ending by 65535 for the last holding register 4x65535 as a decimal number.

<MBDataType> defines a datatype for the MODBUS registers. This is an ASCII text string in capital letters. Choose one of the following strings: UINT16, SINT16, UINT32, SINT32, UINT32R, SINT32R, FLOAT32, FLOAT32R, DOUBLE64, DOUBLE64R, GENERIC, ASCII or ERR. See the explanation of the MODBUS datatypes for more details about this strings.

<MBRegisterCount> defines the amount of MODBUS holding registers, which are used by this configuration entry as a decimal number. For example a UINT16 needs 1 register, a UINT32 or a FLOAT32 needs 2 registers.

<Interval> stands for a time interval in seconds as a decimal number for an automatic polling request on the KNX bus for this KNX group address. This is for future use and not used now!

<KNXGroup> defines the KNX group address with the format <Hi>.<Medium>.<Low>. KNX group addresses in the range from 0.0.0 to 15.7.255 are valid here.

<KNXDataType> is a string defining the data type of the incoming or outgoing KNX data. The system uses the following ASCII strings in capital letters: ERR, BIT, TWOBITS, FOURBITS, SIXBITS, CHARACTER, UINT8, SINT8, UNIT16, SINT16, FLOAT16, TIME, DATE, UINT32, SINT32, FLOAT32, STRING, GENERIC and DATETIME. See the explanation of the KNX datatypes for more details about this strings.

<KNXDirection> is an ASCII text string defining the communication direction for this entry on the KNX bus. The following ASCII string in capital letters are valid: ERR, R, W, RW. See the explanation for KNX directions for more details about this string.

<Factor> is a float value defining the multiplication factor for incoming KNX telegrams and the division factor for outgoing KNX telegrams. Use the float format 1234.567. Don't use a comma as a comma sign!

Here is a cut-out of a real configuration from a terminal program:

```
1    5    10    15    20    25    30    35    40    45    50    55    60    6
#VERSION:1.0.1
TYPE:RESI-KNX-MODBUS
#KNX:I:1=1,SINT16,1,0,1.1.3,FLOAT16,R,10.000000
#KNX:I:2=2,SINT16,1,0,1.1.4,FLOAT16,R,10.000000
#KNX:I:3=3,SINT16,1,0,1.1.7,FLOAT16,R,10.000000
#KNX:I:4=4,SINT16,1,0,1.1.8,FLOAT16,R,10.000000
#KNX:I:5=5,SINT16,1,0,1.1.9,FLOAT16,R,10.000000
#KNX:I:6=6,SINT16,1,0,1.2.1,FLOAT16,R,10.000000
#KNX:I:7=7,SINT16,1,0,1.2.6,FLOAT16,R,10.000000
#KNX:I:8=8,SINT16,1,0,1.2.11,FLOAT16,R,10.000000
#KNX:I:9=9,SINT16,1,0,1.2.17,FLOAT16,R,10.000000
#KNX:I:10=10,SINT16,1,0,1.2.22,FLOAT16,R,10.000000
#KNX:I:11=11,SINT16,1,0,1.3.1,FLOAT16,R,10.000000
#KNX:I:12=12,SINT16,1,0,1.3.2,FLOAT16,R,10.000000
#KNX:I:13=13,SINT16,1,0,1.4.1,FLOAT16,R,10.000000
#KNX:I:14=14,SINT16,1,0,1.4.2,FLOAT16,R,10.000000
#KNX:I:15=15,SINT16,1,0,1.4.3,FLOAT16,R,10.000000
#KNX:I:16=16,SINT16,1,0,1.4.4,FLOAT16,R,10.000000
#KNX:I:17=17,UINT16,1,0,10.3.5,BIT,R,1.000000
#KNX:I:18=18,UINT16,1,0,10.3.6,BIT,R,1.000000
#KNX:I:19=19,SINT32,2,0,9.3.4,UINT32,R,0.001000
#KNX:I:20=21,SINT32,2,0,9.3.6,UINT32,R,0.001000
#KNX:I:21=23,SINT16,1,0,1.3.1,FLOAT16,R,10.000000
#KNX:I:22=24,SINT16,1,0,1.3.2,FLOAT16,R,10.000000
#KNX:I:23=25,SINT32,2,0,9.3.2,UINT32,R,0.001000
#KNX:I:24=27,SINT32,2,0,9.3.5,UINT32,R,0.001000
#KNX:I:25=29,SINT32,2,0,9.4.29,UINT32,R,0.001000
#KNX:I:26=31,SINT32,2,0,9.4.30,UINT32,R,0.001000
#KNX:I:27=33,SINT16,1,0,1.4.2,FLOAT16,R,10.000000
#KNX:I:28=34,SINT16,1,0,1.4.3,FLOAT16,R,10.000000
#KNX:I:29=35,SINT32,2,0,9.4.21,UINT32,R,0.001000
#KNX:I:30=37,SINT32,2,0,9.4.31,UINT32,R,0.001000
#KNX:I:31=39,SINT32,2,0,9.4.32,UINT32,R,0.001000
#KNX:I:32=41,SINT32,2,0,9.4.34,UINT32,R,0.001000
#KNX:I:33=43,SINT16,1,0,1.4.1,FLOAT16,R,10.000000
#KNX:I:34=44,SINT16,1,0,1.4.4,FLOAT16,R,10.000000
#KNX:I:35=45,SINT32,2,0,9.4.27,UINT32,R,0.001000
#KNX:I:36=47,SINT32,2,0,9.4.33,UINT32,R,0.001000
#KNX:I:37=49,SINT16,1,0,1.3.4,FLOAT16,R,10.000000
#KNX:I:38=50,SINT16,1,0,1.3.5,FLOAT16,R,10.000000
#KNX:I:39=51,SINT32,2,0,9.4.35,UINT32,R,0.001000

Selection (-)
```

8.1.7 The Add Configuration Line

The ASCII commands ADD CONFIG uses a complex configuration line to add a new entry to the current configuration table. Here is a detailed description of each field of this line.

The common syntax for the line is the following:
#<BusAdr>,ADD CONFIG:<MBRegister>,<MBDataType>,<Interval>,
<KNXGroup>,<KNXDataType>,<KNXDirection>,<Factor>

<BusAdr> stand for the current MODBUS bus address of the module as a decimal number e.g. 255

<MBRegister> stands for the starting index of the holding registers, starting with 1 for the first holding register 4x00001 and ending by 65535 for the last holding register 4x65535 as a decimal number. If you use 0 as a MODBUS register index, the next free MODBUS register is used for this entry.

<MBDataType> defines a datatype for the MODBUS registers. This is an ASCII text string in capital letters. Choose one of the following strings: UINT16, SINT16, UINT32, SINT32, UINT32R, SINT32R, FLOAT32, FLOAT32R, DOUBLE64, DOUBLE64R, GENERIC, ASCII or ERR. See the explanation of the MODBUS datatypes for more details about this strings.

<Interval> stands for a time interval in seconds as a decimal number for an automatic polling request on the KNX bus for this KNX group address. This is for future use and not used now!

<KNXGroup> defines the KNX group address with the format <Hi>.<Medium>.<Low>. KNX group addresses in the range from 0.0.0 to 15.7.255 are valid here.

<KNXDataType> is a string defining the data type of the incoming or outgoing KNX data. You can use the following ASCII string in capital letters: ERR, BIT, TWOBITS, FOURBITS, SIXBITS, CHARACTER, UINT8, SINT8, UNIT16, SINT16, FLOAT16, TIME, DATE, UINT32, SINT32, FLOAT32, STRING, GENERIC and DATETIME. See the explanation of the KNX datatypes for more details about this strings.

<KNXDirection> is a string defining the communication direction for this entry on the KNX bus. The following ASCII string in capital letters are valid: ERR, READ, WRITE, READ-WRITE, READWRITE, R, W, RW. See the explanation for KNX directions for more details about this string.

<Factor> is a float value defining the multiplication factor for incoming KNX telegrams and the division factor for outgoing KNX telegrams. Use the float format 1234.567. Don't use a comma as a comma sign!

A simple example for a valid ADD CONFIG command:

#AC:1,UINT16,0,1.0.0,BIT,READ,1.0
#255,AC:0,UINT16,0,1.0.1,FLOAT16,RW,1.0

8.2 MODBUS – register description

8.2.1 Table of holding registers

The module holds internally a list of 16 bit wide holding register. Those registers can be read by the host with the function READ HOLDING REGISTER (function code: 3). If the register can also be modified by the host, the host can use the functions PRESET SINGLE REGISTER (function code: 6) and PRESET MULTIPLE REGISTERS (function code: 16).

The MODBUS convention defines 65535 possible holding register with the notation 4x00001 to 4x65536. Input register are usually noted with 3x00001 to 3x65536. Please refer the software MODBUS POLL as a sample for this notation. Internally in the MODBUS/RTU frames an index notation is used, which starts with 0 and ends with 65535. So we decided to note in the following document a register with: 4x00100 for the holding register 100, 3x00100 as a hint, that you can read this register also as the input register 100, and in addition also the real index of the protocol index 99 with the notation I:99.

Due to the fact, that you can generate almost every MODBUS mapping, we show only a sample configuration as a hint, how the mapping looks like:

MODBUS register	MODBUS datatype	MODBUS interval	KNX group	KNX datatype	KNX direction	Factor	Value	Comment
4x1	SINT16	0	1.1.3	FLOAT16	READ	10	????	F1.03 VL-Pelletskessel
4x2	SINT16	0	1.1.4	FLOAT16	READ	10	????	F1.04 RL-Pelletskessel
4x3	SINT16	0	1.1.7	FLOAT16	READ	10	????	F1.07 VL-Pelletskessel
4x4	SINT16	0	1.1.8	FLOAT16	READ	10	????	F1.08 RL-Pelletskessel
4x5	SINT16	0	1.1.9	FLOAT16	READ	10	????	F1.09 Aussentemperatur
4x6	SINT16	0	1.2.1	FLOAT16	READ	10	????	F2.01 Pufferspeicher 1
4x7	SINT16	0	1.2.6	FLOAT16	READ	10	????	F2.06 Pufferspeicher 2
4x8	SINT16	0	1.2.11	FLOAT16	READ	10	????	F2.11 Pufferspeicher 3
4x9	SINT16	0	1.2.17	FLOAT16	READ	10	????	F2.17 Pufferspeicher 4
4x10	SINT16	0	1.2.22	FLOAT16	READ	10	????	F2.18 Pufferspeicher 5
4x11	SINT16	0	1.3.1	FLOAT16	READ	10	????	F3.01 VL-Solarsystem
4x12	SINT16	0	1.3.2	FLOAT16	READ	10	????	F3.02 RL-Solarsystem
4x13	SINT16	0	1.4.1	FLOAT16	READ	10	????	F4.01 VL-Heizsystem
4x14	SINT16	0	1.4.2	FLOAT16	READ	10	????	F4.02 VL-Heizsystem
4x15	SINT16	0	1.4.3	FLOAT16	READ	10	????	F4.03 RL-Heizsystem
4x16	SINT16	0	1.4.4	FLOAT16	READ	10	????	F4.04 RL-Zirkulation
4x17	UINT16	0	10.3.5	BIT	READ	1	????	V3.01 Ventil unterer WT
4x18	UINT16	0	10.3.6	BIT	READ	1	????	V3.02 Ventil unterer WT
4x19	SINT32	0	9.3.4	UINT32	READ	0.001	????	Z3.01.01 WMZ RL-Solar Q
4x21	SINT32	0	9.3.6	UINT32	READ	0.001	????	Z3.01.02 WMZ RL-Solar V
4x23	SINT16	0	1.3.1	FLOAT16	READ	10	????	Z3.01.03 WMZ T-VL
4x24	SINT16	0	1.3.2	FLOAT16	READ	10	????	Z3.01.04 WMZ T-RL
4x25	SINT32	0	9.3.2	UINT32	READ	0.001	????	Z3.01.05 WMZ RL-Solar P
4x27	SINT32	0	9.3.5	UINT32	READ	0.001	????	Z3.01.06 WMZ RL-Solar dV
4x29	SINT32	0	9.4.29	UINT32	READ	0.001	????	Z4.01.01 WMZ RL-Heizsystem Q
4x31	SINT32	0	9.4.30	UINT32	READ	0.001	????	Z4.01.02 WMZ RL-Heizsystem V
4x33	SINT16	0	1.4.2	FLOAT16	READ	10	????	Z4.01.03 WMZ T-VL
4x34	SINT16	0	1.4.3	FLOAT16	READ	10	????	Z4.01.04 WMZ T-RL
4x35	SINT32	0	9.4.21	UINT32	READ	0.001	????	Z4.01.05 WMZ RL-Heizsystem P
4x37	SINT32	0	9.4.31	UINT32	READ	0.001	????	Z4.01.06 WMZ RL-Heizsystem dV
4x39	SINT32	0	9.4.32	UINT32	READ	0.001	????	Z4.02.01 WMZ VL-Zirkulation Q
4x41	SINT32	0	9.4.34	UINT32	READ	0.001	????	Z4.02.02 WMZ VL-Zirkulation V
4x43	SINT16	0	1.4.1	FLOAT16	READ	10	????	Z4.02.03 WMZ T-VL
4x44	SINT16	0	1.4.4	FLOAT16	READ	10	????	Z4.02.04 WMZ T-RL
4x45	SINT32	0	9.4.27	UINT32	READ	0.001	????	Z4.02.05 WMZ VL-Zirkulation P
4x47	SINT32	0	9.4.33	UINT32	READ	0.001	????	Z4.02.06 WMZ VL-Zirkulation dV
4x49	SINT16	0	1.3.4	FLOAT16	READ	10	????	F3.04 T-Kollektoren 1
4x50	SINT16	0	1.3.5	FLOAT16	READ	10	????	F3.05 T-Kollektoren 2
4x51	SINT32	0	9.4.35	UINT32	READ	0.001	????	Z4.04.01 WMZ RL-Heizsystem Haus A Q
4x53	SINT32	0	9.4.37	UINT32	READ	0.001	????	Z4.04.02 WMZ RL-Heizsystem Haus A V

8.2.2 MODBUS datatype storage and common pitfalls

In general MODBUS uses 16 bit wide registers. So if you use only datatypes, which needs also only one register, the mapping is easy. But as soon as you use datatypes, e.g. UINT32, which need two or more MODBUS registers, you can map the values in different ways.

We do a simple sample. We want to store the 32 bit unsigned integer value in hexadecimal 0x12345678 in MODBUS holding registers starting with index 4x00010. The mapping can be done in two different ways:

MODBUS Register	**Storage of UINT32 datatype**
4x00010 I:9	The high word of the 32 bit value 0x12345678 is stored in the first 16 bit wide MODBUS register. This means the value 0x1234 is stored here.
4x00011 I:10	The low word of the 32 bit value 0x12345678 is stored in the second 16 bit wide MODBUS register. This means the value 0x5678 is stored here.

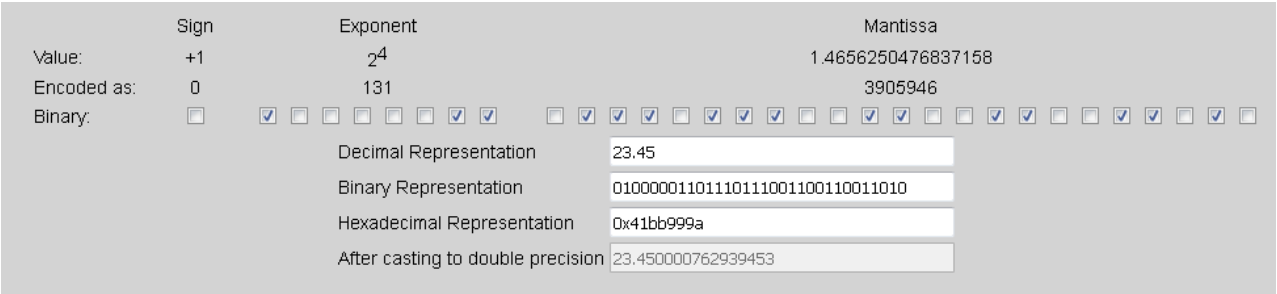
But it is only one possibility, that we store the high word in the first MODBUS register. With the same right, we can define to store the low word in the first register, and the high word in the second.

The result will look like this:

MODBUS Register	Storage of UINT32R datatype
4x00010 I:9	The low word of the 32 bit value 0x12345678 is stored in the first 16 bit wide MODBUS register. This means the value 0x5678 is stored here.
4x00011 I:10	The high word of the 32 bit value 0x12345678 is stored in the second 16 bit wide MODBUS register. This means the value 0x1234 is stored here.

More complicated is the storage of a FLOAT32 value into two consecutive holding registers. We use a standard room temperature e.g. 23,45 °C as a value, we want to store into two registers.

First we have to translate this value into a valid IEE754 float value. Therefore we use a perfect site in the internet (http://www.h-schmidt.net/FloatConverter/IEEE754.html):



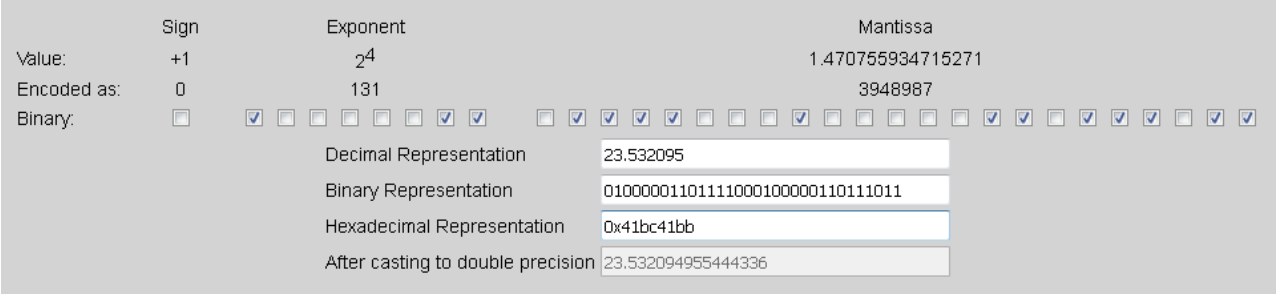
We enter the value 23.45 and we get a 32 bit hexadecimal representation of the float value. It is the number 0x41BB999A. Now we store this value in the same way, we have stored the UINT32 value into two registers:

MODBUS Register	Storage of FLOAT32 datatype
4x00010 I:9	The high word of the 32 bit float value 0x41BB999A is stored in the first 16 bit wide MODBUS register. This means the value 0x41BB is stored here.
4x00011 I:10	The low word of the 32 bit float value 0x41BB999A is stored in the second 16 bit wide MODBUS register. This means the value 0x999A is stored here.

But we can also use the reverse notation:

MODBUS Register	Storage of FLOAT32R datatype
4x00010 I:9	The low word of the 32 bit float value 0x41BB999A is stored in the first 16 bit wide MODBUS register. This means the value 0x999A is stored here.
4x00011 I:10	The high word of the 32 bit float value 0x41BB999A is stored in the second 16 bit wide MODBUS register. This means the value 0x41BB is stored here.

Now we show a common pitfall in writing and reading more than one MODBUS register and rebuilding a value. We use a different float value. In hexadecimal it is 0x41BC41BB. Again we use the online converter:

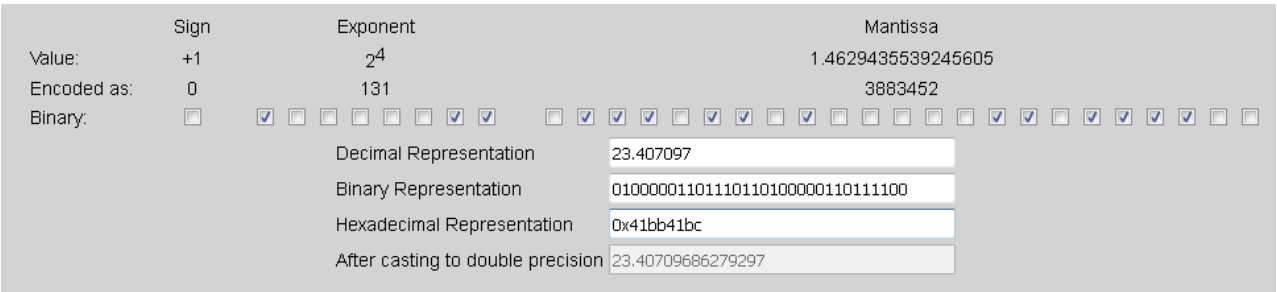


You notice, the float value is 23.532095.

Now we store it with HIGH word first into two registers:

MODBUS Register	**Storage of FLOAT32 datatype**
4x00010 I:9 HIGH WORD	The high word of the 32 bit float value 0x41BC41BB is stored in the first 16 bit wide MODBUS register. This means the value 0x41BC is stored here.
4x00011 I:10 LOW WORD	The low word of the 32 bit float value 0x41BC41BB is stored in the second 16 bit wide MODBUS register. This means the value 0x41BB is stored here.

But now we make a very big mistake, we read the two registers and restore the hexadecimal value in our host software in the reverse word order. First low word, then high word. The result is the 32 bit value 0x41BB41BC instead the correct value 0x41BC41BB. Then we convert this into an IEE754 float value.



The result is 23.407097. This is not far away from the original number of 23.532095! So this massive software error can be undiscovered for a long time. Only if the reverse float value generates numbers which are physically not possible for the measured signal, this error is discovered!

9 Specifications

9.1 Dimensions

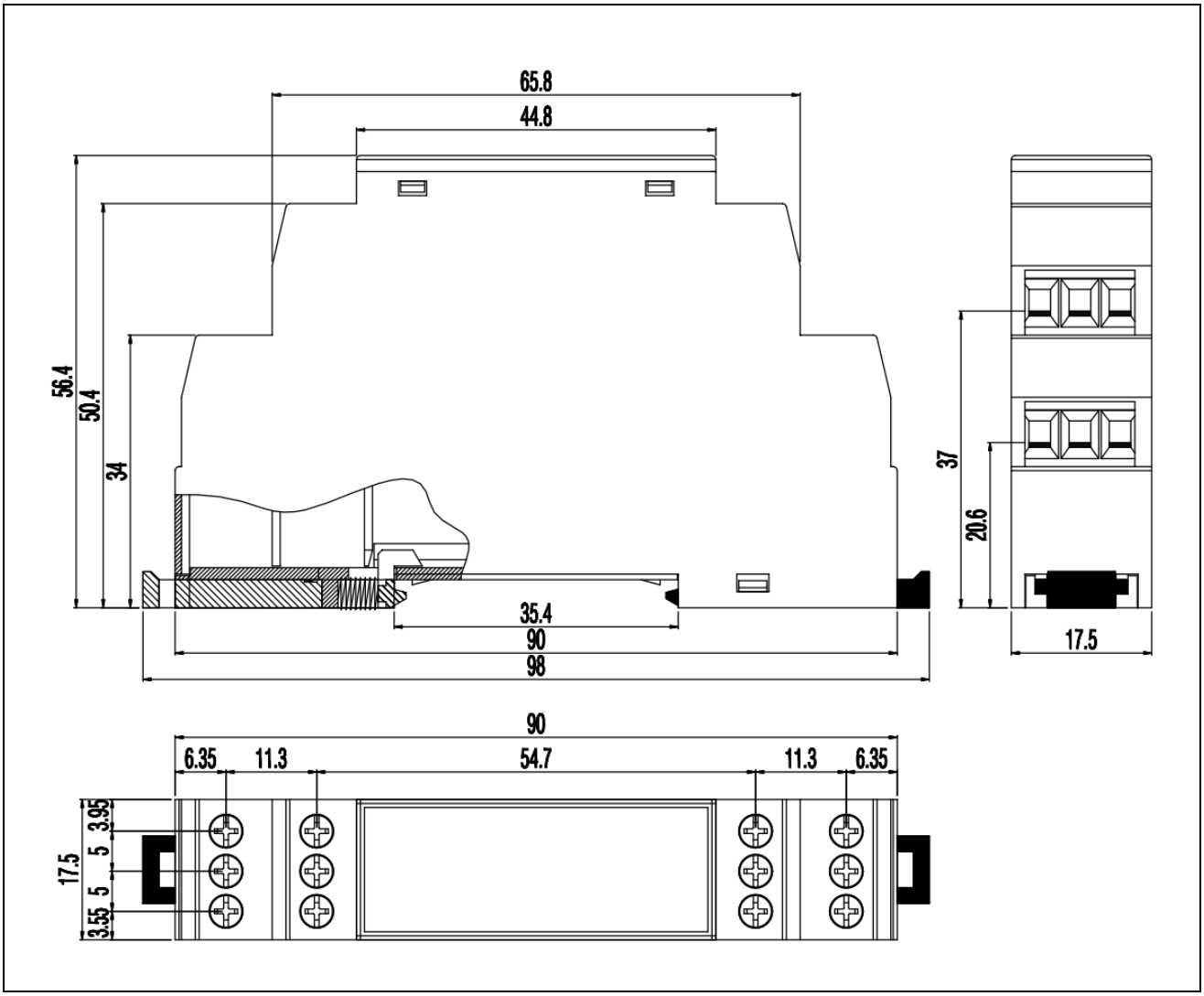


Illustration: dimension illustration in mm

Dimensions	
Enclosure dimensions L x W x H (mm)	17,5 x 90 x 58
Weight	55 g
Colour	Grey RAL7035
Material	PA - UL 94 V0
Protection class	IP20 based on DIN 40050/EN 60529

Table: Data of enclosure

9.2 3D Drawing

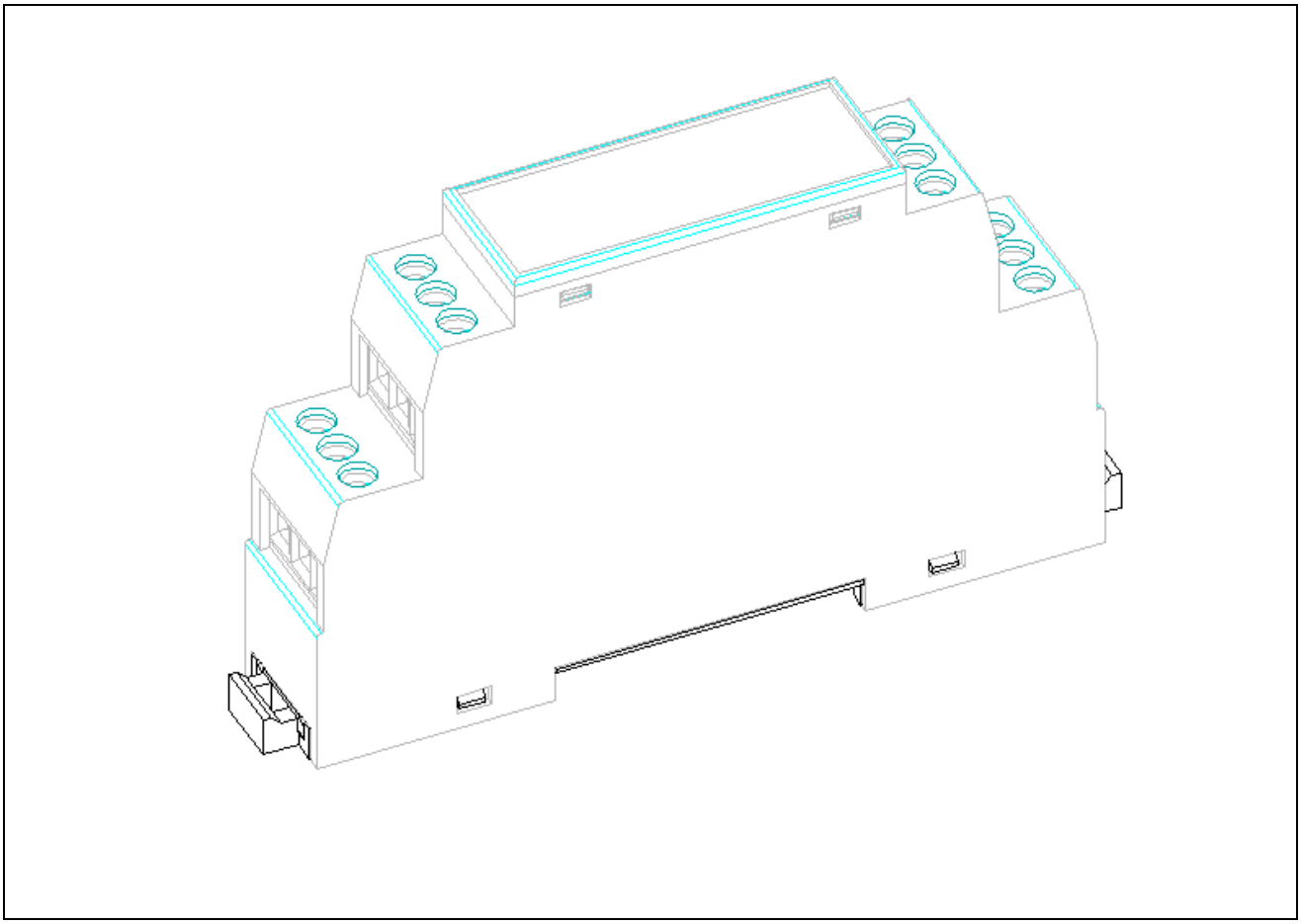


Illustration: Dimension illustration in 3D