MANUAL

...feel the spirit of a new generation

RESI-DMX-ASCII

Great care has been taken in the creation of the text, illustrations and program examples in this manual. RESI Informatik & Automation GmbH company, 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.

This manual may not be copied in part or whole in any form including electronic media without the written consent of RESI Informatik & Automation GmbH. Neither may it be transferred in any other language suitable for machines or data processing facilities. Also rights for reproduction through lecture, radio or television transmission are reserved.

This documentation as well as the corresponding software are subject to copyright law and protected by RESI company. All rights reserved.

1 History

Date	Editor	Description
08.03.14	DI HC Sigl	First English version
23.02.15	DI HC Sigl	Minor revisions

2 Contents

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 fulfill 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

Our RESI-DMX-ASCII converter is designed for controlling a DMX light system with a DMX universe of 512 DMX addresses. The control is done with simple ASCII commands or with MODBUS/RTU registers.

Our RESI-DMX-MODBUS converter is designed to control a DMX light system with the MODBUS/RTU protocol.

To control our DMX converter you need a host system with a serial interface (RS232 or RS485), which is able to send ASCII command strings and which can receive ASCII characters. This feature is implemented in almost any media control system like CRESTRON®, AMX® or CONTROL4®. But almost every standard PLC can handle serial ASCII interfaces. Therefore your converter can be integrated everywhere. If the host system offers a MODBUS/RTU master interface, our converter can be controlled via host registers.

- Connection of up to 512 DMX lamps (depending on the extension of the DMX network)
- DMX interface is galvanic insulated from the RS232 and RS485 interface
- ASCII interface: RS232 or RS485, 9600 up to 57600 bps, 8 data bits, no parity, 1 stop bit
- ASCII address is configurable via software
- MODBUS/RTU slave interface: RS232 or RS485, 9600 up to 57600 bps, 8 data bits, no parity, 1 stop bit
- MODBUS unit address is configurable via software
- DMX interface: 250kBaud, refresh rate 1/10s
- Power supply with 24VDC
- Power consumption <0.5W
- Mountable onto a EN50022 DIN rail

Type	Description	Voltage	Power	Weight
RESI-DMX-MODBUS	MODBUS/RTU slave to DMX converter with RS232/RS485 interface, supports up to 512 DMX lamps, DIP switches for configuration	24 V=	<0.5W	55 g
RESI-DMX-ASCII	ASCII and MODBUS/RTU slave to DMX converter with RS232/RS485 interface, supports up to 512 DMX lamps, DIP switches for configuration	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
ASCII/Modbus interface		**Factory settings**	
Protocol	ASCII or Modbus/RTU	ASCII/Modbus address	255
Type	RS232 or RS485	ASCII/Modbus baud rate	9600
Baud rate	9600 to 57600/8/N or E/1	ASCII/Modbus interface	RS232
Cable connection	Via clamps		
LED indicator	Yes		
Galvanic insulation to the DMX interface	Yes		
DMX bus interface			
Protocol	DMX512		
Baud rate	250kBits/s		
Refresh rate	10/Second		
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

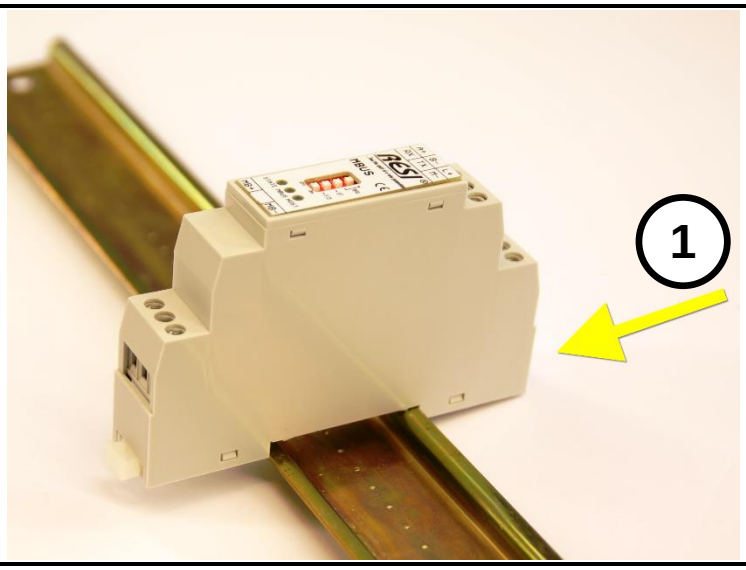
IT Accessories	
RESI-MODBUS-Configurator	Free configuration software for RESI Modbus devices. Download this software from www.RESI.cc

5 Mounting and Connections

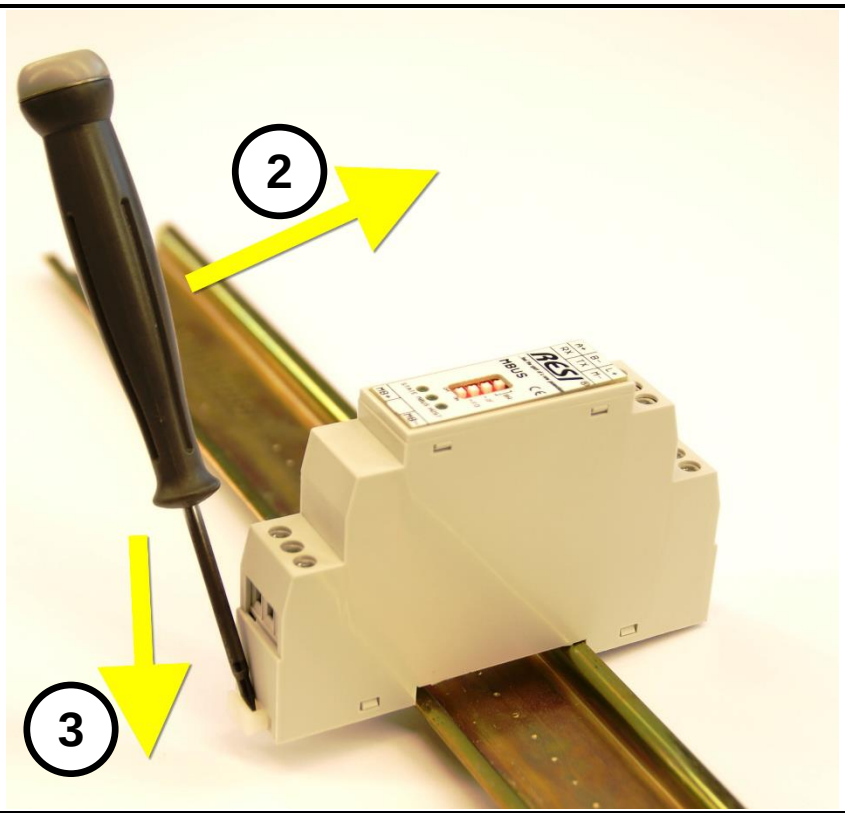
5.1 Assembling

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

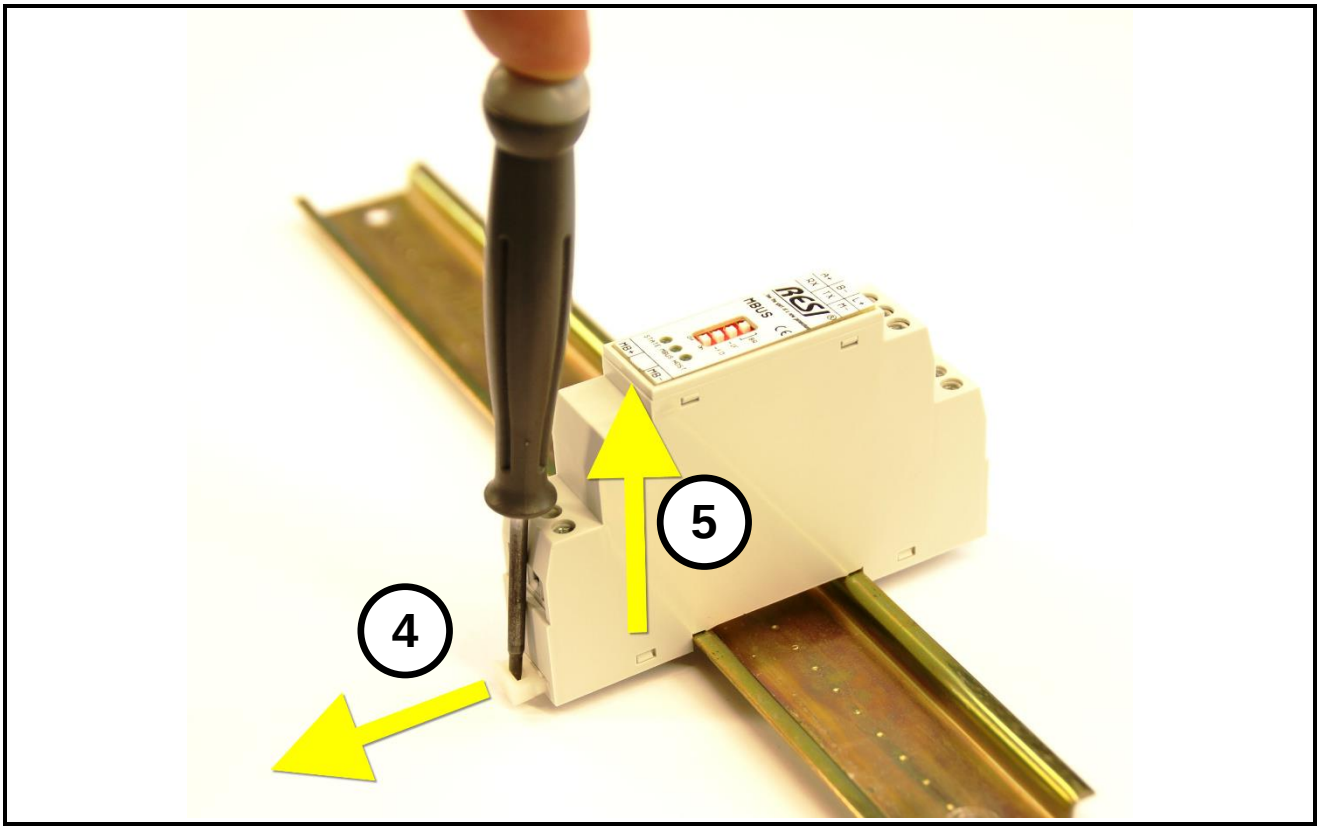
At first, put the converter with the top side on the DIN rail (1).



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 converter from the bar with while pulling it on the top side.



5.2 Clamps and LEDs

	RESI-DMX-ASCII
L+ M-	Power supply L+: 24 V= M-: Ground
A B	RS485 ASCII or Modbus/RTU slave interface A: DATA+ B: DATA-
RX TX M-	RS232 ASCII or Modbus/RTU slave interface RX: serial receive TX: serial transmit M-: Ground for RS232
D+ D- DG	DMX Output, D+ is the DMX+ wire, D- is the DMX- wire and DG is the DMX ground wire
STATE	State LED, flashes slowly, if the converter is ok, flashes fast if there is an internal error in the converter
DMX	DMX bus LED is always on, if the DMX converter sends cyclic data to the DMX lamps. Flashes fast (every 250ms), if the DMX sending is stopped by the user and no data is send to the DMX lamps.
HOST	HOST LED, Flashes shortly, if the HOST communicates with the converter.

Table: Description of connectors and LEDs of the RESI-DMX-ASCII converter

5.3 DIP switch settings

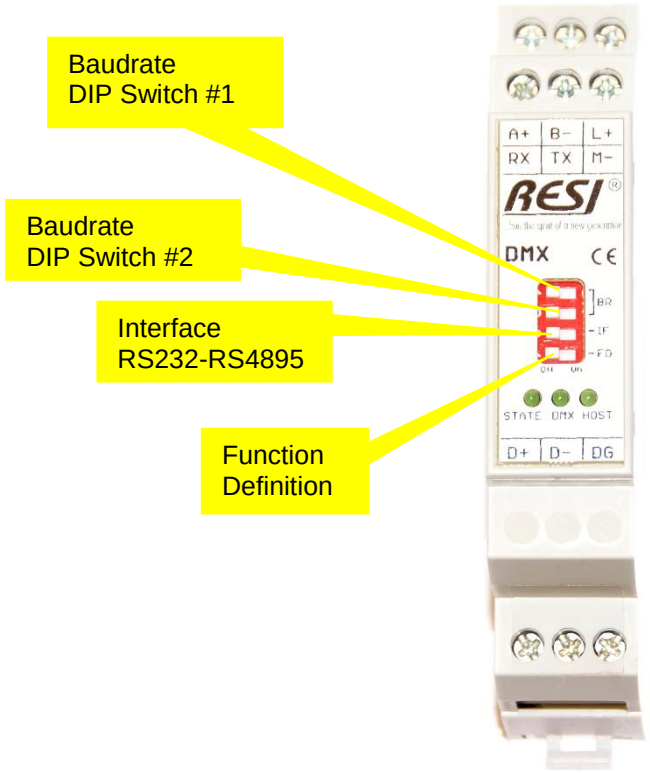


Illustration: Description of the DIP switch settings and LED status displays

DIP Switch	RESI-DMX-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 MODBUS/RTU slave or ASCII protocol OFF=RS232 ON=RS485
Function Definition FD	Selects special Functions OFF=Use Modbus RTU slave and ASCII bus address from FLASH memory ON=Use always slave address 255
HINT	After changing the DIP switches the converter reboots immediately, so no power off or on is necessary. After reboot all the LEDs are on for half a second to signal the power on sequence.

Table: Description of DIP Switch functions

5.4 Wiring diagram

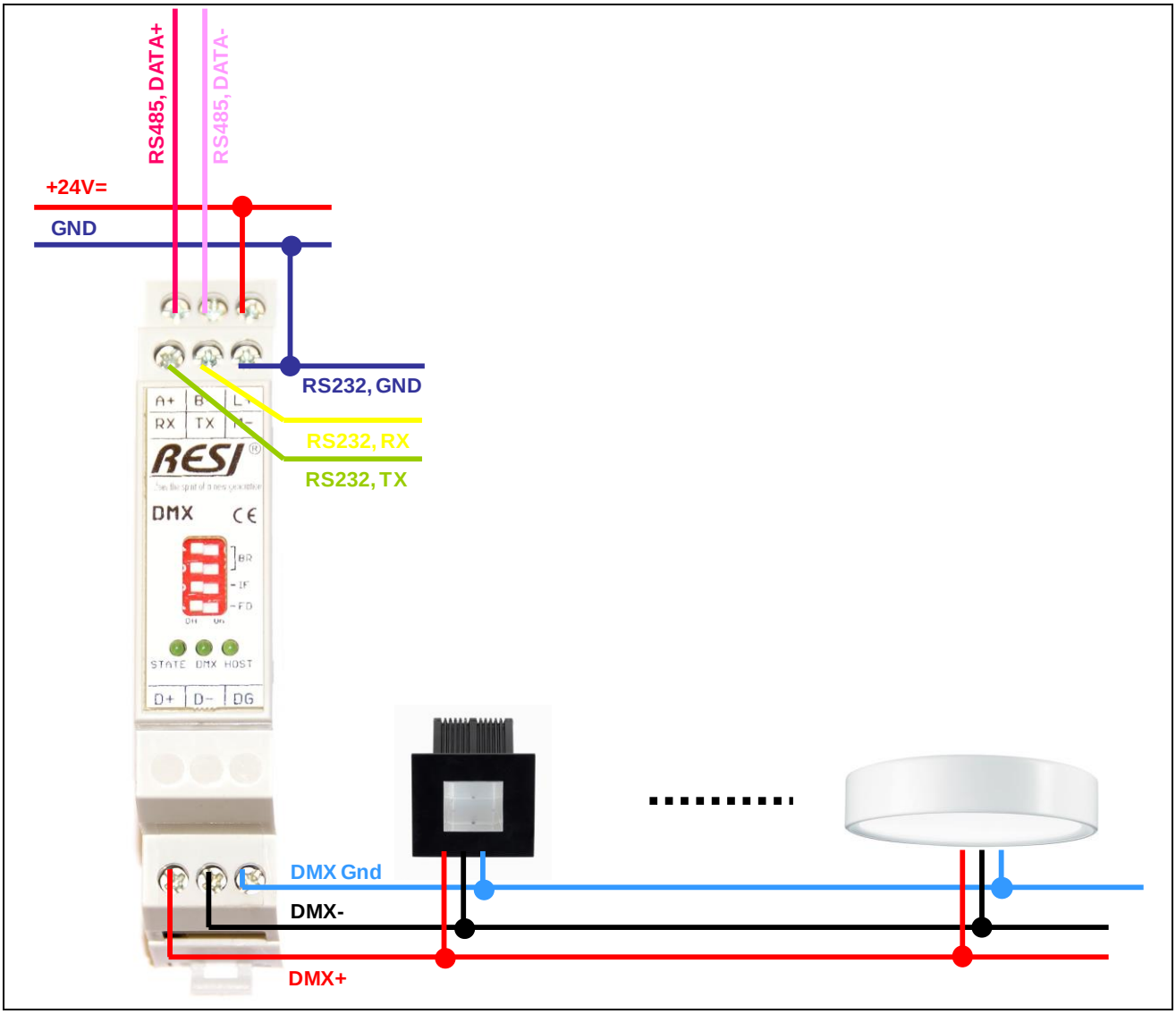


Illustration: Wiring diagram of the RESI-DMX-ASCII converter

6 Function description

The RESI-DMX-ASCII converter is designed to integrate a DMX light subsystem into an automation device very easily. The automation device has to offer a serial interface (RS232 or RS485), which is capable of sending and receiving simple ASCII messages. The converter also offers the possibility to control the 512 DMX lights via a MODBUS/RTU based serial interface of an automation device, because the converter is also a MODBUS/RTU slave. The RESI-DMX-MODBUS converter is designed to control a DMX light system with MODBUS/RTU protocol.

The bases for this integration are lamps with ballasts, which are compatible to the DMX standard.

Please refer to the DIN 56930-2, ANSI E1.1 norm for more operational details about the DMX light system.

If the converter works properly, the state LED flashes with a 1 second period. Is there an internal error in the converter, then the state LED flashes with a 250m period.

Is the DMX connection ok and the converter is cyclically sending data to the DMX lamps, the DMX LED is always on. If the user has stopped the cyclic sending of DMX data, then this LED flashes faster with 250ms cycle.

For the communication with ASCII strings, the ASCII messages starts with a special start character # (0x23, 35dec) and the message ends with a special end character (0x0d, 13dec or CARRIAGE RETURN). The answer of the converter uses the same start and end characters. Please refer the chapter ASCII command description.

For communication via MODBUS the following MODBUS functions are available:

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

Note: The functions READ HOLDING REGISTER and PRESET MULTIPLE REGISTERS are limited to max. 50 register per query!

More information about the MODBUS/RTU slave functionality see the chapter MODBUS Registers.

7 ASCII command description

7.1 Hint

Only our RESI-DMX-ASCII converter offers this ASCII protocol.

7.2 Overview

The RESI-DMX-ASCII converter communicates with simple ASCII strings. The following special characters are used in the following description:

stands for the **number sign** with the ASCII code 35dec or 0x23
: stands for the **colon** with the ASCII code 58dec or 0x3A
= stands for the **equal sign** with the ASCII code 61ec or 0x3D
- stands for the **minus sign** with the ASCII code 45dec or 0x2D
, stands for the **comma** with the ASCII code 44dec or 0x2C
<CR> stands for **CARRIAGE RETURN** with the ASCII code 13dec or 0x0D. We use also the symbol $_{CR}$ in the following text.
<SP> stands for the **SPACE** character. This is the ASCII code 32dec or 0x20. We use also the symbol □ in the following text.
<ADR> will be the converters **bus address**. This address can be send as a decimal or hexadecimal number and is separated with a colon (ASCII character 44dec or 0x2C) from the following command. Hexadecimal numbers will always start with the two ASCII characters 0x. Valid characters for the hexadecimal numbers are only ASCII characters ‚0'-‚9' 48-57dec, 0x30-0x39 and ‚A' to ‚F', 65dec-70dec, 0x41-0x46. All converters react to the broadcast address 0 and to its own configured address. With the DIP Switch FD you can switch very easily between the bus address 255 and your programmed bus address.

7.3 Communication protocol

The RESI_DMX_ASCII converter is a slave unit. This means, the converter sends no characters, unless it receives a valid ASCII request and answer to this request. So the communication is driven by the host. If you use the RS232 mode, you can avoid a bus address, because you can use only one unit on the RS232 (It's a point-to-point interface). If you use the RS485 mode, more than one converter can be on the RS485 line. Therefore you have to use individual bus addresses for each converter.

The command structure is like this:

The host sends a command or a command with parameter without a bus address:
#<Command><CR> or
#<Command>:<Parameters><CR>

The converter answers with the following telegram:
#<Answer><CR>

If you use a bus address, the protocol is like this:
#<ADR>,<Command><CR> or
#<ADR>,<Command>:<Parameters><CR>

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

The bus address lies in the range of 0dec to 255dec or 0x00 to 0xFF hexadecimal. The programming of the address is done with our free software tool RESI MODBUS configurator. Each converter reacts always to the bus address 0 as a broadcast address too!

For each command there are two different command strings. One is the long version of the command and one is the short form of the command, to send less characters. For example you can retrieve the software version of the converter with the command **VERSION** or with the command **VER**.

7.4 Request VERSION

This command retrieves the current software version of the converter.

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> stands for the actual software version eg. 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 und 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$_{CR}$**

7.5 Request converter TYPE

This command requests the actual type of the converter:

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> is the text of the actual type of the converter, currently RESI-DMX-ASCII

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

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

7.6 Stop the DMX bus update

After powering the DMX converter, the converter sends cyclically the DMX data with an update frequency of 10 updates per second. The output of the DMX data can be stopped with this command. Is the DMX update stopped, the DMX LED flashes with fast speed (every 250ms).

Host long version:
#DMX<SP>STOP<CR> or
#<ADR>,DMX<SP>STOP<CR>

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

Answer:
Is the command successful, the converter sends the following text:
#OK<CR> or
#<ADR>,OK<CR>

Is there a problem with the syntax of the command, the converter sends the following text:
#ERR:SYNTAX<SP>ERROR<CR> or
#<ADR>,ERR:SYNTAX<SP>ERROR<CR>

Samples:

Stop the DMX Bus update:
→ **#DMX□STOP$_{CR}$**
← **#OK$_{CR}$**

→ **#255,DSTOP$_{CR}$**
← **#255,OK$_{CR}$**

7.7 Start the DMX bus update

After powering the DMX converter, the converter sends cyclically the DMX data with an update frequency of 10 updates per second. The output of the DMX data can be stopped and restarted. This command restarts the DMX bus update cycle. Is the DMX update done by the converter, the DMX LED flashes with slow speed (every 1s)

Host long version:
#DMX<SP>START<CR> or
#<ADR>,DMX<SP>START<CR>

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

Answer:
Is the command successful, the converter sends the following text:
#OK<CR> or
#<ADR>,OK<CR>

Is there a problem with the syntax of the command, the converter sends the following text:
#ERR:SYNTAX<SP>ERROR<CR> or
#<ADR>,ERR:SYNTAX<SP>ERROR<CR>

Samples:

Restart the DMX bus update cycles:
→ **#DMX□START$_{CR}$**
← **#OK$_{CR}$**

→ **#255,DSTART$_{CR}$**
← **#255,OK$_{CR}$**

7.8 Request state of DMX bus update

After powering the DMX converter, the converter sends cyclically the DMX data with an update frequency of 10 updates per second. The output of the DMX data can be stopped and restarted. This command retrieves the current status of the DMX bus update cycle. Is the DMX update done by the converter, the DMX LED flashes with slow speed (every 1s). Is the DMX update cycle stopped, the DMX LED flashes fast with 250ms period.

Host long version:
#DMX<SP>RUN<CR> or
#<ADR>,DMX<SP>RUN<CR>

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

Answer:
Is the command request ok and the DMX bus update is active (DMX data is send to the connected DMX lamps), then this is the answer of the converter:
#DMX<SP>RUN:1,0x01<CR> or
#<ADR>,DMX<SP>RUN:1,0x01<CR>

Is the DMX bus update cycle turned off, this is the answer of the converter:
#DMX<SP>RUN:0,0x00<CR> or
#<ADR>,DMX<SP>RUN:0,0x00<CR>

Is there a problem with the syntax of the command, the converter returns:
#ERR:SYNTAX<SP>ERROR<CR> or
#<ADR>,ERR:SYNTAX<SP>ERROR<CR>

Samples:

Request the current status of the DMX bus update cycle:
→ **#DMX□RUN$_{CR}$**
← **#DMX□RUN:1,0x01$_{CR}$**

→ **#255,DRUN$_{CR}$**
← **#DMX□RUN:0,0x00$_{CR}$**

7.9 Define the amount of DMX data for DMX bus update

The converter supports a DMX universe with 512 DMX addresses. Internally the 512 DMX register are indexed from 0 to 511. This command defines, how many registers are send with each DMX bus update cycle to the DMX bus. By default the converter sends all 512 DMX registers. If you redefine this value, the setting gets lost after a restart of the converter.

Host long version:
#DMX<SP>SET<SP>LENGTH:<DMXLEN><CR> or
#<ADR>,DMX<SP>SET<SP>LENGTH:<DMXLEN><CR>

Host Short version:
#DSL:<DMXLEN><CR> or
#<ADR>,DSL:<DMXLEN><CR>

Answer:

Has the converter succeeded, the result text is:
#OK<CR> or
#<ADR>,OK<CR>

Is there a syntax problem with the command, the converter sends:
#ERR:SYNTAX<SP>ERROR<CR> or
#<ADR>,ERR:SYNTAX<SP>ERROR<CR>

<DMXLEN> describe the amount of registers to send to the DMX bus. A value between 1 and 511 is valid.

Samples:

Update only the first 32 DMX registers:
→ **#DMX□SET□LENGTH:32$_{CR}$**
← **#OK$_{CR}$**

→ **#255,DSL:0x20$_{CR}$**
← **#OK$_{CR}$**

7.10 Request the current amount of DMX registers sent to the DM bus

Our converter supports the full DMX universe with 512 DMX registers. This command returns the current amount of DMX registers, which are sent during a update cycle to the DMX bus. By default our converter sends all 512 DMX registers..

Host long version:
#DMX<SP>GET<SP>LENGTH<CR> or
#<ADR>,DMX<SP>GET<SP>LENGTH<CR>

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

Answer:
If the command is successful, the converter returns
#DMX<SP>LENGTH:<DMXLEN>,<DMXLEN><CR>
or
#<ADR>,DMX<SP>LENGTH:<DMXLEN>,<DMXLEN><CR>

<DMXLEN> is a value in the range of 1 to 512. The first value is the length as decimal value, the second value is the length as hexadecimal value.

Is there a syntax problem with the command, the converter sends:
#ERR:SYNTAX<SP>ERROR<CR> or
#<ADR>,ERR:SYNTAX<SP>ERROR<CR>

Samples:
Request the amount of DMX registers currently send by a DMX update cycle to the DMX bus:
→ **#DMX□GET□LENGTH$_{CR}$**
← **#DMX□LENGTH:512,0x200$_{CR}$**

→ **#255,DGL$_{CR}$**
← **#DMX□LENGTH:32,0x20$_{CR}$**

7.11 Set all DMX registers to the same value

Our converter supports a DMX universe with 512 registers. Those registers are indexed from 0 to 511. This command set ALL 512 registers to the same 8 Bit Value.

Host long version:
#DMX<SP>SET<SP>ALL:<DMXVALUE><CR> or
#<ADR>,DMX<SP>SET<SP>ALL:<DMXVALUE><CR>

Host short version:
#DSA:<DMXVALUE><CR> or
#<ADR>,DSA:<DMXVALUE><CR>

Answer:
If the command is successful, the converter returns:
#OK<CR> or
#<ADR>,OK<CR>

If there is a syntax problem with the command, the converter returns:
#ERR:SYNTAX<SP>ERROR<CR> or
#<ADR>,ERR:SYNTAX<SP>ERROR<CR>

<DMXVALUE> defines the new 8 bit value for all DMX registers in the range of 0..255dec or 0x00-0xFF.

Samples:
Set all DMX registers to 0:
→ **#DMX□SET□ALL:0$_{CR}$**
← **#OK$_{CR}$**

Set all DMX registers to 255:
→ **#255,DSA:0xFF$_{CR}$**
← **#OK$_{CR}$**

7.12 Set DMX Registers to individual values

Our converter supports the whole DMX universe with 512 DMX registers. We index the DMX registers from 0 to 511. This command fill a DMX register area with new 8 bit values. With one command you can set a maximum of 32 DMX registers with new values. If you want to set up more than 32 DMX registers, the host has to send more than one command.

Host long version:
#DMX:<DMXSTART>=<V1>,<V2>, … ,<Vn><CR>
or
#<ADR>,DMX:<DMXSTART>=<V1>,<V2>, … ,<Vn><CR>

Host short version:
#D:<DMXSTART>=<V1>,<V2>, … ,<Vn><CR>
or
#<ADR>,D:<DMXSTART>=<V1>,<V2>, … ,<Vn><CR>

Answer:
If the command is successful, the converter returns:
#OK<CR> or
#<ADR>,OK<CR>

If there is a syntax problem with the command, the converter returns:
#ERR:SYNTAX<SP>ERROR<CR> or
#<ADR>,ERR:SYNTAX<SP>ERROR<CR>

<DMXSTART> is the start index in the DMX register buffer. You can use an index between 0 and 511.
<V1>,<V2>,<Vn> stand for the new 8 bit values of the consecutive DMX registers in the range of 0..255dec or 0x00 to 0xFF.
Samples:

Set RGB DMX lamp with start index 0 (Red to 100, Green to 255, Blue to 50):
→ **#DMX:0=100,255,50$_{CR}$**
← **#OK$_{CR}$**

Set RGB DMX lamp with starting index 3 (Red to 255, Green to 255, Blue to 255):
→ **#255,D:3=0xFF,0xFF,0xFF$_{CR}$**
← **#OK$_{CR}$**

7.13 Request content of DMX Registers

Our converter supports the whole DMX universe with 512 DMX registers. We index the DMX registers from 0 to 511. This command returns the current contents of a DMX register range starting with a start index in the DMX register buffer. With one command you can read out the values of a maximum of 32 DMX registers. If you want to read out more than 32 DMX registers, the host has to send more than one command.

Host long version:
#SHOW<SP>DMX:<DMXSTART>=<DMXCOUNT><CR>
or
#<ADR>,SHOW<SP>DMX:<DMXSTART>=<DMXCOUNT><CR>

Host short version:
#SD:<DMXSTART>=<DMXCOUNT><CR>
or
#<ADR>,SD:<DMXSTART>=<DMXCOUNT><CR>

Answer:
If the command is successful, the converter sends back:
#DMX:<DMXSTART>=<V1>,<V2>, … ,<Vn><CR>
or
#<ADR>,DMX:<DMXSTART>=<V1>,<V2>, … ,<Vn><CR>

If there is a syntax problem with the command, the converter returns:
#ERR:SYNTAX<SP>ERROR<CR> or
#<ADR>,ERR:SYNTAX<SP>ERROR<CR>

<DMXSTART> is the start index in the DMX register buffer. You can use an index between 0 and 511.
<DMXCOUNT> defines the amount of DMX registers. The content of those registers are returned by this command. A maximum of 32 registers are allowed by this command.
<V1>,<V2>,<Vn> stand for the returned DMX registers contents in the range of 0x00-0xFF. This command returns ALWAYS hexadecimal numbers, NO decimal numbers.

Samples:

Read the content of the DMX registers 0 to 2 (eg a DMX RBG lamp):
→ **#SHOW□DMX:0=3$_{CR}$**
← **#DMX:0=0x64,0xFF,0x32:$_{CR}$**

Read the content of 10 DMX registers starting with index 3:
→ **#255,SD:3=10$_{CR}$**
← **#DMX:3=0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x9$_{CR}$**

8 MODBUS - Registers

8.1 Hint

Both converters, the RESI-DMX-ASCII and the RESI-DMX-MODBUS offer a MODBUS/RTU slave protocol for communication.

8.2 Overview

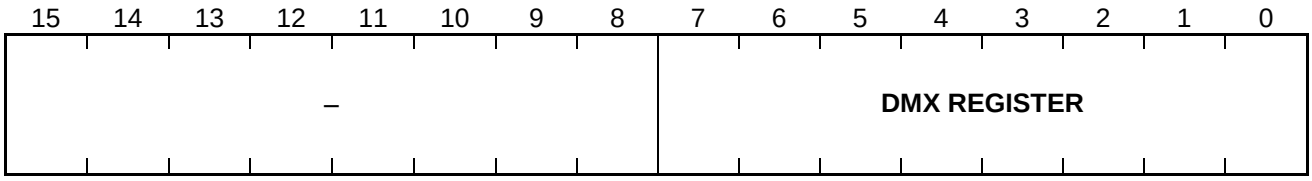
Register Range	***Function***
4x00001 4x00512	DMX Universe with 512 DMX register
4x00513 4x05999	unused
4x06000 4x06009	Converter internally
4x06010 4x09999	Unused
4x10000 4x10099	DMX special commands
4x10100 4x59999	Unused
4x60000 4x65536	Converter internally

Note:
Register that are not described in detail below, but are not explicitly stated as inoperable in this table, are reserved for internal commands and updates as well as program extensions in the future and therefore must not be read or written. The index into the holding registers start with 1. A maximum of 50 registers can be read with one telegram. All inoperable registers return the value 0 when access read.

MODBUS Register

8.3.1 Read / Write a DMX 8 Bit register in the DMX universe

Read Write Holding Register (4x**00001-**4x**00512**) Factory setting: 0

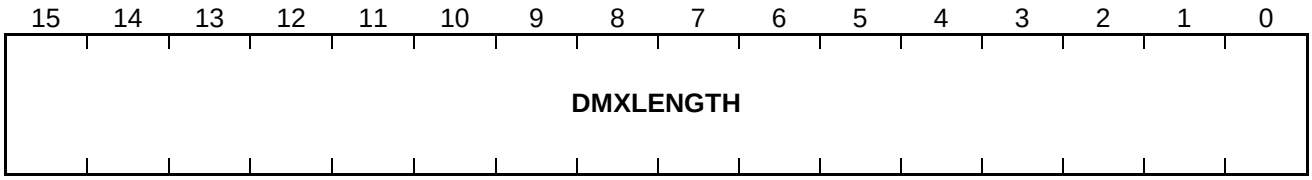


Bit	Description
DMXREGISTER	**Write: New value for the 8 bit DMX register** **Read: The current value of the 8 bit DMX register** Writing to this register sets the new 8 bit value of the corresponding DMX register. This value will be send to the DMX bus cyclically. Reading this register returns the current 8 bit value of the corresponding DMX register.

Note:
How many DMX registers are sent to the DMX bus is defined with the DMXLENGHT register. This register is read/writeable via the MODBUS.

8.3.2 DMX Set amount of sending DMX registers to the DMX bus

Read Write Holding Register (4x**10000**) Factory setting: 512



Bit	Description
DMXLENGTH	**Read: Amount of DMX registers, which are currently send to the DMX bus with a DMX update cycle** **Write: Define the new amount of DMX registers, which should be sent to the DMX bus** DMXLENGTH is a value between 1 and 512 and defines the amount of DMX registers, which are sent cyclically to the DMX bus. After a reset or power on, this register has always the value 512 and the converter sends ALL 512 DMX registers.

8.3.3 DMX start/stop sending DMX bus data

Read Write Holding Register (4x**10001**) Factory setting: 1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMXRUN															

Bit	Description
DMXRUN	**DMX bus output** Write: Defines the operation mode of the DMX converter. Write 1 to this register, if you want to activate the DMX bus update, write 0 to stop the DMX bus update Read: Returns the current status of the bus update. If the converter currently sends DMX data to the DMX bus, 1 is returned. If no data is updated on the DMX bus, 0 is returned.

8.3.4 Fill al DMX registers with same value

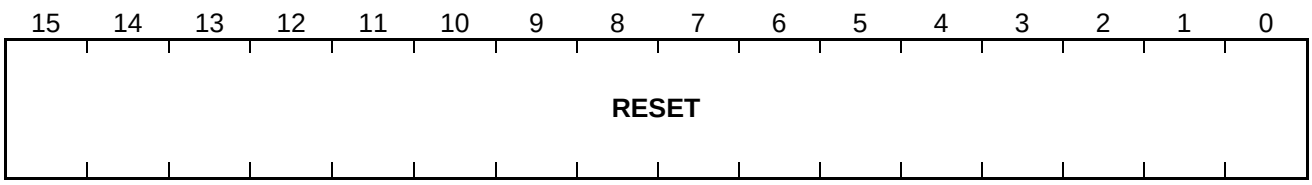
Read Write Holding Register (4x**10002**) Factory setting: 0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMXFILL															

Bit	Description
DMXFILL	**DMX 8 Bit value used for filling ALLE DMX Register** Write: Fills ALL 512 DMX registers with the new 8 Bit value. Read: Returns the last used 8 bit fill value.

8.3.5 Converter Reset

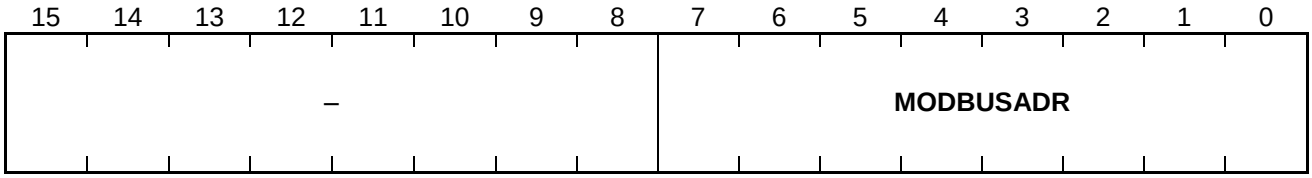
WRITE only Holding Register (4x06001) Factory setting: 0



Bit	Description
RESET	**Reset of the converter** Write: If you write 1 into this register, the converter does a software reset.

8.3.6 MODBUS address

READ / WRITE holding register (4x65222) factory settings: 255

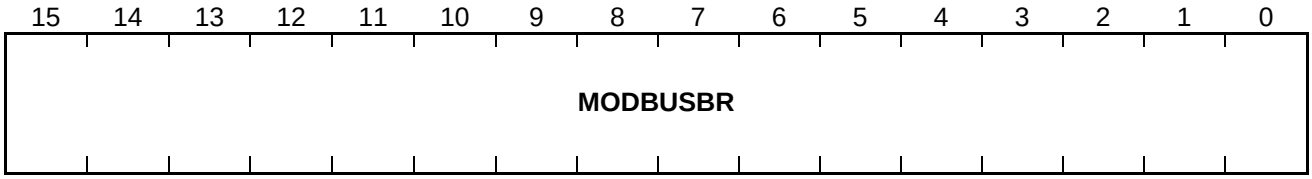


bit	description
MODBUSADR	**Modbus address** defines the MODBUS address of the converter. Range: 0 to 255

Note:
To overwrite the old values the converter must be restarted.

8.3.7 MODBUS baud rate

READ / WRITE holding register (4x65223) factory settings: 1



bit	description
MODBUSBR	**Modbus Baudrate –** defines the Modbus baud rate of the converter. 0: 9600 bps 1: 19200 bps 2: 38400 bps 3: 57600 bps

Note:
If the set value is greater than 3, the standard baud rate of 19200 bps is used!
To overwrite the old values it is necessary to save the new configuration and to restart the converter.

9 Specifications

9.1 Dimensions

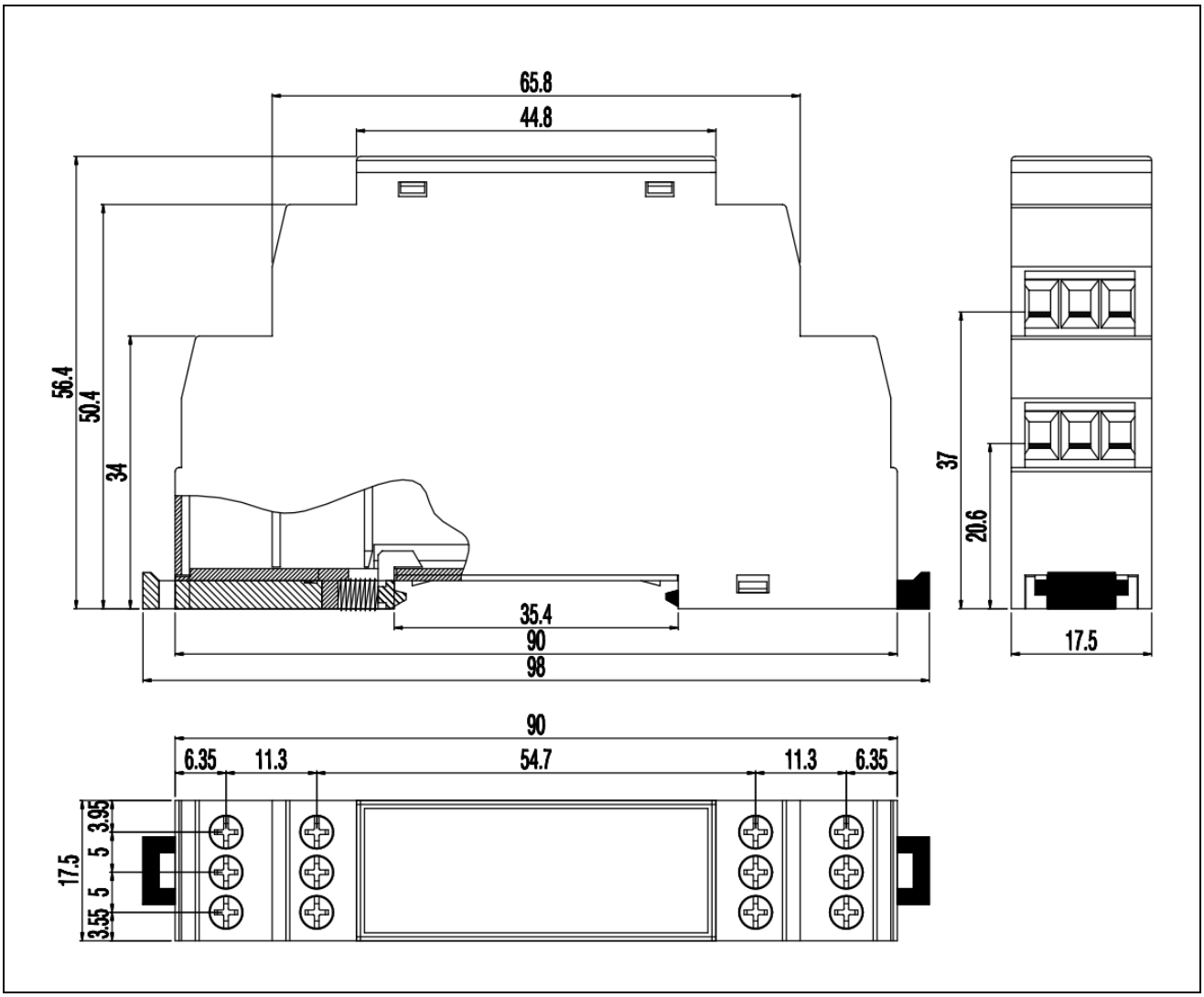


Illustration: dimension illustration in mm

Dimensions	
Enclosure dimensions L x W x H (mm)	17,5 x 90 x 58
Weight	60 g
Colour	Grey RAL7035
Material	PA - UL 94 V0
Protection class	IP20 based on DIN 40050/EN 60529

Table: Data of enclosure