2.1.1 Modbus Bridge

2.1.1.1 Explanation Configuration

Baud rate:	The transmission speed of the bus data. Usually to be found in the manual of the terminal device. There can only be one baud rate for each RS-Bridge.
Databits:	This defines how many relevant bits a character has. It can also be taken from the instructions of the terminal device.
Stopbits:	Marks the end of the character. Can also be found in the manual of the terminal device.
Parity:	In message transmission, a simple method of error detection. To be found in the manual of the terminal device.
Stopbyte:	The character that marks the end of a message. Specified as hexadecimal character of the ASCII character (see ASCII table).
Minimum Time between messages:	If no “Stopbyte“ is given, a minimum time must elapse between messages in order to recognize the end of a message. Here e.g. 100ms are sufficient.

Calculation example:

9600 Baud = 9600 character per second.

1 character is transmitted in approx. 1/9600 s.

It must take two and a half times the time for a character to be sent before the next message can be sent.

1/9600 = 0,1041 ms * 2,5 → 0,25 ms would theoretically be sufficient.

2.1.1.2 Create new device

Name: The name of the device, for example, air conditioner.

Description: Description of the device in the device view.

Fixed device address: Modbus RTU requires that the device address is appended to the beginning of each message. If you select the address, every message will automatically insert the address in the first place of the message.

If the checkmark is not set, the address, if available, must be set manually for each message.

CRC-TYPE: Another error detection method assumed by the Modbus protocol. Two bytes are appended to the end of each message. The end devices compare the CRC bytes to detect whether an error occurred during transmission. Disable if end devices do not have CRC control. Missing CRC may cause ignoring all incoming messages at the end device.

2.1.1.3 Create commands

Name: Name of the command, e.g. On/Off.

Description: Description of the command.

Digital: Selection option whether the generated data point is digital or analog.

COMEXIO Input: Here it is defined whether it is an output or an input.

- Output: COMEXIO sends a command to the end device (e.g. switch-on command or request for a status).
- Input: COMEXIO receives response from a terminal device (e.g., status feedback).

Response length (optional): Additional security to be able to assign incoming messages in advance (leave default=0 to ignore the function).

Answer pattern (optional): If the response length matches with the sent message, it can be checked in the response pattern if this message matches with the command.

The response pattern is a LUA pattern.

See Example 2: Command 2: Interpret.

Assignment: A clear assignment of requirement to interpreter and conversely is mandatory.

Function code: A selection of predefined Modbus function codes:

- **None:** The command must be written by yourself.
- **Read Holding:** Standard read command for holding registers.
- **Read Input:** Read command for input register.
- **Write Coil:** It is written in a coil. Can only be On (0xFF00) or Off (0x0000). Is used if you can see from the instruction that it is to be written into the coil.
- **Write Register:** Default write command for Holding Register.
- **Write multiple:** Write command for multiple holding registers. The terminal device must support the write command for multiple registers.

Register address: The respective register which is read or described. Refer to the manual of the terminal device.

Number of registers: Relevant specification for function code “Read holding“, “Read input“ and “Write multiple“. Specifies how many registers are read or written from the “Register address“.

Data: Only relevant for write commands. You can either write a fixed value into the register (fixed value) or pass a value via the COMEXIO Logic, which is then written into the register (input).

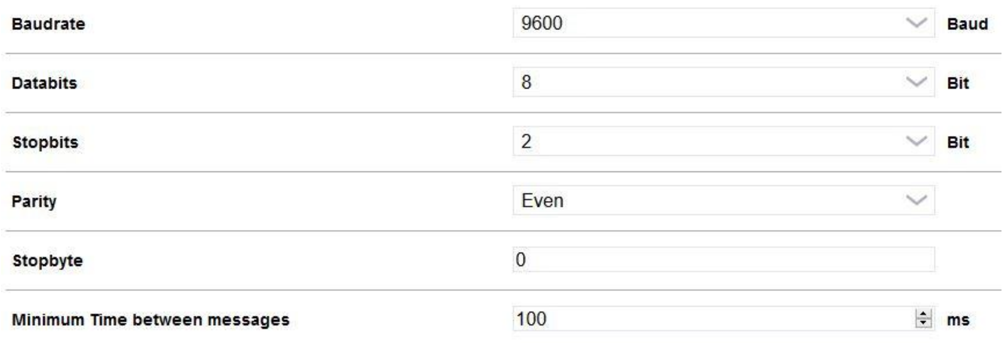
The fixed value can be entered in HEX in the line below.

Interpreter code: The resulting code. Must start with “function rs_interpreter (v)“ and end with “end“.

ATTENTION: Manual code changes are discarded without comment when the selection fields in the command mask are changed.

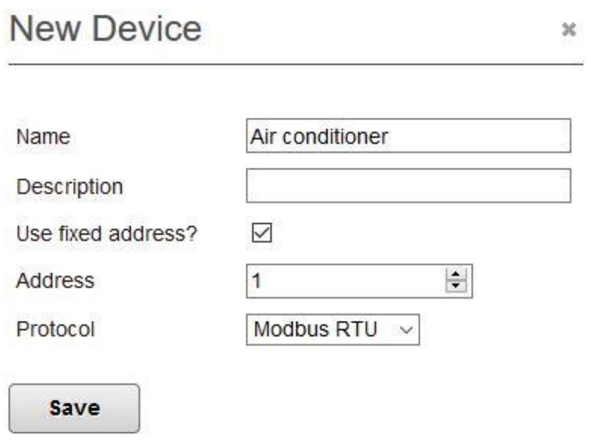
2.1.1.3.1 Example

This is about a Modbus RTU (slave) interface and it is connected to a RS485 bus line. 8N2 (8N1 compatible) communication (8 data bits, no parity and 2 stop bits) is carried out with several available baud rates: 2400 Bd, 9600 Bd (default), 19200 Bd and 57600 Bd.



Baudrate	9600	Baud
Databits	8	Bit
Stopbits	2	Bit
Parity	Even	
Stopbyte	0	
Minimum Time between messages	100	ms

Screenshot from the Modbus Bridge (configuration)



New Device

Name	Air conditioner
Description	
Use fixed address?	☑
Address	1
Protocol	Modbus RTU

Screenshot from the Modbus Bridge (create device)

Control and status register (terminal device):

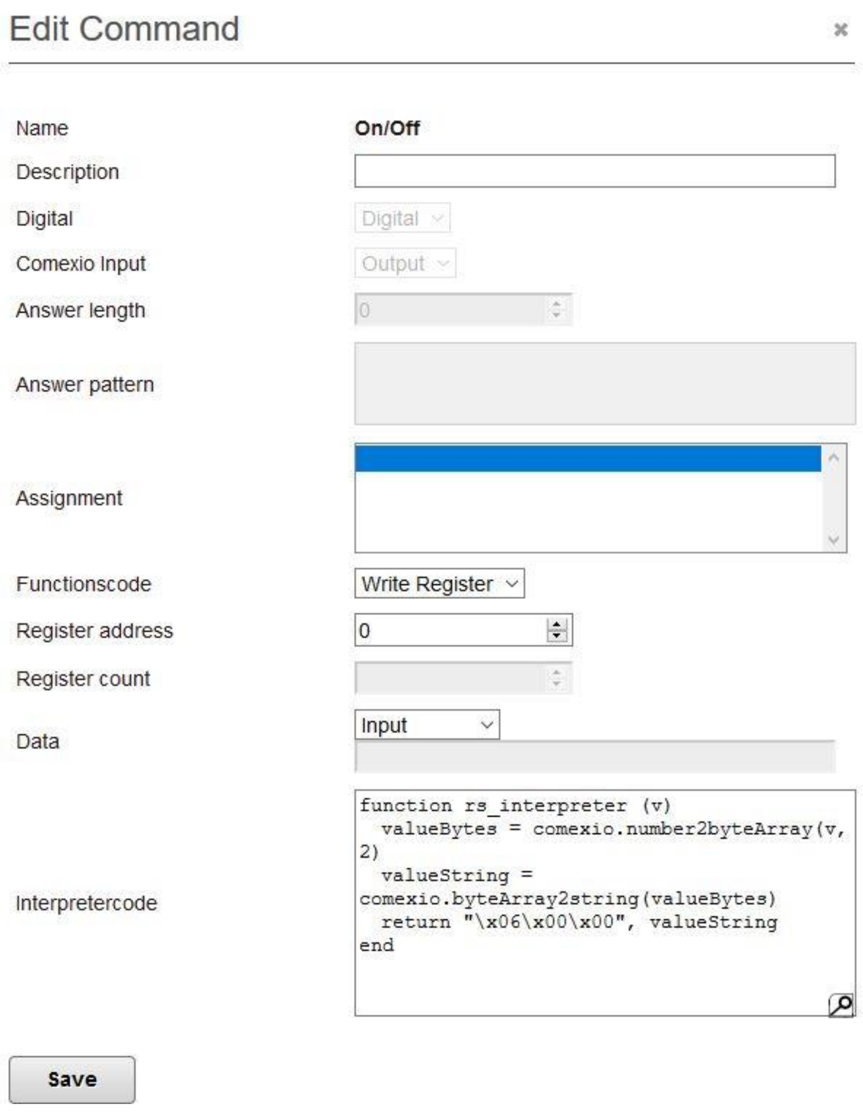
Registry address (protocol address)	Read/Write	Description
0	R/W	Room air conditioner On/Off • 0: On • 1: Off
1	R/W	Room air conditioner operating mode *1 • 0: Auto • 1: Heating • 2: Drying / Dehumidification • 3: Blower • 4: Cooling
2	R/W	Air conditioner blower level *1 • 0: Auto • 1: Low • 2: Medium 1 • 3: Medium 2 4: High
3	R/W	Air conditioner Discharge direction (Vane) Position *1 • 0: Auto • 1: Horizontal • 2: Position 2 • 3: Postition 3

		• 4: Position 4 • 5: Vertical • 6: Swing
4	R/W	Air conditioner setpoint room temperature *2*3 • 16-32 °C (°C/x10 °C) • 60-90 °F
5	R	Air conditioner actual value room temperature *2*3 • 10-38 °C (°C/x10 °C) • 50-100 °F

ATTENTION: The address of the terminal device was defined when it was created and will not change again. The address is automatically appended by COMEXIO at the first position of each message. For this reason, the address does not appear in the code.

2.1.1.3.2 Example 1: Switching on the air conditioner

The device has been successfully configured and created. The command to switch the air conditioner on or off looks like this:



Edit Command

Name	On/Off
Description	
Digital	Digital
Comexio Input	Output
Answer length	0
Answer pattern	
Assignment	
Functionscode	Write Register
Register address	0
Register count	
Data	Input
Interpretercode	see below

```
function rs_interpreter (v)
  valueBytes = comexio.number2byteArray(v,
2)
  valueString =
comexio.byteArray2string(valueBytes)
  return "\x06\x00\x00", valueString
end
```

Save

function rs_interpreter (v)

The header of the function.

valueBytes = comexio.number2byteArray(v,2)

The number passed by the COMEXIO system is split into an array of individual bytes. The bracket **(v,2)** splits the number into at least two bytes.

valueString = comexio.byteArray2string(valueBytes)

Since an array cannot simply be output, the array must be converted to a string.

return "\x06\x00\x00", valueString

The string is output. Consists of three + the length of “valueString“ bytes (consists here of at least two bytes).

1. The function code x06 → Writing Register.
2. The address of the register consisting of two bytes (bytes 2 and 3).
3. The address of the register consisting of two bytes (bytes 2 and 3).
4. The value which has to be written. Here at least two bytes consisting of “valueString“.

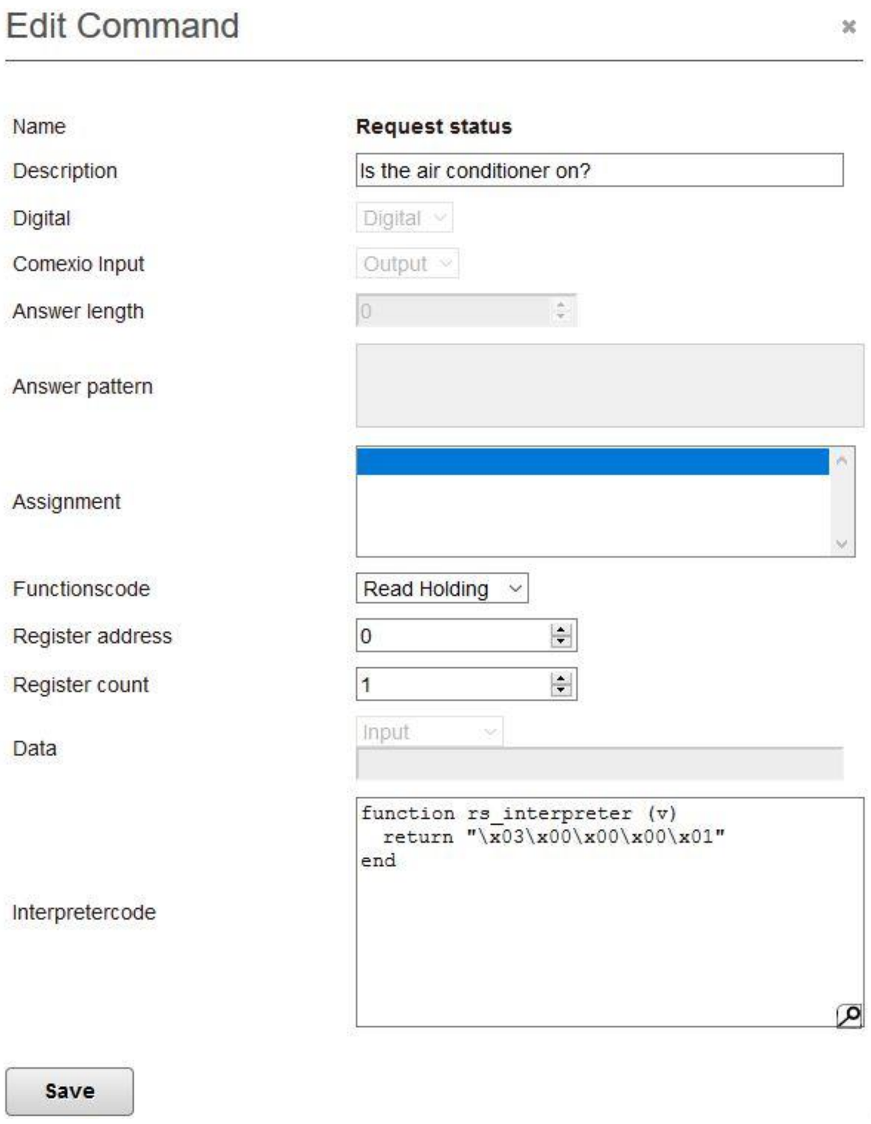
end

The end of the function.

2.1.1.3.3 Example 2: Read out status

To be able to read out the status of the air conditioner, two commands must be created. The 1st command is the request for the status and the 2nd command is used to interpret the response.

Command 1: Request



Name	**Request status**
Description	Is the air conditioner on?
Digital	Digital
Comexio Input	Output
Answer length	0
Answer pattern	
Assignment	
Functionscode	Read Holding
Register address	0
Register count	1
Data	Input
Interpretercode	

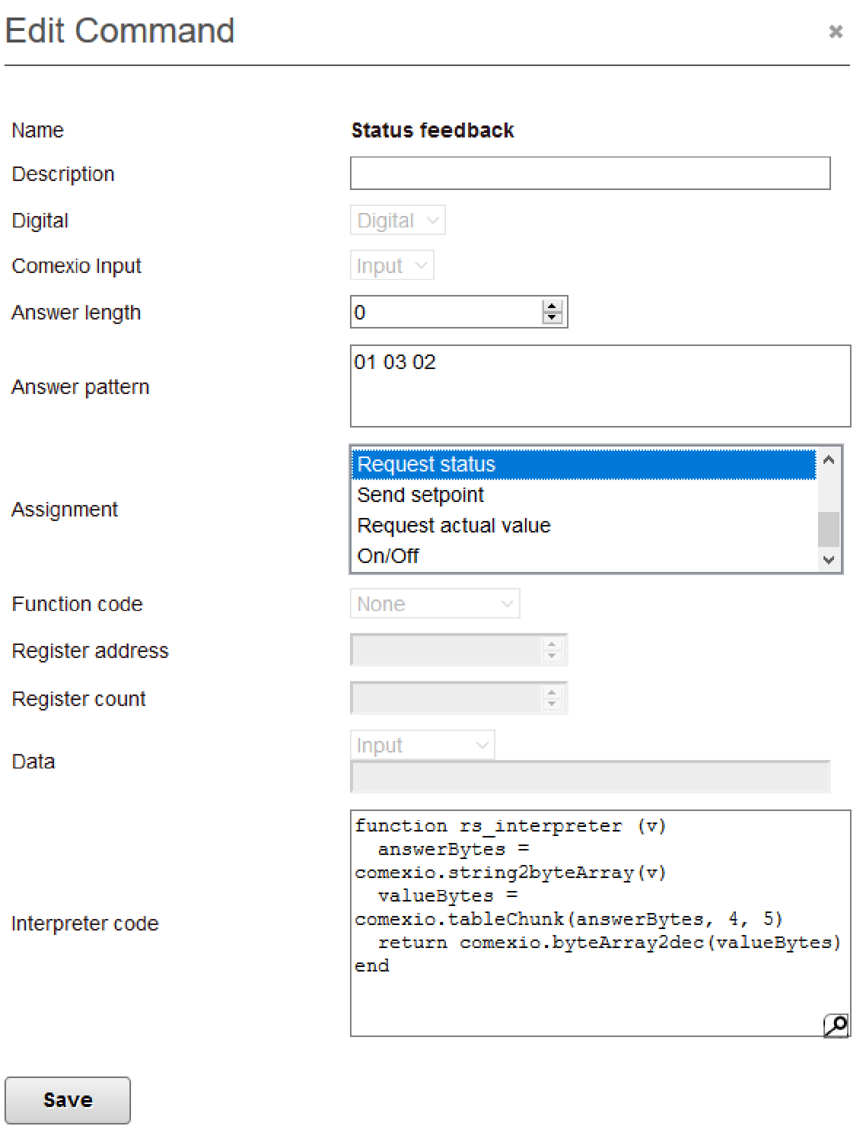
```
function rs_interpreter (v)
  return "\x03\x00\x00\x00\x01"
end
```

return "\x03\x00\x00\x00\x01"

The output value consisting of five bytes.

1. The function code in this case x03 → Read Holding Register.
2. The address of the register consisting of two bytes (bytes 2 and 3).
3. The address of the register consisting of two bytes (bytes 2 and 3).
4. The number of registers to be read consisting of two bytes (bytes 4 and 5).
5. The number of registers to be read consisting of two bytes (bytes 4 and 5).

Command 2: Interpret



Edit Command

Name	**Status feedback**
Description	
Digital	Digital
Comexio Input	Input
Answer length	0
Answer pattern	01 03 02
Assignment	Request status
Function code	None
Register address	
Register count	
Data	Input

Interpreter code:

```
function rs_interpreter (v)
  answerBytes =
comexio.string2byteArray(v)
  valueBytes =
comexio.tableChunk(answerBytes, 4, 5)
  return comexio.byteArray2dec(valueBytes)
end
```

Answer pattern: In the message of the terminal device the byte sequence “03 02“ must occur. “03“ indicates the function code of the request and “02“ the length of the sent data.

Assignment: This command is uniquely assigned to the “Request status“ command.

answerBytes = comexio.string2byteArray(v)

The answer of the device is written into an array with the single byte values.

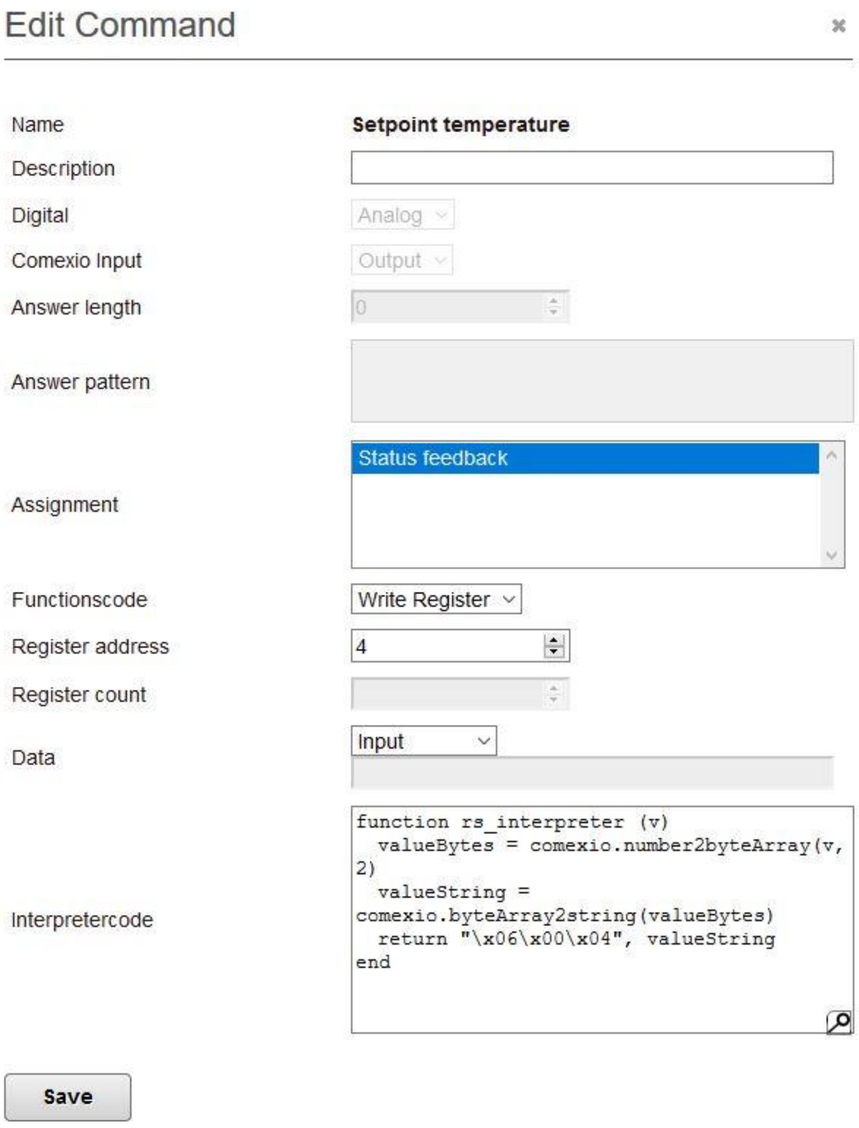
valueBytes = comexio.tableChunk(answerBytes, 4, 5)

The array is reduced to the relevant places. Position 4 and 5 of the array contain the return value of the sent message.

return comexio.byteArray2dec(valueBytes)

The reduced array is converted to a decimal number and output.

2.1.1.3.4 Example 3: Send setpoint



Edit Command

Name	**Setpoint temperature**
Description	
Digital	Analog
Comexio Input	Output
Answer length	0
Answer pattern	
Assignment	Status feedback
Functionscode	Write Register
Register address	4
Register count	
Data	Input

Interpretercode

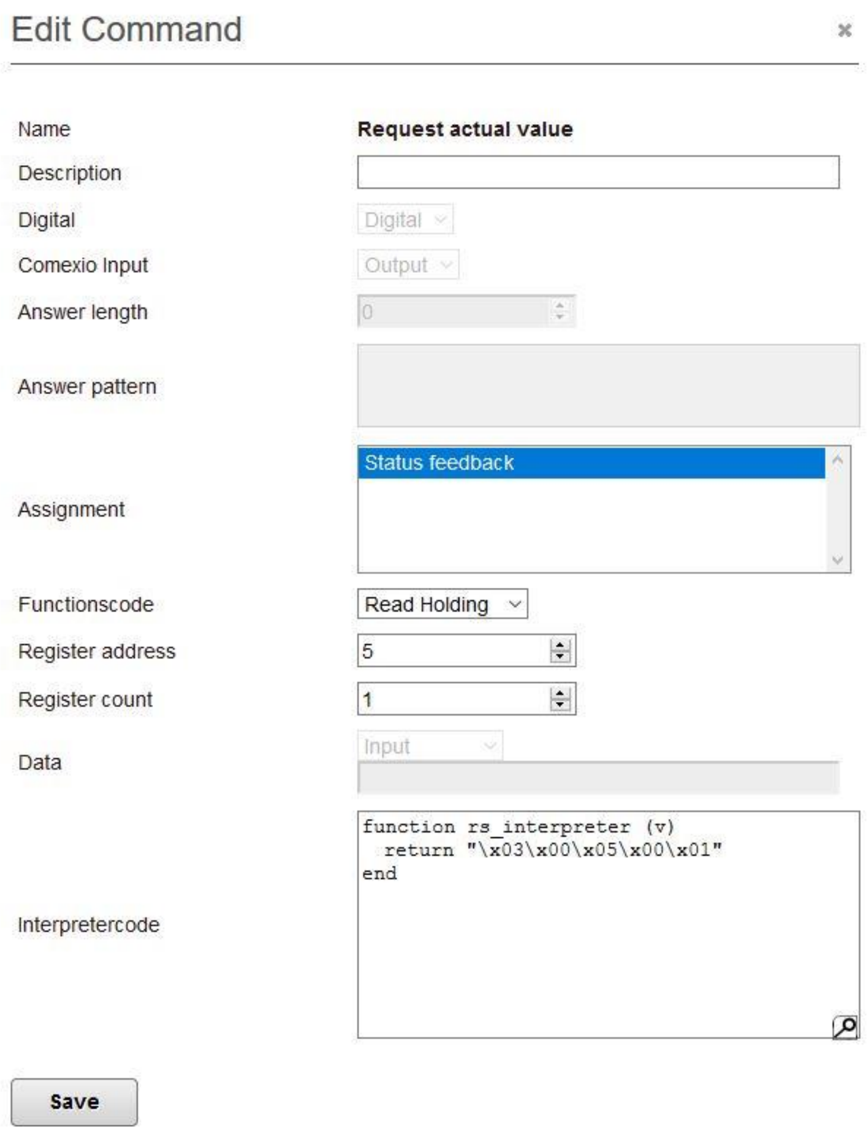
```
function rs_interpreter (v)
  valueBytes = comexio.number2byteArray(v,
2)
  valueString =
comexio.byteArray2string(valueBytes)
  return "\x06\x00\x04", valueString
end
```

The following lines were used to set up that only values between 16 and 32 can be sent (see excerpt of the instructions). Values outside this range are not sent.

```
if(v < 16 or v > 32) then
   return
end
```

2.1.1.3.5 Example 4: Request actual value

Command 1: Request actual value

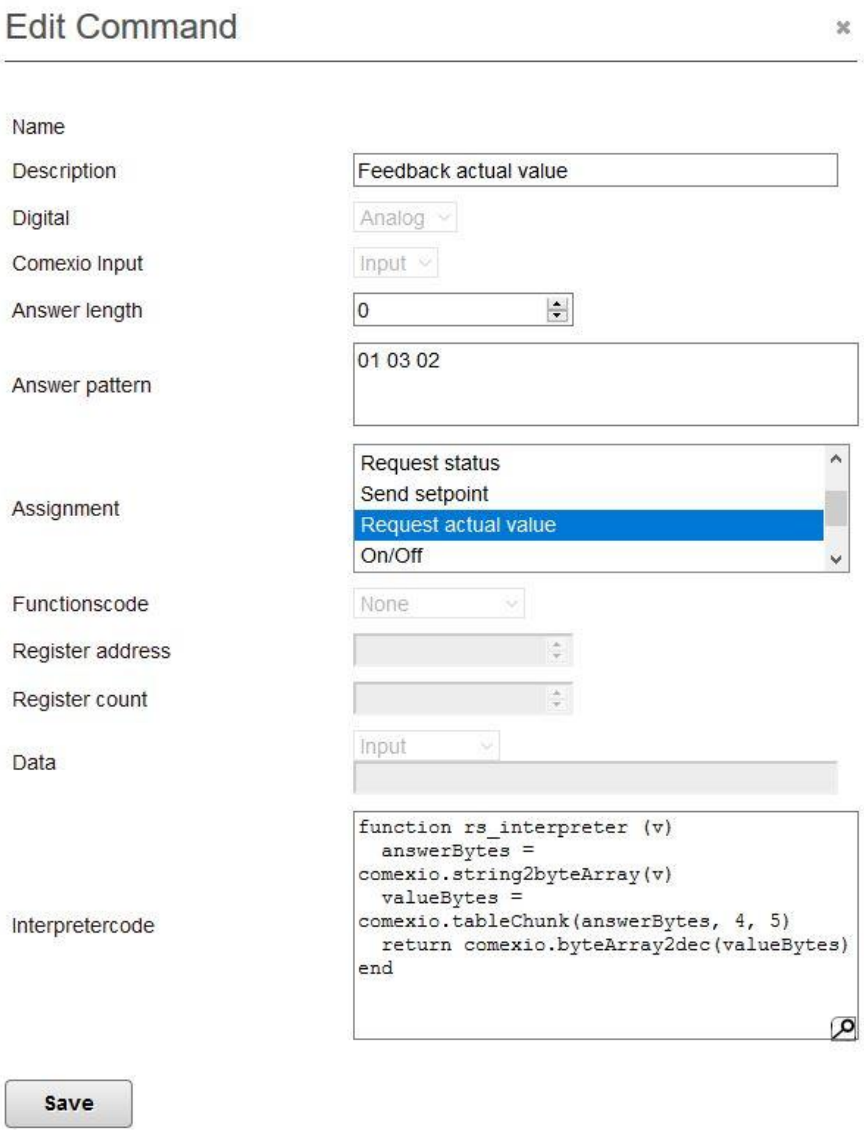


Edit Command

Name	**Request actual value**
Description	
Digital	Digital
Comexio Input	Output
Answer length	0
Answer pattern	
Assignment	Status feedback
Functionscode	Read Holding
Register address	5
Register count	1
Data	Input
Interpretercode	see below

```
function rs_interpreter (v)
  return "\x03\x00\x05\x00\x01"
end
```

Command 2: Interpret actual value


Edit Command

Name	
Description	Feedback actual value
Digital	Analog
Comexio Input	Input
Answer length	0
Answer pattern	01 03 02
Assignment	Request status Send setpoint Request actual value On/Off
Functionscode	None
Register address	
Register count	
Data	Input
Interpretercode	(see below)

```
function rs_interpreter (v)
  answerBytes =
comexio.string2byteArray(v)
  valueBytes =
comexio.tableChunk(answerBytes, 4, 5)
  return comexio.byteArray2dec(valueBytes)
end
```