Easergy P3

Modbus configuration instructions

Application Book

P3APS18025EN

04/2019

www.schneider-electric.com

Table of Contents

1. About this document

1.1. Purpose

The purpose of this document is to describe the configuration and use of the Modbus RTU and Modbus TCP in Easergy P3 protection devices.

1.2. Abbreviations and terms

Table 1 - Abbreviations and terms

Term	Description
COMTRADE	Common format for transient data exchange. File format for storing oscillography and status data related to transient power system disturbances.
DHCP	Dynamic host configuration protocol
EF	Earth fault
HR	Holding register
NTP	Network time protocol
OC	Overcurrent
RTC	Real-time clock
RTU	Remote terminal unit
SNTP	Simple network time protocol
TCP	Transmission control protocol

2. Modbus overview

The Modbus protocol is a master-slave protocol typically implemented on an RS-232 or RS-485 physical interface. There are two versions of the serial Modbus protocol: Modbus RTU and Modbus ASCII, of which Modbus RTU is available in Easergy P3 devices. There is also a TCP/IP implementation of the protocol, Modbus TCP.

Modbus TCP is simply the Modbus RTU protocol with a TCP interface that runs on Ethernet. TCP/IP refers to the transmission control protocol and Internet Protocol which provide the transmission medium for Modbus TCP messaging. In practice, Modbus TCP embeds a standard Modbus data frame into a TCP frame.

Modbus RTU and Modbus TCP can be used to transfer the following types of data:

- events
- status information
- measurements
- control commands
- clock synchronization
- protection settings
- disturbance recordings

In the master-slave setup of Modbus, only the master can initiate transactions (queries). The slaves respond by supplying the requested data to the master or by taking the action requested in the query. The master can address individual slaves or initiate a broadcast message to all slaves. The address space for slaves in Modbus RTU ranges from 1-247 on one data link. Address 0 is reserved for broadcast.

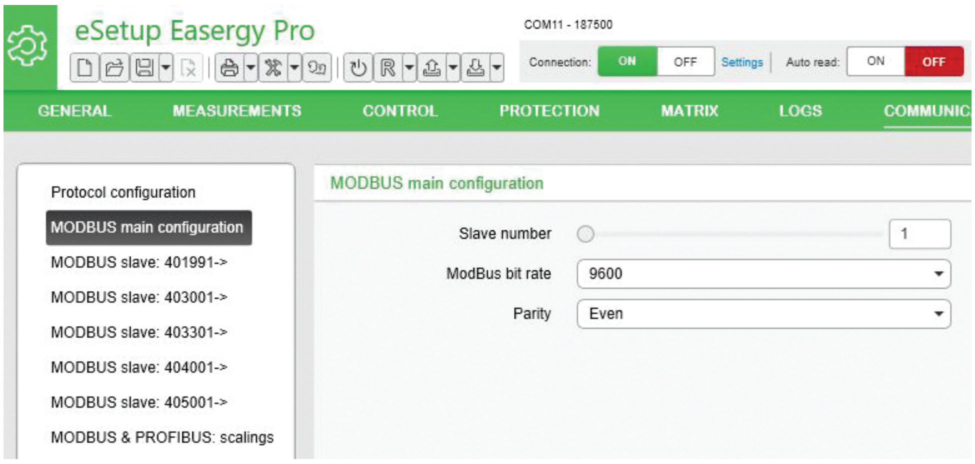
The Modbus protocol defines four different categories of data. The mapping of data to these categories is implementation dependent. In Easergy P3 devices, all data is mapped to the holding registers.

Table 2 - Modbus data organisation

Primary table	Object type	Access	Comments
Discrete input	Single bit	Read-only	This type data can be provided by an I/O system.
Coils	Single bit	Read-write	This type of data can be altered by an application program.
Input registers	16-bit word	Read-only	This type of data can be provided by an I/O system.
Holding registers	16-bit word	Read-write	This type of data can be altered by an application program.

The actual mapping of particular data items to holding registers in Easergy P3 devices can be checked with the setting tool by selecting the items under the **Modbus main configuration** setting view.

Figure 1 - Modbus main configuration setting view



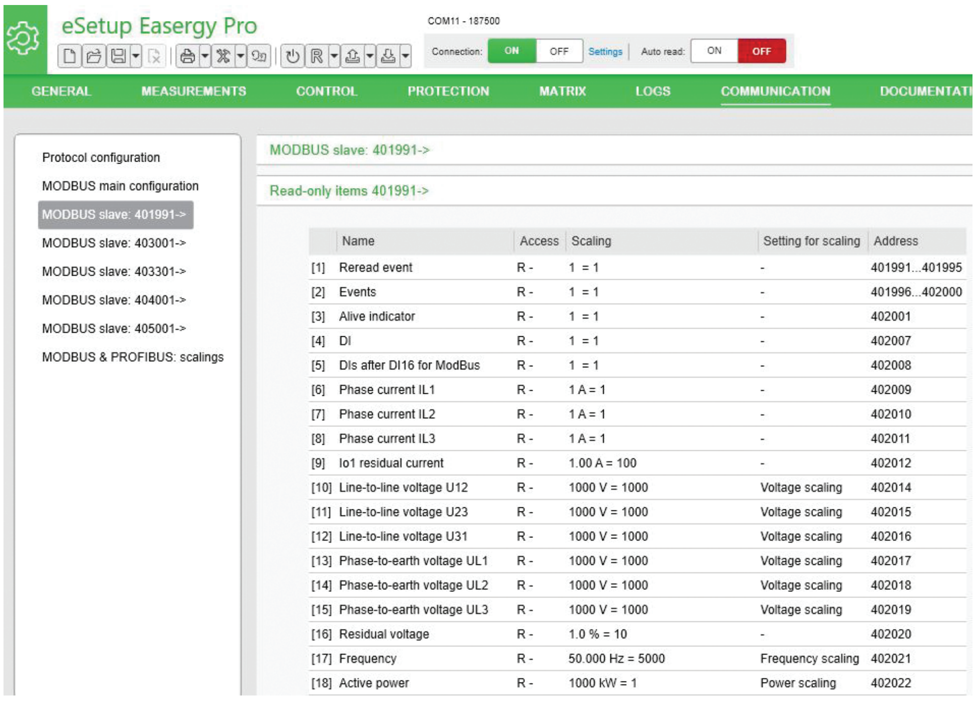
Slave number: 1
ModBus bit rate: 9600
Parity: Even

NOTE:

- The data mappings are visible only when the Modbus protocol is activated. For instructions on how to activate the protocol, see *Chapter 3. Modbus configuration*.
- The values in the **Address** column (for example 40xxxx) consist of two parts: “40” and “xxxx”, where the prefix “40” denotes a reference to *Read/ Write Output or Holding Registers* and “xxxx” denotes the address of a particular register.
- The holding register address sent via Modbus is one less than that indicated by a device. For example, if the Alive Indicator, holding register address 2001, is sent over a Modbus data link, the frame indicates that the address of the holding register is 2000.

The bit rate of Modbus RTU is typically 9600 bps and for the transmission of frames, a parity check mode must be selected (even, odd or none parity checking).

Figure 2 - Modbus data mappings



MODBUS slave: 401991->

Read-only items 401991->

	Name	Access	Scaling	Setting for scaling	Address
[1]	Reread event	R -	1 = 1	-	401991...401995
[2]	Events	R -	1 = 1	-	401996...402000
[3]	Alive indicator	R -	1 = 1	-	402001
[4]	DI	R -	1 = 1	-	402007
[5]	DIs after DI16 for ModBus	R -	1 = 1	-	402008
[6]	Phase current IL1	R -	1 A = 1	-	402009
[7]	Phase current IL2	R -	1 A = 1	-	402010
[8]	Phase current IL3	R -	1 A = 1	-	402011
[9]	Io1 residual current	R -	1.00 A = 100	-	402012
[10]	Line-to-line voltage U12	R -	1000 V = 1000	Voltage scaling	402014
[11]	Line-to-line voltage U23	R -	1000 V = 1000	Voltage scaling	402015
[12]	Line-to-line voltage U31	R -	1000 V = 1000	Voltage scaling	402016
[13]	Phase-to-earth voltage UL1	R -	1000 V = 1000	Voltage scaling	402017
[14]	Phase-to-earth voltage UL2	R -	1000 V = 1000	Voltage scaling	402018
[15]	Phase-to-earth voltage UL3	R -	1000 V = 1000	Voltage scaling	402019
[16]	Residual voltage	R -	1.0 % = 10	-	402020
[17]	Frequency	R -	50.000 Hz = 5000	Frequency scaling	402021
[18]	Active power	R -	1000 kW = 1	Power scaling	402022

Related tasks

Activating Modbus RTU on page 8

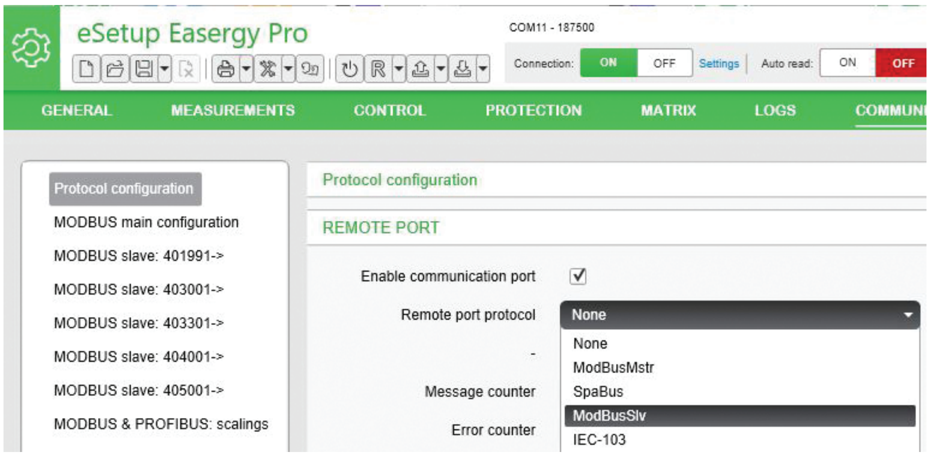
Configuring and activating Modbus TCP on page 9

3. Modbus configuration

3.1. Activating Modbus RTU

1. In Easergy Pro, go to **Communication > Protocol configuration**.
2. Under **Remote port**, select “ModBusSlv” from the **Remote port protocol** drop-down menu.

   Figure 3 - Selecting the remote port protocol



- After changing the communication protocol, for the changes to take effect, reboot the device.
- The protocol can also be enabled via the local panel of the device. You must enter the password for the configurator access level via the local panel before this is possible. See the manual for the device at hand for more instructions.

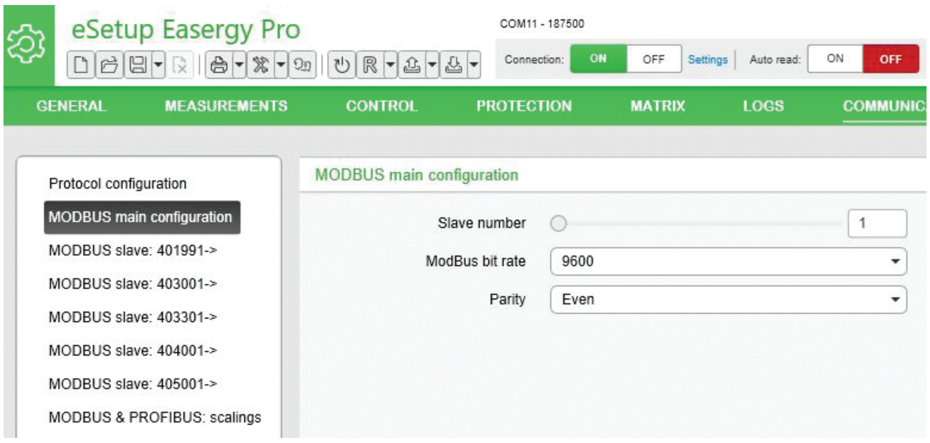
3.2. Configuring Modbus RTU

Before configuring Modbus, you must activate it.

Configure the Modbus protocol using Easergy Pro.

1. Go to **Protocol configuration > Modbus main configuration**.
2. Set the Modbus slave numer, bit rate and parity.

Figure 4 - Modbus main configuration setting view



MODBUS main configuration

Slave number	1
ModBus bit rate	9600
Parity	Even

NOTE: The parity and bit rate must be set to the same value on all devices connected to the same data link.

Related tasks

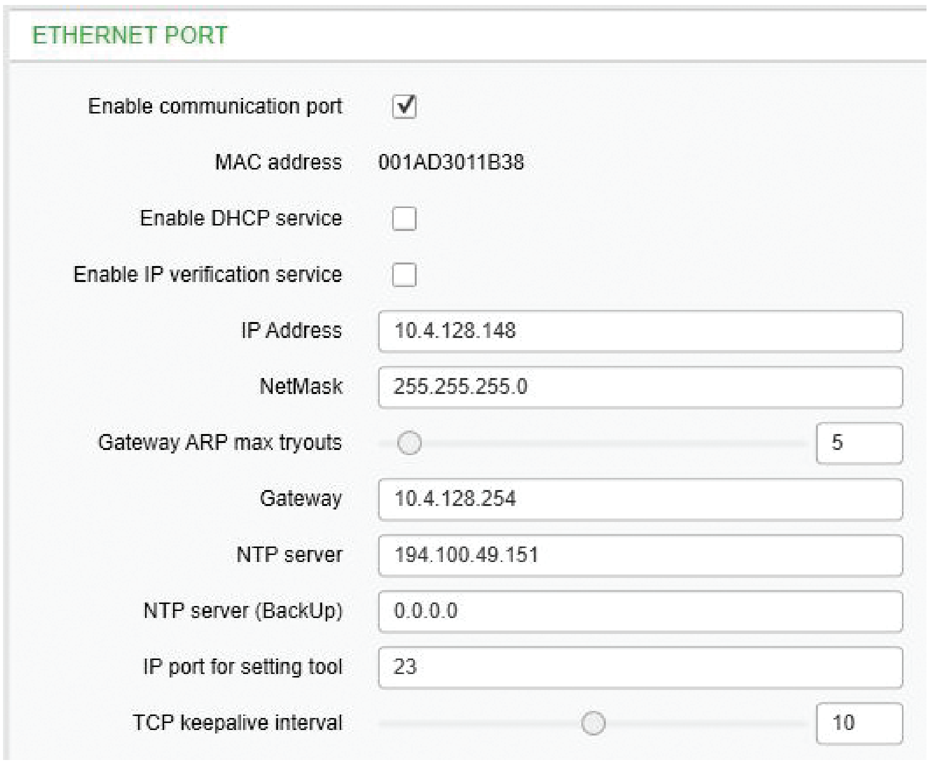
Activating Modbus RTU on page 8

3.3. Configuring and activating Modbus TCP

Configure the Ethernet port settings with Easergy Pro.

1. Go to **Protocol configuration > Ethernet port**.
2. Set the IP address, subnet mask (NetMask) and gateway.

Figure 5 - Ethernet port setting view



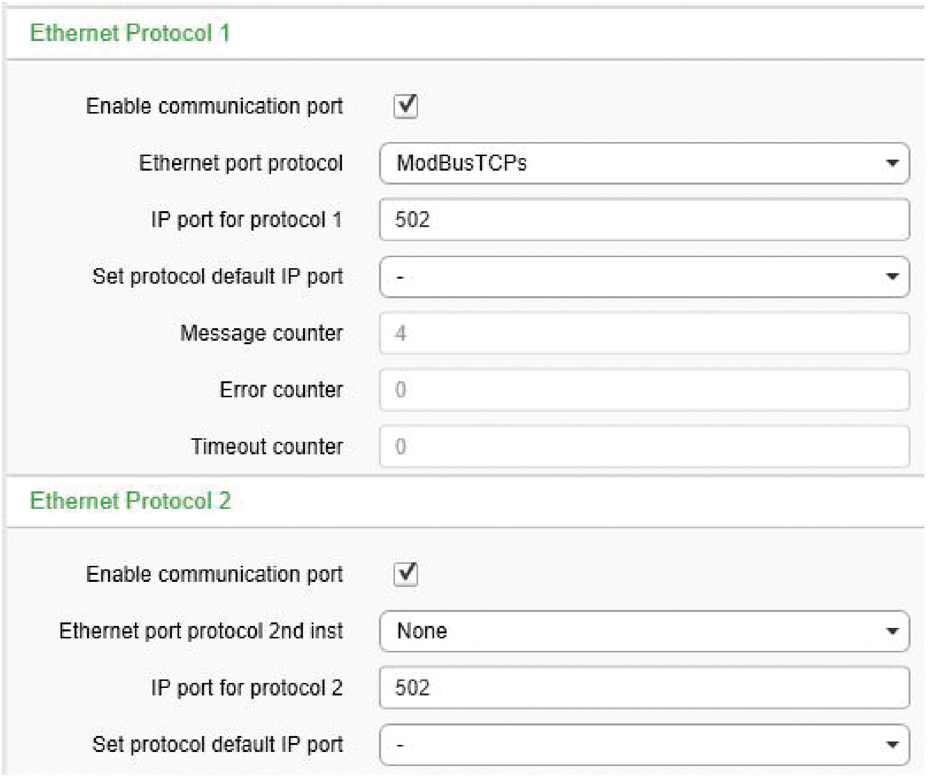
ETHERNET PORT

Enable communication port	✓
MAC address	001AD3011B38
Enable DHCP service	☐
Enable IP verification service	☐
IP Address	10.4.128.148
NetMask	255.255.255.0
Gateway ARP max tryouts	5
Gateway	10.4.128.254
NTP server	194.100.49.151
NTP server (BackUp)	0.0.0.0
IP port for setting tool	23
TCP keepalive interval	10

NOTE: Easergy P3 devices support dynamic host configuration protocol (DHCP) but usually a static IP address is used. Consider this before connecting a device to an existing network, so that no conflicts emerge.

3. Reboot the device for the changes to take effect.
4. To activate Modbus TPC, set **Ethernet port protocol** to “ModbusTCPs” (Modbus TCP, slave).

Figure 6 - Ethernet protocol setting view



Ethernet Protocol 1

Enable communication port	✓
Ethernet port protocol	ModBusTCPs
IP port for protocol 1	502
Set protocol default IP port	-
Message counter	4
Error counter	0
Timeout counter	0

Ethernet Protocol 2

Enable communication port	✓
Ethernet port protocol 2nd inst	None
IP port for protocol 2	502
Set protocol default IP port	-

NOTE:

- There are two different Ethernet protocols, Ethernet protocol 1 and 2. Modbus TCP can be selected to only one of them. The other protocol can be for example IEC 61850.
- Two Modbus TCP masters can communicate with the device simultaneously.
- Serial Modbus RTU can be used at the same time with Modbus TCP masters.

3.4. Clock synchronization

The device’s internal clock can be synchronised via the Modbus protocol. However, this is not a native feature of the Modbus protocol, but an Easergy-specific system. The accuracy of the clock synchronisation is in the scale of a few hundred milliseconds.

The clock can be synchronized either completely (all fields: seconds, minutes, hours, days, month and year) or by synchronizing only the minutes, which in turn sets the seconds and milliseconds to zero.

An example of how minute synchronisation can be done: when the reference clock (the clock which is assumed to be correct) is exactly seven minutes past (any hour), a minute synchronisation is performed. The result is that the internal clock of the device is set to HH:07:00.000 ("Hours:Minutes:Seconds.Milliseconds") "HH" is not changed.

These two ways of synchronizing the clock are denoted "Set RTC", where "RTC" stands for real-time clock and "Synchronise Minutes" in the data map. The holding register address of the minute synchronisation is 2502.

Table 3 - **Description of holding registers allocated to "Set RTC" synchronization**

Holding register	Content
2504	Lower byte: seconds. Milliseconds are zero.
2505	Upper byte: minutes. Lower byte: hours.
2506	Upper byte: day. Lower byte: month.
2507	Year

Time synchronization is recommended to be performed hourly.

For Modbus TCP, clock synchronisation can also be done by using simple network time protocol (SNTP). This requires a NTP server, the address of which is set in the **Protocol configuration > Ethernet port** setting view in Easergy Pro.

3.5. Events

The device's event buffer can be read via the Modbus RTU protocol by reading one event at a time, from holding registers 2101…2105. The event registers contain the latest event, and are cleared when they are read. The registers are then updated to contain the following event from the event buffer.

Table 4 - **Description of Events in holding registers**

Holding register	Content
2101	Event code
2102	Event timestamp Bits 15-6 = milliseconds Bits 5-0 = seconds
2103	Event timestamp Upper byte = minute Lower byte = hour

Holding register	Content
2104	Event timestamp Upper byte = day Lower byte = month
2105	Event timestamp, year

If an error occurs when reading an event from registers 2101…2105, the previously read event is available in registers 2490…2494 for re-reading.

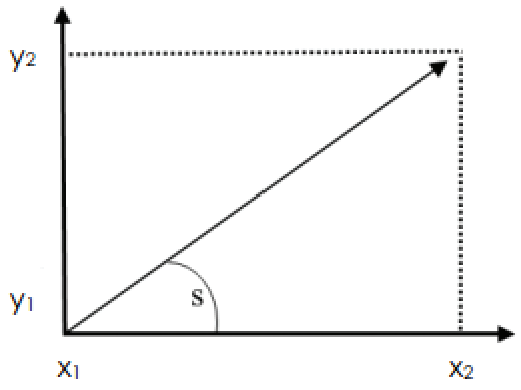
NOTE: Events can also be read from holding registers 1996…2000.

Events are coded with a numbering starting from 0. To obtain the meaning of these event codes, run the GETSET-command “get eventcodes” in the terminal software (for example PuTTY).

3.6. Scaling

Holding registers are 16 bits in size, so they can directly represent 2^{16} = 65535 different values, which might not be enough to describe the values of some physical quantity such as voltage or power. Thus, values transmitted over a Modbus data link must be scaled to account for this.

Figure 7 - Line defined by two points in a two-dimensional space



The scaling is determined by the equation of the line connecting two points in a two-dimensional space, (x_1, y_1) and (x_2, y_2). That is, the trajectory of the line and the offset from origin. Note that there is no offset in the example if the point (x1, y1) is assumed to lie at the origin.

It is common to use scaling factors with base ten (10, 100, 1000…) since, in such cases, the original measurements only lose decimals and such values are easy to read and re-scale to actual values on the client side after transmission. Note that the scaled measurements get rounded.

Different settings for scaling can be used for the power, power factor, tan phi, voltage and frequency scaling. These can be set in the **Communication > Modbus & Profibus scalings** setting view item in Easergy Pro.

A short example: The frequency is internally (in the device) stored as a integer value which also holds three decimal places, that is, 50.000 Hz is represented as

50000. This is a value too large to be represented with 16 bits (signed integer), however, frequency is by default scaled with the points: (x_1, y_1) = (0, 0) and (x_2, y_2) = (10, 1), enabling it to be sent over Modbus.

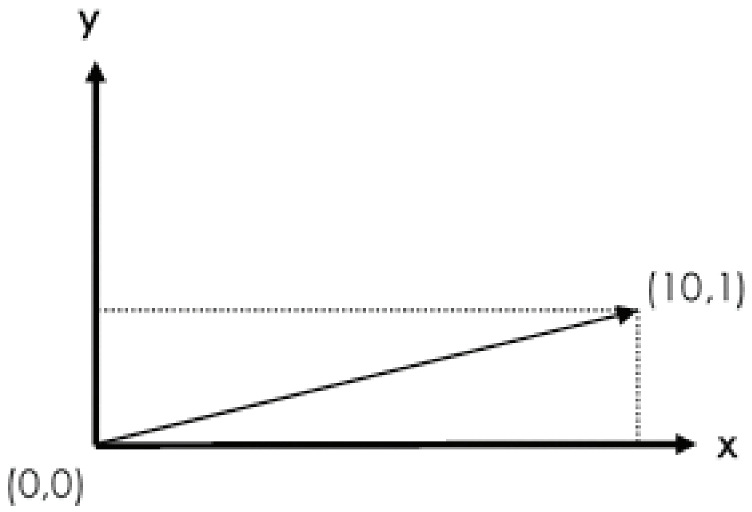
The slope is 0.1 and there is no offset.

$$k = \frac{y_2 - y_1}{x_2 - x_1} = \frac{1 - 0}{10 - 0} = 0.1$$

Thus, the value on the receiving side (the Modbus value) is:

$$value_{modbus} = k * value_{internal} = 0.1 * 50000 = 5000$$

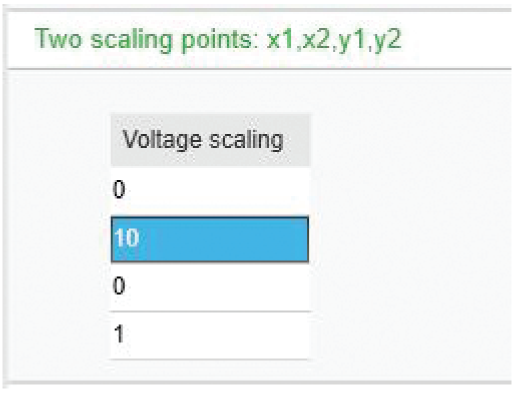
Figure 8 - Default frequency scaling



To make sure that a scaling has been set as intended, you can check it in Easergy Pro by viewing the **Scaling** column for each holding register in the Modbus data mapping lists, for example in **Communication > Modbus slave 401491**.

For an example of scaling change, see *Figure 9 - Voltage scaling changed from default (x2 changed to 10 from 1) on page 14*, *Figure 10 - Result of the change in voltage scaling on page 14* and *Figure 11 - Default scaling for some holding registers on page 14*.

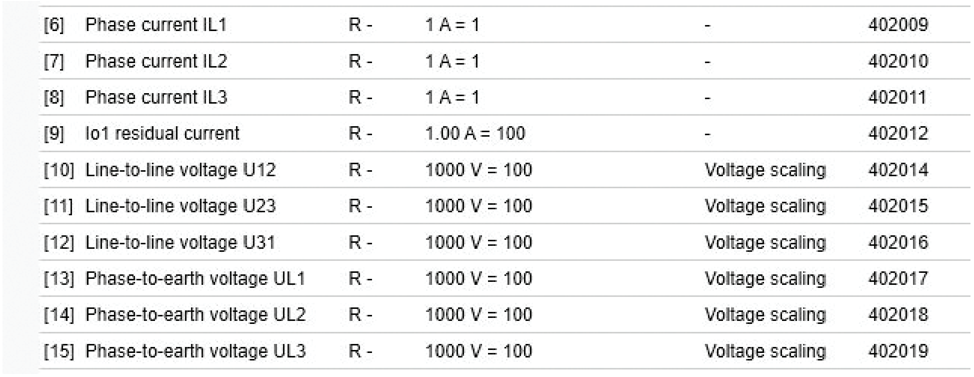
Figure 9 - Voltage scaling changed from default (x_2 changed to 10 from 1)



Two scaling points: x1,x2,y1,y2

Voltage scaling
0
10
0
1

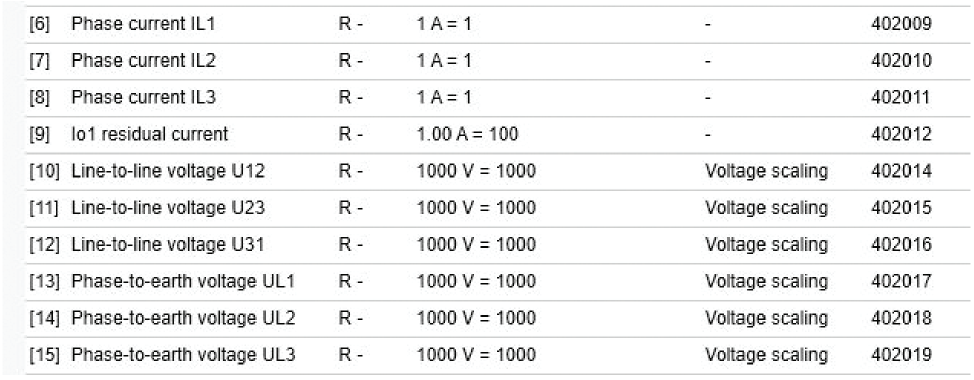
Figure 10 - Result of the change in voltage scaling



[6]	Phase current IL1	R -	1 A = 1	-	402009
[7]	Phase current IL2	R -	1 A = 1	-	402010
[8]	Phase current IL3	R -	1 A = 1	-	402011
[9]	Io1 residual current	R -	1.00 A = 100	-	402012
[10]	Line-to-line voltage U12	R -	1000 V = 100	Voltage scaling	402014
[11]	Line-to-line voltage U23	R -	1000 V = 100	Voltage scaling	402015
[12]	Line-to-line voltage U31	R -	1000 V = 100	Voltage scaling	402016
[13]	Phase-to-earth voltage UL1	R -	1000 V = 100	Voltage scaling	402017
[14]	Phase-to-earth voltage UL2	R -	1000 V = 100	Voltage scaling	402018
[15]	Phase-to-earth voltage UL3	R -	1000 V = 100	Voltage scaling	402019

Compare the scalings in *Figure 10 - Result of the change in voltage scaling on page 14* and *Figure 11 - Default scaling for some holding registers on page 14*.

Figure 11 - Default scaling for some holding registers



[6]	Phase current IL1	R -	1 A = 1	-	402009
[7]	Phase current IL2	R -	1 A = 1	-	402010
[8]	Phase current IL3	R -	1 A = 1	-	402011
[9]	Io1 residual current	R -	1.00 A = 100	-	402012
[10]	Line-to-line voltage U12	R -	1000 V = 1000	Voltage scaling	402014
[11]	Line-to-line voltage U23	R -	1000 V = 1000	Voltage scaling	402015
[12]	Line-to-line voltage U31	R -	1000 V = 1000	Voltage scaling	402016
[13]	Phase-to-earth voltage UL1	R -	1000 V = 1000	Voltage scaling	402017
[14]	Phase-to-earth voltage UL2	R -	1000 V = 1000	Voltage scaling	402018
[15]	Phase-to-earth voltage UL3	R -	1000 V = 1000	Voltage scaling	402019

NOTE: To avoid overflow, scale values so that they are kept in the interval 0–32768 .

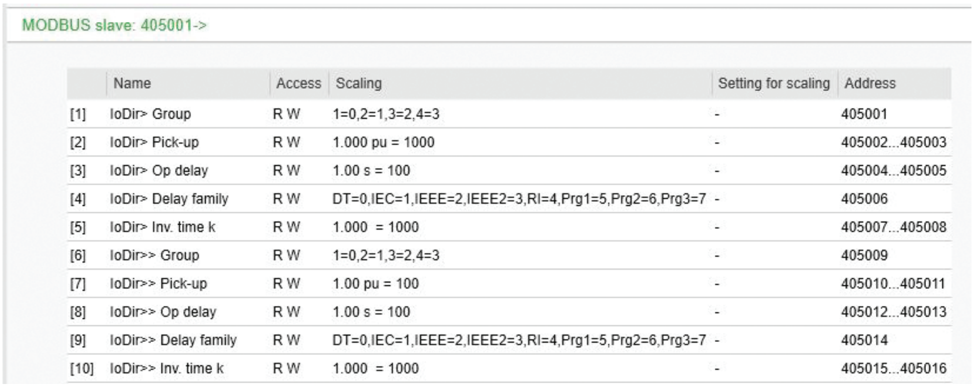
4. Additional information about some Modbus registers

4.1. Protection settings

4.1.1. Modbus registers for protection settings

The most important protection settings can be read and written using Modbus registers starting from address 5001.

Figure 12 - Modbus register 5001



MODBUS slave: 405001->

	Name	Access	Scaling	Setting for scaling	Address
[1]	IoDir> Group	R W	1=0,2=1,3=2,4=3	-	405001
[2]	IoDir> Pick-up	R W	1.000 pu = 1000	-	405002...405003
[3]	IoDir> Op delay	R W	1.00 s = 100	-	405004...405005
[4]	IoDir> Delay family	R W	DT=0,IEC=1,IEEE=2,IEEE2=3,RI=4,Prg1=5,Prg2=6,Prg3=7	-	405006
[5]	IoDir> Inv. time k	R W	1.000 = 1000	-	405007...405008
[6]	IoDir>> Group	R W	1=0,2=1,3=2,4=3	-	405009
[7]	IoDir>> Pick-up	R W	1.00 pu = 100	-	405010...405011
[8]	IoDir>> Op delay	R W	1.00 s = 100	-	405012...405013
[9]	IoDir>> Delay family	R W	DT=0,IEC=1,IEEE=2,IEEE2=3,RI=4,Prg1=5,Prg2=6,Prg3=7	-	405014
[10]	IoDir>> Inv. time k	R W	1.000 = 1000	-	405015...405016

4.1.2. Reading and writing protection settings

Modbus holding registers are 16 bits wide. However, some of the parameters which have been mapped to these registers in the device have 32-bit values. These parameters are mapped to two consecutive 16-bit holding registers, so that the register with the smaller address represents the LSB 16-bit word for the parameter and the second register represents the MSB 16-bit word for the parameter.

In the current firmware version, all parameters (and only the parameters) which occupy two consecutive holding registers have 32-bit values.

When the value of such parameter is updated through Modbus, the client must perform two write operations: one for the LSB word and another for the MSB word. Each write operation updates the value of the parameter, so that only the word being written to is modified. The intermediate state for the parameter—where only one word has been updated—may represent an invalid value for the parameter. For example, the value may be out of allowed range. In such case the first write operation gives an error message, indicating that the value of the parameter was not updated. The client can nevertheless continue with the second write operation. This second write operation modifies the incomplete intermediate state of the parameter. If the intermediate state then reaches a valid 32-bit value for the parameter, the second write operation should indicate a success.

The client can, for example, first write the LSB 16-bit word for the parameter and then write the MSB 16-bit word for the parameter, and the parameter should have the new 32-bit value after the second write operation. The two 16-bit words can be written in any order. These two write operations should occur within two seconds of each other.

4.2. Disturbance recordings

4.2.1. COMTRADE files

COMTRADE is a standard file format for exchanging disturbance recordings. These are (oscillographic) measurements of power system variables, such as (instantaneous) currents and voltages, over an event like a transient fault.

One recording consists of two files: a CFG file describes what was recorded in the recording, and a DAT file contains the recorded values.

4.2.2. Modbus registers for disturbance recordings

In P3 devices, one register is used to issue commands to the COMTRADE reading mechanism.

Table 5 - Modbus registers for disturbance recordings

Register	Content
5400 (Command register)	• MSB byte: Number of the recording to be downloaded, so that value zero means the oldest record, value one means the second oldest record, and so on. • LSB byte: A set of flags ◦ b5: Trigger to remove oldest record. ◦ b4: Trigger to request the next chunk. ◦ b3: Flag for auto-increment the chunk. ◦ b2: Flag for load DAT file. ◦ b1: Flag for load CFG file. • (LSB) b0: Trigger start of download when value 1 is written to this bit.
5401 (Status register 1)	• MSB byte: Number of recordings available. • LSB byte: A set of flags. ◦ b5: Flag for error in download. ◦ b4: Flag for last chunk of data for this file. ◦ b3: Flag for chunk auto-increment in use. ◦ b2: Flag for DAT file now being loaded. ◦ b1: Flag for CFG file now being loaded. • (LSB) b0: Flag for valid data in data registers. Implies that a download is in progress.

Register	Content
5402 (Status register 2)	• MSB byte: Number of bytes of data available in data registers. • LSB byte: Approximate progress downloading current file in percentage.
5403 to 5498 (96 Data registers)	Content (fragment) for the file being downloaded.

NOTE: The maximum length of a Modbus message is 253 bytes. It is thus possible to read the two status registers and all of the data registers in a single command.

4.2.3. Reading disturbance records as COMTRADE files

Disturbance recordings (COMTRADE files) can be read via registers starting from 5400.

1. Open Modbus connection to the device.
2. Read status register 1 (5401), and check that there is at least one recording available.
3. Construct a value for the command which starts the download process.

   For example, value 271 (0x0x10F) starts download of the second oldest record.

   MSB byte: 0x01

   Number of the recording to be downloaded

   LSB byte: 0x0F

   - b5: (0) Trigger to remove the oldest record.
   - b4: (0) Trigger to request the next chunk.
   - b3: (1) Flag to auto-increment the chunk.
   - b2: (1) Flag to load the DAT file.
   - b1: (1) Flag to load the CFG file.
   - (LSB) b0: (1) Trigger start of download when value 1 is written to this bit.

   Write this value to the command register (5400). This command indicates which recording you want to read and whether you want to download both CFG file and DAT file, or only either one at a time. This enables “chunk auto-incrementing”, which means that whenever we read the value of status register 1, the contents of the data registers are updated to contain the next fragments of data for the file which is currently downloading.
4. Make a read request that starts from status register 1 (5401) and continues until the end of the data registers (registers 5403...5498).

   This is a total of 96 + 2 = 98 registers.
5. Inspect the value of status register 1.

Table 6 - Register 5401 (status register 1)

Byte / bit	Meaning	Explanation
MSB byte	Number of recordings available	If this value changes in the middle of a download, a new recording has been made. It is recommended that in this case the download is aborted and restarted, by returning to step 3 (or to step 1).
LSB byte	Set of flags	
b5	Flag for error in download.	If this flag is set, there was an internal error in the device, and the download is automatically aborted by the device. Contents of all other registers are invalid.
b4	Flag for last chunk of data for this file	If this flag is set, the data registers contain the last pieces of data for the file which we are currently downloading. When next chunk is triggered, we get the first pieces of data for another file. (With auto-increment, this triggering happens when the status register 1 is read.)
b3	Flag for chunk auto-increment in use	Indicates whether auto-increment is in use for this download. If this is not used, a separate write is needed to the command register, with correct flags, to get the next pieces of data in the data registers.
b2	Flag for DAT file now being loaded	If set, indicates that we are now downloading the DAT file.
b1	Flag for CFG file now being loaded	If set, indicates that we are now downloading the CFG file.
(LSB) b0	Flag for valid data in data registers	If this flag is set, the data registers DO contain valid data for the files being downloaded. Absence of this flag indicates either an error or that the download is complete.

6. Assuming that you now have valid data for the download, inspect the value of status register 2 (4602).

MSB byte	Number of bytes of data available in data registers
LSB byte	Approximate progress downloading current file in percentage

NOTE: The data registers hold 16-bit values (two bytes), while the amount of data is reported in bytes. The data registers should be filled with data (giving value 192 here), unless the last pieces of content for the file are now in the data registers.

You can always read all data registers, and then ignore those which do not contain valid data, or first read the two status registers, and then, with a separate read command, read only those data registers which actually contain data. Also, you may choose to read only at most M registers at a time, so that M is smaller than the total number of data registers (96). These options may be useful if the used communication link is slow or expensive.

If the link is prone to transfer errors, you can choose to read a given data register several times, and use the value which occurs most commonly. In this use case, it is recommended not to use the auto-increment feature.

7. Copy data from the data registers to the file which is currently being downloaded.

   Note the actual amount of (valid) data in the registers that was discovered in step 5. If this is the last chunk of data for this file, then after the data has been written to the file, close the file. When the next pieces of data are received, you can open a new file. Flags on status register 1 indicate which file is then being downloaded.

8. Start to read the next chunk by returning to step 4.

4.3. Modbus fault current registers

Here you can find explanations for some additional Modbus registers.

Table 7 - Modbus fault current registers

Register	Explanation
2110	Last fault current from any dir/undir OC stage (can be cleared)
2111	Fault current from `I>` stage
2112	Fault current from `I>>` stage
2113	Fault current from `I>>>` stage
2130	Last fault Io current from any undirectional EF stage
2131	Fault current from `Io>` stage
2132	Fault current from `Io>>` stage
2133	Fault current from `Io>>>` stage
2134	Fault current from `Io>>>>` stage

5. Testing Modbus with Simple Tester

5.1. Simple Tester

Simple Tester is a basic protocol testing tool (for PC) that supports several different protocols, including Modbus RTU and TCP. Simple Tester can be used to test reading data from holding registers, writing to holding registers, viewing events and performing clock synchronization from the PC.

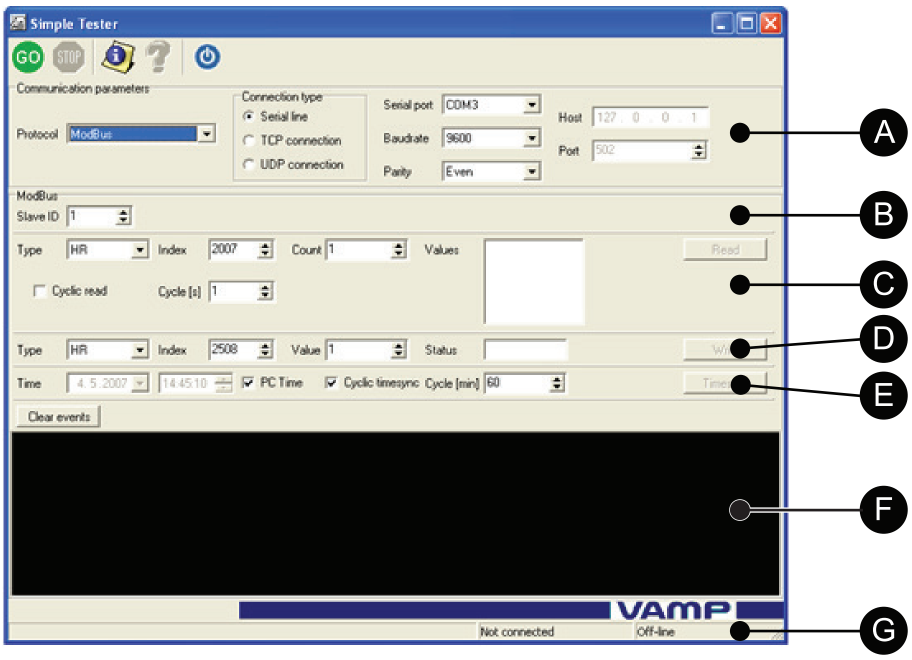


Figure 13 - Simple Tester user interface with Modbus selected as protocol

Simple Tester

Communication parameters
Protocol: ModBus
Connection type: [✓ Serial line] TCP connection UDP connection
Serial port: COM3
Baudrate: 9600
Parity: Even
Host: 127 . 0 . 0 . 1
Port: 502

ModBus
Slave ID: 1
Type: HR Index: 2007 Count: 1 Values
Cyclic read: ☐ Cycle [s]: 1
Type: HR Index: 2508 Value: 1 Status
Time: 4. 5. 2007 14:45:10 PC Time: ☑ Cyclic timesync: ☑ Cycle [min]: 60

Not connected
Off-line

A.	Communication parameters: protocol selection and device connection configuration
B.	Modbus Slave ID selector
C.	Selections for data reading
D.	Selections for data writing
E.	Time controls
F.	Event view window
G.	Connection status

5.2. Connecting the device to the PC running Simple Tester

Before starting testing, connect the P3 device to the PC running Simple Tester.

You can create the connection either via RS-232 or Ethernet.

- To connect the device to the PC via RS-232 interface:

  a. Connect an RS-232 cable to the device.

  b. Connect an USB-to-RS-232 cable to the PC’s USB port.

c. Connect a cable adapter between the two cables.

d. The cable adapter connects the pins 2,3, and 7 of the device's RS-232 cable's D-connector to pins 2,3, and 5 of the D-connector of the PC's cable.

- To connect the device to the PC via Ethernet interface:
  – Connect an Ethernet cable or Ethernet switch between the PC and the device.

5.3. Connecting the device to Simple Tester with Modbus RTU

Before connecting a device with Modbus RTU:

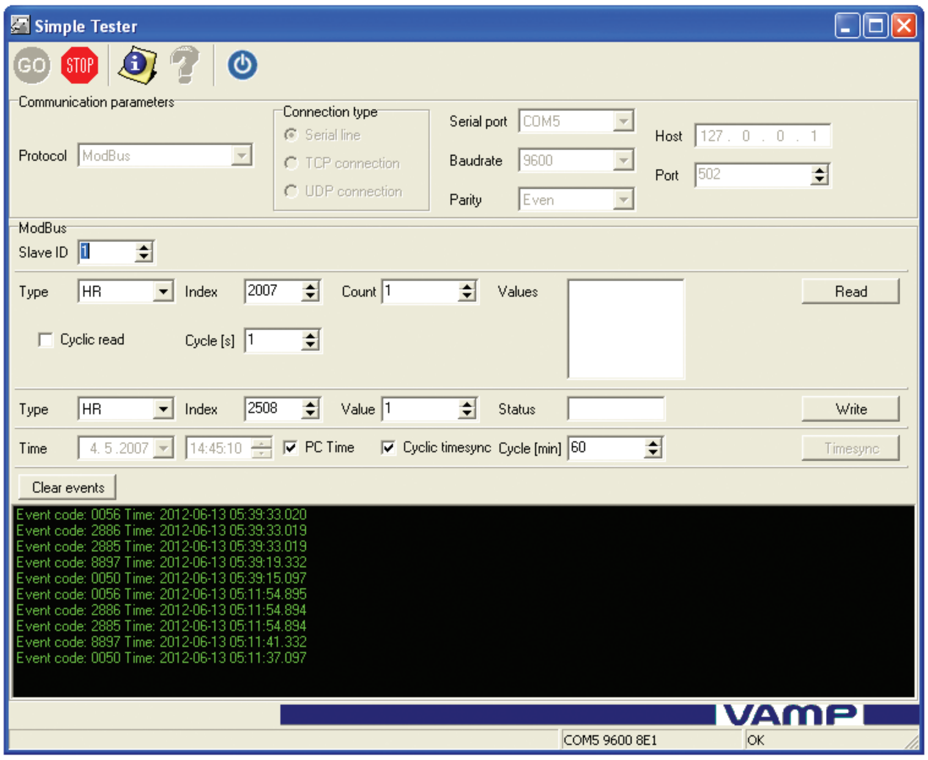
- Configure the device.
- Connect the device to the PC running Simple Tester.

1. Open Simple Tester.
2. Under **Communication parameters**, select ModBus from the **Protocol** drop-down menu.
3. Under **Communication parameters** and **Modbus**, select the settings as follows:

Parameter	Value
Protocol	Modbus
Connection type	Serial line
Serial port	The serial port of the PC
Modbus slave ID	The slave ID configured on the device

4. Click the **Go** button.

Figure 14 - Simple Tester when connected with Modbus RTU



Protocol: ModBus
Serial port: COM5
Baudrate: 9600
Parity: Even
Host: 127 . 0 . 0 . 1
Port: 502
Slave ID: 1
Type: HR Index: 2007 Count: 1 Values
Cyclic read Cycle [s]: 1
Type: HR Index: 2508 Value: 1 Status
Time: 4. 5 .2007 14:45:10 ☑ PC Time ☑ Cyclic timesync Cycle [min]: 60

```
Event code: 0056 Time: 2012-06-13 05:39:33.020
Event code: 2886 Time: 2012-06-13 05:39:33.019
Event code: 2885 Time: 2012-06-13 05:39:33.019
Event code: 8897 Time: 2012-06-13 05:39:19.332
Event code: 0050 Time: 2012-06-13 05:39:15.097
Event code: 0056 Time: 2012-06-13 05:11:54.895
Event code: 2886 Time: 2012-06-13 05:11:54.894
Event code: 2885 Time: 2012-06-13 05:11:54.894
Event code: 8897 Time: 2012-06-13 05:11:41.332
Event code: 0050 Time: 2012-06-13 05:11:37.097
```

COM5 9600 8E1 OK

A successful connection is indicated by an “OK” in the rightmost connection status indicator furthest down in the Simple Tester window.

Related tasks

Configuring Modbus RTU on page 8

Connecting the device to the PC running Simple Tester on page 20

5.4. Connecting the device to Simple Tester with Modbus TCP

Before connecting a device with Modbus TCP:

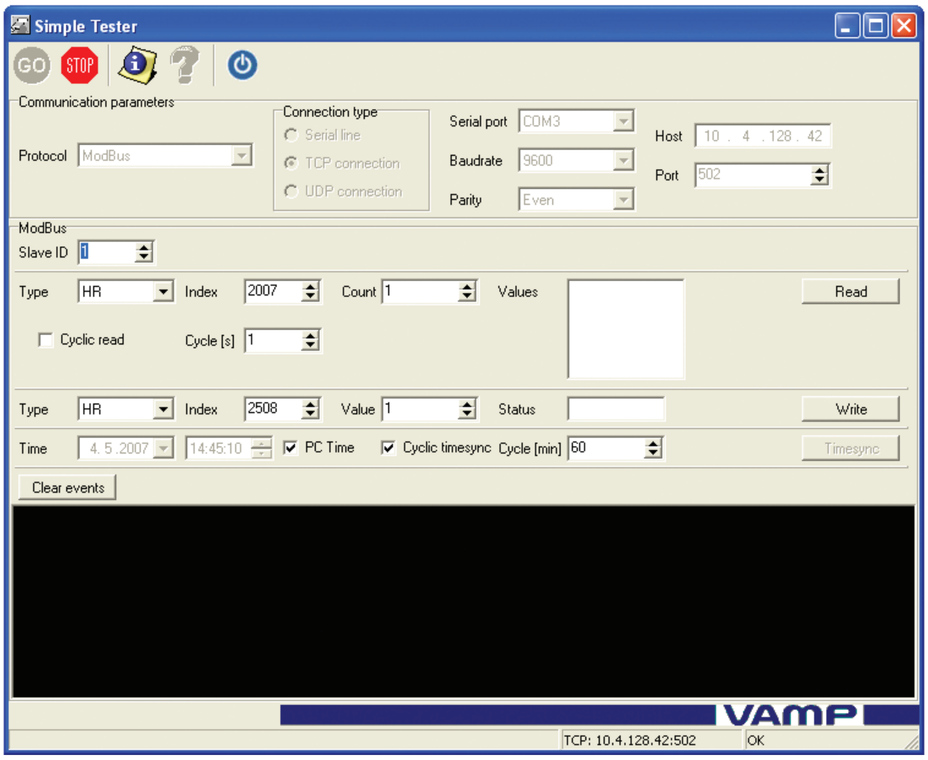
- Configure the device Ethernet port and TCP port instances.
- Establish a physical Ethernet connection from the PC running Simple Tester to the device.

1. Open Simple Tester.
2. Under **Communication parameters**, select **Modbus** from the **Protocol** drop-down menu.
3. Under **Communication parameters** and **Modbus**, select the settings as follows:

Parameter	**Value**
Protocol	Modbus
Connection type	TCP connection
Host	The IP address configured on the device
Port	502
Modbus slave ID	The slave ID configured on the device

4. Click the **Go** button.

Figure 15 - Simple Tester when connected with Modbus TCP



Communication parameters
Protocol: ModBus
Connection type: ( ) Serial line (●) TCP connection ( ) UDP connection
Serial port: COM3
Baudrate: 9600
Parity: Even
Host: 10 . 4 . 128 . 42
Port: 502
ModBus
Slave ID: 1
Type: HR Index: 2007 Count: 1 Values:
☐ Cyclic read Cycle [s]: 1
Type: HR Index: 2508 Value: 1 Status:
Time: 4. 5 .2007 14:45:10 ☑ PC Time ☑ Cyclic timesync Cycle [min]: 60
TCP: 10.4.128.42:502 OK

A successful connection is indicated by an “OK” in the right-most connection status indicator furthest down in the Simple Tester window.

Related tasks

Configuring and activating Modbus TCP on page 9

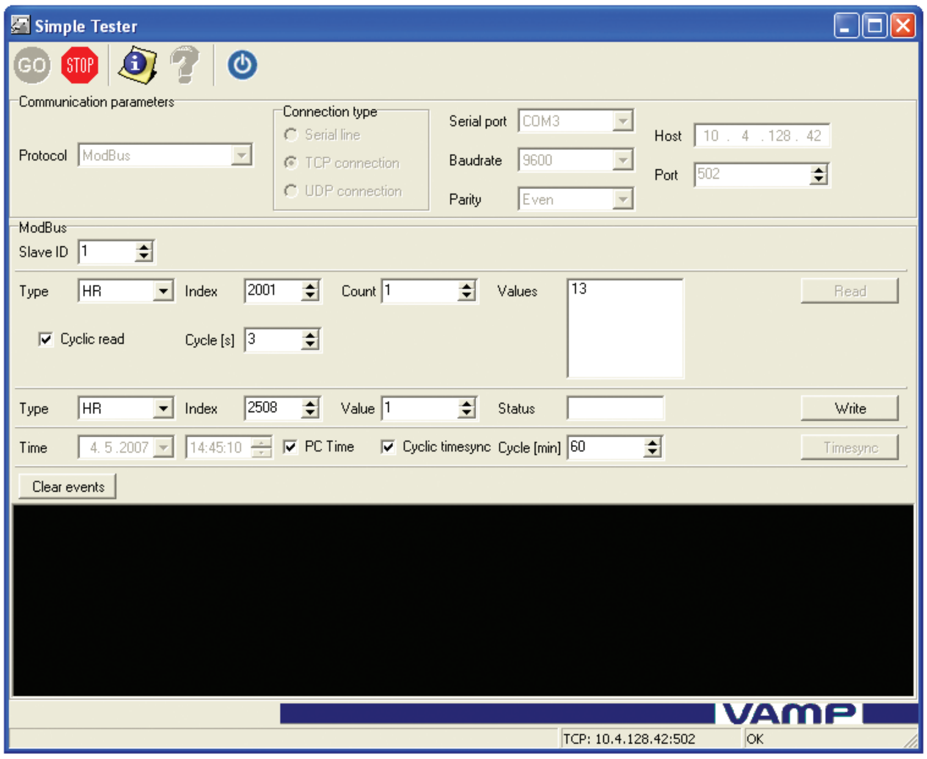
Connecting the device to the PC running Simple Tester on page 20

5.5. Reading data

Before reading data, connect the device to the PC running Simple Tester.

1. Open Simple Tester.
2. Under **Communication parameters**, select ModBus from the **Protocol** drop-down menu.

Figure 16 - Performing a cyclic read on the Alive Counter with Simple Tester



Communication parameters
Protocol: ModBus
Connection type: ( ) Serial line [✓ TCP connection] ( ) UDP connection
Serial port: COM3
Baudrate: 9600
Parity: Even
Host: 10 . 4 . 128 . 42
Port: 502

ModBus
Slave ID: 1
Type: HR Index: 2001 Count: 1 Values: 13
[✓] Cyclic read Cycle [s]: 3
Type: HR Index: 2508 Value: 1 Status:
Time: 4. 5.2007 14:45:10 [✓] PC Time [✓] Cyclic timesync Cycle [min]: 60

TCP: 10.4.128.42:502 OK

3. Set the right values for **Type**, **Index** and **Count**.

Parameter	Description
Type	Type of data: select HR (holding register).
Index	Select which data item to read from the device. See the data mapping provided by Easergy Pro.
Count	Indicates how many consecutive indices are to be read. For example, setting Index to 2001 and Count to 2 displays the values of Index 2001 and Index 2002 in the **Values** box.

4. Click the **Read** button to manually initiate a read.

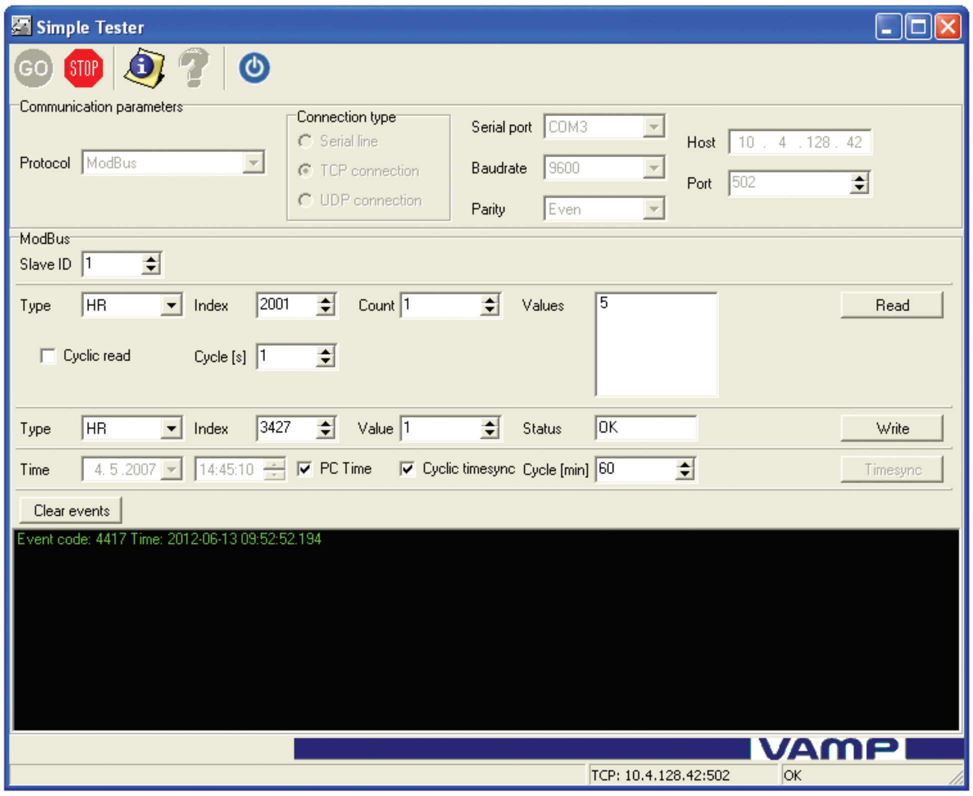
   Reads can also be done cyclically, by selecting the **Cyclic read** checkbox. The length of one cycle can be changed by setting a value (seconds) in the **Cycles** field.

Related tasks

Connecting the device to the PC running Simple Tester on page 20

5.6. Writing data

Figure 17 - Simple Tester after write to Virtual Input 1



Communication parameters
Protocol: ModBus
Connection type: ( ) Serial line [✓ TCP connection] ( ) UDP connection
Serial port: COM3
Baudrate: 9600
Parity: Even
Host: 10 . 4 . 128 . 42
Port: 502

ModBus
Slave ID: 1
Type: HR Index: 2001 Count: 1 Values: 5
☐ Cyclic read Cycle [s]: 1
Type: HR Index: 3427 Value: 1 Status: OK
Time: 4. 5.2007 14:45:10 ☑ PC Time ☑ Cyclic timesync Cycle [min]: 60

```
Event code: 4417 Time: 2012-06-13 09:52:52.194
```

TCP: 10.4.128.42:502 OK

1. Set the right values for **Type**, **Index** and **Value**.
2. Click the **Write** button.

   If the operation is successful, the **Status** field indicates OK.

   NOTE: The event view of Simple Tester now contains an event, corresponding to “Virtual Input 1 ON”.

5.7. Events and clock synchronization in Simple Tester

If there are events in the event buffer when connecting to Simple Tester, they are listed in the event view. New events triggered by writes done with Simple Tester also show up there.

Clock synchronisation can also be performed with Simple Tester by setting the desired behavior in the time controls settings. The two leftmost fields are used for setting the time manually, but the time can also be taken from the PC, by selecting the **PC Time** checkbox.

Clock synchronisation can also be set to be cyclic by selecting the **Cyclic timesync** checkbox. The length of a cycle (minutes) is set in the rightmost field.

Related concepts

Simple Tester on page 20

Schneider Electric
35 rue Joseph Monier
92500 Rueil Malmaison - France
Phone: +33 (0) 1 41 29 70 00
www.schneider-electric.com

P3APS18025EN — 04/2019