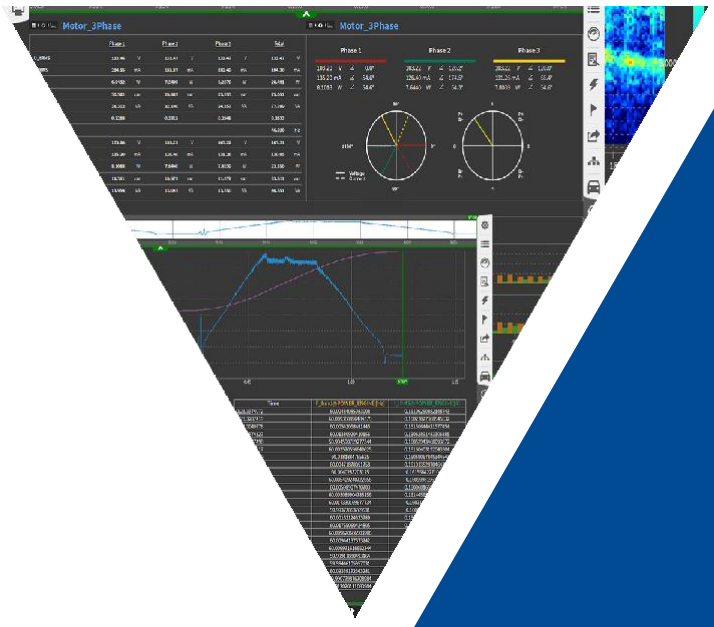
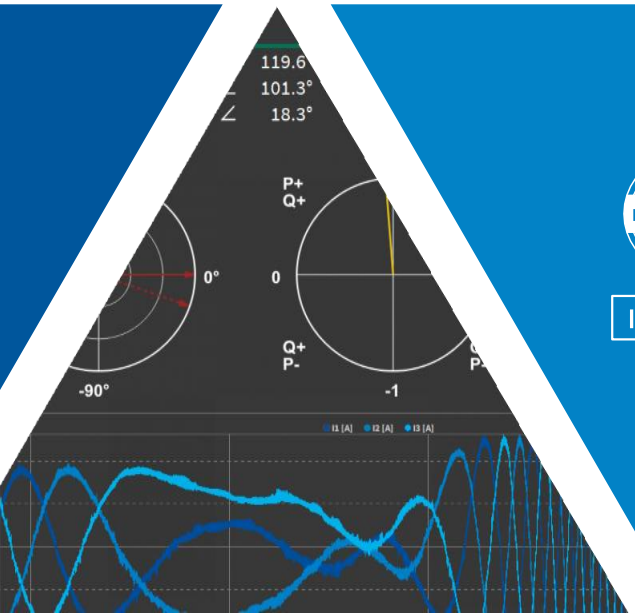
DEWETRON



OXYGEN
Modbus
SOFTWARE MANUAL



EN ISO 14001

ISO 9001

CONTENTS

CHAPTER

ONE

PREFACE

The information contained in this document is subject to change without notice.

DEWETRON GmbH (DEWETRON) shall not be liable for any errors contained in this document. DEWETRON MAKES NO WARRANTIES OF ANY KIND ABOUT THIS DOCUMENT, WHETHER EXPRESS OR IMPLIED. DEWETRON SPECIFICALLY DISCLAIMS THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. DEWETRON shall not be liable for any direct, indirect, special, incidental, or consequential damages, whether based on contract, tort, or any other legal theory, in connection with the furnishing of this document or the use of the information in this document.

Technical Support

Please contact your local authorized DEWETRON representative first for any support and service questions.

For Europe and Asia, please contact:

DEWETRON GmbH
Parkring 4, 8074 Grambach
AUSTRIA
Tel.:+43 316 3070-555
Fax: +43 316 3070-90
Email: support@dewetron.com
Web: http://www.dewetron.com
The telephone hotline is available Monday to Friday between 08:00 and 17:00 CET (GMT +1:00)

For America, please contact:

DEWETRON, Inc.
2850 South County Trail
East Greenwich, RI 02818
USA
Tel.: +1 401 398 7963
Fax: +1 401 284 3755
Email: us.support@dewetron.com
Web: http://www.dewetron.com

The telephone hotline is available Monday to Friday between 08:00 and 17:00 GST (GMT -5:00)

DEWETRON GmbH
Parkring 4
8074 Grambach
AUSTRIA

Please find further information at https://www.dewetron.com.

CHAPTER

TWO

DOCUMENT HISTORY

Table 2.1: Document History

Date	Author	SW Version	Change	Doc. Rev.
15.12.2017			Initial release	1.0
29.05.2018			Changes not tracked	1.1
08.04.2019			Changes not tracked	1.2
26.09.2019			Changes not tracked	1.3
20.12.2019	TK		Updated modbus client polling rate	1.4
09.03.2021	MF	OXYGEN 8.1	Added modbus server	1.5

Authors: T. Klug (TK), M. Fuchs (MF)

CHAPTER

THREE

PREFACE

This documentation describes, how to use the MODBUS Cleint and Server Plugin in OXYGEN.

From Wikipedia [https://en.wikipedia.org/wiki/Modbus]

Modbus is a data communication protocol used for programmable logic controllers (PLCs). Modbus has become a de facto standard communication protocol and is now a commonly available means of connecting industrial electronic devices. Modbus is popular in industrial environments because it is openly published and royalty-free. It was developed for industrial applications, is relatively easy to deploy and maintain compared to other standards, and places few restrictions - other than the datagram (packet) size - on the format of the data to be transmitted. Modbus uses the RS485 or Ethernet as its wiring type. Modbus supports communication to and from multiple devices con-nected to the same cable or Ethernet network. For example, a device that measures temperature and a different device to measure humidity, both of which communicates the measurements to a computer.

CHAPTER

FOUR

MODBUS CLIENT

The MODBUS Client Plugin for OXYGEN is an extension for the popular OXYGEN measurement software to read data from Modbus devices. This allows the user, to use Modbus devices as data sources.

Features:

- Reading Data from Modbus TCP Devices
- Support of various data types (int, float, …)
- Support of different scaling modes (linear and sunspec)
- Independent definition of register and endpoints for simple reuse
- Per endpoint definable refresh rate

Known Limitation:

- No support of RTU

4.1 Concept

- TCP Endpoint: An Endpoint is a Node with a unique IP address and port an can hold one to many devices.
- Device: Is the representation of a physical modbus device. It is described by its UNIT_ID.

4.2 Datatypes and scaling

The Plugin supports two different types of data scaling:

- const_linear: Linear scaling with factor and offset
- sunssf: Variable scaling with another register value, compatible with the sunspec scaling (INT+SF)

4.3 Constant linear scaling

Typically, the linear scaling is used for converting the binary data to physical values. Output Value = read_value * scale + offset

Example:

```
read_value = 2314
scale = 0.1
offset = 0
output_value = 2314 * 0.1 + 0.0 = 231.4
```

4.4 SunSpec scaling

Within Solar inverters, the sunssf scaling is often used. It is defined by Sunspec. In this case, a separate register is read, which holds the actual scaling value. This specific value is an ex-ponent of 10. Output Value = read_value * 10^scale_reg_val

Example:

```
read_value = 2314
scale_reg_val = -1
output_value = 2314 * 10^-1 = 231.4
```

4.5 Activate a MODBUS client

- Start OXYGEN (if not already started)
- Open the channel list
- Press “+” add channel
- Select *MODBUS Receiver* in the Data Input section (see in Fig. 4.1)

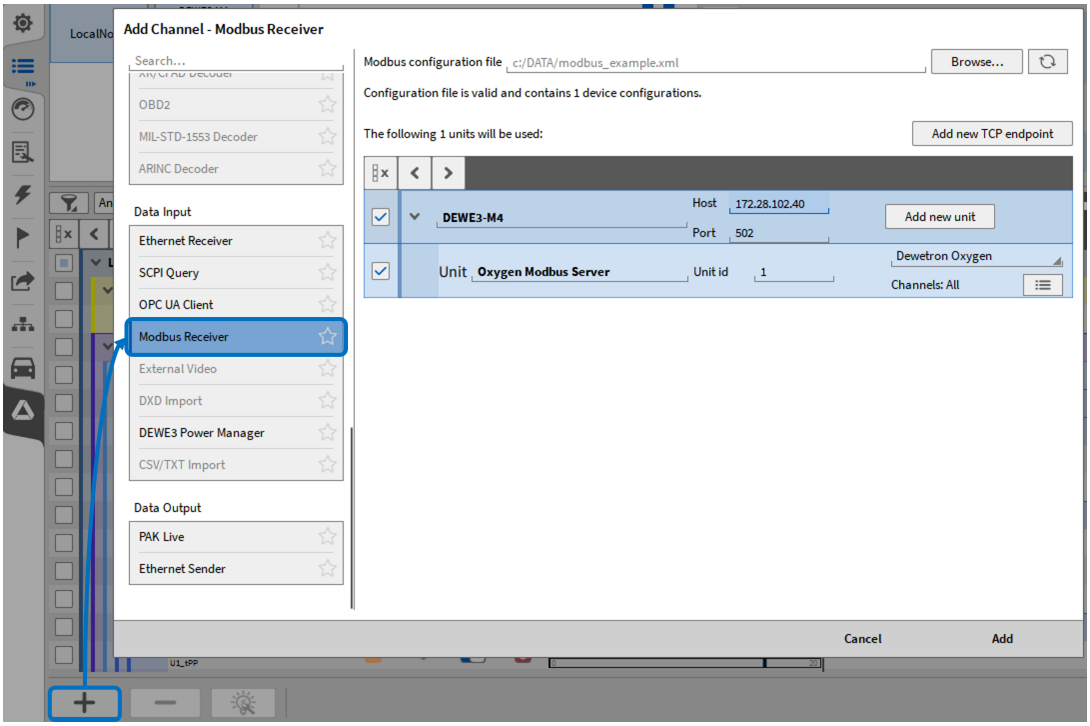


Fig. 4.1: Activate and MODBUS client in OXYGEN

- Load a XML Document with the Device and Network description
- Adapt settings if necessary (e.g. load only selected channels, change IP-address, Unit Id)
- Click “Add”

The following modbus client properties (see in Fig. 4.2) are loaded based on the imported .xml.

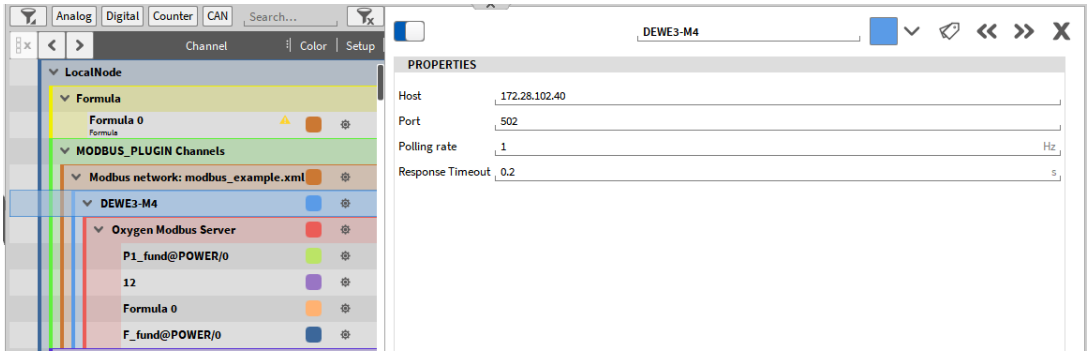


Fig. 4.2: Imported modbus client properties

- Host IP
- Port
- Polling rate - polling rate of the modbus client.
- Response timeout - time before a disconnect warning is triggered. *Modbus: unable to connect to ‘‘IP-address’’*

CHAPTER

FIVE

MODBUS SERVER

5.1 Configuring a MODBUS Server in OXYGEN

OXYGEN (v8.1 and higher) includes an integrated MODBUS Server for standardized data exchange with external MODBUS Clients.

5.1.1 Setup steps

- Navigate to Measurement Settings > Servers (See ① in Fig. 5.1)
- Click + to create a new MODBUS Server instance.
- Configure the MODBUS Server parameters (See ② in Fig. 5.1)
- Select the channels to be published for MODBUS Clients (See ③ in Fig. 5.1).
- Store the set configuration as .xml for MODBUS Clients (See ④ in Fig. 5.1).
- Click the Play button to start the server (See ⑤ in Fig. 5.1).

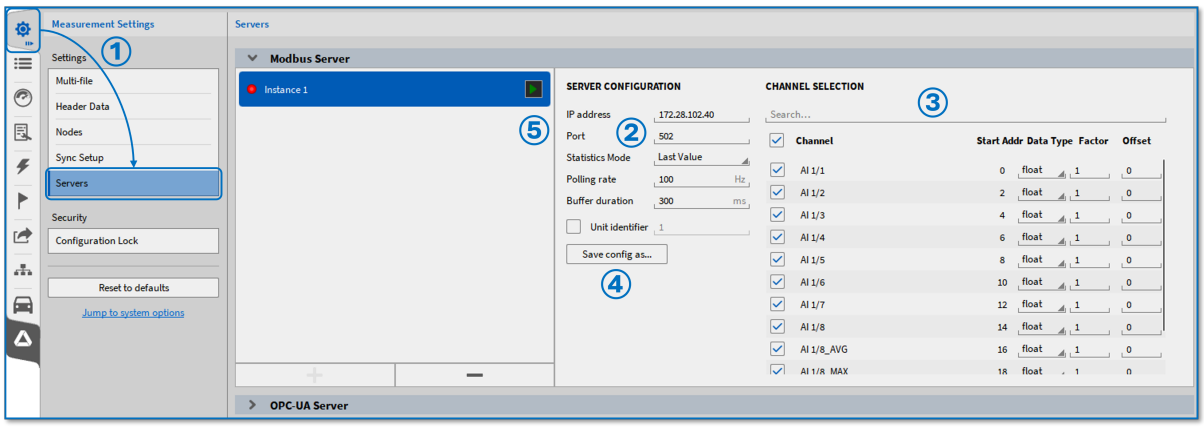


Fig. 5.1: Activate a MODBUS Server in OXYGEN

5.1.2 Configuration Parameters

(See ② in Fig. 5.1)

- **IP address:** Set the communication IP address. Adjust this to meet your IT or firewall requirements.
- **Port:** Set the communication port (default: 502). Adjust this to meet your IT or firewall requirements.
- **Statistic Mode:** Select between *Last Value* (default) or *Average* based on polling rate.
- **Polling Rate:** Define update rate of the MODBUS Server from 1 to 100 Hz (default: 100 Hz).
- **Buffer duration:** Define the duration data is kept before discarding, from 0 to 1000 ms before publishing the data (default: 300 ms).
- **Unit identifier:** Activate/Deactivate a *Unit identifier*. If activated, set a value between 0 - 255. Only request from MODBUS Clients including the selected *Unit identifier* will be accepted by the server.

5.1.3 Channel Selection

(See ③ in Fig. 5.1)

Enable or disable specific measurement channels. Only selected channels are accessible in the MODBUS server's address space. Valid channels include synchronous and asynchronous scalar channels.

- **Channel:** Valid channels from the OXYGEN channel list. These are enabled by checking the corresponding checkbox.
- **Start Addr:** Will be automatically set according to the selected *Data Type* (see below).
- **Data Type:** Select the data type of the respective channel. Following data types are available:
  - float: Address space of 2 will be used
  - double: Address space of 4 will be used
  - int16: Address space of 1 will be used
  - uint16: Address space of 1 will be used
  - int32: Address space of 2 will be used
  - uint32: Address space of 2 will be used
- **Factor:** Select a scaling factor for the respective channel
- **Offset:** Select a scaling offset for the respective channel

CHAPTER

SIX

XML CONFIG FILE

Most of the configuration is provided via a XML document. It is designed, to be very flexible in usage as well as easy to create.

6.1 Working Example

This example consists of one TCP endpoint, one Modbus device with one register.

```
<Modbus version="1.0">
    <TCPEndpoint name = "MyEndpoint" host = "192.168.1.100" port = "502"␣
↪polling_rate = "1Hz">
        <Device name = "MyDevice1" device_type = "MyDeviceType" unit_id =
↪"1"/>
    </TCPEndpoint>
    <ModbusDevice name="MyDeviceType">
        <HoldingRegister address="40000" description="This is a Test Value
↪" name="Value 1" type="float"/>
    </ModbusDevice>
</Modbus>
```

The first part in this Example is the TCPEndpoint.

```
<TCPEndpoint name = "MyEndpoint" host = "192.168.1.100" port = "502" poll-
↪ing_rate = "1Hz">
    <Device name = "MyDevice1" device_type = "MyDeviceType" unit_id = "1"/>
</TCPEndpoint>
```

The TCPEndpoint Node in the XML describes an endpoint with the name “MyEndpoint”. The host can be an ip-address or a hostname, the ip-port is 502 by default. In this case, the polling rate is set to 1Hz, which means, that the register values are read once per second. An Endpoint consists of one or more devices, which are separated by their UNIT_ID. The Endpoint is the dedicated device itself, which is holding the Modbus registers. Typically, the Endpoint is equal to the device. But in some cases, an Endpoint consists of 2 or more devices (e.g. a gateway). The important value here is the device_type. It holds the name of the device, which is defined in the bottom part of the XML (ModbusDevice). This allows the user, to re-use the description of one device in multiple Endpoints.

```
<ModbusDevice name="MyDeviceType">
    <HoldingRegister address="40000" description="This is a test value"␣
↪name="Value 1" type="float"/>
</ModbusDevice>
```

Continuous register numbering is necessary for fast one-block retrieval. If an address is omitted in the register numbering, OXYGEN divides it into individual blocks, and is then not inter-preted as a single block. By filling in the unneeded registers, the query is interpreted by OXYGEN as one block. This can also be done by filling the unneeded registers with dummy values. In the follow-ing example, Oxygen would store 3 individual blocks.

```
<ModbusDevice name="MyDeviceType">
    <HoldingRegister address="0" description=" Value" name="Value 1" type=
↪"float"/>
    <HoldingRegister address="1" description=" Value" name="Value 1" type=
↪"float"/>
    <HoldingRegister address="2" description=" Value" name="Value 1" type=
↪"float"/>

    <HoldingRegister address="5" description=" Value" name="Value 1" type=
↪"float"/>
    <HoldingRegister address="6" description=" Value" name="Value 1" type=
↪"float"/>
    <HoldingRegister address="7" description=" Value" name="Value 1" type=
↪"float"/>
    <HoldingRegister address="8" description=" Value" name="Value 1" type=
↪"float"/>

    <HoldingRegister address="13" description=" Value" name="Value 1" type=
↪"float"/>
    <HoldingRegister address="14" description=" Value" name="Value 1" type=
↪"float"/>
    <HoldingRegister address="15" description=" Value" name="Value 1" type=
↪"float"/>
</ModbusDevice>
```

To prevent this and to query a single block it is necessary to fill the missing addresses.

```
<ModbusDevice name="MyDeviceType">
    <HoldingRegister address="0" description=" Value" name="Value 1" type=
↪"float"/>
    <HoldingRegister address="1" description=" Value" name="Value 1" type=
↪"float"/>
    <HoldingRegister address="2" description=" Value" name="Value 1" type=
↪"float"/>
    <HoldingRegister address="3" description=" dummy" name="dummy_3" type=
↪"float"/>
    <HoldingRegister address="4" description=" dummy" name=”dummy_4" type=
↪"float"/>
    <HoldingRegister address="5" description=" Value" name="Value 1" type=
↪"float"/>
    <HoldingRegister address="6" description=" Value" name="Value 1" type=
↪"float"/>
    <HoldingRegister address="7" description=" Value" name="Value 1" type=
↪"float"/>
    <HoldingRegister address="8" description=" Value" name="Value 1" type=
↪"float"/>
    <HoldingRegister address="9" description=" dummy" name="dummy_9" type=
↪"float"/>
    <HoldingRegister address="10" description=" dummy" name="dummy_10"␣
↪type="float"/>
    <HoldingRegister address="11" description=" dummy" name="dummy_11"␣
↪type="float"/>
```

(continues on next page)

(continued from previous page)

```
    <HoldingRegister address="12" description=" dummy" name="dummy_12"␣
↪type="float"/>
    <HoldingRegister address="13" description=" Value" name="Value 1" type=
↪"float"/>
    <HoldingRegister address="14" description=" Value" name="Value 1" type=
↪"float"/>
    <HoldingRegister address="15" description=" Value" name="Value 1" type=
↪"float"/>
</ModbusDevice>
```

6.1.1 TCP Endpoint

Table 6.1: TCPENDPOINT

Property	Value Options	Mandatory	Example	Description
name	string	yes	“My Endpoint”	Friendly name of the Endpoint
host	string	yes	“192.168.1.100”	Hostname of the Endpoint, IP address or hostname allowed
port	number	yes	“502”	IP-Port of the Endpoint, default is 502
polling_rate	rate [0.1Hz - 100Hz]	no	“1Hz”	Polling / reading rate of the endpoint
response_timeout	time	no	“0.25s”	Timeout for waiting on response of the endpoint

6.1.2 Device

Table 6.2: DEVICE

Property	Value Options	Mandatory	Example	Description
name	string	yes	“My Device”	Friendly name of the Device
device_type	string	yes	“MyDevice-Type”	Name of the used device, must be available as ModbusDevice
unit_id	number [0-255]	yes	“1”	Unit_Id of the device, typically “1”

6.1.3 MODBUS Device

Table 6.3: MODBUSDEVICE

Property	Value Options	Mandatory	Example	Description
name	string	yes	“MyDeviceType”	Friendly name of the Modbus Device
byte_order	see byte order	no	“big_endian”	Byte / Word Order

6.1.4 Register

- *Coil* use function code 0x01 for reading.
- *DiscreteInput* use function code 0x02 for reading.
- *HoldingRegister* use function code 0x03 for reading.
- *InputRegister* use function code 0x04 for reading.

Table 6.4: Register

Property	Value Options	Mandatory	Example	Description
address	number [0-65535]	yes	“40000”	Register Start address, starting with 0
name	string	yes	“Value 1”	Channel name of the Modbus Register in OXYGEN
type	data_type	yes	“int16”	Data type of the register value
scale_mode	“const_linear” or “sunssf”	no	“const_linear”	Scaling Mode
scale	number	no	“0.1”	Scaling factor, only valid if scale_mode == “const_linear”
offset	number	no	“100”	Scaling offset, only valid if scale_mode == “const_linear”
scale_reg	register	no	“1234”	Scaling register, only valid if scale_mode == “sunssf”
min	number	no	“-100”	Minimum display value range [RESERVED]
max	number	no	“100”	Maximum display value range [RESERVED]
unit	string	no	“V”	Value Unit
nan	value	no	“0xffff”	Value to be treated as NaN
byte_order	see byte order	no	“big_endian”	Byte / Word Order
description	string	no	“Description”	Channel description of the Modbus Register in OXYGEN

6.1.5 Datatypes

Table 6.5: Datatypes

Name	Word Count	Description	Value Range
“uint16”	1	Unsigned Integer 16 Bit	0 to 32767
“int16”	1	Signed Integer 16 Bit	-16384 to 16383
“uint32””	2	Unsigned Integer 32 Bit	0 to 2^32-1
“int32”	2	Signed Integer 32 Bit	-2^31 to 2^31-1
“float”	2	IEE 754 Floating Point Single	+-3.402823e+38
“double”	4	IEE 754 Floating Point Double	+-1e+308

6.1.6 Byte Order

Table 6.6: Byte Order

Name	Synonym	Description
“abcd”	“big_endian”	Decode Data in Big Endian matter (High-Byte before Low-Byte)
“dcba”	“little_endian”	Decode Data in Little Endian matter (Low-Byte before High-Byte)
“cdab”	“-”	Decode Data in Mixed-Byte Order
“badc”	“-”	Decode Data in Mixed-Byte Order