



ibaPDA-Interface-Modbus-TCP-Server

Data Interface Modbus-TCP-Server

Manual
Issue 3.0

Measurement Systems for Industry and Energy
www.iba-ag.com

Manufacturer

iba AG
Koenigswarterstrasse 44
90762 Fuerth
Germany

Contacts

Main office +49 911 97282-0
Fax +49 911 97282-33
Support +49 911 97282-14
Engineering +49 911 97282-13
E-mail iba@iba-ag.com
Web www.iba-ag.com

Unless explicitly stated to the contrary, it is not permitted to pass on or copy this document, nor to make use of its contents or disclose its contents. Infringements are liable for compensation.

© iba AG 2023, All rights reserved.

The content of this publication has been checked for compliance with the described hardware and software. Nevertheless, discrepancies cannot be ruled out, and we do not provide guarantee for complete conformity. However, the information furnished in this publication is updated regularly. Required corrections are contained in the following regulations or can be downloaded on the Internet.

The current version is available for download on our web site www.iba-ag.com.

Version	Date	Revision	Author	Version SW
3.0	10-2023	New version ibaPDA v8	RM/IP	8.4.0

Windows® is a brand and registered trademark of Microsoft Corporation. Other product and company names mentioned in this manual can be labels or registered trademarks of the corresponding owners.

Contents

1	About this documentation	5
1.1	Target group and previous knowledge	5
1.2	Notations	5
1.3	Used symbols.....	6
2	System requirements	7
3	Modbus-TCP-Server data interface	8
3.1	General information	8
3.1.1	Modbus TCP/IP	8
3.1.2	Client/Server architecture	9
3.1.3	Modbus protocol	9
3.1.4	Modbus TCP/IP - Message layout	12
3.1.4.1	Modbus Integer and Modbus Dig512	12
3.1.4.2	Modbus Real.....	13
3.1.4.3	Modbus Generic	13
3.1.4.4	Response	14
3.1.5	References	14
3.2	Configuration and engineering ibaPDA.....	15
3.2.1	General settings.....	15
3.2.2	General interface settings.....	16
3.2.3	Adding a module.....	17
3.2.3.1	General module settings.....	18
3.2.3.2	General signal configuration.....	18
3.2.3.3	Module type "Integer"	19
3.2.3.4	Module type "Dig512"	19
3.2.3.5	Module type "Real"	20
3.2.3.6	Module type "Generic".....	20
3.2.4	Module diagnostics.....	21
4	Diagnostics.....	22
4.1	License	22
4.2	Visibility of the interface.....	22
4.3	Log files.....	23
4.4	Connection diagnostics with PING.....	24

- 4.5 Checking the connection 25
- 4.6 Diagnostic modules 27
- 5 Appendix 32**
 - 5.1 Troubleshooting..... 32
 - 5.1.1 TCP performance problems caused by Delayed Acknowledge 32
 - 5.1.2 TCP data corruption resulting from the Nagle’s Algorithm..... 34
 - 5.2 Engineering examples..... 36
 - 5.2.1 Engineering example Modicon Quantum 36
 - 5.2.1.1 Configuration of the TCP/IP Interface in ProWORX NxT 36
 - 5.2.1.2 Ladder Program for the PLC..... 38
 - 5.2.1.3 ConCept Program for the PLC..... 40
 - 5.2.1.4 Unity Pro XL Program for the PLC (Generic module sample)..... 42
 - 5.2.2 Engineering example in PL7 Pro 44
 - 5.2.2.1 Network configuration..... 44
 - 5.2.2.2 Message configuration (example) 45
- 6 Support and contact..... 47**

1 About this documentation

This documentation describes the function and application of the software interface *ibaPDA-Interface-Modbus-TCP-Server*.

This documentation is a supplement to the *ibaPDA* manual. Information about all the other characteristics and functions of *ibaPDA* can be found in the *ibaPDA* manual or in the online help.

1.1 Target group and previous knowledge

This documentation is aimed at qualified professionals who are familiar with handling electrical and electronic modules as well as communication and measurement technology. A person is regarded as professional if he/she is capable of assessing safety and recognizing possible consequences and risks on the basis of his/her specialist training, knowledge and experience and knowledge of the standard regulations.

This documentation in particular addresses persons, who are concerned with the configuration, test, commissioning or maintenance of Programmable Logic Controllers of the supported products. For the handling *ibaPDA-Interface-Modbus-TCP-Server* the following basic knowledge is required and/or useful:

- Windows operating system
- Basic knowledge of *ibaPDA*
- Knowledge of configuration and operation of the relevant measuring device/system

1.2 Notations

In this manual, the following notations are used:

Action	Notation
Menu command	Menu <i>Logic diagram</i>
Calling the menu command	<i>Step 1 – Step 2 – Step 3 – Step x</i> Example: Select the menu <i>Logic diagram – Add – New function block</i> .
Keys	<Key name> Example: <Alt>; <F1>
Press the keys simultaneously	<Key name> + <Key name> Example: <Alt> + <Ctrl>
Buttons	<Key name> Example: <OK>; <Cancel>
Filenames, paths	Filename , Path Example: Test.docx

1.3 Used symbols

If safety instructions or other notes are used in this manual, they mean:

Danger!



The non-observance of this safety information may result in an imminent risk of death or severe injury:

- Observe the specified measures.

Warning!



The non-observance of this safety information may result in a potential risk of death or severe injury!

- Observe the specified measures.

Caution!



The non-observance of this safety information may result in a potential risk of injury or material damage!

- Observe the specified measures

Note



A note specifies special requirements or actions to be observed.

Tip



Tip or example as a helpful note or insider tip to make the work a little bit easier.

Other documentation



Reference to additional documentation or further reading.

2 System requirements

The following system requirements are necessary for the use of the Modbus-TCP-Server data interface:

- *ibaPDA* v8.0.0 or higher
- License for *ibaPDA-Interface-Modbus-TCP-Server*
- Network connection 10/100 Mbits

For further requirements for the used computer hardware and the supported operating systems, refer to the *ibaPDA* documentation.

Note



It is recommended carrying out the TCP/IP communication on a separate network segment to exclude a mutual influence by other network components.

System restrictions

- The maximum length of a Modbus TCP/IP message is limited to 244 bytes.
- For different ways of handling the TCP/IP acknowledge, see ↗ *TCP performance problems caused by Delayed Acknowledge*, page 32

Licenses

Order No.	Product name	Description
31.001020	ibaPDA-Interface-Modbus-TCP-Server	Extension license for an ibaPDA system providing an additional Modbus-TCP-Server interface. Number of connections: 64
31.101020	one-step-up-Interface-Modbus over TCPIP-Client	Extension license for the extension of an existing interface by another 64 Modbus-TCP-Server connections, max. 3 permitted

3 Modbus-TCP-Server data interface

3.1 General information

3.1.1 Modbus TCP/IP

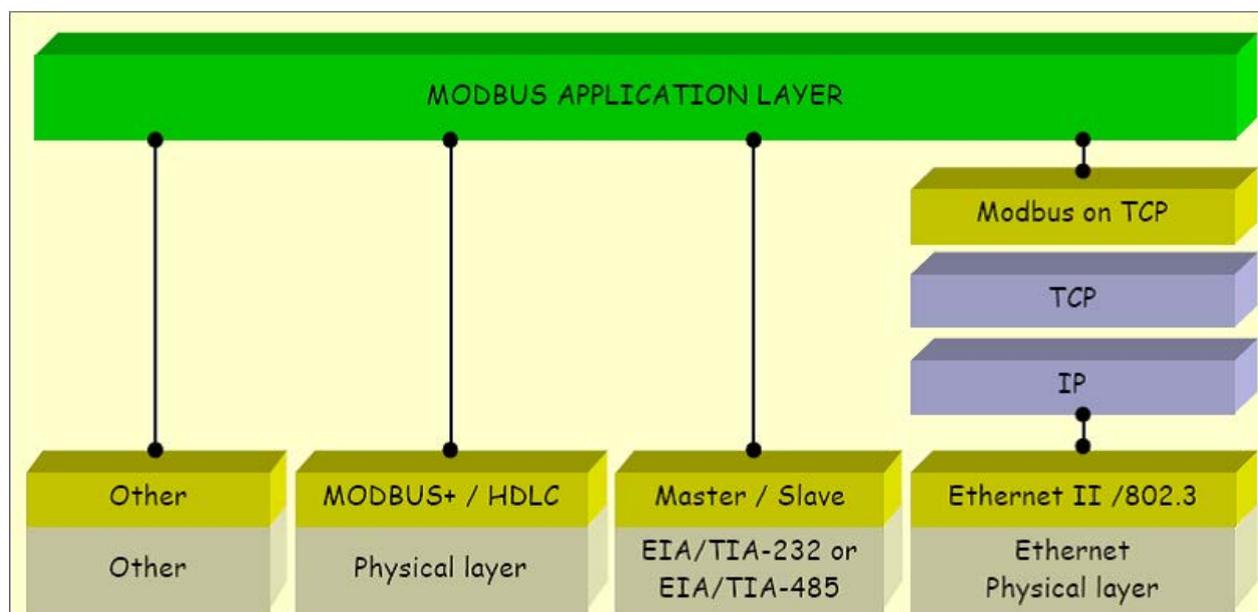
The Transmission Control Protocol (TCP) is one of the core protocols of the Internet protocol suite.

IP handles lower-level transmissions from computer to computer as a message makes its way across the Internet. TCP operates at a higher level (transport level), concerned with the two end systems. TCP provides a reliable data stream of bytes from a program on one computer to another program on another computer. TCP is explained in chapter 6 of RFC1180 and in RFC768, see [References](#), page 14.

Modbus is a protocol for the client/server communication between devices connected on different types of buses or networks.

Modbus is currently implemented in the following buses or networks as shown in the following figure:

- TCP/IP over Ethernet
- Asynchronous serial transmission over a variety of media
- Modbus PLUS (a high speed communication via a token passing network)



ibaPDA has the possibility to measure signals via the Modbus protocol over serial connections (Modbus ASCII and Modbus RTU) and over TCP/IP. This manual describes the connection via TCP/IP and as variant the transmission of the Modbus RTU protocol over TCP/IP, with *ibaPDA* acting as client.

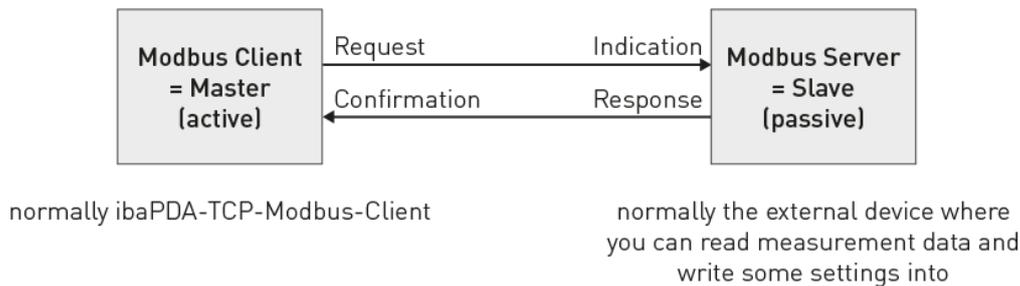
All systems that can receive and respond to messages with the Modbus-TCP protocol as server, can also communicate with *ibaPDA*.

3.1.2 Client/Server architecture

The Modbus service supports a client/server communication for devices which are connected via Ethernet TCP/IP.

The client/server model is based on 4 message types:

- Request
- Indication
- Response
- Confirmation



Read data: The Modbus-TCP-Client (*ibaPDA*) establishes the connection to the Modbus server, sends periodically the request and waits for the response, which contains the requested data.

Write data: The Modbus-TCP-Client (*ibaPDA*) establishes the connection to the Modbus server which contains the output data and waits for the response.

The port 502 is used for the Modbus TCP/IP communication by default, however you have got the possibility to enter other port numbers in *ibaPDA*.

With a *ibaPDA-Interface-Modbus-TCP-Server* license, *ibaPDA* can receive up to 64 connections, i.e. up to 64 Modbus servers can establish connections to *ibaPDA*. The number can be extended to a max. of 256 by loading the license more than once.

3.1.3 Modbus protocol

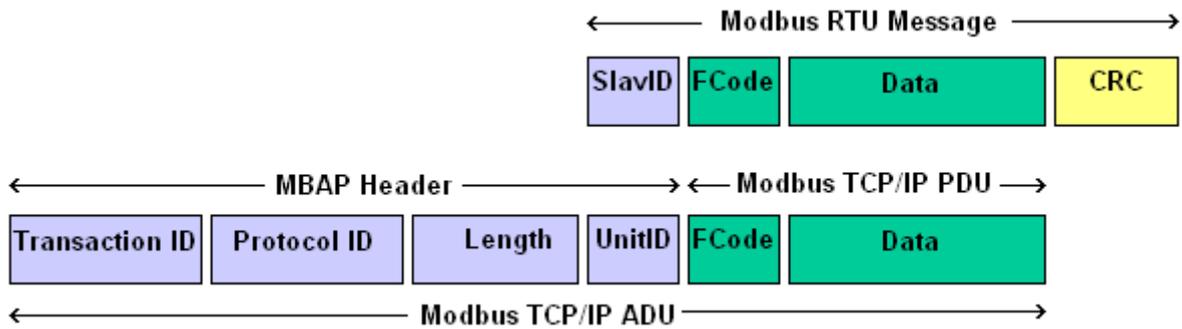
Byte sequence

Modbus uses "BIG ENDIAN", i.e. in the messages the bytes with a high significance are sent first and are thus stored in the addresses of low significance.

ibaPDA swaps all received 16- and 32-bit-values to the Intel format "LITTLE ENDIAN" ("Swapping"). You can select the Swapping method in *ibaPDA* for data that do not come from a Modbus controller. See [↗ General interface settings](#), page 16.

Modbus RTU / Modbus TCP

In the following representation, you can see the basic structure of the Modbus protocol and the differences between Modbus RTU and Modbus TCP.



RTU	Remote Terminal Unit
MBAP	Modbus Application Protocol
ADU	Application Data Unit
PDU	Protocol Data Unit

For Modbus TCP, the MBAP Header is put in front of the function code. The Unit Id. corresponds to the Slave Id. of the RTU protocol. The CRC code is omitted.

MBAP Header

The MBAP Header is a dedicated header used for the communication with TCP/IP to identify the Modbus Application Data.

The header contains the following fields:

Fields	Bytes	Description
Transaction Id.	2	Identification of Modbus request/response transaction
Protocol Id.	2	0 = Modbus protocol
Length	2	Number of following bytes
Unit Id.	1	Addressing a remote slave connected to the Modbus server

- **Transaction Identifier:** It is used for transaction pairing. The Modbus client sends it in the request; the Modbus server copies in the response message the transaction identifier of the request. *ibaPDA* evaluates this field as sequence count and waits for the value to be incremented by 1 in each cycle. In the event of an overflow, the counter must jump from 32767 to -32768 (0x7FFF → 0x8000) or from 65535 to 0 (0xFFFF → 0x0000).
- **Protocol Identifier:** It is used in multiplexing procedures. The Modbus protocol has the value 0.
- **Length** The length field is a count of the following bytes, including the Unit Identifier, Function Code and Data Fields. The maximum value is 251 (max. length of the user data bytes 244 + 7).

■ **Unit Identifier (device address):**

This field is sent by the Modbus client in the request and must be returned with the same value in the response by the server.

This field is not evaluated by *ibaPDA*.

Function code:

One byte contains the function code that determines which function the server has to carry out depending on the request.

The *ibaPDA-Interface-Modbus-TCP-Server* driver supports only the function

- 0x10: Write Multiple Registers

Data fields

The user data fields contain several subfields like starting address, number of registers, number of bytes and the actual data. The content of these fields depends on the used function code. For the function code 0x10, the data fields contain the following values:

Fields	Bytes	Description
Starting address	2	Starting address of the used storage area
Number of objects	2	Number of used registers or coils
Number of bytes	1	Number of data
Data range	n	n data bytes

■ **Starting address**

The *ibaPDA-Interface-Modbus-TCP-Server* driver uses the Modbus starting address. The starting address, that is called "Module index" in *ibaPDA*, is a number in which the data are assigned to a data module.

In *ibaPDA*, 4 module types are defined:

- Integer: 32 analog values (integer) and 32 binary signals
- Real: 32 analog values (real) and 32 binary signals
- Generic: data structure with a maximum length of 244 bytes.
- Dig512: 512 Binary signals (32 status words with 16 bits, each)

The module index is created by a serial number 00...63 and an offset that corresponds to the module type and the license.

Module type	1st License	2nd License	3rd License	4th License
Integer and Dig512	0-63	1000-1063	2000-2063	3000-3063
Real	100-163	1100-1163	2100-2163	3100-3163
Generic	200-263	1200-1263	2200-2263	3200-3263

The module index complies with the index in the *ibaPDA* module settings. This value must be unique and must not be changed during data transmission.

- Number of objects: This field shows the number of registers that are transmitted in one message. Modbus Integer and Modbus Dig512 send 34 registers. A Modbus Real module sends 66 registers. In the Generic module, the number of registers can be varied. However, it is limited to a max of 122. In this case, you have to enter the number of registers that are to be sent.
- Number of bytes: This value is always the number of registers multiplied by 2, as the registers are word-based (2 bytes). The maximum number is 244.
- Data range: The field contains the actual data sent to *ibaPDA*. The data type depends on the used Modbus module. Each module type has a maximum number of values that can be sent. An exception is the Generic module. In the Generic module, all available data types can be used simultaneously. In *ibaPDA*, you only have to configure the address of the signal and the data type.

3.1.4 Modbus TCP/IP - Message layout

The Modbus messages have the following module layout, corresponding to the module type:

3.1.4.1 Modbus Integer and Modbus Dig512

For the Integer module type, the 32 analog values are of the Integer type (16-bit) and the 32 digital values are densely packed as DWORD.

For the Dig512 module type, the 32 analog values are evaluated as 16-bit status words. The DWORD is not used.

Request Modbus Client -> ibaPDA (Modbus Server):

	Offs	Bytes	Type	Modbus Description	Contents (hex)	Remark:
	00	2	UINT	Transaction Id.	xx xx	Is evaluated as sequence count by <i>ibaPDA</i> , i.e. the ID has to be incremented each cycle.
MBAP	02	2	UINT	Protocol Id.	00 00	0
	04	2	UINT	Cmd Length	4B	75
	06	1	Bytes	Unit-ID	xx	not used
Fcode	07	1	Bytes	Function Code	10	"Write Multiple Registers"
Data	08	2	UINT	Starting Address	xx xx	Module index i000 – i063
	10	2	UINT	Number of objects	22	34 number of registers
	12	1	Bytes	Number of bytes	44	68 Number of bytes
User data	13	64	INT	Data	xx	32 analog values
	77	4	DWORD	Data	xx	32 digital values

3.1.4.2 Modbus Real

The analog values are of the FLOAT (IEEE format) type and the 32 digital values are densely packed as DWORD.

Request Modbus Client -> ibaPDA (Modbus Server):

	Offs	Bytes	Type	Modbus Description	Contents (hex)	Remark:
MBAP	00	2	UINT	Transaction Id.	xx xx	Is evaluated by the <i>ibaPDA</i> sequence count
	02	2	UINT	Protocol Id.	00 00	0
	04	2	UINT	Cmd Length	8B	139
	06	1	Bytes	Unit-ID	xx	not used
Fcode	07	1	Bytes	Function Code	10	"Write Multiple Registers"
Data	08	2	UINT	Starting Address	xx xx	Module index i100 to i163
	10	2	UINT	Number of objects	42	66 number of registers
	12	1	Bytes	Number of bytes	84	132 Number of bytes
User data	13	128	FLOAT	Data	xx	32 analog values
	141	4	DWORD	Data	xx	32 digital values

3.1.4.3 Modbus Generic

The user data area can have any desired data structure with different data formats. *ibaPDA* supports the following data formats:

BYTE, WORD, DWORD, INT, DINT and FLOAT.

The data structure defined here has to be modeled in *ibaPDA*. The BYTE, WORD and DWORD variables may also be interpreted as 8, 16 or 32 single bits (and vice versa).

Request Modbus Client -> ibaPDA (Modbus Server):

	Offs	Bytes	Type	Modbus Description	Contents (hex)	Remark:
MBAP	00	2	UINT	Transaction Id.	xx xx	Is evaluated by the <i>ibaPDA</i> sequence count
	02	2	UINT	Protocol Id.	00 00	0
	04	2	UINT	Cmd Length	xx	n + 7
	06	1	Bytes	Unit-ID	xx	not used
Fcode	07	1	Bytes	Function Code	10	"Write Multiple Registers"

	Offs	Bytes	Type	Modbus Description	Contents (hex)	Remark:
Data	08	2	UINT	Starting Address	xx xx	Module index i200 to i263
	10	2	UINT	Number of objects	42	n/2: rounded up
	12	1	Bytes	Number of bytes	84	n (max. 122)
User data	13	nn	nnnn	Data	xx	Arbitrary data structure (max. 244 bytes)

3.1.4.4 Response

Each telegram is answered by the Modbus server with a response telegram by default. You can suppress this function in *ibaPDA*.

Response *ibaPDA* (Modbus Server) → Modbus Client:

	Offs	Bytes	Type	Modbus Description	Content (hex)	<i>ibaPDA</i> Description
MBAP	00	2	UINT	Transaction Id.	xx xx	Sequence count, Mirror of request
	02	2	UINT	Protocol Id.	00 00	0
	04	2	UINT	Cmd Length	00 06	6
	06	1	Bytes	Unit-ID	xx	Mirror of request
Fcode	07	1	Bytes	Function Code	10	Mirror of request
Data	08	2	UINT	Starting Address	xx	Mirror of request
	10	2	UINT	Number of objects	xx	Mirror of request

3.1.5 References

Other documentation



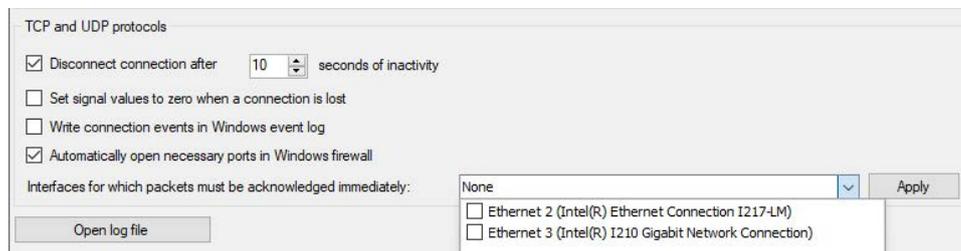
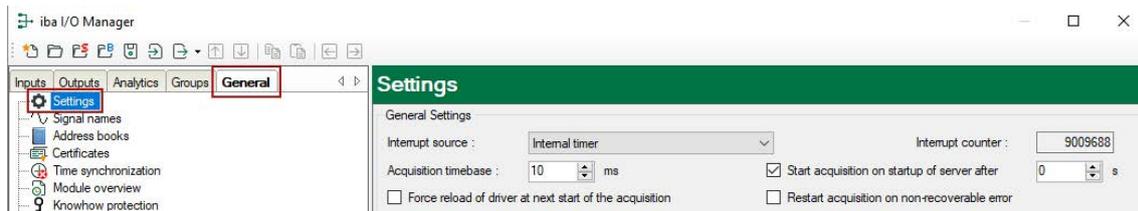
- *ibaPDA* manual
- A TCP/IP tutorial, RFC1180 (<ftp://ftp.ripe.net/rfc/rfc1180.txt>)
- Transmission Control Protocol, RFC793 (<ftp://ftp.ripe.net/rfc/rfc793.txt>)
- Modbus Messaging Implementation Guide V1 (<http://www.modbus.org>)
- Modbus Application Protocol V1.1 (<http://www.modbus.org>)
- Modbus Protocol Reference Guide Rev J, Modicon

3.2 Configuration and engineering ibaPDA

The engineering for *ibaPDA* is described in the following. If all system requirements are fulfilled, *ibaPDA* displays the *Modbus TCP Server* interface in the interface tree of the I/O Manager.

3.2.1 General settings

The "Alive timeout" is configured jointly for all TCP/IP and UDP protocols supported by *ibaPDA*.



Disconnect connection after ... seconds of inactivity

Behavior and timeout duration can be specified.

Set signal values to zero when a connection is lost

If this option is disabled, the value read last will be kept.

Write connection events in Windows event log

Current events are logged in Windows.

Automatically open necessary ports in Windows firewall

If this option is enabled, all ports required for the currently licensed interfaces are automatically opened in the firewall by the *ibaPDA* server service.

If this option is disabled, the required ports can be opened manually in the I/O Manager of the licensed interfaces via <Allow port through firewall>.

Interfaces for which packets must be acknowledged immediately

Selection of required interfaces.

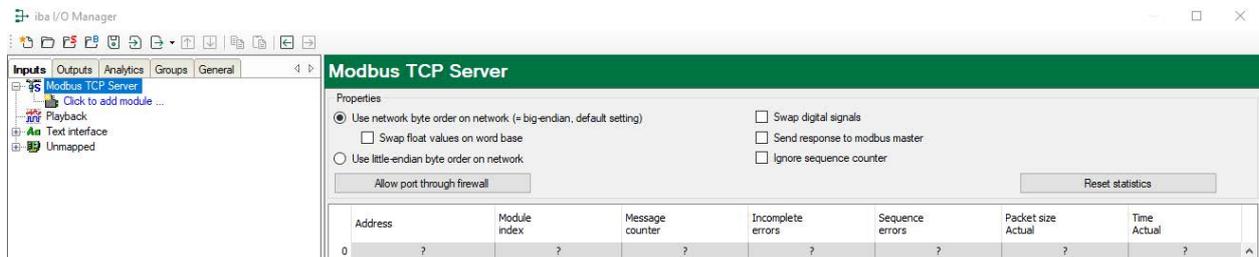
Note



In case *ibaPDA* is the active partner (Client), *ibaPDA* reestablishes the connection after only a few seconds. Thus, it gives to the passive partner the possibility to send data again.

3.2.2 General interface settings

The interface provides the following functions and configuration options:



Use network byte order on network (= big endian, default setting)

Optional:

Use little endian byte order on network

see [↗ Modbus protocol](#), page 9.

Swap float values on word base

Here, you can change the byte order for data that are not provided by Modbus devices. The bytes are swapped according to the pattern ABCD → CDAB. This option concerns only the module type Real; for the module type Generic, specific setting options are defined.

Swap digital signals

Here, you can change the byte order for data that are not provided by Modbus devices. The bytes are swapped according to the pattern ABCD → BADC.

Send response to modbus master

Each telegram is acknowledged by a response telegram, see [↗ Response](#), page 14. You can suppress the response by de-activating this option.

Ignore sequence counter

If this option is enabled, the "Sequence errors" column is hidden in the connection overview.

Allow ports through firewall

When installing *ibaPDA*, the default port numbers of the used protocols are automatically entered in the firewall. If you change the port number or enable the interface subsequently, you have to enable this port in the firewall with this button.

<Reset statistics>

Click this button to reset the calculated times and error counters in the table to 0.

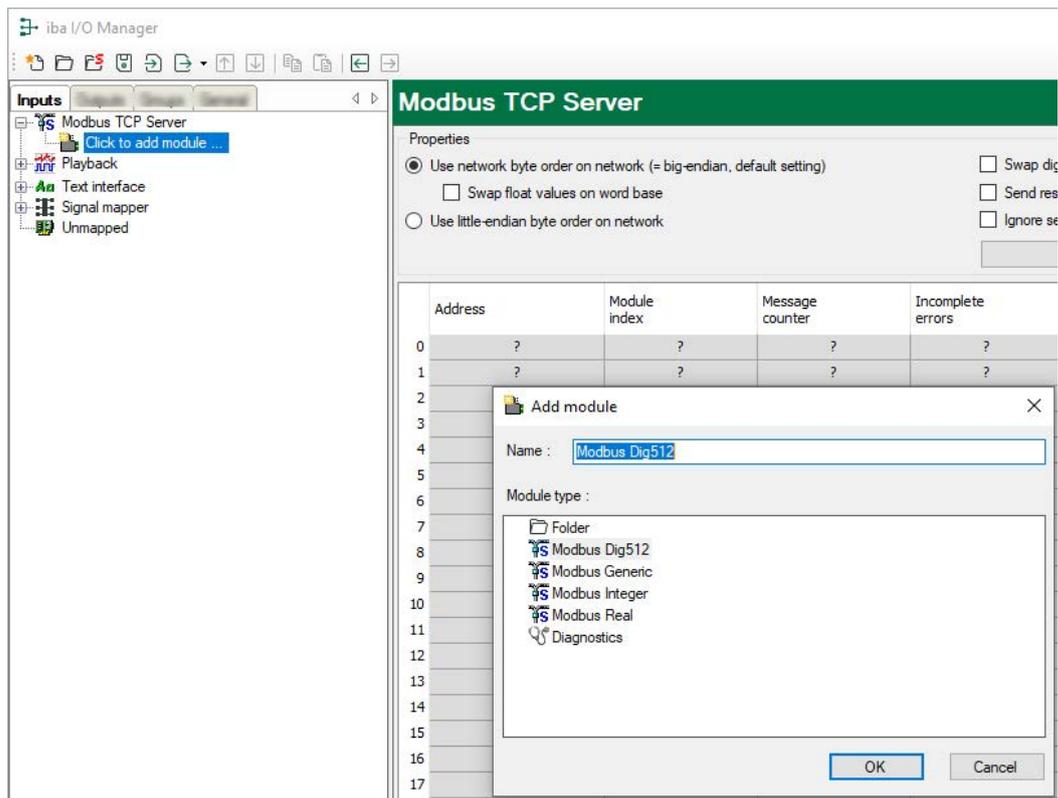
Overview of connections:

As soon as the connection has been established, you can see the live data in the overview. See [↗ Checking the connection](#), page 25.

3.2.3 Adding a module

Procedure

1. Click on the blue command *Click to add module...* located under each data interface in the *Inputs* or *Outputs* tab.
2. Select the desired module type in the dialog box and assign a name via the input field if required.
3. Confirm the selection with <OK>.



Tip



If a TCP/IP connection already exists, right-click the interface and select "Auto-detect". Then, the correct modules are automatically created for all available connections.

Module types

The following modules types are available:

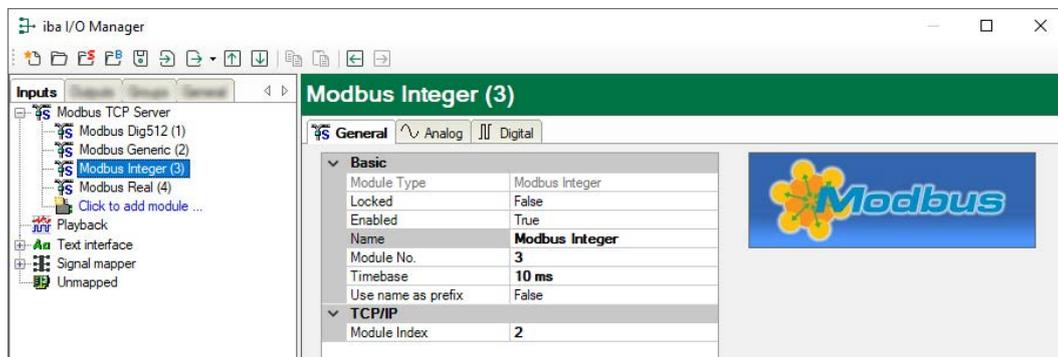
- Generic
- Dig512
- Integer
- Real

For further information, refer to the respective chapters under [General information](#), page 8 and [Configuration and engineering ibaPDA](#), page 15

3.2.3.1 General module settings

To configure a module, select it in the tree structure.

All modules have the following setting options.



Basic settings

Module Type (information only)

Indicates the type of the current module.

Locked

You can lock a module to avoid unintentional or unauthorized changing of the module settings.

Enabled

Enable the module to record signals.

Name

You can enter a name for the module here.

Module No.

This internal reference number of the module determines the order of the modules in the signal tree of *ibaPDA* client and *ibaAnalyzer*.

Timebase

All signals of the module are sampled on this timebase.

Use name as prefix

This option puts the module name in front of the signal names.

TCP/IP

Module index:

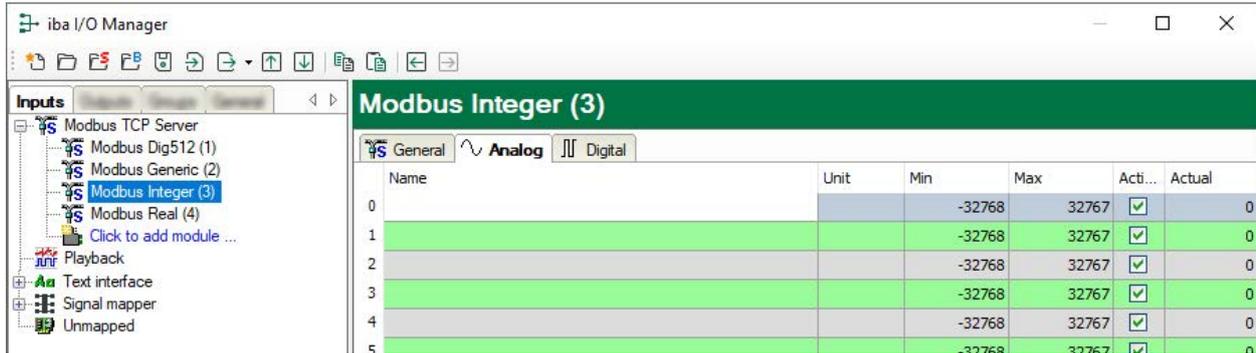
The module indices are created by a serial number 00...63 and an offset that corresponds to the module type and the license. ↗ *Modbus protocol*, page 9

For a detailed description of the parameters, see the *ibaPDA* manual.

3.2.3.2 General signal configuration

The data to be measured are selected on the Modbus client side by mapping the signals to the respective data ranges.

Analog and digital tab



Name	Unit	Min	Max	Acti...	Actual
0		-32768	32767	<input checked="" type="checkbox"/>	0
1		-32768	32767	<input checked="" type="checkbox"/>	0
2		-32768	32767	<input checked="" type="checkbox"/>	0
3		-32768	32767	<input checked="" type="checkbox"/>	0
4		-32768	32767	<input checked="" type="checkbox"/>	0
5		-32768	32767	<input checked="" type="checkbox"/>	0

You can assign name, unit, scale factor and comments to the signals. Moreover, you can enable or disable the signals.

Other documentation



For a description of the columns, please see the *ibaPDA* manual.

Note



The module *TDC TCP/UDP Generic* supports the acquisition and processing of strings as text signals. Therefore, you can select the datatype `STRING[32]` in the *Analog* tab. In order to convert a text signal order to split it up into several text signals use the *text splitter* module under the *Virtual* interface.

Tip



You can use the automatic fill function in the columns (see *ibaPDA* manual).

3.2.3.3 Module type "Integer"

The "Integer" module allows up to 32 analog signals (Integer) and 32 binary signals to be acquired.

The module does not have any module specific settings.

3.2.3.4 Module type "Dig512"

With the "Dig512" module, you can acquire up to 512 digital values, organized as 32 status words (type Integer) with 16 bits, each.

The module does not have any module specific settings, there are only the *General* and *Digital* tabs.

3.2.3.5 Module type "Real"

The "Real" module allows up to 32 analog signals (Real) and 32 binary signals to be acquired.

The following module settings are module-specific:

- No. analog signals
You can set up the number of analog signals to be measured (1 to 32).
Please, consider the following:
In case you want to use digital signals, the message must have the structure described in [↗ Modbus Real](#), page 13. If you do not use digital signals, the message can be shortened.
- In the analog table, Gain/Offset are scaled.

3.2.3.6 Module type "Generic"

Any data block with a max. length of 244 bytes can be measured by means of the module "Generic".

The following module settings are module-specific:

- Swap analog signals, swap digital signals
You can change the byte evaluation order. (The interface settings are not valid here!)
- No. analog signals, No. digital signals
Maximum number of analog and digital signals that can be configured.
- In the analog table, Gain/Offset are scaled.
- For each variable, you have to enter the address, i.e. the offset in the telegram buffer and the data type. Bear in mind that counting starts from the beginning of user data without header.

Note



The module *Modbus Generic* supports the acquisition and processing of strings as text signals. Therefore, you can select the datatype `STRING[32]` in the *Analog* tab. In order to convert a text signal oder to split it up into several text signals use the *text splitter* module under the *Virtual* interface.

Description of the columns:

Name, Unit , Gain, Offset, Active

see *ibaPDA* manual

Address

The address defines the byte offset of the value within the user data of the telegram and depends on the data type of the preceding data. After changing data types, it may thus be necessary to adjust the address entries.

Data Type (analog signals only)

The following data types are supported: SINT, BYTE, INT, WORD, DWORD, DINT, FLOAT, DOUBLE, `STRING[32]`.

Tip

Please consider, that the address depends on the data type of the preceding data. This is why we recommend setting the data types and then the addresses using the autofill function

3.2.4 Module diagnostics

The tables "Analog" and "Digital" of the modules show the telegram contents.

General Analog Digital									
Name	Unit	Gain	Offset	Address	DataType	Active	Actual		
0			1	0	0 INT	<input checked="" type="checkbox"/>	1		
1 Sine INT		1	0	2	INT	<input checked="" type="checkbox"/>	-834		
2 Cosine INT			1	0	4 INT	<input checked="" type="checkbox"/>	554		
3 Triangle INT			1	0	6 INT	<input checked="" type="checkbox"/>	372		
4			1	0	8 INT	<input checked="" type="checkbox"/>	0		
5			1	0	10 INT	<input checked="" type="checkbox"/>	0		
6			1	0	12 INT	<input checked="" type="checkbox"/>	0		
7			1	0	14 INT	<input checked="" type="checkbox"/>	0		
8			1	0	16 DINT	<input checked="" type="checkbox"/>	1		
9 Sine DINT		1	0	20	DINT	<input checked="" type="checkbox"/>	-834		
10 Cosine DINT			1	0	24 DINT	<input checked="" type="checkbox"/>	554		
11 Triangle DINT			1	0	28 DINT	<input checked="" type="checkbox"/>	372		
12			1	0	32 FLOAT	<input checked="" type="checkbox"/>	1		
13 Sine REAL		1	0	36	FLOAT	<input checked="" type="checkbox"/>	-833,918		
14 Cosine REAL			1	0	40 FLOAT	<input checked="" type="checkbox"/>	554,079		
15 Triangle REAL			1	0	44 FLOAT	<input checked="" type="checkbox"/>	372,185		

The following errors may occur:

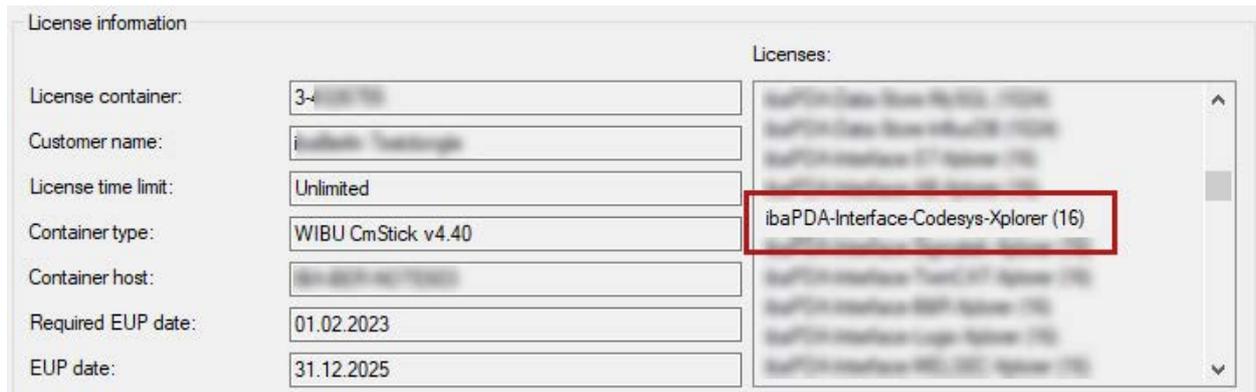
- No data are displayed:
 - The telegram buffer on the sender side is not filled correctly
 - The connectors of the send block are connected incorrectly.
- Incorrect values are displayed:
 - The telegram buffer on the controller side is not filled correctly (offset error)
 - The byte order is set incorrectly (see ↗ *General module settings*, page 18)
 - There are multiple modules with the same module index.
- The digital signals are sorted incorrectly:
 - The byte order is set incorrectly (see ↗ *General module settings*, page 18)
- The telegrams do not arrive faster than approx. 200 ms with sequence error:
 - Problem with "Delayed Acknowledge", see ↗ *TCP performance problems caused by Delayed Acknowledge*, page 32
 - Problem caused by "Nagle's Algorithm", see ↗ *TCP data corruption resulting from the Nagle's Algorithm*, page 34

4 Diagnostics

4.1 License

If the interface is not displayed in the signal tree, you can either check in *ibaPDA* in the I/O Manager under *General – Settings* or in the *ibaPDA* service status application whether your license for this interface has been properly recognized. The number of licensed connections is shown in brackets.

The figure below shows the license for the *Codesys Xplorer* interface as an example.



4.2 Visibility of the interface

If the interface is not visible despite a valid license, it may be hidden.

Check the settings in the *General* tab in the *Interfaces* node.

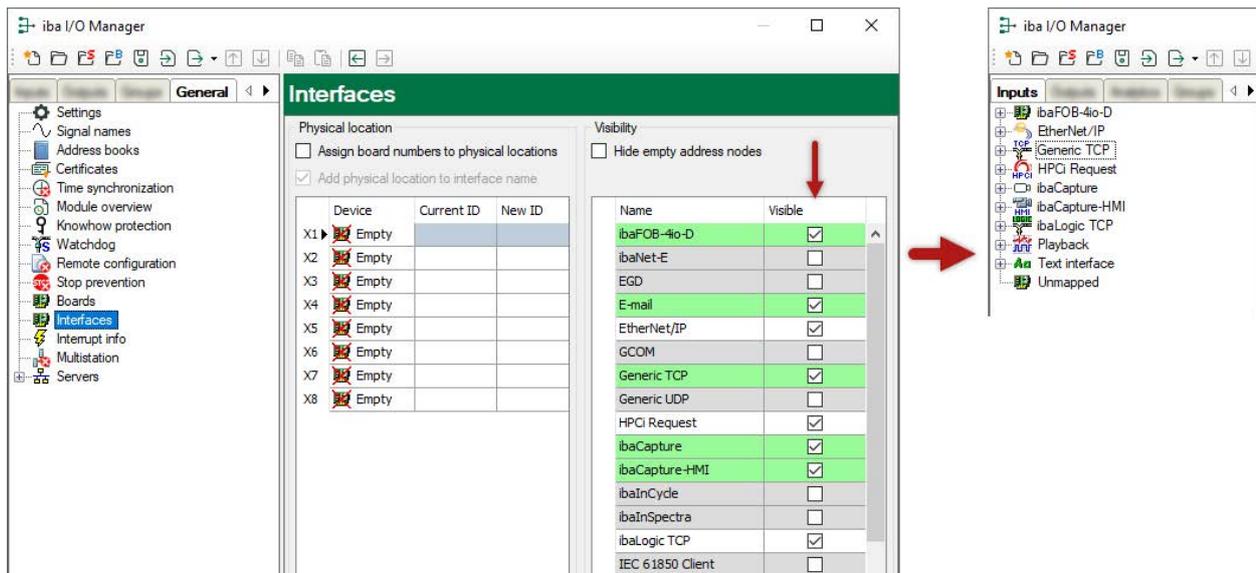
Visibility

The table *Visibility* lists all the interfaces that are available either through licenses or installed cards. These interfaces can also be viewed in the interface tree.

You can hide or display the interfaces not required in the interface tree by using the checkbox in the *Visible* column.

Interfaces with configured modules are highlighted in green and cannot be hidden.

Selected interfaces are visible, the others are hidden:



4.3 Log files

If connections to target platforms or clients have been established, all connection-specific actions are logged in a text file. You can open this (current) file and, e.g., scan it for indications of possible connection problems.

You can open the log file via the button <Open log file>. The button is available in the I/O Manager:

- for many interfaces in the respective interface overview
- for integrated servers (e.g. OPC UA server) in the *Diagnostics* tab.

In the file system on the hard drive, you can find the log files of the *ibaPDA* server (...\[ProgramData\iba\ibaPDA\Log](#)). The file names of the log files include the name or abbreviation of the interface type.

Files named [interface.txt](#) are always the current log files. Files named [Interface_yyyy_mm_dd_hh_mm_ss.txt](#) are archived log files.

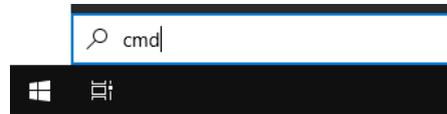
Examples:

- [ethernetipLog.txt](#) (log of EtherNet/IP connections)
- [AbEthLog.txt](#) (log of Allen-Bradley Ethernet connections)
- [OpcUAServerLog.txt](#) (log of OPC UA server connections)

4.4 Connection diagnostics with PING

PING is a system command with which you can check if a certain communication partner can be reached in an IP network.

1. Open a Windows command prompt.



2. Enter the command "ping" followed by the IP address of the communication partner and press <ENTER>.

→ With an existing connection you receive several replies.

```
Administrator: Command Prompt
Microsoft Windows [Version 10.0]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>ping 192.168.1.10

Pinging 192.168.1.10 with 32 bytes of data:
Reply from 192.168.1.10: bytes=32 time=1ms TTL30
Reply from 192.168.1.10: bytes=32 time<1ms TTL30
Reply from 192.168.1.10: bytes=32 time<1ms TTL30
Reply from 192.168.1.10: bytes=32 time<1ms TTL30

Ping statistics for 192.168.1.10:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 1ms, Average = 0ms

C:\Windows\system32>
```

→ With no existing connection you receive error messages.

```
Administrator: Command Prompt
Microsoft Windows [Version 10.0]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>ping 192.168.1.10

Pinging 192.168.1.10 with 32 bytes of data:
Reply from 192.168.1.10: Destination host unreachable.

Ping statistics for 192.168.1.10:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),

C:\Windows\system32>
```

4.5 Checking the connection

If you mark the data interface "Modbus TCP Server" in the signal tree of the I/O manager, you will see a table in the right part of the window which shows all available connections of this interface.

Modbus TCP Server

Properties

Use network byte order on network (= big-endian, default setting)
 Swap float values on word base
 Use little-endian byte order on network

Swap digital signals
 Send response to modbus master
 Ignore sequence counter

	Address	Module index	Message counter	Incomplete errors	Sequence errors	Packet size Actual	Time Actual
0	192.168.50.209	1	1465	0	0	81	21,5 ms
1	192.168.50.209	101	1465	0	0	145	21,5 ms
2	192.168.50.209	201	1469	1469	0	69	21,5 ms
3	?	?	?	?	?	?	?
4	?	?	?	?	?	?	?
5	?	?	?	?	?	?	?
6	?	?	?	?	?	?	?

Buttons:

Allow ports through firewall

When installing *ibaPDA*, the default port numbers of the used protocols are automatically entered in the firewall. If you change the port number or enable the interface subsequently, you have to enable this port in the firewall with this button.

<Reset statistics>

Click this button to reset the calculated times and error counters in the table to 0.

The list of connections shows the following values:

- Address: Address of the Modbus server
- Module index: "Starting address" field from the telegram header.
- Message ounter: Number of messages received
- Incomplete errors: Is incremented each time the length of the telegram does not equal the length defined in the telegram header.
- Sequence errors: Is incremented each time, the counter in the "Transaction Id." field of the header is not incremented each cycle by 1.
- Packet size Actual: The complete telegram length
- Time Actual: Cycle in which the telegrams of the Modbus client arrive

Colors:

- Green: Connection OK. the "Time Actual" approximately corresponds to the timebase of the module
- Orange: The connection is OK, the timebase of the module is much faster than "Time Actual". The timebase of the module can be adapted for optimizing purposes.

A failed connection may have the following causes:

- Modbus client is in stop
- No Ethernet connection between *ibaPDA* PC and the Modbus PLC
- Error in the connection configuration:
 - incorrect remote IP address
 - The *ibaPDA* port number and the connection configuration do not match.

Other errors:

- If values in the columns "Incomplete errors" and/or "Sequence errors" are incremented, this points to one of the following errors:
 - Error in the message header
 - Error in the byte order
 - The "delayed acknowledge" problem occurs, see [↗ TCP performance problems caused by Delayed Acknowledge](#), page 32

4.6 Diagnostic modules

Diagnostic modules are available for most Ethernet based interfaces and Xplorer interfaces. Using a diagnostic module, information from the diagnostic displays (e.g. diagnostic tabs and connection tables of an interface) can be acquired as signals.

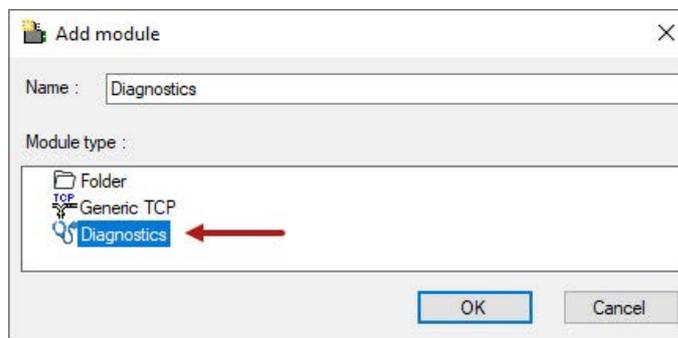
A diagnostic module is always assigned to a data acquisition module of the same interface and supplies its connection information. By using a diagnostic module you can record and analyze the diagnostic information continuously in the *ibaPDA* system.

Diagnostic modules do not consume any license connections because they do not establish their own connection, but refer to another module.

Example for the use of diagnostic modules:

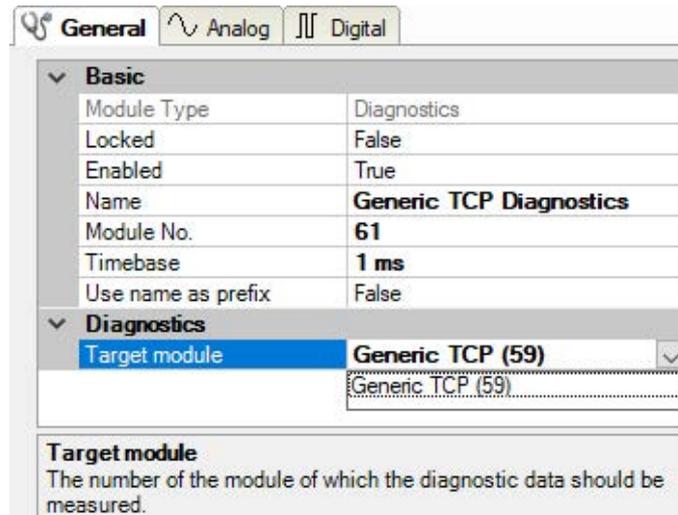
- A notification can be generated, whenever the error counter of a communication connection exceeds a certain value or the connection gets lost.
- In case of a disturbance, the current response times in the telegram traffic may be documented in an incident report.
- The connection status can be visualized in *ibaQPanel*.
- You can forward diagnostic information via the SNMP server integrated in *ibaPDA* or via OPC DA/UA server to superordinate monitoring systems like network management tools.

In case the diagnostic module is available for an interface, a "Diagnostics" module type is shown in the "Add module" dialog (example: Generic TCP).



Module settings diagnostic module

For a diagnostic module, you can make the following settings (example: Generic TCP):



The basic settings of a diagnostic module equal those of other modules.

There is only one setting which is specific for the diagnostic module: the target module.

By selecting the target module, you assign the diagnostic module to the module on which you want to acquire information about the connection. You can select the supported modules of this interface in the drop down list of the setting. You can assign exactly one data acquisition module to each diagnostic module. When having selected a module, the available diagnostic signals are immediately added to the *Analog* and *Digital* tabs. It depends on the type of interface, which signals exactly are added. The following example lists the analog values of a diagnostic module for a Generic TCP module.

General Analog Digital						
Name	Unit	Gain	Offset	Active	Actual	
0 IP address (part 1)			1	0	<input checked="" type="checkbox"/>	
1 IP address (part 2)			1	0	<input checked="" type="checkbox"/>	
2 IP address (part 3)			1	0	<input checked="" type="checkbox"/>	
3 IP address (part 4)			1	0	<input checked="" type="checkbox"/>	
4 Port			1	0	<input checked="" type="checkbox"/>	
5 Message counter			1	0	<input checked="" type="checkbox"/>	
6 Incomplete errors			1	0	<input checked="" type="checkbox"/>	
7 Packet size (actual)	bytes		1	0	<input checked="" type="checkbox"/>	
8 Packet size (max)	bytes		1	0	<input checked="" type="checkbox"/>	
9 Time between data (actual)	ms		1	0	<input checked="" type="checkbox"/>	
10 Time between data (min)	ms		1	0	<input checked="" type="checkbox"/>	

For example, the IP (v4) address of a Generic TCP module (see fig. above) will always be split into 4 parts derived from the dot-decimal notation, for better reading. Also other values are being determined, as there are port number, counters for telegrams and errors, data sizes and telegram cycle times. The following example lists the digital values of a diagnostic module for a Generic TCP module.

General Analog Digital			
Name	Active	Actual	
0 Active connection mode	<input checked="" type="checkbox"/>		
1 Invalid packet	<input checked="" type="checkbox"/>		
2 Connecting	<input checked="" type="checkbox"/>		
3 Connected	<input checked="" type="checkbox"/>		

Diagnostic signals

Depending on the interface type, the following signals are available:

Signal name	Description
Active	Only relevant for redundant connections. Active means that the connection is used to measure data, i.e. for redundant standby connections the value is 0. For normal/non-redundant connections, the value is always 1.
Buffer file size (actual/avg/max)	Size of the file for buffering statements
Buffer memory size (actual/avg/max)	Size of the memory used by buffered statements
Buffered statements	Number of unprocessed statements in the buffer
Buffered statements lost	Number of buffered but unprocessed and lost statements
Connected	Connection is established
Connected (in)	A valid data connection for the reception (in) is available
Connected (out)	A valid data connection for sending (out) is available
Connecting	Connection being established
Connection attempts (in)	Number of attempts to establish the receive connection (in)
Connection attempts (out)	Number of attempts to establish the send connection (out)
Connection ID O->T	ID of the connection for output data (from the target system to <i>ibaPDA</i>). Corresponds to the assembly instance number
Connection ID T->O	ID of the connection for input data (from <i>ibaPDA</i> to target system). Corresponds to the assembly instance number
Connection phase (in)	Status of the <i>ibaNet-E</i> data connection for reception (in)
Connection phase (out)	Status of the <i>ibaNet-E</i> data connection for sending (out)
Connections established (in)	Number of currently valid data connections for reception (in)
Connections established (out)	Number of currently valid data connections for sending (out)
Data length	Length of the data message in bytes
Data length O->T	Size of the output message in byte
Data length T->O	Size of the input message in byte
Destination IP address (part 1-4) O->T	4 octets of the IP address of the target system Output data (from target system to <i>ibaPDA</i>)
Destination IP address (part 1-4) T->O	4 octets of the IP address of the target system Input data (from <i>ibaPDA</i> to target system)
Disconnects (in)	Number of currently interrupted data connections for reception (in)
Disconnects (out)	Number of currently interrupted data connections for sending (out)
Error counter	Communication error counter
Exchange ID	ID of the data exchange
Incomplete errors	Number of incomplete messages

Signal name	Description
Incorrect message type	Number of received messages with wrong message type
Input data length	Length of data messages with input signals in bytes (<i>ibaPDA</i> receives)
Invalid packet	Invalid data packet detected
IP address (part 1-4)	4 octets of the IP address of the target system
Keepalive counter	Number of KeepAlive messages received by the OPC UA Server
Lost images	Number of lost images (in) that were not received even after a retransmission
Lost Profiles	Number of incomplete/incorrect profiles
Message counter	Number of messages received
Messages per cycle	Number of messages in the cycle of the update time
Messages received since configuration	Number of received data telegrams (in) since start of acquisition
Messages received since connection start	Number of received data telegrams (in) since the start of the last connection setup. Reset with each connection loss.
Messages sent since configuration	Number of sent data telegrams (out) since start of acquisition
Messages sent since connection start	Number of sent data telegrams (out) since the start of the last connection setup. Reset with each connection loss.
Multicast join error	Number of multicast login errors
Number of request commands	Counter for request messages from <i>ibaPDA</i> to the PLC/CPU
Output data length	Length of the data messages with output signals in bytes (<i>ibaPDA</i> sends)
Packet size (actual)	Size of the currently received message
Packet size (max)	Size of the largest received message
Ping time (actual)	Response time for a ping telegram
Port	Port number for communication
Producer ID (part 1-4)	Producer ID as 4 byte unsigned integer
Profile Count	Number of completely recorded profiles
Read counter	Number of read accesses/data requests
Receive counter	Number of messages received
Response time (actual/average/max/min)	Response time is the time between measured value request from <i>ibaPDA</i> and response from the PLC or reception of the data. Actual: current value Average/max/min: static values of the update time since the last start of the acquisition or reset of the counters.
Retransmission requests	Number of data messages requested again if lost or delayed

Signal name	Description
Rows (last)	Number of resulting rows by the last SQL query (within the configured range of result rows)
Rows (maximum)	Maximum number of resulting rows by any SQL query since the last start of acquisition (possible maximum equals the configured number of result rows)
Send counter	Number of send messages
Sequence errors	Number of sequence errors
Source IP address (part 1-4) O->T	4 octets of the IP address of the target system Output data (from target system to <i>ibaPDA</i>)
Source IP address (part 1-4) T->O	4 octets of the IP address of the target system Input data (from <i>ibaPDA</i> to target system)
Statements processed	Number of executed statements since last start of acquisition
Synchronization	Device is synchronized for isochronous acquisition
Time between data (actual/ max/min)	Time between two correctly received messages Actual: between the last two messages Max/min: statistical values since start of acquisition or reset of counters
Time offset (actual)	Measured time difference of synchronicity between <i>ibaPDA</i> and the <i>ibaNet-E</i> device
Topics Defined	Number of defined topics
Topics Updated	Number of updated topics
Unknown sensor	Number of unknown sensors
Update time (actual/average/ configured/max/min)	Specifies the update time in which the data is to be retrieved from the PLC, the CPU or from the server (configured). Default is equal to the parameter "Timebase". During the measurement the real actual update time (actual) can be higher than the set value, if the PLC needs more time to transfer the data. How fast the data is really updated, you can check in the connection table. The minimum achievable update time is influenced by the number of signals. The more signals are acquired, the greater the update time becomes. Average/max/min: static values of the update time since the last start of the acquisition or reset of the counters.
Write counter	Number of successful write accesses
Write lost counter	Number of failed write accesses

5 Appendix

5.1 Troubleshooting

5.1.1 TCP performance problems caused by Delayed Acknowledge

Symptoms:

ibaPDA measurements of automation devices using TCP/IP sometimes do not work with cycle times < 200 ms.

Errors shown in *ibaPDA*:

Incomplete telegrams and/or spikes in data values (depending on the sending controller type)

Cause:

There are different variants of handling "acknowledge" in the TCP/IP protocol:

The standard WinSocket works in accordance with RFC1122 using the "delayed acknowledge" mechanism (Delayed ACK). It specifies that the "acknowledge" is delayed until other telegrams arrive in order to acknowledge them jointly. If no other telegrams arrive, the ACK telegram is sent after 200 ms at the latest (depending on the socket).

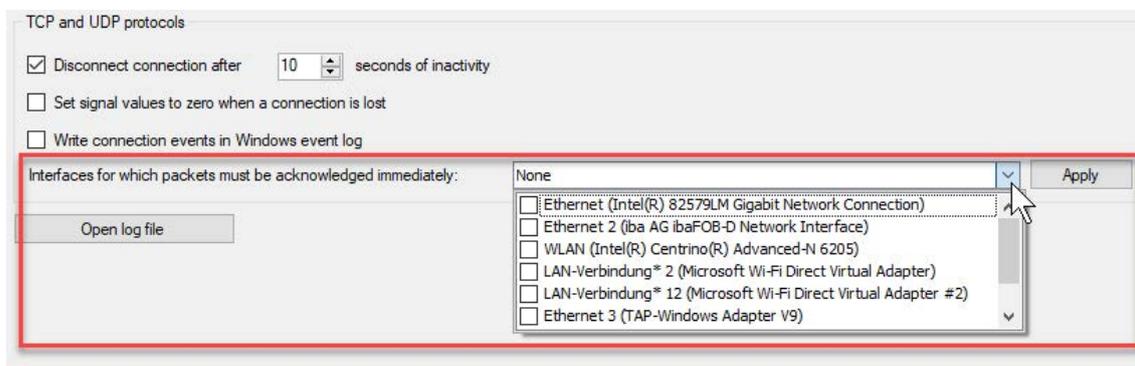
The data flow is controlled by a "sliding window" (parameter Win=nnnn). The recipient specifies how many bytes it can receive without sending an acknowledgment.

Some controllers do not accept this response, but instead, wait for an acknowledgment after each data telegram. If it does not arrive within a certain period of time (200 ms), it will repeat the telegram and include any new data to be sent, causing an error with the recipient, because the old one was received correctly.

Remedy:

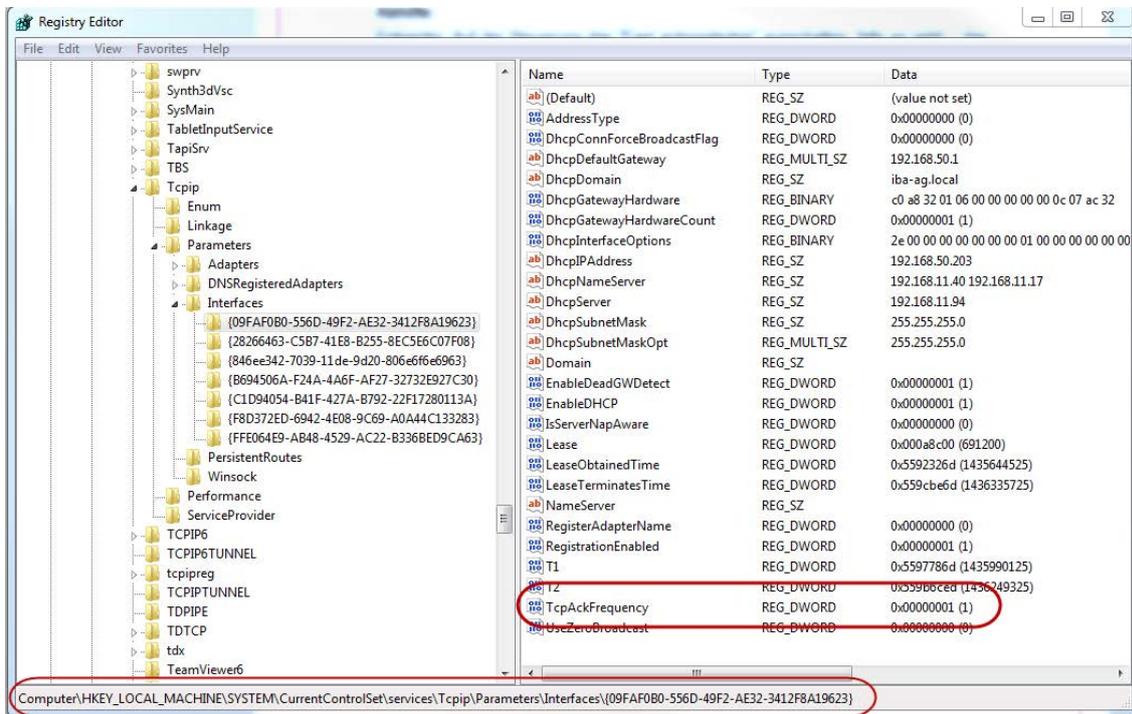
The "delayed acknowledge" can be switched off individually for each network adapter via an entry in the Windows Registry. For easy modification, *ibaPDA* offers a corresponding dialog in the I/O manager under *General* in the tab *Settings*.

In the list of network adapters, select those for which you want to disable "delayed acknowledge" and click <Apply>.



Thus, the parameter "TcpAckFrequency" (REG_DWORD = 1) is created in the registry path of the selected network adapters:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\Interfaces\
{InterfaceGUID}



Note



Basically, you can avoid such TCP-specific problems by using *UDP* instead of *TCP*.

The User Datagram Protocol (UDP) is a minimal network protocol that is not connection-oriented and is unsecured against telegram loss. Among other things, reception acknowledgement of the sent data is dispensed with. In stable and high-performance networks, however, this is not of significant importance and can be neglected due to the cyclic data transmission common with *ibaPDA*.

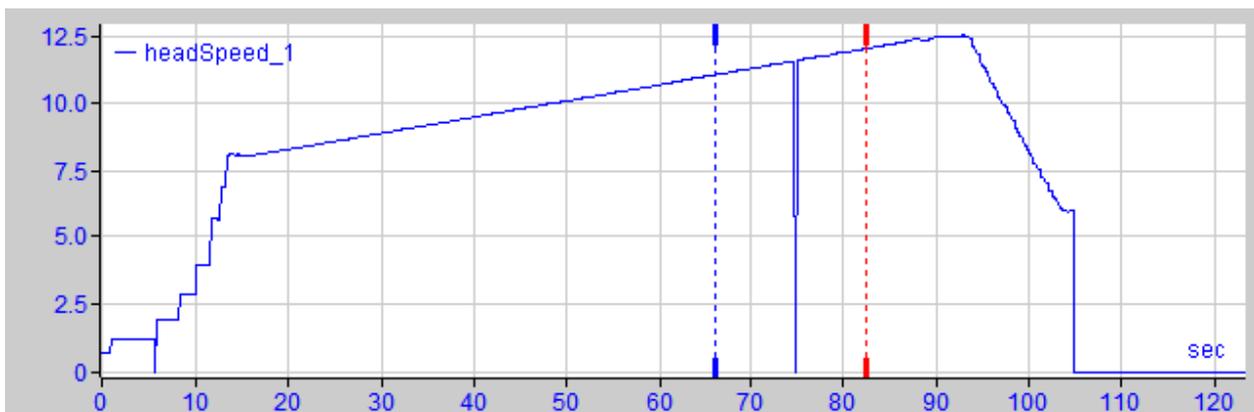
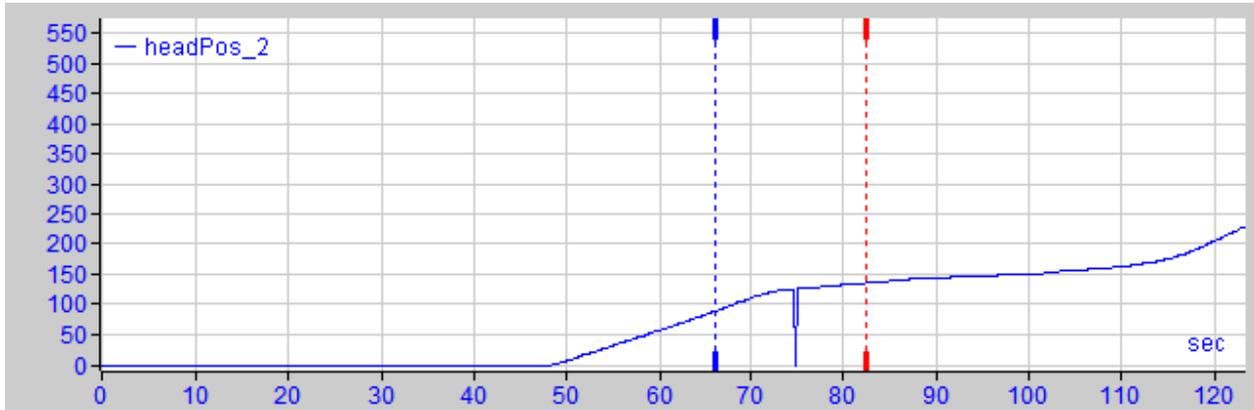
5.1.2 TCP data corruption resulting from the Nagle's Algorithm

Symptoms:

ibaPDA measurements of automation devices using TCP/IP show spikes in the data.

Errors shown in *ibaPDA*:

Incomplete telegrams and/or spikes in the data values (see examples in the following figures)



Cause:

Nagle's algorithm, named after its creator John Nagle, is one mechanism for improving TCP efficiency by reducing the number of small packets sent over the network and collecting several data blocks before sending the data over the network.

Since the Generic TCP interface does not use an application level protocol, the receiver *ibaPDA* cannot handle these merged messages correctly. The Generic TCP interface expects only 1 datagram per TCP message with always the same layout and length.

But the Nagle's Algorithm and the option *Delayed ACK* (Delayed Acknowledge, see 5.1.1, page 32) do not play well together in a TCP/IP Network:

The Delayed ACK mechanism tries to send more data per segment if it can.

But part of Nagle's algorithm depends on an ACK to send data.

Nagle's algorithm and Delayed ACKs together create a problem because Delayed ACKs are waiting around to send the ACK while "Nagle's" is waiting around to receive the ACK!

This creates random stalls of 200 ms - 500 ms on segments that could otherwise be sent immediately and delivered to the receive-side stack of *ibaPDA* as application.

Remedy:

We recommend starting with disabling the *Delayed ACK* mechanism as explained in chapter 5.1.1, page 32. In a typical real-time application, the transmitter will then send the new data to *ibaPDA* with a certain cycle time, since the previous data has been acknowledged immediately. Depending on the implementation of the TCP/IP stack on the sender's side, the Nagle's algorithm can still become active and automatically aggregate a number of small buffer messages, causing the algorithm to purposely slow down the transmission.

This can also happen sporadically due to a momentary overload on the sender side that causes the stack to merge some messages.

To disable Nagle's buffering algorithm, use the *TCP_NODELAY* socket option. The *TCP_NODELAY* socket option allows the network to bypass Nagle's-induced Delays by disabling Nagle's algorithm, and sending the data as soon as it is available.

Enabling *TCP_NODELAY* forces a socket to send the data in its buffer, whatever the packet size. The *TCP_NODELAY* flag is an option that can be enabled on a per-socket basis and is applied when a TCP socket is created.

(See *Socket.NoDelay* property in .NET applications in the *System.Net.Sockets* namespace.)

Note

Basically, you can avoid such TCP-specific problems by using *UDP* instead of *TCP*.

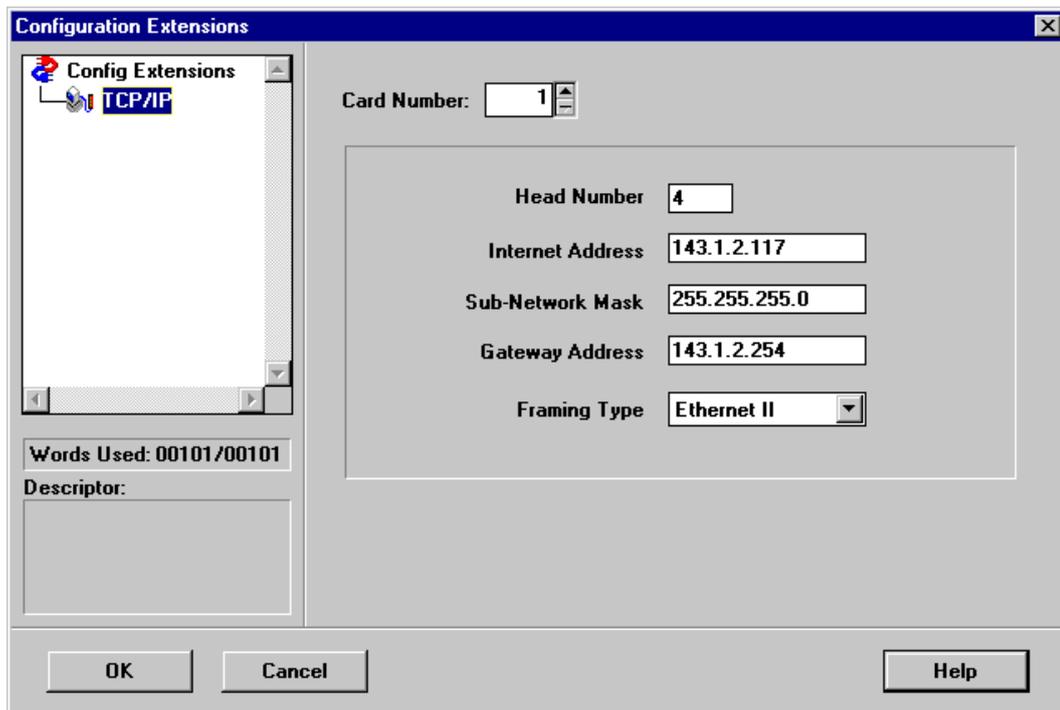
The User Datagram Protocol (UDP) is a minimal network protocol that is not connection-oriented and is unsecured against telegram loss. Among other things, reception acknowledgement of the sent data is dispensed with. In stable and high-performance networks, however, this is not of significant importance and can be neglected due to the cyclic data transmission common with *ibaPDA*.

5.2 Engineering examples

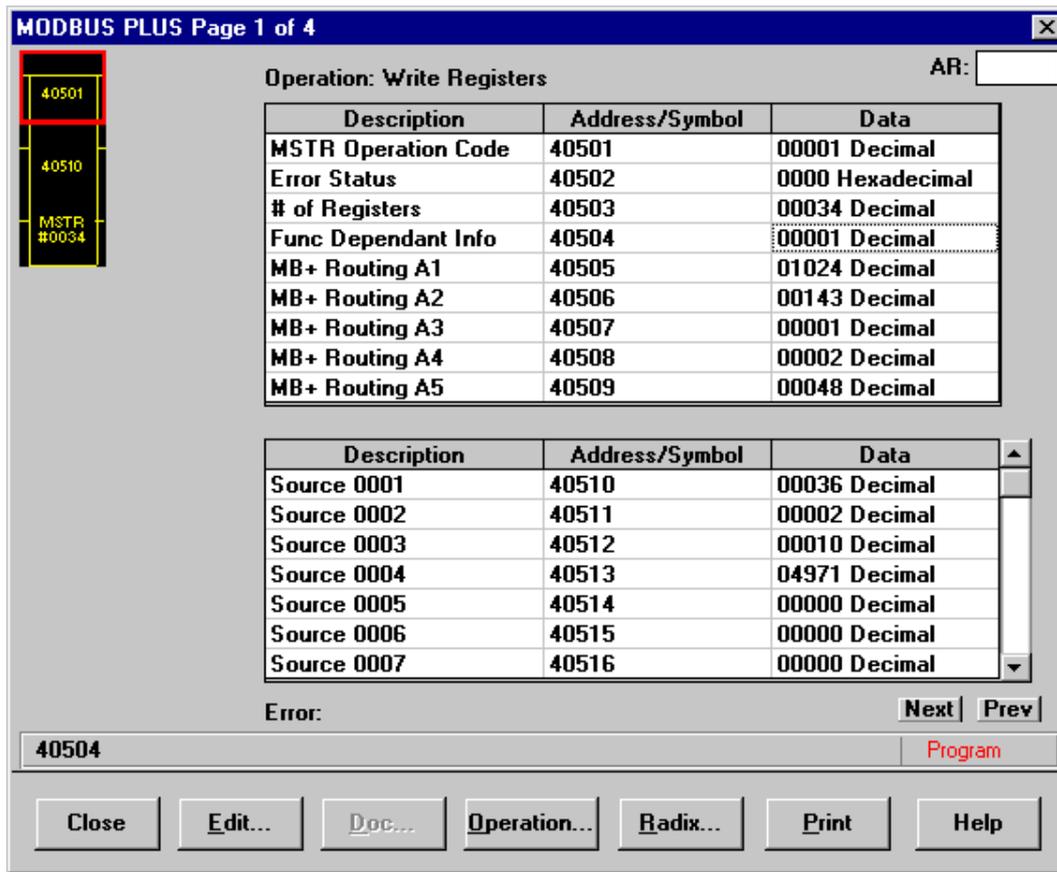
5.2.1 Engineering example Modicon Quantum

5.2.1.1 Configuration of the TCP/IP Interface in ProWORX NxT

1. Install your Ethernet Module (NOE).
2. Configure your IP address for the NOE (next figure).



3. Make a new ladder or ConCept program placing a MSTR block configured as shown in the next figure.



- MSTR operation Code:
1 Decimal; stands for a write command
- Error Status:
0 hex; stands for possible communication errors
- # of Registers:
34 Decimal; this stands for the 34 registers to be sent to *ibaPDA* (32 analog + 32 digital values). In case you use the MODBUS_FLOAT (or MODBUS Real) module type, 66 registers have to be sent. When a Modbus Generic module type is used, a maximum of 122 registers can be sent.

Note



Entering a number other than 34 or 66 (with Int, Dig512 and Float) will display errors both in *ibaPDA* and the MSTR block, without any communication at all.

- Func. Dependant Info:
1 Decimal; this stands for *ibaPDA* module number 1. Use different numbers for different *ibaPDA* module numbers. Add 100 when a MODBUS_FLOAT module is used and 200 when a MODBUS_Generic module is used.

- **MB+Routing A1:**

In the example shown here, the NOE was installed in slot number 4. The actual number entered in this field is 0x400 whereas “4” represents the slot number (Decimal = 1024). For a NOE with slot number 5, the field will be 0x500, which equates to a decimal entry of 1280. This screenshot is taken from ProWORX NxT.

- **MB+Routing A2..A5:**

Stands for IP address 143.1.2.48; this should be the IP address of *ibaPDA*.

5.2.1.2 Ladder Program for the PLC

There are two ways to program the PLC:

If *ibaPDA* doesn't send an acknowledge answer

- **Advantage:**

Faster communication, there is less throughput in the LAN

- **Disadvantage:**

Doesn't work with NOE series 700

If set too fast, sometimes it can freeze the PLC

***ibaPDA* sends an acknowledge answer**

- **Advantage:**

Support of NOE series 700

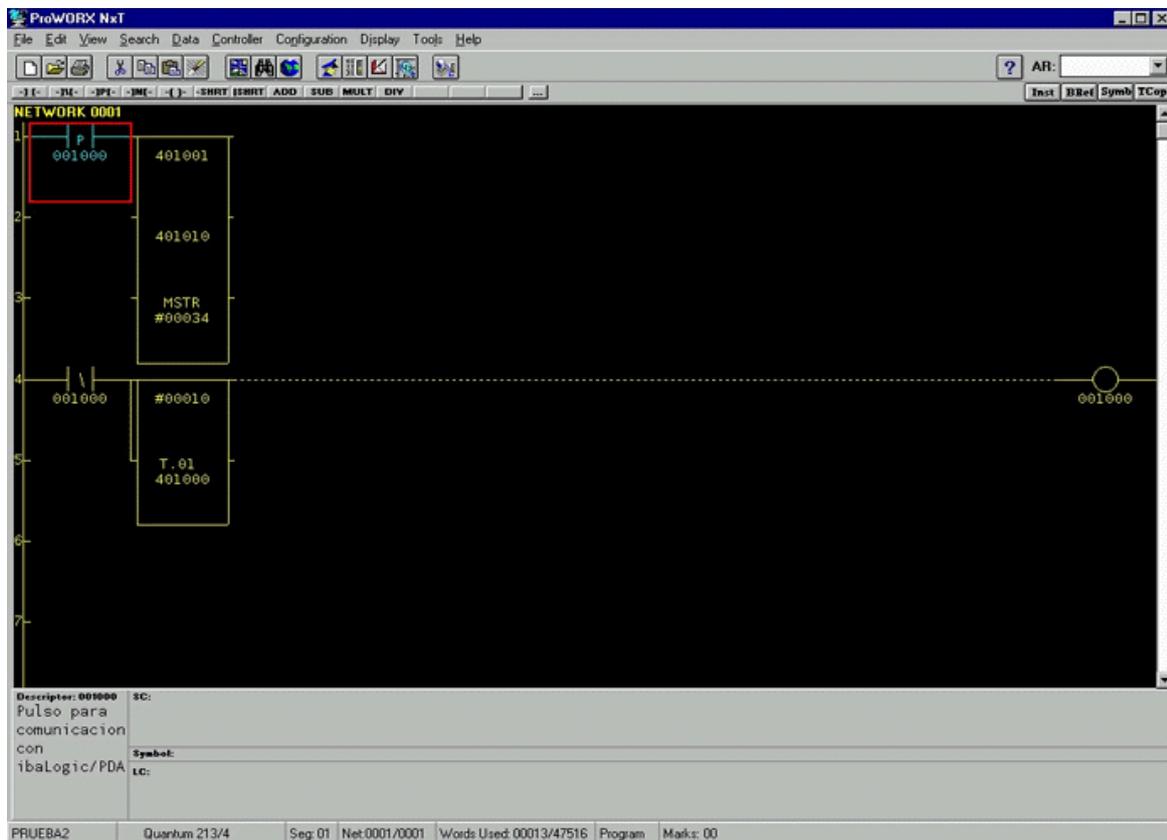
- **Disadvantage:**

Slower communication

For option 2 please select the “Send response to modbus master” option in the general settings of Modbus TCP Server, see ↗ *General interface settings*, page 16. If option 1 is preferable simply de-select it.

5.2.1.2.1 ibaPDA doesn't send an acknowledge answer (example)

Complete the program as shown in the figure below (example: Ladder program in ProWORX NxT).



T.01 is the timer generating pulses, in this case every 10 milliseconds.

Notice that the marked switch “P 001000” in the red square is the pulse output generated by T.01

Notice also that *ibaPDA* does not send a response back to the PLC so the MSTR block is sending registers every 10ms even if errors occur or there is no response. Do not expect response from the communication.

When the PLC is restarted the MSTR block should establish the communication with *ibaPDA*. You should be able to see the connection status in *ibaPDA* Diagnostics for Modbus. Look for a few minutes what happens with the connection. If the connection is held for more than approx. 2 minutes, it should be working fine.

Note



One (1) sequence error displayed is ok. This is shown as the first intent to communicate with *ibaPDA*.

Every time the MSTR block sends data, the message counter is incremented in the connection overview (I/O manager *ibaPDA*), for the available active connection.

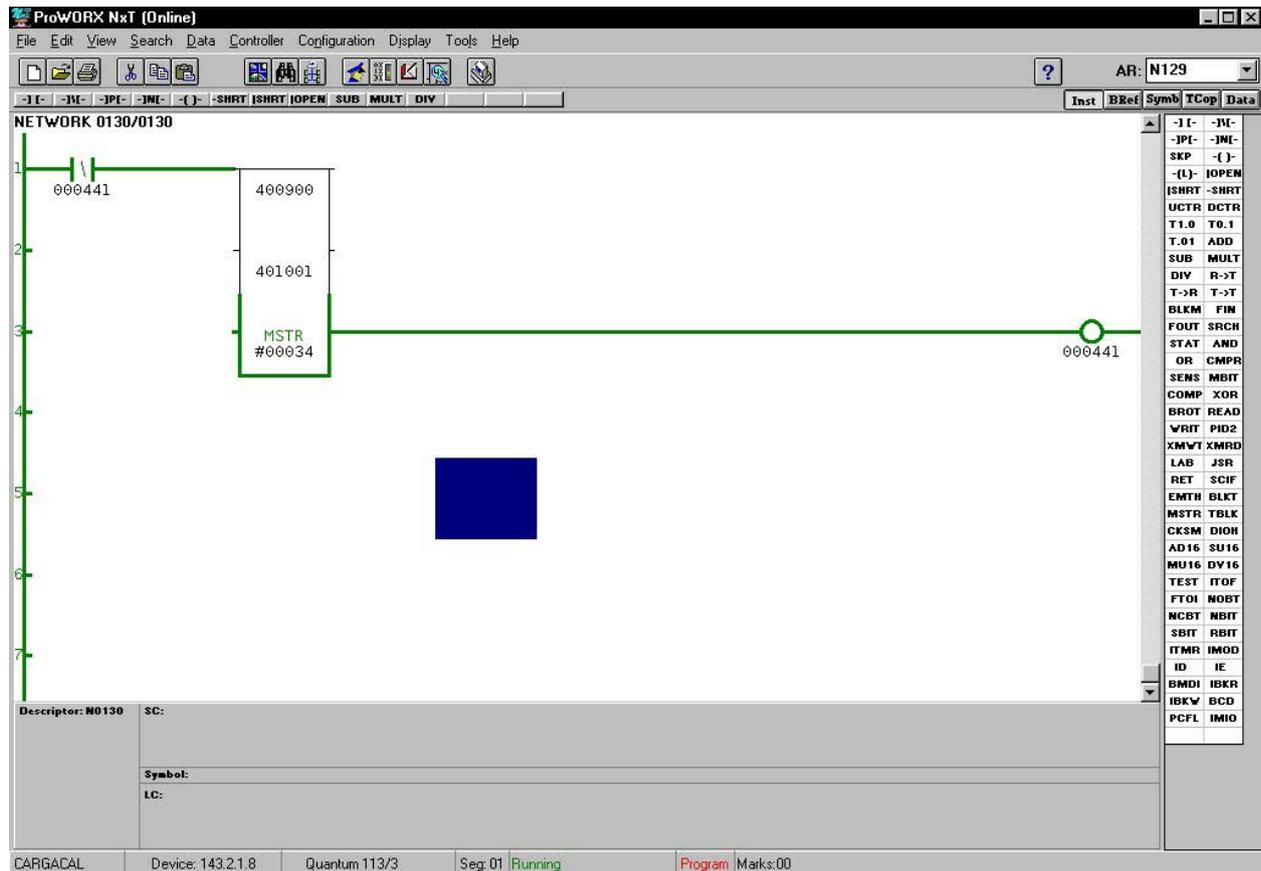
It is recommended to test live data with an up-counter which should be configured to write data into the first register sent by the MSTR block.

For more than 32 analog and 32 digital values, follow the MSTR block configuration procedure as explained but change the *Func. Dependant Info* (ibaPDA module number).

In *ibaPDA* add Modbus Server modules as required.

5.2.1.2.2 ibaPDA sends an acknowledge answer (example)

Complete the program as shown in the figure below (example: Ladder-Programm in ProWORX NxT).



Notice that this time there is no timer that enables the MSTR block to write. Instead, after each successful transaction, *ibaPDA* sends an ACK answer back which returns a SUCCESSFUL output and prepares the MSTR block to be ready for the next one.

5.2.1.3 ConCept Program for the PLC

Basically the steps of configuration in ConCept are the same as described in the preceding sections.

In ConCept, the parameter names of the MSTR block differ.

ProWORX NxT	ConCept
MSTR operation Code	w1
Error Status	w2
# of Registers	w3

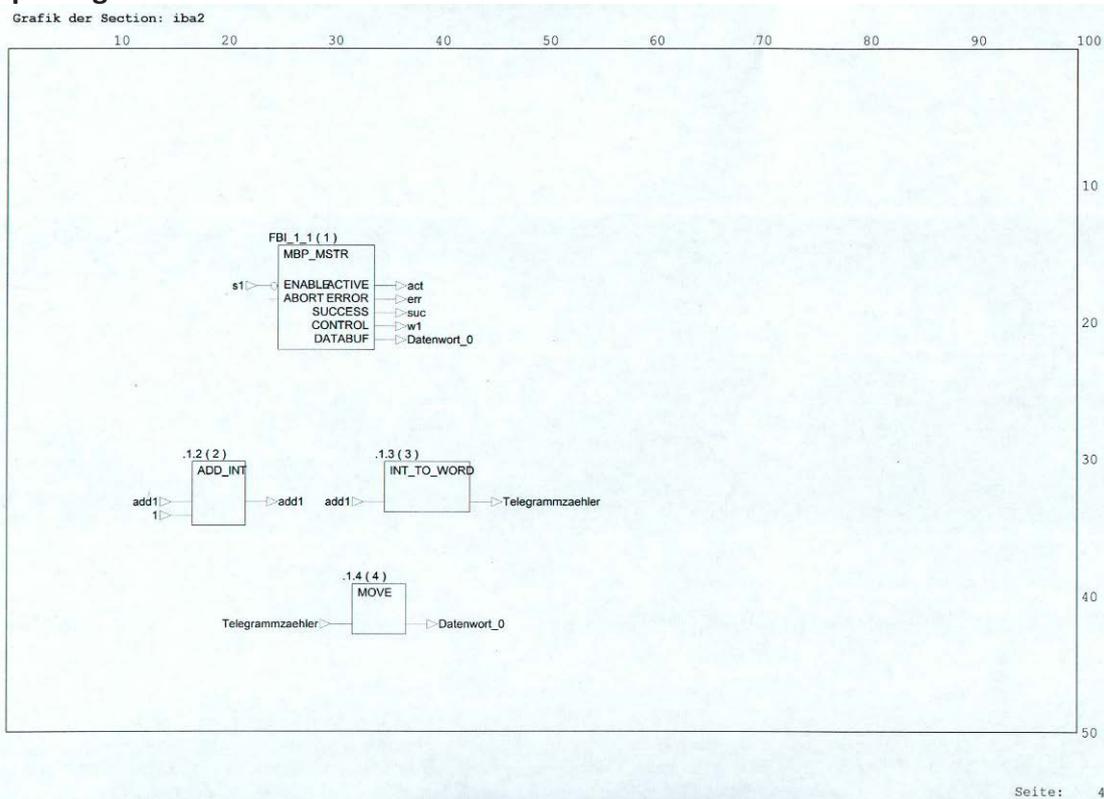
ProWORX NxT	ConCept
Func. Dependant Info	w4
MB+Routing A1	w5
MB+Routing A2	w6
MB+Routing A3	w7
MB+Routing A4	w8
MB+Routing A5	w9

Example for ConCept Configuration and Program

Variablenliste (Name: Alles, Typ: Alles, Datentyp: Alles, Sortiert nach: Name)						
Variablenname	Typ	DTyp	Adresse	Initial-Wert	Kommentar	Verw
act	VAR	BOOL				1
add1	VAR	INT				3
data1	VAR	WORD	400101			0
data2	VAR	WORD	400102			0
data3	VAR	WORD	400103			0
data4	VAR	WORD	400104			0
data5	VAR	WORD	400105			0
Datenwort_0	VAR	WORD	400106			2
err	VAR	BOOL				1
iba2	IVAR	SECT_CTRL				0
s1	VAR	BOOL				1
suc	VAR	BOOL				1
Telegrammzaehler	VAR	WORD	400100			2
w1	VAR	WORD	400001	1		1
w2	VAR	WORD	400002			0
w3	VAR	WORD	400003	10		0
w4	VAR	WORD	400004	139		0
w5	VAR	WORD	400005	0300		0
w6	VAR	WORD	400006	222		0
w7	VAR	WORD	400007	95		0
w8	VAR	WORD	400008	1		0
w9	VAR	WORD	400009	198		0

A more detailed description of the configuration and programming in ConCept is being prepared.

ConCept Program with MSTR block:

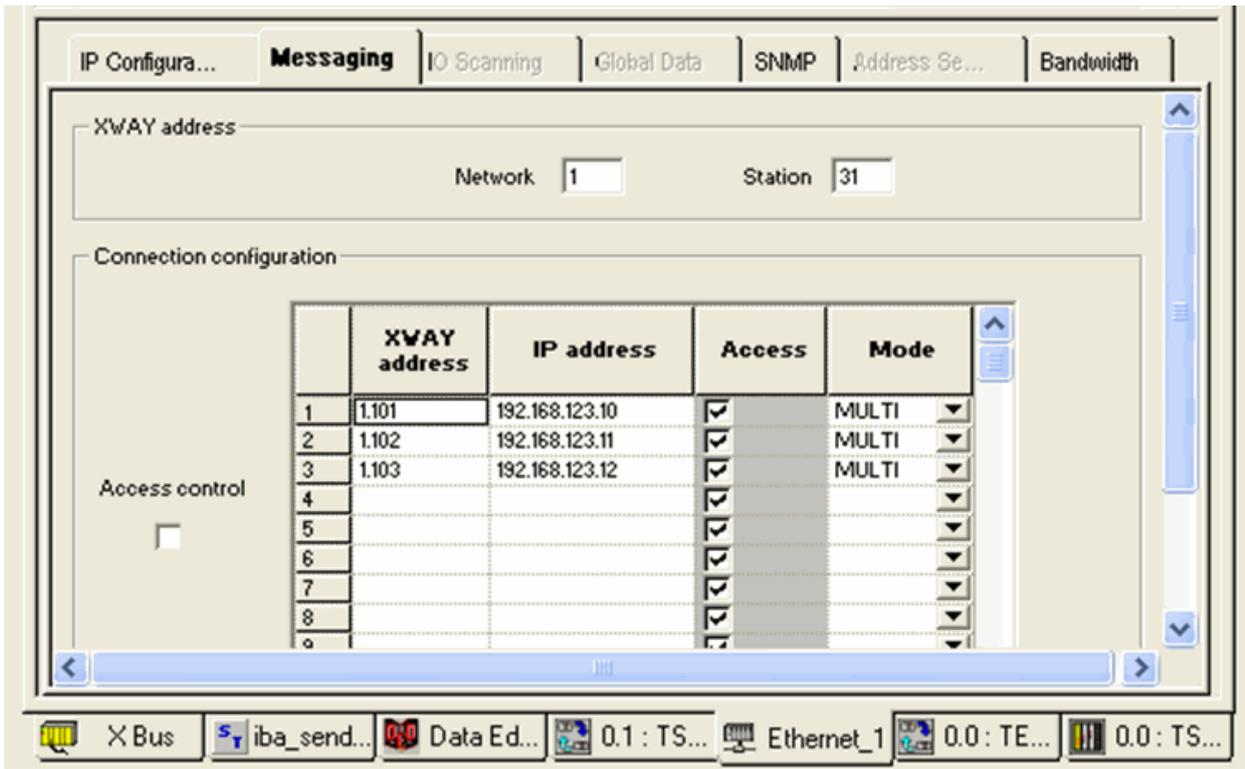


ConCept RDE table:

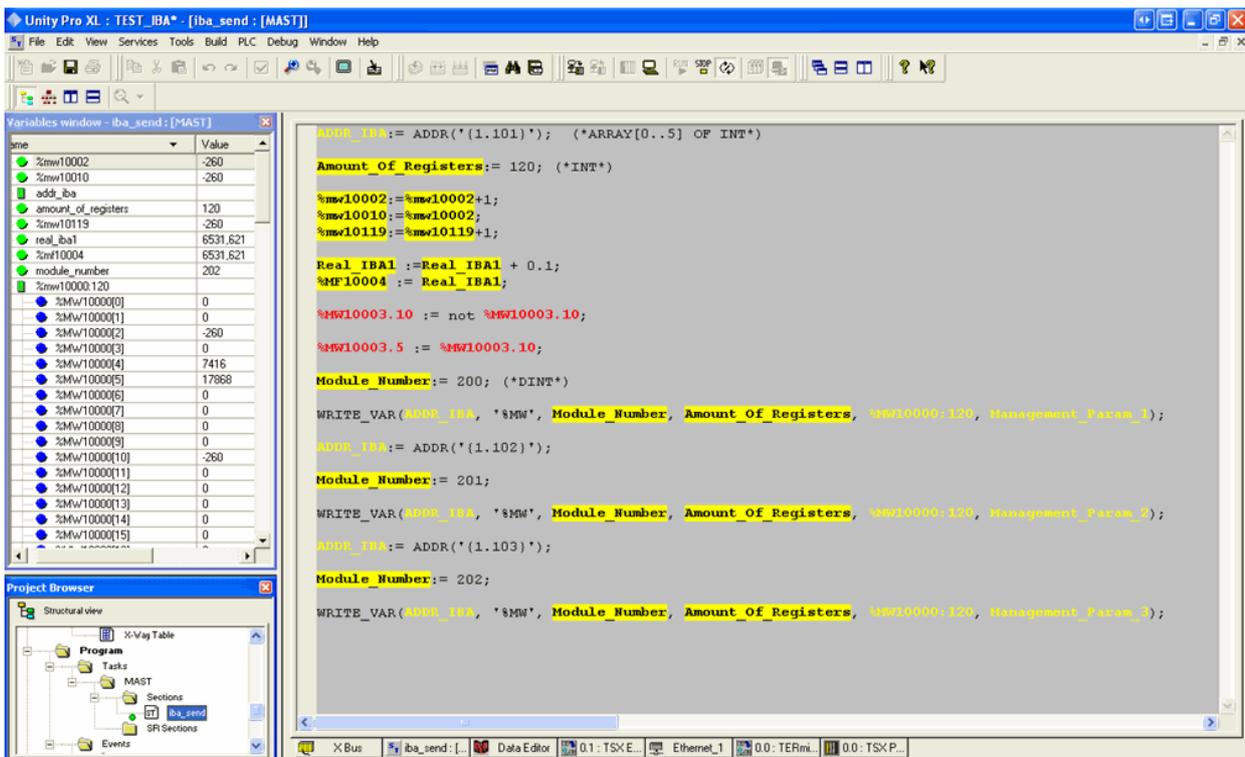
	Variablenname	Datentyp	Adresse	Wert	Wert eingeben	Format	Sperren	Zykl.Setz.	Anim.-Status
1	w5	WORD	400005	300		Hex			
2	w6	WORD	400006	222		Dez			
3	w7	WORD	400007	95		Dez			
4	w8	WORD	400008	1		Dez			
5	w9	WORD	400009	198		Dez			
6	w1	WORD	400001	1		Dez			
7	w2	WORD	400002	0		Hex			
8	w3	WORD	400003	34		Dez			
9	w4	WORD	400004	1		Dez			
10	Telegrammzaehler	WORD	400100	21615		Dez			
11	data1	WORD	400101	0		Dez			
12	data2	WORD	400102	88		Hex			
13	data3	WORD	400103	10		Dez			
14	data4	WORD	400104	100		Dez			
15	data5	WORD	400105	34		Dez			
16	Datenwort_0	WORD	400106	21418		Dez			
17			400107	22		Dez			
18			400108	14		Dez			
19									
20									
21									
22									
23									
24									
25									
26									
27									
28									

5.2.1.4 Unity Pro XL Program for the PLC (Generic module sample)

1. Install your Ethernet module.
2. Configure the IP addresses of the Ethernet module.
The Ethernet module can be found in the project browser under
Station\Communication\networks\



3. Create a new ladder (LAD) or structure text (ST) program. A new program can be added under Station\Program\Tasks\MAST\Sections (example: ST program in Unity Pro XI):



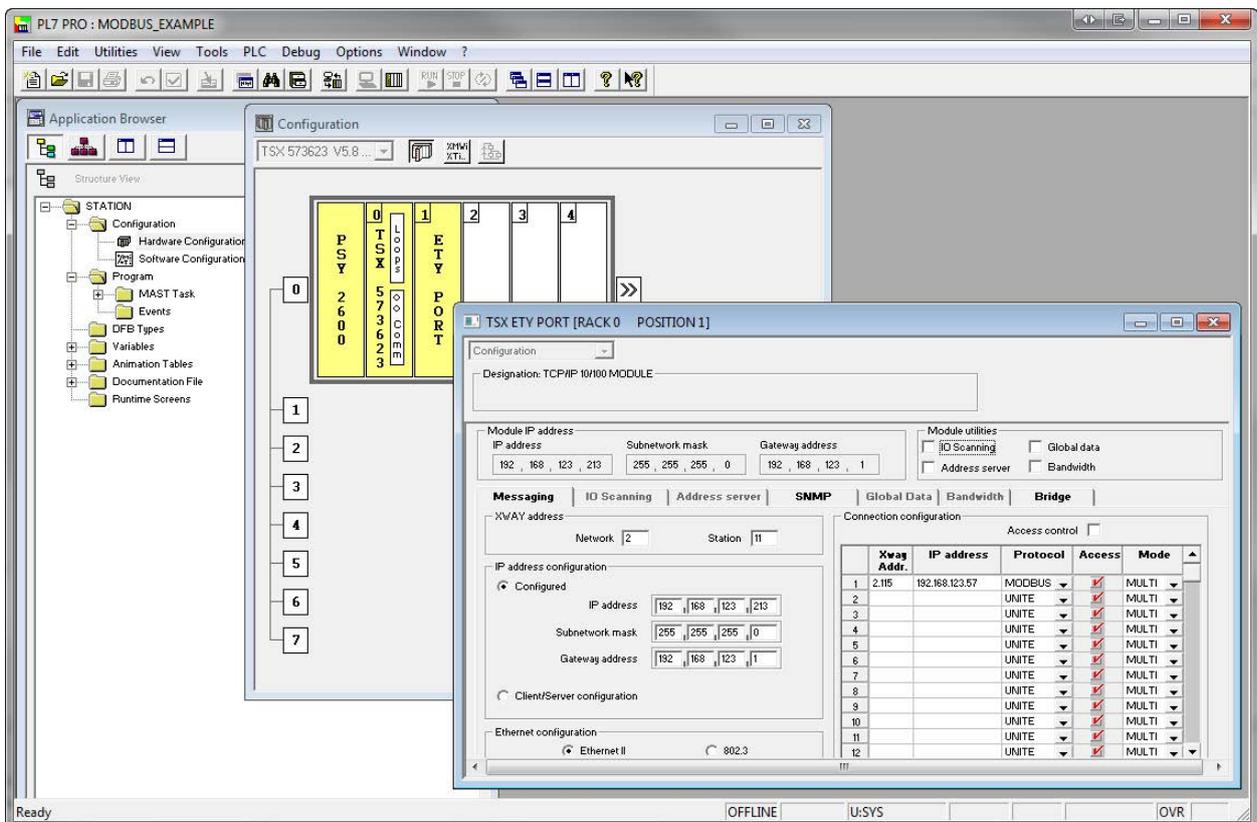
To send a message via MODBUS TCP/IP, the WRITE_VAR function is used. This function needs a few parameters like the IP address, variable type, module number, amount of registers, etc.

The IP addresses are represented by the ADDR_IBA variable. This variable is the XWAY address configured in the Ethernet module. For each module a different IP address is necessary. Thus for 3 modules, 3 addresses need to be configured.

In the example different variables were created to preserve a clear overview of the program. The module numbers are set to 200, 201 and 202 because we want to send Generic modules towards *ibaPDA*. Each module contains floats, integers and Booleans with a length of 120 registers in total.

5.2.2 Engineering example in PL7 Pro

5.2.2.1 Network configuration



In the hardware configuration of the ETY (network) card, an XWAY address needs to be created in order to be able to send the data to *ibaPDA* via the MODBUS protocol.

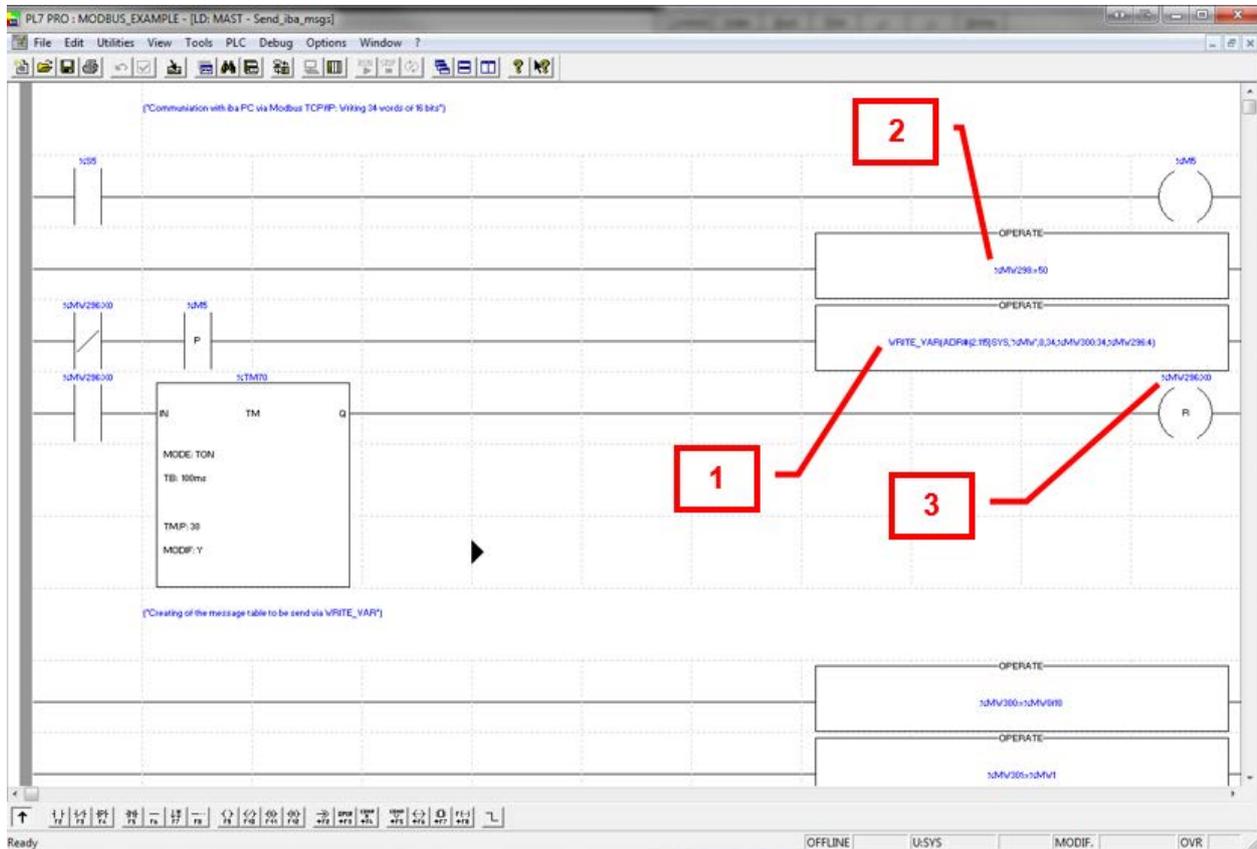
As shown in the table “Connection configuration” above (example: Network configuration in PL7 Pro), the XWAY consists of:

- XWAY Address: This address will be used in the program as “gateway”.
- IP address: Here the IP address of the *ibaPDA* data acquisition system needs to be filled in.
- Protocol: The MODBUS protocol should be selected to send messages via MODBUS.
- Access: This needs to be selected
- Mode: Multimode is required.

After finishing these settings and the IP address configuration of the card itself, you need to validate (confirm) the configuration.

5.2.2.2 Message configuration (example)

Create a program in the "Mast" task section (example: Program in PL7 Pro).



The message needs to be sent cyclically towards *ibaPDA*. Therefore, a system variable “%S5” is used. This system variable is a toggling digital signal with a 100 ms period. On the rising edge of the digital signal, the message will be sent towards *ibaPDA* (network 3).

The WRITE_VAR operation (1) is used to send the data towards *ibaPDA*. This operation uses the following settings:

```
WRITE_VAR (ADR#2.115SYS, '%MW', 0, 34, %MW300:34, %MW296:4)
```

- XWAY Addr. The previous created XWAY Addr. needs to be filled in here ADR#<XWAY Addr>SYS
- Type: Here the variable type is selected ‘%MW’
- Module index: Here the same module index as used in *ibaPDA* should be applied.
- Number of registers: 34 is the number of registers (words of 16 bit) which will be sent towards *ibaPDA*.
- Start address: The address %MW300 to %MW334 will be sent
- Management words: These words contain the status, send settings, etc.

The operation (2) where the word %MW298 is set towards 50 is used to set the timeout of the WRITE_VAR function.

Value MW269:X0 (3) shows if the variable has been sent or not.

The timer (%TM70) is used to prevent that the message doesn't take more than 1 minute (normally set to for example 100 ms) to be sent towards the *ibaPDA* system.

As a result a table will be created consisting of signals which are linked to the configured addresses and which will be copied into the signal table of *ibaPDA*. This table can be found below the timer.

6 Support and contact

Support

Phone: +49 911 97282-14
Fax: +49 911 97282-33
Email: support@iba-ag.com

Note



If you need support for software products, please state the number of the license container. For hardware products, please have the serial number of the device ready.

Contact

Headquarters

iba AG
Koenigswarterstrasse 44
90762 Fuerth
Germany

Phone: +49 911 97282-0
Fax: +49 911 97282-33
Email: iba@iba-ag.com

Mailing address

iba AG
Postbox 1828
D-90708 Fuerth, Germany

Delivery address

iba AG
Gebhardtstrasse 10
90762 Fuerth, Germany

Regional and Worldwide

For contact data of your regional iba office or representative please refer to our web site:

www.iba-ag.com