MODBUS/TCP

Modbus/TCP with Phoenix Contact controllers

Application note
8294_en_03

1 Description

This application note describes how Phoenix Contact controllers receive and process I/O signals from I/O devices using Modbus/TCP.

Startup of the Modbus/TCP client function and the Modbus/TCP server function is described using a project creation process with PC Worx as an example, see Section 5 and Section 7.

Controller	Modbus/TCP client support	Modbus/TCP server support	
		As of firmware version	As of AX Software Suite version
ILC 131 ETH	Yes	4.40	1.82 AddOn V1
ILC 151 ETH	Yes	4.40	1.82 AddOn V1
ILC 171 ETH 2TX	Yes	4.40	1.82 AddOn V1
ILC 191 ETH 2TX	Yes	4.40	1.82 AddOn V1
ILC 131 ETH/XC	Yes	4.40	1.82 AddOn V1
ILC 151 ETH/XC	Yes	4.40	1.82 AddOn V1
ILC 191 ME/AN	Yes	4.40	1.82 AddOn V1
ILC 191 ME/INC	Yes	4.40	1.82 AddOn V1
ILC 151 GSM/GPRS	Yes	4.40	1.82 AddOn V1
AXC 1050	Yes	–	–
AXC 1050 XC	Yes	–	–
AXC 3050	Yes	–	–
PC WORX RT BASIC	Yes	–	–
PC WORX SRT	Yes	–	–

Make sure you always use the latest documentation.
It can be downloaded at phoenixcontact.net/products.

This application note is valid for all products listed under “Ordering data” on page 4.

2 Table of contents

3 Ordering data

Products

Description	Type	Order No.	Pcs./Pkt.
Inline controller with Ethernet interface for coupling to other controllers and systems, with programming options according to IEC 61131-3, complete with connector plug and marking field.	ILC 131 ETH	2700973	1
Inline controller with Ethernet interface for coupling to other controllers and systems, with programming options according to IEC 61131-3, complete with connector plug and marking field.	ILC 151 ETH	2700974	1
Inline controller with Ethernet interface for coupling to other controllers and systems, with programming options according to IEC 61131-3, complete with connector plug and marking field.	ILC 171 ETH 2TX	2700975	1
Inline controller with Ethernet interface for coupling to other controllers and systems, with programming options according to IEC 61131-3, complete with connector plug and marking field.	ILC 191 ETH 2TX	2700976	1
Inline controller with Ethernet interface for coupling to other controllers and systems, with programming options according to IEC 61131-3, complete with connector plug and marking field.	ILC 131 ETH/XC	2701034	1
Inline controller with Ethernet interface for coupling to other controllers and systems, with programming options according to IEC 61131-3, complete with connector plug and marking field.	ILC 151 ETH/XC	2701141	1
Inline controller with integrated pulse/direction interface, RS-485/RS-422, analog inputs (0 ... 10 V), and analog outputs (0 ... 10 V), with programming options according to IEC 61131-3	ILC 191 ME/AN	2700074	1
Inline controller with integrated pulse/direction interface, RS-485/RS-422, fast counters, and incremental encoder inputs, with programming options according to IEC 61131-3	ILC 191 ME/INC	2700075	1
Inline controller with Ethernet interface and GSM modem for coupling to other controllers and systems, with programming options according to IEC 61131-3, complete with connector plug and marking field	ILC 151 GSM/GPRS	2700977	1
Axiocontrol for the direct control of Axioline I/Os. With 2 Ethernet interfaces and programming options according to IEC 61131-3. Complete with connector plug and marking field.	AXC 1050	2700988	1
Axiocontrol for the direct control of Axioline I/Os. With 2 Ethernet interfaces, extended temperature range, and programming options according to IEC 61131-3. Complete with connector plug and marking field.	AXC 1050/XC	2701295	1
Axiocontrol for the direct control of Axioline I/Os. With 3 Ethernet interfaces and programming options according to IEC 61131-3. Complete with connector plug and marking field.	AXC 3050	2700989	1
Software PLC	PC WORX RT BASIC	2700291	1
Software PLC	PC WORX SRT	2701680	1

Documentation

Description	Type	Order No.	Pcs./Pkt.
“Installing and operating the ILC 131 ETH, ILC 151 ETH, ILC 171 ETH 2TX, ILC 191 ETH 2TX, ILC 131 ETH/XC, and ILC 151 ETH/XC Inline controllers” user manual	UM EN ILC 1X1	–	–
“Installing and operating the ILC 191 ME/AN and ILC 191 ME/INC Inline controllers” user manual	UM EN ILC 191 ME/X	–	–
“Installing, starting up, and operating the ILC 151 GSM/GPRS Inline controller” user manual	UM EN ILC 151 GSM/GPRS	–	–
“Installing and operating the AXC 1050 and AXC 1050 XC controllers” user manual	UM EN AXC 1050 (XC)	–	–
“Installing and operating the AXC 3050 controller” user manual	UM EN AXC 3050 (/XC)	–	–
“Installing and operating the PC WORX RT BASIC software PLC with Valueline industrial PCs and PROFINET” user manual	UM EN PC WORX RT BASIC	–	–
“Installing and operating the PC WORX SRT software PLC” user manual	UM EN PC WORX SRT	–	–
“PC Worx” quick start guide	UM QS EN PC WORX	–	–

4 Modbus/TCP

Modbus/TCP is a communication protocol used to exchange process data between a client and a server in an Ethernet network.

A Modbus/TCP client function is therefore integrated in Phoenix Contact controllers (see “Ordering data” on page 4). The I/O devices must include a Modbus/TCP server function.

As of firmware Version 4.40, ILC 1X1 controllers can also be configured as Modbus/TCP servers (see Section 7).

The TCP protocol (**T**ransmission **C**ontrol **P**rotocol) or the UDP protocol (**U**ser **D**atagram **P**rotocol) can be used for data transmission. The Modbus protocol data to be transmitted is embedded in the relevant protocol. The controller supports two methods of communication:

– Cyclic communication

  Process data between the Modbus/TCP client and the Modbus/TCP server is exchanged cyclically. This is triggered by the corresponding task in which the process data is defined.
  The Modbus/TCP client can maintain process data watchdogs of Modbus/TCP servers in a time period that can be defined. To do so, the Modbus/TCP client repeats all the write access operations to the relevant Modbus/TCP server using the current data.
  During cyclic communication, the established TCP/IP connection between the client and server remains permanently active.

– Acyclic communication

  Process data between the Modbus/TCP client and the Modbus/TCP server is exchanged as required. The “MB_ASYNC_RW” function block, which is used on the client side, is available in PC Worx for this (see Section 5.2).
  The TCP/IP connection is established, it is disconnected once the data has been transmitted, and is then re-established in the event of a new communication request. Connection establishment and connection abort are initiated automatically by the Modbus/TCP client.

By default, port 502, which is reserved for Modbus/TCP, is used for communication.

The Modbus/TCP client initiates communication between the Modbus/TCP client and server. The client sends a request in the form of a Modbus/TCP function code (and data, if necessary) to the Modbus/TCP server. After successful receipt of the request, the server sends a corresponding response to the client which includes the requested data and status information or an error message. The data may contain bit or word information.

The device-internal organization of the data (Modbus memory addresses, etc.) and of the supported Modbus function codes varies depending on the device and manufacturer. For more information, please refer to the documentation for the relevant device.

For communication between the client and server, Modbus/TCP provides various services for read and write access to digital inputs and outputs and to registers.
The following table shows the supported Modbus function codes:

Modbus function codes			
Code No.	**Function code**	**Description**	**Method**
FC01	Read Coils	Read several internal bits or digital outputs	Bit-by-bit
FC02	Read Discrete Inputs	Read several digital inputs	Bit-by-bit
FC03	Read Holding Registers	Read several internal registers or output registers	Word-by-word
FC04	Read Input Registers	Read several input registers	Word-by-word
FC05[1]	Write Single Coil	Write one internal bit or digital output	Bit-by-bit
FC06[1]	Write Single Register	Write one internal register	Word-by-word
FC15	Write Multiple Coils	Write several internal bits or digital outputs	Bit-by-bit
FC16	Write Multiple Registers	Write several internal registers or output registers	Word-by-word
FC23	Read/Write Multiple Registers	Read and write several internal registers or output registers simultaneously	Word-by-word

[1] In the PC Worx bus configuration, only the Modbus/TCP server supports the Modbus function code.

5 Configuring a controller as a Modbus/TCP client – example project

The following example project consists of the ILC 151 ETH (controller) and the ILB ETH 24 DI16 DIO16-2TX (Inline Block IO module for Ethernet with 16 digital inputs and 16 digital inputs or outputs).

A Modbus/TCP network is used exclusively in the example project.
For information on using a gateway to connect a Modbus/TCP network to Modbus/RTU devices, please refer to Section 6.

5.1 Cyclic communication: sequence for creating the Modbus project

The complete sequence for creating the Modbus project in PC Worx is shown in Figure 1.

For more detailed information on creating a project, please refer to the UM QS EN PC WORX quick start guide or the PC Worx online help.

When creating the PC Worx project, most of the tasks are performed offline (without a connection to the Modbus system).

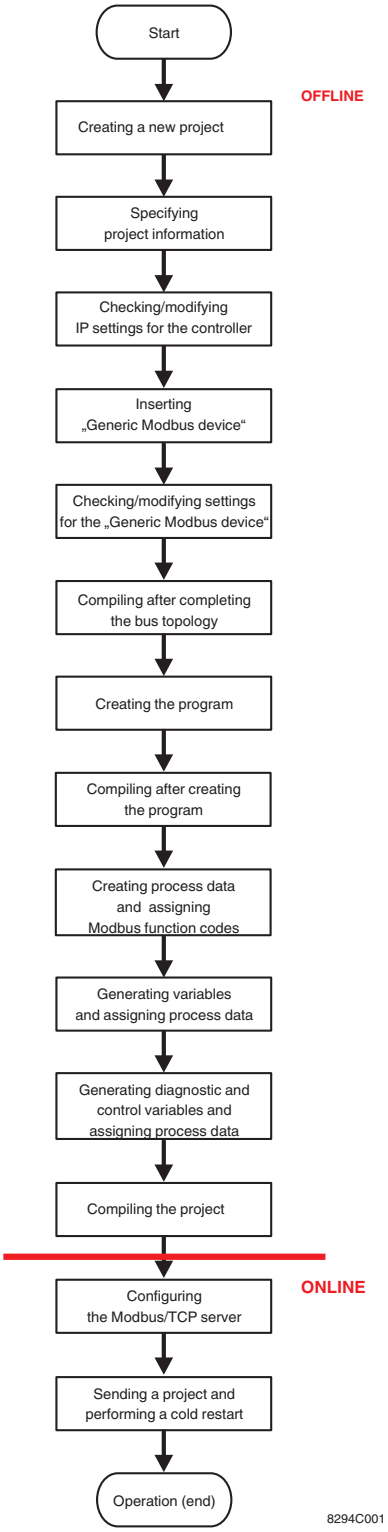


Figure 1 Sequence for creating the Modbus project

The sequence described here for creating a Modbus project applies to projects based on cyclic communication between the controller (here: Modbus/TCP client) and any Modbus/TCP server.
Acyclic communication is explained in Section 5.2 “Acyclic communication using the MB_ASYNC_RW function block” .

5.1.1 Creating a new project

- Select the “New Project...” command from the “File” menu.
- Select the controller and confirm your selection with “OK”.
- Select the “File, Save Project As/Zip Project As...” command.
- Enter a unique and meaningful project name and save the project.

5.1.2 Specifying project information

- Switch to the bus configuration workspace.
- Adapt the project information to your project.

5.1.3 Checking/modifying IP settings for the controller

The IP settings for the controller are made when the project is created.

Adapt these settings, if required.

- Switch to the bus configuration workspace.
- Select the controller node.
- In the “Device Details” window, switch to the “IP Settings” tab.
- Check the IP settings and modify them, if necessary.
- Assign an IP address, if it has not yet been assigned. For detailed information on assigning the IP address, please refer to the UM QS EN PC WORX quick start guide.

The IP address that is assigned here for the controller is also implemented as the IP address for the communication path via TCP/IP.

5.1.4 Inserting a “Generic Modbus Device”

In the example, the ILB ETH 24 DI16 DIO16-2TX Inline Block IO module for Ethernet is used as a Modbus/TCP server.

Specific settings are required for communication between the Modbus/TCP client and server. These settings are made in PC Worx via the “Generic Modbus Device”.
The “Generic Modbus Device” must be inserted in the bus configuration so as to enable communication between the Modbus/TCP client and server.

- Make sure you are in the bus configuration workspace.
- If the device catalog is hidden, show it by selecting the “View, Device Catalog” menu.
- Open the “Phoenix Contact, Generic, Device” device catalog.
- Select the “Generic Modbus Device”.

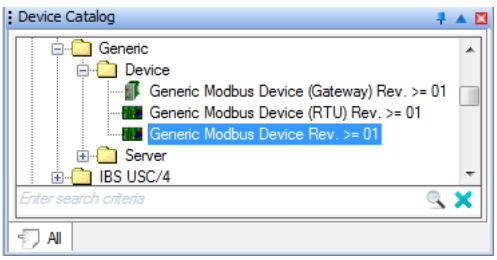


Figure 2 Selecting the “Generic Modbus Device”

- Hold down the left mouse button and move the “Generic Modbus Device” to the “Bus Structure” window to the right of the MODBUS_CLT icon[1] until the “Insert in the lower level” icon appears.
- Move all other “Generic Modbus Devices” to below the preceding “Generic Modbus Device” until the “Insert at the same level” icon appears.

Figure 3 shows the bus configuration with the inserted “Generic Modbus Device”.

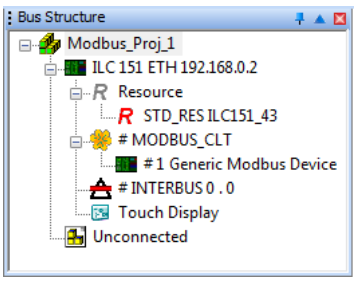


Figure 3 Generic Modbus Device inserted

[1] If a 1X1 controller with firmware Version 4.40 or later is used, the MODBUS icon is displayed instead of the MODBUS_CLT icon.

"Hide Device" or "Deactivate Bus" can be selected via the context menu for the device.
To show the device again or to activate the bus, select the aforementioned settings from the context menu again.

5.1.5 Modifying the settings for the "Generic Modbus Device"

Following insertion in the bus configuration, default values are set for each "Generic Modbus Device". The settings can be modified via the "Modbus-Settings" tab.

The settings for the "Generic Modbus Device" are required for communication between the Modbus/TCP client and server.

- Make sure you are in the bus configuration workspace.
- Select the "Generic Modbus Device" in the "Bus Structure" window.
- Select the "Modbus-Settings" tab in the "Device Details" window.
- Modify the Modbus settings according to your requirements.

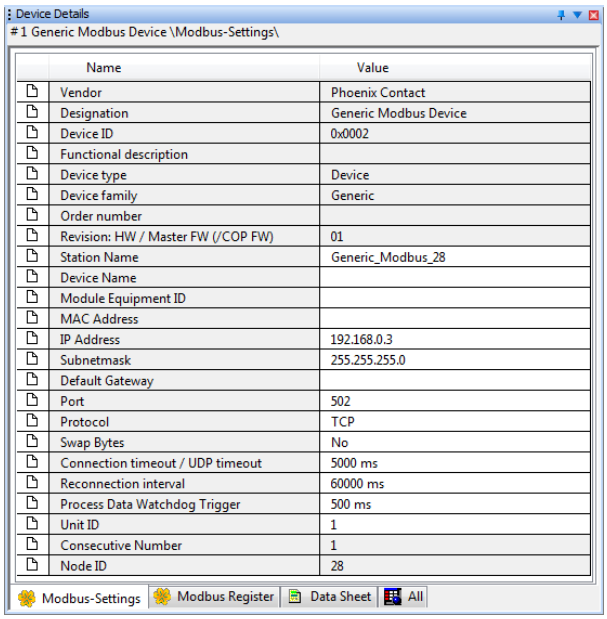


Figure 4 Modbus settings for the "Generic Modbus Device"

The settings required for communication between the Modbus/TCP client and server include:

Station Name

This name is the unique identification for the Modbus/TCP server (here: ILB ETH 24 DI16 DIO16-2TX) in the network.

MAC Address

Enter the MAC address of the Modbus/TCP server (here: ILB ETH 24 DI16 DIO16-2TX). It is printed on the respective device. It starts with "00.a0.45." on Phoenix Contact devices.
Entry of the MAC address is optional and is not absolutely necessary.

IP Address

The IP address allows the Modbus/TCP server (here: ILB ETH 24 DI16 DIO16-2TX) to be accessed during operation. PC Worx selects the address from the area that is set on the project node.

Subnetmask

The subnet mask that was specified on the project node is assigned to each Modbus/TCP server (here: ILB ETH 24 DI16 DIO16-2TX). It can be modified specifically for each individual device.

Protocol

Select the protocol (TCP or UDP) to be used for communication here.

Port

Select the port for communication here. By default, port 502, which is reserved for Modbus/TCP, is used.

Swap Bytes

If required, the byte order for the entire communication with the Modbus/TCP server can be swapped here, provided that the Modbus/TCP server and the Modbus/TCP client (here: ILC 151 ETH) operate using a different byte order. Select "Yes" to reverse the byte order used on the client side.
For information on the byte order, refer to Section 9.

Connection timeout / UDP timeout

This value specifies the minimum time required to identify a connection interruption.
When using UDP, it is not possible to determine whether a connection interruption has occurred. Therefore, the last package sent after UDP timeout is sent again.
In addition, the value specifies the time within which a Modbus/TCP server needs to have responded to a Modbus/TCP client request.

Reconnection interval

When the TCP connection is interrupted (connection timeout) and the set time interval has elapsed, an attempt is made to establish a new TCP connection.

This setting is not relevant for communication via UDP.

Process Data Watchdog Trigger

The process data watchdog trigger specifies a time period during which the Modbus/TCP server must be accessed again (write access) at the latest.
The time set here should be half the process data watchdog time set on the Modbus/TCP server.
The setting “0 ms” means that the watchdog is switched off.

Unit ID

Address of the device in the Modbus/RTU network below the gateway.
Default value: 255 (operation without gateway)

5.1.6 Compiling after completing the bus topology

- Select the “Build, Make” command.

5.1.7 Creating the program

- Create the program.

To program the example program, proceed as described in the UM QS EN PC WORX quick start guide.

5.1.8 Compiling after creating the program

- Select the “Build, Make” command.

5.1.9 Creating process data and assigning Modbus function codes

For the Modbus/TCP client (here: ILC 151 ETH), you must specify which Modbus function codes of the Modbus/TCP client should be used to access specific register addresses of the Modbus/TCP server (here: ILB ETH 24 DI16 DIO16-2TX). To do this, define specific process data and assign the data to the corresponding Modbus function codes.

- Make sure you are in the bus configuration workspace.
- Select the Generic Modbus Device in the “Bus Structure” window.
- Select the “Modbus Register” tab in the “Device Details” window.
- Right-click to open the context menu for the “Name” field.
- Select the “Add” menu item.
- Specify a unique and meaningful name for the process data item in the “Name” field.
- Select the desired “Function Code”, see Table “Modbus function codes” on page 5.
- Select the desired data type.
- Enter the number of bits or registers to be read or written.
- For the Modbus/TCP server, enter the memory area of the process data item as the “Address” for which the selected function code should be used.
  For the memory area of the process data item, please refer to the documentation for the device being used as the Modbus/TCP server.
  In the example, a 16-bit word should be read from an internal register. The Modbus function code FC03 is used here. The value “1” is set as the address, since the memory area for the process data item used to read an internal register word-by-word for the
  ILB ETH 24 DI16 DIO16-2TX has the value “1” (see terminal-specific data sheet).
- The “Data Direction” indicates whether the function accesses a digital input/an internal register or a digital output/an output register. The data direction depends on the selected function code and cannot be modified manually.

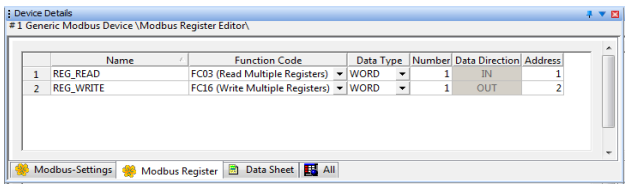


Figure 5 Creating process data and assigning Modbus function codes

In the “Modbus Register”, the function code FC23 is displayed in two rows (one row for the IN data direction and one row for the OUT data direction). When one row is deleted, the second row will also be removed automatically.

NOTE:

When linking a process variable of Array [] of (D)WORD data type, the byte order may be reversed. For further information, please refer to Section 9.

5.1.10 Generating variables and assigning process data

Process data and variables are assigned in the process data assignment workspace.

- Switch to the process data assignment workspace to assign the variables to the process data.

- Select the “Generic Modbus Device” in the top right window. The standard configuration is then displayed in the top left window (“Symbols/Variables”).
- In the top left window (“Symbols/Variables”), select the standard resource.
- In the top right window, select the device for which you would like to link the process data to variables (in Figure 6: “Generic Modbus Device”; the ILB ETH 24 DI16 DIO16-2TX module is used in the example).
- Select the process data item to be linked.
- Variables are created when the program is created. Using drag and drop, link the selected variable to one of the variables displayed on the left-hand side.
  If you would like to link further process data but no corresponding variables have been created yet, select “Create Variable” in the context menu.

The created variable is displayed in the bottom left window.

- Repeat this procedure for all inputs to be evaluated and for all outputs to be controlled.

The result of process data assignment is shown in Figure 6.

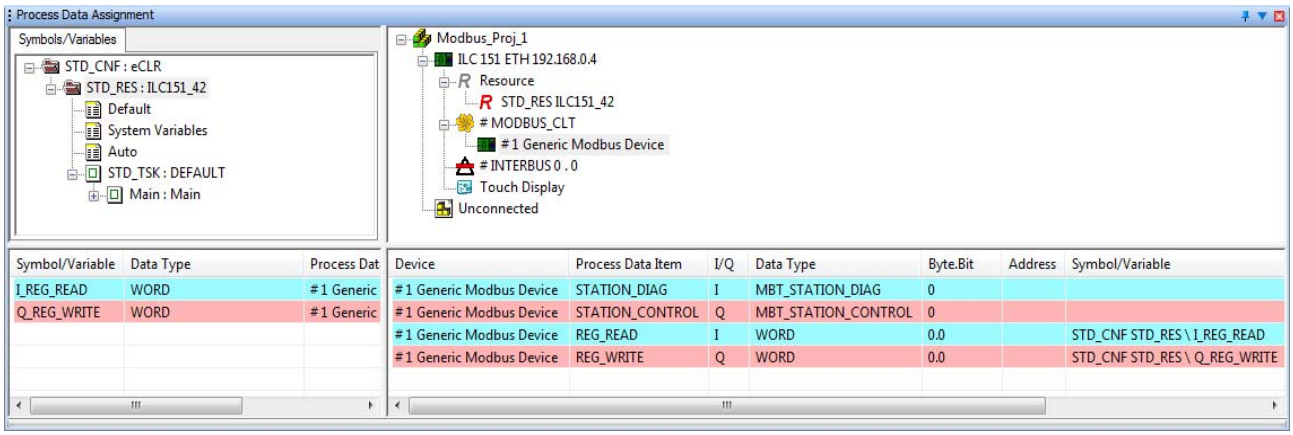


Figure 6 All used process data assigned to variables

5.1.11 Generating diagnostic and control variables and assigning process data

In the IEC programming workspace, the diagnostic and control structure is declared in the project tree window under “Data Types, sys_flag_types”.

PC Worx provides a diagnostic and control structure for each “Generic Modbus Device”, allowing the connection status, connection statistics, and connection interruptions to be read.

PC Worx provides the following diagnostic and control structure:

– “STATION_DIAG” diagnostic structure

Parameter	Description
Status	Status of the communication connection to the Modbus/TCP server (bit 0 = 1: communication connection is OK, bit 1 = 1: an exception has occurred)
IP	IP address of the Modbus/TCP server in the bus configuration
OfflineCounter	Number of times the Modbus/TCP server was offline
NetCycleAvg	Average response time of the Modbus/TCP server
NetCycleMin	Minimum response time of the Modbus/TCP server
NetCycleMax	Maximum response time of the Modbus/TCP server
NetInCycleAvg	Average response time of the Modbus/TCP server in the case of read access
NetInCycleMin	Minimum response time of the Modbus/TCP server in the case of read access
NetInCycleMax	Maximum response time of the Modbus/TCP server in the case of read access
NetOutCycleAvg	Average response time of the Modbus/TCP server in the case of write access
NetOutCycleMin	Minimum response time of the Modbus/TCP server in the case of write access
NetOutCycleMax	Maximum response time of the Modbus/TCP server in the case of write access
InternalError1	Used internally by Phoenix Contact
InternalError2	Used internally by Phoenix Contact
ExceptionCounter	Number of exception codes received (Modbus exception codes)
DiagReserved1	Reserved
DiagReserved2	Reserved
UnitID	Address of the device in the Modbus/RTU network below the gateway. Default value: 255 (operation without gateway)
DiagReserved3	Reserved

– “STATION_CONTROL” control structure

Parameter	Description
ctrlStatistics	All parameters of the diagnostic structure are reset with a rising edge at bit 0.
ControlReserved1	Reserved
ControlReserved2	Reserved
ControlReserved3	Reserved

In order to use the diagnostic and control structure, create a diagnostic variable and a control variable.

- Switch to the IEC programming workspace.
- Double-click on “Global_Variables” in the project tree window.

The global variables of the standard resource are displayed.

- Insert a new variable via the context menu which should be used as the control variable.
- Select the MBT_STATION_CONTROL type for the control variable.
- Insert another new variable via the context menu which should be used as the diagnostic variable.
- Select the MBT_STATION_DIAG type for the diagnostic variable.

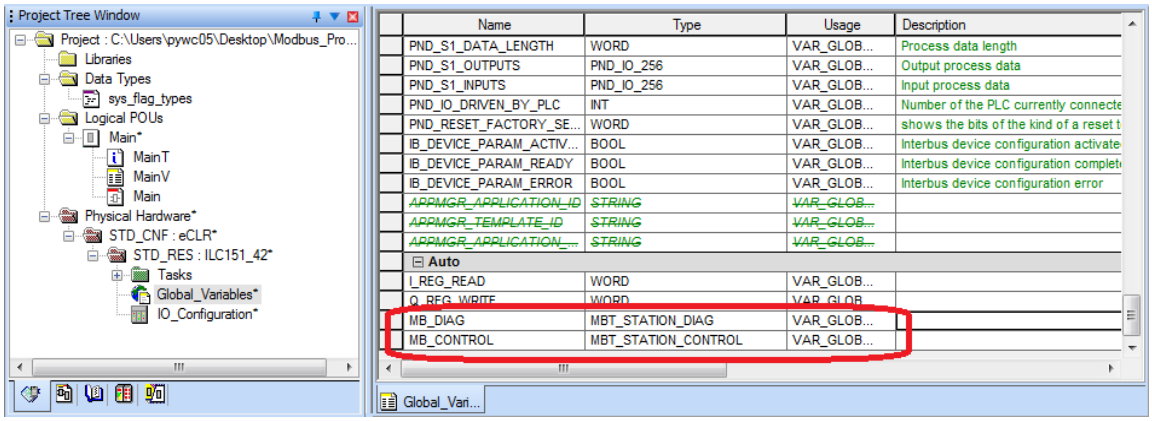


Figure 7 Diagnostic and control variables generated

- Switch to the process data assignment workspace to assign the process data to the control and diagnostic variables, as described in Section 5.1.10 on page 9.

The result of process data assignment is shown in Figure 8.

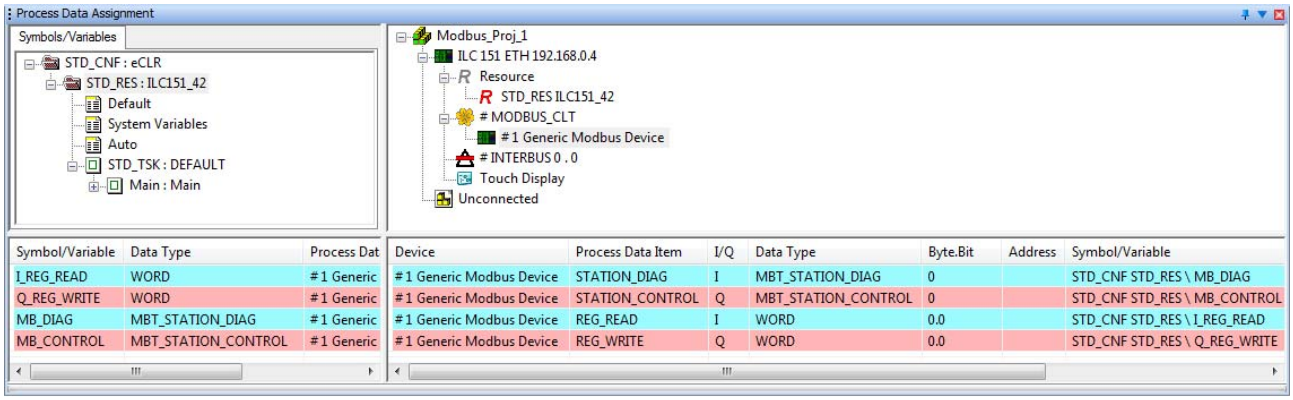


Figure 8 Process data assigned to the diagnostic and control variables

5.1.12 Compiling a project

- Select the “Build, Make” command.

5.1.13 Configuring the Modbus/TCP server

- Perform all of the required configurations (e.g., setting the IP address and process data watchdog) directly on the Modbus/TCP server (here: ILB ETH 24 DI16 DIO16-2TX).

For information on how to configure and start up the device, please refer to the corresponding documentation.

5.1.14 Sending a project and performing a cold restart

- Open the “Project Control Dialog”.
- Activate the “Include Bootproject” check box in the “Project” area.
- Click on “Download” in the area on the left.

5.2 Acyclic communication using the MB_ASYNC_RW function block

The MB_ASYNC_RW function block is used for acyclic communication between the controller (Modbus/TCP client) and the Modbus/TCP server.

The MB_ASYNC_RW function block is available in two different versions. The two versions differ with respect to the “ServerID” input and the “IP” and “UnitID” inputs. The function block either has the “ServerID” input (old block) or the “IP” and “UnitID” inputs (new block). The new function block (see Figure 9) is supported by the following devices:

– ILC 1x1 ≥ firmware 4.20
– AXC 1050 (XC) ≥ firmware 2.10
– AXC 3050 ≥ firmware 5.50

All other controllers (see “Ordering data” on page 4) support the old block (“ServerID” input).

5.2.1 Old version of the function block

When the function block is inserted in the variable worksheet for the first time, the new function block is used by default.
If you are using a controller that supports the old function block, proceed as follows to display the old block in the variable worksheet:

- Insert the new block in the variable worksheet.
- In the project tree window, right-click to open the context menu for “Logical POUs”.
- Select the “Properties” menu item.
- In the “Change of POU/POUs properties:” dialog that opens, select the “PLC/Processor” tab.
- Select the “eCLR” entry from the drop-down list as the “PLC type”.
- Select the controller used with the corresponding firmware version from the drop-down list as the “Processor type”.
- Confirm your settings by clicking “OK”.
- In the variable worksheet, right-click to open the context menu for the function block.
- Select the “Update FB/FU” entry.

Following the update, the old block (“ServerID” input) is displayed.

In order to use the old function block, some data types need to be adapted in the predefined data structures:

- Double-click on “sys_flag_types” in the project tree window.
- In the “MBT_STATION_DIAG” structure, replace the “IP:DWORD;” data type with the “Handle:DWORD;” data type.
- In the “MBT_STATION_DIAG” structure, replace the “UnitID:BYTE;” data type with the “StationNumber:BYTE;” data type.

The old function block can now be used.

For further information on the function block, please refer to the online help for PC Worx.

5.2.2 New version of the function block

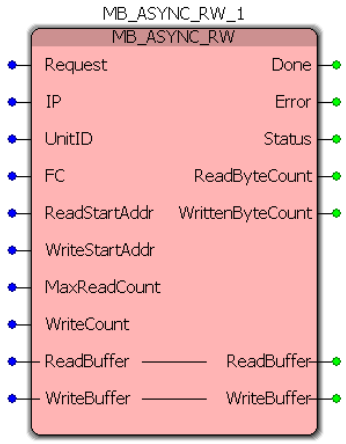


Figure 9 MB_ASYNC_RW function block

Input and output parameters of the function block		
Name	**Data type**	**Description**
Request	BOOL	The input parameters are checked and the function block is activated with a positive edge at this input. After the function code has been executed successfully, the block is deactivated and can only be activated by a new positive edge.
IP	DWORD	IP address of the “Generic Modbus Device” in the bus configuration. **Recommended:** Use the IP address from the previously created diagnostic structure (see Section 5.4). This ensures that the current settings from the bus configuration are always used. Alternatively, the IP address can be entered directly. When operating Modbus/RTU devices, the IP address of the corresponding gateway must be used.
UnitID	BYTE	UnitID of the Modbus/RTU device (Modbus/RTU slave) in the bus configuration. **Recommended:** Use the UnitID from the previously created diagnostic structure (see Section 5.4). This ensures that the current settings from the bus configuration are always used. Alternatively, the UnitID can be entered directly.
FC	BYTE	Modbus function code (see Table “Modbus function codes” on page 5)
ReadStartAddr	WORD	Start address of the memory from which data should be read.
WriteStartAddr	WORD	Start address from which data should be written to the memory.
MaxReadCount	INT	Number of bits or registers to be read.
WriteCount	INT	Number of bits or registers to be written.

Input and output parameters of the function block		
Name	**Data type**	**Description**
Done	BOOL	TRUE: The block has been executed successfully and acyclic communication has taken place. FALSE: The function block is still being executed or has not been executed.
Error	BOOL	TRUE: An error has occurred. Details are provided by the “Status” output. FALSE: No error has occurred. “Error” indicates an error as long as the “Request” input is active.
Status	DWORD	In the event of an error (Error = TRUE), the “Status” output contains an error code. The possible error codes are shown in Table “Error codes of the “Status” output” on page 15.
ReadByteCount	INT	Number or read bits/registers
WrittenByteCount	INT	Number of written bits/registers
ReadBuffer	BYTE WORD DWORD ARRAY OF BYTE, ARRAY OF WORD ARRAY OF DWORD	Buffer (250 bytes, maximum) in which the read bits/registers are stored (depending on the function code used). To define the size of the buffer, see Section 5.5 “Specifying the size of the ReadBuffer/WriteBuffer” .
WriteBuffer	BYTE WORD DWORD ARRAY OF BYTE, ARRAY OF WORD ARRAY OF DWORD	Buffer (250 bytes, maximum) for the written bits/registers (depending on the function code used). To define the size of the buffer, see Section 5.5 “Specifying the size of the ReadBuffer/WriteBuffer” .

5.3 Error codes of the “Status” output (Error = TRUE)

Error codes of the “Status” output	
Value	**Meaning**
0x0101 0000	Unsupported/unknown Modbus function
0x0102 0000	The value for MaxReadCount is outside the permissible range.
0x0103 0000	The value for WriteCount is outside the permissible range.
0x0104 0000	Wrong data type for ReadBuffer.
0x0105 0000	The elements of the ReadBuffer array are not of type WORD, DWORD or BYTE.
0x0106 0000	Wrong data type for WriteBuffer.
0x0107 0000	The elements of the WriteBuffer array are not of type WORD, DWORD or BYTE.
0x0108 0000	ReadBuffer is too small. Reduce the number of elements for MaxReadCount or increase the ReadBuffer size.
0x0109 0000	WriteBuffer is too small. Reduce the number of elements for WriteCount or increase the WriteBuffer size.
0x010A 0000	ReadBuffer is too large. To define the size of the buffer, see Section 5.5 “Specifying the size of the ReadBuffer/WriteBuffer” .

Error codes of the “Status” output	
Value	**Meaning**
0x010B 0000	WriteBuffer is too large. To define the size of the buffer, see Section 5.5 “Specifying the size of the ReadBuffer/WriteBuffer” .
0x0201 0000	Unknown server ID (not configured)
0x0202 xxxx	Modbus protocol error code For detailed information on the error codes of the Modbus/TCP protocol, please refer to the “MODBUS APPLICATION PROTOCOL SPECIFICATION” document. Current document versions can be found at modbus.org
0x0203 0016	Timeout when receiving the response from the Modbus/TCP server.
0x0203 0019	The connection was terminated by the Modbus/TCP server.
0x0203 001F	The request was not sent. The Modbus/TCP server is not accessible.
0x0301 0000	Timeout when receiving the response from the Modbus stack. The Modbus/TCP server is not accessible.

5.4 Using the MBT_STATION_DIAG diagnostic structure

More detailed information on function block diagram (FBD) programming can be found in the UM QS EN PC WORX quick start guide.

- Double-click on the “IP” input parameter of the function block to specify the variable properties.
- In the “Variable Properties” window, select the name of the previously created diagnostic structure (here: “MB_DIAG”).

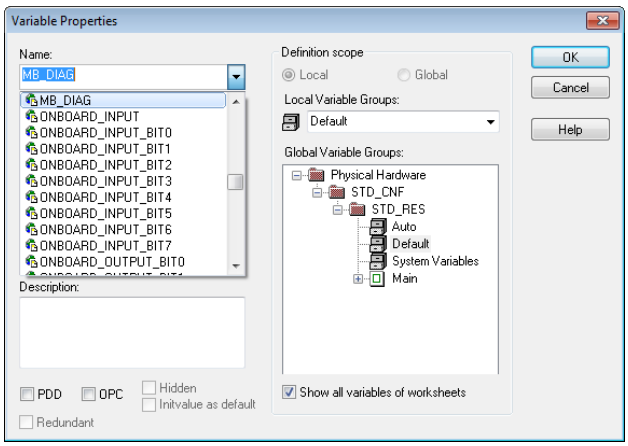


Figure 10 Creating the MB_DIAG.IP variable (1)

- Put a period after the selected name and select the “IP” entry from the list that appears.

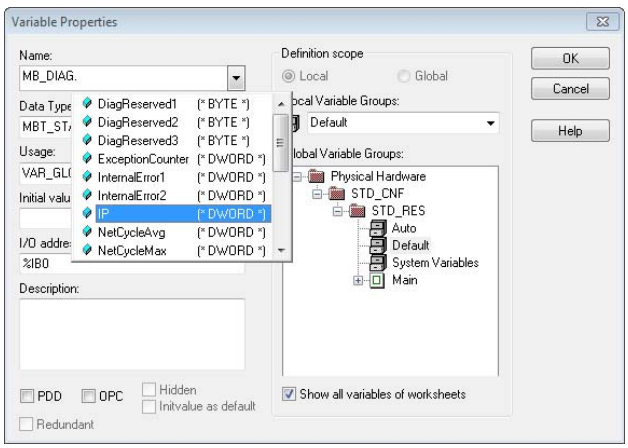


Figure 11 Creating the MB_DIAG.IP variable (2)

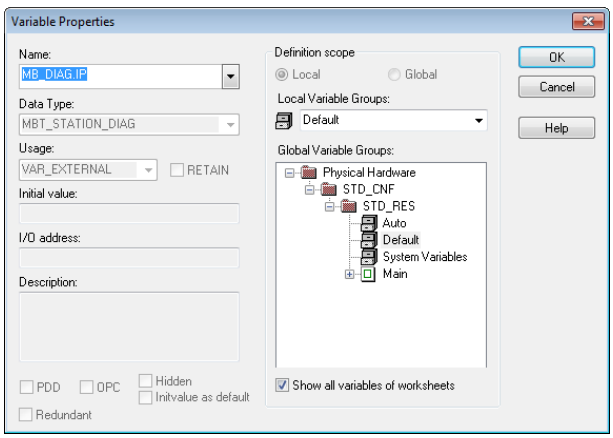


Figure 12 Creating the MB_DIAG.IP variable (3)

- Confirm your entries with “OK”.

The IP address of the “MB_DIAG” diagnostic structure has now been assigned to the “IP” input parameter of the function block (see Figure 13).

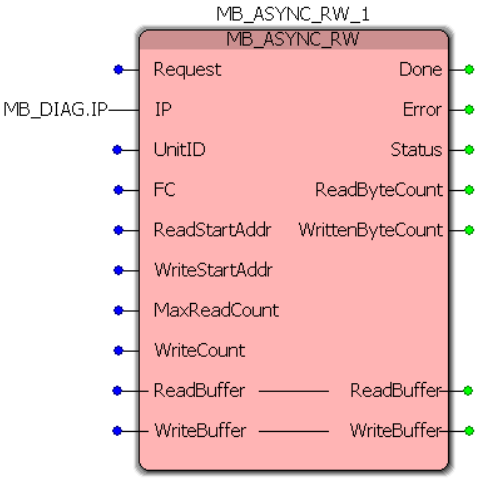


Figure 13 MB_DIAG.IP variable as “IP” in the function block

- Create the MB_DIAG.UnitID variable for the “UnitID” input parameter. Proceed as described above for the MB_DIAG.IP variable.

5.5 Specifying the size of the ReadBuffer/WriteBuffer

The size of the “ReadBuffer” and “WriteBuffer” parameters can be specified individually by defining the corresponding data types (here: “ReadBuffer” of ARRAY OF BYTE data type, “WriteBuffer” of ARRAY OF WORD data type).

The maximum size of ReadBuffer and WriteBuffer is 250 bytes each. The parameters may be of BYTE, WORD or DWORD data type, or an array of these data types.

- Double-click on “sys_flag_types” in the project tree window.
- Define the desired data types and their size as shown in Figure 14.

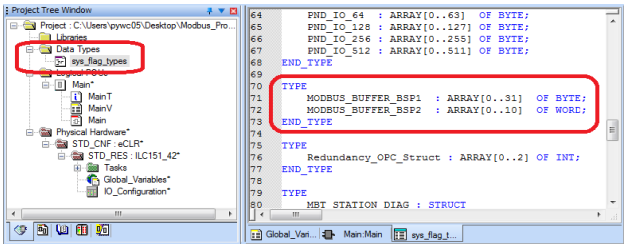


Figure 14 Creating data types

- Once you have defined the data types, select the “Build, Make” command.

Once compiled, the data types can be used in the variable worksheet. In Figure 15, the previously created “MODBUS_BUFFER_BSP1” data type is used for the “ReadBuffer” input/output parameter.

- Double-click on the “ReadBuffer” input parameter in the MB_ASYNC_RW_1 function block.
- In the “Variable Properties” window, enter a name for the variable (in the example: “Read_Buffer_1”).
- In the “Data Type” list, select the previously created data type you wish to use (in the example: “MODBUS_BUFFER_BSP1”).

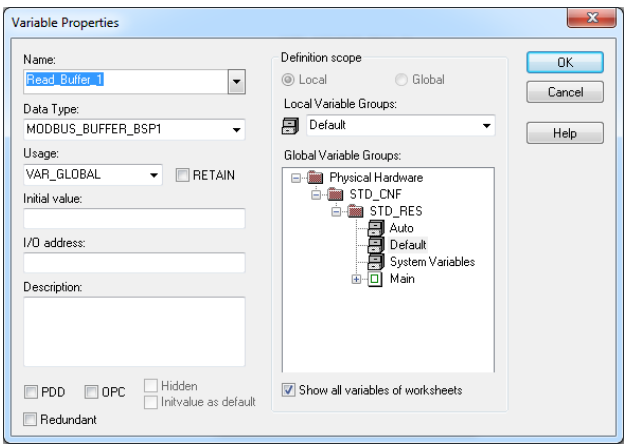


Figure 15 Creating a variable

- Confirm your entries with “OK”.

The “Read_Buffer_1” variable has now been assigned to the “ReadBuffer” input/output parameter.

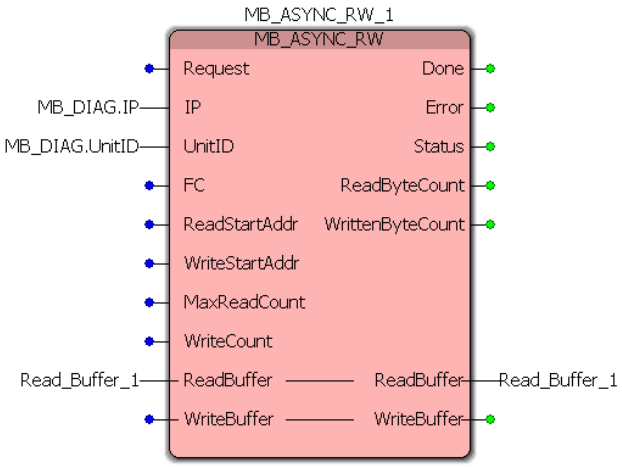


Figure 16 “Read_Buffer_1” variable as “ReadBuffer” in the function block

For further information on user-defined data types and their use, please refer to the PC Worx online help.

6 Operating lower-level Modbus/RTU devices via a gateway

This function is available as of AUTOMATIONWORX Software Suite Version 1.81 AddOn V1.

Lower-level Modbus/RTU devices can be operated via a gateway. In order to do this, all Modbus/TCP and Modbus/RTU devices as well as the gateway must be displayed in the “Bus Structure” window in PC Worx (see Figure 17).

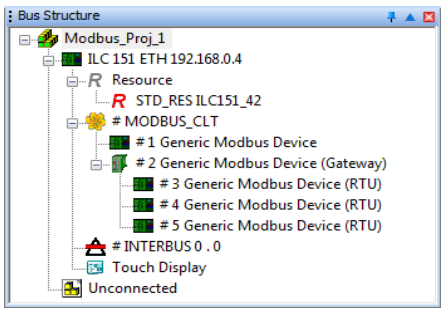


Figure 17 Bus configuration with Modbus/TCP devices and lower-level Modbus/RTU devices

Figure 17 shows two Modbus/TCP devices as an example. Device “# 1 Generic Modbus Device” is a Modbus/TCP server, device “# 2 Generic Modbus Device (Gateway)” is a gateway for switching from Modbus/TCP to Modbus/RTU. In the Modbus/TCP network, the gateway acts as the Modbus/TCP server, and in the Modbus/RTU network, it acts as the Modbus/RTU master. The individual Modbus/RTU devices (# 3 Generic Modbus Device (RTU), etc.) represent the Modbus/RTU slaves and are addressed via their UnitID.

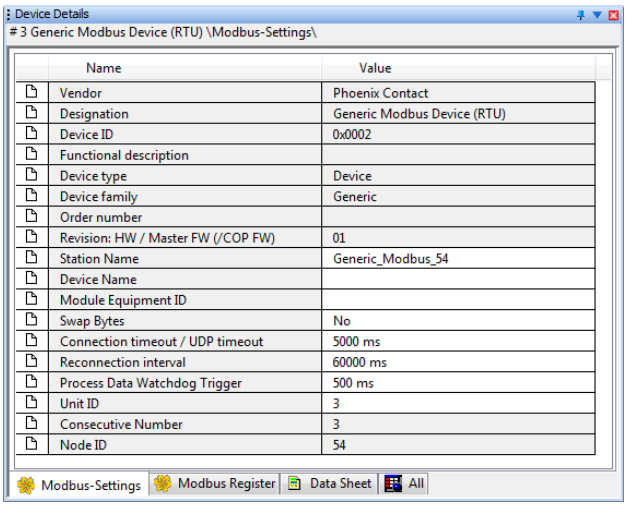


Figure 18 Settings of a Modbus/RTU device

The MB_ASYNC_RW function block is used for acyclic communication with the Modbus/RTU devices, see Section 5.2.

7 Configuring controllers as Modbus/TCP servers

As of firmware Version 4.40, ILC 1X1 controllers can also be configured as Modbus/TCP servers. This function is available as of AUTOMATIONWORX Software Suite Version 1.82 AddOn V1.

Communication connections to Modbus/TCP clients

Each individual ILC 1X1 controller supports a different number of communication connections to Modbus/TCP clients:

Controller	Max. number of communication connections to Modbus/TCP clients
ILC 131 ETH ILC 131 ETH/XC	4
ILC 151 ETH ILC 151 ETH/XC	6
ILC 171 ETH 2TX	8
ILC 191 ETH 2TX ILC 191 ME/AN ILC 191 ME/INC	8

7.1 Modbus/TCP server in PC Worx – example project

The procedure for creating a Modbus project with a controller as the Modbus/TCP server is essentially identical to the procedure described in Section 5.1 for creating a Modbus project with a controller as the Modbus/TCP client.
Only the differences between the procedures are described in more detail in this section.
Where the procedures are identical, reference is made to Section 5.1.

7.1.1 Creating a new project

See Section 5.1.1

7.1.2 Specifying project information

See Section 5.1.2

7.1.3 Checking/modifying IP settings for the controller

See Section 5.1.3

7.1.4 Inserting the Modbus/TCP server

- Make sure you are in the bus configuration workspace.
- If the device catalog is hidden, show it by selecting the “View, Device Catalog” menu.
- Open the “Phoenix Contact, Generic, Server” device catalog.
- Insert the controller as the generic Modbus/TCP server by selecting “Modbus Server”.

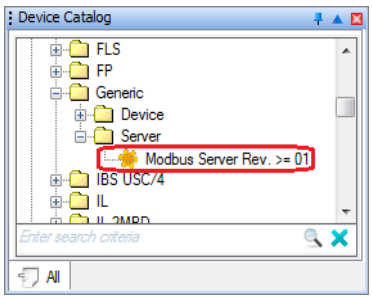


Figure 19 Selecting “Modbus Server”

- Hold down the left mouse button and move the “Modbus Server” to the “Bus Structure” window to the right of the MODBUS icon until the “Insert in the lower level” icon appears.

Figure 20 shows the bus configuration with the inserted Modbus/TCP server.

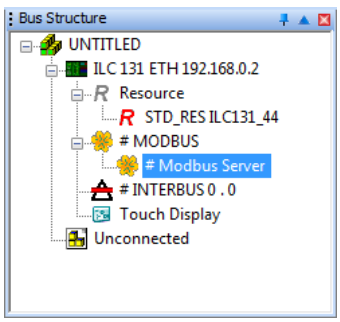


Figure 20 “Modbus Server” inserted

Only one controller can be inserted as the generic “Modbus Server”.
The IP settings for the inserted Modbus/TCP server match the IP settings that were previously made for the controller.

7.1.5 Modifying the settings for the Modbus/TCP server

Following insertion in the bus configuration, default values are set for the Modbus/TCP server. The settings can be modified via the “Modbus-Settings” tab.

- Make sure you are in the bus configuration workspace.
- Select the “Modbus Server” in the “Bus Structure” window.

- Select the “Modbus-Settings” tab in the “Device Details” window.
- Modify the Modbus settings according to your requirements.

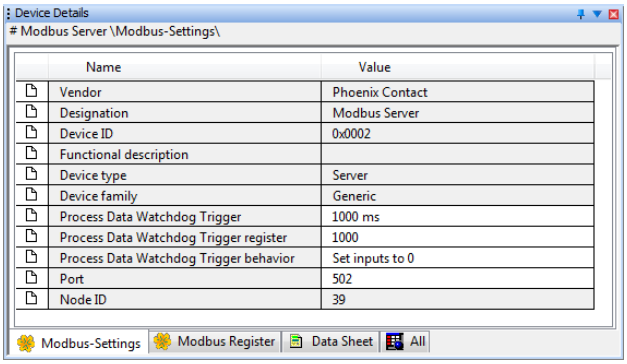


Figure 21 Modbus settings for the Modbus/TCP server

The Modbus settings for the Modbus/TCP server include:

Process Data Watchdog Trigger

The process data watchdog trigger specifies a time period during which the Modbus/TCP server must be accessed again (write access) at the latest.
The time set here should be double the process data watchdog time set on the Modbus/TCP client (see Section 5.1.5).
The time starts running after the first write access operation performed by the Modbus/TCP client.
The setting “0 ms” means that the process data watchdog is switched off.

Process Data Watchdog Trigger register

Register address (default value: 1000)

A write access operation by the Modbus/TCP client to bit 0 of this register address resets the process data watchdog with a rising edge.
The process data watchdog cannot be reset by the Modbus/TCP server.

In the case of read access operations to this register address, the current state of the process data watchdog (bit 1 = 0: watchdog not triggered, bit 1 = 1: watchdog triggered) is read.

Process Data Watchdog Trigger behavior

In the event that the process data watchdog has been triggered, substitute value behavior can be set for the Modbus/TCP server. The following settings can be made:

– “Set inputs to 0” (default setting)
  Inputs are set to zero.
– “Inputs keep the last value”
  Inputs retain their last value.
– “Inputs perform write accesses”
  Inputs are updated. The state of the process data watchdog is displayed in the “NODE_STATUS_255” diagnostic structure under “Flags”. If the watchdog has been triggered, the corresponding flag is set. In this case, you need to decide what should be done with the values of the inputs and a corresponding algorithm must be programmed.

Port

TCP/UDP port via which communication with the Modbus client takes place.
Port 502 is used by default.

7.1.6 Compiling after completing the bus topology

- Select the “Build, Make” command.

7.1.7 Creating the program

- Create the program.

To program the example program, proceed as described in the UM QS EN PC WORX quick start guide.

7.1.8 Compiling after creating the program

- Select the “Build, Make” command.

7.1.9 Creating process data and defining registers

For successful communication between Modbus/TCP client(s) and the Modbus/TCP server, registers must be defined for write and read access on both the client and server side. The corresponding process data must be created for this.
In addition, Modbus function codes for write and read access must also be specified for the Modbus/TCP client, see Section 5.1.9.

Proceed as follows to create the process data and to define the registers on the server side:

- Make sure you are in the bus configuration workspace.
- Select the “Modbus Server” in the “Bus Structure” window.
- Select the “Modbus Register” tab in the “Device Details” window.
- Right-click to open the context menu for the “Name” field.
- Select the “Add” menu item.
- Specify a unique and meaningful name for the process data item in the “Name” field.
- Select the desired data type.
- Specify the number of registers.

- Select the desired data direction.

Please note:

– Data direction “IN”:
  Registers are created to which a Modbus/TCP client has both read and write access:
  These types of registers are represented as input variables in PC Worx.

– Data direction “OUT”:
  Registers are created to which a Modbus/TCP client only has read access:
  These types of registers are represented as output variables in PC Worx.

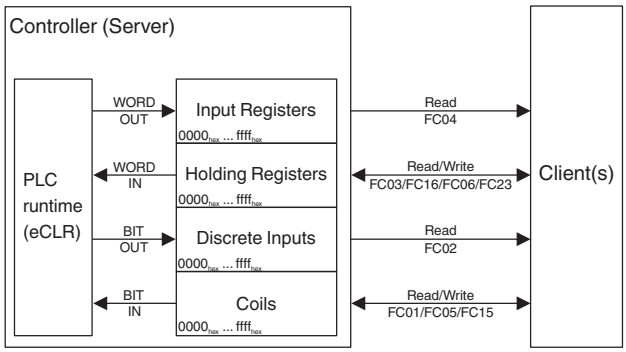


Figure 22 Data direction for internal and external Modbus/TCP server communication (ILC 1X1 as of FW 4.40)

In the PC Worx bus configuration, only the Modbus/TCP server supports Modbus function codes FC05 and FC06.

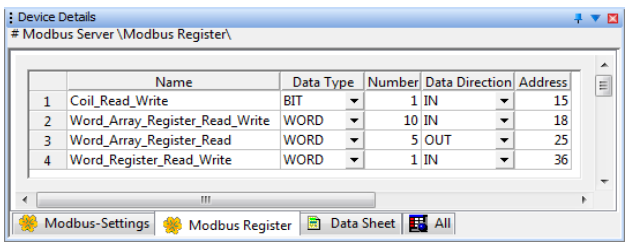


Figure 23 Creating process data and defining registers

If one of the controllers listed in the “Ordering data” on page 4 is used as a Modbus/TCP client and an ILC 1X1 controller with firmware Version ≥ 4.40 is used as the Modbus/TCP server, the following should be noted:

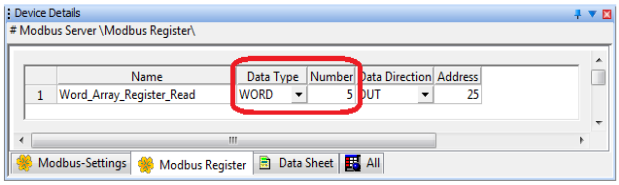
If the data types of the defined registers on the client and server side are not coordinated, the byte order is reversed when reading or writing registers.

NOTE:
When linking a process variable of Array [] of (D)WORD data type, the byte order may be reversed. For further information, please refer to Section 9.

Example:

Figure 24 shows register definitions on the server and client side that are not coordinated.

Server



Client

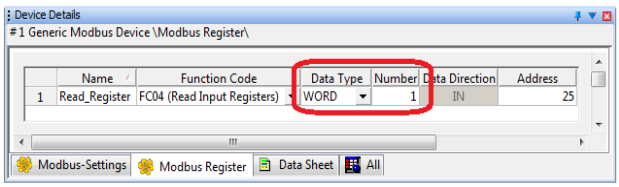


Figure 24 Register definitions that are not coordinated

A WORD array with a length of 5 has been created for the register on the server side (see Figure 25).

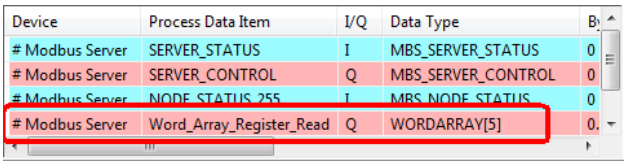


Figure 25 Data type of the defined register for the Modbus server in the process data assignment workspace

A process variable of Array [0 ... 4] of WORD data type must be created and linked to the Word_Array_Register_Read process data item (see Section 7.1.10).

A simple, single WORD data type (not an array) has been created for the register on the client side (see Figure 26).

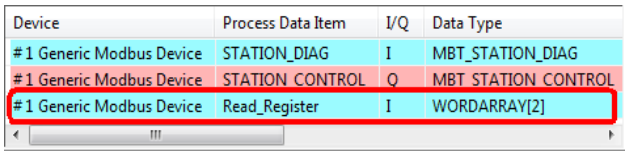


Figure 26 Data type of the defined register for the Modbus client in the process data assignment workspace

A process variable of WORD data type must be created and linked to the Read_Register process data item (see Section 7.1.10).

If the function code defined for the client is now executed, the first element of the WORD array is read by the server and copied to the process variable defined for the client in the reverse byte order (see Figure 27).

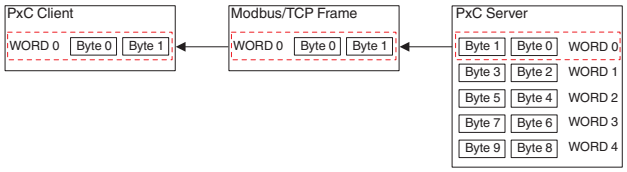
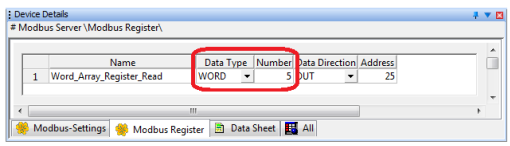


Figure 27 Reverse byte order in the client process variable

In order to avoid a reversed byte order, a WORD array must be defined on both the client side and the server side when using arrays.
In order to do this in the example, a minimum value of “2” must be entered under “Number” on the “Modbus Register” tab on the client side and on the server side, see Figure 28.

Server



Client

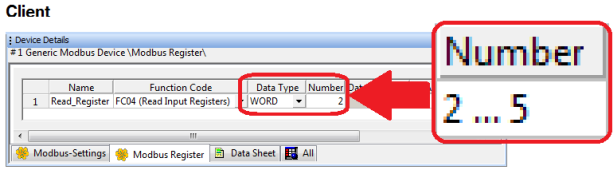


Figure 28 Register definitions that are coordinated

Please note:
The selected data type of the process variable on the server side must match the data type on the client side (WORD array → WORD array or WORD → WORD).
If not, the byte order is reversed.

Figure 29 shows the byte order in the process variable for the client when the register definitions are coordinated.

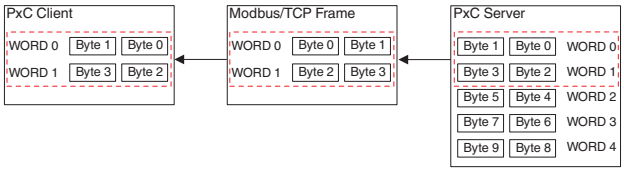


Figure 29 Byte order in the client process variable for coordinated process variable linking when using the WORD array data type on both sides

7.1.10 Generating variables and assigning process data

- Assign the created process data to the corresponding variables.
- If corresponding variables have not been created yet, generate them.

For information on assigning process data to variables and generating variables, please refer to Section 5.1.10.

7.1.11 Generating diagnostic and control variables and assigning process data

In the IEC programming workspace, the diagnostic and control structure is declared in the project tree window under “Data Types, sys_flag_types”.

PC Worx provides the following diagnostic and control structures for the Modbus server:

– “MBS_SERVER_STATUS” diagnostic structure:

Parameter	Description
AcceptCnt	Number of existing and currently established connections
AcceptErrorCnt	Number of errors during connection establishment
ReceiveErrorCnt	Number of errors when receiving requests
ModbusRequesCnt	Number of requests received
ModbusExceptionCnt	Number of exceptions sent
SocketErrorCode	Error code of the socket error

Parameter	Description
ConnectedClients	Number of Modbus clients currently connected
LastExceptionRegister	Register in which the last exception occurred
LastExceptionRegLength	Length of the register in which the last exception occurred
LastExceptionNode	Server instance (node) where the last exception occurred
LastExceptionFunction	Function field of the request for which the last exception occurred
LastExceptionCode	Modbus exception code of the last exception that occurred
align	Used for alignment

An ILC 1X1 controller operated as the Modbus/TCP server has **one** server instance (node). The server instance has the UnitID 255.
The UnitID can be freely selected in the Modbus/TCP client request in order to access specific registers of the Modbus/TCP server.

– “MBS_NODE_STATUS” diagnostic structure:

Parameter	Description
ModbusRequestCnt	Number of requests received for this server instance (node)
ModbusExceptionCnt	Number of exceptions sent for this server instance (node)
Flags	If the process data watchdog has been triggered, the flag for this server instance (node) is set (bit 0 = 1).
reserved	Reserved

– “MBS_SERVER_CONTROL” control structure:

Parameter	Description
Flags	All parameters of the diagnostic structures are reset with a rising edge at bit 0.
reserved1	Reserved
reserved2	Reserved
reserved3	Reserved

In order to use the diagnostic and control structures, create two diagnostic variables and one control variable.

- Switch to the IEC programming workspace.
- Double-click on “Global_Variables” in the project tree window.

The global variables of the standard resource are displayed.

- Insert a new variable via the context menu which should be used as the diagnostic variable for the “SERVER_STATUS” structure.
- Select the MBS_SERVER_STATUS type for the diagnostic variable.
- Insert another variable via the context menu which should be used as the diagnostic variable for the “MBS_NODE_STATUS” structure.
- Select the MBS_NODE_STATUS type for the diagnostic variable.
- Insert another new variable via the context menu which should be used as the control variable for the “SERVER_CONTROL” structure.
- Select the MBS_SERVER_CONTROL type for the diagnostic variable.

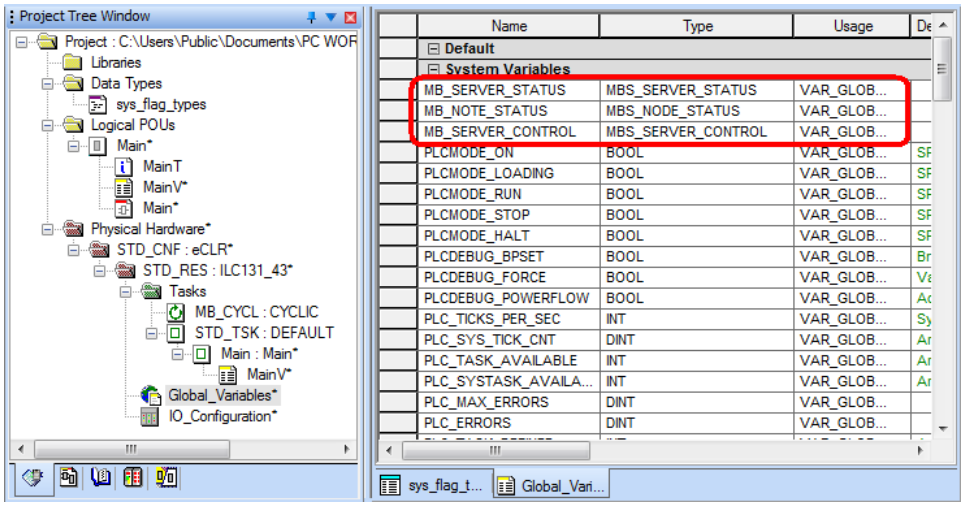


Figure 30 Generating diagnostic and control variables

- Switch to the process data assignment workspace to assign the process data to the control and diagnostic variables, as described in Section 5.1.10 on page 9.

The result of process data assignment is shown in Figure 31.

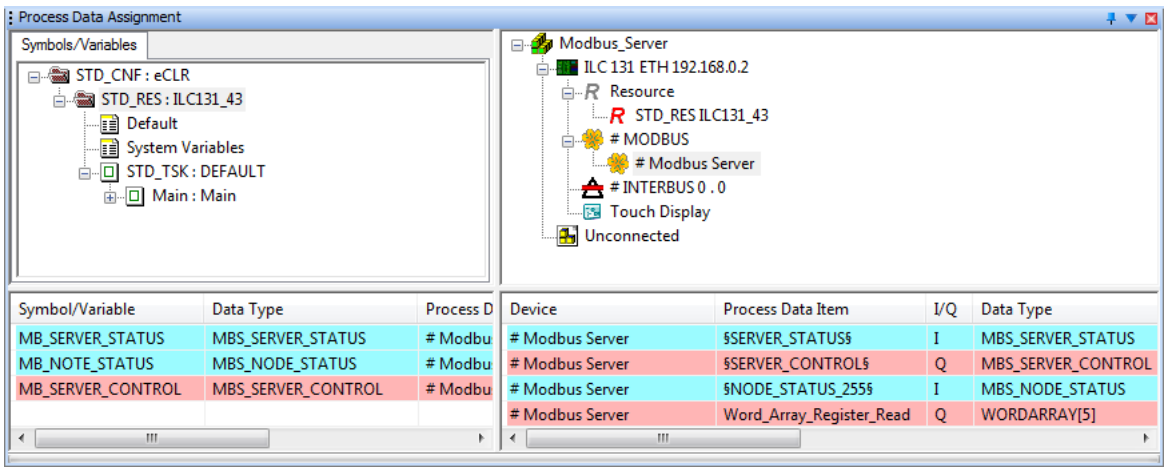


Figure 31 Process data assigned to the diagnostic and control variables

7.1.12 Defining the cyclic task for the update time of the Modbus registers

The Modbus registers are updated with the standard task (STD_TASK) by default. The standard task runs with the fastest possible cycle, where the cycle time is often 4 ms. This results in a high CPU load. In order to reduce the CPU load, a cyclic task can be defined with a cycle time that is greater than the standard task cycle time. Since the Modbus registers are updated within a longer cycle time, the load on the CPU is lower and more resources are available for the PC Worx project.

- Create a cyclic task and specify an appropriate cycle time (“Interval” in the “Task Settings” window; recommended setting: > 10 ms).
  For information on creating a task, please refer to the UM QS EN PC WORX quick start guide or the PC Worx online help.
- In the “Bus Structure” window, select the standard resource and in the “Device Details” window select the cyclic task created previously for “I/O Update by Task” (see Figure 32).

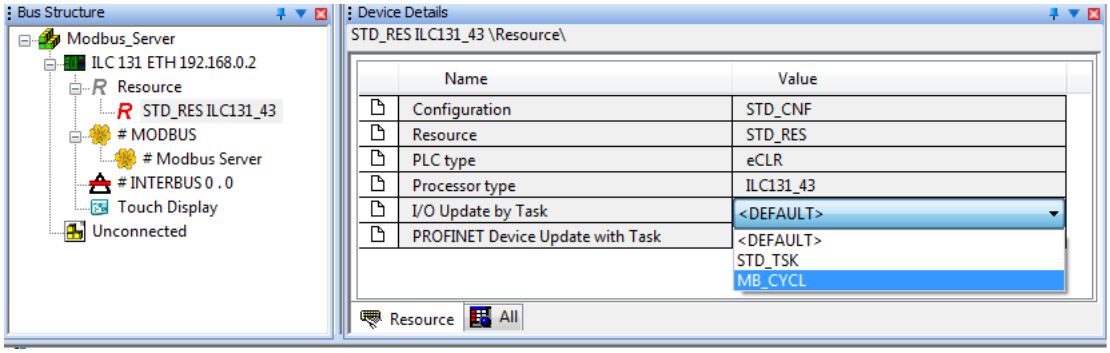


Figure 32 Selecting the cyclic task for “I/O Update by Task”

7.1.13 Compiling a project

- Select the “Build, Make” command.

7.1.14 Sending a project and performing a cold restart

See Section 5.1.14

8 Configuring multi-controller projects – example project

As of firmware Version 4.40, ILC 1X1 controllers can also be configured as Modbus/TCP servers. This function is available as of AUTOMATIONWORX Software Suite Version 1.82 AddOn V1.

In multi-controller projects, controllers can either be operated as Modbus/TCP clients, as Modbus/TCP servers or simultaneously as Modbus/TCP clients and Modbus/TCP servers.
The bus configuration of a multi-controller project can be mapped in **one** PC Worx project.

Figure 33 shows a typical client/server structure.

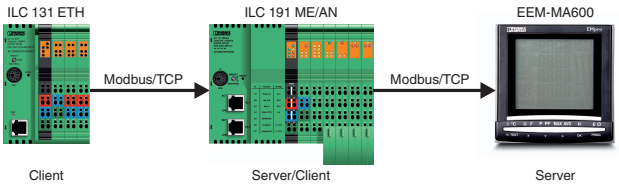


Figure 33 Typical client/server structure

In the example, the controller on the left
is operated as a Modbus/TCP client. The controller in the middle is simultaneously operated as a Modbus/TCP server (in relation to the controller on the left) and as a Modbus/TCP client (in relation to the energy meter on the right).

The step-by-step procedure for creating the example as a PC Worx project is described below.
The procedure for creating a multi-controller project is generally identical to the procedures described in Sections 5.1 and 7.1.
Only the differences between the procedures are described in more detail in this section.
Where the procedures are identical, reference is made to the relevant section.

8.1 Creating a new project

- Create a new project and select the ILC 131 ETH as the controller, see Section 5.1.1.

8.2 Specifying project information

See Section 5.1.2

8.3 Checking/modifying IP settings for the controller

See Section 5.1.3

8.4 Inserting a specific Modbus device

In the example, the ILC 191 ME/AN controller is used as the Modbus/TCP server.

Specific settings are required for communication between the Modbus/TCP client (here: ILC 131 ETH) and server. These settings are made in PC Worx via the specific Modbus device of the controller used as the Modbus/TCP server. The specific Modbus device of the controller used as the Modbus/TCP server must be inserted in the bus configuration so as to enable communication between the Modbus/TCP client and server.

- Make sure you are in the bus configuration workspace.
- If the device catalog is hidden, show it by selecting the “View, Device Catalog” menu.
- Open the “Phoenix Contact, ILC1xx, PLC” device catalog.
- Select the specific Modbus device for the ILC 191 ME/AN controller (“ILC 191 ME/AN (Modbus Device)”).

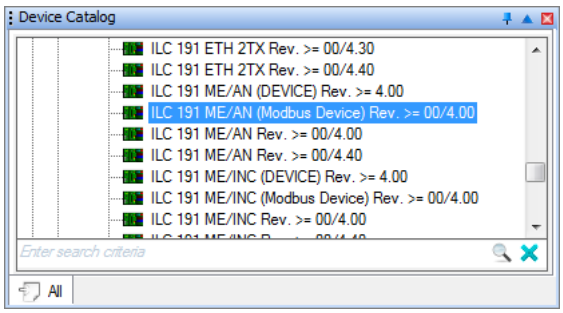


Figure 34 Selecting the specific Modbus device “ILC 191 ME/AN (Modbus Device)”

- Hold down the left mouse button and move the specific Modbus device “ILC 191 ME/AN (Modbus Device)” to the “Bus Structure” window to the right of the MODBUS icon until the “Insert in the lower level” icon appears.

Figure 35 shows the bus configuration with the inserted specific Modbus device.

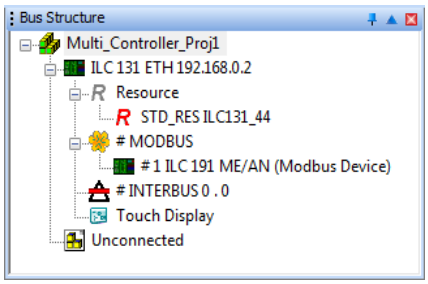


Figure 35 Specific Modbus device “ILC 191 ME/AN (Modbus Device)” inserted

8.5 Modifying the settings for the specific Modbus device

Following insertion in the bus configuration, default values are set for each “Generic Modbus Device”. The settings can be modified via the “Modbus-Settings” tab.

The settings for the specific Modbus device are required for communication between the Modbus/TCP client (here: ILC 131 ETH) and the Modbus/TCP server (here: ILC 191 ME/AN).

- Make sure you are in the bus configuration workspace.
- Select the specific Modbus device “ILC 191 ME/AN (Modbus Device)” in the “Bus Structure” window.
- Select the “Modbus-Settings” tab in the “Device Details” window.
- Modify the Modbus settings according to your requirements.

For information on the Modbus settings, please refer to Section 5.1.5.

8.6 Inserting the ILC 191 ME/AN controller

Once you have inserted the specific Modbus device “ILC 191 ME/AN (Modbus Device)”, you must insert the ILC 191 ME/AN controller.

- Make sure you are in the bus configuration workspace.
- If the device catalog is hidden, show it by selecting the “View, Device Catalog” menu.
- Open the “Phoenix Contact, ILC1xx, PLC” device catalog.
- Select the “ILC 191 ME/AN Rev. >= 00/4.40” controller.
- Hold down the left mouse button and move the “ILC 191 ME/AN Rev. >= 00/4.40” controller to the “Bus Structure” window.

Figure 36 shows the bus configuration with the inserted ILC 191 ME/AN controller.

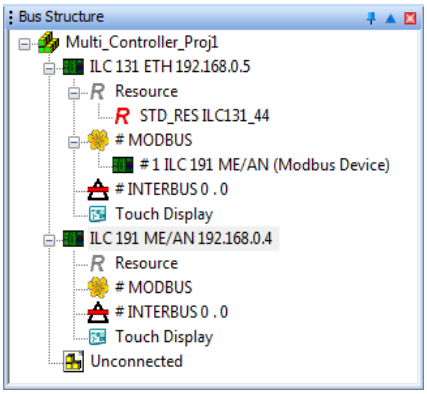


Figure 36 Inserted ILC 191 ME/AN controller

8.7 Checking/modifying IP settings for the controller

See Section 5.1.3

8.8 Inserting the Modbus/TCP server

In the example, the ILC 191 ME/AN controller should act as the Modbus/TCP server in relation to the ILC 131 ETH. You must therefore insert the controller as the generic Modbus/TCP server, see Section 7.1.4.

Figure 37 shows the bus configuration with the inserted generic Modbus/TCP server.

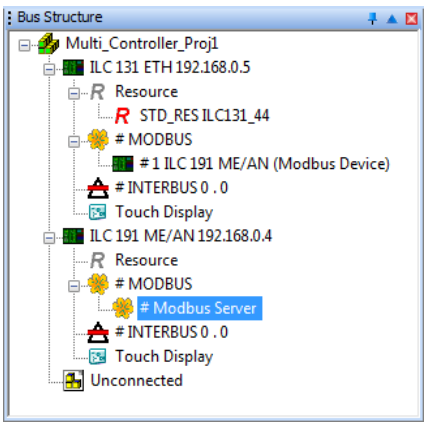


Figure 37 “Modbus Server” inserted

Only one controller can be inserted as the generic “Modbus Server”.
The IP settings for the inserted Modbus/TCP server match the IP settings that were previously made for the controller.

8.9 Modifying the settings for the Modbus/TCP server

See Section 7.1.5

8.10 Inserting a “Generic Modbus Device”

In the example, the ILC 191 ME/AN controller should simultaneously be operated as a Modbus/TCP server (in relation to the ILC 131 ETH controller) and
as a Modbus/TCP client (in relation to the EEM-MA400 energy meter) (see Figure 33).

Specific settings are required for communication between the ILC 191 ME/AN controller and the EEM-MA400 energy meter.
These settings are made in PC Worx via the “Generic Modbus Device”.

- To insert the “Generic Modbus Device”, proceed as described in Section 5.1.4.

Figure 38 shows the bus configuration with the inserted “Generic Modbus Device”.

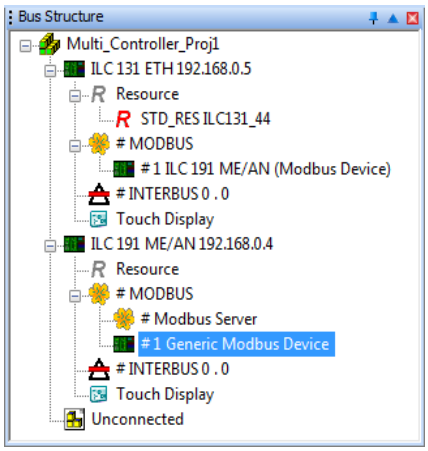


Figure 38 “Generic Modbus Device” inserted

8.11 Modifying the settings for the “Generic Modbus Device”

See Section 5.1.5

The bus topology for the example project is now complete.

8.12 Compiling after completing the bus topology

- Select the “Build, Make” command.

8.13 Creating the program

- Create the program.

To program the example program, proceed as described in the UM QS EN PC WORX quick start guide.

8.14 Compiling after creating the program

- Select the “Build, Make” command.

8.15 Creating process data and assigning Modbus function codes

- Create the necessary process data for the specific Modbus device of the ILC 191 ME/AN controller and for the “Generic Modbus Device” and assign it to the corresponding Modbus function codes.
  Proceed as described in Section 5.1.9.

8.16 Generating variables and assigning process data

- Generate variables for the specific Modbus device of the ILC 191 ME/AN controller and for the “Generic Modbus Device” and assign the corresponding process data to the variables.
  Proceed as described in Section 5.1.10.

8.17 Generating diagnostic and control variables and assigning process data

- Generate diagnostic and control variables for the specific Modbus device of the ILC 191 ME/AN controller and for the “Generic Modbus Device” and assign the corresponding process data to the variables.
  Proceed as described in Section 5.1.11.

8.18 Creating process data and defining registers

- Create process data for the ILC 191 ME/AN controller and define the required registers.
  Proceed as described in Section 7.1.9.

8.19 Generating variables and assigning process data

- Generate variables for the ILC 191 ME/AN controller and assign the corresponding process data to the variables.
  Proceed as described in Section 7.1.10.

8.20 Generating diagnostic and control variables and assigning process data

- Generate diagnostic and control variables for the ILC 191 ME/AN controller and assign the corresponding process data to the variables.
  Proceed as described in Section 7.1.11.

8.21 Defining the cyclic task for the update time of the Modbus registers

- Create a cyclic task for the ILC 191 ME/AN controller.
  Proceed as described in Section 7.1.12.

8.22 Compiling a project

- Select the “Build, Make” command.

8.23 Configuring the Modbus/TCP server

- Perform all of the required configurations (e.g., setting the IP address and process data watchdog) directly on the Modbus/TCP server (here: EEM-MA600 energy meter).

For information on how to configure and start up the device, please refer to the corresponding documentation.

8.24 Sending a project and performing a cold restart

See Section 5.1.14

9 Special considerations when reading and writing WORD arrays

If you use one of the controllers listed in the “Ordering data” on page 4 as the Modbus/TCP client and a device from another manufacturer as the Modbus/TCP server (or an ILC 1X1 controller as the Modbus/TCP server and a device from another manufacturer as the Modbus/TCP client), the byte order for reading/writing is reversed if WORD arrays are used (“Swap Bytes” setting equals “no”, see Section 5.1.5).

9.1 General examples

The examples below show how the transmitted data of the Modbus/TCP frame is mapped in the process variables of the PC Worx project.

Example 1

Process variable of WORD data type, “Swap Bytes” setting equals “no”.

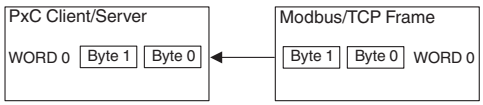


Figure 39 Process variable of WORD data type

Example 2

Process variable of Array [0 ... 1] of WORD data type, “Swap Bytes” setting equals “no”.

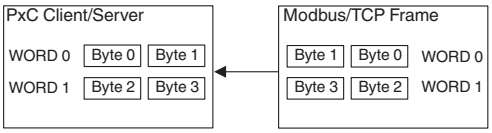


Figure 40 Process variable of Array [0 ... 1] of WORD data type

Example 3

Process variable of WORD data type, “Swap Bytes” setting equals “yes”.

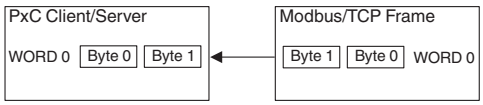


Figure 41 Process variable of WORD data type

Example 4

Process variable of Array [0 ... 1] of WORD data type, “Swap Bytes” setting equals “yes”.

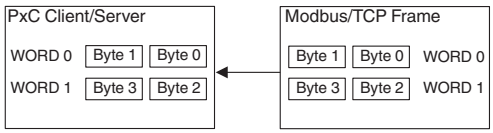


Figure 42 Process variable of Array [0 ... 1] of WORD data type

Example 5

Process variable of DWORD data type, “Swap Bytes” setting equals “no”.

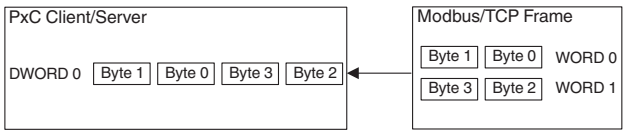


Figure 43 Process variable of DWORD data type

Example 6

Process variable of Array [0 ... 1] of DWORD data type, “Swap Bytes” setting equals “no”.

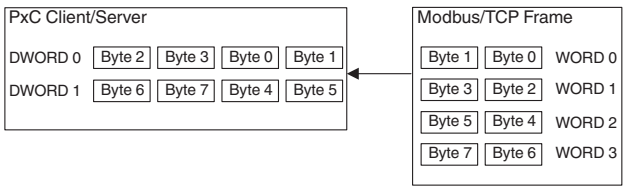


Figure 44 Process variable of Array [0 ... 1] of DWORD data type

9.2 Concrete examples for the general examples in Section 9.1

The following concrete examples from PC Worx show the linking of the data of a Modbus/TCP telegram (Modbus/TCP frame) to the possible process variables and their data types in PC Worx.

In the examples, one of the controllers listed in the “Ordering data” on page 4 is used as the Modbus/TCP client and a device from another manufacturer is used as the Modbus/TCP server. It is always the Modbus/TCP telegram that is taken into consideration and not the Modbus/TCP server, as the byte order can differ from Modbus/TCP server to Modbus/TCP server.
The byte order shown in the examples when linking the data of a Modbus/TCP telegram to the possible process variables and their data types in PC Worx also applies when a device from another manufacturer is used as the Modbus/TCP client and one of the controllers listed in the “Ordering data” on page 4 is used as the Modbus/TCP server.

9.2.1 Example 1: Read access for the Modbus/TCP client

The Modbus/TCP client should read up to four words of the Modbus/TCP server, starting with the address 0.
The read data is linked to various data types in PC Worx.

Configuration in PC Worx

Figure 45 shows the configuration and Modbus function codes used in PC Worx.

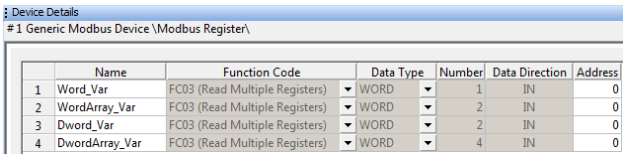


Figure 45 Configuration and Modbus function codes used for the Modbus registers

Figure 46 shows the declaration of the corresponding required process variables.

Name	Type	Usage
Default		
Word_Var	WORD	VAR_GLOBAL
WordArray_Var	MB_2Word	VAR_GLOBAL
DWordArray_Var	MB_2DWord	VAR_GLOBAL
DWord_Var	DWORD	VAR_GLOBAL

Figure 46 Declaration of the required process variables

Figure 47 shows how the “MB_Word” and “MB_2DWord” data types must be defined.

```
TYPE
    MB_2Word   : ARRAY[0..1] OF WORD;
    MB_2DWord  : ARRAY[0..1] OF DWORD;
END_TYPE
```

Figure 47 Definition of the “MB_2Word” and “MB_2DWord” data types

Figure 48 shows the linking of the process variables to the Modbus registers in the process data assignment workspace.

Symbol/Variable	Data Type	Process Data Item
Word_Var	WORD	#1 Generic Modbus Device \ Word_Var
WordArray_Var	MB_2Word	#1 Generic Modbus Device \ WordArray_Var
DWordArray_Var	MB_2DWord	#1 Generic Modbus Device \ DwordArray_Var
DWord_Var	DWORD	#1 Generic Modbus Device \ Dword_Var

Figure 48 Linking of the process variables to the Modbus registers

Online values during operation

Figure 49 shows the online values during operation. The Modbus/TCP telegram for the last configured Modbus register (Array [0 ... 3] of WORD, see Figure 45) is shown. The byte order for the three other Modbus registers in the example is identical to the byte order in Figure 49.

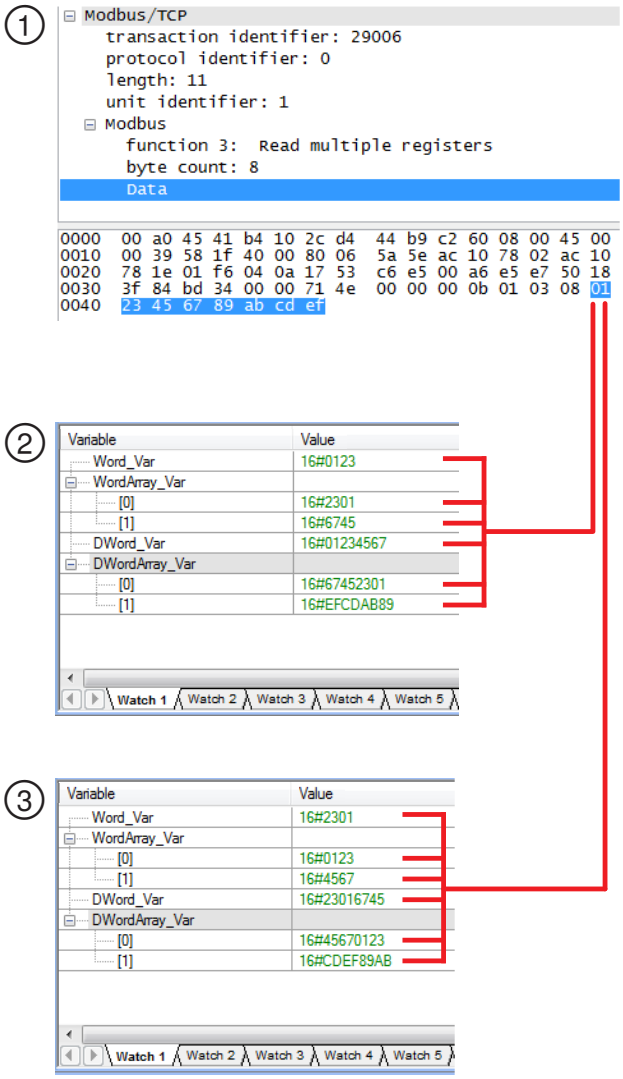


Figure 49 Online values during operation

Key:

1. Modbus/TCP telegram in the Wireshark software (software for analyzing data traffic in the network)
2. Linked process variables in the Watch window in PC Worx when the “Swap Bytes” setting equals “no”
3. Linked process variables in the Watch window in PC Worx when the “Swap Bytes” setting equals “yes”

9.2.2 Example 2: Write access for the Modbus/TCP client

The Modbus/TCP client writes to the Modbus/TCP server. Up to four words should be written in the Modbus/TCP telegram.

Configuration in PC Worx

Figure 50 shows the Modbus registers and the Modbus function codes used in PC Worx.

Device Details
2 Generic Modbus Device \Modbus Register\

	Name	Function Code	Data Type	Number	Data Direction	Address
1	Word_Var	FC16 (Write Multiple Registers)	WORD	1	OUT	0
2	WordArray_Var	FC16 (Write Multiple Registers)	WORD	2	OUT	1
3	DWord_Var	FC16 (Write Multiple Registers)	WORD	2	OUT	3
4	DWordArray_Var	FC16 (Write Multiple Registers)	WORD	4	OUT	5

Figure 50 Configuration and Modbus function codes used

Figure 51 shows the declaration of the corresponding required process variables.

Name	Type	Usage
Default		
Word_Var	WORD	VAR_GLOBAL
WordArray_Var	MB_2Word	VAR_GLOBAL
DWordArray_Var	MB_2DWord	VAR_GLOBAL
DWord_Var	DWORD	VAR_GLOBAL

Figure 51 Declaration of the required process variables

Figure 52 shows how the “MB_Word” and “MB_2DWord” data types must be defined.

```
TYPE
    MB_2Word   : ARRAY[0..1] OF WORD;
    MB_2DWord  : ARRAY[0..1] OF DWORD;
END_TYPE
```

Figure 52 Definition of the “MB_2Word” and “MB_2DWord” data types

Figure 53 shows the linking of the process variables to the Modbus registers in the process data assignment workspace.

Symbol/Variable	Data Type	Process Data Item
Word_Var	WORD	# 2 Generic Modbus Device \ Word_Var
WordArray_Var	MB_2Word	# 2 Generic Modbus Device \ WordArray_Var
DWordArray_Var	MB_2DWord	# 2 Generic Modbus Device \ DWordArray_Var
DWord_Var	DWORD	# 2 Generic Modbus Device \ DWord_Var

Figure 53 Linking of the process variables to the Modbus function codes

Online values during operation

Figure 54 shows the online values during operation when the “Swap Bytes” setting equals “no”.

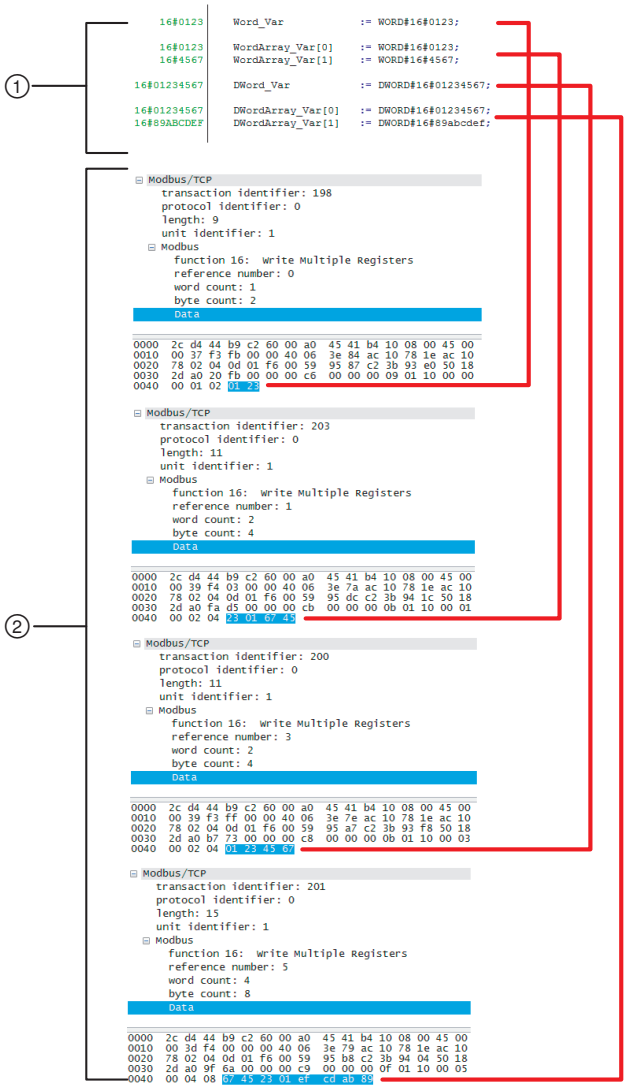


Figure 54 Online values during operation, “Swap Bytes” setting equals “no”

Key:

1. Fixed values that are written to the various process variables in PC Worx.
2. Fixed values from PC Worx, how they are written in the Modbus/TCP telegram, shown in the Wireshark software.

Figure 55 shows the online values during operation when the “Swap Bytes” setting equals “yes”.

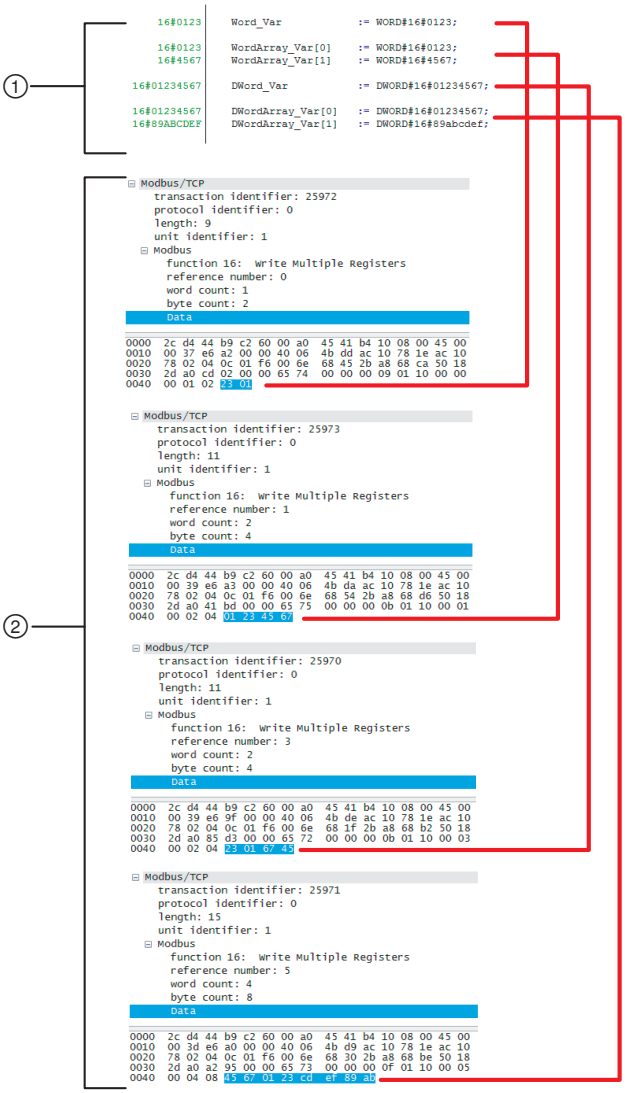


Figure 55 Online values during operation, “Swap Bytes” setting equals “yes”

Key:

1. Fixed values that are written to the various process variables in PC Worx.
2. Fixed values from PC Worx, how they are written in the Modbus/TCP telegram, shown in the Wireshark software.

SCATTERGOOD & JOHNSON LTD
ELECTRICAL ENGINEERING & FLUID CONTROL DISTRIBUTORS
Est.1899

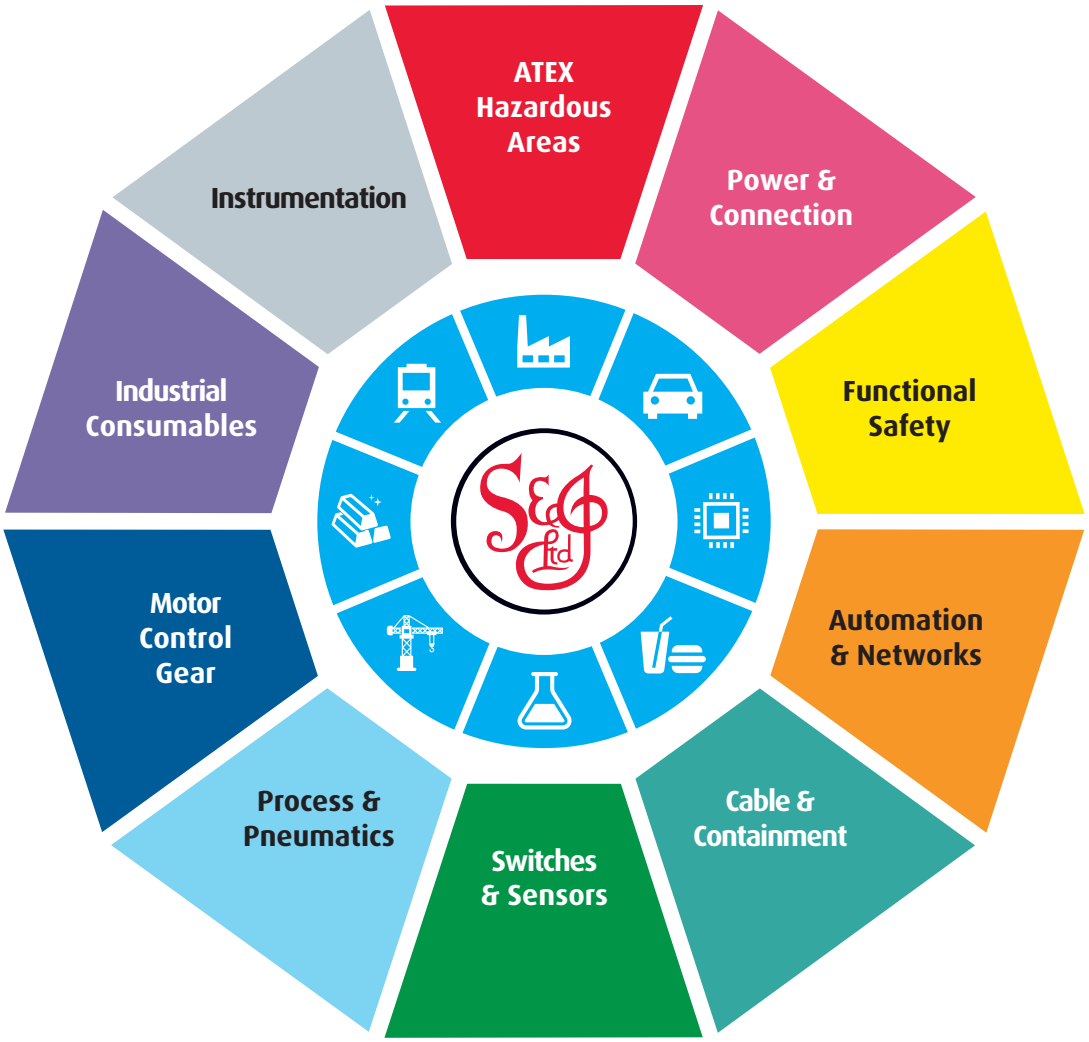
At Scattergood & Johnson Ltd, we pride ourselves on being a technical distributor to specialist industries.

Working with a range of quality product suppliers across a number of specialist markets, we are not your average 'box shifter' - we are your technical and supply chain partner.

We fully support every product we sell - for free! Our internal team and external sales engineers can answer any product or application question, no matter the complexity.

Backing up this technical ability is a range of 50,000+ products available from stock for nationwide next day delivery (same day if required!), or you can collect what you need from any of our trade counters around the UK.

Select your specialist interest below to learn more about how we can help.



Online, In Branch and On the Road - Scattergood & Johnson Ltd, there when you need us.

www.scatts.co.uk