Function block library

Modbus_TCP_7

for PLCnext Engineer

Documentation for
PHOENIX CONTACT function blocks
PHOENIX CONTACT GmbH Co. KG
Flachsmarktstrasse 8
D32825 Blomberg, Germany

This documentation is available in English only.

Table of Contents

1 Installation hint

If you did not specify a different directory during **library** installation all data in the MSI file will be unpacked to

c:\Users\Public\Documents\Phoenix Contact Libraries\PLCnext Engineer (former: PC Worx Engineer)

Please copy the library data to your PLCnext Engineer (former: PC Worx Engineer) working library directory.

If you did not specify a different directory during **PLCnext Engineer** installation the default PLCnext Engineer working library directory is

c:\Users\Public\Documents\PLCnext Engineer\Libraries (former: PC Worx Engineer\Libraries)

2 General information

2.1 Modbus

Modbus is an open protocol for control communication with various functions for read and write access for digital inputs, outputs and registers.
Modbus is maintained by a dedicated user organization; documents with a detailed protocol description are available from the organization's homepage at www.modbus.org. Depending on the transmission system, a distinction is made between the Modbus RTU, Modbus ASCII, and Modbus TCP protocols.

2.2 Modbus TCP

For the MB_TCP_Server function block, the registers used for Modbus correspond to the elements of the udtTCP_ComData array. Each WORD array element is to be considered as a 16-bit Modbus register. These registers can be accessed from the local user program, in which the Modbus server is embedded, as well as from a remote control system when a Modbus client is used.

For this purpose the mentioned function codes are available.

3 Change notes

Library version	Library build	PLCnext Engineer version	Change notes	Supported PLCs
7	20200730	>= 2020.0 LTS	MB_TCP_Server_2: • Adapted uiOffsetInputRegister and uiOffsetHoldingRegister • Enabled UDP mode • Missing parameters in udtDiag added MB_TCP_Client_4: • Add the inputs strBindIp and uiBindPort (important for UDP-Mode) • Missing parameters in udtDiag added • New wDiagCode: 16#C410 MB_TCP_FC23_4: • Bugfix: Adapted max. uiQuantityToWrite and diagnostic codes Structures MB_TCP_UDT_CLI_DIAG and MB_TCP_UDT_SER_DIAG: • udtIP_UDP_SOCKET changed to udtUDP_SOCKET • udtIP_UDP_SEND changed to udtUDP_SEND • udtIP_UDP_RECEIVE changed to udtUDP_RECEIVE	AXC F 1152 (1151412) AXC F 2152 (2404267) AXC F 3152 (1069208)
6	20200206	>= 2020.0 LTS	Documentation: - FC23 added	AXC F 1152 (1151412) AXC F 2152 (2404267)
6	20191002	2019.0 LTS 2019.3 2019.6 2019.9	Adapted to 2019.9	AXC F 2152 (2404267)
5	20190723	2019.0 LTS 2019.3 2019.6	Adapted to 2019.6	AXC F 2152 (2404267)
4	20190226	2019.0 LTS	Supports “Allow extended identifiers” = ON	AXC F 2152 (2404267)
4	20190219	2019.0 LTS	Adapted to 2019.0 LTS	AXC F 2152 (2404267)

3	20181119	7.2.3	MB_TCP_Server_3: • Bugfix - Diag codes adapted to new IP function blocks MB_TCP_Client_3: • Enabled UDP mode • Bugfix - Diag codes adapted to new IP function blocks MB_TCP_DiagInfo_EN_2: • Bugfix - Diag codes adapted to new IP function blocks MB_TCP_FC1,2,3,4,23_3: • Bugfix - uiByteCount	AXC F 2152 (2404267)
2	20180926	7.2.3	Modbus_TCP_2: • Adapted to PLCnext Engineer 7.3 MB_TCP_Server_2: • Bugfix for “udtDiag.udtTCP_SOCKET.dwHandle not connected to firmware function block”	AXC F 1050 (2404701) AXC F 2152 (2404267)
1			Converted from PC Worx 6	AXC F 1050 (2404701) AXC F 2152 (2404267)

New version number: Functional changes of at least one function block, incompatibilities (e.g. change of library format)

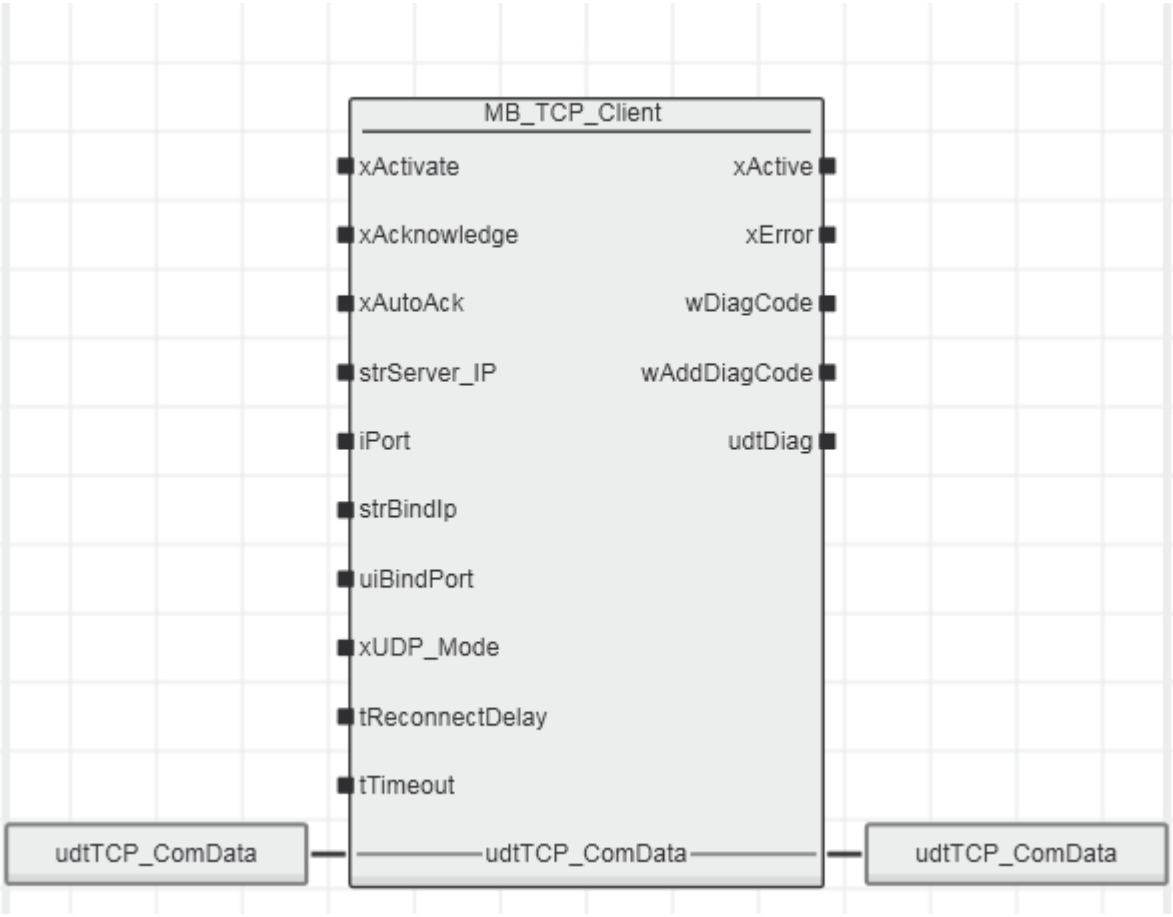
New build number: No functional changes, but changes in the MSI file (e.g. documentation update, additional examples)

4 Function blocks

Function block	Description	Version	Supported articles	License
MB_TCP_Client	The function block enables communication as client with Modbus TCP devices (server).	4	-	none
MB_TCP_Server	The function block enables communication as a server with a TCP client.	4	-	none
MB_TCP_FC1	Function block for function codes 1	3	-	none
MB_TCP_FC2	Function block for function codes 2	3	-	none
MB_TCP_FC3	Function block for function codes 3	3	-	none
MB_TCP_FC4	Function block for function codes 4	3	-	none
MB_TCP_FC5	Function block for function codes 5	1	-	none
MB_TCP_FC6	Function block for function codes 6	1	-	none
MB_TCP_FC15	Function block for function codes 15	1	-	none
MB_TCP_FC16	Function block for function codes 16	1	-	none
MB_TCP_FC23	Function block for function codes 23	4	-	none
MB_TCP_DiagInfo_EN	This optional function block displays diagnostic messages of the Modbus client as clear text in English.	2	-	none

5 MB_TCP_Client

5.1 Function block call



5.2 Input parameters

Name	Type	Description
xActivate	BOOL	Rising edge: Activates the function block. FALSE: Deactivates the function block.
xAcknowledge	BOOL	Rising edge: Error messages are deleted. The function block is not re-initialized.
xAutoAck	BOOL	Automatic acknowledgment of errors TRUE: The error is acknowledged automatically. Error data is present at the xError, wDiagCode and wAddDiagCode outputs for only one cycle.
strServer_IP	STRING	IP address of the Modbus server. The address should be specified as follows: xxx.xxx.xxx.xxx
iPort	INT	TCP port of the Modbus server. If no address is specified here, the default address 502 is used.
strBindIp	STRING	Connected to TCP_SOCKET.BIND_IP / UDP_SOCKET.BIND_IP. Defines the local IP address (IPv4 address) of the Ethernet adapter to which the created socket is bound. The selected Ethernet adapter is used for the data exchange between the network devices (useful, for example, with TCP servers for controllers that use several Ethernet adapters). Together with the port number bound to the created socket (BIND_PORT input) the IP address identifies the network interface. If the string is empty or contains the value 0.0.0.0. or the input is not connected, a suitable Ethernet adapter is selected.

uiBindPort	UINT	Connected to TCP_SOCKET.BIND_PORT / UDP_SOCKET.BIND_PORT. Defines the local port number associated with the IP address of the selected Ethernet adapter (BIND_IP input) to which the created socket is bound (necessary for setting up TCP servers). The selected port number is used by the TCP server for incoming data. If the value is 0 or the input is not connected, the stack assigns a temporary port. **UDP-Mode:** The uiBindPort of the MB_TCP_Client has to be the uiDestPort of the MB_TCP_Server.
xUDP_Mode	BOOL	FALSE: Modbus-TCP is used. TRUE: Modbus-UDP is used.
tReconnectDelay	TIME	Delay between two IP connect executions. Needed by PLC. Default value = 500ms.
tTimeout	TIME	Timeout for communication monitoring. The Modbus server must respond to a request within the time specified here, otherwise an error is triggered. If no value is specified here, the block operates with a timeout of 2 seconds.

5.3 Output parameters

Name	Type	Description
xActive	BOOL	FALSE: Function block is not active. TRUE: Function block is active.
xError	BOOL	TRUE: An error has occurred. For details refer to wDiagCode and wAddDiagCode.
wDiagCode	WORD	Diagnosis code. Refer to diagnostic table.
wAddDiagCode	WORD	Additional diagnosis code. Refer to diagnostic table.
udtDiag	MB_TCP_UDT_CLI_DIAG	Additional diagostic data.

5.4 Inout parameters

Name	Type	Description
udtTCP_ComData	MB_TCP_UDT_COMMUNICATION	Communication structure of the block family. The blocks are connected with each other by using this structure.

5.5 Diagnosis

wDiagCode	wAddDiagCode	Description
16#0000	16#0000	Function block is deactivated.
16#8000	16#0000	Function block is in regular operation.
16#C010		A parameterization error has occurred
	16#0010	The IP address of the Modbus server was not specified.
16#C020	16#0000	Timeout error
16#C030	16#xxxx	TCP_SOCKET error 16#xxxx. Refer to appendix.
16#C040	16#xxxx	TCP_SEND error 16#xxxx. Refer to appendix.
16#C050	16#xxxx	TCP_RECEIVE error 16#xxxx. Refer to appendix.
16#C410	16#0000	Connect timeout. TCP_SOCKET / UDP_SOCKET cannot connect for tTimeout. E.g. due to TCP_SOCKET / UDP_SOCKET error codes (refer to appendix): • C208: The connection was reset by the remote peer • C213: The remote host is actively refusing a connection

5.5.1 udtDiag

The udtDiag structure contains all inputs and outputs of the used firmware function blocks: For details refer to appendix "Data types".

6 MB_TCP_Server

This function block organizes the read and writes accesses of a remote control system to internal data via Modbus TCP. The data is stored in the arrModbusData block parameter, which is a WORD array with 7167 elements. Since this array has been defined as IN/OUT parameter, it can be read and written to in the local user program as required.

Attention: Depending on the control system used, the number of functions that the Modbus server can process in a given time is limited. In the event of overloading it may happen that the server does not respond to a telegram. In this case the accessing control system must send the telegram again.

Only one Modbus client can access the Modbus server at a time.

Access to Registers

The Read Holding Registers, Write Multiple Registers and Read/Write Multiple Registers Modbus functions access the WORD elements in the udt_TCP_ComData array. The array index and the register address to be specified when calling the Modbus functions are identical.

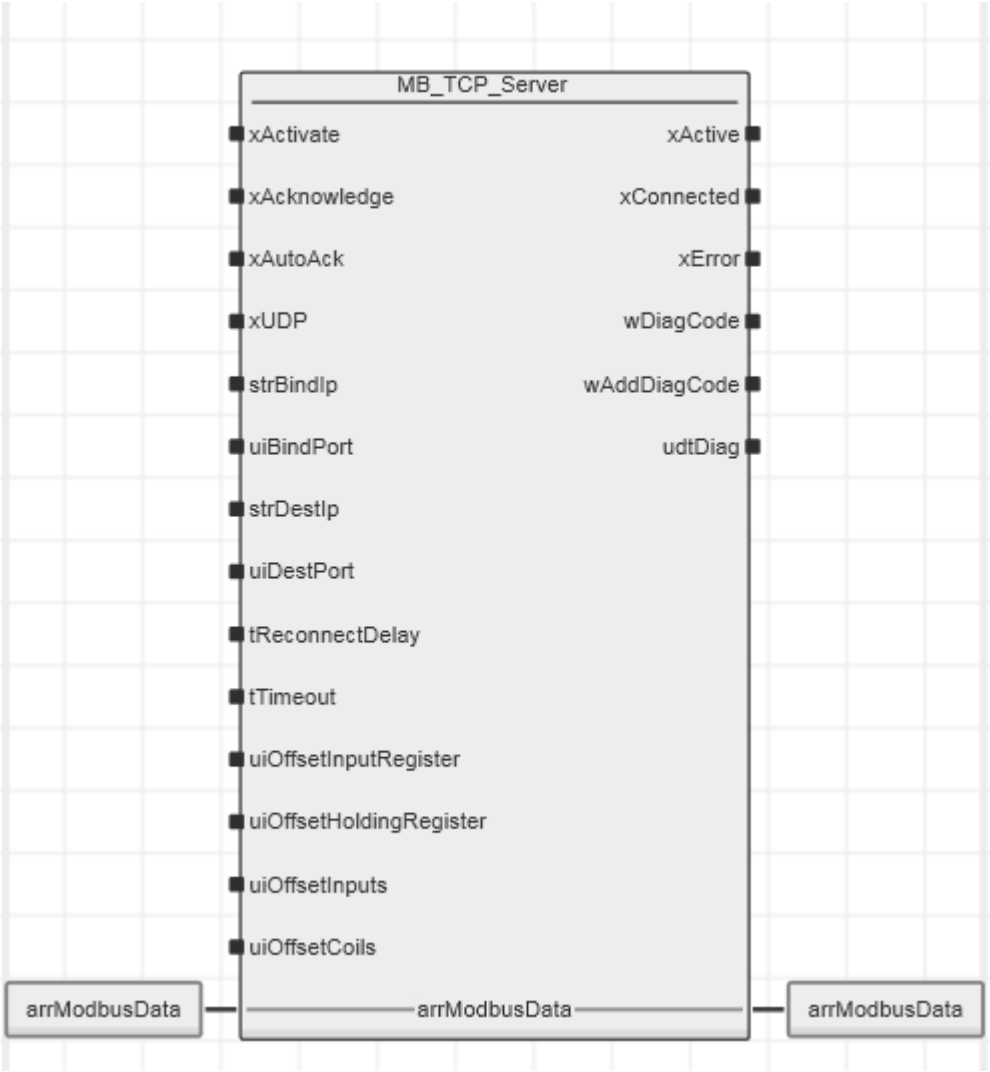
Access to internal bits

When accessing internal bits all bits existing in the udt_TCP_ComData array are numbered consecutively. These numbers can be used to read this data with the Read Coils service. The numbering scheme is shown in below. The Modbus bit number must be specified when the data is accessed via the Modbus function.

	MODBUS_DATA[0] (WORD Data Type)															
	Byte 1								Byte 0							
Bit number	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Modbus Bit number	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

	MODBUS_DATA[1] (WORD Data Type)															
	Byte 1								Byte 0							
Bit number	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Modbus Bit number	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16

6.1 Function block call



6.2 Input parameters

Name	Type	Description
xActivate	BOOL	Rising edge: Activates the function block. FALSE: Deactivates the function block.
xAcknowledge	BOOL	Rising edge: Error messages are deleted. The function block is not re-initialized.
xAutoAck	BOOL	Automatic acknowledgment of errors TRUE: The error is acknowledged automatically. Error data is present at the xError, wDiagCode and wAddDiagCode outputs for only one cycle.
xUDP	BOOL	Selection of communication protocol. FALSE: Modbus-TCP is used TRUE: Modbus-UDP is used

strBindIp	STRING	Connected to TCP_SOCKET.BIND_IP / UDP_SOCKET.BIND_IP. Defines the local IP address (IPv4 address) of the Ethernet adapter to which the created socket is bound. The selected Ethernet adapter is used for the data exchange between the network devices (useful, for example, with TCP servers for controllers that use several Ethernet adapters). Together with the port number bound to the created socket (BIND_PORT input) the IP address identifies the network interface. If the string is empty or contains the value 0.0.0.0. or the input is not connected, a suitable Ethernet adapter is selected.
uiBindPort	UINT	Connected to TCP_SOCKET.BIND_PORT / UDP_SOCKET.BIND_PORT. Defines the local port number associated with the IP address of the selected Ethernet adapter (BIND_IP input) to which the created socket is bound (necessary for setting up TCP servers). The selected port number is used by the TCP server for incoming data. If the value is 0 or the input is not connected, the stack assigns a temporary port.
strDestIp	STRING	**TCP-Mode:** • Connected to TCP_SOCKET.DEST_IP. • FB creates a listening socket, only requests from the client with this IP address are accepted. If the string is empty, contains the value 0.0.0.0. or the input is not connected, any client IP address is accepted. **UDP-Mode:** • Connected to UDP_SEND.DEST_IP • The strDestIp has to be the IP address of the client.
uiDestPort	UINT	**TCP-Mode:** • Connected to TCP_SOCKET.DEST_PORT. • FB creates a listening socket, only requests from the client with this IP port are accepted. If the string is empty, contains the value 0 or the input is not connected, any client IP port is accepted. **UDP-Mode:** • Connected to UDP_SEND.DEST_PORT • The uiDestPort has to be the uiBindPort of the MB_TCP_Client.
tReconnectDelay	TIME	Delay between two IP connect executions. Needed by PLC. Default value = 500ms.
tTimeout	Time	Timeout monitoring when there is a write access. After a write function was executed, a write telegram must be received again within the time specified here. A timeout error is reported if this is not the case. Specify the timeout value T#0s to disable this function. The following variables are used to define an offset for different data access starting with element 0 of arrModbusData. The offset hat no effect on the element address.
uiOffsetInputRegister	UINT	Virtual address of the first input register. Example: The value 1000 means that the first Modbus-Register has the address 1000. Required for FC4.

uiOffsetHoldingRegister	UINT	Virtual address of the first holding register. Example: The value 1000 means that the first Modbus-Register has the address 1000. Required for FC3, FC6, FC16.
uiOffsetInputs	UINT	Register-address of the first Input (0 to 7167). Required for FC2.
uiOffsetCoils	UINT	Register-address of the first Coil (0 to 7167). Required for FC1, FC5, FC15.

6.3 Output parameters

Name	Type	Description
xActive	BOOL	FALSE: Function block is not active. TRUE: Function block is active.
xConnected	BOOL	TRUE: There is a connection to a Modbus client.
xError	BOOL	TRUE: An error has occurred. For details refer to wDiagCode and wAddDiagCode.
wDiagCode	WORD	Diagnosis code. Refer to diagnostic table.
wAddDiagCode	WORD	Additional diagnosis code. Refer to diagnostic table.
udtDiag	MB_TCP_UDT_SER_DIAG	Additional diagostic data.

6.4 Inout parameters

Name	Type	Description
arrModbusData	MB_TCP_ARR_W_0_7167	WORD array First index: 0 Last index: 7167 This array contains the Modbus registers.

6.5 Diagnosis

wDiagCode	wAddDiagCode	Description
16#0000	16#0000	Function block is deactivated.
16#8000	16#0000	Function block is in regular operation.
16#C010		Wrong Parameter
	16#0010	Invalid parameter at uiOffsetInputs (> 58368)
	16#0020	Invalid parameter at uiOffsetCoils (> 58368)
	16#0030	Invalid parameter at uiOffsetHoldingRegister (> 58368)
	16#0040	Invalid parameter at uiOffsetInputRegister. (> 58368)
16#C020	16#0000	Timeout error. A write services was executed and another write telegram was not received within the time specified in the tTimeout parameter. Enter the value T#0s in the tTimeout parameter to disable this function.
16#C030	16#xxxx	TCP_SOCKET error 16#xxxx. Refer to appendix.
16#C040	16#xxxx	TCP_SEND error 16#xxxx. Refer to appendix.
16#C050	16#xxxx	TCP_RECEIVE error 16#xxxx. Refer to appendix.

6.5.1 udtDiag

The udtDiag structure contains all inputs and outputs of the used firmware function blocks. For details refer to appendix "Data types".

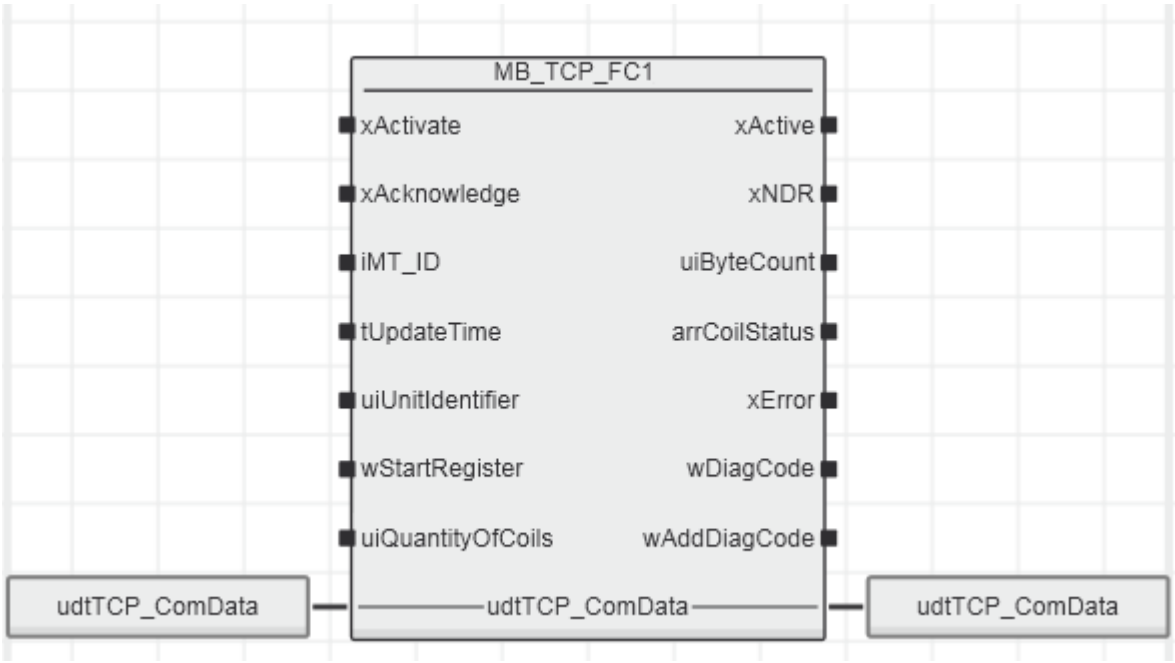
7 MB_TCP_FC1

This function block reads digital inputs from a Modbus slave.

The data is read in the form of a byte array (CoilStatus parameter), the first bit in the first byte of this array corresponds to the first output to be read. The following diagram shows the assignment of the bits in the byte array to the outputs and the internal bits, in this example output 1 to 16 are read:

	Byte 1 (Coil Status)								Byte 2 (Coil Status)							
Bit number	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Bit number	8	7	6	5	4	3	2	1	16	15	14	13	12	11	10	9

7.1 Function block call



7.2 Input parameters

Name	Type	Description
xActivate	BOOL	Rising edge: Activates the function block. FALSE: Deactivates the function block.
xAcknowledge	BOOL	Rising edge: Error messages are deleted. The function block is not re-initialized.
iMT_ID	INT	Value in the range from 1 to 10. Represents the connection between the blocks and may only be assigned once per MB_TCP_Client. This parameter should be static.
tUpdateTime	TIME	Waiting time in ms between two read tasks.
uiUnitIdentifier	UINT	Parameter for addressing of a Modbus RTU device, which is connected via a gateway
wStartRegister	WORD	Address of the first bit / output / register to be read.
uiQuantityOfCoils	UINT	Number of bits to be read Valid value range: 1 - 2000

7.3 Output parameters

Name	Type	Description
xActive	BOOL	FALSE: Function block is not active. TRUE: Function block is active.
xNDR	BOOL	TRUE for one cycle: New data has been received.
uiByteCount	UINT	Number of bytes received.
arrCoilStatus	MB_TCP_ARR_B_1_260	Status of the outputs/internal bits. BYTE array with the limits 1 to 250. The first output corresponds to the first bit of the first byte.
xError	BOOL	TRUE: An error has occurred. For details refer to wDiagCode and wAddDiagCode.
wDiagCode	WORD	Diagnosis code. Refer to diagnostic table.
wAddDiagCode	WORD	Additional diagnosis code. Refer to diagnostic table.

7.4 Inout parameters

Name	Type	Description
udtTCP_ComData	MB_TCP_UDT_COMMUNICATION	Communication structure of the block family. The blocks are connected with each other by using this structure.

7.5 Diagnosis

wDiagCode	wAddDiagCode	Description
16#0000	16#0000	Function block is deactivated.
16#8000	16#0000	Function block is in regular operation.
16#C010	16#0000	The MB_TCP_Client block is not ready.
16#C030	16#0000	The iMT_ID used is invalid. The iMT_ID must be in the range from 1 to 10.
16#C040	16#0000	The iMT_ID used is assigned twice. Several Modbus function blocks use the same iMT_ID, in this case only the first called block operates, all other blocks indicate this error.
16#C060		Modbus error
	16#0001	Illegal function
	16#0002	Illegal data address
	16#0003	Illegal data value
	16#0004	Slave device failure
	16#0005	Acknowledge
	16#0006	Slave device busy
	16#0008	Memory parity error
	16#000A	Gateway path unavailable
	16#000B	Gateway target device failed to respond

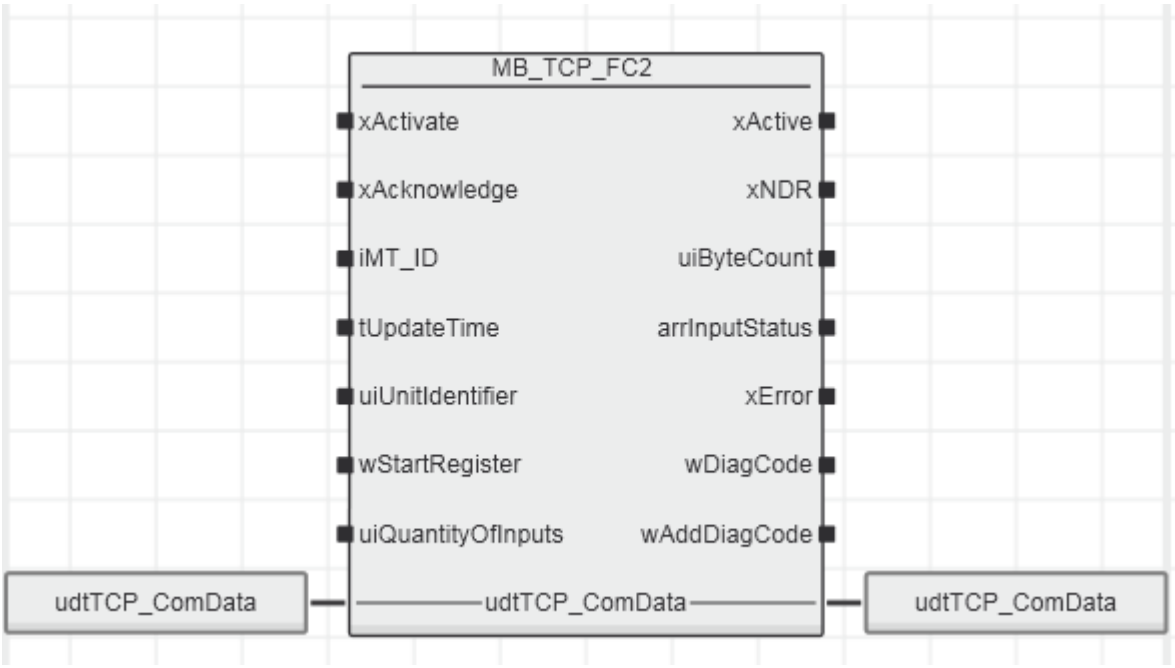
8 MB_TCP_FC2

This function block reads digital inputs from a Modbus slave.

The data is read in the form of a byte array (InputStatus parameter), the first bit in the first byte of this array corresponds to the first input to be read. The following diagram shows the assignment of the bits in the byte array to the inputs in this example inputs 1 to 16 are read:

	Byte 1 (Input Status)								Byte 2 (Input Status)							
Bit number	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Bit number	8	7	6	5	4	3	2	1	16	15	14	13	12	11	10	9

8.1 Function block call



8.2 Input parameters

Name	Type	Description
xActivate	BOOL	Rising edge: Activates the function block. FALSE: Deactivates the function block.
xAcknowledge	BOOL	Rising edge: Error messages are deleted. The function block is not re-initialized.
iMT_ID	INT	Value in the range from 1 to 10. Represents the connection between the blocks and may only be assigned once per MB_TCP_Client. This parameter should be static.
tUpdateTime	TIME	Waiting time in ms between two read tasks.
uiUnitIdentifier	UINT	Parameter for addressing of a Modbus RTU device, which is connected via a gateway
wStartRegister	WORD	Address of the first bit / output / register to be read.

8.3 Output parameters

Name	Type	Description
xActive	BOOL	FALSE: Function block is not active. TRUE: Function block is active.
xNDR	BOOL	TRUE for one cycle: New data has been received.
uiByteCount	UINT	Number of bytes received.
arrInputStatus	MB_TCP_ARR_B_1_250	Status of the inputs. BYTE array with the limits 1 to 250. Each input is represented by a bit, the first input corresponds to the first bit of the first byte.
xError	BOOL	TRUE: An error has occurred. For details refer to wDiagCode and wAddDiagCode.
wDiagCode	WORD	Diagnosis code. Refer to diagnostic table.
wAddDiagCode	WORD	Additional diagnosis code. Refer to diagnostic table.

8.4 Inout parameters

Name	Type	Description
udtTCP_ComData	MB_TCP_UDT_COMMUNICATION	Communication structure of the block family. The blocks are connected with each other by using this structure.

8.5 Diagnosis

wDiagCode	wAddDiagCode	Description
16#0000	16#0000	Function block is deactivated.
16#8000	16#0000	Function block is in regular operation.
16#C010	16#0000	The MB_TCP_Client block is not ready.
16#C030	16#0000	The iMT_ID used is invalid. The iMT_ID must be in the range from 1 to 10.
16#C040	16#0000	The iMT_ID used is assigned twice. Several Modbus function blocks use the same iMT_ID, in this case only the first called block operates, all other blocks indicate this error.
16#C060		Modbus error
	16#0001	Illegal function
	16#0002	Illegal data address
	16#0003	Illegal data value
	16#0004	Slave device failure
	16#0005	Acknowledge
	16#0006	Slave device busy
	16#0008	Memory parity error
	16#000A	Gateway path unavailable
	16#000B	Gateway target device failed to respond

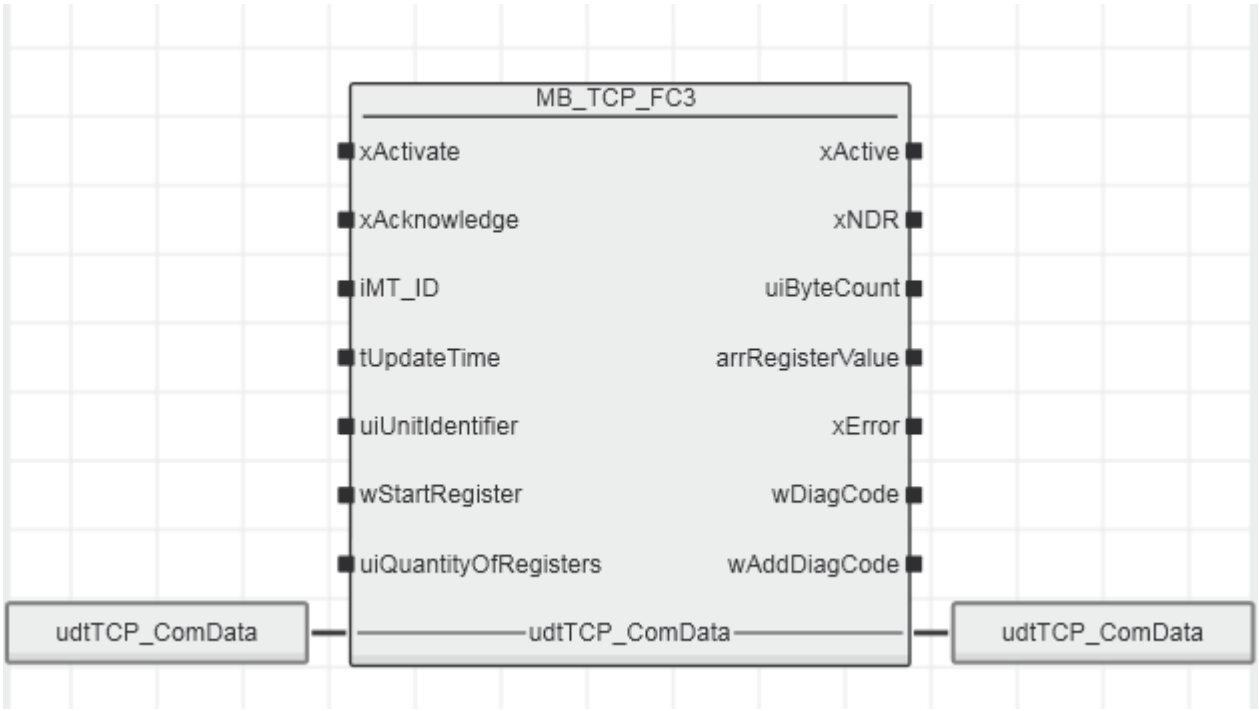
9 MB_TCP_FC3

This function block reads holding register from a Modbus slave.

The data is read in the form of a byte array (InputStatus parameter), the first bit in the first byte of this array corresponds to the first input to be read. The following diagram shows the assignment of the bits in the byte array to the inputs in this example inputs 1 to 16 are read:

	Byte 1 (InputStatus)								Byte 2 (InputStatus)							
Bit number	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Bit number	8	7	6	5	4	3	2	1	16	15	14	13	12	11	10	9

9.1 Function block call



9.2 Input parameters

Name	Type	Description
xActivate	BOOL	Rising edge: Activates the function block. FALSE: Deactivates the function block.
xAcknowledge	BOOL	Rising edge: Error messages are deleted. The function block is not re-initialized.
iMT_ID	INT	Value in the range from 1 to 10. Represents the connection between the blocks and may only be assigned once per MB_TCP_Client. This parameter should be static.
tUpdateTime	TIME	Waiting time in ms between two read tasks.
uiUnitIdentifier	UINT	Parameter for addressing of a Modbus RTU device, which is connected via a gateway
wStartRegister	WORD	Address of the first bit / output / register to be read.
uiQuantityOfRegisters	UINT	Number of registers to be read. Valid value range: 1 - 125

9.3 Output parameters

Name	Type	Description
xActive	BOOL	FALSE: Function block is not active. TRUE: Function block is active.
xNDR	BOOL	TRUE for one cycle: New data has been received.
uiByteCount	UINT	Number of bytes received.
arrRegisterValue	MB_TCP_ARR_W_1_125	Values read from the registers. WORD array with the limits 1 to 125.
xError	BOOL	TRUE: An error has occurred. For details refer to wDiagCode and wAddDiagCode.
wDiagCode	WORD	Diagnosis code. Refer to diagnostic table.
wAddDiagCode	WORD	Additional diagnosis code. Refer to diagnostic table.

9.4 Inout parameters

Name	Type	Description
udtTCP_ComData	MB_TCP_UDT_COMMUNICATION	Communication structure of the block family. The blocks are connected with each other by using this structure.

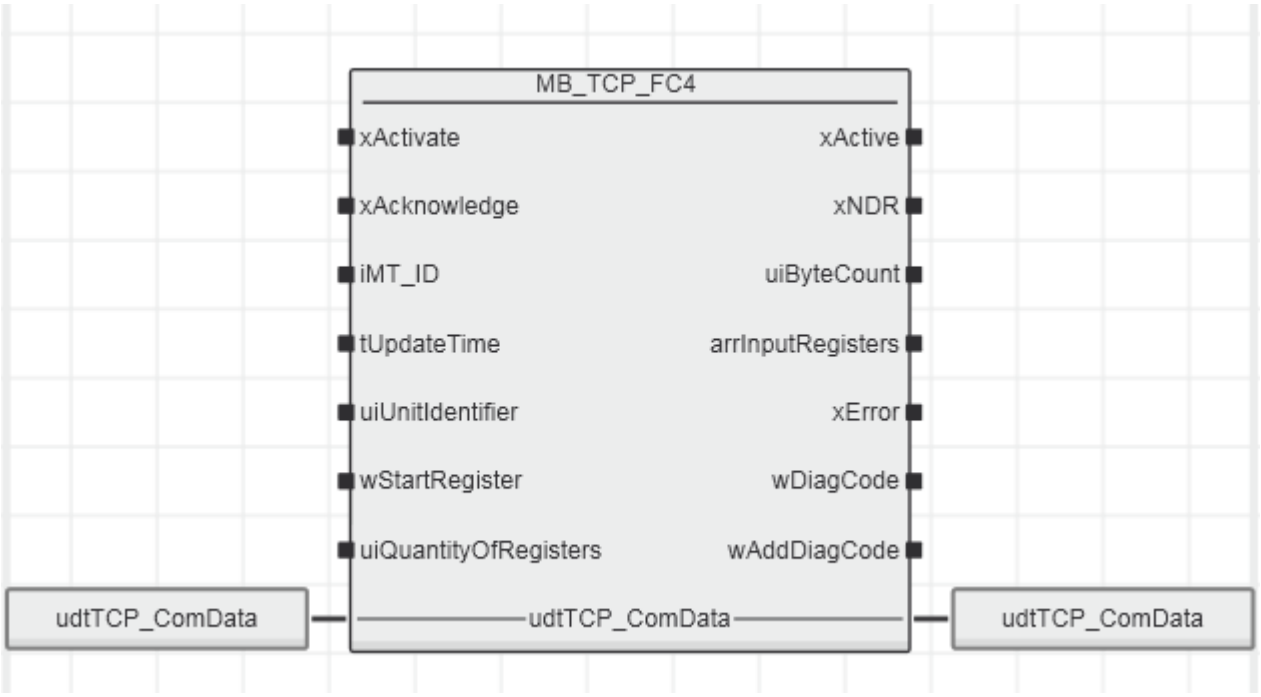
9.5 Diagnosis

wDiagCode	wAddDiagCode	Description
16#0000	16#0000	Function block is deactivated.
16#8000	16#0000	Function block is in regular operation.
16#C010	16#0000	The MB_TCP_Client block is not ready.
16#C030	16#0000	The iMT_ID used is invalid. The iMT_ID must be in the range from 1 to 10.
16#C040	16#0000	The iMT_ID used is assigned twice. Several Modbus function blocks use the same iMT_ID, in this case only the first called block operates, all other blocks indicate this error.
16#C060		Modbus error
	16#0001	Illegal function
	16#0002	Illegal data address
	16#0003	Illegal data value
	16#0004	Slave device failure
	16#0005	Acknowledge
	16#0006	Slave device busy
	16#0008	Memory parity error
	16#000A	Gateway path unavailable
	16#000B	Gateway target device failed to respond

10 MB_TCP_FC4

This function block reads digital registers from a Modbus slave.

10.1 Function block call



10.2 Input parameters

Name	Type	Description
xActivate	BOOL	Rising edge: Activates the function block. FALSE: Deactivates the function block.
xAcknowledge	BOOL	Rising edge: Error messages are deleted. The function block is not re-initialized.
iMT_ID	INT	Value in the range from 1 to 10. Represents the connection between the blocks and may only be assigned once per MB_TCP_Client. This parameter should be static.
tUpdateTime	TIME	Waiting time in ms between two read tasks.
uiUnitIdentifier	UINT	Parameter for addressing of a Modbus RTU device, which is connected via a gateway
wStartRegister	WORD	Address of the first bit / output / register to be read.
uiQuantityOfRegisters	UINT	Number of registers to be read. Valid value range: 1 - 125

10.3 Output parameters

Name	Type	Description
xActive	BOOL	FALSE: Function block is not active. TRUE: Function block is active.
xNDR	BOOL	TRUE for one cycle: New data has been received.
uiByteCount	UINT	Number of bytes received.
arrInputRegisters	MB_TCP_ARR_W_1_125	Values read from the registers. WORD array with the limits 1 to 125.
xError	BOOL	TRUE: An error has occurred. For details refer to wDiagCode and wAddDiagCode.
wDiagCode	WORD	Diagnosis code. Refer to diagnostic table.
wAddDiagCode	WORD	Additional diagnosis code. Refer to diagnostic table.

10.4 Inout parameters

Name	Type	Description
udtTCP_ComData	MB_TCP_UDT_COMMUNICATION	Communication structure of the block family. The blocks are connected with each other by using this structure.

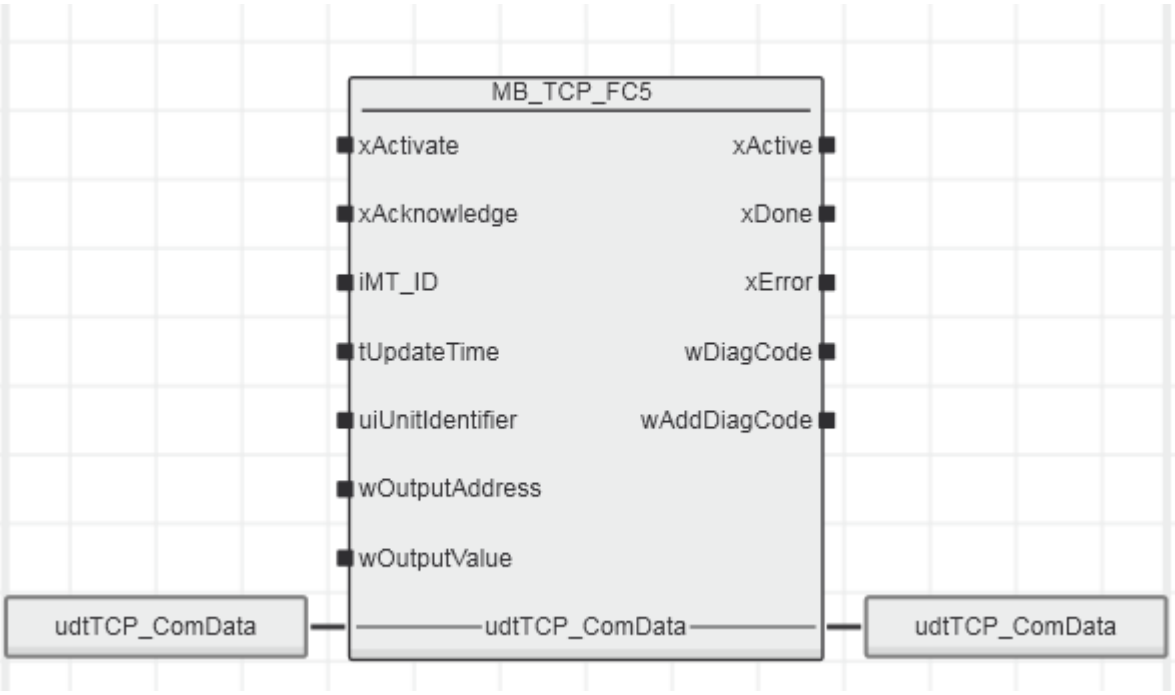
10.5 Diagnosis

wDiagCode	wAddDiagCode	Description
16#0000	16#0000	Function block is deactivated.
16#8000	16#0000	Function block is in regular operation.
16#C010	16#0000	The MB_TCP_Client block is not ready.
16#C030	16#0000	The iMT_ID used is invalid. The iMT_ID must be in the range from 1 to 10.
16#C040	16#0000	The iMT_ID used is assigned twice. Several Modbus function blocks use the same iMT_ID, in this case only the first called block operates, all other blocks indicate this error.
16#C060		Modbus error
	16#0001	Illegal function
	16#0002	Illegal data address
	16#0003	Illegal data value
	16#0004	Slave device failure
	16#0005	Acknowledge
	16#0006	Slave device busy
	16#0008	Memory parity error
	16#000A	Gateway path unavailable
	16#000B	Gateway target device failed to respond

11 MB_TCP_FC5

This function block writes a bit in the memory of a Modbus slave.

11.1 Function block call



11.2 Input parameters

Name	Type	Description
xActivate	BOOL	Rising edge: Activates the function block. FALSE: Deactivates the function block.
xAcknowledge	BOOL	Rising edge: Error messages are deleted. The function block is not re-initialized.
iMT_ID	INT	Value in the range from 1 to 10. Represents the connection between the blocks and may only be assigned once per MB_TCP_Client. This parameter should be static.
tUpdateTime	TIME	Waiting time in ms between two read tasks.
uiUnitIdentifier	UINT	Parameter for addressing of a Modbus RTU device, which is connected via a gateway
wOutputAddress	WORD	Address of the bit/output to be written. During addressing, the address of the first digital output is zero.
wOutputValue	WORD	Value to be written. Only the following two values are permitted: TRUE: 16#FF00 FALSE: 16#0000

11.3 Output parameters

Name	Type	Description
xActive	BOOL	FALSE: Function block is not active. TRUE: Function block is active.
xDone	BOOL	TRUE: Request was sent and the response from communication partner received successfully.
xError	BOOL	TRUE: An error has occurred. For details refer to wDiagCode and wAddDiagCode.
wDiagCode	WORD	Diagnosis code. Refer to diagnostic table.
wAddDiagCode	WORD	Additional diagnosis code. Refer to diagnostic table.

11.4 Inout parameters

Name	Type	Description
udtTCP_ComData	MB_TCP_UDT_COMMUNICATION	Communication structure of the block family. The blocks are connected with each other by using this structure.

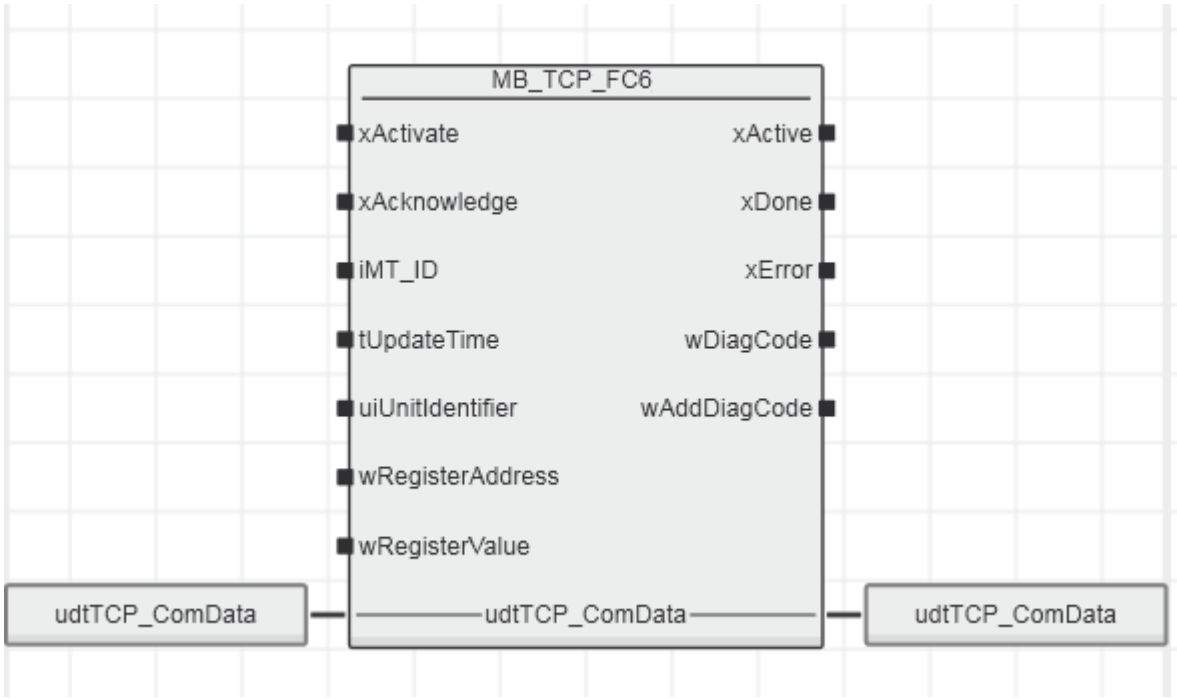
11.5 Diagnosis

wDiagCode	wAddDiagCode	Description
16#0000	16#0000	Function block is deactivated.
16#8000	16#0000	Function block is in regular operation.
16#C010	16#0000	The MB_TCP_Client block is not ready.
16#C030	16#0000	The iMT_ID used is invalid. The iMT_ID must be in the range from 1 to 10.
16#C040	16#0000	The iMT_ID used is assigned twice. Several Modbus function blocks use the same iMT_ID, in this case only the first called block operates, all other blocks indicate this error.
16#C060		Modbus error
	16#0001	Illegal function
	16#0002	Illegal data address
	16#0003	Illegal data value
	16#0004	Slave device failure
	16#0005	Acknowledge
	16#0006	Slave device busy
	16#0008	Memory parity error
	16#000A	Gateway path unavailable
	16#000B	Gateway target device failed to respond

12 MB_TCP_FC6

This function block writes a register into the memory of a Modbus slave.

12.1 Function block call



12.2 Input parameters

Name	Type	Description
xActivate	BOOL	Rising edge: Activates the function block. FALSE: Deactivates the function block.
xAcknowledge	BOOL	Rising edge: Error messages are deleted. The function block is not re-initialized.
iMT_ID	INT	Value in the range from 1 to 10. Represents the connection between the blocks and may only be assigned once per MB_TCP_Client. This parameter should be static.
tUpdateTime	TIME	Waiting time in ms between two read tasks.
uiUnitIdentifier	UINT	Parameter for addressing of a Modbus RTU device, which is connected via a gateway
wRegisterAddress	WORD	Address of the register to be written.
wRegisterValue	WORD	Value to be written to the register.

12.3 Output parameters

Name	Type	Description
xActive	BOOL	FALSE: Function block is not active. TRUE: Function block is active.
xDone	BOOL	TRUE: Request was sent and the response from communication partner received successfully.
xError	BOOL	TRUE: An error has occurred. For details refer to wDiagCode and wAddDiagCode.
wDiagCode	WORD	Diagnosis code. Refer to diagnostic table.
wAddDiagCode	WORD	Additional diagnosis code. Refer to diagnostic table.

12.4 Inout parameters

Name	Type	Description
udtTCP_ComData	MB_TCP_UDT_COMMUNICATION	Communication structure of the block family. The blocks are connected with each other by using this structure.

12.5 Diagnosis

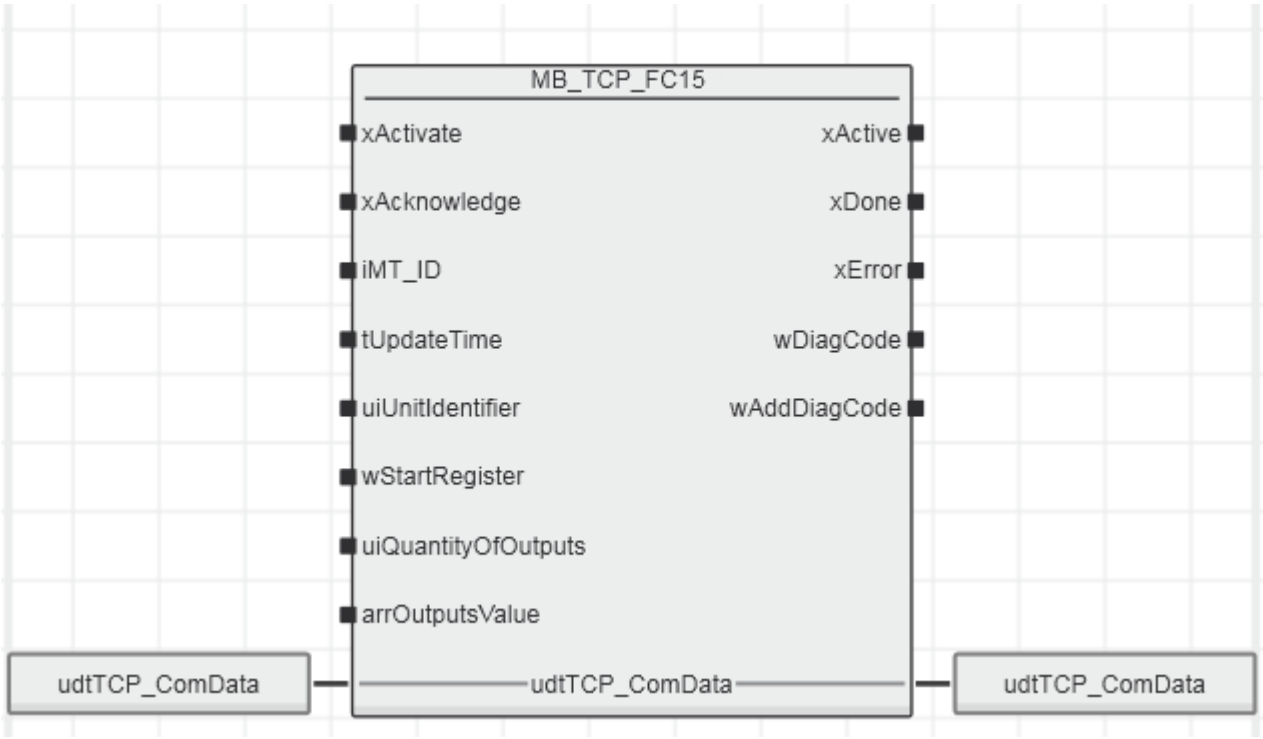
wDiagCode	wAddDiagCode	Description
16#0000	16#0000	Function block is deactivated.
16#8000	16#0000	Function block is in regular operation.
16#C010	16#0000	The MB_TCP_Client block is not ready.
16#C030	16#0000	The iMT_ID used is invalid. The iMT_ID must be in the range from 1 to 10.
16#C040	16#0000	The iMT_ID used is assigned twice. Several Modbus function blocks use the same iMT_ID, in this case only the first called block operates, all other blocks indicate this error.
16#C060		Modbus error
	16#0001	Illegal function
	16#0002	Illegal data address
	16#0003	Illegal data value
	16#0004	Slave device failure
	16#0005	Acknowledge
	16#0006	Slave device busy
	16#0008	Memory parity error
	16#000A	Gateway path unavailable
	16#000B	Gateway target device failed to respond

13 MB_TCP_FC15

This function block writes multiple bits into the memory of the Modbus slave. The data is read in the form of a byte array, the first bit in the first byte of this array corresponds to the first output to be written. The following diagram shows the assignment of the bits in the byte array to the outputs; in this example outputs 1 to 16 are written:

	Byte 1 (Output value)								Byte 2 (Output value)							
Bit number	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Bit number	8	7	6	5	4	3	2	1	16	15	14	13	12	11	10	9

13.1 Function block call



13.2 Input parameters

Name	Type	Description
xActivate	BOOL	Rising edge: Activates the function block. FALSE: Deactivates the function block.
xAcknowledge	BOOL	Rising edge: Error messages are deleted. The function block is not re-initialized.
iMT_ID	INT	Value in the range from 1 to 10. Represents the connection between the blocks and may only be assigned once per MB_TCP_Client. This parameter should be static.
tUpdateTime	TIME	Waiting time in ms between two read tasks.
uiUnitIdentifier	UINT	Parameter for addressing of a Modbus RTU device, which is connected via a gateway
wStartRegister	WORD	Address of the first bit / output / register to be read.
uiQuantityOfOutputs	UINT	Number of outputs to be written. Valid value range: 1 - 1000
wOutputsValue	MB_TCP_ARR_W_1_125	Values that are to be written to the outputs. BYTE array with the limits 1 to 250. Each output is represented by a bit; the first output corresponds to the first bit of the first byte.

13.3 Output parameters

Name	Type	Description
xActive	BOOL	FALSE: Function block is not active. TRUE: Function block is active.
xDone	BOOL	TRUE: Request was sent and the response from communication partner received successfully.
xError	BOOL	TRUE: An error has occurred. For details refer to wDiagCode and wAddDiagCode.
wDiagCode	WORD	Diagnosis code. Refer to diagnostic table.
wAddDiagCode	WORD	Additional diagnosis code. Refer to diagnostic table.

13.4 Inout parameters

Name	Type	Description
udtTCP_ComData	MB_TCP_UDT_COMMUNICATION	Communication structure of the block family. The blocks are connected with each other by using this structure.

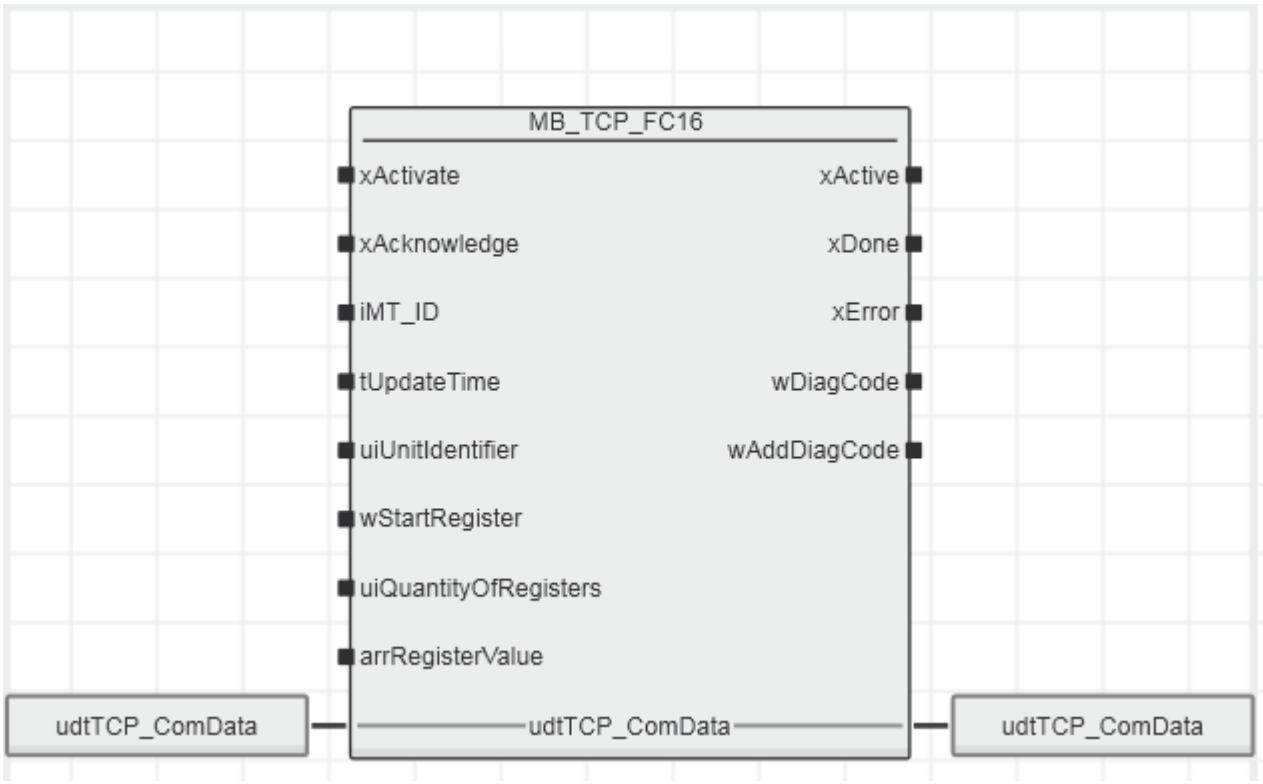
13.5 Diagnosis

wDiagCode	wAddDiagCode	Description
16#0000	16#0000	Function block is deactivated.
16#8000	16#0000	Function block is in regular operation.
16#C010	16#0000	The MB_TCP_Client block is not ready.
16#C030	16#0000	The iMT_ID used is invalid. The iMT_ID must be in the range from 1 to 10.
16#C040	16#0000	The iMT_ID used is assigned twice. Several Modbus function blocks use the same iMT_ID, in this case only the first called block operates, all other blocks indicate this error.
16#C060		Modbus error
	16#0001	Illegal function
	16#0002	Illegal data address
	16#0003	Illegal data value
	16#0004	Slave device failure
	16#0005	Acknowledge
	16#0006	Slave device busy
	16#0008	Memory parity error
	16#000A	Gateway path unavailable
	16#000B	Gateway target device failed to respond

14 MB_TCP_FC16

This function block writes multiple registers into the memory of a Modbus slave.

14.1 Function block call



14.2 Input parameters

Name	Type	Description
xActivate	BOOL	Rising edge: Activates the function block. FALSE: Deactivates the function block.
xAcknowledge	BOOL	Rising edge: Error messages are deleted. The function block is not re-initialized.
iMT_ID	INT	Value in the range from 1 to 10. Represents the connection between the blocks and may only be assigned once per MB_TCP_Client. This parameter should be static.
tUpdateTime	TIME	Waiting time in ms between two read tasks.
uiUnitIdentifier	UINT	Parameter for addressing of a Modbus RTU device, which is connected via a gateway
wStartingAddress	WORD	Address of the first register to be written.
uiQuantityOfRegisters	UINT	Number of registers to be read. Valid value range: 1 - 125
arrRegisterValue	MB_TCP_ARR_W_1_125	Values read from the registers. WORD array with the limits 1 to 125.

14.3 Output parameters

Name	Type	Description
xActive	BOOL	FALSE: Function block is not active. TRUE: Function block is active.
xDone	BOOL	TRUE: Request was sent and the response from communication partner received successfully.
xError	BOOL	TRUE: An error has occurred. For details refer to wDiagCode and wAddDiagCode.
wDiagCode	WORD	Diagnosis code. Refer to diagnostic table.
wAddDiagCode	WORD	Additional diagnosis code. Refer to diagnostic table.

14.4 Inout parameters

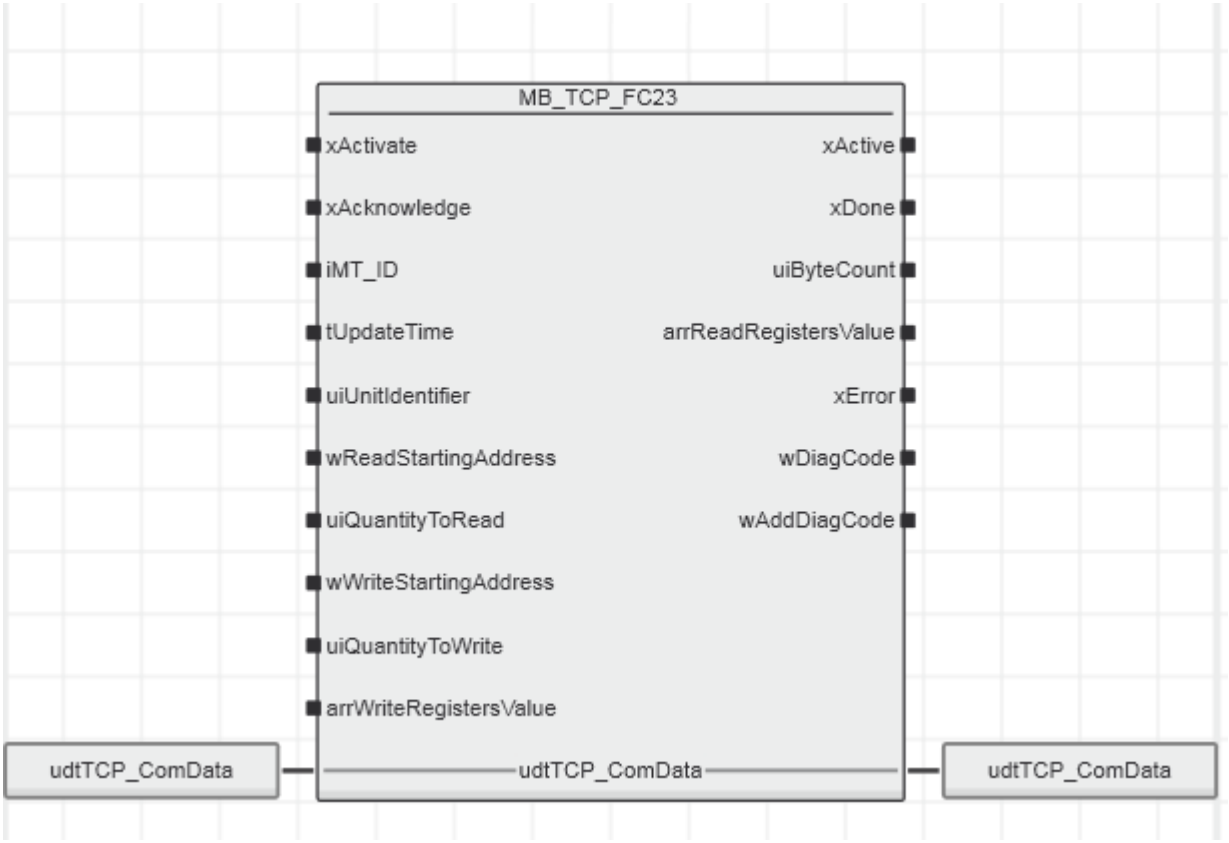
Name	Type	Description
udtTCP_ComData	MB_TCP_UDT_COMMUNICATION	Communication structure of the block family. The blocks are connected with each other by using this structure.

14.5 Diagnosis

wDiagCode	wAddDiagCode	Description
16#0000	16#0000	Function block is deactivated.
16#8000	16#0000	Function block is in regular operation.
16#C010	16#0000	The MB_TCP_Client block is not ready.
16#C030	16#0000	The iMT_ID used is invalid. The iMT_ID must be in the range from 1 to 10.
16#C040	16#0000	The iMT_ID used is assigned twice. Several Modbus function blocks use the same iMT_ID, in this case only the first called block operates, all other blocks indicate this error.
16#C060		Modbus error
	16#0001	Illegal function
	16#0002	Illegal data address
	16#0003	Illegal data value
	16#0004	Slave device failure
	16#0005	Acknowledge
	16#0006	Slave device busy
	16#0008	Memory parity error
	16#000A	Gateway path unavailable
	16#000B	Gateway target device failed to respond

15 MB_TCP_FC23

15.1 Function block call



15.2 Input parameters

Name	Type	Description
xActivate	BOOL	Rising edge: Activates the function block. FALSE: Deactivates the function block.
xAcknowledge	BOOL	Rising edge: Error messages are deleted. The function block is not re-initialized.
iMT_ID	INT	Value in the range from 1 to 10. Represents the connection between the blocks and may only be assigned once per MB_TCP_Client. This parameter should be static.
tUpdateTime	TIME	Waiting time in ms between two read tasks.
uiUnitIdentifier	UINT	Parameter for addressing of a Modbus RTU device, which is connected via a gateway
wReadStartingAddress	WORD	Address of the first register to be read.
uiQuantityToRead	UINT	Number of registers to be read (1..125).
wWriteStartingAddress	WORD	Address of the first register to be written.
uiQuantityToWrite	UINT	Number of registers to be written (1..121).
arrWriteRegistersValue	MB_TCP_ARR_W_1_125	Values to be written to the register. WORD array with the limits 1 to 125.

15.3 Output parameters

Name	Type	Description
xActive	BOOL	FALSE: Function block is not active. TRUE: Function block is active.
xDone	BOOL	The data was sent and received successfully. The parameter is TRUE in a cycle.
uiByteCount	UINT	Number of bytes received.
arrReadRegistersValue	MB2_TCP_ARR_W_1_125	Values read from the registers. WORD array with the limits 1 to 125.
xError	BOOL	TRUE: An error has occurred. For details refer to wDiagCode and wAddDiagCode.
wDiagCode	WORD	Diagnosis code. Refer to diagnostic table.
wAddDiagCode	WORD	Additional diagnosis code. Refer to diagnostic table.

15.4 Inout parameters

Name	Type	Description
udtTCP_ComData	MB_TCP_UDT_COMMUNICATION	Communication structure of the block family. The blocks are connected with each other by using this structure.

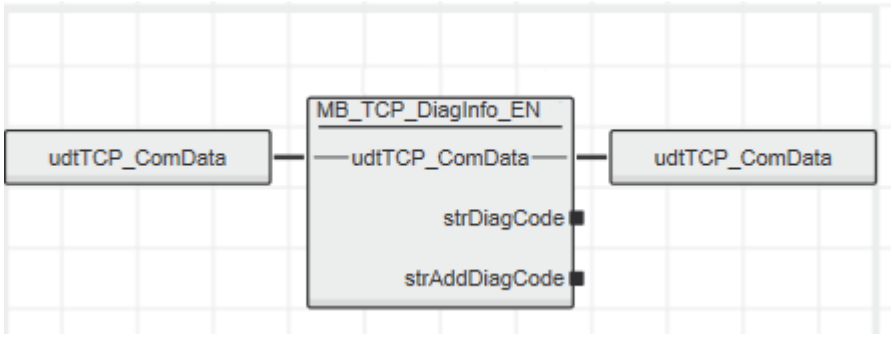
15.5 Diagnosis

wDiagCode	wAddDiagCode	Description
16#0000	16#0000	Function block is deactivated.
16#8000	16#0000	Function block is in regular operation.
16#C010	16#0000	The MB_TCP_Client block is not ready.
16#C030	16#0000	The iMT_ID used is invalid. The iMT_ID must be in the range from 1 to 10.
16#C040	16#0000	The iMT_ID used is assigned twice. Several Modbus function blocks use the same iMT_ID, in this case only the first called block operates, all other blocks indicate this error.
16#C060		Modbus error
	16#0001	Illegal function
	16#0002	Illegal data address
	16#0003	Illegal data value
	16#0004	Slave device failure
	16#0005	Acknowledge
	16#0006	Slave device busy
	16#0008	Memory parity error
	16#000A	Gateway path unavailable
	16#000B	Gateway target device failed to respond

16 MB_TCP_DiagInfo_EN

If there is an error, this block shows the diagnostics of the master block as a text in English. The source code of the block can be read and modified. To show the diagnostic messages in other languages, copy the block and translate the diagnostic text into the desired language. The text outputs (strDiagCode and strAddDiagCode) are limited to 80 characters.

16.1 Function block call



16.2 Input parameters

None

16.3 Output parameters

Name	Type	Description
strDiagCode	STRING	If there is an error, the variable shows the description for the current wDiagCode in English.
strAddDiagCode	STRING	If there is an error, the variable shows the description for the current wAddDiagCode in English.

16.4 Inout parameters

Name	Type	Description
udtTCP_ComData	MB_TCP_UDT_COMMUNICATION	Communication structure of the block family. The blocks are connected with each other by using this structure.

16.5 Diagnosis

None

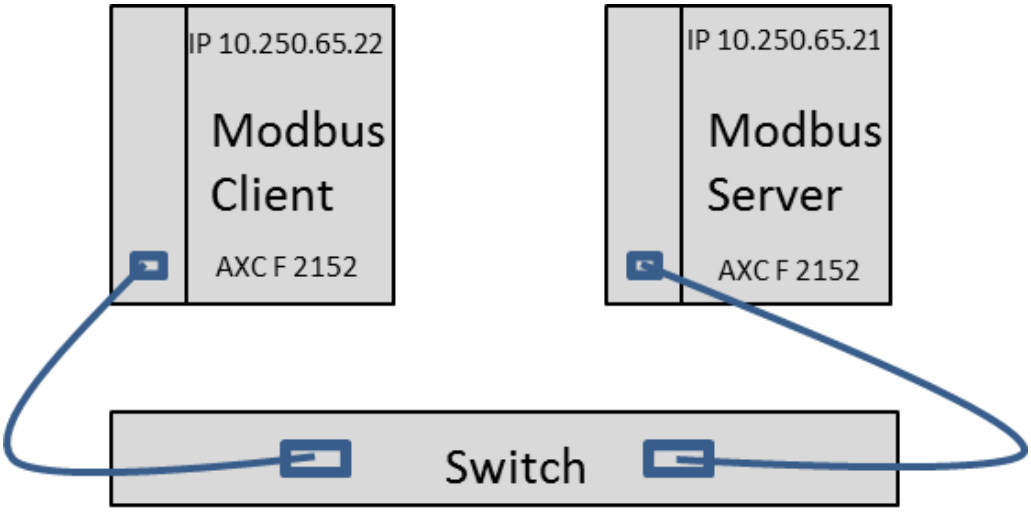
17 Startup examples

For the startup instructions of the Modbus_TCP library the following examples are available:

- MB_TCP_7_EXA_TCP_Client.pcwex
- MB_TCP_7_EXA_TCP_Server.pcwex
- MB_TCP_7_EXA_UDP_Client.pcwex
- MB_TCP_7_EXA_UDP_Server.pcwex

The examples are located in the “Example” folder to the unzipped msi-file of the library.

17.1 System



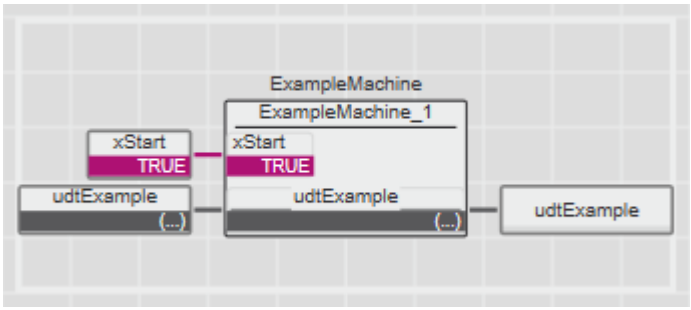
17.2 Modbus_TCP server TCP-Mode

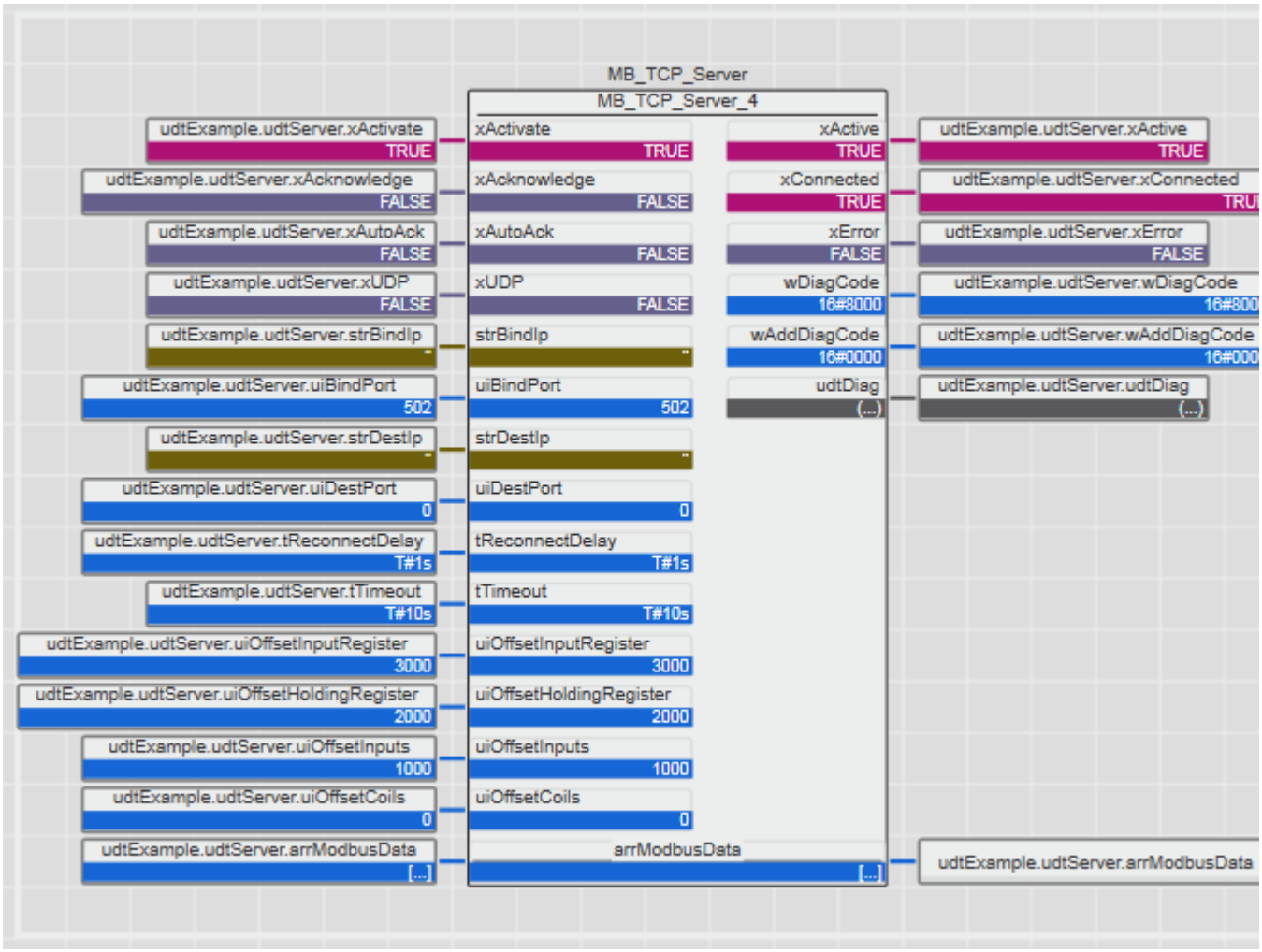
The MB_TCP_7_EXA_TCP_Server.pcwex file shows an example how to use a controller as a Modbus server with TCP-Mode. Here, the Modbus server provides its data to the client for reading.

17.2.1 Plant

17.2.2 Block connection

Main





Watches

arrModbusData[2000]	16#07D0	WORD
arrModbusData[2001]	16#07D1	WORD
arrModbusData[2002]	16#07D2	WORD
arrModbusData[2003]	16#07D3	WORD
arrModbusData[2004]	16#07D4	WORD
arrModbusData[2005]	16#07D5	WORD
arrModbusData[2006]	16#07D6	WORD
arrModbusData[2007]	16#07D7	WORD
arrModbusData[2008]	16#07D8	WORD
arrModbusData[2009]	16#07D9	WORD
arrModbusData[2010]	16#07DA	WORD
arrModbusData[2011]	16#07DB	WORD
arrModbusData[2012]	16#07DC	WORD
arrModbusData[2013]	16#07DD	WORD
arrModbusData[2014]	16#07DE	WORD
arrModbusData[2015]	16#07DF	WORD
arrModbusData[2016]	16#07E0	WORD

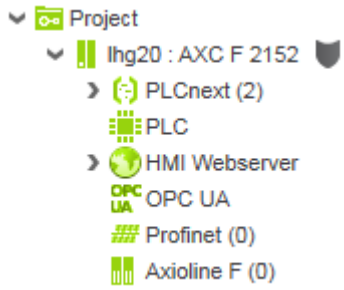
17.2.3 Programm startup

```
IF (xStart) THEN

    CASE iState OF
    0: (* Reset state *)
        udtExample.udtServer.xActivate                  := FALSE;
        udtExample.udtServer.xAcknowledge               := FALSE;
        udtExample.udtServer.xAutoAck                   := FALSE;
        udtExample.udtServer.xUDP                       := FALSE;
        udtExample.udtServer.strBindIp                  := '';
        udtExample.udtServer.uiBindPort                 := UINT#502;
        udtExample.udtServer.strDestIp                  := '';
        udtExample.udtServer.uiDestPort                 := UINT#0;
        udtExample.udtServer.tReconnectDelay            := TIME#1s;
        udtExample.udtServer.tTimeout                   := TIME#10s;
        udtExample.udtServer.uiOffsetInputRegister      := UINT#3000;
        udtExample.udtServer.uiOffsetHoldingRegister    := UINT#2000;
        udtExample.udtServer.uiOffsetInputs             := UINT#1000;
        udtExample.udtServer.uiOffsetCoils              := UINT#0;
        FOR iIndex := 0 TO 7167 DO
            udtExample.udtServer.arrModbusData[iIndex] := INT_TO_WORD(iIndex);
        END_FOR;
        udtExample.udtServer.arrModbusData[0] := WORD#16#FEDC;
        IF (udtExample.udtServer.xActive = FALSE) THEN
            iState := 100;
        END_IF;

    100: (* Activation of server*)
        udtExample.udtServer.xActivate := TRUE;
        IF (udtExample.udtServer.xActive = TRUE) THEN
            iState := 9000;
        END_IF;

    9000: (* Error handling *)
        IF (udtExample.udtServer.xError = TRUE) THEN
            iState := 0; (* Auto restart *)
        END_IF;

    END_CASE;
END_IF;
```

17.3 Modbus_TCP client TCP-Mode

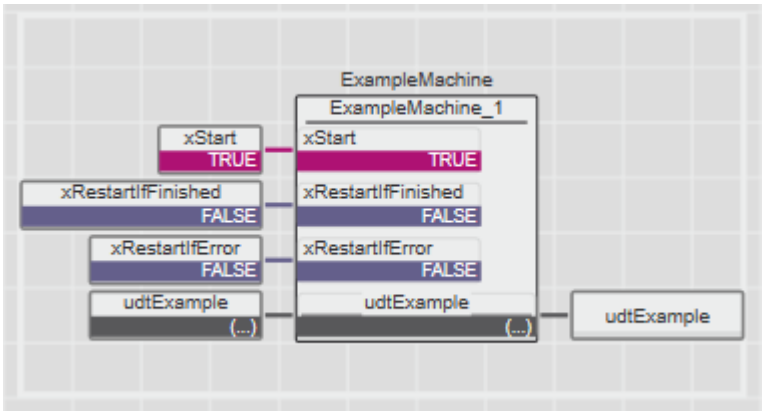
The MB_TCP_7_EXA_TCP_Client.pcwex file shows an example how to use a controller as a Modbus client with TCP-Mode. Here, the Modbus client reads the provided data from the Modbus server using the MB_TCP_FC3 function block.

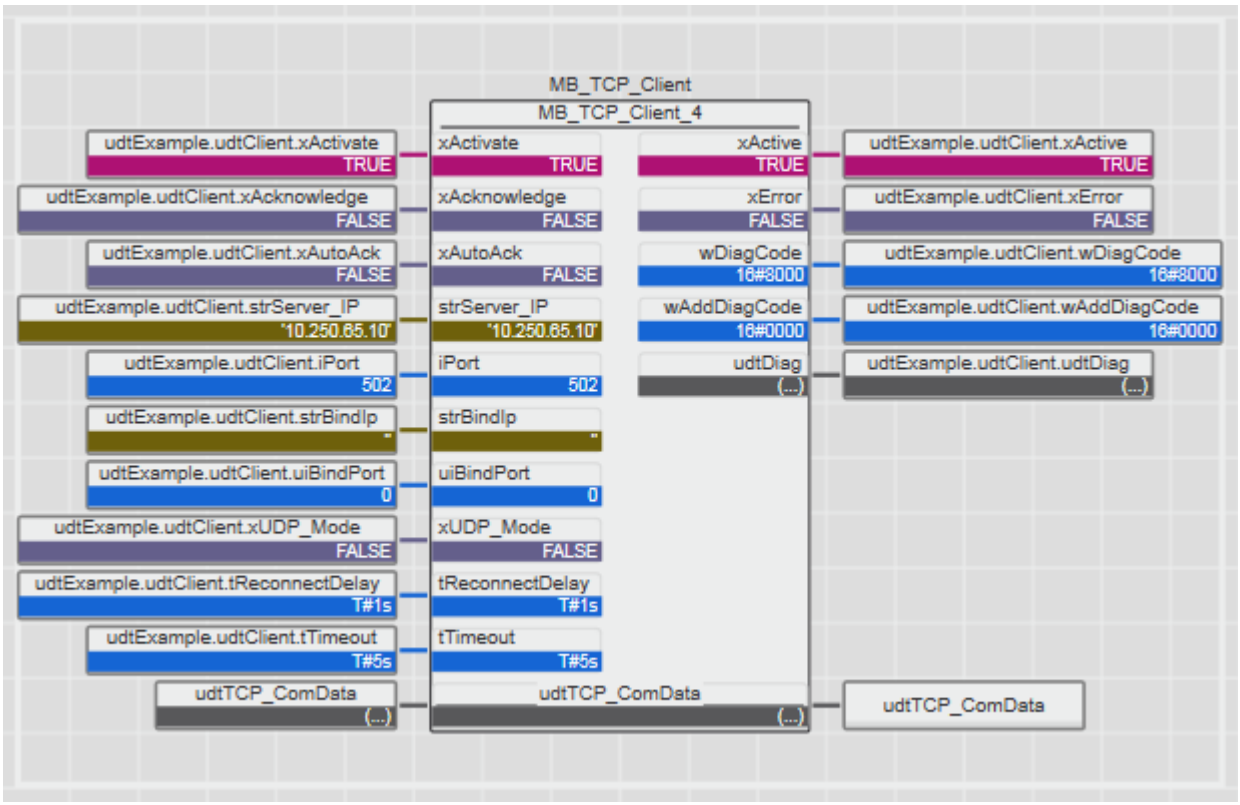
17.3.1 Plant

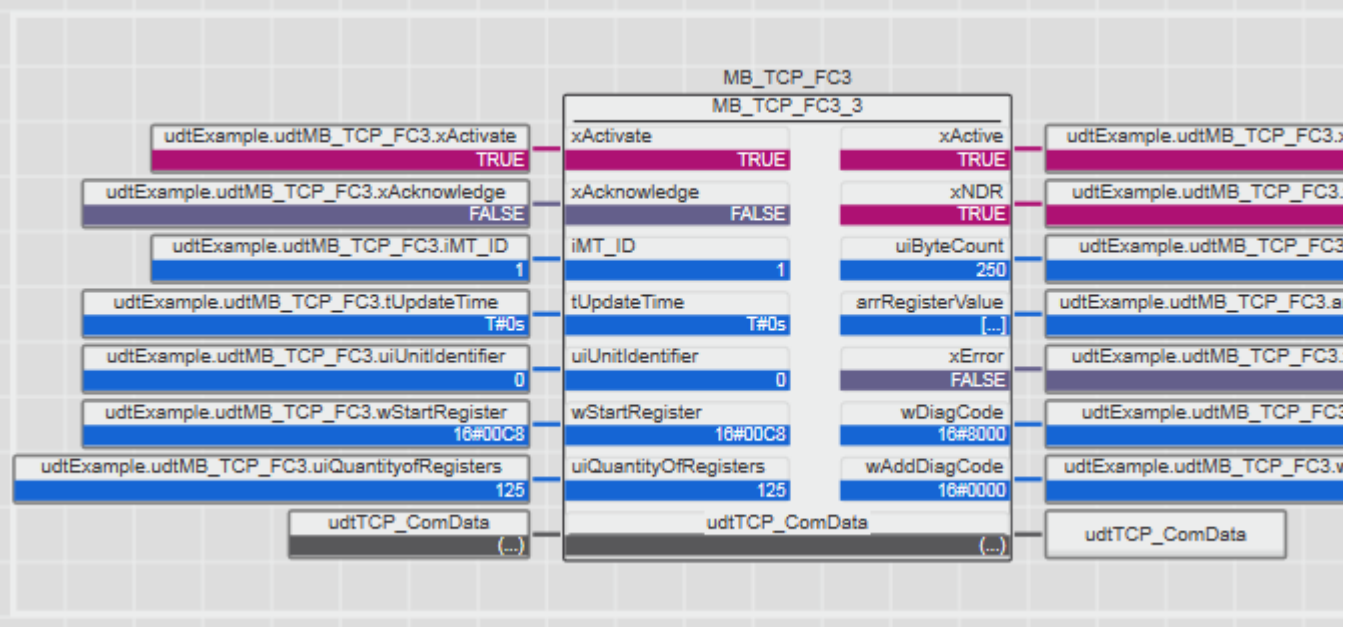


17.3.2 Block connection

Main






MB_TCP_FC3
MB_TCP_FC3_3
udtExample.udtMB_TCP_FC3.xActivate
TRUE
xActivate
TRUE
xActive
TRUE
udtExample.udtMB_TCP_FC3.xAcknowledge
FALSE
xAcknowledge
FALSE
xNDR
TRUE
udtExample.udtMB_TCP_FC3.iMT_ID
1
iMT_ID
1
uiByteCount
250
udtExample.udtMB_TCP_FC3.tUpdateTime
T#0s
tUpdateTime
T#0s
arrRegisterValue
[...]
udtExample.udtMB_TCP_FC3.uiUnitIdentifier
0
uiUnitIdentifier
0
xError
FALSE
udtExample.udtMB_TCP_FC3.wStartRegister
16#00C8
wStartRegister
16#00C8
wDiagCode
16#8000
udtExample.udtMB_TCP_FC3.uiQuantityofRegisters
125
uiQuantityOfRegisters
125
wAddDiagCode
16#0000
udtTCP_ComData
(...)
udtTCP_ComData
udtTCP_ComData
(...)

17.3.3 Programm startup

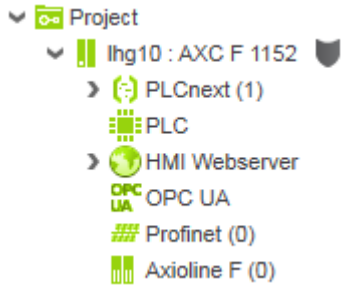
```
CASE iState OF

    0: (* Wait for start of Example program *)
        IF xStart = TRUE THEN
            iState := 10;
        END_IF;

    10: (* Initialization *)
        (* Client *)
        udtExample.udtClient.xActivate := FALSE;
        udtExample.udtClient.xAutoAck := FALSE;
        udtExample.udtClient.strServer_IP := '10.250.65.10';
        udtExample.udtClient.iPort := 502;
        udtExample.udtClient.xUDP_Mode := FALSE;
        udtExample.udtClient.tTimeout := TIME#5s;
        udtExample.udtClient.tReconnectDelay := TIME#1s;

        (* FC3 function block *)
        udtExample.udtMB_TCP_FC3.xActivate := FALSE;
        udtExample.udtMB_TCP_FC3.iMT_ID := 1;

        iState := 100;

    100: (* Start client function block first and wait for xActive *)
        udtExample.udtClient.xActivate := TRUE;
        IF (udtExample.udtClient.xActive = TRUE) THEN
            iState := 1000;
        END_IF;
        IF(
            udtExample.udtClient.xError = TRUE
            AND xRestartIfError = TRUE
        ) THEN
            udtExample.uiRestartIfError := udtExample.uiRestartIfError + UINT#1;
            iState := 10;
        END_IF;

    (* Read min. number of registers *)

    1000: (* Now activate FC3 function block and set inputs *)
        udtExample.udtMB_TCP_FC3.xActivate := TRUE;
        udtExample.udtMB_TCP_FC3.uiQuantityOfRegisters := UINT#1;
        (* udtFC3.arrRegisterValues = ARRAY [1..125] OF WORD *)
        udtExample.udtMB_TCP_FC3.wStartRegister := WORD#0;
        (* Wait for xActive of FC3 function block *)
        IF (udtExample.udtMB_TCP_FC3.xActive = TRUE) THEN
            iState := 1010;
        END_IF;
        IF(
            udtExample.udtMB_TCP_FC3.xError = TRUE
            AND xRestartIfError = TRUE
        ) THEN
            udtExample.uiRestartIfError := udtExample.uiRestartIfError + UINT#1;
            iState := 10;
        END_IF;

    1010: (* Check values - the FC reads Register 2000 from the Modbus_TCP_Server*)
        (* Modbus server has to deliver these values! *)
        IF (udtExample.udtMB_TCP_FC3.xError = FALSE)
            AND (udtExample.udtMB_TCP_FC3.arrRegisterValue[1] = WORD#16#07D0)
        THEN
            iState := 2000;
```

```
        END_IF;

    (* Read max. number of registers *)

    2000: (* Deactivation of FC3 function block before read again *)
        udtExample.udtMB_TCP_FC3.xActivate := FALSE;
        (* Wait for xActive = FALSE after deactivation *)
        IF (udtExample.udtMB_TCP_FC3.xActive = FALSE) THEN
            iState := 2010;
        END_IF;

    2010: (* Activation of FC3 function block and set inputs *)
        udtExample.udtMB_TCP_FC3.xActivate := TRUE;
        (* udtFC3.arrRegisterValues = ARRAY [1..125] OF WORD *)
        udtExample.udtMB_TCP_FC3.uiQuantityOfRegisters := UINT#125;
        (* read Register 2200 to 2324 *)
        udtExample.udtMB_TCP_FC3.wStartRegister := WORD#200;
        (* Wait for xActive = TRUE *)
        IF
            udtExample.udtMB_TCP_FC3.xActive = TRUE AND
            udtExample.udtMB_TCP_FC3.xError = FALSE
        THEN
            iState := 2020;
        END_IF;

    2020: (* Check values, if uiByteCount = 250 values are read correctly *)
        (* Modbus server has to deliver these values! *)
        IF
            udtExample.udtMB_TCP_FC3.xError = FALSE AND
            udtExample.udtMB_TCP_FC3.uiByteCount = UINT#250
        THEN
            iState := 2100;
        END_IF;
    2100: (* Deactivate FC3 function block *)
        udtExample.udtMB_TCP_FC3.xActivate := FALSE;
        IF udtExample.udtMB_TCP_FC3.xActive = FALSE THEN
            iState  :=  2200;
        END_IF;
    2200: (* wait for xStart = FASLE *)
        IF (xRestartIfFinished = TRUE) THEN
            udtExample.uiRestartIfFinished := udtExample.uiRestartIfFinished + UINT#1;
            iState := 10;
        END_IF;
        IF (xStart = FALSE) THEN
            iState  := 3000;
        END_IF;
    3000: (* Finish *)
        (* Deactivate client function block *)
        udtExample.udtClient.xActivate := FALSE;
        IF (
            udtExample.udtMB_TCP_FC3.xActive = FALSE
        ) THEN
            iState := 0;
        END_IF;
END_CASE;
```

17.4 Modbus_TCP server UDP-Mode

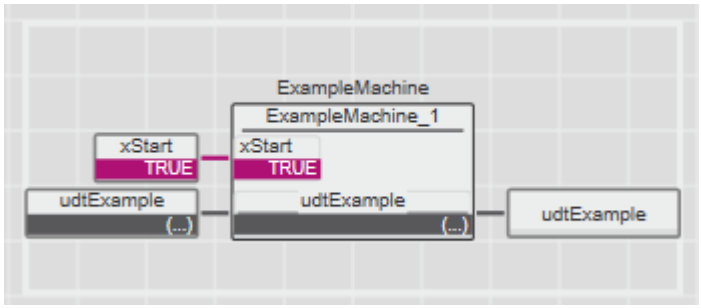
The MB_TCP_7_EXA_UDP_Server.pcwex file shows an example how to use a controller as a Modbus server with UDP-Mode. Here, the Modbus server provides its data to the client for reading.

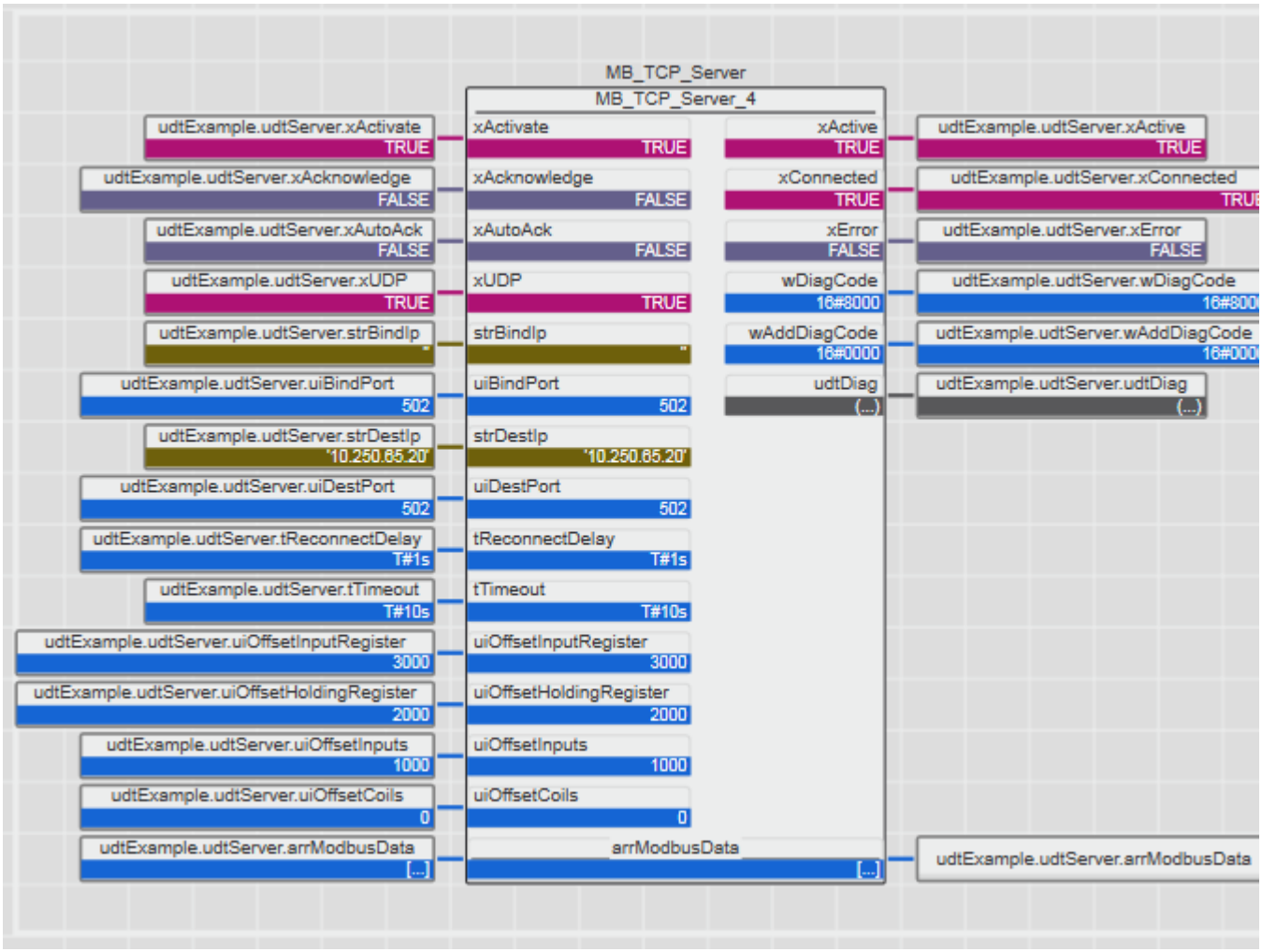
17.4.1 Plant



17.4.2 Block connection

Main





Watches

arrModbusData[2000]	16#07D0	WORD
arrModbusData[2001]	16#07D1	WORD
arrModbusData[2002]	16#07D2	WORD
arrModbusData[2003]	16#07D3	WORD
arrModbusData[2004]	16#07D4	WORD
arrModbusData[2005]	16#07D5	WORD
arrModbusData[2006]	16#07D6	WORD
arrModbusData[2007]	16#07D7	WORD
arrModbusData[2008]	16#07D8	WORD
arrModbusData[2009]	16#07D9	WORD
arrModbusData[2010]	16#07DA	WORD
arrModbusData[2011]	16#07DB	WORD
arrModbusData[2012]	16#07DC	WORD
arrModbusData[2013]	16#07DD	WORD
arrModbusData[2014]	16#07DE	WORD
arrModbusData[2015]	16#07DF	WORD
arrModbusData[2016]	16#07E0	WORD

17.4.3 Programm startup

```
IF (xStart) THEN

    CASE iState OF
    0: (* Reset state *)
        udtExample.udtServer.xActivate                  := FALSE;
        udtExample.udtServer.xAcknowledge               := FALSE;
        udtExample.udtServer.xAutoAck                   := FALSE;
        udtExample.udtServer.xUDP                       := TRUE;
        udtExample.udtServer.strBindIp                  := '';
        udtExample.udtServer.uiBindPort                 := UINT#502;
        udtExample.udtServer.strDestIp                  := '10.250.65.20';
            (* !IP of the Client! *)
        udtExample.udtServer.uiDestPort                 := UINT#502;
            (* !Bind-Port of the MB_TCP_Client! *)
        udtExample.udtServer.tReconnectDelay            := TIME#1s;
        udtExample.udtServer.tTimeout                   := TIME#10s;
        udtExample.udtServer.uiOffsetInputRegister      := UINT#3000;
        udtExample.udtServer.uiOffsetHoldingRegister    := UINT#2000;
        udtExample.udtServer.uiOffsetInputs             := UINT#1000;
        udtExample.udtServer.uiOffsetCoils              := UINT#0;
        FOR iIndex := 0 TO 7167 DO
            udtExample.udtServer.arrModbusData[iIndex] := INT_TO_WORD(iIndex);
        END_FOR;
        udtExample.udtServer.arrModbusData[0] := WORD#16#FEDC;
        IF (udtExample.udtServer.xActive = FALSE) THEN
            iState := 100;
        END_IF;

    100: (* Activation of server*)
        udtExample.udtServer.xActivate := TRUE;
        IF (udtExample.udtServer.xActive = TRUE) THEN
            iState := 9000;
        END_IF;

    9000: (* Error handling *)
        IF (udtExample.udtServer.xError = TRUE) THEN
            iState := 0; (* Auto restart *)
        END_IF;

    END_CASE;
END_IF;
```

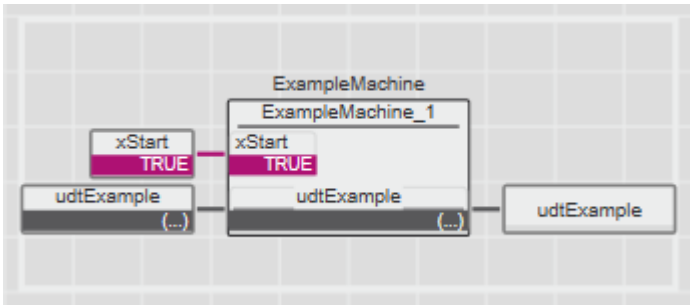
17.5 Modbus_TCP client UDP-Mode

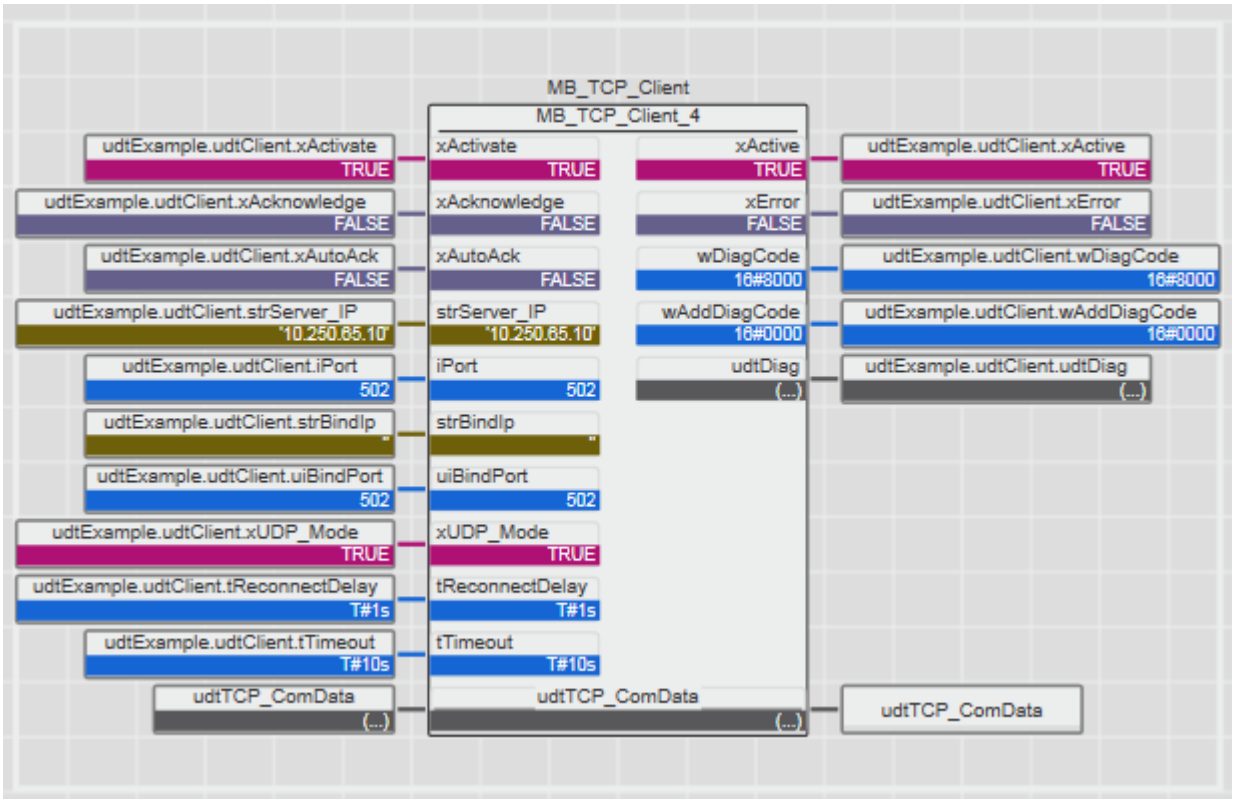
The MB_TCP_7_EXA_UDP_Client.pcwex file shows an example how to use a controller as a Modbus client with UDP-Mode. Here, the Modbus client reads the provided data from the Modbus server using the MB_TCP_FC3 function block.

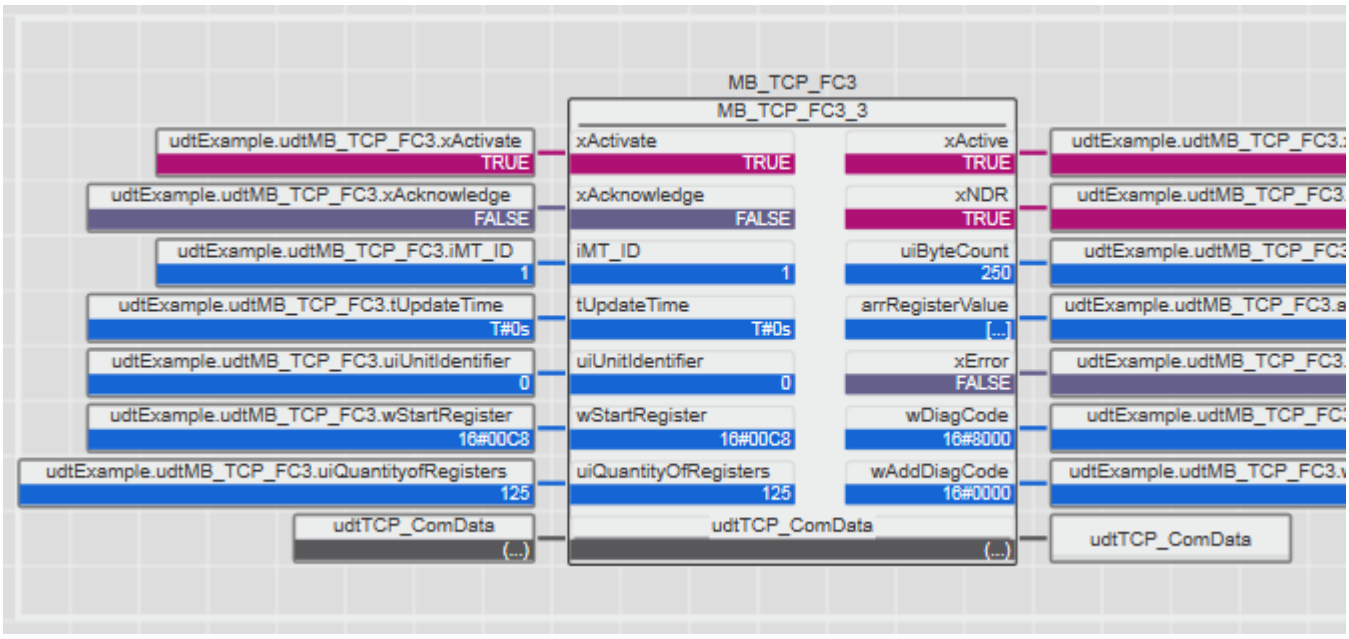
17.5.1 Plant

17.5.2 Block connection

Main







17.5.3 Programm startup

```
CASE iState OF

    0: (* Wait for start of Example program *)
        IF xStart = TRUE THEN
            iState := 10;
        END_IF;

    10: (* Initialization of client and FC3 function block *)
        (* Client *)
        udtExample.udtClient.xActivate := FALSE;
        udtExample.udtClient.xAutoAck := FALSE;
        udtExample.udtClient.strServer_IP := '10.250.65.10';
        udtExample.udtClient.iPort := 502;
        udtExample.udtClient.xUDP_Mode := TRUE;
        udtExample.udtClient.uiBindPort := 502; (* IMPORTANT! for UDP-Mode*)
        (* The Bind-Port of the Client has to be the Dest-Port of the Server *)
        udtExample.udtClient.tReconnectDelay := TIME#1s;
        udtExample.udtClient.tTimeout := TIME#10s;

        (* FC3 function block *)
        udtExample.udtMB_TCP_FC3.xActivate := FALSE;
        udtExample.udtMB_TCP_FC3.iMT_ID := 1;

        iState := 100;

    100: (* Start client function block first and wait for xActive *)
        udtExample.udtClient.xActivate := TRUE;
        IF (udtExample.udtClient.xActive = TRUE) THEN
            iState := 1000;
        END_IF;

    (* Read min. number of registers *)

    1000: (* Now activate FC3 function block and set inputs *)
        udtExample.udtMB_TCP_FC3.xActivate := TRUE;
        udtExample.udtMB_TCP_FC3.uiQuantityOfRegisters := UINT#1;
        (* udtFC3.arrRegisterValues = ARRAY [1..125] OF WORD *)
        udtExample.udtMB_TCP_FC3.wStartRegister := WORD#0;
        (* Wait for xActive of FC3 function block *)
        IF (udtExample.udtMB_TCP_FC3.xActive = TRUE) THEN
            iState := 1010;
        END_IF;

    1010: (* Check values - the FC reads Register 2000 from the Modbus_TCP_Server*)
        (* Modbus server has to deliver these values! *)
        IF (udtExample.udtMB_TCP_FC3.xError = FALSE)
            AND (udtExample.udtMB_TCP_FC3.arrRegisterValue[1] = WORD#16#07D0)
        THEN
            iState := 2000;
        END_IF;

    (* Read max. number of registers *)

    2000: (* Deactivation of FC3 function block before read again *)
        udtExample.udtMB_TCP_FC3.xActivate := FALSE;
        (* Wait for xActive = FALSE after deactivation *)
        IF (udtExample.udtMB_TCP_FC3.xActive = FALSE) THEN
            iState := 2010;
        END_IF;

    2010: (* Activation of FC3 function block and set inputs *)
```

```
        udtExample.udtMB_TCP_FC3.xActivate := TRUE;
        (* udtFC3.arrRegisterValues = ARRAY [1..125] OF WORD *)
        udtExample.udtMB_TCP_FC3.uiQuantityOfRegisters := UINT#125;
        (* read Register 2200 to 2324 *)
        udtExample.udtMB_TCP_FC3.wStartRegister := WORD#200;
        (* Wait for xActive = TRUE *)
        IF
            udtExample.udtMB_TCP_FC3.xActive = TRUE AND
            udtExample.udtMB_TCP_FC3.xError = FALSE
        THEN
            iState := 2020;
        END_IF;

    2020: (* Check values, if uiByteCount = 250 values are read correctly *)
        (* Modbus server has to deliver these values! *)
        IF
            udtExample.udtMB_TCP_FC3.xError = FALSE AND
            udtExample.udtMB_TCP_FC3.uiByteCount = UINT#250
        THEN
            iState := 2100;
        END_IF;
    2100: (* Deactivate FC3 function block *)
        udtExample.udtMB_TCP_FC3.xActivate := FALSE;
        IF udtExample.udtMB_TCP_FC3.xActive = FALSE THEN
            iState  :=  2200;
        END_IF;
    2200: (* wait for xStart = FASLE *)
        IF xStart = FALSE THEN
            iState  := 3000;
        END_IF;
    3000: (* Finish *)
        (* Deactivate client function block *)
        udtExample.udtClient.xActivate := FALSE;
        IF (
            udtExample.udtMB_TCP_FC3.xActive = FALSE
        ) THEN
            iState := 0;
        END_IF;
END_CASE;
```

18 Appendix

18.1 Call Modbus data with associated function codes

	Data access	Functioncode		Offset	Start Address	Length	arrModbusData (Array of 7167 Word)
Bit	Discret Input	2	Read Discret Input	200	0	16	200
		2	Read Discret Input	200	16	16	201
Bit	Discrete Output (Coils)	1	Read Coils	300	0	16	300
		1	Read Coils	300	16	16	301
		5	Write Single Coils	300	0	1	300
		15	Write Multiple Coils	300	0	16	300
		15	Write Multiple Coils	300	16	16	301
Word	Input Register	4	Read Input Register	0	0	10	0-9
		4	Read Input Register	0	20	10	20-29
Word	Holding Register	3	Read Holding Register	100	100	10	0-9
		6	Write Singel Register	100	100	1	0
		16	Write Multiple register	100	100	10	0-9

18.2 Diag codes of used firmware function blocks

18.2.1 TCP_SOCKET, TCP_RECEIVE, TCP_SEND, UDP_SOCKET, UDP_RECEIVE, UDP_SEND

ERROR = FALSE

Status code	Description
16#0000	Situation is normal (no error).
16#8000	Socket is trying to connect the partner.
16#8001	Server is listening for a client.
16#8002	Server has rejected a client because the IP address and port number do not match.
16#8003	Not all data could be sent. Remaining data will be sent in the next cycle(s).
16#8004	Not all data received: Received length < Expected length

ERROR = TRUE

Error code	Description	Error only for
16#C001	Socket creation failed.	
16#C002	IP has wrong format.	
16#C003	Memory allocation failed.	
16#C100	Unexpected error during connecting of a client to a server.	TCP/TLS_SOCKET
16#C101	Unexpected error during receive operation.	UDP/TCP/TLS_RECEIVE
16#C102	Unexpected error during send operation.	UDP/TCP/TLS_SEND
16#C103	Unexpected error during bind operation.	UDP_SOCKET
16#C104	Unexpected error during listen operation.	TCP/TLS_SOCKET
16#C105	Unexpected error during accept operation.	TCP/TLS_SOCKET
16#C150	The TLS parameterization of the TLS_SEND/TLS_RECEIVE function blocks is inconsistent with the TLS_SOCKET function block. This is the case when: • TLS_SEND/TLS_RECEIVE require secure transmission/reception of data (SEND_SECURE/RECEIVE_SECURE input = TRUE), but the socket is not yet initialized for TLS communication (START_TLS input of TLS_SOCKET is FALSE). • TLS_SEND/TLS_RECEIVE require insecure transmission/reception of data (SEND_SECURE/RECEIVE_SECURE input = FALSE), but the socket is already initialized for TLS communication (START_TLS input of TLS_SOCKET is TRUE).	TLS_*
16#C151	An error regarding the START_TLS input of the TLS_SOCKET function block has occurred. START_TLS was set from TRUE to FALSE during opened TLS socket (ACTIVE input = TRUE). This is the case when:	TLS_*
16#C201	There are too many open sockets in the underlying socket provider.	
16#C202	An operation on a nonblocking socket cannot be completed immediately.	
16#C204	The datagram is too long.	

16#C205	Only one use of an address is normally permitted.	
16#C206	The selected IP address is not valid in this context.	
16#C207	The connection was aborted by the .NET Framework or the underlying socket provider.	
16#C208	The connection was reset by the remote peer.	
16#C210	The application tried to send or receive data, and the Socket is not connected (_SOCKET.ACTIVE == False).	
16#C211	No such host is known. The name is not an official host name or alias.	
16#C212	An unspecified System.Net.Sockets. Socket error has occurred.	
16#C213	The remote host is actively refusing a connection.	
16#C214	An invalid argument was supplied to a System.Net.Sockets.Socket member.	
16#C215	A blocking operation is in progress.	
16#C216	The overlapped operation was aborted due to the closure of the System.Net.Sockets.Socket.	
16#C217	The application has initiated an overlapped operation that cannot be completed immediately.	
16#C218	A blocking System.Net.Sockets.Socket call was canceled.	
16#C219	An attempt was made to access a System.Net.Sockets.Socket in a way that is forbidden by its access permissions.	
16#C21A	An invalid pointer address was detected by the underlying socket provider.	
16#C21B	A System.Net.Sockets.Socket operation was attempted on a non-socket.	
16#C21C	A required address was omitted from an operation on a System.Net.Sockets.Socket.	
16#C21D	An unknown, invalid, or unsupported option or level was used with a System.Net.Sockets.Socket.	
16#C21E	The protocol type is incorrect for this System.Net.Sockets.Socket.	
16#C21F	The protocol is not implemented or has not been configured.	
16#C220	The support for the specified socket type does not exist in this address family.	
16#C221	The address family is not supported by the protocol family.	
16#C222	The protocol family is not implemented or has not been configured.	
16#C223	he address family specified is not supported. This error is returned if the IPv6 address family was specified and the IPv6 stack is not installed on the local machine. This error is returned if the IPv4 address family was specified and the IPv4 stack is not installed on the local machine.	
16#C224	The network is not available.	
16#C225	No route to the remote host exists.	
16#C226	The application tried to set System.Net.Sockets.SocketOptionName. KeepAlive on a connection that has already timed out.	
16#C227	No free buffer space is available for a System.Net.Sockets.Socket operation.	
16#C228	A request to send or receive data was disallowed because the System.Net.Sockets.Socket has already been closed.	
16#C229	The connection attempt timed out, or the connected host has failed to respond.	
16#C22A	The operation failed because the remote host is down.	

16#C22B	There is no network route to the specified host. Could not connect to DEST_IP.	
16#C22C	Too many processes are using the underlying socket provider.	
16#C22D	The network subsystem is unavailable.	
16#C22E	The version of the underlying socket provider is out of range.	
16#C22F	The underlying socket provider has not been initialized.	
16#C230	A graceful shutdown is in progress.	
16#C231	The specified class was not found.	
16#C232	The name of the host could not be resolved. Try again later.	
16#C233	The error is unrecoverable or the requested database cannot be located.	
16#C234	The requested name or IP address was not found on the name server.	

18.3 Data types

```
TYPE

    (* Byte arrays *)

    MB_TCP_ARR_B_1_250  : ARRAY [1..260] OF BYTE;

    (* Word arrays *)

    MB_TCP_ARR_W_1_125  : ARRAY [1..125] OF WORD;
    MB_TCP_ARR_W_0_7167 : ARRAY [0..7167] OF WORD;

    MB_TCP_UDT_WINSOCK : STRUCT
        DataCnt             : WORD;
        Transaction_ID      : WORD;
        Protocol_ID         : WORD;
        Length              : WORD;
        arrData : MB_TCP_ARR_B_1_250;
    END_STRUCT;

    MB_TCP_UDT_READ_WRITE : STRUCT
        Write                   : BOOL; (* FALSE = Read, TRUE = Write *)
        xActive                 : BOOL; (* TRUE: Read data *)
        FC                      : INT;  (* Modbus function code *)
        uiUnitIdentifier        : BYTE; (* Unit Identifier *)
        (* Read *)
        ReadStartAddress        : WORD; (* Address of the first register *)
        QuantityToRead          : WORD; (* Number of registers *)
        arrReadBuffer           : MB_TCP_ARR_B_1_250; (* Read buffer *)
        ByteCount               : BYTE; (* Number of received bytes *)
        xNDR                    : BOOL; (* TRUE (for one cycle): New data available *)
        (* Write *)
        WriteStartAddress       : WORD; (* Address of first register *)
        QuantityToWrite         : WORD; (* Number of registers *)
        arrWriteBuffer          : MB_TCP_ARR_B_1_250;   (* Write buffer *)
        xDone                   : BOOL; (* TRUE (for one cycle): Write is finished *)
        (* Diagnosis *)
        xError                  : BOOL; (* TRUE: Error *)
        wDiagCode               : WORD; (* Diag code *)
        wAddDiagCode            : WORD; (* Additional dig code *)
        (* Internal *)
        StartAction             : BOOL; (* Used by scheduler -> TRUE = Read data *)
        tUpdateTime             : Time; (* Time between two read operations in ms *)
    END_STRUCT;

    MB_TCP_ARR_READ_WRITE   : ARRAY [1..10] OF MB_TCP_UDT_READ_WRITE;

    MB_TCP_UDT_COMMUNICATION    : STRUCT
        ClientReady     : BOOL;
        xError          : BOOL;
        wDiagCode       : WORD;
        wAddDiagCode    : WORD;
        ReadWriteGroups : MB_TCP_ARR_READ_WRITE;
    END_STRUCT;

    MB_TCP_UDT_MBS_REQUEST  : STRUCT
        DATA_CNT:       WORD;   (* Not part of Modbus protocol *)
        Transaction_ID: WORD;
        Protocol_ID:    WORD;
        Length:         WORD;
        Unit_ID:        BYTE;
        FC:             BYTE;
```

```
    Reference:      WORD;
    Word_Count:     WORD;
    DATA:           MB_TCP_ARR_B_1_250;
END_STRUCT;

MB_TCP_UDT_MBS_RESPONSE : STRUCT
    DATA_CNT:       WORD;   (* Not part of Modbus protocol *)
    Transaction_ID: WORD;
    Protocol_ID:    WORD;
    Length:         WORD;
    Unit_ID:        BYTE;
    FC:             BYTE;
    DATA:           MB_TCP_ARR_B_1_250;
END_STRUCT;

(* Buffers for requests *)
MB_TCP_ARR_MBS_BUFFER   : ARRAY [0..10] OF MB_TCP_UDT_MBS_REQUEST;

(* I/O parameter of firmware iFunction blocks as struct *)

(* Inputs and outputs of TCP_SOCKET *)

MB_TCP_UDT_TCP_SOCKET : STRUCT
    (* Inputs *)
    xActivate       : BOOL;
    xIS_SRV         : BOOL;
    strBIND_IP      : STRING;
    uiBIND_Port     : UINT;
    strDEST_IP      : STRING;
    uiDEST_Port     : UINT;
    (* Outputs *)
    dwHandle        : DWORD;
    xActive         : BOOL;
    xBusy           : BOOL;
    xError          : BOOL;
    wStatus         : WORD;
    uiUSED_Port     : UINT;
END_STRUCT;

(* Inputs and outputs of TCP_SEND *)

MB_TCP_UDT_TCP_SEND : STRUCT
    (* Inputs *)
    xReq            : BOOL;
    udiData_CNT     : UDINT;
    (* Outputs *)
    xDone           : BOOL;
    xBusy           : BOOL;
    xError          : BOOL;
    wStatus         : WORD;
END_STRUCT;

(* Inputs and outputs of TCP_RECEIVE *)

MB_TCP_UDT_TCP_RECEIVE  : STRUCT
    (* Inputs *)
    xEN_R           : BOOL;
    udiEXP_Data_CNT : UDINT;
    (* Outputs *)
    xNDR            : BOOL;
    xError          : BOOL;
    wStatus         : WORD;
    strSource_IP    : STRING;
    uiSource_Port   : UINT;
```

```
    udiData_CNT     : UDINT;
END_STRUCT;

(* Inputs and outputs of TCP_SOCKET *)

MB_TCP_UDT_UDP_SOCKET   : STRUCT
    (* Inputs *)
    xActivate       : BOOL;
    strBIND_IP      : STRING;
    uiBIND_Port     : UINT;
    (* Outputs *)
    dwHandle        : DWORD;
    xActive         : BOOL;
    xBusy           : BOOL;
    xError          : BOOL;
    wStatus         : WORD;
    uiUSED_Port     : UINT;
END_STRUCT;

(* Inputs and outputs of TCP_SEND *)

MB_TCP_UDT_UDP_SEND : STRUCT
    (* Inputs *)
    xReq            : BOOL;
    strDEST_IP      : STRING;
    uiDEST_Port     : UINT;
    udiData_CNT     : UDINT;
    (* Outputs *)
    xDone           : BOOL;
    xBusy           : BOOL;
    xError          : BOOL;
    wStatus         : WORD;
END_STRUCT;

(* Inputs and outputs of TCP_RECEIVE *)

MB_TCP_UDT_UDP_RECEIVE  : STRUCT
    (* Inputs *)
    xEN_R           : BOOL;
    (* Outputs *)
    xNDR            : BOOL;
    xError          : BOOL;
    wStatus         : WORD;
    strSource_IP    : STRING;
    uiSource_Port   : UINT;
    udiData_CNT     : UDINT;
END_STRUCT;

MB_TCP_UDT_SER_DIAG : STRUCT
    iState          : INT;
    wDiagCode       : WORD;
    wAddDiagCode    : WORD;
    udtTCP_SOCKET   : MB_TCP_UDT_TCP_SOCKET;
    udtTCP_SEND     : MB_TCP_UDT_TCP_SEND;
    udtTCP_RECEIVE  : MB_TCP_UDT_TCP_RECEIVE;
    udtUDP_SOCKET   : MB_TCP_UDT_UDP_SOCKET;
    udtUDP_SEND     : MB_TCP_UDT_UDP_SEND;
    udtUDP_RECEIVE  : MB_TCP_UDT_UDP_RECEIVE;
END_STRUCT;

(* Diag structure for MB_TCP_Client *)

MB_TCP_UDT_CLI_DIAG : STRUCT
    iState          : INT;
```

```
        wDiagCode       : WORD;
        wAddDiagCode    : WORD;
        udtTCP_SOCKET   : MB_TCP_UDT_TCP_SOCKET;
        udtTCP_SEND     : MB_TCP_UDT_TCP_SEND;
        udtTCP_RECEIVE  : MB_TCP_UDT_TCP_RECEIVE;
        udtUDP_SOCKET   : MB_TCP_UDT_UDP_SOCKET;
        udtUDP_SEND     : MB_TCP_UDT_UDP_SEND;
        udtUDP_RECEIVE  : MB_TCP_UDT_UDP_RECEIVE;
    END_STRUCT;

    (*Struct for outputs of TCP/UDP_SOCKET*)
    MB_TCP_UDT_SOCKET   : STRUCT
        xActive         : BOOL;
        xBusy           : BOOL;
        xError          : BOOL;
        wStatus         : WORD;
        uiUsed_Port     : UINT;
    END_STRUCT;

    (*Struct for outputs of TCP/UDP_SEND*)
    MB_TCP_UDT_SEND : STRUCT
        xDone           : BOOL;
        xBusy           : BOOL;
        xError          : BOOL;
        wStatus         : WORD;
    END_STRUCT;

    (*Struct for outputs of TCP/UDP_RECEIVE*)
    MB_TCP_UDT_RECEIVE  : STRUCT
        xNDR            : BOOL;
        xError          : BOOL;
        wStatus         : WORD;
        strSource_IP    : STRING;
        uiSource_Port   : UINT;
        udiDataCNT      : UDINT;
    END_STRUCT;

END_TYPE
```

19 Support

For technical support please contact your local PHOENIX CONTACT agency

at https://www.phoenixcontact.com

Owner:

PHOENIX CONTACT Electronics GmbH
Business Unit Automation Systems
System Services
Library Services