Modbus TCP

PLC300

User's Manual

Modbus TCP User's Manual

Series: PLC300

Language: English

Document Number: 10002233461 / 00

Publication Date: 04/2013

CONTENTS

ABOUT THIS MANUAL

This manual supplies the necessary information for the operation of the PLC300 programable controller using the Modbus TCP protocol. This manual must be used together with the PLC300 user manual.

ABBREVIATIONS AND DEFINITIONS

ASCII	American Standard Code for Information Interchange
CSMA/CD	Carrier Sense Multiple Access/Collision Detection
IP	Internet Protocol
MAC	Medium Access Control
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

NUMERICAL REPRESENTATION

Decimal numbers are represented by means of digits without suffix. Hexadecimal numbers are represented with the letter ‘h’ after the number. Binary numbers are represented with the letter ‘b’ after the number.

DOCUMENTS

The Modbus TCP protocol was developed based on the following specifications and documents:

Document	Version	Source
MODBUS Application Protocol Specification, December 28th 2006.	V1.1b	MODBUS.ORG
MODBUS Messaging On TCP/IP Implementation Guide, October 24th 2006.	V1.0b	MODBUS.ORG

In order to obtain this documentation, consult MODBUS.ORG, which is nowadays the organization that keeps, publishes and updates the information related to the Modbus protocol.

1 NETWORK CONNECTIONS

The PLC300 programable controller has a standard Ethernet interface. Information about the connection and installation of the equipment to the network is presented bellow.

1.1 ETHERNET INTERFACE CHARACTERISTICS

- The interface follows the T-568A / T-568B standard
- It operates as a client or server in the Modbus TCP network.
- It allows communication using the 10 or 100 Mbps rates in *half* or *full* duplex mode.

1.2 INDICATIONS

Besides the system markers, which provide different kinds of information about the Ethernet interface, the PLC300 programable controller has LEDs in the RJ45 connector, used for Ethernet Interface indication.

Figure 1.1: *Indication LEDs of the Ethernet interface*

The SPD LED (1) indicates the current communication rate of the Ethernet interface.

Table 1.1: *SPD LED status*

LED Status	Description	Comments
Off	10 Mbps	Using 10 Mbps communication rate
Green	100 Mbps	Using 100 Mbps communication rate

The LINK LED (2) indicates the state of the physical connection of the network, as well as the activity in the bus.

Table 1.2: *LINK LED status*

LED Status	Description	Comments
Off	Without link	Without connection, without activity
Yellow	Link	Ethernet link established but without data exchange
Flashing yellow	Activity in the bus	It effectively indicates that there is exchange of telegrams with the network.

1.3 INSTALLATION OF THE ETHERNET NETWORK

For the connection of the programable controller using the Ethernet interface, the following points must be observed:

1.3.1 Communication Rate

The Ethernet interfaces of the programable controller can communicate using the 10 or 100 Mbps rates in *half* or *full* duplex mode.

The baud rate can be set by means of the Setup menu or WEB browser.

1.3.2 MAC Address

Each PLC300 programable controller has a unique MAC address, which is indicated on the display (Status de IO).

1.3.3 IP Address

Every equipment in an Ethernet network needs an IP address and subnet mask.

The IP addressing is unique in the network, and each equipment must have a different IP. The subnet mask is used to define which IP address range is valid in the network.

These attributes can be automatically configured by means of a DHCP server present in the network, as long as this option is enabled on PLC300.

The configurations of the IP address, subnet mask, gateway and DHCP can be set by means of the Setup menu or WEB browser.

1.3.4 Cables

To perform the installation, it is recommended the use of shielded Ethernet cables specific for use in industrial environment.

The metallic housing of the Ethernet connector of the PLC300 is grounded, and grounds the cable in case the cable connector features metallic enclosure.

Normally, a direct cable is used to connect the PLC300 to a switch, or a cross-over cable for direct connection between the PLC300 and the PC/PLC. In spite of that, the Ethernet interface of the PLC300 works with Auto-MDIX (automatic medium-dependent interface crossover), a technology which automatically detects the type of cable used and configures the connection accordingly, eliminating the need of cross-over cables.

1.3.5 Installation recommendations

- It is recommended the use of equipment (cables, switches) suitable for the industrial environment.
- Each cable segment must be 90 m long at most.
- The cable must be laid separately (and far away if possible) from the power cables.
- All the network devices must be properly grounded, preferably at the same ground connection.
- The most usual topology is star, exactly as it is done with computer networks. In this case all the equipments must be connected to a concentrating element (switch).

Figure 1.2: *Star topology*

1.4 ACCESS USING THE WEB BROWSER

It is possible to use a web browser to access the configuration and status data of the PLC300 programable controller. Typing the IP address in the address bar of the browser, you will see a webpage with the equipment information.

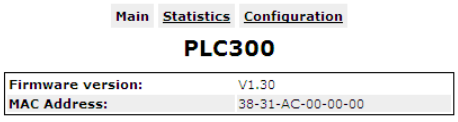


Figure 1.3: *Initial WEB page*

1.5 SNTP CLIENT

The programable controller PLC300 has a built-in SNTP client. This client uses the SNTP protocol to request date and time information from a server, and automatically changes these configurations in the PLC300.

Using the WEB interface, it is possible to configure the SNTP client. The IP addresses of the main server and redundant server to which the PLC300 must connect in order to search date and time information must be informed. The redundant server will be used when the main server is not accessible on the network.
It is possible to configure the time interval between the updates of date and time and the maximum waiting time for the response of the time server.

2 MODBUS PROTOCOL

The Modbus protocol was initially developed in 1979. Nowadays, it is a widely spread open protocol, used by several manufactures in many equipments. It is a protocol of application layer for communication between devices, especially used by industrial automation systems.

2.1 MESSAGE STRUCTURE

Modbus is a protocol based on transactions, which consist of a request followed by a response. Every communication begins with the client (master) making a request to a server (slave), which answers what has been asked.

The communication is based on a packet called PDU (Protocol Data Unit) which is defined by the specification of the protocol in three types:

- Request PDU:
  - *Function Code*: specifies the kind of service or function requested (1 byte)
  - *Function Data*: specific function data (variable number of bytes)
- Response PDU:
  - *Function Code*: code of the function corresponding to the request (1 byte)
  - *Response Data*: specific function data (variable number of bytes)
- Exception PDU:
  - *Error Code*: function code corresponding to the request with the most significant bit set (1 byte)
  - *Exception Code*: code specifying the exception (1 byte)

A transaction can be viewed in figure 2.1.

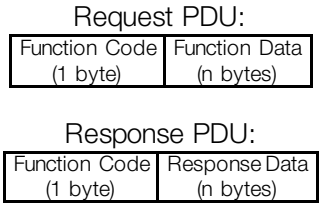


Figure 2.1: *Modbus Transaction*

The function code field specifies the kind of service or function requested to the server (reading, writing, etc.). For the list of available functions to access data, and the description of the data field for each function, refer to item 4.

According to the protocol, each function is used to access a specific type of data. Table 2.1 contains the basic types defined in the specification.

Table 2.1: *Modbus data type*

Name	Size	Access
Discrete Input	1 bit	Read Only
Discrete Output (Coils)	1 bit	Read and Write
Input Registers	16 bits	Read Only
Holding Registers	16 bits	Read and Write

Each implantation of the Modbus protocol can add to the PDU specific data for the proper processing of the messages through the interface used.

2.2 MODBUS TCP IMPLEMENTATION

Modbus TCP is an implementation of the Modbus protocol based on TCP/IP. The TCP/IP stack is used for communication and it adds to the Modbus PDU a specific header called MBAP Header. The association of the header to the PDU is called ADU (Application Data Unit).

The header size is 7 bytes, and it is composed of the following fields:

- *Transaction identifier:* used to identify the response to the transaction (2 bytes).
- *Protocol identifier:* 0 (zero) indicates Modbus (2 bytes).

- *Length:* count of all next bytes (2 bytes).
- *Unit identifier:* used to identify the remote slave in a Modbus RTU network (1 byte).

Figure 2.2: *Modbus TCP ADU*

Modbus TCP does not add to the PDU an error checking field; however, the Ethernet frame already uses CRC-32, making another checking field unnecessary.

The Modbus TCP client must start a TCP connection with the server in order to send the requests. The TCP port 502 is the standard port for the connection with the Modbus TCP servers.

3 INTERFACE CONFIGURATION

In order to configure the Ethernet interface, the following menus are provided by the Setup of the PLC300 programmable controller:

3.1 ETHERNET CONFIGURATION

IP ADDRESS

Range: 0.0.0.0 to 255.255.255.255 **Default:** 192.168.0.10

Description:
It allows programming the IP address used by the PLC300. The IP addressing is unique in the network, and each equipment must have a different IP.

NOTE!
These attributes can be automatically configured by means of a DHCP server present in the network, as long as this option is enabled.

SUB-NET MASK

Range: 0.0.0.0 to 255.255.255.255 **Default:** 255.255.255.0

Description:
It allows programming the sub-net mask used by the PLC300. The subnet mask defines which IP address ranges are valid on the network.

NOTE!
These attributes can be automatically configured by means of a DHCP server present in the network, as long as this option is enabled.

DEFAULT GATEWAY

Range: 0.0.0.0 to 255.255.255.255 **Default:** 0.0.0.0

Description:
It allows programming the default gateway used by the PLC300. The default gateway provides a route for the communication with remote networks.

NOTE!
These attributes can be automatically configured by means of a DHCP server present in the network, as long as this option is enabled.

DHCP

Range: Disabled
Enabled
Default: Disabled

Description:
It allows enabling or disabling the configuration via DHCP server. The DHCP can automatically assign IP addresses, subnet mask, etc. to the devices on the network.

In case the DHCP is enabled, the configurations made for IP address, subnet mask and gateway will be disregarded. The IP address assigned by the DHCP server can be viewed on the I/O Status screen.

COMMUNICATION RATE

Range:	Auto 10MBps Full Duplex 10MBps Half Duplex 100MBps Full Duplex 100MBps Half Duplex	**Default:** Auto

Description:
The Ethernet interface can communicate using the 10 or 100 Mbps rates in *half* or *full* duplex mode.
When the Auto option is selected, the automatic detection of baud rate and communication mode occurs.

3.2 MODBUS TCP CONFIGURATION

TCP PORT

Range:	0 to 65535	**Default:** 502

Description:
It configures the TCP port used for communication with the Modbus TCP server.
The 502 port remains open, even if another port is configured. Is this case, it is possible to connect to the server by means of any of these ports.

UNIT ID

Range:	1 to 255	**Default:** 255

Description:
Identifier of the Modbus TCP protocol for this equipment.
Telegrams received with identifier different from the configured one will be discarded.
If the configuration value is 255, the server will act as a Modbus TCP/RTU Gateway and will forward received messages that have Unit ID between 1 and 247 to the RS485 serial interface, if that is configured as Modbus RTU master.
Thus, it is possible to establish communication between the client connected to the Ethernet interface of the PLC300 and a device connected to the RS485 interface.

IP AUTHENTICATION

Range:	0.0.0.0 to 255.255.255.255	**Default:** 0.0.0.0

Description:
When set for a value different from 0.0.0.0, only the device/PC with this IP address can establish communication with the server. Connection requests from other addresses will be denied.

GATEWAY TIMEOUT

Range:	20 to 5000 ms	**Default:** 1000 ms

Description:
It indicates the response timeout that must be used when a request is forwarded by the gateway to a Modbus RTU slave. If this time expires, and the answer is not received, the gateway will return a telegram indicating error to the Modbus TCP client that originated the request.

4 OPERATION IN THE MODBUS TCP NETWORK – SERVER MODE

The PLC300 programable controller has the following characteristics when operated as a server in Modbus TCP network:

- It may operate simultaneously as Modbus TCP client and server.
- It may act as gateway for Modbus RTU via RS485 interface.
- It allows the connection of up to 8 Modbus TCP clients simultaneously.
- TCP Port, Unit ID, etc. defined by means of setup of the equipment.
- It allows accessing all the markers and data used in the ladder program of the PLC300.

NOTE!
The RS232, RS485, USB and Ethernet interfaces, for using the same functions to access the data and programming of the equipment, must not be used simultaneously to perform program download or on-line monitoring functions of the programmable controller PLC300, because conflicts may occur during the simultaneous access to the data.

4.1 AVAILABLE FUNCTIONS AND RESPONSE TIMES

In the Modbus specification are defined the functions used to access different types of data. In the PLC300, in order to access those data the following services (or functions) have been made available:

- Read Coils
  Description: reading of bit blocks of the coil type.
  Function code: 01.

- Read Discrete Inputs
  Description: reading of bit blocks of the discrete input type.
  Function code: 02.

- Read Holding Registers
  Description: reading of register blocks of the holding register type.
  Function code: 03.

- Read Input Registers
  Description: reading of register blocks of the input register type.
  Function code: 04.

- Write Single Coil
  Description: writing in a single bit of the coil type.
  Function code: 05.

- Write Single Register
  Description: writing in a single register of the holding type.
  Function code: 06.

- Write Multiple Coils
  Description: writing in bit blocks of the coil type.
  Function code: 15.

- Write Multiple Registers
  Description: writing in register blocks of the holding register type.
  Function code: 16.

- Read Device Identification
  Description: identification of the device model.
  Function code: 43.

The response time, from the end of transmission of the client until the response of the server, varies according to the equipment scan cycle value.

4.2 MEMORY MAP

The PLC300 programable controller has different types of data accessible through the Modbus communication. These data are mapped at data addresses and access functions as described in the following items.

NOTE!
The WPS programming software has lists that allow the viewing of all types of markers available for the PLC300. In these lists, there is a field for indication of the address of the Modbus register to access the marker.

4.2.1 Reading System Marker – %SB / %SW / %SD

The reading system markers represent the data of the PLC300 used for indication of status and monitoring of the equipment functions.

- Access: read only.
- Data type: input register or input discrete.
- Modbus access functions: 02 and 04.
- Modbus address range for access via *input register*: 3000 ... 4999.
- Modbus address range for access via *input discrete*: 0 ... 15999.

The system markers related to the ethernet communication available for the PLC300 are described in item 7. For the description of other markers available and function of each marker, refer to the user's manual of the PLC300.

4.2.2 Command System Marker – %CB / %CW / %CD

The writing system markers represent the data of the PLC300 used to configure and control the equipment functions.

- Access: read/write.
- Data type: holding register or coil.
- Modbus access functions: 01, 03, 05, 06, 15 and 16.
- Modbus address range for access via holding register: 3000 ... 4999.
- Modbus address range for access via coil. 0 ... 15999.

The system markers related to the ethernet communication available for the PLC300 are described in item 7. For the description of other markers available and function of each marker, refer to the user's manual of the PLC300.

4.2.3 Inputs – %IB / %IW / %ID

Markers that represent the data related to the physical analog and digital inputs, available on the hardware of the PLC300.

- Access: read only.
- Data type: input register or input discrete.
- Modbus access functions: 02 and 04.
- Modbus address range for access via input register: 5000 ... 5999.
- Modbus address range for access via input discrete: 16000 ... 23999.

For the precise description of which markers are available and function of each marker, refer to the user's manual of the PLC300.

4.2.4 Outputs – %QB / %QW / %QD

Markers that represent the data related to the physical analog and digital outputs, available in hardware of the PLC300.

- Access: read/write.
- Data type: holding register or coil.

- Modbus access functions: 01, 03, 05, 06, 15 and 16.
- Modbus address range for access via holding register: 5000 ... 5999.
- Modbus address range for access via coil. 16000 ... 23999.

For the precise description of which markers are available and function of each marker, refer to the user's manual of the PLC300.

4.2.5 Network Inputs – %IB / %IW / %ID

Markers that represent data related to values received through the PLC300 network interfaces. They have the same terminology as the physical inputs, but their numbering begins from marker 2000 (example: %IB2000).

- Access: read only.
- Data type: input register or input discrete.
- Modbus access functions: 02 and 04.
- Modbus address range for access via input register: 6000 ... 7999.
- Modbus address range for access via input discrete: 24000 ... 39999.

4.2.6 Network Outputs – %QB / %QW / %QD

Markers that represent data related to values transmitted through the PLC300 network interfaces. They have the same nomenclature as the physical outputs, but their numbering begins from marker 2000 (example: %QB2000).

- Access: read/write.
- Data type: holding register or coil.
- Modbus access functions: 01, 03, 05, 06, 15 and 16.
- Modbus address range for access via holding register: 6000 ... 7999.
- Modbus address range for access via coil. 24000 ... 39999.

4.2.7 Internal Marker – %MB / %MW / %MD

General purposes markers for programming in ladder of the PLC300. They represent the global variables, dynamically created during the development of the program on the WPS software.

- Access: read/write.
- Data type: holding register or coil.
- Modbus access functions: 01, 03, 05, 06, 15 and 16.
- Volatile markers:
  - Modbus address range for access via holding register: 8000 ... 27999.
  - Modbus address range for access via coil. 40000 ... 49999.
- Retentive markers:
  - Modbus address range for access via holding register: 28000 ... 47999.
  - Modbus address range for access via coil. 50000 ... 59999.

The quantity of markers available in this area depends on the markers created on the PLC300 programming software. In order to be able to access the desired marker, first it is necessary to create this marker and download the user's program using the programming software.

NOTE!
The quantity of data accessible via coils and input discretes do not correspond to the entire memory area accessible via registers. For example, if it is created a quantity of markers on memory greater than the quantity accessible via coil (10000 bits = 1250 bytes), the additional markers can only be accessed via holding registers.

4.3 DATA ACCESS

Each of the memory regions described above is distributed in bytes. The Modbus protocol, though, allows the access to be performed only by bits of by registers of 16 bits. In order to access these memory regions, it is necessary to establish the relationship between the type and numbering of the data on the PLC300 with the

Modbus address type. The following tables show how the relationship between the numbering of the data on the PLC300 and the address of the Modbus registers that access these data is done.

Reading System Markers		
Marker Number PLC300		Register Address (input register) Modbus
%SB3001	%SB3000	3000
%SB3003	%SB3002	3001
⋮		⋮
%SB3101	%SB3100	3050
⋮		⋮

Command System Markers		
Marker Number PLC300		Register Address (holding register) Modbus
%CB3001	%CB3000	3000
%CB3003	%CB3002	3001
⋮		⋮
%CB3101	%CB3100	3050
⋮		⋮

Inputs		
Marker Number PLC300		Register Address (input register) Modbus
%IB1	%IB0	5000
%IB3	%IB2	5001
⋮		⋮
%IB2001	%IB2000	6000
⋮		⋮

Outputs		
Marker Number PLC300		Register Address (holding register) Modbus
%QB1	%QB0	5000
%QB3	%QB2	5001
⋮		⋮
%QB2001	%QB2000	6000
⋮		⋮

Internal markers (volatile and retentive)		
Marker Number PLC300		Register Address (holding register) Modbus
%MB1	%MB0	8000
%MB3	%MB2	8001
⋮		⋮
%MB40001	%MB40000	28000
⋮		⋮

The following table illustrates how the Modbus address is calculated with access via registers, for different types of data available for the PLC300:

Data	Description	Data type	Base address	Base address offset	Modbus address
%SW3002	Reading system marker which represents the scan cycle time.	Input Register	3000	2 bytes (1 word)	3001
%CW3030	Writing system marker to set the hour of the RTC.	Holding Register	3000	30 bytes (15 words)	3015
%IB0	Physical inputs, representing the digital inputs 1 to 8.	Input Register	5000	0 bytes (0 words)	5000 byte low
%MB11	Marker in volatile memory, representing a global variable created by the user with a size of one byte.	Holding Register	8000	11 bytes (5 words)	8005 byte high
%MD40004	Marker on retentive memory, representing a global variable created by the user with a size of four bytes.	Holding Register	28000	4 bytes (2 words)	28002 e 28003

Similarly, the access via binary data (coils or input discretes) also uses a base address plus the offset given by the number of the marker. However, since each byte has eight bits, for each byte from the base address, eight bits must be added to the address for access via binary data.

The data format and function in the memory area accessed, though, are not pre-defined, and depend on the programming done on the WPS software. For example, for the memory marker %M_0, it is possible to create the following variables on the WPS software:

- *%MB0:* byte marker, it takes only one memory byte, and it can represent an 8-bit integer with or without signal. In the access via registers, since the Modbus protocol allows the reading or writing access of at least 16 bits, whenever this marker is read or written, the bytes %MB0 and %MB1 are accessed.
- *%MW0:* word marker, it takes two memory bytes, and it can represent a 16-bit integer with or without signal. In this case, the bytes %MB0 and %MB1 are reserved for this marker.
- *%MD0:* double marker, it takes four memory bytes, and it can represent a 32-bit integer with or without signal. In this case, the bytes %MB0 and %MB3 are reserved for this marker. In the access by registers, it is necessary to read or write two registers in a row, with the least significant value in the first register, so that the four bytes will be accessed.

Table 4.1: *Example of data addressing for volatile markers on the PLC300*

Marker Type / Modbus addr. Register (bit)	Byte (%MB)	Word (%MW)	Double (%MD)
8000 (40000 ... 40015)	X	X	X
	X		
8001 (40016 ... 40031)	X	X	
	X		
8002 (40032 ... 40047)	X	X	X
	X		
8003 (40048 ... 40063)	X	X	
	X		

Similarly, it is possible to access the data using the bit access functions. In this case, a bit can be accessed individually, or in a group of bits that represents a marker. For example, if it is defined on the WPS software a word marker in address 8000 – %MW0 – it is possible to access this marker by using the functions of multiple coil reading or writing, using the bits 40000 to 40015.

At the memory addresses of the PLC300, variables with size above one byte are always stored with the least significant byte first. Thus, the available space on memory for Byte, Word or Double values follows the description of the following table.

Table 4.2: *Example of data addressing for volatiles markers on the PLC300*

Marker Type / Modbus addr. Register (bit)	Byte (%MB)		Word (%MW)		Double (%MD)	
8000 (40000 ... 40015)	%MB0	Unique value	%MW0	Value -signf. Value +signf.	%MD0	Value -signf. ... Value +signf.
	%MB1	Unique value				
8001 (40016 ... 40031)	%MB2	Unique value	%MW2	Value -signf. Value +signf.		
	%MB3	Unique value				
8002 (40032 ... 40047)	%MB4	Unique value	%MW4	Value -signf. Value +signf.	%MD4	Value -signf. ... Value +signf.
	%MB5	Unique value				
8003 (40048 ... 40063)	%MB6	Unique value	%MW6	Value -signf. Value +signf.		
	%MB7	Unique value				

Since the Modbus protocol defines that in order to transmit a 16-bit register, the most significant byte must be transmitted first, when accessing any register, the following memory address is transmitted first. Therefore, if four registers are read in a row, from the register 8000, the content of each register will be transmitted the following way:

1st Register – 8000		2nd Register – 8001		3rd Register – 8002		4th Register – 8003	
%MB1	%MB0	%MB3	%MB2	%MB5	%MB4	%MB7	%MB6

4.4 USE AS A MODBUS RTU GATEWAY

When the Unit ID of the Modbus TCP server is configured with the value 255, and the RS485 interface is configured as Modbus RTU master, messages received by the server that contain Unit ID with values between 1 and 247 will be forwarded through the Modbus RTU master via RS485 to the slaves of this network. Messages with Unit ID equal to 0 or 255 will be interpreted by the PLC300.

In case a timeout of the Modbus RTU slave response occurs, the gateway will return a telegram indicating error to the Modbus TCP client that originated the request.

5 DETAILED DESCRIPTION OF THE FUNCTIONS

A detailed description of the functions available in the PLC300 programable controller for the Modbus communication is provided in this section. In order to elaborate the telegrams it is important to observe the following:

- The values are always transmitted in hexadecimal.
- The address of a datum, the number of data and the value of registers are always represented in 16 bits. Therefore, it is necessary to transmit those fields using two bytes – superior (high) and inferior (low).

5.1 FUNCTION 01 – READ COILS

It reads the content of a group of bits (coils) that must be necessarily in a numerical sequence. This function has the following structure for the request and response telegrams (each field represents a byte):

Request	Response
Function	Function
Address of the initial bit (high byte)	Byte count (number of data bytes)
Address of the initial bit (low byte)	Byte 1
Number of bits (high byte)	Byte 2
Number of bits (low byte)	Byte 3
	etc...

Each response bit is placed in a position of the data bytes sent. The first byte receives the eight first bits from the initial address indicated in the request. The other bytes keep the sequence if the number of reading bits is greater than eight. If the number of read bits is not a multiple of eight, the remaining bits of the last byte must be filled in with zero.

Example: reading of the eight bits of the output marker 2000, mapped as coil from address 24000, considering this marker with value 100 (64h).
- Number of the initial bit: 24000 = 5DC0h
- Number of read bits: 8 = 0008h

Request		Response	
Field	*Value*	*Field*	*Value*
Function	01h	Function	01h
Initial bit (high)	5Dh	Byte count	01h
Initial bit (low)	C0h	Bit status 1 to 8	64h
Number of bits (high)	00h		50h
Number of bits (low)	08h		63h

5.2 FUNCTION 02 – READ INPUT DISCRETE

NOTE!
Function 02 – Read Input Discrete – has exactly the same structure as function 01. Only the function code and accessible data are different.

5.3 FUNCTION 03 – READ HOLDING REGISTER

It reads the content of a group of registers that must be necessarily in a numerical sequence. This function has the following structure for the request and response telegrams (each field represents a byte):

Request	Response
Function	Function
Address of the initial register (high byte)	Byte count
Address of the initial register (low byte)	Datum 1 (high)
Number of registers (high byte)	Datum 1 (low)
Number of registers (low byte)	Datum 2 (high)
	Datum 2 (low)
	etc...

Example: reading of the memory marker %MD0, representing a float IEEE that takes four bytes of the memory. Considering the float value equal to 1.0 (3F800000h in representation of float IEEE).

- Initial register address: 8000 = 1F40h
- Number of read registers: 2 = 0002h

Request		Response	
Field	*Value*	*Field*	*Value*
Function	03h	Function	03h
Initial register (high)	1Fh	Byte Count	04h
Initial register (low)	40h	Float value (low-high)	00h
Number of registers (high)	00h	Float value (low-low)	00h
Number of registers (low)	02h	Float value (high-high)	3Fh
		Float value (high-low)	80h

5.4 FUNCTION 04 – READ INPUT REGISTER

NOTE!
Function 04 – Read Input Register – has exactly the same structure as function 03. Only the function code and accessible data are different.

5.5 FUNCTION 05 – WRITE SINGLE COIL

This function is used to write a value for a single bit (coil). The value for the bit is represented using two bytes, the value FF00h represents the bit in 1, and the value 0000h represents the bit in 0 (zero). It has the following structure (each field represents a byte):

Request	Response
Function	Function
Bit address (byte high)	Bit address (high byte)
Bit address (byte low)	Bit address (low byte)
Value for the bit (byte high)	Value for the bit (high byte)
Value for the bit (byte low)	Value for the bit (low byte)

Example: writing of the first bit of the output marker %QB0, mapped as coil from address 16000.
- Bit number: 16000 = 3E80h
- Bit value: 1, so the value that must be written is FF00h

Request		Response	
Field	*Value*	*Field*	*Value*
Function	05h	Function	05h
Bit number (high)	3Eh	Bit number (high)	1Fh
Bit number (low)	80h	Bit number (low)	40h
Value for the bit (high)	FFh	Value for the bit (high)	FFh
Value for the bit (low)	00h	Value for the bit (low)	00h

Note that for this function the response is an identical copy of the request.

5.6 FUNCTION 06 – WRITE SINGLE REGISTER

This function is used to write a value for a single register. It has the following structure (each field represents a byte):

Request	Response
Function	Function
Register address (high byte)	Register address (high byte)
Register address (low byte)	Register address (low byte)
Value for the register (high byte)	Value for the register (high byte)
Value for the register (low byte)	Value for the register (low byte)

Example: writing of the writing system marker %CB3000. Since the writing is always done by sending a 16-bit register, the bytes mapped at addresses %CB3000 and %CB3001 will be written.
- Initial register address: 3000 = 0BB8h
- Marker value: 50 = 0032h

Request		Response	
Field	*Value*	*Field*	*Value*
Function	06h	Function	06h
Register (high)	0Bh	Register (high)	0Bh
Register (low)	B8h	Register (low)	B8h
Value (high – %CB3001)	00h	Value (high – %CB3001)	00h
Value (low – %CB3000)	32h	Value (low – %CB3000)	32h

Note that for this function the response is an identical copy of the request.

5.7 FUNCTION 15 – WRITE MULTIPLE COILS

This function allows writing values for a group of bits (coils), which must be in a numerical sequence. It can also be used to write in a single bit (each field represents a byte):

Request	Response
Function	Function
Initial bit address (high byte)	Initial bit address (high byte)
Initial bit address (low byte)	Initial bit address (low byte)
Number of bits (high byte)	Number of bits (high byte)
Number of bits (low byte)	Number of bits (low byte)
Byte count (number of data bytes)	
Byte 1	
Byte 2	
Byte 3	
etc...	

The value of each bit being written is placed in a position of the data bytes of the request. The first byte receives the first eight bits from the initial address indicated in the request. The other bytes (if the number of written bits is greater than eight) continue the sequence. If the number of written bits is not multiple of eight, the remaining bits of the last byte must be filled in with zero

Example: writing of 16 bits from output marker %QW0, mapped as coil from address 16000.
- Number of the first bit: 16000 = 3E80h
- Quantity of bits: 16 = 0010h
- Value for the bits 0 to 7: 10 = 0Ah
- Value for bits 8 to 15: 20 = 14h

Request		Response	
Field	*Value*	*Field*	*Value*
Function	0Fh	Function	0Fh
Initial bit (high)	3Eh	Initial bit (high)	1Fh
Initial bit (low)	80h	Initial bit (low)	40h
Number of bits (high)	00h	Number of bits (high)	00h
Number of bits (low)	10h	Number of bits (low)	10h
Byte count	02h		
Value for bits 0 to 7	0Ah		
Value for bits 8 to 15	14h		

5.8 FUNCTION 16 – WRITE MULTIPLE REGISTERS

This function allows writing values for a group of registers, which must be in a numerical sequence. It can also be used to write in a single register (each field represents a byte):

Request	Response
Function	Function
Initial register address (high byte)	Initial register address (high byte)
Initial register address (low byte)	Initial register address (low byte)
Number of registers (high byte)	Number of registers (high byte)
Number of registers (low byte)	Number of registers (low byte)
Byte count (number of data bytes)	
Datum 1 (high)	
Datum 1 (low)	
Datum 2 (high)	
Datum 2 (low)	
etc...	

Example: writing of the writing memory marker %MD0, representing a 32-bit integer value – 4 bytes of the memory. Considering the value to be written equal to 16909060 decimal (01020304h).

- Initial register address: 8000 = 1F40h
- Number of registers: 2 = 0002h

Request		Response	
Field	*Value*	*Field*	*Value*
Function	10h	Function	10h
Initial register (high)	1Fh	Initial register (high)	1Fh
Initial register (low)	40h	Initial register (low)	40h
Number of registers (high)	00h	Number of registers (high)	00h
Number of registers (low)	02h	Number of registers (low)	02h
Byte Count	04h		
Value for the integer (low-high)	03h		
Value for the integer (low-low)	04h		
Value for the integer (high-high)	01h		
Value for the integer (high-low)	02h		

5.9 FUNCTION 43 – READ DEVICE IDENTIFICATION

It is an auxiliary function that allows the reading of the product manufacturer, model and firmware version. It has the following structure:

Request	Response
Function	Function
MEI Type	MEI Type
Reading code	Conformity Level
Object number	More Follows
	Next object
	Number of objects
	Code of the first object
	Size of the first object
	Value of the first object (n bytes)
	Code of the second object
	Size of the second object
	Value of the second object (n bytes)
	etc...

This function allows the reading of three information categories: Basic, Regular and Extended, and each category is formed by a group of objects. Each object is formed by a sequence of ASCII characters. For the PLC300 programable controller, only basic information formed by three objects is available:

- Objeto 00h – VendorName: represents the product manufacturer.
- Objeto 01h – ProductCode: formed by the product code (PLC300).
- Objeto 02h – MajorMinorRevision: it indicates the product firmware version, in the format 'VX.XX'.

The reading code indicates what information categories are read, and if the objects are accessed in sequence or individually. The PLC300 supports the codes 01 (basic information in sequence) and 04 (individual access to the objects). The other fields are specified by the protocol, and for the PLC300 they have fixed values.

Example: reading of basic information in sequence, starting from the object 02h:

Request		Response	
Field	*Value*	*Field*	*Value*
Function	2Bh	Function	2Bh
MEI Type	0Eh	MEI Type	0Eh
Reading code	01h	Reading code	01h
Object number	02h	Conformity Level	81h
		More Follows	00h
		Next object	00h
		Number of objects	01h
		Object code	02h
		Object size	05h
		Object value	'V1.00'

In this example the value of the objects was not represented in hexadecimal, but using the corresponding ASCII characters instead. E.g.: for the object 02h, the value 'V1.00' was transmitted as being five ASCII characters, which in hexadecimal have the values 56h ('V'), 31h ('1'), 2Eh ('.'), 30h ('0') and 30h ('0').

5.10 COMMUNICATION ERRORS

Communication errors may occur in the transmission of telegrams, as well as in the contents of the transmitted telegrams. Transmission and connection errors are directly processed by the Ethernet interface and by the TCP/IP protocol.

In the event of a successful reception, during the treatment of the telegram, the server may detect problems and send an error message, indicating the kind of problem found:

Error code	Description
1	Invalid function: the requested function is not implemented for the equipment.
2	Invalid data address: the data address (register or bit) does not exist.
3	Invalid data value: ▪ Value out of the allowed range. ▪ Writing on data that cannot be changed (read only register or bit).
4	Modbus TCP/RTU Gateway cannot forward message because the slave address is invalid.
10	Modbus TCP/RTU Gateway disabled.
11	Modbus TCP/RTU Gateway identified timeout waiting response from the slave.

NOTE!
It is important that it be possible to identify at the client what type of error occurred, in order to be able to diagnose problems during the communication.

The error messages have the following structure:

Request	Response
Function	Function (with the most significant bit in 1)
Data	Error code

Example: write request to register 2900 (nonexistent register):

Request		Response	
Field	*Value*	*Field*	*Value*
Function	06h	Function	86h
Register (high)	0Bh	Error code	02h
Register (low)	54h		
Value (high)	00h		
Value (low)	00h		

6 OPERATION IN THE MODBUS TCP NETWORK – CLIENT MODE

Besides the operation as a server, the PLC300 programmable controller also allows operation as a client for the Modbus TCP network. For this operation, it is necessary to observe the following points:

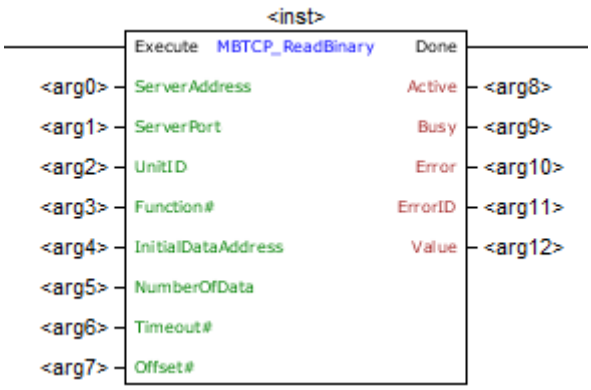
- Sending and receiving telegrams via Ethernet interface using the Modbus TCP protocol is programmed by using blocks in ladder programming language. It is necessary to know the available blocks and the ladder programming software in order to be able to program the network client.
- The following functions are available for the sending of requisitions by the Modbus TCP client:
  Function 01: Read Coils
  Function 02: Read Discrete Inputs
  Function 03: Read Holding Registers
  Function 04: Read Input Registers
  Function 05: Write Single Coil
  Function 06: Write Single Register
  Function 15: Write Multiple Coils
  Function 16: Write Multiple Registers

6.1 BLOCKS TO PROGRAM THE CLIENT

In order to control and monitor the Modbus TCP communication using the PLC300 programmable controller, the following blocks were developed, and they must be used when programming in ladder.

6.1.1 MB TCP Read Binary – Reading of Bits

Block for reading bits. It allows to read up to 128 bits in sequence of the server, using the functions 1 (Read Coils) and 2 (Read Discrete Inputs) of the Modbus.



It has an enabling input of the “Execute" block and a “Done” output, which is activated after the end of the successful execution of the function. After the positive transition of ”Execute", a new telegram is sent by the Modbus TCP client when the client connection is free. At the successful end of the operation – response received from the server – the “Done” output is activated, remaining active while the input is active, and the data received are copied to “Value”. In case of error in the execution of the request, the output “ERROR” is activated and the error code is placed in “ERRORID”.

Inputs:

<arg0>: “ServerAddress” – VAR_IN: insert a variable (tag).
Data type: DWORD
Description: IP address of the server.

<arg1>: “ServerPort” – VAR_IN: insert a variable (tag).
Data type: WORD
Description: Modbus TCP Port of the server.

<arg2>: “UnitID” – VAR_IN: insert a variable (tag).
Data type: BYTE
Description: UnitID of the server.

<arg3>: “Function#” – VAR_IN: insert a constant.
Data type: BYTE
Description: Reading function code: 1= "Read Coils"; 2= "Read Discrete Inputs".

<arg4>: “InitialDataAddress” – VAR_IN: insert a variable (tag).
Data type: WORD
Description: Initial bit address – 0 to 65535.

<arg5>: “NumberOfData” – VAR_IN: insert a variable (tag).
Data type: BYTE
Description: Number of bits read in sequence from the initial address – 1 to 128.

<arg6>: “Timeout#” – VAR_IN: insert a constant.
Data type: WORD
Description: Waiting time for the server response to arrive, from the beginning of the sending by the client – 20 to 5000 ms.

<arg7>: “Offset#” – VAR_IN: insert a constant.
Data type: BOOL
Description: It indicates if the address of the programmed data in "InitialDataAddress#" has offset, that is, if the address of the programmed data in the block must be subtracted from 1 to send by the Modbus network: FALSE= "Without Offset"; TRUE= "With Offset of 1".

Outputs:

<arg8>: “Active” – VAR_OUT: insert a variable (tag).
Data type: BOOL
Description: Active block, reading request sent to the server and waiting for response.
Note: The variable must contain writing permission.

<arg9>: “Busy” – VAR_OUT: insert a variable (tag).
Data type: BOOL
Description: Block enabled, but the resource is not available (connection busy with another request), waiting for release for the request to be sent by the block. If the enabling input is removed while the block executes this indication, the request is discarded.
Note: The variable must contain writing permission.

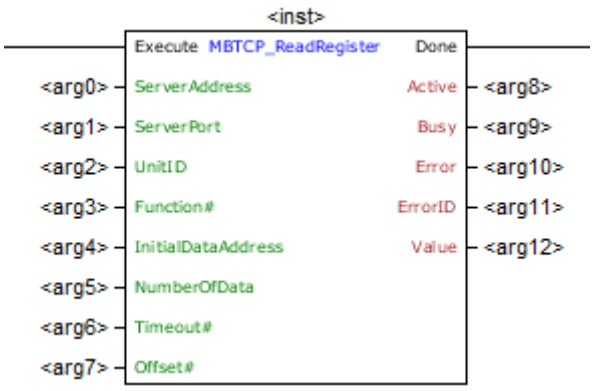
<arg10>: “Error” – VAR_OUT: insert a variable (tag).
Data type: BOOL
Description: Error in the execution of the request.
Note: The variable must contain writing permission.

<arg11>: “ErrorID” – VAR_OUT: insert a variable (tag).
Data type: BYTE, USINT or SINT
Description: In case of error in the request, it indicates the type of error occurred. Possible results: 0= "Executed successfully"; 1= "Some invalid input data"; 2= "Client not enabled"; 4= "Timeout in the server response"; 5= "Server returned error".
Note: The variable must contain writing permission.

<arg12>: “Value” – VAR_OUT: insert a variable (tag).
Data type: BOOL, BOOL[1 … 128]
Description: Variable or array where the data read in the server will be saved.
Note: The variable must contain writing permission.

6.1.2 MB TCP Read Register – Reading of Registers

Block for reading the registers of 16 bits. It allows to read up to 8 registers in sequence of the server, using the functions 3 (Read Holding Registers) and 4 (Read Inputs Registers) of the Modbus.



It has an enabling input of the “Execute" block and a “Done” output, which is activated after the end of the successful execution of the function. After the positive transition of ”Execute", a new telegram is sent by the Modbus TCP client when the connection is free. At the successful end of the operation – response received from the server – the “Done” output is activated, remaining active while the input is active, and the data received are copied to “Value”. In case of error in the execution of the request, the output “ERROR” is activated and the error code is placed in “ERRORID”.

Inputs:

<arg0>: “ServerAddress” – VAR_IN: insert a variable (tag).
Data type: DWORD
Description: IP address of the server.

<arg1>: “ServerPort” – VAR_IN: insert a variable (tag).
Data type: WORD
Description: Modbus Port TCP of the server.

<arg2>: “UnitID” – VAR_IN: insert a variable (tag).
Data type: BYTE
Description: UnitID of the server.

<arg3>: “Function#” – VAR_IN: insert a constant.
Data type: BYTE
Description: Reading function code: 3= "Read Holding Registers"; 4= "Read Input Registers".

<arg4>: “InitialDataAddress” – VAR_IN: insert a variable (tag).
Data type: WORD
Description: Initial register address – 0 to 65535.

<arg5>: “NumberOfData” – VAR_IN: insert a variable (tag).
Data type: BYTE
Description: Number of registers read from the initial address – 1 to 16.

<arg6>: “Timeout#” – VAR_IN: insert a constant.
Data type: WORD
Description: Waiting time for the server response to arrive, from the beginning of the sending by the client – 20 to 5000 ms.

<arg7>: “Offset#” – VAR_IN: insert a constant.
Data type: BOOL
Description: It indicates if the address of the programmed data in "InitialDataAddress#" has offset, that is, if the address of the programmed data in the block must be subtracted from 1 to send by the Modbus network: FALSE= "Without Offset"; TRUE= "With Offset of 1".

Outputs:

<arg8>: “Active” – VAR_OUT: insert a variable (tag).
Data type: BOOL
Description: Active block, reading request sent to the server and waiting for response.
Note: The variable must contain writing permission.

<arg9>: “Busy” – VAR_OUT: insert a variable (tag).
Data type: BOOL
Description: Block enabled, but the resource is not available (connection busy with another request), waiting for release for the requests to be sent by the block. If the enabling input is removed while the block executes this indication, the request is discarded.
Note: The variable must contain writing permission.

<arg10>: “Error” – VAR_OUT: insert a variable (tag).
Data type: BOOL
Description: Error in the execution of the request.
Note: The variable must contain writing permission.

<arg11>: “ErrorID” – VAR_OUT: insert a variable (tag).
Data type: BYTE, USINT or SINT
Description: In case of error in the request, it indicates the type of error occurred. Possible results: 0= "Executed successfully"; 1= "Some invalid input data"; 2= "Client not enabled"; 4= "Timeout in the server response"; 5= "Server returned error".
Note: The variable must contain writing permission.

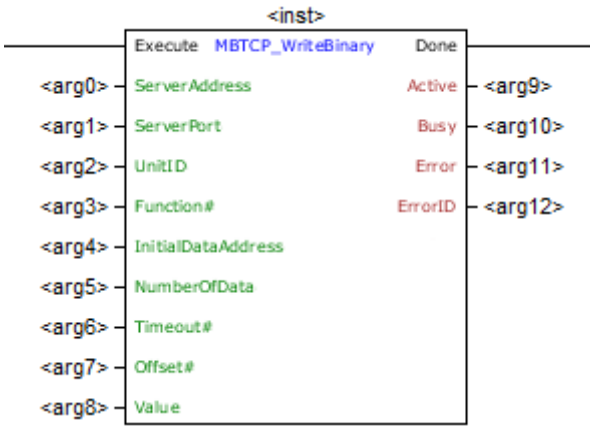
<arg12>: “Value” – VAR_OUT: insert a variable (tag).
Data type: BYTE[2 ... 32], SINT[2 ... 32], USINT[2 ... 32], WORD, WORD[1 ... 16], UINT, UINT[1 ... 16], INT, INT[1 ... 16], DWORD, DWORD[1 ... 8], UDINT, UDINT[1 ... 8], DINT, DINT[1 ... 8], REAL or REAL[1 ... 8]
Description: Variable or array where the data read in the server will be saved.
Note: The variable must contain writing permission.

NOTE!

- The Modbus protocol, using functions 3 and 4, allows the reading of registers of 16 bits only. For reading data with more than 16 bits (one REAL, for instance), it is possible to read multiple registers, and save the value in a variable with size over 16 bits.
- It is important that the number of registers read be compatible with the size of the variable or array where the data will be saved.

6.1.3 MB TCP Write Binary – Writing of Bits

Block for writing bits. It allows to write up to 128 bits using functions 5 (Write Single Coil) and 15 (Write Multiple Coils) of the Modbus.



It has is an enabling input of the “Execute" block and a “Done” output, which is activated after the end of the successful execution of the function. After the positive transition of ”Execute", a new telegram is sent by the Modbus TCP client when the connection is free. At the successful end of the operation – response received from the server – the “Done” output is activated, remaining active while the input is active. In case of error in the execution of the request, the output “ERROR” is activated and the error code is placed in “ERRORID”.

Inputs:

<arg0>: “ServerAddress” – VAR_IN: insert a variable (tag).
Data type: DWORD
Description: Server IP address.

<arg1>: “ServerPort” – VAR_IN: insert a variable (tag).
Data type: WORD
Description: Modbus TCP Port of the server.

<arg2>: “UnitID” – VAR_IN: insert a variable (tag).
Data type: BYTE
Description: UnitID of the server.

<arg3>: “Function#” – VAR_IN: insert a constant.
Data type: BYTE
Description: Writing function code: 5= "Write Single Coil"; 15= "Write Multiple Coils".

<arg4>: “InitialDataAddress” – VAR_IN: insert a variable (tag).
Data type: WORD
Description: Initial bit address – 0 to 65535.

<arg5>: “NumberOfData” – VAR_IN: insert a variable (tag).
Data type: BYTE
Description: Number of bits written in sequence from the initial address – 1 to 128.

<arg6>: “Timeout#” – VAR_IN: insert a constant.
Data type: WORD
Description: Waiting time for the server response to arrive, from the beginning of the sending by the client – 20 to 5000 ms.

<arg7>: “Offset#” – VAR_IN: insert a constant.
Data type: BOOL
Description: It indicates if the address of the programmed data in "InitialDataAddress#" has offset, that is, if the address of the programmed data in the block must be subtracted from 1 to send by the Modbus network: FALSE= "Without Offset"; TRUE= "With Offset of 1".

<arg8>: “Value” – VAR_IN: insert a variable (tag).
Data type: BOOL, BOOL[1 ... 128]
Description: Variable or array with the data that will be written in the server.

Outputs:

<arg9>: “Active” – VAR_OUT: insert a variable (tag).
Data type: BOOL
Description: Active block, writing request sent to the server and waiting for response.
Note: The variable must contain writing permission.

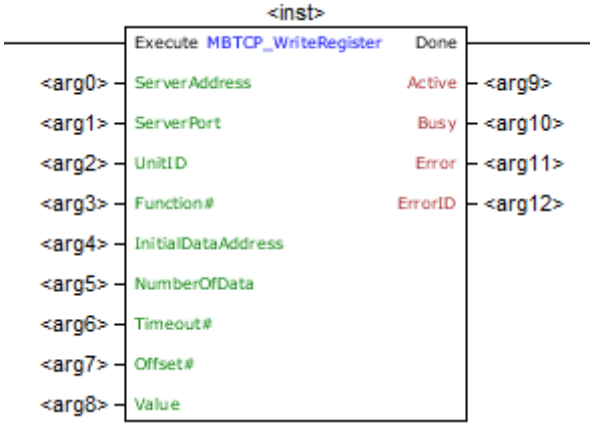
<arg10>: “Busy” – VAR_OUT: insert a variable (tag).
Data type: BOOL
Description: Block enabled, but the resource is not available (connection busy with another request), waiting for release for the request to be sent by the block. If the enabling input is removed while the block executes this indication, the request is discarded.
Note: The variable must contain writing permission.

<arg11>: “Error” – VAR_OUT: insert a variable (tag).
Data type: BOOL
Description: Error in the execution of the request.
Note: The variable must contain writing permission.

<arg12>: “ErrorID” – VAR_OUT: insert a variable (tag).
Data type: BYTE, USINT or SINT
Description: In case of error in the request, it indicates the type of error occurred. Possible results: 0= "Executed successfully"; 1= "Some invalid input data"; 2= "Client not enabled"; 4= "Timeout in the server response"; 5= "Server returned error".
Note: The variable must contain writing permission.

6.1.4 MB TCP Write Register – Writing of Registers

Block for writing registers. It allows to write one or more registers using function 6 (Write Holding Register) or 16 (Write Multiple Registers) of the Modbus.



It has an enabling input of the “Execute" block and a “Done” output, which is activated after the end of the successful execution of the function. After the positive transition of ”Execute", a new telegram is sent by the Modbus TCP client when the connection is free. At the successful end of the operation – response received from the server – the “Done” output is activated, remaining active while the input is active. In case of error in the execution of the request, the output “ERROR” is activated and the error code is placed in “ERRORID”.

Inputs:

<arg0>: “ServerAddress” – VAR_IN: insert a variable (tag).
Data type: DWORD
Description: Server IP address.

<arg1>: “ServerPort” – VAR_IN: insert a variable (tag).
Data type: WORD
Description: Modbus TCP Port of the server.

<arg2>: “UnitID” – VAR_IN: insert a variable (tag).
Data type: BYTE
Description: UnitID of the server.

<arg3>: “Function#” – VAR_IN: insert a constant.
Data type: BYTE
Description: Writing function code: 6= "Write Single Register"; 16= "Write Multiple Registers".

<arg4>: “InitialDataAddress” – VAR_IN: insert a variable (tag).
Data type: WORD
Description: Initial register address – 0 to 65535.

<arg5>: “NumberOfData” – VAR_IN: insert a variable (tag).
Data type: BYTE
Description: Number of registers written from the initial address – 1 to 16.

<arg6>: “Timeout#” – VAR_IN: insert a constant.
Data type: WORD
Description: Waiting time for the server response to arrive, from the beginning of the sending by the client – 20 to 5000 ms.

<arg7>: “Offset#” – VAR_IN: insert a constant.
Data type: BOOL
Description: It indicates if the address of the programmed data in "InitialDataAddress#" has offset, that is, if the address of the programmed data in the block must be subtracted from 1 to send by the Modbus network: FALSE= "Without Offset"; TRUE= "With Offset of 1".

<arg8>: “Value” – VAR_IN: insert a variable (tag).
Data type: BYTE[2 ... 32], USINT[2 ... 32], SINT[2 ... 32], WORD, WORD[1 ... 16], UINT, UINT[1 ... 16], INT, INT[1 ... 16], DWORD, DWORD[1 ... 8], UDINT, UDINT[1 ... 8], DINT, DINT[1 ... 8], REAL or REAL[1 ... 8]
Description: Variable or array with the data that will be written in the server.

Outputs:

<arg9>: “Active” – VAR_OUT: insert a variable (tag).
Data type: BOOL
Description: Active block, writing request sent to the server and waiting for response.
Note: The variable must contain writing permission.

<arg10>: “Busy” – VAR_OUT: insert a variable (tag).
Data type: BOOL
Description: Block enabled, but the resource is not available (connection busy with another request), waiting for release for the request to be sent by the block. If the enabling input is removed while the block executes this indication, the request is discarded.
Note: The variable must contain writing permission.

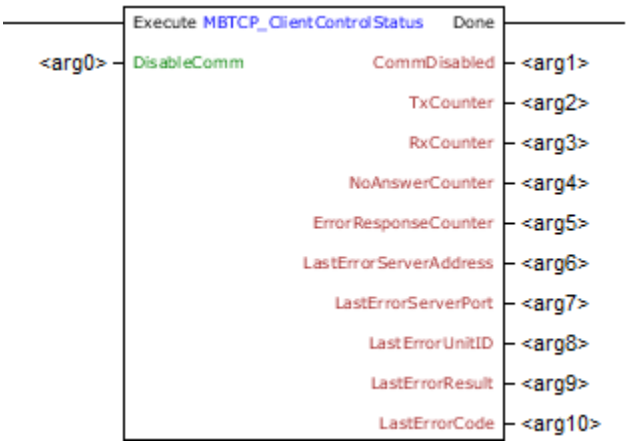
<arg11>: “Error” – VAR_OUT: insert a variable (tag).
Data type: BOOL
Description: Error in the execution of the request.
Note: The variable must contain writing permission.

<arg12>: “ErrorID” – VAR_OUT: insert a variable (tag).
Data type: BYTE, USINT or SINT
Description: In case of error in the request, it indicates the type of error occurred. Possible results: 0= "Executed successfully"; 1= "Some invalid input data"; 2= "Client not enabled"; 4= "Timeout in the server response"; 5= "Server returned error".
Note: The variable must contain writing permission.

NOTE!
- The Modbus protocol, using function 16, allows the writing of registers of 16 bits only. For reading data with more than 16 bits (one REAL, for instance), it is possible to write multiple registers, and use a variable with size over 16 bits as data source.
- It is important that the number of registers written be compatible with the size of the variable or array from where the data are used.

6.1.5 MB TCP Client Control/Status – Control and Status of Modbus TCP Client

Block to control and monitor the Modbus TCP client. Whenever the Modbus TCP network is assembled with the PLC300 as client, it is recommended to use this block to obtain information on the communication state.



It has an enabling input of the “Execute" block and a “Done” output, which is activated after the end of the execution of the function. While the “Execute” enabling input is active, the input data are used and the output data are updated. In case the input is reset, the input values are disregarded and the output arguments are reset. The “Done” output reflects the input value.

Inputs:

<arg0>: “DisableComm” – VAR_IN: insert a constant or a variable (tag).
Data type: BOOL
Description: It allows to disable the Modbus TCP client. When disabling the client, the status markers and counters of the Modbus TCP client are also reset. 0= "Client in execution"; 1= "Disable client".

Outputs:

<arg1>: “CommDisabled” – VAR_OUT: insert a variable (tag).
Data type: BOOL
Description: It indicates whether the client is disabled or not: 0= "Client enabled"; 1= "Client disabled".
Note: The variable must contain writing permission.

<arg2>: “TxCounter” – VAR_OUT: insert a variable (tag).
Data type: WORD or UINT
Description: Counter of requests sent by the client to the servers. It is reset whenever the equipment is shut down or the client is disabled – 0 to 65535.
Note: The variable must contain writing permission.

<arg3>: “RxCounter” – VAR_OUT: insert a variable (tag).
Data type: WORD or UINT
Description: Counter of telegrams received by the client. It is reset whenever the equipment is shut down or the client is disabled – 0 to 65535.
Note: The variable must contain writing permission.

<arg4>: “NoAnswerCounter” – VAR_OUT: insert a variable (tag).
Data type: WORD or UINT
Description: Counter of requests of the client that were not answered by the servers. It is reset whenever the equipment is shut down or the client is disabled – 0 to 65535.
Note: The variable must contain writing permission.

<arg5>: “ErrorResponseCounter” – VAR_OUT: insert a variable (tag).
Data type: WORD or UINT
Description: Counter of client requests in which the servers answered with some error response. The error code can be obtained in the marker that indicates the code of the last error detected. It is reset whenever the equipment is shut down or the client is disabled – 0 to 65535.
Note: The variable must contain writing permission.

<arg6>: “LastErrorServerAddress” – VAR_OUT: insert a variable (tag).
Data type: DWORD
Description: It indicates the IP address of the server in which the last communication error was detected. It is reset whenever the equipment is shut down or the client is disabled.
Note: The variable must contain writing permission.

<arg7>: “LastErrorServerPort” – VAR_OUT: insert a variable (tag).
Data type: WORD or UINT
Description: It indicates the TCP port of the server in which the last communication error was detected. It is reset whenever the equipment is shut down or the client is disabled – 0 to 65535.
Note: The variable must contain writing permission.

<arg8>: “LastErrorUnitID” – VAR_OUT: insert a variable (tag).
Data type: BYTE or USINT
Description: It indicates the Unit ID of the server in which the last communication error was detected. It is reset whenever the equipment is shut down or the client is disabled – 0 to 255.
Note: The variable must contain writing permission

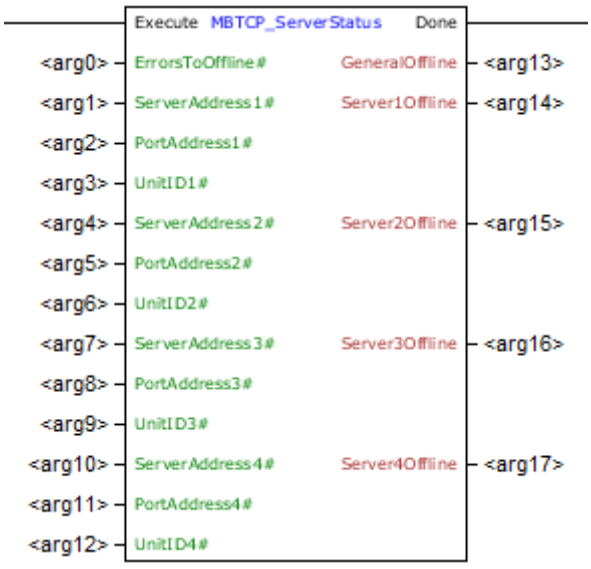
<arg9>: “LastErrorResult” – VAR_OUT: insert a variable (tag).
Data type: BYTE or USINT
Description: It indicates the result of the operation – timeout or error response, according to ERROR ID of the block – for the server in which the last communication error was detected. It is reset whenever the equipment is shut down or the client disabled: 0= "No error detected"; 4= "Timeout in the answer of the server"; 5= "Server returned error", 6 = “Fail to connect to server”, 7 = “TCP/IP connection prematurely terminated”.
Note: The variable must contain writing permission.

<arg10>: “LastErrorCode” – VAR_OUT: insert a variable (tag).
Data type: BYTE or USINT
Description: It indicates the error code in case the client receives and error response from some server. It is reset whenever the equipment is shut down or the client is disabled – 0 to 255.
Note: The variable must contain writing permission.

NOTE!
The data accessed through the use of this block is also available through reading and writing system markers, as described in item 7.

6.1.6 MB TCP Server Status – Modbus TCP Server Status

Block to monitor the servers in the Modbus TCP network. It must be used in case it is desired to identify problems in the communication between the client and some server in the Modbus TCP network.



It has an "Execute" block enabling input, and a "Done" output which is activated after the end of the function's successful performance. While the "Execute" enabling input is active, the input data is used and the output data is updated at every performance of the block. Output "Done" reflects the value of the input.

Inputs:

<arg0>: “ErrorsToSetOffline#” – VAR_IN: insert a constant.
Data type: BYTE or USINT
Description: It allows programming, for this block, the quantity of communication errors which the client must identify until the communication with a network server is considered offline. The following is considered communication error: every request (reading or writing) a client sent to a server and did not receive a response.

<arg1>: “ServerAddress1#” – VAR_IN: insert a constant.
<arg4>: “ServerAddress2#” – VAR_IN: insert a constant.
<arg7>: “ServerAddress3#” – VAR_IN: insert a constant.
<arg10>: “ServerAddress4#” – VAR_IN: insert a constant.
Data type: DWORD
Description: It allows programming the IP address of up to 4 servers which quantity of communication errors will be monitored in order to make known if they are online or offline. In case the quantity of sequential communication errors detected in the reading and writing blocks via Modbus TCP reaches the value programmed in "ErrorsToSetOffline", the respective output is activated. In case it is desired to monitor a smaller number of slaves, any of the inputs may be set in zero: 0= "Ignores input".

<arg2>: “ServerPort1#” – VAR_IN: insert a constant.
<arg5>: “ServerPort2#” – VAR_IN: insert a constant.
<arg8>: “ServerPort3#” – VAR_IN: insert a constant.
<arg11>: “ServerPort4#” – VAR_IN: insert a constant.
Data type: WORD
Description: It allows programming the Modbus TCP port of up to 4 servers which quantity of communication errors will be monitored in order to make known if they are online or offline. In case the quantity of sequential communication errors detected in the reading and writing blocks via Modbus TCP reaches the value programmed in "ErrorsToSetOffline", the respective output is activated.

<arg3>: “UnitID1#” – VAR_IN: insert a constant.
<arg6>: “UnitID2#” – VAR_IN: insert a constant.
<arg9>: “UnitID3#” – VAR_IN: insert a constant.
<arg12>: “UnitID4#” – VAR_IN: insert a constant.
Data type: BYTE
Description: It allows programming the Unit ID of up to 4 servers which quantity of communication errors will be monitored in order to make known if they are online or offline. In case the quantity of sequential communication errors detected in the reading and writing blocks via Modbus TCP reaches the value programmed in "ErrorsToSetOffline", the respective output is activated.

Outputs:

<arg13>: “GeneralOffline#” – VAR_OUT: insert a variable (tag).
Data type: BOOL
Description: If any of the outputs of the indicated servers is activated, this output will also be activated. It works as an OR logic between the 4 outputs of server indication.
Note: The variable must have writing permission.

<arg14>: “Server1Offline#” – VAR_OUT: insert a variable (tag).
<arg15>: “Server2Offline#” – VAR_OUT: insert a variable (tag).
<arg16>: “Server3Offline#” – VAR_OUT: insert a variable (tag).
<arg17>: “Server4Offline#” – VAR_OUT: insert a variable (tag).
Data type: BOOL
Description: Output activated in case the quantity of sequential communication errors for the servers indicated in the respective inputs reaches the value programmed in "ErrorsToSetOffline".
Note: The variable must have writing permission.

7 SYSTEM MARKERS FOR ETHERNET

For the Ethernet interface, the following reading system markers (%S) and writing system markers (%C) were provided for control and monitoring:

7.1 READING SYSTEM MARKERS

Ethernet Interface Status: reading marker set that indicates the status of the Ethernet interface.	
Marker	**Description**
%SB3492 %SB3493 %SB3494 %SB3495 %SB3496 %SB3497	Physical Address (MAC).
%SB3498	Communication mode: 0 = Automatic (Ethernet interface is being configured) 1 = 10 Mbps Full Duplex 2 = 10 Mbps Half Duplex 3 = 100 Mbps Full Duplex 4 = 100 Mbps Half Duplex.
%SB3499	Reserved.
%SD3500	IP Address.
%SD3504	Subnet mask.
%SD3508	Default gateway.

Modbus TCP Server Status: reading marker set that indicates the amount of telegrams sent and received by the Modbus TCP Server.	
Marker	**Description**
%SW3512	Number of received telegrams.
%SW3514	Number of sent telegrams.
%SB3516	Number of active connections.

Modbus TCP Client Status: reading marker set that indicates the status of the Modbus TCP client, besides information for network diagnosis.	
Marker	**Description**
%SB3520	Modbus TCP Client Status: 0 = Normal operation. 1 = Client disabled.
%SB3521	Reserved.
%SW3522	Counter of requests made by the client. Counter incremented every time a new telegram is sent by the Modbus TCP client. It is reset whenever it reaches the maximum limit.
%SW3524	Counter of successfully received responses. Counter incremented every time the client receives a successful response from a server. It is reset whenever it reaches the maximum limit.
%SW3526	Counter of requests without response – timeout. Counter incremented every time a timeout occurs for a request made by the Modbus TCP client to a server. It is reset whenever it reaches the maximum limit or the interface is disabled.
%SW3528	Counter of responses received with error. Counter incremented whenever the server returns an error response to a request made by the Modbus TCP client. It is reset whenever it reaches the maximum limit or the interface is disabled. Whenever this error is detected, the data for the server address, error type and error code will be saved on the markers %SB3530 to %SB3538.
%SW3530	Last error occurred: TCP port of the server.
%SD3532	Last error occurred: server IP address.
%SB3536	Last error occurred: server Unit ID.
%SB3537	Last error occurred: error type. 0 = No error. 4 = Response Timeout. 5 = Server returned error response. 6 = Fail to connect to server. 7 = TCP/IP connection prematurely terminated. It is reset whenever the interface is disabled.
%SB3538	Last error occurred: code of the received error, if the type is error response. It is reset whenever the interface is disabled.

7.2 WRITING SYSTEM MARKERS

Configuration of the Ethernet Interface: set of writing markers to program the configurations of the Ethernet interface. They are also accessible through the Setup menu.	
Marker	**Description**
%CD3424	IP address.
%CD3428	Subnet mask.
%CD3432	Default gateway.
%CB3436	0 = DHCP disabled (default) 1 = DHCP enabled.
%CB3437	Comunication mode: 0 = Auto (default) 1 = 10 Mb Full Duplex 2 = 10 Mb Half Duplex 3 = 100 Mb Full Duplex 4 = 100 Mb Half Duplex.

Configuration of the Modbus TCP Server: set of writing markers to program the configurations of the Modbus TCP server. They are also accessible through the Setup menu.	
Marker	**Description**
%CD3440	IP authentication. If different from zero, only this IP address can connect to the Modbus TCP server.
%CW3444	TCP port (default 502).
%CB3446	UnitID (default 255).
%CB3447	Reserved.
%CW3448	Modbus RTU slave reception timeout (default 1000 ms).

Modbus TCP Client Control: set of writing markers for control of the Modbus TCP client.	
Marker	**Description**
%CW3452	Modbus TCP Client Control: 0 = Normal operation. 1 = Disable interface.

Configuration of the SNTP Client: set of writing markers to program the configurations of the SNTP Client.	
Marker	**Description**
%CD3456	SNTP server IP address.
%CD3460	Redundant SNTP server IP address.
%CW3464	Update frequency.
%CW3466	Reception timeout.

WEG Equipamentos Elétricos S.A.
Jaraguá do Sul - SC - Brazil
Phone 55 (47) 3276-4000 - Fax 55 (47) 3276-4020
São Paulo - SP - Brazil
Phone 55 (11) 5053-2300 - Fax 55 (11) 5052-4212
automacao@weg.net
www.weg.net