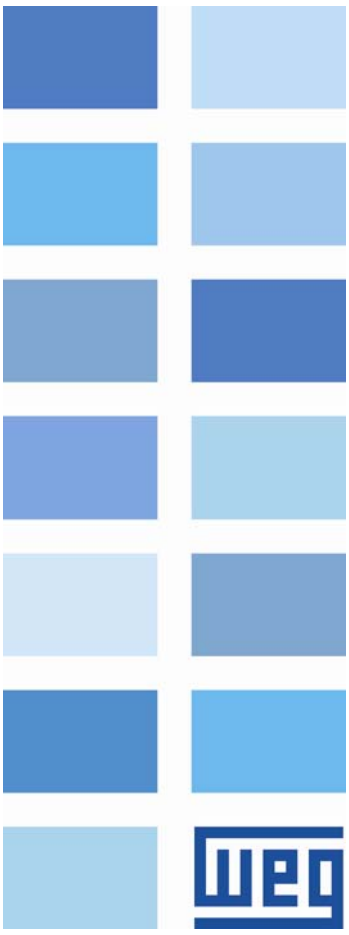
Modbus RTU

SCA06

User's Manual



Modbus RTU User's Manual

Series: SCA06

Language: English

Document Number: 10001625775 / 00

Publication Date: 12/2012

CONTENTS

ABOUT THIS MANUAL

This manual supplies the necessary information for the operation of the SCA06 servo drive using the Modbus protocol. This manual must be used together with the SCA06 user manual.

ABBREVIATIONS AND DEFINITIONS

ASCII	American Standard Code for Information Interchange
CRC	Cycling Redundancy Check
EIA	Electronic Industries Alliance
TIA	Telecommunications Industry Association
RTU	Remote Terminal Unit

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 RTU 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 Protocol Reference Guide, June 1996.	Rev. J	MODICON
MODBUS over Serial Line, December 20th 2006.	V1.02	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 INTRODUCTION TO SERIAL COMMUNICATION

In a serial interface the data bits are sent sequentially through a communication channel or bus. Several technologies use the serial communication for data transfer, including the RS232 and RS485 interfaces.

The directions that specify the RS232 and RS485 standards, however, do neither specify the character format, nor its sequence for the data transmission and reception. Therefore, besides the interface, it is also necessary to identify the protocol used for the communication. Among the several existent protocols, one used a lot in the industry is the Modbus RTU protocol.

In the sequence the characteristics of the RS232 and RS485 serial interfaces available for the product will be presented, as well as the protocols for the use of those interfaces.

2 NETWORK CONNECTIONS

In order to allow the serial communication in the servo drive SCA06, it's necessary to use the expansion and communication module ECO1 described below. Information about the installation of this module can be obtained in the guide that comes with the accessory.

2.1 EXPANSION AND COMMUNICATION RS232 AND RS485 ECO1 MODULE

- WEG Item: 11330271.
- Composed by the communication module ECO1 (in the figure beside) and one assembly guide.
- Galvanically insulated interface and with differential signal, providing more resistance against electromagnetic interference.
- The RS232 and RS485 signals are independent channels, and they may be used simultaneously.

2.2 RS232

2.2.1 Indications

LED LA121 indicates (lights) when there is data transmission by the RS232 communication.

2.2.2 Connection to the RS232 Network

- The inverter signals RX and TX must be respectively connected to the TX and RX signals of the master, besides the connection of the reference signal (GND).
- The interface RS232 is very susceptible to interferences. For this reason, the cable used for communication should be as short as possible – always shorter than 10 meters – and must be placed separately from power wiring that feeds the inverter and the motor.

2.2.3 Cables for Connection in RS232

If desired, items of the following cables are available for connection in RS232 between the servo drive and a network master, like a PC:

Table 2.1: *RS232 Cable*

Cable	Item
RS232 shielded cable with DB9 connectors Length: 3 meters	10050328
RS232 shielded cable with DB9 female connectors Length: 10 meters	10191117

Other cables, however, can be found on the market – commonly referred to as null-modem – or mounted as desired for installation.

2.2.4 Connector Pin Assignment

The connection for the RS232 interface is available via connectors XA121 and XA122 using the following pin assignment:

Table 2.1: *Connector pin assignment for RS232 XA121*

Pin	Function
1	Ground
2	RX_232
3	TX_232
4	Reserved[1]
5	GND
6	Reserved1[1]
7	232 RTS
8	A
9	B

Table 2.2: *Connector pin assignment for RS232 XA122*

Pin	Function
1	NC
2	RX_232
3	TX_232
4	Reserved[1]
5	GND
6	NC
7	232 RTS
8	NC
9	NC
Frame	Ground

2.3 RS485

2.3.1 Indications

LED LA122 indicates (lights) when there is data transmission by the RS485 communication.

2.3.2 Characteristics of RS485 interface

- Interface follows the EIA/TIA-485 standard.
- It can operate as a slave of the Modbus RTU network.
- It allows communication using rates from 9600 to 57600 Kbit/s.
- Galvanically insulated interface and with differential signal, providing more resistance against electromagnetic interference.
- It allows the connection of up to 32 devices to the same segment. A larger number of devices can be connected with the use of repeaters.[2]
- Maximum bus length of 1000 meters.

2.3.3 Connector Pin Assignment

The connection for the RS485 interface is available via XC1 connector using the following pin assignment:

Table 2.3: *Connector pin assignment for RS485 XA121*

Pin	Function
1	Ground
2	RX_232
3	TX_232
4	Reserved[1]
5	GND
6	Reserved[1]
7	232 RTS
8	A
9	B

[1] Do not connect the reserved pins.
[2] The limit of devices that can be connected to the network also depends on the used protocol.

Table 2.4: *Connector pin assignment for RS485 XA123*

Pin	Function
1	NC
2	NC
3	NC
4	NC
5	GND
6	Reserved[1]
7	NC
8	A (date -)
9	B (date +)
Frame	Ground

2.3.4 Termination resistor

For each segment of the RS485 network, it is necessary to enable a termination resistor at the ends of the main bus. The accessory ECO1 has two DIP Switches that can be activated (by putting both SA121 switches in the ON position) to enable the termination resistor.

Figure 2.1: *Termination resistor*

2.3.5 Connection to the RS485 Network

For the connection of the servo drive SCA06 using the RS485 interface, the following points must be observed:

- It's recommended to use a cable with shielded twisted pair.
- It's also recommended that the cable have an additional wire for connecting the reference signal (GND). If the cable doesn't have the additional wire, you should leave the GND signal disconnected.
- The passage of the cable must be done separately (and if possible distant) from the power supply cables.
- All network devices must be properly grounded, preferably to the same connection with the ground. The cable shield must also be grounded.
- Enable the termination resistors only in two points, at the ends of the main bus, even if there are derivations from the bus.

3 PROGRAMMING

Next, the SCA06 servo drive parameters related to the Modbus RTU communication will be presented.

3.1 SYMBOLS FOR THE PROPERTIES DESCRIPTION

RW	Reading and writing parameter
AC	Parameter visible on the HMI only when the corresponding accessory is connected

P00650 – SERVO DRIVE ADDRESS IN THE SERIAL COMMUNICATION 1 – RS232

P00656 – SERVO DRIVE ADDRESS IN THE SERIAL COMMUNICATION 2 – RS485

Range:	1 to 247	**Default:** 1
Properties:	RW, AC	

Description:
It allows programming the used address for serial communication of the equipment. It is necessary that each device on the network have a different address from one another. The valid addresses for this parameter depend on the protocol programmed on the servo drive.

- P00654/P00660 = 1 (WEGTP) → Valid addresses: 1 to 30.
- P00654/P00660 = 2 (Modbus RTU) → Valid addresses: 1 to 247.

P00652 – BIT RATE SERIAL 1 – RS232

P00658 – BIT RATE SERIAL 2 – RS485

Range:	0 = 4800 bits/s 1 = 9600 bits/s 2 = 14400 bits/s 3 = 19200 bits/s 4 = 24000 bits/s 5 = 28800 bits/s 6 = 33600 bits/s 7 = 38400 bits/s 8 = 43200 bits/s 9 = 48000 bits/s 10 = 52800 bits/s 11 = 57600 bits/s	**Default:** 1
Properties:	RW, AC	

Description:
It allows programming the desired value for the baud rate of the serial interface in bits per second. This rate must be the same for all the devices connected to the network.

P00653 – SERIAL CONFIGURATION 1 – RS232

P00659 – SERIAL CONFIGURATION 2 – RS485

Range:	0 = 8 data bits, no parity, 1 stop bit 1 = 8 data bits, parity even, 1 stop bit 2 = 8 data bits, parity odd, 1 stop bit 3 = 8 data bits, no parity, 2 stop bits 4 = 8 data bits, parity even, 2 stop bits 5 = 8 data bits, parity odd, 2 stop bits 6 = 7 data bits, no parity, 1 stop bit 7 = 7 data bits, parity even, 1 stop bit 8 = 7 data bits, parity odd, 1 stop bit 9 = 7 data bits, no parity, 2 stop bits 10 = 7 data bits, parity even, 2 stop bits 11 = 7 data bits, parity odd, 2 stop bits	**Default:** 3
Properties:	RW, AC	

Description:
It allows the configuration of the number of data bits, parity and stop bits in the bytes of the serial interface. This configuration must be the same for all the devices connected to the network.

P00654 – SELECT SERIAL PROTOCOL 1 – RS232

P00660 – SELECT SERIAL PROTOCOL 2 – RS485

Range:	1 = WEGTP 2 = Modbus RTU	**Default:** 1
Properties:	RW	

Description:
It allows selecting the desired protocol for the serial interface. The detailed description of the protocol is presented in item 4 of this manual.

P0662 – ACTION FOR COMMUNICATION ERROR

Range:	0 = Cause Alarm 1 = Cause Fault 2 = Cause alarm and execute STOP 3 = Cause alarm and disable drive	**Default:** 0
Properties:	CFG	

Description:
This parameter allows selecting which action must be executed by the equipment in case it is controlled via network and a communication error is detected.

Table 3.1: *Options for parameter P0662*

Option	Description
0 = Cause Alarm	It just indicates alarm.
1 = Cause Fault	Instead of alarm, a communication error causes a fault on the equipment, and it is necessary to reset the faults so as to return to normal operation.
2 = Execute STOP	The alarm will be indicated together with the execution of the STOP command. It is necessary to reset the faults or disable the drive for the servo to exit this condition.
3 = Disable drive	The alarm will be indicated together with the execution of the disable command.

The followings events are considered communication errors:

Serial Communication (RS232/RS485):
- Alarm A00128/Fault F00028: timeout of the serial interface.

P00663 – WATCHDOG SERIAL

Range:	0.0 to 999.0s	**Default:** 0.0
Properties:	RW	

Description:
It allows programming a time for the detection of the communication error via serial interface. 1 - In case the servo drive does not receive valid telegrams for a period longer than that adjusted in this parameter, it will be assumed a communication error occurred, the alarm A128 will be displayed on the HMI (or fault F228, depending on the settings on P0313) and the action programmed on P0313 will be executed.

After energized, the servo drive will begin counting this time from the first valid telegram received. The value 0,0 disables this function.

P00664 – SAVE PARAMETERS IN NON-VOLATILE MEMORY

Range:	0 = Parameter is not saved in non-volatile memory 1 = Parameter is saved in non-volatile memory	**Default:** 1
Properties:	RW	

Description:
It allows selecting if the writing of parameters via serial must save the content of the parameters in the non-volatile memory (EEPROM) or not. When using the Modbus protocol, it is only this parameter that determines if the parameters written via serial will be saved or not in the non-volatile memory. However, when using the WEGTP protocol, it must be observed that the information about saving or not the parameter in the EEPROM is contained in the telegram code byte. In order to save them in non-volatile memory via WEGTP, it is necessary that these two pieces of information, the telegram code byte and the parameter P00664, be true.

NOTE!
This type of memory features a limited number of records (100,000 times). Depending on the application, this limit can be exceeded if some parameters are written cyclically via serial (speed, torque reference, etc). In these cases, it may be desired that, during the operation of the servo drive, the writing via serial does not save the content of the parameters in non-volatile memory so as not to exceed the number of writings on the servo drive.

NOTE!
This parameter does not apply when writing is performed using the USB interface.

P00667 – SAVE ON MARKERS

Range :	0 = Reads and writes normally the content on the corresponding parameter 1 = Reads and writes content in volatile WORD markers from the MW13000	**Default:** 0
Properties:	RW	

Description:
Property verified when parameter is written and read via serial. It selects whether the content to be written/read must be saved on parameter or in volatile WORD marker.

NOTE!
If this parameter P00667 = 1, when writing in parameter P00105 = 30 via serial, the content of the parameter will be stored in the Word marker 13105 (Initial MW + Even_number => 13000 + 105). Therefore, MW13105 = 30.

Note: Once P00667 = 1, it cannot be changed via serial. Because when trying to write in parameter P00667, you will be writing in Word marker P13667.

4 MODBUS RTU PROTOCOL

The Modbus RTU protocol was initially developed in 1979. Nowadays, it is a widely spread open protocol, used by several manufactures in many equipments.

4.1 TRANSMISSION MODES

Two transmission modes are defined in the protocol specification: ASCII and RTU. The modes define the way the message bytes are transmitted. It is not possible to use the two transmission modes in the same network.

The SCA06 servo drive uses only the RTU mode for the telegram transmission. The bytes are transmitted in hexadecimal format and its configuration depends on the programming done by means of P00659 or P00663.

4.2 MESSAGE STRUCTURE FOR RTU MODE

The Modbus RTU structure uses a master-slave system for message exchange. It allows up to 247 slaves, but only one master. Every communication begins with the master making a request to a slave, which answers to the master what has been asked. In both telegrams (request and answer), the used structure is the same: Address, Function Code, Data and CRC. Only the data field can have a variable size, depending on what is being requested.

Master (request telegram):

Address (1 byte)	Function (1 byte)	Request Data (n bytes)	CRC (2 bytes)

Slave (response telegram):

Address (1 byte)	Function (1 byte)	Response Data (n bytes)	CRC (2 bytes)

4.2.1 Address

The master initiates the communication sending a byte with the address of the slave to which the message is destined. When sending the answer, the slave also initiates the telegram with its own address. The master can also send a message to the address 0 (zero), which means that the message is destined to all the slaves in the network (broadcast). In that case, no slave will answer to the master.

4.2.2 Function Code

This field also contains a single byte, where the master specifies the kind of service or function requested to the slave (reading, writing, etc.). According to the protocol, each function is used to access a specific type of data.

For the available list of supported functions, refer to item 5.

4.2.3 Data Field

It is a variable size field. The format and contents of this field depend on the used function and the transmitted value. This field is described together with the function description (refer to item 5).

4.2.4 CRC

The last part of the telegram is the field for checking the transmission errors. The used method is the CRC-16 (Cycling Redundancy Check). This field is formed by two bytes; where first the least significant byte is transmitted (CRC-), and then the most significant (CRC+). The CRC calculation form is described in the protocol specification; however, information for its implementation is also supplied in the Appendix B.

4.2.5 Time Between Messages

In the RTU mode there is no specific character that indicates the beginning or the end of a telegram. The indication of when a new message begins or when it ends is done by the absence of data transmission in the network, for a minimum period of 3.5 times the transmission time of a data byte (11 bits). Thus, in case a telegram has initiated after the elapsing of this minimum time, the network elements will assume that the first

received character represents the beginning of a new telegram. And in the same manner, the network elements will assume that the telegram has reached its end when after receiving the telegram elements, this time has elapsed again.

If during the transmission of a telegram the time between the bytes is longer than this minimum time, the telegram will be considered invalid because the servo drive will discard the bytes already received and will mount a new telegram with the bytes that were being transmitted.

For communication rates higher than 19200 bits/s, the used times are the same as for that rate. The next table shows us the times for different communication transmission rates:

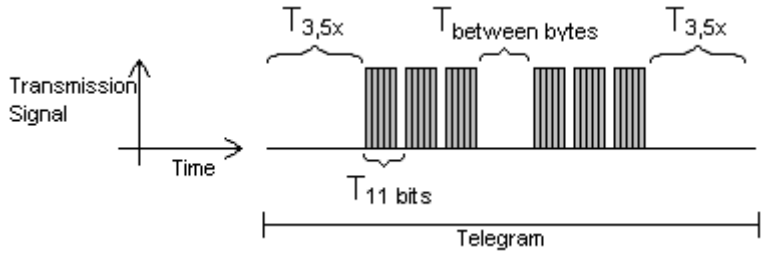


Table 4.1: *Communication rates and the time periods involved in the telegram transmission*

Baud rate	T11 bits	T3,5x
1200 bits/s	9.167 ms	32.083 ms
2400 bits/s	4.583 ms	16.042 ms
4800 bits/s	2.292 ms	8.021 ms
9600 bits/s	1.146 ms	4.010 ms
19200 bits/s	573 µs	2.005 ms
38400 bits/s	573 µs	2.005 ms
57600 bits/s	573 µs	2.005 ms

- $T_{11\ bits}$ = Time for transmitting one byte of the telegram.
- $T_{between\ bytes}$ = Time between bytes.
- $T_{3,5x}$ = Minimum interval to indicated beginning and end of a telegram (3.5 x T11bits).

5 OPERATION IN THE MODBUS RTU NETWORK – SLAVE MODE

The SCA06 servo drive has the following characteristics when operated in Modbus RTU network:

- Network connection via RS485 serial interface.
- Address, communication rate and byte format defined by means of parameters.
- It allows the device programming and control via the access to parameters.

5.1 AVAILABLE FUNCTIONS AND RESPONSE TIMES

In the Modbus RTU specification are defined the functions used to access different types of data. In the SCA06 the parameters have been defined as being holding type registers. In order to access those data the following services (or functions) have been made available:

- Read Coils[3]
  Description: reading of bit blocks of the coil type.
  Function code: 01.

- Read Discrete Inputs[3]
  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[3]
  Description: reading of register blocks of the input register type.
  Function code: 04.

- Write Single Coil[3]
  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[3]
  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 master until the response of the slave, ranges from 2 to 10 ms for any of the functions above.

5.2 MEMORY MAP

The SCA06 Modbus communication is based on the reading/writing of the equipment parameters. All the drive parameters list is made available as holding type 16-bit registers. The data addressing is done with the offset equal to zero, which means that the parameter number corresponds to the register number. The following table illustrates the parameters addressing, which can be accessed as holding type register.

[3] Functions used to access SoftPLC data.

Table 5.1: *Modbus RTU Memory Map*

Parameter number	Modbus data address	
	Decimal	Hexadecimal
P0000	0	0000h
P0001	1	0001h
⋮	⋮	⋮
P0100	100	0064h
⋮	⋮	⋮

It is necessary to know the inverter list of parameters to be able to operate the equipment. Thus, it is possible to identify what data are needed for the status monitoring and the control of the functions. The main parameters are:

Monitoring (reading):
- P0680 (holding register 680): Status word
- P0681 (holding register 681): Motor speed

Command (writing):
- P0682 (holding register 682): Command Word
- P0683 (holding register 683): Speed Reference

Refer to the Programming Manual for a complete parameter list of the equipment.

NOTE!
- All the parameters are treated as holding type registers. Depending on the master that is used, those registers are referenced starting from the base address 40000 or 4x. In this case, the address that must be programmed in the master for a parameter is the address showed in the table above added to the base address. Refer to the master documentation to find out how to access holding type registers.
- It should be noted that read-only parameters can only be read from the equipment, while other parameters can be read and written through the network.
- Besides the parameters, other types of data as bit markers, word or float, can also be accessed using the Modbus RTU interface. Those markers are used mainly by the SoftPLC function, available for the SCA06. Refer to the SoftPLC Manual for the description of those markers, as well as for the addresses via Modbus.

6 DETAILED DESCRIPTION OF THE FUNCTIONS

A detailed description of the functions available in the SCA06 servo drive for the Modbus RTU 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).
- The telegrams for request, as well as for response, cannot exceed 64 bytes.
- The transmitted values are always integer, regardless of having a representation with decimal point. Thus, the value 9.5 would be transmitted via serial as being 95 (5Fh). Refer to the SCA06 parameter list to obtain the resolution used for each parameter.

6.1 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 (Master)	Response (Slave)
Slave Address	Slave Address
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)
CRC-	Datum 2 (low)
CRC+	etc...
	CRC-
	CRC+

Example: reading of the motor speed (P0002) and the motor current (P0003) of slave at address 1 (assuming that P0002 = 1000 rpm and P0003 = 3.5 A).

- Address: 1 = 01h (1 byte)
- Initial register address: 2 = 0002h (2 bytes)
- Value of the fist parameter: 1000 = 03E8h (2 bytes)
- Value of the second parameter: 35 = 0023h (2 bytes)

Request (Master)		Response (Slave)	
Field	*Value*	*Field*	*Value*
Slave Address	01h	Slave Address	01h
Function	03h	Function	03h
Initial register (high)	00h	Byte count	04h
Initial register (low)	02h	P0002 (high)	03h
Number of registers (high)	00h	P0002 (low)	E8h
Number of registers (low)	02h	P0003 (high)	00h
CRC-	65h	P0003 (low)	23h
CRC+	CBh	CRC-	3Bh
		CRC+	9Ah

6.2 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 (Master)	Response (Slave)
Slave Address	Slave Address
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)
CRC-	CRC-
CRC+	CRC+

Example: writing of 2000 rpm as the speed reference (P0121) for the slave at address 3.
- Address: 3 = 03h (1 byte)
- Initial register address: 121 = 0079h (2 bytes)
- Value for parameter: 07D0h (2 bytes)

Request (Master)		**Response (Slave)**	
Field	*Value*	*Field*	*Value*
Slave Address	03h	Slave Address	03h
Function	06h	Function	06h
Register (high)	00h	Register (high)	00h
Register (low)	79h	Register (low)	79h
Value (high)	07h	Value (high)	07h
Value (low)	D0h	Value (low)	D0h
CRC-	5Ah	CRC-	5Ah
CRC+	5Dh	CRC+	5Dh

Note that for this function the slave response is an identical copy of the request made by the master.

6.3 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 (Master)	**Response (Slave)**
Slave Address	Slave Address
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)	CRC-
Datum 1 (high)	CRC+
Datum 1 (low)	
Datum 2 (high)	
Datum 2 (low)	
etc...	
CRC-	
CRC+	

Example: writing of the digital inputs functions (P0300, P0301 and P0302) equal to 4, 4 and 10, respectively, of a slave at address 15.
- Address: 15 = 0Fh (1 byte)
- Initial register address: 300 = 012Ch (2 bytes)
- Value for the fist parameter: 4 = 0004h (2 bytes)
- Value for the second parameter: 4 = 0004h (2 bytes)
- Value for the third parameter: 10 = 000Ah (2 bytes)

Request (Master)		**Response (Slave)**	
Field	*Value*	*Field*	*Value*
Slave Address	0Fh	Slave Address	0Fh
Function	10h	Function	10h
Initial register (high)	01h	Initial register (high)	01h
Initial register (low)	2Ch	Initial register (low)	2Ch
Number of registers (high)	00h	Number of registers (high)	00h
Number of registers (low)	03h	Number of registers (low)	03h
Byte count	06h	CRC-	41h
P0300 (high)	00h	CRC+	13h
P0300 (low)	04h		
P0301 (high)	00h		
P0301 (low)	04h		
P0302 (high)	00h		
P0302 (low)	0Ah		
CRC-	05h		
CRC+	A1h		

6.4 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 (Master)	Response (Slave)
Slave Address	Slave Address
Function	Function
MEI Type	MEI Type
Reading code	Conformity Level
Object number	More Follows
CRC-	Next object
CRC+	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...
	CRC-
	CRC+

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 SCA06 servo drive, only basic information formed by three objects is available:

- Objeto 00h – VendorName: represents the product manufacturer.
- Objeto 01h – ProductCode: formed by the product code (SCA06).
- 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 SCA06 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 SCA06 they have fixed values.

Example: reading of basic information in sequence, starting from the object 02h, from a SCA06 at address 1:

Request (Master)		Response (Slave)	
Field	*Value*	*Field*	*Value*
Slave Address	01h	Slave Address	01h
Function	2Bh	Function	2Bh
MEI Type	0Eh	MEI Type	0Eh
Reading code	01h	Reading code	01h
Object number	02h	Conformity Level	81h
CRC-	70h	More Follows	00h
CRC+	77h	Next object	00h
		Number of objects	01h
		Object code	02h
		Object size	05h
		Object value	'V1.00'
		CRC-	3Ch
		CRC+	53h

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').

6.5 COMMUNICATION ERRORS

Communication errors may occur in the transmission of telegrams, as well as in the contents of the transmitted telegrams. Depending on the type of error, the slave may or not send a response to the master.

When the master sends a message for an inverter configured in a specific network address, the product will not respond to the master if the following occurs:

- Parity bit error.
- CRC error.
- *Timeout* between the transmitted bytes (3.5 times the transmission time of a byte).

In those cases, the master must detect the occurrence of the error by means of the timeout while waiting for the slave response. In the event of a successful reception, during the treatment of the telegram, the slave may detect problems and send an error message, indicating the kind of problem found:

- Invalid function (Error code = 1): The requested function has not been implemented for the equipment.
- Invalid datum address (Error code = 2): the datum address does not exist.
- Invalid datum value (Error code = 3): It occurs in the following situations:
  - The value is out of the permitted range.
  - An attempt to write in a datum that cannot be changed (reading only register/bit).

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

In the event of any of those errors, the slave must send a message to the master indicating the type of error that occurred. The error messages sent by the slave have the following structure:

Request (Master)	**Response (Slave)**
Slave Address	Slave Address
Function	Function (with the most significant bit in 1)
Data	Error code
CRC-	CRC-
CRC+	CRC+

Example: the master requests to the slave at the address 1 the writing in the register 2900 (nonexistent register):

Request (Master)		**Response (Slave)**	
Field	*Value*	*Field*	*Value*
Slave Address	01h	Slave Address	01h
Function	06h	Function	86h
Register (high)	0Bh	Error code	02h
Register (low)	54h	CRC-	C3h
Value (high)	00h	CRC+	A1h
Value (low)	00h		
CRC-	CAh		
CRC+	3Eh		

7 FAULTS AND ALARMS RELATED TO THE MODBUS RTU COMMUNICATION

A00128/F00028 – SERIAL COMMUNICATION TIMEOUT

Description:
It is the only alarm/fault related to the serial communication. It indicates that the device has stopped receiving valid serial telegrams for a period longer than the programmed in P00663.

Working:
The parameter P00663 allows the programming of a period during which the slave must receive at least one valid telegram via the RS-485 serial interface – with address and error checking field correct – otherwise, it will be considered that there was any problem in the serial communication. The time counting initiates after the reception of the first valid telegram.

After the timeout for serial communication is identified, the alarm A00128 or the fault F00028, depending on the P00662 programming, will be signalized through the HMI. In case of alarms, if the communication is reestablished and new valid telegrams are received, the alarm indication will be removed from the HMI.

Possible causes/correction:
- Verify factors that could cause failures in the communication (cables, installation and grounding).
- Make sure that the master sends telegrams to the slave in intervals shorter than the programmed in P00663.
- Disable this function in P00663.

I. APPENDICES

APPENDIX A. ASCII TABLE

Table I.1: *ASCII Characters*

Dec	Hex	Chr		Dec	Hex	Chr	Dec	Hex	Chr	Dec	Hex	Chr
0	00	NUL	(Null char.)	32	20	Sp	64	40	@	96	60	`
1	01	SOH	(Start of Header)	33	21	!	65	41	A	97	61	a
2	02	STX	(Start of Text)	34	22	"	66	42	B	98	62	b
3	03	ETX	(End of Text)	35	23	#	67	43	C	99	63	c
4	04	EOT	(End of Transmission)	36	24	$	68	44	D	100	64	d
5	05	ENQ	(Enquiry)	37	25	%	69	45	E	101	65	e
6	06	ACK	(Acknowledgment)	38	26	&	70	46	F	102	66	f
7	07	BEL	(Bell)	39	27	'	71	47	G	103	67	g
8	08	BS	(Backspace)	40	28	(	72	48	H	104	68	h
9	09	HT	(Horizontal Tab)	41	29	)	73	49	I	105	69	i
10	0A	LF	(Line Feed)	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	(Vertical Tab)	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	(Form Feed)	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	(Carriage Return)	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	(Shift Out)	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	(Shift In)	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	(Data Link Escape)	48	30	0	80	50	P	112	70	p
17	11	DC1	(Device Control 1)	49	31	1	81	51	Q	113	71	q
18	12	DC2	(Device Control 2)	50	32	2	82	52	R	114	72	r
19	13	DC3	(Device Control 3)	51	33	3	83	53	S	115	73	s
20	14	DC4	(Device Control 4)	52	34	4	84	54	T	116	74	t
21	15	NAK	(Negative Acknowledgement)	53	35	5	85	55	U	117	75	u
22	16	SYN	(Synchronous Idle)	54	36	6	86	56	V	118	76	v
23	17	ETB	(End of Trans. Block)	55	37	7	87	57	W	119	77	w
24	18	CAN	(Cancel)	56	38	8	88	58	X	120	78	x
25	19	EM	(End of Medium)	57	39	9	89	59	Y	121	79	y
26	1A	SUB	(Substitute)	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	(Escape)	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	(File Separator)	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	(Group Separator)	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	(Record Separator)	62	3E	>	94	5E	^	126	7E	~
31	1F	US	(Unit Separator)	63	3F	?	95	5F	_	127	7F	DEL

APPENDIX B. CRC CALCULATION USING TABLES

Next, a function using programming language “C” is presented, which implements the CRC calculation for the Modbus RTU protocol. The calculation uses two tables to supply pre-calculated values of the necessary displacement for the calculation.

```
/* Table of CRC values for high–order byte */
static unsigned char auchCRCHi[] = {
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40 };


/* Table of CRC values for low–order byte */
static char auchCRCLo[] = {
0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06, 0x07, 0xC7, 0x05, 0xC5, 0xC4, 0x04,
0xCC, 0x0C, 0x0D, 0xCD, 0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09, 0x08, 0xC8,
0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A, 0x1E, 0xDE, 0xDF, 0x1F, 0xDD, 0x1D, 0x1C, 0xDC,
0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3, 0x11, 0xD1, 0xD0, 0x10,
0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3, 0xF2, 0x32, 0x36, 0xF6, 0xF7, 0x37, 0xF5, 0x35, 0x34, 0xF4,
0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A, 0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38,
0x28, 0xE8, 0xE9, 0x29, 0xEB, 0x2B, 0x2A, 0xEA, 0xEE, 0x2E, 0x2F, 0xEF, 0x2D, 0xED, 0xEC, 0x2C,
0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26, 0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0,
0xA0, 0x60, 0x61, 0xA1, 0x63, 0xA3, 0xA2, 0x62, 0x66, 0xA6, 0xA7, 0x67, 0xA5, 0x65, 0x64, 0xA4,
0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F, 0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68,
0x78, 0xB8, 0xB9, 0x79, 0xBB, 0x7B, 0x7A, 0xBA, 0xBE, 0x7E, 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C,
0xB4, 0x74, 0x75, 0xB5, 0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71, 0x70, 0xB0,
0x50, 0x90, 0x91, 0x51, 0x93, 0x53, 0x52, 0x92, 0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54,
0x9C, 0x5C, 0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B, 0x99, 0x59, 0x58, 0x98,
0x88, 0x48, 0x49, 0x89, 0x4B, 0x8B, 0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42, 0x43, 0x83, 0x41, 0x81, 0x80, 0x40 };


/* The function returns the CRC as a unsigned short type */
unsigned short CRC16(puchMsg, usDataLen)
unsigned char *puchMsg;                 /* message to calculate CRC upon    */
unsigned short usDataLen;               /* quantity of bytes in message     */
{
    unsigned char uchCRCHi = 0xFF;      /* high byte of CRC initialized     */
    unsigned char uchCRCLo = 0xFF;      /* low byte of CRC initialized      */
    unsigned uIndex;                    /* will index into CRC lookup table */
    while (usDataLen--)                 /* pass through message buffer      */
    {
        uIndex = uchCRCLo ^ *puchMsgg++; /* calculate the CRC               */
        uchCRCLo = uchCRCHi ^ auchCRCHi[uIndex};
        uchCRCHi = auchCRCLo[uIndex];
    }

    return (uchCRCHi << 8 | uchCRCLo);
}
```

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