

Modicon Modbus Protocol Reference Guide

PI-MBUS-300 Rev. J

Modicon Modbus Protocol Reference Guide

PI-MBUS-300 Rev. J

June 1996

**MODICON, Inc., Industrial Automation Systems
One High Street
North Andover, Massachusetts 01845**

Preface

This guide is written for the person who will use Modicon Modbus protocols and messages for communication in Modicon programmable controller applications. It describes how messages are constructed, and how transactions take place using Modbus protocol.

This guide should be used in conjunction with Modicon user guides for the types of networks and programmable controllers present in the application. Familiarity with your network layout, and with your control application, is assumed.

The data and illustrations in this book are not binding. We reserve the right to modify our products in line with our policy of continuous product improvement. Information in this document is subject to change without notice and should not be construed as a commitment by Modicon, Inc., Industrial Automation Systems.

Modicon, Inc. assumes no responsibility for any errors that may appear in this document. If you have any suggestions for improvements, or have found any errors in this publication, please notify us.

No part of this document may be reproduced in any form or by any means, electronic or mechanical, without the express written permission of Modicon, Inc., Industrial Automation Systems. All rights reserved.

The following are trademarks of Modicon, Inc.:

Modbus	984	P190	SM85
ModConnect	BM85	RR85	SQ85
Modcom	BP85	SA85	

DEC® is a registered trademark of Digital Equipment Corporation.

VAX™ and DECNET™ are trademarks of Digital Equipment Corporation.

IBM® is a registered trademark of International Business Machines Corporation. IBM AT™, IBM XT™, Micro Channel™, and Personal System/2™ are trademarks of International Business Machines Corporation.

Microsoft® and MS-DOS® are registered trademarks of Microsoft Corporation.

Western Digital® is a registered trademark of Western Digital Corporation.

Ethernet™ is a trademark of Xerox Corporation.

Copyright © 1996, Modicon, Inc.
Printed in U. S. A.

Related Publications

Refer to the following publication for details about the application of Modicon 984 Programmable Controller systems:

GM-0984-SYS 984 Programmable Controller Systems Manual.
Modicon, Inc.

Refer to the following publications for details about the application and installation of the Modbus Plus network and related communications devices:

GM-MBPL-001 Modbus Plus Network Planning and Installation Guide.
Modicon, Inc.

GM-BM85-001 Modbus Plus Bridge/Multiplexer User's Guide.
Modicon, Inc.

Refer to the following publication for details about the Modcom III Communications Software Library for host computer applications:

GM-MC3A-001 Modcom III Communications Software Library User's Guide.
Modicon, Inc.

Contents

Chapter 1 Modbus Protocol	1
Introducing Modbus Protocol	2
Transactions on Modbus Networks	4
Transactions on Other Kinds of Networks	4
The Query–Response Cycle	5
The Two Serial Transmission Modes	6
ASCII Mode	6
RTU Mode	7
Modbus Message Framing	8
ASCII Framing	8
RTU Framing	9
How the Address Field is Handled	10
How the Function Field is Handled	10
Contents of the Data Field	11
Contents of the Error Checking Field	12
How Characters are Transmitted Serially	13
Error Checking Methods	14
Parity Checking	14
LRC Checking	15
CRC Checking	16

Chapter 2 Data and Control Functions	17
Modbus Function Formats	18
How Numerical Values are Expressed	18
Data Addresses in Modbus Messages	18
Field Contents in Modbus Messages	18
Field Contents on Modbus Plus	20
Function Codes Supported by Controllers	22
01 Read Coil Status	24
02 Read Input Status	26
03 Read Holding Registers	28
04 Read Input Registers	30
05 Force Single Coil	32
06 Preset Single Register	34
07 Read Exception Status	36
11 (0B Hex) Fetch Comm Event Ctr	38
12 (0C Hex) Fetch Comm Event Log	40
15 (0F Hex) Force Multiple Coils	44
16 (10 Hex) Preset Multiple Regs	46
17 (11 Hex) Report Slave ID	48
20 (14Hex) Read General Reference	58
21 (15Hex) Write General Reference	62
22 (16Hex) Mask Write 4X Register	66
23 (17Hex) Read/Write 4X Registers	68
24 (18Hex) Read FIFO Queue	70

Chapter 3 Diagnostic Subfunctions	73
Function 08 – Diagnostics	74
Diagnostic Codes Supported by Controllers	76
Diagnostic Subfunctions	77
00 Return Query Data	77
01 Restart Communications Option	77
02 Return Diagnostic Register	78
03 Change ASCII Input Delimiter	81
04 Force Listen Only Mode	81
10 (0A Hex) Clear Counters and Diagnostic Register	81
11 (0B Hex) Return Bus Message Count	82
12 (0C Hex) Return Bus Communication Error Count	82
13 (0D Hex) Return Bus Exception Error Count	82
14 (0E Hex) Return Slave Message Count	83
15 (0F Hex) Return Slave No Response Count	83
16 (10 Hex) Return Slave NAK Count	83
17 (11 Hex) Return Slave Busy Count	84
18 (12 Hex) Return Bus Character Overrun Count	84
19 (13 Hex) Return IOP Overrun Count (884)	84
20 (14 Hex) Clear Overrun Counter and Flag (884)	85
21 (15 Hex) Get/Clear Modbus Plus Statistics	86
Modbus Plus Network Statistics	87
Appendix A Exception Responses	93
Exception Responses	94
Exception Codes	96
Appendix B Application Notes	99
Maximum Query/Response Parameters	100
Estimating Serial Transaction Timing	106
Notes for the 584 and 984A/B/X	108
Appendix C LRC/CRC Generation	109
LRC Generation	110
CRC Generation	112

Figures

Figure 1	Overview of Modbus Protocol Application	3
Figure 2	Master–Slave Query–Response Cycle	5
Figure 3	ASCII Message Frame	8
Figure 4	RTU Message Frame	9
Figure 5	Bit Order (ASCII)	13
Figure 6	Bit Order (RTU)	13
Figure 7	Master Query with ASCII/RTU Framing	19
Figure 8	Slave Response with ASCII/RTU Framing	19
Figure 9	Field Contents on Modbus Plus	21
Figure 10	Read Coil Status – Query	24
Figure 11	Read Coil Status – Response	25
Figure 12	Read Input Status – Query	26
Figure 13	Read Input Status – Response	27
Figure 14	Read Holding Registers – Query	28
Figure 15	Read Holding Registers – Response	29
Figure 16	Read Input Registers – Query	30
Figure 17	Read Input Registers – Response	31
Figure 18	Force Single Coil – Query	32
Figure 19	Force Single Coil – Response	33
Figure 20	Preset Single Register – Query	34
Figure 21	Preset Single Register – Response	35
Figure 22	Read Exception Status – Query	36
Figure 23	Read Exception Status – Response	37
Figure 24	Fetch Communications Event Counter – Query	38
Figure 25	Fetch Communications Event Counter – Response	39
Figure 26	Fetch Communications Event Log – Query	40
Figure 27	Fetch Communications Event Log – Response	41

Figure 28	Force Multiple Coils – Query	45
Figure 29	Force Multiple Coils – Response	45
Figure 30	Preset Multiple Registers – Query	46
Figure 31	Preset Multiple Registers – Response	47
Figure 32	Report Slave ID – Query	48
Figure 33	Report Slave ID – Response	49
Figure 34	Read General Reference – Query	60
Figure 35	Read General Reference – Response	61
Figure 36	Write General Reference – Query	64
Figure 37	Write General Reference – Response	65
Figure 38	Mask Write 4X Register – Query	67
Figure 39	Mask Write 4X Register – Response	67
Figure 40	Read/Write 4X Registers – Query	68
Figure 41	Read/Write 4X Registers – Response	69
Figure 42	Read FIFO Queue – Query	70
Figure 43	Read FIFO Queue – Response	71
Figure 44	Diagnostics – Query	75
Figure 45	Diagnostics – Response	75
Figure 46	Master Query and Slave Exception Response	95
Figure 47	LRC Character Sequence	110
Figure 48	CRC Byte Sequence	113

Chapter 1

Modbus Protocol

- Introducing Modbus Protocol
- The Two Serial Transmission Modes
- Modbus Message Framing
- Error Checking Methods

Introducing Modbus Protocol

Modicon programmable controllers can communicate with each other and with other devices over a variety of networks. Supported networks include the Modicon Modbus and Modbus Plus industrial networks, and standard networks such as MAP and Ethernet. Networks are accessed by built-in ports in the controllers or by network adapters, option modules, and gateways that are available from Modicon. For original equipment manufacturers, Modicon ModConnect 'partner' programs are available for closely integrating networks like Modbus Plus into proprietary product designs.

The common language used by all Modicon controllers is the Modbus protocol. This protocol defines a message structure that controllers will recognize and use, regardless of the type of networks over which they communicate. It describes the process a controller uses to request access to another device, how it will respond to requests from the other devices, and how errors will be detected and reported. It establishes a common format for the layout and contents of message fields.

The Modbus protocol provides the internal standard that the Modicon controllers use for parsing messages. During communications on a Modbus network, the protocol determines how each controller will know its device address, recognize a message addressed to it, determine the kind of action to be taken, and extract any data or other information contained in the message. If a reply is required, the controller will construct the reply message and send it using Modbus protocol.

On other networks, messages containing Modbus protocol are imbedded into the frame or packet structure that is used on the network. For example, Modicon network controllers for Modbus Plus or MAP, with associated application software libraries and drivers, provide conversion between the imbedded Modbus message protocol and the specific framing protocols those networks use to communicate between their node devices.

This conversion also extends to resolving node addresses, routing paths, and error-checking methods specific to each kind of network. For example, Modbus device addresses contained in the Modbus protocol will be converted into node addresses prior to transmission of the messages. Error-checking fields will also be applied to message packets, consistent with each network's protocol. At the final point of delivery, however – for example, a controller – the contents of the imbedded message, written using Modbus protocol, define the action to be taken.

Figure 1 shows how devices might be interconnected in a hierarchy of networks that employ widely differing communication techniques. In message transactions, the Modbus protocol imbedded into each network's packet structure provides the common language by which the devices can exchange data.

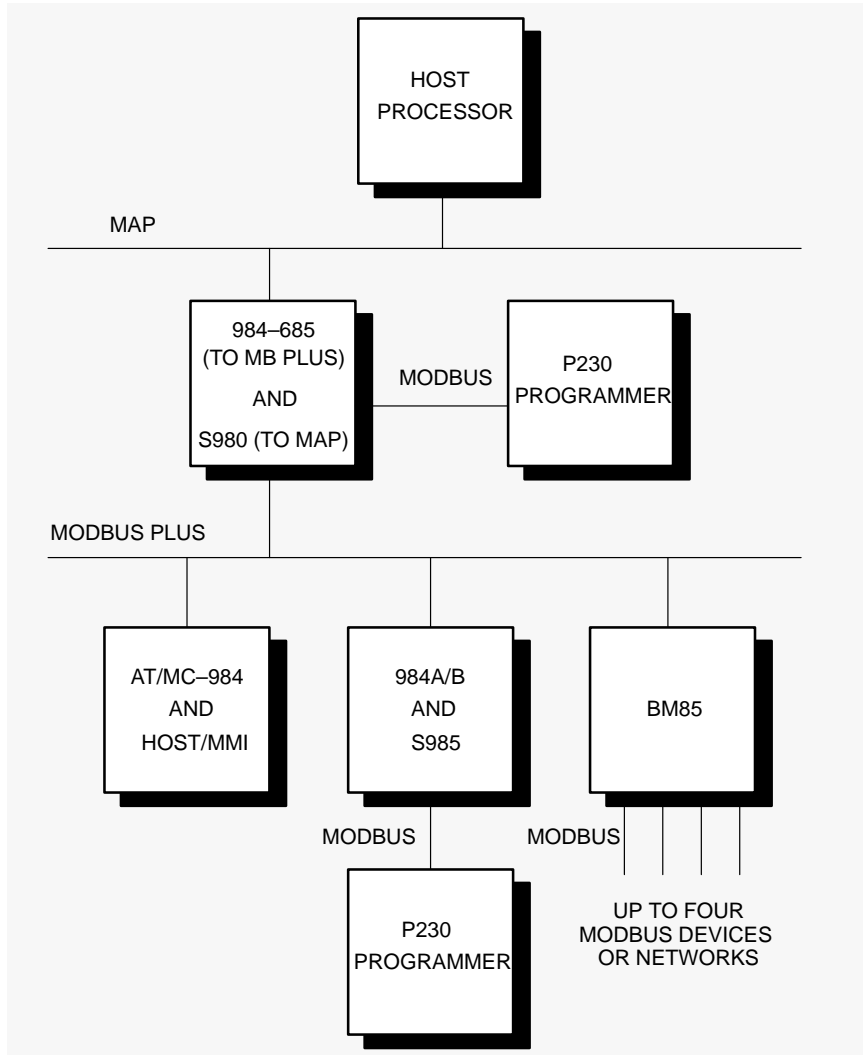


Figure 1 Overview of Modbus Protocol Application

Introducing Modbus Protocol (Continued)

Transactions on Modbus Networks

Standard Modbus ports on Modicon controllers use an RS-232C compatible serial interface that defines connector pinouts, cabling, signal levels, transmission baud rates, and parity checking. Controllers can be networked directly or via modems.

Controllers communicate using a master-slave technique, in which only one device (the master) can initiate transactions (called 'queries'). The other devices (the slaves) respond by supplying the requested data to the master, or by taking the action requested in the query. Typical master devices include host processors and programming panels. Typical slaves include programmable controllers.

The master can address individual slaves, or can initiate a broadcast message to all slaves. Slaves return a message (called a 'response') to queries that are addressed to them individually. Responses are not returned to broadcast queries from the master.

The Modbus protocol establishes the format for the master's query by placing into it the device (or broadcast) address, a function code defining the requested action, any data to be sent, and an error-checking field. The slave's response message is also constructed using Modbus protocol. It contains fields confirming the action taken, any data to be returned, and an error-checking field. If an error occurred in receipt of the message, or if the slave is unable to perform the requested action, the slave will construct an error message and send it as its response.

Transactions on Other Kinds of Networks

In addition to their standard Modbus capabilities, some Modicon controller models can communicate over Modbus Plus using built-in ports or network adapters, and over MAP, using network adapters.

On these networks, the controllers communicate using a peer-to-peer technique, in which any controller can initiate transactions with the other controllers. Thus a controller may operate either as a slave or as a master in separate transactions. Multiple internal paths are frequently provided to allow concurrent processing of master and slave transactions.

At the message level, the Modbus protocol still applies the master–slave principle even though the network communication method is peer–to–peer. If a controller originates a message, it does so as a master device, and expects a response from a slave device. Similarly, when a controller receives a message it constructs a slave response and returns it to the originating controller.

The Query–Response Cycle

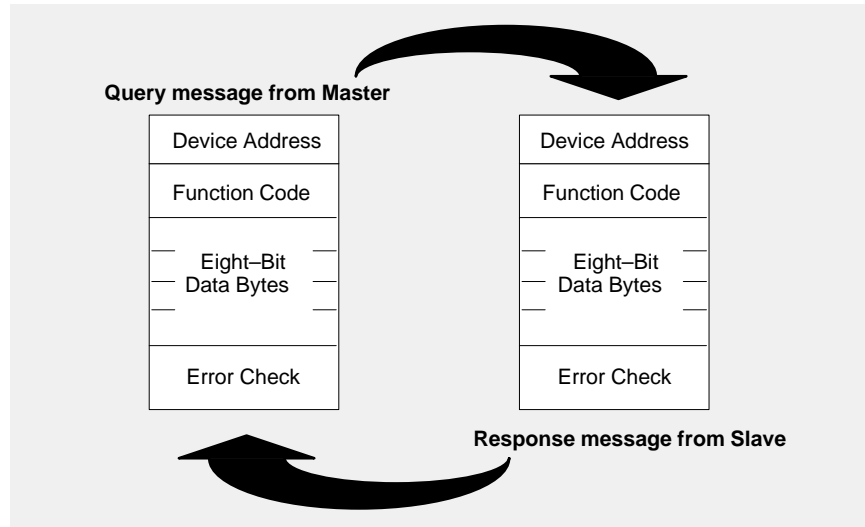


Figure 2 Master–Slave Query–Response Cycle

The Query: The function code in the query tells the addressed slave device what kind of action to perform. The data bytes contain any additional information that the slave will need to perform the function. For example, function code 03 will query the slave to read holding registers and respond with their contents. The data field must contain the information telling the slave which register to start at and how many registers to read. The error check field provides a method for the slave to validate the integrity of the message contents.

The Response: If the slave makes a normal response, the function code in the response is an echo of the function code in the query. The data bytes contain the data collected by the slave, such as register values or status. If an error occurs, the function code is modified to indicate that the response is an error response, and the data bytes contain a code that describes the error. The error check field allows the master to confirm that the message contents are valid.

The Two Serial Transmission Modes

Controllers can be setup to communicate on standard Modbus networks using either of two transmission modes: ASCII or RTU. Users select the desired mode, along with the serial port communication parameters (baud rate, parity mode, etc), during configuration of each controller. The mode and serial parameters *must be the same for all devices on a Modbus network* .

The selection of ASCII or RTU mode pertains only to standard Modbus networks. It defines the bit contents of message fields transmitted serially on those networks. It determines how information will be packed into the message fields and decoded.

On other networks like MAP and Modbus Plus, Modbus messages are placed into frames that are not related to serial transmission. For example, a request to read holding registers can be handled between two controllers on Modbus Plus without regard to the current setup of either controller's serial Modbus port.

ASCII Mode

When controllers are setup to communicate on a Modbus network using ASCII (American Standard Code for Information Interchange) mode, each 8-bit byte in a message is sent as two ASCII characters. The main advantage of this mode is that it allows time intervals of up to one second to occur between characters without causing an error.

The format for each byte in ASCII mode is:

Coding System:	Hexadecimal, ASCII characters 0–9, A–F One hexadecimal character contained in each ASCII character of the message
Bits per Byte:	1 start bit 7 data bits, least significant bit sent first 1 bit for even/odd parity; no bit for no parity 1 stop bit if parity is used; 2 bits if no parity
Error Check Field:	Longitudinal Redundancy Check (LRC)

RTU Mode

When controllers are setup to communicate on a Modbus network using RTU (Remote Terminal Unit) mode, each 8-bit byte in a message contains two 4-bit hexadecimal characters. The main advantage of this mode is that its greater character density allows better data throughput than ASCII for the same baud rate. Each message must be transmitted in a continuous stream.

The format for each byte in RTU mode is:

Coding System:	8-bit binary, hexadecimal 0-9, A-F Two hexadecimal characters contained in each 8-bit field of the message
Bits per Byte:	1 start bit 8 data bits, least significant bit sent first 1 bit for even/odd parity; no bit for no parity 1 stop bit if parity is used; 2 bits if no parity
Error Check Field:	Cyclical Redundancy Check (CRC)

Modbus Message Framing

In either of the two serial transmission modes (ASCII or RTU), a Modbus message is placed by the transmitting device into a frame that has a known beginning and ending point. This allows receiving devices to begin at the start of the message, read the address portion and determine which device is addressed (or all devices, if the message is broadcast), and to know when the message is completed. Partial messages can be detected and errors can be set as a result.

On networks like MAP or Modbus Plus, the network protocol handles the framing of messages with beginning and end delimiters that are specific to the network. Those protocols also handle delivery to the destination device, making the Modbus address field imbedded in the message unnecessary for the actual transmission. (The Modbus address is converted to a network node address and routing path by the originating controller or its network adapter.)

ASCII Framing

In ASCII mode, messages start with a 'colon' (:) character (ASCII 3A hex), and end with a 'carriage return – line feed' (CRLF) pair (ASCII 0D and 0A hex).

The allowable characters transmitted for all other fields are hexadecimal 0–9, A–F. Networked devices monitor the network bus continuously for the 'colon' character. When one is received, each device decodes the next field (the address field) to find out if it is the addressed device.

Intervals of up to one second can elapse between characters within the message. If a greater interval occurs, the receiving device assumes an error has occurred. A typical message frame is shown below.

START	ADDRESS	FUNCTION	DATA	LRC CHECK	END
1 CHAR :	2 CHARS	2 CHARS	<i>n</i> CHARS	2 CHARS	2 CHARS CRLF

Figure 3 ASCII Message Frame

Exception: With the 584 and 984A/B/X controllers, an ASCII message can normally terminate after the LRC field without the CRLF characters being sent. An interval of at least one second must then occur. If this happens, the controller will assume that the message terminated normally.

RTU Framing

In RTU mode, messages start with a silent interval of at least 3.5 character times. This is most easily implemented as a multiple of character times at the baud rate that is being used on the network (shown as T1–T2–T3–T4 in the figure below). The first field then transmitted is the device address.

The allowable characters transmitted for all fields are hexadecimal 0–9, A–F. Networked devices monitor the network bus continuously, including during the ‘silent’ intervals. When the first field (the address field) is received, each device decodes it to find out if it is the addressed device.

Following the last transmitted character, a similar interval of at least 3.5 character times marks the end of the message. A new message can begin after this interval.

The entire message frame must be transmitted as a continuous stream. If a silent interval of more than 1.5 character times occurs before completion of the frame, the receiving device flushes the incomplete message and assumes that the next byte will be the address field of a new message.

Similarly, if a new message begins earlier than 3.5 character times following a previous message, the receiving device will consider it a continuation of the previous message. This will set an error, as the value in the final CRC field will not be valid for the combined messages. A typical message frame is shown below.

START	ADDRESS	FUNCTION	DATA	CRC CHECK	END
T1–T2–T3–T4	8 BITS	8 BITS	$n \times 8$ BITS	16 BITS	T1–T2–T3–T4

Figure 4 RTU Message Frame

Modbus Message Framing (Continued)

How the Address Field is Handled

The address field of a message frame contains two characters (ASCII) or eight bits (RTU). Valid slave device addresses are in the range of 0 – 247 decimal. The individual slave devices are assigned addresses in the range of 1 – 247. A master addresses a slave by placing the slave address in the address field of the message. When the slave sends its response, it places its own address in this address field of the response to let the master know which slave is responding.

Address 0 is used for the broadcast address, which all slave devices recognize. When Modbus protocol is used on higher level networks, broadcasts may not be allowed or may be replaced by other methods. For example, Modbus Plus uses a shared global database that can be updated with each token rotation.

How the Function Field is Handled

The function code field of a message frame contains two characters (ASCII) or eight bits (RTU). Valid codes are in the range of 1 – 255 decimal. Of these, some codes are applicable to all Modicon controllers, while some codes apply only to certain models, and others are reserved for future use. Current codes are described in Chapter 2.

When a message is sent from a master to a slave device the function code field tells the slave what kind of action to perform. Examples are to read the ON/OFF states of a group of discrete coils or inputs; to read the data contents of a group of registers; to read the diagnostic status of the slave; to write to designated coils or registers; or to allow loading, recording, or verifying the program within the slave.

When the slave responds to the master, it uses the function code field to indicate either a normal (error-free) response or that some kind of error occurred (called an exception response). For a normal response, the slave simply echoes the original function code. For an exception response, the slave returns a code that is equivalent to the original function code with its most-significant bit set to a logic 1.

For example, a message from master to slave to read a group of holding registers would have the following function code:

0000 0011 (Hexadecimal 03)

If the slave device takes the requested action without error, it returns the same code in its response. If an exception occurs, it returns:

1000 0011 (Hexadecimal 83)

In addition to its modification of the function code for an exception response, the slave places a unique code into the data field of the response message. This tells the master what kind of error occurred, or the reason for the exception.

The master device's application program has the responsibility of handling exception responses. Typical processes are to post subsequent retries of the message, to try diagnostic messages to the slave, and to notify operators.

Contents of the Data Field

The data field is constructed using sets of two hexadecimal digits, in the range of 00 to FF hexadecimal. These can be made from a pair of ASCII characters, or from one RTU character, according to the network's serial transmission mode.

The data field of messages sent from a master to slave devices contains additional information which the slave must use to take the action defined by the function code. This can include items like discrete and register addresses, the quantity of items to be handled, and the count of actual data bytes in the field.

For example, if the master requests a slave to read a group of holding registers (function code 03), the data field specifies the starting register and how many registers are to be read. If the master writes to a group of registers in the slave (function code 10 hexadecimal), the data field specifies the starting register, how many registers to write, the count of data bytes to follow in the data field, and the data to be written into the registers.

If no error occurs, the data field of a response from a slave to a master contains the data requested. If an error occurs, the field contains an exception code that the master application can use to determine the next action to be taken.

The data field can be nonexistent (of zero length) in certain kinds of messages. For example, in a request from a master device for a slave to respond with its communications event log (function code 0B hexadecimal), the slave does not require any additional information. The function code alone specifies the action.

Modbus Message Framing (Continued)

Contents of the Error Checking Field

Two kinds of error-checking methods are used for standard Modbus networks. The error checking field contents depend upon the method that is being used.

ASCII

When ASCII mode is used for character framing, the error checking field contains two ASCII characters. The error check characters are the result of a Longitudinal Redundancy Check (LRC) calculation that is performed on the message contents, exclusive of the beginning 'colon' and terminating CRLF characters.

The LRC characters are appended to the message as the last field preceding the CRLF characters.

RTU

When RTU mode is used for character framing, the error checking field contains a 16-bit value implemented as two 8-bit bytes. The error check value is the result of a Cyclical Redundancy Check calculation performed on the message contents.

The CRC field is appended to the message as the last field in the message. When this is done, the low-order byte of the field is appended first, followed by the high-order byte. The CRC high-order byte is the last byte to be sent in the message.

Additional information about error checking is contained later in this chapter. Detailed steps for generating LRC and CRC fields can be found in Appendix C.

How Characters are Transmitted Serially

When messages are transmitted on standard Modbus serial networks, each character or byte is sent in this order (left to right):

Least Significant Bit (LSB) . . . Most Significant Bit (MSB)

With ASCII character framing, the bit sequence is:

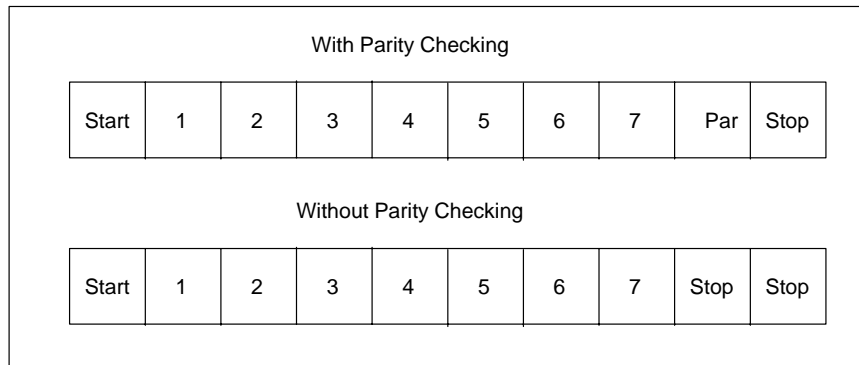


Figure 5 Bit Order (ASCII)

With RTU character framing, the bit sequence is:

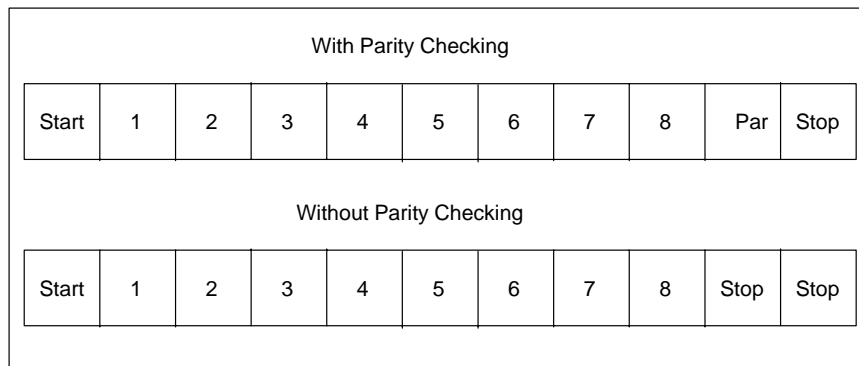


Figure 6 Bit Order (RTU)

Error Checking Methods

Standard Modbus serial networks use two kinds of error checking. Parity checking (even or odd) can be optionally applied to each character. Frame checking (LRC or CRC) is applied to the entire message. Both the character check and message frame check are generated in the master device and applied to the message contents before transmission. The slave device checks each character and the entire message frame during receipt.

The master is configured by the user to wait for a predetermined timeout interval before aborting the transaction. This interval is set to be long enough for any slave to respond normally. If the slave detects a transmission error, the message will not be acted upon. The slave will not construct a response to the master. Thus the timeout will expire and allow the master's program to handle the error. Note that a message addressed to a nonexistent slave device will also cause a timeout.

Other networks such as MAP or Modbus Plus use frame checking at a level above the Modbus contents of the message. On those networks, the Modbus message LRC or CRC check field does not apply. In the case of a transmission error, the communication protocols specific to those networks notify the originating device that an error has occurred, and allow it to retry or abort according to how it has been setup. If the message is delivered, but the slave device cannot respond, a timeout error can occur which can be detected by the master's program.

Parity Checking

Users can configure controllers for Even or Odd Parity checking, or for No Parity checking. This will determine how the parity bit will be set in each character.

If either Even or Odd Parity is specified, the quantity of 1 bits will be counted in the data portion of each character (seven data bits for ASCII mode, or eight for RTU). The parity bit will then be set to a 0 or 1 to result in an Even or Odd total of 1 bits.

For example, these eight data bits are contained in an RTU character frame:

1100 0101

The total quantity of 1 bits in the frame is four. If Even Parity is used, the frame's parity bit will be a 0, making the total quantity of 1 bits still an even number (four). If Odd Parity is used, the parity bit will be a 1, making an odd quantity (five).

When the message is transmitted, the parity bit is calculated and applied to the frame of each character. The receiving device counts the quantity of 1 bits and sets an error if they are not the same as configured for that device (all devices on the Modbus network must be configured to use the same parity check method).

Note that parity checking can only detect an error if an odd number of bits are picked up or dropped in a character frame during transmission. For example, if Odd Parity checking is employed, and two 1 bits are dropped from a character containing three 1 bits, the result is still an odd count of 1 bits.

If No Parity checking is specified, no parity bit is transmitted and no parity check can be made. An additional stop bit is transmitted to fill out the character frame.

LRC Checking

In ASCII mode, messages include an error-checking field that is based on a Longitudinal Redundancy Check (LRC) method. The LRC field checks the contents of the message, exclusive of the beginning 'colon' and ending CRLF pair. It is applied regardless of any parity check method used for the individual characters of the message.

The LRC field is one byte, containing an 8-bit binary value. The LRC value is calculated by the transmitting device, which appends the LRC to the message. The receiving device calculates an LRC during receipt of the message, and compares the calculated value to the actual value it received in the LRC field. If the two values are not equal, an error results.

The LRC is calculated by adding together successive 8-bit bytes of the message, discarding any carries, and then two's complementing the result. It is performed on the ASCII message field contents *excluding* the 'colon' character that begins the message, and *excluding* the CRLF pair at the end of the message.

In ladder logic, the CKSM function calculates a LRC from the message contents. For applications using host computers, a detailed example of LRC generation is contained in Appendix C.

Error Checking Methods (Continued)

CRC Checking

In RTU mode, messages include an error-checking field that is based on a Cyclical Redundancy Check (CRC) method. The CRC field checks the contents of the entire message. It is applied regardless of any parity check method used for the individual characters of the message.

The CRC field is two bytes, containing a 16-bit binary value. The CRC value is calculated by the transmitting device, which appends the CRC to the message. The receiving device recalculates a CRC during receipt of the message, and compares the calculated value to the actual value it received in the CRC field. If the two values are not equal, an error results.

The CRC is started by first preloading a 16-bit register to all 1's. Then a process begins of applying successive 8-bit bytes of the message to the current contents of the register. Only the eight bits of data in each character are used for generating the CRC. Start and stop bits, and the parity bit, do not apply to the CRC.

During generation of the CRC, each 8-bit character is exclusive ORed with the register contents. Then the result is shifted in the direction of the least significant bit (LSB), with a zero filled into the most significant bit (MSB) position. The LSB is extracted and examined. If the LSB was a 1, the register is then exclusive ORed with a preset, fixed value. If the LSB was a 0, no exclusive OR takes place.

This process is repeated until eight shifts have been performed. After the last (eighth) shift, the next 8-bit byte is exclusive ORed with the register's current value, and the process repeats for eight more shifts as described above. The final contents of the register, after all the bytes of the message have been applied, is the CRC value.

When the CRC is appended to the message, the low-order byte is appended first, followed by the high-order byte.

In ladder logic, the CKSM function calculates a CRC from the message contents. For applications using host computers, a detailed example of CRC generation is contained in Appendix C.

Chapter 2

Data and Control Functions

- Modbus Function Formats
- A Summary of Function Codes
- Details of Modbus Functions

Modbus Function Formats

How Numerical Values are Expressed

Unless specified otherwise, numerical values (such as addresses, codes, or data) are expressed as *decimal values in the text* of this section. They are expressed as *hexadecimal values in the message fields* of the figures,

Data Addresses in Modbus Messages

All data addresses in Modbus messages are referenced to zero. The first occurrence of a data item is addressed as item number zero. For example:

- The coil known as 'coil 1' in a programmable controller is addressed as coil 0000 in the data address field of a Modbus message.
- Coil 127 decimal is addressed as coil 007E hex (126 decimal).
- Holding register 40001 is addressed as register 0000 in the data address field of the message. The function code field already specifies a 'holding register' operation. Therefore the '4XXXX' reference is implicit.
- Holding register 40108 is addressed as register 006B hex (107 decimal).

Field Contents in Modbus Messages

Figure 7 shows an example of a Modbus query message. Figure 8 is an example of a normal response. Both examples show the field contents in hexadecimal, and also show how a message could be framed in ASCII or in RTU mode.

The master query is a Read Holding Registers request to slave device address 06. The message requests data from three holding registers, 40108 through 40110. Note that the message specifies the starting register address as 0107 (006B hex).

The slave response echoes the function code, indicating this is a normal response. The 'Byte Count' field specifies how many *8-bit data items* are being returned.

It shows the count of 8-bit bytes to follow in the data, for either ASCII or RTU.. With ASCII, this value is one-half the actual count of ASCII characters in the data. In ASCII, each 4-bit hexadecimal value requires one ASCII character, therefore two ASCII characters must follow in the message to contain each 8-bit data item.

For example, the value 63 hex is sent as one 8-bit byte in RTU mode (01100011). The same value sent in ASCII mode requires two bytes, for ASCII '6' (0110110) and '3' (0110011). The 'Byte Count' field counts this data as one 8-bit item, regardless of the character framing method (ASCII or RTU).

How to Use the Byte Count Field: When you construct responses in buffers, use a Byte Count value that equals the count of 8-bit bytes in your message data. The value is exclusive of all other field contents, including the Byte Count field. Figure 8 shows how the byte count field is implemented in a typical response.

QUERY			
Field Name	Example (Hex)	ASCII Characters	RTU 8-Bit Field
Header		: (colon)	None
Slave Address	06	0 6	0000 0110
Function	03	0 3	0000 0011
Starting Address Hi	00	0 0	0000 0000
Starting Address Lo	6B	6 B	0110 1011
No. of Registers Hi	00	0 0	0000 0000
No. of Registers Lo	03	0 3	0000 0011
Error Check		LRC (2 chars.)	CRC (16 bits)
Trailer		CR LF	None
Total Bytes:		17	8

Figure 7 Master Query with ASCII/RTU Framing

RESPONSE			
Field Name	Example (Hex)	ASCII Characters	RTU 8-Bit Field
Header		: (colon)	None
Slave Address	06	0 6	0000 0110
Function	03	0 3	0000 0011
Byte Count	06	0 6	0000 0110
Data Hi	02	0 2	0000 0010
Data Lo	2B	2 B	0010 1011
Data Hi	00	0 0	0000 0000
Data Lo	00	0 0	0000 0000
Data Hi	00	0 0	0000 0000
Data Lo	63	6 3	0110 0011
Error Check		LRC (2 chars.)	CRC (16 bits)
Trailer		CR LF	None
Total Bytes:		23	11

Figure 8 Slave Response with ASCII/RTU Framing

Modbus Function Formats (Continued)

Field Contents on Modbus Plus

Modbus messages sent on Modbus Plus networks are imbedded into the Logical Link Control (LLC) level frame. Modbus message fields consist of 8-bit bytes, similar to those used with RTU framing.

The Slave Address field is converted to a Modbus Plus routing path by the sending device. The CRC field is not sent in the Modbus message, because it would be redundant to the CRC check performed at the High-level Data Link Control (HDLC) level.

The rest of the message remains as in the standard serial format. The application software (e.g., MSTR blocks in controllers, or Modcom III in hosts) handles the framing of the message into a network packet.

Figure 9 shows how a Read Holding Registers query would be imbedded into a frame for Modbus Plus transmission.

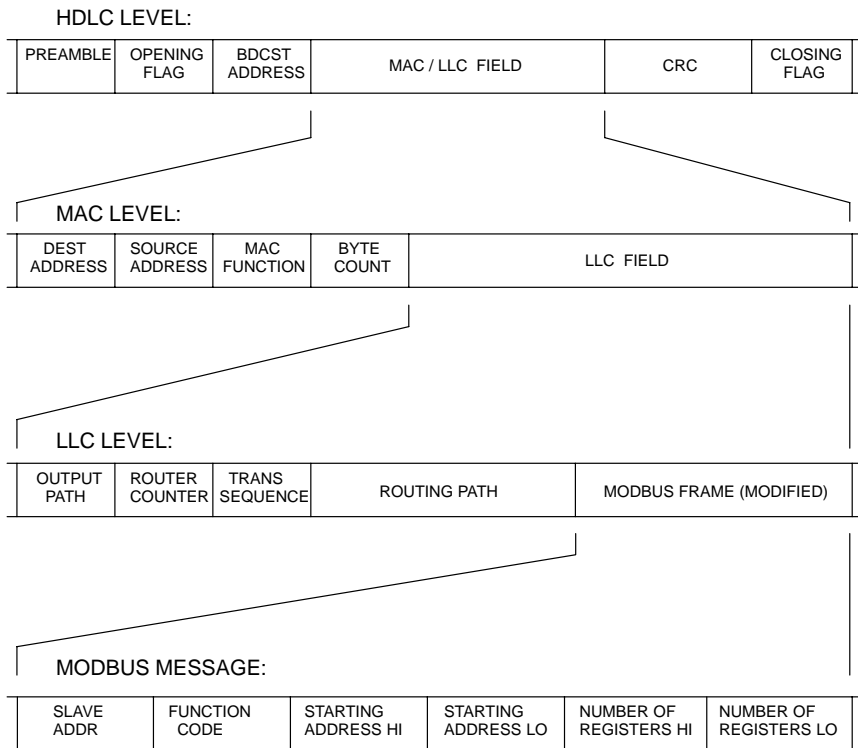


Figure 9 Field Contents on Modbus Plus

Function Codes Supported by Controllers

The listing below shows the function codes supported by Modicon controllers. Codes are listed in decimal.

'Y' indicates that the function is supported. 'N' indicates that it is not supported.

Code	Name	384	484	584	884	M84	984
01	Read Coil Status	Y	Y	Y	Y	Y	Y
02	Read Input Status	Y	Y	Y	Y	Y	Y
03	Read Holding Registers	Y	Y	Y	Y	Y	Y
04	Read Input Registers	Y	Y	Y	Y	Y	Y
05	Force Single Coil	Y	Y	Y	Y	Y	Y
06	Preset Single Register	Y	Y	Y	Y	Y	Y
07	Read Exception Status	Y	Y	Y	Y	Y	Y
08	Diagnostics (see Chapter 3)						
09	Program 484	N	Y	N	N	N	N
10	Poll 484	N	Y	N	N	N	N
11	Fetch Comm. Event Ctr.	Y	N	Y	N	N	Y
12	Fetch Comm. Event Log	Y	N	Y	N	N	Y
13	Program Controller	Y	N	Y	N	N	Y
14	Poll Controller	Y	N	Y	N	N	Y
15	Force Multiple Coils	Y	Y	Y	Y	Y	Y
16	Preset Multiple Registers	Y	Y	Y	Y	Y	Y
17	Report Slave ID	Y	Y	Y	Y	Y	Y
18	Program 884/M84	N	N	N	Y	Y	N
19	Reset Comm. Link	N	N	N	Y	Y	N
20	Read General Reference	N	N	Y	N	N	Y
21	Write General Reference	N	N	Y	N	N	Y

Code	Name	384	484	584	884	M84	984
22	Mask Write 4X Register	N	N	N	N	N	(1)
23	Read/Write 4X Registers	N	N	N	N	N	(1)
24	Read FIFO Queue	N	N	N	N	N	(1)

Notes:

(1) Function is supported in 984–785 only.

01 Read Coil Status

Description

Reads the ON/OFF status of discrete outputs (0X references, coils) in the slave. Broadcast is not supported.

Appendix B lists the maximum parameters supported by various controller models.

Query

The query message specifies the starting coil and quantity of coils to be read. Coils are addressed starting at zero: coils 1–16 are addressed as 0–15.

Here is an example of a request to read coils 20–56 from slave device 17:

QUERY	
Field Name	Example (Hex)
Slave Address	11
Function	01
Starting Address Hi	00
Starting Address Lo	13
No. of Points Hi	00
No. of Points Lo	25
Error Check (LRC or CRC)	—

Figure 10 Read Coil Status – Query

Response

The coil status in the response message is packed as one coil per bit of the data field. Status is indicated as: 1 = ON; 0 = OFF. The LSB of the first data byte contains the coil addressed in the query. The other coils follow toward the high order end of this byte, and from 'low order to high order' in subsequent bytes.

If the returned coil quantity is not a multiple of eight, the remaining bits in the final data byte will be padded with zeros (toward the high order end of the byte). The Byte Count field specifies the quantity of complete bytes of data.

Here is an example of a response to the query on the opposite page:

RESPONSE	
Field Name	Example (Hex)
Slave Address	11
Function	01
Byte Count	05
Data (Coils 27–20)	CD
Data (Coils 35–28)	6B
Data (Coils 43–36)	B2
Data (Coils 51–44)	0E
Data (Coils 56–52)	1B
Error Check (LRC or CRC)	—

Figure 11 Read Coil Status – Response

The status of coils 27–20 is shown as the byte value CD hex, or binary 1100 1101. Coil 27 is the MSB of this byte, and coil 20 is the LSB. Left to right, the status of coils 27 through 20 is: ON–ON–OFF–OFF–ON–ON–OFF–ON.

By convention, bits within a byte are shown with the MSB to the left, and the LSB to the right. Thus the coils in the first byte are '27 through 20', from left to right. The next byte has coils '35 through 28', left to right. As the bits are transmitted serially, they flow from LSB to MSB: 20 . . . 27, 28 . . . 35, and so on.

In the last data byte, the status of coils 56–52 is shown as the byte value 1B hex, or binary 0001 1011. Coil 56 is in the fourth bit position from the left, and coil 52 is the LSB of this byte. The status of coils 56 through 52 is: ON–ON–OFF–ON–ON. Note how the three remaining bits (toward the high order end) are zero-filled.

02 Read Input Status

Description

Reads the ON/OFF status of discrete inputs (1X references) in the slave.
Broadcast is not supported.

Appendix B lists the maximum parameters supported by various controller models.

Query

The query message specifies the starting input and quantity of inputs to be read.
Inputs are addressed starting at zero: inputs 1–16 are addressed as 0–15.

Here is an example of a request to read inputs 10197–10218 from slave device 17:

QUERY	
Field Name	Example (Hex)
Slave Address	11
Function	02
Starting Address Hi	00
Starting Address Lo	C4
No. of Points Hi	00
No. of Points Lo	16
Error Check (LRC or CRC)	—

Figure 12 Read Input Status – Query

Response

The input status in the response message is packed as one input per bit of the data field. Status is indicated as: 1 = ON; 0 = OFF. The LSB of the first data byte contains the input addressed in the query. The other inputs follow toward the high order end of this byte, and from 'low order to high order' in subsequent bytes.

If the returned input quantity is not a multiple of eight, the remaining bits in the final data byte will be padded with zeros (toward the high order end of the byte). The Byte Count field specifies the quantity of complete bytes of data.

Here is an example of a response to the query on the opposite page:

RESPONSE	
Field Name	Example (Hex)
Slave Address	11
Function	02
Byte Count	03
Data (Inputs 10204–10197)	AC
Data (Inputs 10212–10205)	DB
Data (Inputs 10218–10213)	35
Error Check (LRC or CRC)	—

Figure 13 Read Input Status – Response

The status of inputs 10204–10197 is shown as the byte value AC hex, or binary 1010 1100. Input 10204 is the MSB of this byte, and input 10197 is the LSB. Left to right, the status of inputs 10204 through 10197 is: ON–OFF–ON–OFF–ON–ON–OFF–OFF.

The status of inputs 10218–10213 is shown as the byte value 35 hex, or binary 0011 0101. Input 10218 is in the third bit position from the left, and input 10213 is the LSB. The status of inputs 10218 through 10213 is: ON–ON–OFF–ON–OFF–ON. Note how the two remaining bits (toward the high order end) are zero-filled.

03 Read Holding Registers

Description

Reads the binary contents of holding registers (4X references) in the slave. Broadcast is not supported.

Appendix B lists the maximum parameters supported by various controller models.

Query

The query message specifies the starting register and quantity of registers to be read. Registers are addressed starting at zero: registers 1–16 are addressed as 0–15.

Here is an example of a request to read registers 40108–40110 from slave device 17:

QUERY	
Field Name	Example (Hex)
Slave Address	11
Function	03
Starting Address Hi	00
Starting Address Lo	6B
No. of Points Hi	00
No. of Points Lo	03
Error Check (LRC or CRC)	—

Figure 14 Read Holding Registers – Query

Response

The register data in the response message are packed as two bytes per register, with the binary contents right justified within each byte. For each register, the first byte contains the high order bits and the second contains the low order bits.

Data is scanned in the slave at the rate of 125 registers per scan for 984–X8X controllers (984–685, etc), and at the rate of 32 registers per scan for all other controllers. The response is returned when the data is completely assembled.

Here is an example of a response to the query on the opposite page:

RESPONSE	
Field Name	Example (Hex)
Slave Address	11
Function	03
Byte Count	06
Data Hi (Register 40108)	02
Data Lo (Register 40108)	2B
Data Hi (Register 40109)	00
Data Lo (Register 40109)	00
Data Hi (Register 40110)	00
Data Lo (Register 40110)	64
Error Check (LRC or CRC)	—

Figure 15 Read Holding Registers – Response

The contents of register 40108 are shown as the two byte values of 02 2B hex, or 555 decimal. The contents of registers 40109–40110 are 00 00 and 00 64 hex, or 0 and 100 decimal.

04 Read Input Registers

Description

Reads the binary contents of input registers (3X references) in the slave. Broadcast is not supported.

Appendix B lists the maximum parameters supported by various controller models.

Query

The query message specifies the starting register and quantity of registers to be read. Registers are addressed starting at zero: registers 1–16 are addressed as 0–15.

Here is an example of a request to read register 30009 from slave device 17:

QUERY	
Field Name	Example (Hex)
Slave Address	11
Function	04
Starting Address Hi	00
Starting Address Lo	08
No. of Points Hi	00
No. of Points Lo	01
Error Check (LRC or CRC)	—

Figure 16 Read Input Registers – Query

Response

The register data in the response message are packed as two bytes per register, with the binary contents right justified within each byte. For each register, the first byte contains the high order bits and the second contains the low order bits.

Data is scanned in the slave at the rate of 125 registers per scan for 984–X8X controllers (984–685, etc), and at the rate of 32 registers per scan for all other controllers. The response is returned when the data is completely assembled.

Here is an example of a response to the query on the opposite page:

RESPONSE	
Field Name	Example (Hex)
Slave Address	11
Function	04
Byte Count	02
Data Hi (Register 30009)	00
Data Lo (Register 30009)	0A
Error Check (LRC or CRC)	—

Figure 17 Read Input Registers – Response

The contents of register 30009 are shown as the two byte values of 00 0A hex, or 10 decimal.

05 Force Single Coil

Description

Forces a single coil (0X reference) to either ON or OFF. When broadcast, the function forces the same coil reference in all attached slaves.



Note The function will override the controller's memory protect state and the coil's disable state. The forced state will remain valid until the controller's logic next solves the coil. The coil will remain forced if it is not programmed in the controller's logic.

Appendix B lists the maximum parameters supported by various controller models.

Query

The query message specifies the coil reference to be forced. Coils are addressed starting at zero: coil 1 is addressed as 0.

The requested ON/OFF state is specified by a constant in the query data field. A value of FF 00 hex requests the coil to be ON. A value of 00 00 requests it to be OFF. All other values are illegal and will not affect the coil.

Here is an example of a request to force coil 173 ON in slave device 17:

QUERY	
Field Name	Example (Hex)
Slave Address	11
Function	05
Coil Address Hi	00
Coil Address Lo	AC
Force Data Hi	FF
Force Data Lo	00
Error Check (LRC or CRC)	—

Figure 18 Force Single Coil – Query

Response

The normal response is an echo of the query, returned after the coil state has been forced.

Here is an example of a response to the query on the opposite page:

RESPONSE	
Field Name	Example (Hex)
Slave Address	11
Function	05
Coil Address Hi	00
Coil Address Lo	AC
Force Data Hi	FF
Force Data Lo	00
Error Check (LRC or CRC)	—

Figure 19 Force Single Coil – Response

06 Preset Single Register

Description

Presets a value into a single holding register (4X reference). When broadcast, the function presets the same register reference in all attached slaves.



Note The function will override the controller's memory protect state. The preset value will remain valid in the register until the controller's logic next solves the register contents. The register's value will remain if it is not programmed in the controller's logic.

Appendix B lists the maximum parameters supported by various controller models.

Query

The query message specifies the register reference to be preset. Registers are addressed starting at zero: register 1 is addressed as 0.

The requested preset value is specified in the query data field. M84 and 484 controllers use a 10-bit binary value, with the six high order bits set to zeros. All other controllers use 16-bit values.

Here is an example of a request to preset register 40002 to 00 03 hex in slave device 17:

QUERY	
Field Name	Example (Hex)
Slave Address	11
Function	06
Register Address Hi	00
Register Address Lo	01
Preset Data Hi	00
Preset Data Lo	03
Error Check (LRC or CRC)	—

Figure 20 Preset Single Register – Query

Response

The normal response is an echo of the query, returned after the register contents have been preset.

Here is an example of a response to the query on the opposite page:

RESPONSE	
Field Name	Example (Hex)
Slave Address	11
Function	06
Register Address Hi	00
Register Address Lo	01
Preset Data Hi	00
Preset Data Lo	03
Error Check (LRC or CRC)	—

Figure 21 Preset Single Register – Response

07 Read Exception Status

Description

Reads the contents of eight Exception Status coils within the slave controller. Certain coils have predefined assignments in the various controllers. Other coils can be programmed by the user to hold information about the controller's status, for example, 'machine ON/OFF', 'heads retracted', 'safeties satisfied', 'error conditions exist', or other user-defined flags. Broadcast is not supported.

The function provides a simple method for accessing this information, because the Exception Coil references are known (no coil reference is needed in the function). The predefined Exception Coil assignments are:

Controller Model	Coil	Assignment
M84, 184/384, 584, 984	1 – 8	User defined
484	257	Battery Status
	258 – 264	User defined
884	761	Battery Status
	762	Memory Protect Status
	763	RIO Health Status
	764–768	User defined

Query

Here is an example of a request to read the exception status in slave device 17:

Field Name	Example (Hex)
QUERY	
Slave Address	11
Function	07
Error Check (LRC or CRC)	—

Figure 22 Read Exception Status – Query

Response

The normal response contains the status of the eight Exception Status coils. The coils are packed into one data byte, with one bit per coil. The status of the lowest coil reference is contained in the least significant bit of the byte.

Here is an example of a response to the query on the opposite page:

RESPONSE	
Field Name	Example (Hex)
Slave Address	11
Function	07
Coil Data	6D
Error Check (LRC or CRC)	—

Figure 23 Read Exception Status – Response

In this example, the coil data is 6D hex (0110 1101 binary). Left to right, the coils are: OFF–ON–ON–OFF–ON–ON–OFF–ON. The status is shown from the highest to the lowest addressed coil.

If the controller is a 984, these bits are the status of coils 8 through 1.

If the controller is a 484, these bits are the status of coils 264 through 257. In this example, coil 257 is ON, indicating that the controller's batteries are OK.

11 (0B Hex) Fetch Comm Event Counter

Description

Returns a status word and an event count from the slave's communications event counter. By fetching the current count before and after a series of messages, a master can determine whether the messages were handled normally by the slave. Broadcast is not supported.

The controller's event counter is incremented once for each successful message completion. It is not incremented for exception responses, poll commands, or fetch event counter commands.

The event counter can be reset by means of the Diagnostics function (code 08), with a subfunction of Restart Communications Option (code 00 01) or Clear Counters and Diagnostic Register (code 00 0A).

Query

Here is an example of a request to fetch the communications event counter in slave device 17:

QUERY	
Field Name	Example (Hex)
Slave Address	11
Function	0B
Error Check (LRC or CRC)	—

Figure 24 Fetch Communications Event Counter – Query

Response

The normal response contains a two-byte status word, and a two-byte event count. The status word will be all ones (FF FF hex) if a previously-issued program command is still being processed by the slave (a busy condition exists). Otherwise, the status word will be all zeros.

Here is an example of a response to the query on the opposite page:

RESPONSE	
Field Name	Example (Hex)
Slave Address	11
Function	0B
Status HI	FF
Status Lo	FF
Event Count Hi	01
Event Count Lo	08
Error Check (LRC or CRC)	—

Figure 25 Fetch Communications Event Counter – Response

In this example, the status word is FF FF hex, indicating that a program function is still in progress in the slave. The event count shows that 264 (01 08 hex) events have been counted by the controller.

12 (0C Hex) Fetch Comm Event Log

Description

Returns a status word, event count, message count, and a field of event bytes from the slave. Broadcast is not supported.

The status word and event count are identical to that returned by the Fetch Communications Event Counter function (11, 0B hex).

The message counter contains the quantity of messages processed by the slave since its last restart, clear counters operation, or power-up. This count is identical to that returned by the Diagnostic function (code 08), subfunction Return Bus Message Count (code 11, 0B hex).

The event bytes field contains 0-64 bytes, with each byte corresponding to the status of one Modbus send or receive operation for the slave. The events are entered by the slave into the field in chronological order. Byte 0 is the most recent event. Each new byte flushes the oldest byte from the field.

Query

Here is an example of a request to fetch the communications event log in slave device 17:

QUERY	
Field Name	Example (Hex)
Slave Address	11
Function	0C
Error Check (LRC or CRC)	—

Figure 26 Fetch Communications Event Log – Query

Response

The normal response contains a two-byte status word field, a two-byte event count field, a two-byte message count field, and a field containing 0-64 bytes of events. A byte count field defines the total length of the data in these four fields.

Here is an example of a response to the query on the opposite page:

RESPONSE	
Field Name	Example (Hex)
Slave Address	11
Function	0C
Byte Count	08
Status HI	00
Status Lo	00
Event Count Hi	01
Event Count Lo	08
Message Count Hi	01
Message Count Lo	21
Event 0	20
Event 1	00
Error Check (LRC or CRC)	—

Figure 27 Fetch Communications Event Log – Response

In this example, the status word is 00 00 hex, indicating that the slave is not processing a program function. The event count shows that 264 (01 08 hex) events have been counted by the slave. The message count shows that 289 (01 21 hex) messages have been processed.

The most recent communications event is shown in the Event 0 byte. Its contents (20 hex) show that the slave has most recently entered the Listen Only Mode.

The previous event is shown in the Event 1 byte. Its contents (00 hex) show that the slave received a Communications Restart.

The layout of the response's event bytes is described on the next page.

12 (0C Hex) Fetch Comm Event Log (Continued)

What the Event Bytes Contain

An event byte returned by the Fetch Communications Event Log function can be any one of four types. The type is defined by bit 7 (the high-order bit) in each byte. It may be further defined by bit 6. This is explained below.

Slave Modbus Receive Event

This type of event byte is stored by the slave when a query message is received. It is stored before the slave processes the message. This event is defined by bit 7 set to a logic '1'. The other bits will be set to a logic '1' if the corresponding condition is TRUE. The bit layout is:

Bit	Contents
0	Not Used
1	Communications Error
2	Not Used
3	Not Used
4	Character Overrun
5	Currently in Listen Only Mode
6	Broadcast Received
7	1

Slave Modbus Send Event

This type of event byte is stored by the slave when it finishes processing a query message. It is stored if the slave returned a normal or exception response, or no response. This event is defined by bit 7 set to a logic '0', with bit 6 set to a '1'. The other bits will be set to a logic '1' if the corresponding condition is TRUE. The bit layout is:

Bit	Contents
0	Read Exception Sent (Exception Codes 1-3)
1	Slave Abort Exception Sent (Exception Code 4)
2	Slave Busy Exception Sent (Exception Codes 5-6)
3	Slave Program NAK Exception Sent (Exception Code 7)
4	Write Timeout Error Occurred
5	Currently in Listen Only Mode

6	1
7	0

Slave Entered Listen Only Mode

This type of event byte is stored by the slave when it enters the Listen Only Mode. The event is defined by a contents of 04 hex. The bit layout is:

Bit	Contents
0	0
1	0
2	1
3	0
4	0
5	0
6	0
7	0

Slave Initiated Communication Restart

This type of event byte is stored by the slave when its communications port. is restarted. The slave can be restarted by the Diagnostics function (code 08), with subfunction Restart Communications Option (code 00 01).

That function also places the slave into a 'Continue on Error' or 'Stop on Error' mode. If the slave is placed into 'Continue on Error' mode, the event byte is added to the existing event log. If the slave is placed into 'Stop on Error' mode, the byte is added to the log and the rest of the log is cleared to zeros.

The event is defined by a contents of zero. The bit layout is:

Bit	Contents
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0

15 (0F Hex) Force Multiple Coils

Description

Forces each coil (0X reference) in a sequence of coils to either ON or OFF. When broadcast, the function forces the same coil references in all attached slaves.



Note The function will override the controller's memory protect state and a coil's disable state. The forced state will remain valid until the controller's logic next solves each coil. Coils will remain forced if they are not programmed in the controller's logic.

Appendix B lists the maximum parameters supported by various controller models.

Query

The query message specifies the coil references to be forced. Coils are addressed starting at zero: coil 1 is addressed as 0.

The requested ON/OFF states are specified by contents of the query data field. A logical '1' in a bit position of the field requests the corresponding coil to be ON. A logical '0' requests it to be OFF.

The following page shows an example of a request to force a series of ten coils starting at coil 20 (addressed as 19, or 13 hex) in slave device 17.

The query data contents are two bytes: CD 01 hex (1100 1101 0000 0001 binary). The binary bits correspond to the coils in the following way:

Bit:	1	1	0	0	1	1	0	1	0	0	0	0	0	0	1
Coil:	27	26	25	24	23	22	21	20	-	-	-	-	-	-	29 28

The first byte transmitted (CD hex) addresses coils 27-20, with the least significant bit addressing the lowest coil (20) in this set.

The next byte transmitted (01 hex) addresses coils 29-28, with the least significant bit addressing the lowest coil (28) in this set. Unused bits in the last data byte should be zero-filled.

QUERY	
Field Name	Example (Hex)
Slave Address	11
Function	0F
Coil Address Hi	00
Coil Address Lo	13
Quantity of Coils Hi	00
Quantity of Coils Lo	0A
Byte Count	02
Force Data Hi (Coils 27-20)	CD
Force Data Lo (Coils 29-28)	01
Error Check (LRC or CRC)	—

Figure 28 Force Multiple Coils – Query

Response

The normal response returns the slave address, function code, starting address, and quantity of coils forced.

Here is an example of a response to the query shown above.

RESPONSE	
Field Name	Example (Hex)
Slave Address	11
Function	0F
Coil Address Hi	00
Coil Address Lo	13
Quantity of Coils Hi	00
Quantity of Coils Lo	0A
Error Check (LRC or CRC)	—

Figure 29 Force Multiple Coils – Response

16 (10 Hex) Preset Multiple Registers

Description

Presets values into a sequence of holding registers (4X references). When broadcast, the function presets the same register references in all attached slaves.



Note The function will override the controller's memory protect state. The preset values will remain valid in the registers until the controller's logic next solves the register contents. The register values will remain if they are not programmed in the controller's logic.

Appendix B lists the maximum parameters supported by various controller models.

Query

The query message specifies the register references to be preset. Registers are addressed starting at zero: register 1 is addressed as 0.

The requested preset values are specified in the query data field. M84 and 484 controllers use a 10-bit binary value, with the six high order bits set to zeros. All other controllers use 16-bit values. Data is packed as two bytes per register.

Here is an example of a request to preset two registers starting at 40002 to 00 0A and 01 02 hex, in slave device 17:

QUERY	
Field Name	Example (Hex)
Slave Address	11
Function	10
Starting Address Hi	00
Starting Address Lo	01
No. of Registers Hi	00
No. of Registers Lo	02
Byte Count	04
Data Hi	00
Data Lo	0A
Data Hi	01
Data Lo	02
Error Check (LRC or CRC)	—

Figure 30 Preset Multiple Registers – Query

Response

The normal response returns the slave address, function code, starting address, and quantity of registers preset.

Here is an example of a response to the query shown above.

RESPONSE	
Field Name	Example (Hex)
Slave Address	11
Function	10
Starting Address Hi	00
Starting Address Lo	01
No. of Registers Hi	00
No. of Registers Lo	02
Error Check (LRC or CRC)	—

Figure 31 Preset Multiple Registers – Response

17 (11 Hex) Report Slave ID

Description

Returns a description of the type of controller present at the slave address, the current status of the slave Run indicator, and other information specific to the slave device. Broadcast is not supported.

Query

Here is an example of a request to report the ID and status of slave device 17:

QUERY	
Field Name	Example (Hex)
Slave Address	11
Function	11
Error Check (LRC or CRC)	—

Figure 32 Report Slave ID – Query

Response

The format of a normal response is shown below. The data contents are specific to each type of controller. They are listed on the following pages.

RESPONSE	
Field Name	Contents
Slave Address	Echo of Slave Address
Function	11
Byte Count	Device Specific
Slave ID	Device Specific
Run Indicator Status	00 = OFF, FF = ON
Additional Data	Device Specific
...	
Error Check (LRC or CRC)	—

Figure 33 Report Slave ID – Response

A Summary of Slave IDs

These are the Slave ID codes returned by Modicon controllers in the first byte of the data field:

Slave ID	Controller
0	Micro 84
1	484
2	184/384
3	584
8	884
9	984

17 (11 Hex) Report Slave ID (Continued)

184/384

The 184 or 384 controller returns a byte count of either 4 or 74 (4A hexadecimal). If the controller's J347 Modbus Slave Interface is setup properly, and its internal PIB table is normal, the byte count will be 74. Otherwise the byte count will be 4.

The four bytes that are always returned are:

Byte	Contents
1	Slave ID (2 for 184/384). See bytes 3, 4 for further definition.
2	RUN indicator status (0 = OFF, FF = ON)
3, 4	Status word: Bit 0 = 0 Bit 1 = Memory Protect status (0 = OFF, 1 = ON) Bit 2, 3 = Controller type: Bit 2 = 0 and Bit 3 = 0 indicates 184 Bit 2 = 1 and Bit 3 = 0 indicates 384 Bits 4 - 15 = Unused

The additional 70 bytes returned for a correct J347 setup and normal PIB are:

Byte	Contents
5, 6	PIB table starting address
7, 8	Controller serial number
9, 10	Executive ID
	Bytes 11 - 74 contain the PIB table. This data is valid only if the controller is running (as shown in Byte 2). The table is as follows:
11, 12	Maximum quantity of output coils
13, 14	Output coil enable table
15, 16	Address of input coil/run table
17, 18	Quantity of input coils
19, 20	Input coil enable table
21, 22	First latch number (must be multiple of 16)
23, 24	Last latch number (must be multiple of 16)

50 Data and Control Functions

PI-MBUS-300

25, 26	Address of input registers
27, 28	Quantity of input registers
29, 30	Quantity of output and holding registers
31, 32	Address of user logic
33, 34	Address of output coil RAM table
35, 36	Function inhibit mask
37, 38	Address of extended function routine
39, 40	Address of data transfer routine
41, 42	Address of traffic cop
43, 44	Unused
45, 46	Function inhibit mask
47, 48	Address of 'A' Mode history table
49, 50	Request table for DX printer
51, 52	Quantity of sequence groups
53, 54	Address of sequence image table
55, 56	Address of sequence RAM
57, 58	Quantity of 50XX registers
59, 60	Address of 50XX table
61, 62	Address of output coil RAM image
63, 64	Address of input RAM image
65, 66	Delayed output start group
67, 68	Delayed output end group
69, 70	Watchdog line
71, 72	RAM Address of latches
73, 74	Quantity of delayed output groups

17 (11 Hex) Report Slave ID (Continued)

584

The 584 controller returns a byte count of 9, as follows:

Byte	Contents
1	Slave ID (3 for 584)
2	RUN indicator status (0 = OFF, FF = ON)
3	Quantity of 4K sections of page 0 memory
4	Quantity of 1K sections of state RAM
5	Quantity of segments of user logic
6, 7	Machine state word (configuration table word 101, 65 hex). The word is organized as follows:

Byte 6:

Bit 15 (MSB of byte 6) = Port 1 setup

Bit 14 = Port 2 setup

Bit 13 = Port 1 address set

Bit 12 = Port 2 address set

Bit 11 = Unassigned

Bit 10 = Constant Sweep status (0 = Constant Sweep OFF, 1 = ON)

Bit 9 = Single Sweep status (0 = Single Sweep OFF, 1 = ON)

Bit 8 = 16/24-bit nodes (0 = 24-bit nodes, 1 = 16-bit nodes)

Byte 7:

Bit 7 (MSB of byte 7) = Power ON (1 = ON, should never = 'OFF')

Bit 6 = RUN indicator status (0 = ON, 1 = OFF)

Bit 5 = Memory Protect status (0 = ON, 1 = OFF)

Bit 4 = Battery OK (0 = OK, 1 = Not OK)

Bits 3 - 0 = Unassigned

8, 9 Machine stop code (configuration table word 105, 69 hex).
The word is organized as follows:

Byte 8:

Bit 15 (MSB of byte 8) = Peripheral port stop (controlled stop)
Bit 14 = Unassigned
Bit 13 = Dim awareness
Bit 12 = Illegal peripheral intervention
Bit 11 = Multirate solve table invalid
Bit 10 = Start of Node did not start segment
Bit 9 = State RAM test failed
Bit 8 = No End of Logic detected, or bad quantity of segments

Byte 9:

Bit 7 (MSB of byte 9) = Watchdog timer expired
Bit 6 = Real time clock error
Bit 5 = CPU diagnostic failed
Bit 4 = Invalid traffic cop type
Bit 3 = Invalid node type
Bit 2 = Logic checksum error
Bit 1 = Backup checksum error
Bit 0 = Illegal configuration

17 (11 Hex) Report Slave ID (Continued)

984

The 984 controller returns a byte count of 9, as follows:

Byte	Contents
1	Slave ID (9 for 984)
2	RUN indicator status (0 = OFF, FF = ON)
3	Quantity of 4K sections of page 0 memory
4	Quantity of 1K sections of state RAM
5	Quantity of segments of user logic
6, 7	Machine state word (configuration table word 101, 65 hex). The word is organized as follows:

Byte 6:

Bit 15 (MSB of byte 6) = Unassigned

Bits 14 - 11 = Unassigned

Bit 10 = Constant Sweep status (0 = Constant Sweep OFF, 1 = ON)

Bit 9 = Single Sweep status (0 = Single Sweep OFF, 1 = ON)

Bit 8 = 16/24-bit nodes (0 = 24-bit nodes, 1 = 16-bit nodes)

Byte 7:

Bit 7 (MSB of byte 7) = Power ON (1 = ON, should never = 'OFF')

Bit 6 = RUN indicator status (0 = ON, 1 = OFF)

Bit 5 = Memory Protect status (0 = ON, 1 = OFF)

Bit 4 = Battery OK (0 = OK, 1 = Not OK)

Bits 3 - 1 = Unassigned

Bit 0 = Memory downsize flag (0 = NO, 1 = Downsize)

Memory Downsize: Bit 0 of the Machine State word defines the use of the memory downsize values in words 99, 100, and 175 (63, 64, and AF hexadecimal) of the configuration table. If bit 0 = logic 1, downsizing is calculated as follows:

$$\text{Page 0 size (16-bit words)} = (\text{Word 99} * 4096) - (\text{Word 175 low byte} * 16)$$

$$\text{State table size (16 bit words)} = (\text{Word 100} * 1024) - (\text{Word 175 high byte} * 16)$$

8, 9 Machine stop code (configuration table word 105, 69 hex).
The word is organized as follows:

Byte 8:

Bit 15 (MSB of byte 8) = Peripheral port stop (controlled stop)
Bit 14 (984A, B, X) = Extended memory parity failure
Bit 14 (Other 984) = Bad IO traffic cop
Bit 13 = Dim awareness
Bit 12 = Illegal peripheral intervention
Bit 11 = Bad segment scheduler table
Bit 10 = Start of Node did not start segment
Bit 9 = State RAM test failed
Bit 8 = No End of Logic detected, or bad quantity of segments

Byte 9:

Bit 7 (MSB of byte 9) = Watchdog timer expired
Bit 6 = Real time clock error
Bit 5 (984A, B, X) = CPU diagnostic failed
Bit 5 (Other 984) = Bad coil used table
Bit 4 = S908 remote IO head failure
Bit 3 = Invalid node type
Bit 2 = Logic checksum error
Bit 1 = Coil disabled while in RUN mode
Bit 0 = Illegal configuration

17 (11 Hex) Report Slave ID (Continued)

Micro 84

The Micro 84 controller returns a byte count of 8, as follows:

Byte	Contents
1	Slave ID (0 for Micro 84)
2	RUN indicator status (0 = OFF, FF = ON)
3	Current port number
4	Memory size (1 = 1K, 2 = 2K)
5	Unused (all zeros)

484

The 484 controller returns a byte count of 5, as follows:

Byte	Contents
1	Slave ID (1 for 484)
2	RUN indicator status (0 = OFF, FF = ON)
3	System state
4	First configuration byte
5	Second configuration byte

884

The 884 controller returns a byte count of 8, as follows:

Byte	Contents
1	Slave ID (8 for 884)
2	RUN indicator status (0 = OFF, FF = ON)
3	Current port number
4	Size of user logic plus state RAM, in kilobytes (1 word = 2 bytes)
5	Reserved
6	Hook bits: Bits 0 - 2 = Reserved Bit 3 = Mapper bypass: 1 = Do not execute standard mapper Bit 4 = End of Scan tests: 1 = Test end of scan hooks Bit 5 = Reserved Bit 6 = Logic Solver bypass: 1 = Do not execute standard logic solver Bit 7 = Reserved
7, 8	Reserved

20 (14Hex) Read General Reference

Description

Returns the contents of registers in Extended Memory file (6XXXXX) references. Broadcast is not supported.

The function can read multiple groups of references. The groups can be separate (non-contiguous), but the references within each group must be sequential.

Query

The query contains the standard Modbus slave address, function code, byte count, and error check fields. The rest of the query specifies the group or groups of references to be read. Each group is defined in a separate 'sub-request' field which contains 7 bytes:

- The reference type: 1 byte (must be specified as 6)
- The Extended Memory file number: 2 bytes (1 to 10, hex 0001 to 000A)
- The starting register address within the file: 2 bytes
- The quantity of registers to be read: 2 bytes.

The quantity of registers to be read, combined with all other fields in the expected response, must not exceed the allowable length of Modbus messages: 256 bytes.

The available quantity of Extended Memory files depends upon the installed size of Extended Memory in the slave controller. Each file except the last one contains 10,000 registers, addressed as 0000-270F hexadecimal (0000-9999 decimal).



Note The addressing of Extended Register (6XXXXX) references differs from that of Holding Register (4XXXX) references.

The lowest Extended Register is addressed as register 'zero' (60000).

The lowest Holding Register is addressed as register 'one' (40001).

For controllers other than the 984–785 with Extended Registers, the last (highest) register in the last file is:

Ext Mem Size	Last File	Last Register (Decimal)
16K	2	6383
32K	4	2767
64K	7	5535
96K	10	8303

For the 984–785 with Extended Registers, the last (highest) register in the last file is shown in the two tables below.

984–785 with AS–M785–032 Memory Cartridge:

User Logic	State RAM	Ext Mem Size	Last File	Last Register (Decimal)
32K	32K	0	0	0
16K	64K	72K	8	3727

984–785 with AS–M785–048 Memory Cartridge:

User Logic	State RAM	Ext Mem Size	Last File	Last Register (Decimal)
48K	32K	24K	3	4575
32K	64K	96K	10	8303

Examples of a query and response are provided starting on the next page.

20 (14 Hex) Read General Reference (Continued)

An example of a request to read two groups of references from slave device 17 is shown below.

Group 1 consists of two registers from file 4, starting at register 1 (address 0001).
Group 2 consists of two registers from file 3, starting at register 9 (address 0009).

QUERY Field Name	Example (Hex)
Slave Address	11
Function	14
Byte Count	0E
Sub-Req 1, Reference Type	06
Sub-Req 1, File Number Hi	00
Sub-Req 1, File Number Lo	04
Sub-Req 1, Starting Addr Hi	00
Sub-Req 1, Starting Addr Lo	01
Sub-Req 1, Register Count Hi	00
Sub-Req 1, Register Count Lo	02
Sub-Req 2, Reference Type	06
Sub-Req 2, File Number Hi	00
Sub-Req 2, File Number Lo	03
Sub-Req 2, Starting Addr Hi	00
Sub-Req 2, Starting Addr Lo	09
Sub-Req 2, Register Count Hi	00
Sub-Req 2, Register Count Lo	02
Error Check (LRC or CRC)	—

Figure 34 Read General Reference – Query

Response

The normal response is a series of 'sub-responses', one for each 'sub-request'. The byte count field is the total combined count of bytes in all 'sub-responses'. In addition, each 'sub-response' contains a field that shows its own byte count.

RESPONSE Field Name	Example (Hex)
Slave Address	11
Function	14
Byte Count	0C
Sub-Res 1, Byte Count	05
Sub-Res 1, Reference Type	06
Sub-Res 1, Register Data Hi	0D
Sub-Res 1, Register Data Lo	FE
Sub-Res 1, Register Data Hi	00
Sub-Res 1, Register Data Lo	20
Sub-Res 2, Byte Count	05
Sub-Res 2, Reference Type	06
Sub-Res 2, Register Data Hi	33
Sub-Res 2, Register Data Lo	CD
Sub-Res 2, Register Data Hi	00
Sub-Res 2, Register Data Lo	40
Error Check (LRC or CRC)	—

Figure 35 Read General Reference – Response

21 (15Hex) Write General Reference

Description

Writes the contents of registers in Extended Memory file (6XXXXX) references. Broadcast is not supported.

The function can write multiple groups of references. The groups can be separate (non-contiguous), but the references within each group must be sequential.

Query

The query contains the standard Modbus slave address, function code, byte count, and error check fields. The rest of the query specifies the group or groups of references to be written, and the data to be written into them. Each group is defined in a separate 'sub-request' field which contains 7 bytes plus the data:

- The reference type: 1 byte (must be specified as 6)
- The Extended Memory file number: 2 bytes (1 to 10, hex 0001 to 000A)
- The starting register address within the file: 2 bytes
- The quantity of registers to be written: 2 bytes
- The data to be written: 2 bytes per register.

The quantity of registers to be written, combined with all other fields in the query, must not exceed the allowable length of Modbus messages: 256 bytes.

The available quantity of Extended Memory files depends upon the installed size of Extended Memory in the slave controller. Each file except the last one contains 10,000 registers, addressed as 0000-270F hexadecimal (0000-9999 decimal).



Note The addressing of Extended Register (6XXXXX) references differs from that of Holding Register (4XXXX) references.

The lowest Extended Register is addressed as register 'zero' (600000).

The lowest Holding Register is addressed as register 'one' (40001).

For controllers other than the 984–785 with Extended Registers, the last (highest) register in the last file is:

Ext Mem Size	Last File	Last Register (Decimal)
16K	2	6383
32K	4	2767
64K	7	5535
96K	10	8303

For the 984–785 with Extended Registers, the last (highest) register in the last file is shown in the two tables below.

984–785 with AS–M785–032 Memory Cartridge:

User Logic	State RAM	Ext Mem Size	Last File	Last Register (Decimal)
32K	32K	0	0	0
16K	64K	72K	8	3727

984–785 with AS–M785–048 Memory Cartridge:

User Logic	State RAM	Ext Mem Size	Last File	Last Register (Decimal)
48K	32K	24K	3	4575
32K	64K	96K	10	8303

Examples of a query and response are provided starting on the next page.

21 (15 Hex) Write General Reference (Continued)

An example of a request to write one group of references into slave device 17 is shown below.

The group consists of three registers in file 4, starting at register 7 (address 0007).

QUERY	Example
Field Name	(Hex)
Slave Address	11
Function	15
Byte Count	0D
Sub-Req 1, Reference Type	06
Sub-Req 1, File Number Hi	00
Sub-Req 1, File Number Lo	04
Sub-Req 1, Starting Addr Hi	00
Sub-Req 1, Starting Addr Lo	07
Sub-Req 1, Register Count Hi	00
Sub-Req 1, Register Count Lo	03
Sub-Req 1, Register Data Hi	06
Sub-Req 1, Register Data Lo	AF
Sub-Req 1, Register Data Hi	04
Sub-Req 1, Register Data Lo	BE
Sub-Req 1, Register Data Hi	10
Sub-Req 1, Register Data Lo	0D
Error Check (LRC or CRC)	—

Figure 36 Write General Reference – Query

Response

The normal response is an echo of the query.

RESPONSE Field Name	Example (Hex)
Slave Address	11
Function	15
Byte Count	0D
Sub-Req 1, Reference Type	06
Sub-Req 1, File Number Hi	00
Sub-Req 1, File Number Lo	04
Sub-Req 1, Starting Addr Hi	00
Sub-Req 1, Starting Addr Lo	07
Sub-Req 1, Register Count Hi	00
Sub-Req 1, Register Count Lo	03
Sub-Req 1, Register Data Hi	06
Sub-Req 1, Register Data Lo	AF
Sub-Req 1, Register Data Hi	04
Sub-Req 1, Register Data Lo	BE
Sub-Req 1, Register Data Hi	10
Sub-Req 1, Register Data Lo	0D
Error Check (LRC or CRC)	—

Figure 37 Write General Reference – Response

22 (16Hex) Mask Write 4X Register

Description

Modifies the contents of a specified 4XXXX register using a combination of an AND mask, an OR mask, and the register's current contents. The function can be used to set or clear individual bits in the register. Broadcast is not supported.

This function is supported in the 984–785 controller only.

Query

The query specifies the 4XXXX reference to be written, the data to be used as the AND mask, and the data to be used as the OR mask.

The function's algorithm is:

$$\text{Result} = (\text{Current Contents AND And_Mask}) \text{ OR } (\text{Or_Mask AND } \overline{\text{And_Mask}})$$

For example:	Hex	Binary
Current Contents	= 12	0001 0010
And_Mask	= F2	1111 0010
Or_Mask	= 25	0010 0101
$\overline{\text{And_Mask}}$	= 0D	0000 1101
Result	= 17	0001 0111

Note that if the Or_Mask value is zero, the result is simply the logical ANDing of the current contents and And_Mask. If the And_Mask value is zero, the result is equal to the Or_Mask value.

Note that the contents of the register can be read with the Read Holding Registers function (function code 03). They could, however, be changed subsequently as the controller scans its user logic program.

An example of a Mask Write to register 5 in slave device 17, using the above mask values, is shown on the next page.

QUERY	
Field Name	Example (Hex)
Slave Address	11
Function	16
Reference Address Hi	00
Reference Address Lo	04
And_Mask Hi	00
And_Mask Lo	F2
Or_Mask Hi	00
Or-Mask Lo	25
Error Check (LRC or CRC)	—

Figure 38 Mask Write 4X Register – Query

Response

The normal response is an echo of the query. The response is returned after the register has been written.

RESPONSE	
Field Name	Example (Hex)
Slave Address	11
Function	16
Reference Address Hi	00
Reference Address Lo	04
And_Mask Hi	00
And_Mask Lo	F2
Or_Mask Hi	00
Or-Mask Lo	25
Error Check (LRC or CRC)	—

Figure 39 Mask Write 4X Register – Response

