Open Networks

TCP/IP ETHERNET

Ready for immediate operation with the worldwide standard TCP/IP protocol. A PC connected to the Ethernet has full access to all PLCs in the MELSECNET, all the way down to the I/Os on the production level.

CC-Link

The new open network for the control and I/O level. Sensors and actuators from different manufacturers can be connected. Up to 64 stations can be integrated.

PROFIBUS/DP

Enables quick and simple connection of sensors and actuators from different manufacturers to MELSEC PLCs, with data transfer rates of up to 12 Mbaud.

DeviceNet

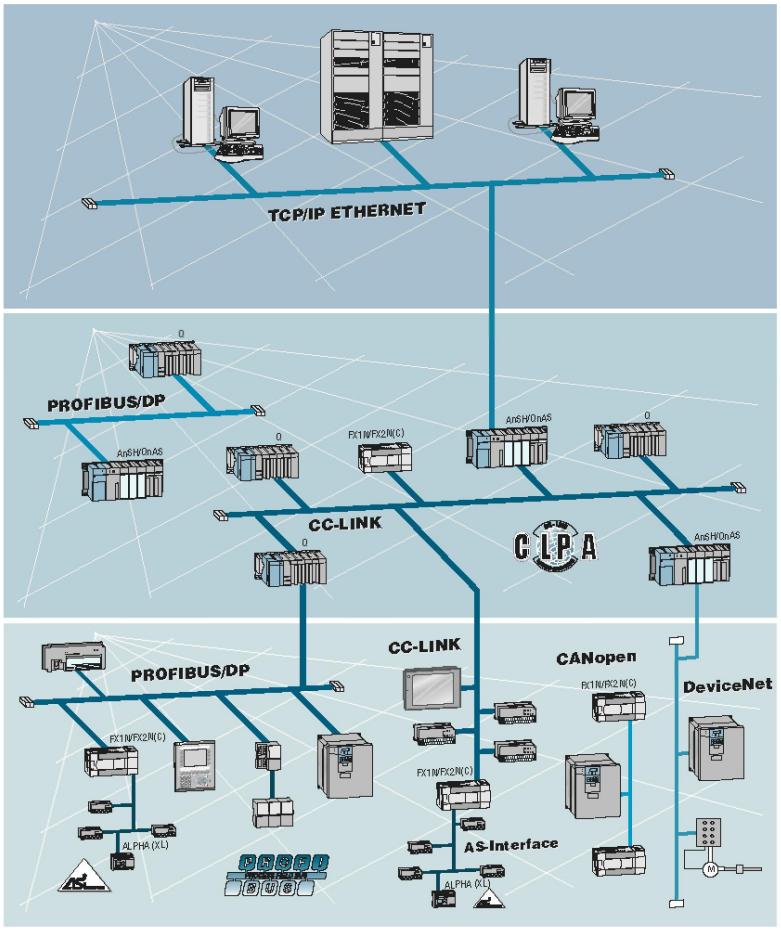
Cost-effective CAN-based network communications. Fault-resistant network structure where components of different manufacturers can be integrated quickly and easily.

AS-Interface

International standard for the lowest field bus level. Connection of conventional sensors and actuators with twisted-pair cable.

CANopen

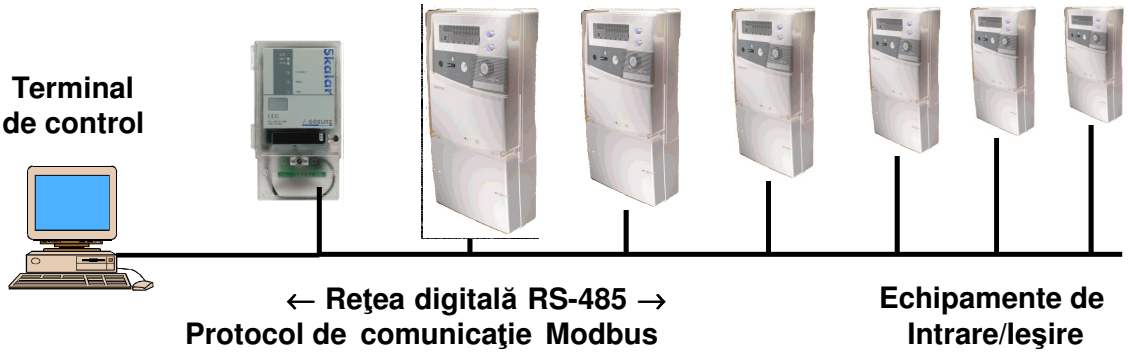
Cost effective network communications with fault-resistant network structure where components of different manufacturers can be integrated quickly and easily.



COMMAND LEVEL
TCP/IP ETHERNET

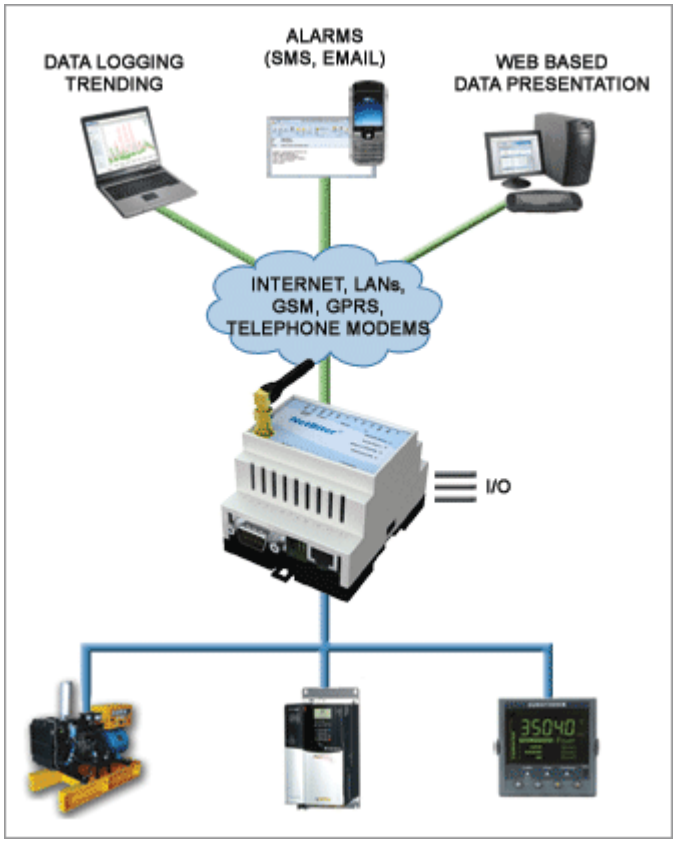
CONTROL LEVEL
PROFIBUS/DP
CC-Link

PRODUCTION LEVEL
PROFIBUS/DP
DeviceNet
AS-Interface
CC-Link
CANopen



Modbus Interface

- History of the Modbus interface
- Modbus message structure
- Modbus serial transmission modes
- Modbus addressing
- Modbus function codes



History of the Modbus protocol

Some communication standards just emerge. Not because they are pushed by a large group of vendors or a special standards organisation. These standards—like the *Modbus interface*—emerge because they are good, simple to implement and are therefore adapted by many manufacturers. Because of this, *Modbus* became the first widely accepted fieldbus standard.

Modbus has its roots in the late seventies of the previous century. It is 1979 when PLC manufacturer Modicon—now a brand of Schneider Electric's Telemecanique—published the Modbus communication interface for a multidrop network based on a master/client architecture. Communication between the Modbus nodes was achieved with messages. It was an open standard that described the messaging structure. The physical layer of the Modbus interface was free to choose. The original Modbus interface ran on RS-232, but most later Modbus implementations used RS-485 because it allowed longer distances, higher speeds and the possibility of a true multi-drop network. In a short time hunderds of vendors implemented the Modbus messaging system in their devices and Modbus became the de facto standard for industrial communication networks.

The nice thing of the Modbus standard is the flexibility, but at the same time the easy implementation of it. Not only intelligent devices like microcontrollers, PLCs etc. are able to communicate with Modbus, also many intelligent sensors are equiped with a Modbus interface to send their data to host systems. While Modbus was previously mainly used on wired serial communication lines, there are also extensions to the standard for wireless communications and TCP/IP networks.

Modbus message structure

The Modbus communication interface is built around messages. The format of these Modbus messages is independent of the type of physical interface used. On plain old **RS232** are the same messages used as on *Modbus/TCP* over ethernet. This gives the Modbus interface definition a very long lifetime. The same protocol can be used regardless of the connection type. Because of this, Modbus gives the possibility to easily upgrade the hardware structure of an industrial network, without the need for large changes in the software. A device can also communicate with several Modbus nodes at once, even if they are connected with different interface types, without the need to use a different protocol for every connection.

On simple interfaces like **RS485** or **RS232**, the Modbus messages are sent in plain form over the network. In this case the network is dedicated to Modbus. When using more versatile network systems like **TCP**/**IP** over ethernet, the Modbus messages are embedded in packets with the format necessary for the physical interface. In that case Modbus and other types of connections can co-exist at the same physical interface at the

same time. Although the main Modbus message structure is *peer-to-peer*, Modbus is able to function on both *point-to-point* and *multidrop* networks.

Each Modbus message has the same structure. Four basic elements are present in each message. The sequence of these elements is the same for all messages, to make it easy to parse the content of the Modbus message. A conversation is always started by a master in the Modbus network. A Modbus master sends a message and—depending of the contents of the message—a slave takes action and responds to it. There can be more masters in a Modbus network. Addressing in the message header is used to define which device should respond to a message. All other nodes on the Modbus network ignore the message if the address field doesn't match their own address.

Modbus message structure

Field	Description
Device address	Address of the receiver
Function code	Code defining message type
Data	Data block with additional information
Error check	Numeric check value to test for communication errors

Modbus serial transmission modes: Modbus/ASCII and Modbus/RTU

Serial Modbus connections can use two basic transmission modes, **ASCII** or **RTU**, *remote terminal unit*. The transmission mode in serial communications defines the way the Modbus messages are coded. With *Modbus/ASCII*, the messages are in a readable ASCII format. The *Modbus/RTU* format uses binary coding which makes the message unreadable when monitoring, but reduces the size of each message which allows for more data exchange in the same time span. All nodes on one Modbus network segment must use the same serial transmission mode. A device configured to use *Modbus/ASCII* cannot understand messages in *Modbus/RTU* and vice versa.

When using *Modbus/ASCII*, all messages are coded in hexadecimal values, represented with readable ASCII characters. Only the characters **0**...**9** and **A**...**F** are used for coding. For every byte of information, two communication-bytes are needed, because every communication-byte can only define 4 bits in the hexadecimal system. With *Modbus/RTU* the data is exchanged in a binary format, where each byte of information is coded in one communication-byte.

Modbus messages on serial connections are not sent in a plain format. They are framed to give receivers an easy way to detect the beginning and end of a message. When using *Modbus/ASCII*, characters are used to start and end a frame. The colon '**:**' is used to flag the start of a message and each message is ended with a CR/LF combination.

Modbus/RTU on the other hand uses time gaps of silence on the communication line for the framing. Each message must be preceded by a time gap with a minimum length of **3.5** characters. If a receiver detects a gap of at least **1.5** characters, it assumes that a new message is comming and the receive buffer is cleared. The main advantage of *Modbus/ASCII* is, that it allowes gaps between the bytes of a message with a maximum length of 1 second. With *Modbus/RTU* it is necessary to send each message as a continuous stream.

Properties of Modbus/ASCII and Modbus/RTU

	Modbus/ASCII		**Modbus/RTU**	
Characters	ASCII **0**...**9** and **A**..**F**		Binary **0**...**255**	
Error check	**LRC** Longitudinal Redundancy Check		**CRC** Cyclic Redundancy Check	
Frame start	character '**:**'		3.5 chars silence	
Frame end	characters **CR**/**LF**		3.5 chars silence	
Gaps in message	1 sec		1.5 times char length	
Start bit	**1**		**1**	
Data bits	**7**		**8**	
Parity	even/odd	none	even/odd	none
Stop bits	**1**	**2**	**1**	**2**

Modbus addressing

Slave Address

The first information in each Modbus message is the address of the receiver. This parameter contains one byte of information. In *Modbus/ASCII* it is coded with two hexadecimal characters, in *Modbus/RTU* one byte is used. Valid addresses are in the range **0**..**247**. The values **1**..**247** are assigned to individual Modbus devices and **0** is used as a broadcast address. Messages sent to the latter address will be accepted by all slaves, but no responses are folowed. A slave always responds to a Modbus message, exception are broadcast requests. When responding it uses the same address as the master put it in the request. In this way the master can see that the device is actually responding to the request.

Device I/O Address

Within a Modbus device, the holding registers, inputs and outputs are assigned a number between **1** and **10000**. One would expect, that the same addresses are used in the

Modbus messages to read or set values. Unfortunately this is not the case. In the Modbus messages addresses are used with a value between **0** and **9999**. If you want to read the value of output (coil) **18** for example, you have to specify the value **17** in the Modbus query message. More confusing is even, that for input and holding registers an offset must be substracted from the device address to get the proper address to put in the Modbus message structure. This leads to common mistakes and should be taken care of when designing applications with Modbus. The following table shows the address ranges for coils, inputs and holding registers and the way the address in the Modbus message is calculated given the actual address of the item in the slave device.

Device and Modbus address ranges

Device address (Absolute)	**Modbus address** (Relative)	**Description**
1...**10000***	address - 1	Coils (Digital outputs)
10001...**20000***	address - 10001	Buttons/Sensors (Dig. Inputs)
30001...4**0000***	address - 30001	Analog Input registers (16 Bits)
40001...**50000***	address - 40001	Holding registers (General 16 Bits)
____/ Offset	Relative address = Absolute address - Offset	

* **Maximum value is device dependent**

Modbus function codes

The second parameter in each Modbus message is the function code. This defines the message type and the type of action required by the slave. The parameter contains one byte of information. In *Modbus/ASCII* this is coded with two hexadecimal characters, in *Modbus/RTU* one byte is used. Valid function codes are in the range **1**..**255**. Not all Modbus devices recognize the same set of function codes. The most common codes are discussed here.

Normally, when a Modbus slave answers a response, it uses the same function code as in the request. However, when an error is detected, the highest bit of the function code is turned on. In that way the master can see the difference between success and failure responses.

Common Modbus function codes

Code	**Description**
01	Read coil status
02	Read input status
03	Read holding registers

04	Read input registers
05	Force single coil
06	Preset single register
07	Read exception status
08	Diagnostics
09	Program
10	Poll
11	Fetch Comm. Event Control
12	Fetch Comm. Event Log
13	Program Controller Y N
14	Poll Controller
15	Force multiple coils
16	Preset multiple registers
17	Report slave ID
18	Program
19	Reset Comm. Link
20	Read General Reference
21	Write General Reference
	Data and Control Functions
22	Mask Write 4X Register
23	Read/Write 4X Registers
24	Read FIFO Queue

Function 01: Read coil status

In Modbus language, a coil is a discrete output value. Modbus function **01** can be used to read the status of such an output. It is only possible to query one device at a time. Broadcast addressing is not supported with this Modbus function. The function can be used to request the status of various coils at once. This is done by defining an output range in the data field of the message.

Function 01 query structure

Byte	Value	Description
1	**1**...**247**	Slave device address
2	**1**	Function code

3	**0**...**255**	Starting address, high byte
4	**0**...**255**	Starting address, low byte
5	**0**...**255**	Number of coils, high byte
6	**0**...**255**	Number of coils, low byte
7(...8)	**LRC**/**CRC**	Error check value

When receiving a Modbus query message with function **01**, the slave collects the necessary output values and constructs an answer message. The length of this message is dependent on the number of values that have to be returned. In general, when **N** values are requested, a number of ((**N**+7) mod 8) bytes are necessary to store these values. The actual number of databytes in the datablock is put in the first byte of the data field. Therefore the general structure of an answer to a Modbus function **01** query is:

Function 01 answer structure

Byte	Value	Description
1	**1**...**247**	Slave device address
2	**1**	Function code
3	**0**...**255**	Number of data bytes **N**
4...**N**+3	**0**...**255**	Bit pattern of coil values
N+4(...**N**+5)	**LRC**/**CRC**	Error check value

Function 02: Read input status

Reading input values with Modbus is done in the same way as reading the status of coils. The only difference is that for inputs Modbus function **02** is used. Broadcast addressing mode is not supported. You can only query the value of inputs of one device at a time. Like with coils, the address of the first input, and the number of inputs to read must be put in the data field of the query message. Inputs on devices start numbering at **10001**. This address value is equivalent to address **0** in the Modbus message.

Function 02 query structure

Byte	Value	Description
1	**1**...**247**	Slave device address
2	**2**	Function code
3	**0**...**255**	Starting address, high byte
4	**0**...**255**	Starting address, low byte
5	**0**...**255**	Number of inputs, high byte

6	**0**...**255**	Number of inputs, low byte
7(...8)	**LRC**/**CRC**	Error check value

After receiving a query message with Modbus function **02**, the slave puts the requested input values in a message structure and sends this message back to the Modbus master. The length of the message depends on the number of input values returned. This causes the length of the output message to vary. The number of databytes in the data field that contain the input values is passed as the first byte in the data field. Each Modbus answering message has the following general structure.

Function 02 answer structure

Byte	Value	Description
1	**1**...**247**	Slave device address
2	**2**	Function code
3	**0**...**255**	Number of data bytes **N**
4...**N**+3	**0**...**255**	Bit pattern of input values
N+4(...**N**+5)	**LRC**/**CRC**	Error check value

Function 03: Read holding registers

Internal values in a Modbus device are stored in holding registers. These registers are two bytes wide and can be used for various purposes. Some registers contain configuration parameters where others are used to return measured values (temperatures etc.) to a host. Registers in a Modbus compatible device start counting at **40001**. They are addressed in the Modbus message structure with addresses starting at **0**. Modbus function **03** is used to request one or more holding register values from a device. Only one slave device can be addressed in a single query. Broadcast queries with function **03** are not supported.

Function 03 query structure

Byte	Value	Description
1	**1**...**247**	Slave device address
2	**3**	Function code
3	**0**...**255**	Starting address, high byte
4	**0**...**255**	Starting address, low byte
5	**0**...**255**	Number of registers, high byte
6	**0**...**255**	Number of registers, low byte
7(...8)	**LRC**/**CRC**	Error check value

After processing the query, the Modbus slave returns the 16 bit values of the requested holding registers. Because of the size of the holding registers, every register is coded with two bytes in the answering message. The first data byte contains the high byte, and the second the low byte of the register. The Modbus answer message starts with the slave device address and the function code **03**. The next byte is the number of data bytes that follow. This value is two times the number of registers returned. An error check is appended for the host to check if a communication error occured.

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:

Example	
Field Name	(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)	—

QUERY

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:

Example	
Field Name	(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)	—

RESPONSE

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:

Example Field Name	(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)	—

QUERY

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:

Example Field Name	(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)	—

RESPONSE

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:

Example Field Name	(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)	—

QUERY

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:

Example Field Name	(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)	—

RESPONSE

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:

Example Field Name	(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)	—

QUERY

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:

Example	
Field Name	(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 reguested 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:

Example	
Field Name	(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)	—

QUERY

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:

Example	
Field Name	(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)	—

RESPONSE

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 reguested 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:

Example Field Name	(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)	—

QUERY

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:

Example Field Name	(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)	—

RESPONSE

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 contoller'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:

Example

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

QUERY

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:

Example

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

RESPONSE

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:

Example

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

QUERY

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:

Example	
Field Name	(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)	—
RESPONSE	

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:

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

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:

Example	
Field Name	(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)	—
RESPONSE	

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 reguested 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 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.

Example Field Name	(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) —	

QUERY

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.

Example	
Field Name	(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)	—
RESPONSE	

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:

Example	
Field Name	(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)	—
QUERY	

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.

Example	
Field Name	(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)	—
RESPONSE	

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:

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

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.

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) —
RESPONSE

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 additonal 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)

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 = Constand 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 = Constand 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:
Page 0 size (16-bit words) = (Word 99 * 4096) – (Word 175 low byte * 16)
State table size (16 bit words) = (Word 100 * 1024) – (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' (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 State
Logic 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 State
Logic 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).

Example

Field Name	(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) —
QUERY
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.
Example
Field Name (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) —
RESPONSE
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 State

Logic 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 State

Logic 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).

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)	—

QUERY

Figure 36 Write General Reference – Query

Response

The normal response is an echo of the 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) —
RESPONSE
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:
Result = (Current Contents AND And_Mask) OR (Or_Mask AND And_Mask)
For example: **Hex Binary**
Current Contents = 12 0001 0010
And_Mask = F2 1111 0010
Or_Mask = 25 0010 0101
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.

Example Field Name	(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) —
QUERY
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.

Example Field Name	(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) —
RESPONSE
Figure 39 Mask Write 4X Register – Response

23 (17Hex) Read/Write 4X Registers

Description

Performs a combination of one read and one write operation in a single Modbus transaction. The function can write new contents to a group of 4XXXX registers, and then return the contents of another group of 4XXXX registers. Broadcast is not supported. This function is supported in the 984–785 controller only.

Query

The query specifies the starting address and quantity of registers of the group to be read. It also specifies the starting address, quantity of registers, and data for the group to be written. The byte count field specifies the quantity of bytes to follow in the write data field.

Here is an example of a query to read six registers starting at register 5, and to write three registers starting at register 16, in slave device 17:

Example Field Name	(Hex)
Slave Address	11
Function	17
Read Reference Address Hi	00
Read Reference Address Lo	04
Quantity to Read Hi	00
Quantity to Read Lo	06
Write Reference Address Hi	00
Write Reference Address Lo	0F
Quantity to Write Hi	00
Quantity to Write Lo	03
Byte Count	06
Write Data 1 Hi	00
Write Data 1 Lo	FF
Write Data 2 Hi	00
Write Data 2 Lo	FF
Write Data 3 Hi	00
Write Data 3 Lo	FF
Error Check (LRC or CRC)	—

QUERY

Figure 40 Read/Write 4X Registers – Query

Response

The normal response contains the data from the group of registers that were read. The byte count field specifies the quantity of bytes to follow in the read data field.

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

Example Field Name (Hex)	
Slave Address	11
Function	17
Byte Count	0C
Read Data 1 Hi	00
Read Data 1 Lo	FE
Read Data 2 Hi	0A
Read Data 2 Lo	CD
Read Data 3 Hi	00
Read Data 3 Lo	01
Read Data 4 Hi	00
Read Data 4 Lo	03
Read Data 5 Hi	00
Read Data 5 Lo	0D
Read Data 6 Hi	00
Read Data 6 Lo	FF
Error Check (LRC or CRC)	—

RESPONSE

Figure 41 Read/Write 4X Registers – Response

24 (18Hex) Read FIFO Queue

Description

Reads the contents of a First–In–First–Out (FIFO) queue of 4XXXX registers. The function returns a count of the registers in the queue, followed by the queued data. Up to 32 registers can be read: the count, plus up to 31 queued data registers. The queue count register is returned first, followed by the queued data registers. The function reads the queue contents, but does not clear them. Broadcast is not supported.

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

Query

The query specifies the starting 4XXXX reference to be read from the FIFO queue. This is the address of the pointer register used with the controller's FIN and FOUT function blocks. It contains the count of registers currently contained in the queue. The FIFO data registers follow this address sequentially.

An example of a Read FIFO Queue query to slave device 17 is shown below. The query is to read the queue starting at the pointer register 41247 (04DE hex).

Example
Field Name (Hex)
Slave Address 11
Function 18
FIFO Pointer Address Hi 04
FIFO Pointer Address Lo DE
Error Check (LRC or CRC) —
QUERY

Figure 42 Read FIFO Queue – Query

Response

In a normal response, the byte count shows the quantity of bytes to follow, including the queue count bytes and data register bytes (but not including the error check field).

The queue count is the quantity of data registers in the queue (not including the count register).

If the queue count exceeds 31, an exception response is returned with an error code of 03 (Illegal Data Value).

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

Example
Field Name (Hex)
Slave Address 11
Function 18
Byte Count Hi 00
Byte Count Lo 08
FIFO Count Hi 00
FIFO Count Lo 03
FIFO Data Reg 1 Hi 01
FIFO Data Reg 1 Lo B8
FIFO Data Reg 2 Hi 12
FIFO Data Reg 2 Lo 84
FIFO Data Reg 3 Hi 13
FIFO Data Reg 3 Lo 22
Error Check (LRC or CRC) —
RESPONSE

Figure 43 Read FIFO Queue – Response

In this example, the FIFO pointer register (41247 in the query) is returned with a queue count of 3. The three data registers follow the queue count. These are: 41248 (contents 440 decimal -- 01B8 hex); 41249 (contents 4740 -- 1284 hex); and 41250 (contents 4898 -- 1322 hex).

Chapter 3. Diagnostic Subfunctions

Modbus Function 08 – Diagnostics
Diagnostic Subfunctions

Function 08 – Diagnostics

Description

Modbus function 08 provides a series of tests for checking the communication system between the master and slave, or for checking various internal error conditions within the slave. Broadcast is not supported.

The function uses a two–byte **subfunction code** field in the query to define the type of test to be performed. The slave echoes both the function code and subfunction code in a normal response.

Most of the diagnostic queries use a two–byte data field to send diagnostic data or control information to the slave. Some of the diagnostics cause data to be returned from the slave in the data field of a normal response.

Diagnostic Effects on the Slave

In general, issuing a diagnostic function to a slave device does not affect the running of the user program in the slave. User logic, like discretes and registers, is not accessed by the diagnostics. Certain functions can optionally reset error counters in the slave.

A slave device can, however, be forced into 'Listen Only Mode' in which it will monitor the messages on the communications system but not respond to them. This can affect the outcome of your application program it it depends upon any further exchange of data with the slave device. Generally, the mode is forced to remove a malfunctioning slave device from the communications system.

How This Information is Organized in Your Guide

An example diagnostics query and response are shown on the opposite page. These show the location of the function code, subfunction code, and data field within the messages.

A list of subfunction codes supported by the controllers is shown on the pages after the example response. Each subfunction code is then listed with an example of the data field contents that would apply for that diagnostic.

Query

Here is an example of a request to slave device 17 to Return Query Data. This uses a subfunction code of zero (00 00 hex in the two–byte field). The data to be returned is sent in the two–byte data field (A5 37 hex).

Example
Field Name (Hex)
Slave Address 11
Function 08
Subfunction Hi 00
Subfunction Lo 00
Data Hi A5
Data Lo 37
Error Check (LRC or CRC) —
QUERY

Figure 44 Diagnostics – Query

Response

The normal response to the Return Query Data request is to loopback the same data. The function code and subfunction code are also echoed.

Example
Field Name (Hex)
Slave Address 11
Function 08
Subfunction Hi 00
Subfunction Lo 00
Data Hi A5
Data Lo 37
Error Check (LRC or CRC) —
RESPONSE

Figure 45 Diagnostics – Response

The data fields in responses to other kinds of queries could contain error counts or other information requested by the subfunction code.

DiagnosticCodes Supportedby Controllers

The listing below shows the subfunction codes supported by Modicon controllers. Codes are listed in decimal.
'Y' indicates that the subfunction is supported. 'N' indicates that it is not supported.

Code Name 384 484 584 884 M84 984
00 Return Query Data Y Y Y Y Y Y
01 Restart Comm Option Y Y Y Y Y Y
02 Return Diagnostic Register Y Y Y Y Y Y
03 Change ASCII Input Delimiter Y Y Y N N Y
04 Force Listen Only Mode Y Y Y Y Y Y
05–09 Reserved
10 Clear Ctrs and Diagnostic Reg. Y Y (1) N N (1)
11 Return Bus Message Count Y Y Y N N Y
12 Return Bus Comm. Error Count Y Y Y N N Y
13 Return Bus Exception Error Cnt Y Y Y N N Y
14 Return Slave Message Count Y Y Y N N N
15 Return Slave No Response Cnt Y Y Y N N N
16 Return Slave NAK Count Y Y Y N N Y
17 Return Slave Busy Count Y Y Y N N Y
18 Return Bus Char. Overrun Cnt Y Y Y N N Y
19 Return Overrun Error Count N N N Y N N
20 Clear Overrun Counter and Flag N N N Y N N
21 Get/Clear Modbus Plus Statistics N N N N N Y
22–up Reserved

Notes:
(1) Clears Counters only.

Diagnostic Subfunctions

00 Return Query Data

The data passed in the query data field is to be returned (looped back) in the response. The entire response message should be identical to the query.

Subfunction Data Field (Query) Data Field (Response)
00 00 Any Echo Query Data

01 Restart Communications Option

The slave's peripheral port is to be initialized and restarted, and all of its communications event counters are to be cleared. If the port is currently in Listen Only Mode, no response is returned. This function is the only one that brings the port out of Listen Only Mode. If the port is not currently in Listen Only Mode, a normal response is returned. This occurs before the restart is executed.
When the slave receives the query, it attempts a restart and executes its power–up confidence tests. Successful completion of the tests will bring the port online.
A query data field contents of FF 00 hex causes the port's Communications Event Log to be cleared also. Contents of 00 00 leave the log as it was prior to the restart.

Subfunction Data Field (Query) Data Field (Response)
00 01 00 00 Echo Query Data
00 01 FF 00 Echo Query Data

08 Diagnostics (Continued)

02 Return Diagnostic Register

The contents of the slave's 16–bit diagnostic register are returned in the response.

Subfunction Data Field (Query) Data Field (Response)
00 02 00 00 Diagnostic Register Contents

How the Register Data is Organized

The assignment of diagnostic register bits for Modicon controllers is listed below. In each register, bit 15 is the high–order bit. The description is TRUE when the corresponding bit is set to a logic '1'.

184/384 Diagnostic Register

Bit Description
0 Continue on Error
1 Run Light Failed
2 T–Bus Test Failed
3 Asynchronous Bus Test Failed
4 Force Listen Only Mode
5 Not Used
6 Not Used
7 ROM Chip 0 Test Failed
8 Continuous ROM Checksum Test in Execution
9 ROM Chip 1 Test Failed
10 ROM Chip 2 Test Failed
11 ROM Chip 3 Test Failed
12 RAM Chip 5000-53FF Test Failed
13 RAM Chip 6000-67FF Test Failed, Even Addresses
14 RAM Chip 6000-67FF Test Failed, Odd Addresses
15 Timer Chip Test Failed

 Diagnostic Subfunctions 79

484 Diagnostic Register

Bit Description
0 Continue on Error
1 CPU Test or Run Light Failed
2 Parallel Port Test Failed
3 Asynchronous Bus Test Failed
4 Timer 0 Test Failed
5 Timer 1 Test Failed
6 Timer 2 Test Failed
7 ROM Chip 0000-07FF Test Failed
8 Continuous ROM Checksum Test in Execution
9 ROM Chip 0800-0FFF Test Failed
10 ROM Chip 1000-17FF Test Failed
11 ROM Chip 1800-1FFF Test Failed
12 RAM Chip 4000-40FF Test Failed
13 RAM Chip 4100-41FF Test Failed
14 RAM Chip 4200-42FF Test Failed
15 RAM Chip 4300-43FF Test Failed

584/984 Diagnostic Register

Bit Description
0 Illegal Configuration
1 Backup Checksum Error in High–Speed RAM
2 Logic Checksum Error
3 Invalid Node Type
4 Invalid Traffic Cop Type
5 CPU/Solve Diagnostic Failed
6 Real Time Clock Failed
7 Watchdog Timer Failed - Scan Time exceeded 250 ms.
8 No End of Logic Node detected, or quantity of end of segment words (DOIO) does not match quantity of segments configured
9 State Ram Test Failed
10 Start of Network (SON) did not begin network
11 Bad Order of Solve Table
12 Illegal Peripheral Intervention
13 Dim Awareness Flag
14 Not Used
15 Peripheral Port Stop Executed, not an error.

80 Diagnostic Subfunctions

08 Diagnostics (Continued)

884 Diagnostic Register

Bit Description
0 Modbus IOP Overrun Errors Flag
1 Modbus Option Overrun Errors Flag

2 Modbus IOP Failed
3 Modlbus Option Failed
4 Ourbus IOP Failed
5 Remote IO Failed
6 Main CPU Failed
7 Table RAM Checksum Failed
8 Scan Task exceeded its time limit - too much user logic
9 Not Used
10 Not Used
11 Not Used
12 Not Used
13 Not Used
14 Not Used
15 Not Used

03 Change ASCII Input Delimiter

The character 'CHAR' passed in the query data field becomes the end of message delimiter for future messages (replacing the default LF character). This function is useful in cases where a Line Feed is not wanted at the end of ASCII messages.

Subfunction Data Field (Query) Data Field (Response)

00 03 CHAR 00 Echo Query Data

04 Force Listen Only Mode

Forces the addressed slave to its Listen Only Mode for Modbus communications. This isolates it from the other devices on the network, allowing them to continue communicating without interruption from the addressed slave. No response is returned.

When the slave enters its Listen Only Mode, all active communication controls are turned off. The Ready watchdog timer is allowed to expire, locking the controls off. While in this mode, any Modbus messages addressed to the slave or broadcast are monitored, but no actions will be taken and no responses will be sent.

The only function that will be processed after the mode is entered will be the Restart Communications Option function (function code 8, subfunction 1).

Subfunction Data Field (Query) Data Field (Response)

00 04 00 00 No Response Returned

10 (0A Hex) Clear Counters and Diagnostic Register

For controllers other than the 584 or 984, clears all counters and the diagnostic register. For the 584 or 984, clears the counters only. Counters are also cleared upon power–up.

Subfunction Data Field (Query) Data Field (Response)

00 0A 00 00 Echo Query Data

08 Diagnostics (Continued)

11 (0B Hex) Return Bus Message Count

The response data field returns the quantity of messages that the slave has detected on the communications system since its last restart, clear counters operation, or power–up.

Subfunction Data Field (Query) Data Field (Response)

00 0B 00 00 Total Message Count

12 (0C Hex) Return Bus Communication Error Count

The response data field returns the quantity of CRC errors encountered by the slave since its last restart, clear counters operation, or power–up.

Subfunction Data Field (Query) Data Field (Response)

00 0C 00 00 CRC Error Count

13 (0D Hex) Return Bus Exception Error Count

The response data field returns the quantity of Modbus exception responses returned by the slave since its last restart, clear counters operation, or power–up. Exception responses are described and listed in Appendix A.

Subfunction Data Field (Query) Data Field (Response)

00 0D 00 00 Exception Error Count

14 (0E Hex) Return Slave Message Count

The response data field returns the quantity of messages addressed to the slave, or broadcast, that the slave has processed since its last restart, clear counters operation, or power–up.

Subfunction Data Field (Query) Data Field (Response)

00 0E 00 00 Slave Message Count

15 (0F Hex) Return Slave No Response Count

The response data field returns the quantity of messages addressed to the slave for which it returned no response (neither a normal response nor an exception response), since its last restart, clear counters operation, or power–up.

Subfunction Data Field (Query) Data Field (Response)

00 0F 00 00 Slave No Response Count

16 (10 Hex) Return Slave NAK Count

The response data field returns the quantity of messages addressed to the slave for which it returned a Negative Acknowledge (NAK) exception response, since its last restart, clear counters operation, or power–up. Exception responses are described and listed in Appendix A.

Subfunction Data Field (Query) Data Field (Response)

00 10 00 00 Slave NAK Count

08 Diagnostics (Continued)

17 (11 Hex) Return Slave Busy Count

The response data field returns the quantity of messages addressed to the slave for which it returned a Slave Device Busy exception response, since its last restart, clear counters operation, or power–up. Exception responses are described and listed in Appendix A.

Subfunction Data Field (Query) Data Field (Response)

00 11 00 00 Slave Device Busy Count

18 (12 Hex) Return Bus Character Overrun Count

The response data field returns the quantity of messages addressed to the slave that it could not handle due to a character overrun condition, since its last restart, clear counters operation, or power–up. A character overrun is caused by data characters arriving at the port faster than they can be stored, or by the loss of a character due to a hardware malfunction.

Subfunction Data Field (Query) Data Field (Response)

00 12 00 00 Slave Character Overrun Count

19 (13 Hex) Return IOP Overrun Count (884)

The response data field returns the quantity of messages addressed to the slave that it could not handle due to an 884 IOP overrun condition, since its last restart, clear counters operation, or power–up. An IOP overrun is caused by data characters arriving at the port faster than they can be stored, or by the loss of a character due to a hardware malfunction. This function is specific to the 884.

Subfunction Data Field (Query) Data Field (Response)

00 13 00 00 Slave IOP Overrun Count

20 (14 Hex) Clear Overrun Counter and Flag (884)

Clears the 884 overrun error counter and resets the error flag. The current state of the flag is found in bit 0 of the 884 diagnostic register (see subfunction 02). This function is specific to the 884.

Subfunction Data Field (Query) Data Field (Response)

00 14 00 00 Echo Query Data

08 Diagnostics (Continued)

21 (15 Hex) Get/Clear Modbus Plus Statistics

Returns a series of 54 16-bit words (108 bytes) in the data field of the response (this function differs from the usual two-byte length of the data field). The data contains the statistics for the Modbus Plus peer processor in the slave device. In addition to the Function code (08) and Subfunction code (00 15 hex) in the query, a two-byte Operation field is used to specify either a ‘Get Statistics’ or a

‘Clear Statistics’ operation. The two operations are exclusive - the ‘Get’ operation cannot clear the statistics, and the ‘Clear’ operation does not return statistics prior to clearing them. Statistics are also cleared on power-up of the slave device.
The operation field immediately follows the subfunction field in the query:
— A value of 00 03 specifies the ‘Get Statistics’ operation.
— A value of 00 04 specifies the ‘Clear Statistics’ operation.
QUERY: This is the field sequence in the query:
Function Subfunction Operation
08 00 15 00 03 (Get Statistics)
08 00 15 00 04 (Clear Statistics)
GET STATISTICS RESPONSE: This is the field sequence in the normal response to a Get Statistics query:
Function Subfunction Operation Byte Count Data
08 00 15 00 03 00 6C Words 00 - 53
CLEAR STATISTICS RESPONSE: The normal response to a Clear Statistics query is an echo of the query:
Function Subfunction Operation
08 00 15 00 04

Modbus Plus Network Statistics

Word Bits Meaning
00 Node type ID:
0 Unknown node type
1 Programmable controller node
2 Modbus bridge node
3 Host computer node
4 Bridge Plus node
5 Peer I/O node
01 0 ... 11 Software version number in hex (to read, strip bits 12–15 from word)
12 ... 14 Reserved
15 Defines Word 15 error counters (see Word 15)
Most significant bit defines use of error counters in Word 15.
Least significant half of upper byte, plus lower byte, contain software version.
Layout: | Upper Byte | Lower Byte |
[ ] [——Software version in hex——]
\
Most significant bit defines Word 15 error counters
(see Word 15)
02 Network address for this station
03 MAC state variable:
0 Power up state
1 Monitor offline state
2 Duplicate offline state
3 Idle state
4 Use token state
5 Work response state
6 Pass token state
7 Solicit response state
8 Check pass state
9 Claim token state
10 Claim response state
04 Peer status (LED code); provides status of this unit
relative to the network:
0 Monitor link operation
32 Normal link operation
64 Never getting token
96 Sole station
128 Duplicate station

08 Diagnostics (Continued)

Modbus Plus Network Statistics (Continued)

Word Bits Meaning
05 Token pass counter; increments each time this station gets the token
06 Token rotation time in ms
07 LO Data master failed during token ownership bit map

HI Program master failed during token ownership bit map
08 LO Data master token owner work bit map
HI Program master token owner work bit map
09 LO Data slave token owner work bit map
HI Program slave token owner work bit map
10 HI Data slave/get slave command transfer request bit map
11 LO Program master/get master rsp transfer request bit map
HI Program slave/get slave command transfer request bit map
12 LO Program master connect status bit map
HI Program slave automatic logout request bit map
13 LO Pretransmit deferral error counter
HI Receive buffer DMA overrun error counter
14 LO Repeated command received counter
HI Frame size error counter
15 If Word 1 bit 15 is not set, Word 15 has the following meaning:
LO Receiver collision–abort error counter
HI Receiver alignment error counter
If Word 1 bit 15 is set, Word 15 has the following meaning:
LO Cable A framing error
HI Cable B framing error
16 LO Receiver CRC error counter
HI Bad packet–length error counter
17 LO Bad link–address error counter
HI Transmit buffer DMA–underrun error counter

Word Byte Meaning

18 LO Bad internal packet length error counter
HI Bad MAC function code error counter
19 LO Communication retry counter
HI Communication failed error counter
20 LO Good receive packet success counter
HI No response received error counter
21 LO Exception response received error counter
HI Unexpected path error counter
22 LO Unexpected response error counter
HI Forgotten transaction error counter
23 LO Active station table bit map, nodes 1 ... 8
HI Active station table bit map, nodes 9 ...16
24 LO Active station table bit map, nodes 17 ... 24
HI Active station table bit map, nodes 25 ... 32
25 LO Active station table bit map, nodes 33 ... 40
HI Active station table bit map, nodes 41 ... 48
26 LO Active station table bit map, nodes 49 ... 56
HI Active station table bit map, nodes 57 ... 64
27 LO Token station table bit map, nodes 1 ... 8
HI Token station table bit map, nodes 9 ... 16
28 LO Token station table bit map, nodes 17 ... 24
HI Token station table bit map, nodes 25 ... 32
29 LO Token station table bit map, nodes 33 ... 40
HI Token station table bit map, nodes 41 ... 48
30 LO Token station table bit map, nodes 49 ... 56
HI Token station table bit map, nodes 57 ... 64
31 LO Global data present table bit map, nodes 1 ... 8
HI Global data present table bit map, nodes 9 ... 16
32 LO Global data present table bit map, nodes 17 ... 24
HI Global data present table bit map, nodes 25 ... 32
33 LO Global data present table bit map, nodes 33 ... 40
HI Global data present table bit map, nodes 41 ... 48
34 LO Global data present table map, nodes 49 ... 56
HI Global data present table bit map, nodes 57 ... 64

08 Diagnostics (Continued)

Modbus Plus Network Statistics (Continued)

Word Bits Meaning

35 LO Receive buffer in use bit map, buffer 1–8
HI Receive buffer in use bit map, buffer 9 ... 16
36 LO Receive buffer in use bit map, buffer 17 ... 24
HI Receive buffer in use bit map, buffer 25 ... 32

37 LO Receive buffer in use bit map, buffer 33 ... 40
HI Station management command processed initiation counter
38 LO Data master output path 1 command initiation counter
HI Data master output path 2 command initiation counter
39 LO Data master output path 3 command initiation counter
HI Data master output path 4 command initiation counter
40 LO Data master output path 5 command initiation counter
HI Data master output path 6 command initiation counter
41 LO Data master output path 7 command initiation counter
HI Data master output path 8 command initiation counter
42 LO Data slave input path 41 command processed counter
HI Data slave input path 42 command processed counter
43 LO Data slave input path 43 command processed counter
HI Data slave input path 44 command processed counter
44 LO Data slave input path 45 command processed counter
HI Data slave input path 46 command processed counter
45 LO Data slave input path 47 command processed counter
HI Data slave input path 48 command processed counter
46 LO Program master output path 81 command initiation counter
HI Program master output path 82 command initiation counter
47 LO Program master output path 83 command initiation counter
HI Program master output path 84 command initiation counter
48 LO Program master command initiation counter
HI Program master output path 86 command initiation counter
49 LO Program master output path 87 command initiation counter
HI Program master output path 88 command initiation counter

Word Bits Meaning

50 LO Program slave input path C1 command processed counter
HI Program slave input path C2 command processed counter
51 LO Program slave input path C3 command processed counter
HI Program slave input path C4 command processed counter
52 LO Program slave input path C5 command processed counter
HI Program slave input path C6 command processed counter
53 LO Program slave input path C7 command processed counter
HI Program slave input path C8 command processed counter

Literature

- The Modbus technical resources on the Modbus-IDA site are a good starting point for those who need the latest information about implementing and using the Modbus interface. Modbus-IDA is the current driving force behind the promotion and implementation of the Modbus protocol. *Modbus-IDA*