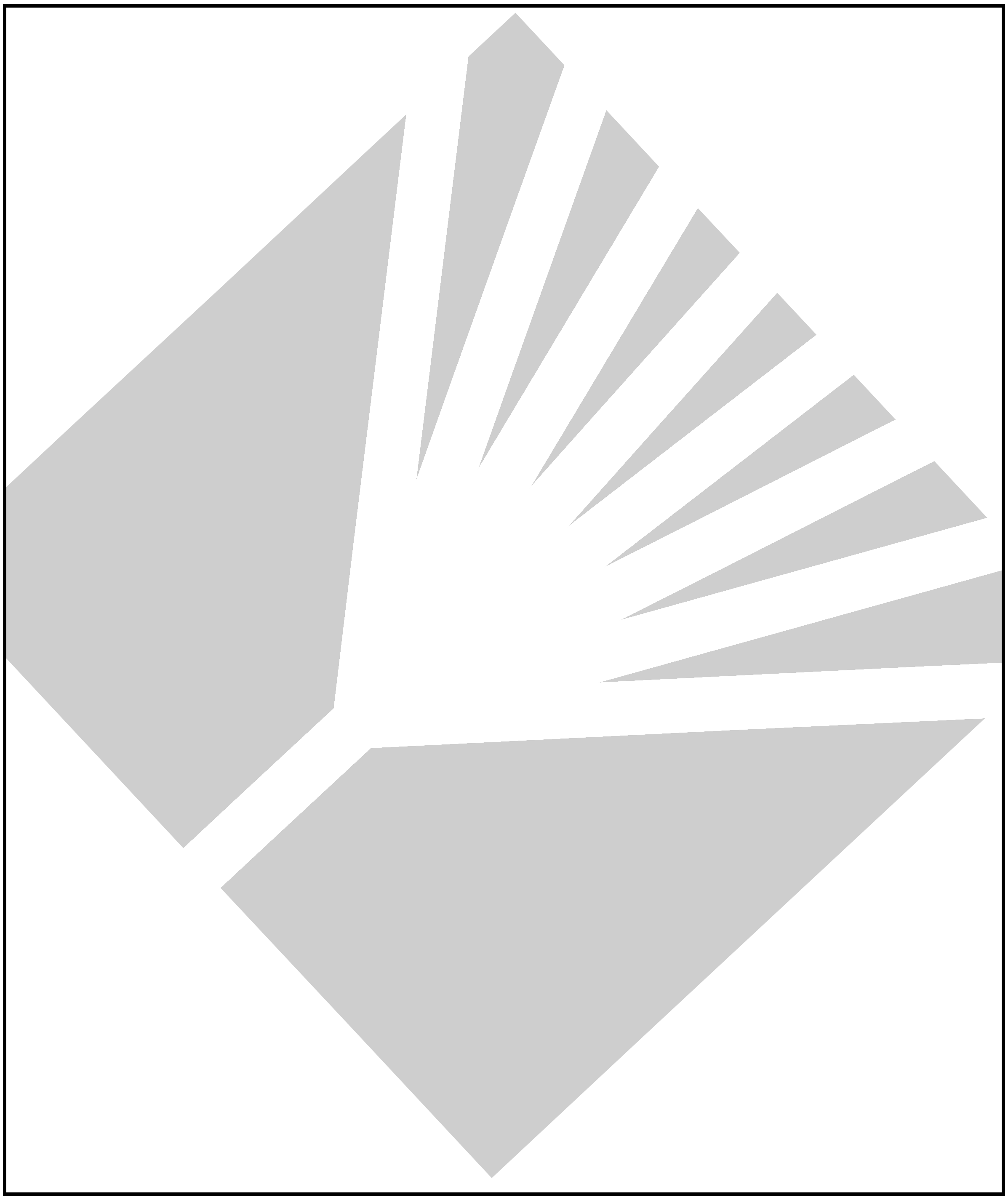
MTL8000

Process Control for Process I/O



Modbus Communications Manual

Application Note

Contents

AN8002 1 April 1999

Modbus Communications

Introduction

This manual provides a background to the Modbus protocol and, broadly, how it is implemented for the MTL8000 Series[1] of bus interface equipment for process I/O. It is only one of a range of protocols available to interface control systems to the above equipment but it has substantial history in the industry and is a protocol that control engineers have become familiar with and is one into which they can comfortably delve. This manual is designed to assist engineers in their understanding of the protocol and possible development of custom interfaces for controlling the MTL8000 equipment.

The manual is organised as follows:

- A discussion of how Modbus originated and its basic structure
- A review of the main Modbus Control Functions and the data involved
- A look at Exception Responses - when instructions from the master can not be executed
- Implementing the physical layer of Modbus

Related publications

Information specific to the Modbus interfaces in the MTL8000 Series is available in the following MTL publications:

INM8505	Instruction Manual, MTL8505 - Modbus BIM
INM8800BUS	BUS8800-MB Interface Manual

The MTL8000 Series Bus Interface Module (BIM) has a dedicated software package to enable the user to configure it and to define parameters such as network address, etc. This software is explained in the following MTL publication:

INM8450	Instruction Manual, BIM Configuration Software

[1] Throughout this manual, references to MTL8000 Series automatically includes the MTL8800L Series equipment.

What is Modbus?

Modbus is a communication protocol developed by AEG-Modicon, and was devised initially for use with their own Programmable Logic Controllers. It has, subsequently, become widely accepted as a communications standard, and many products have now been developed which use this protocol.

The protocol specification is maintained by its originators, AEG-Modicon, independently of any professional body or industry association. Consequently, there is no formal process by which a product may be certified to be 'Modbus compatible'. The onus is on the manufacturer of the product to confirm that their products are, and continue to remain, compatible with other Modbus devices.

The protocol defines a message structure and format, and determines how a slave will recognise messages sent to it by its master, and how it should decode the information contained in the message. Standardisation of these elements has meant that Modbus devices from a number of different manufacturers can be interconnected, without the need for specialised software drivers for each component.

Note: Modicon have also developed a new protocol called Modbus Plus. This has remained proprietary and is not widely used; consequently, Modbus Plus is *not* supported by the MTL8000 bus interfaces.

JBUS is a very similar protocol to Modbus and, insofar as the commands provided for the Modbus slave coincide with those of JBUS, the JBUS protocol will be supported. However, where there are minor differences in interpretation of the diagnostics sub-functions the Modbus version will take precedence.

Modbus Transactions

Modbus controllers communicate using a master-slave technique, in which only one device (the master) can initiate a communication sequence.

Note: Throughout this manual the term slave can normally be interpreted as referring to an MTL8000 bus interface, unless otherwise indicated. The reason for this is so that topics can be expressed in general Modbus terms which will then be of more general use to a reader.

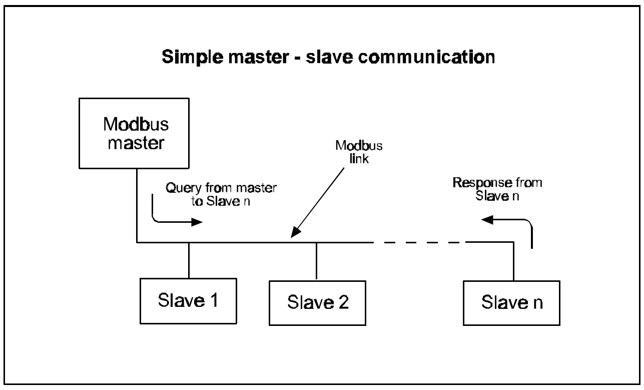


Figure 1 - Master - slave communication

The sequence begins with the master issuing a request or command on to the bus (a 'query'), which is received by the slaves.

The slaves respond by:

- taking appropriate action,
- supplying requested data to the master or
- informing the master that the required action could not be carried out.

The master can address individual slaves or can transmit a message to be received by all slaves - through a 'broadcast' message (Figure 1).

When a slave receives a message addressed specifically to that slave, it will return a message to the master called a 'response'.

The response confirms:

- that the message was received, understood and acted upon, or
- it informs the master that the action required could not be carried out.

If the 'query' requests data from the slave, this will be returned as part of the response.

Messages 'broadcast' to all slaves do not require responses.

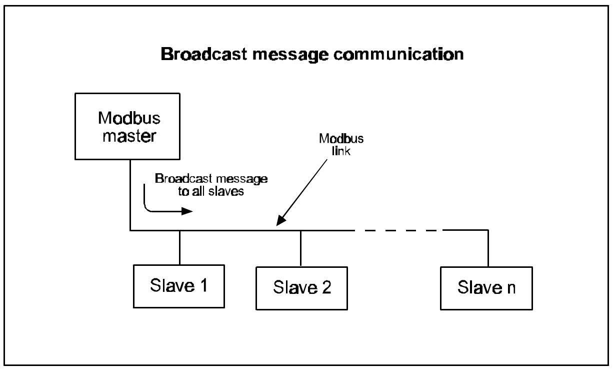

Broadcast message communication
Modbus master
Modbus link
Broadcast message to all slaves
Slave 1
Slave 2
Slave n

Figure 2 - Broadcast communication

Modbus slaves will only transmit on to the network when required to do so by the master. Slaves *never* transmit unsolicited messages.

If the slave cannot carry out the requested action, then it will respond with an error message. This error message, known as an **exception response**, indicates to the master:

- the address of the responding slave,
- the action it was requested to carry out and
- an indication of why the action could not be completed.

If the slave identifies an error during receipt of the message, the message will be ignored. This ensures that a slave does not take action that was really intended for another slave, and does not carry out actions other than those it is commanded to. Should, for some reason, the message be ignored, the master will know that it's query has not been received correctly, as it has not received a response, and will resend it.

Modbus does not define how numerical data shall be encoded within the message. This is decided by the equipment manufacturer and a wide range of options are available.

Modbus ports frequently employ RS232C compatible serial interfaces, though RS422 and RS485 interfaces are also used. The type of interface used defines the connector pin-outs, the cabling and the signal levels; these are not defined in Modbus. Similarly, transmission rates and parity checking are not defined in Modbus and will depend on the serial interface used and the options made available by the manufacturer of each Modbus component.

Modbus will support up to 247 slaves from addresses 1 to 247 (JBUS 1 to 255) - address 0 is reserved for broadcast messages. In practice, the number of slaves that can be used is determined by the physical communications link that is chosen. For example, RS485 is limited to a total of 31 slaves.

The query-response cycle

The query-response cycle forms the basis of all communication on a Modbus network. In all situations it is the master that initiates the query and the slave that responds.

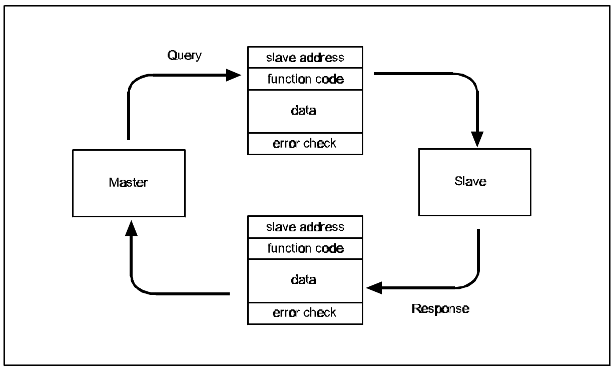


Figure 3 - The query / response cycle

The query

The query is made up of four parts: the device address; the function code; eight bit data bytes; and an error check.

The device address - uniquely identifies a particular slave or indicates that the message is a 'broadcast' addressed to all slaves.

The function code - tells the slave what type of action to perform.

The data - bytes contain any data that the slave will require to carry out the requested function (this may be a register address within the slave, a value to be used by the slave, etc.).

The error check - field allows the slave to confirm the integrity of the message received from the master. If an error is detected, the slave ignores the query and waits for the next query to be addressed to that slave.

The response

A slave will normally be required to provide a response (when a query has been addressed to that slave specifically, and not broadcast to all slaves), which will have the same overall structure format as was used for the query: a device address; a function code; eight bit data bytes; and an error check.

The device address - in the response is that of the addressed slave. This indicates to the master which slave is replying to it's query, and allows it to confirm that the correct slave is responding.

The function code - in the response is normally an exact copy of the function code in the query, and will only vary if the slave is unable to carry out the requested function. In such circumstances the function code returned is a modified form of the original code - this then indicates which function the slave was unable to perform.

The data - contain any data requested in the query.

The error check - allows the master to confirm the integrity of the message received from the slave - if the error check is not correct, the response is ignored.

The transmission mode

There are, in fact, *two* modes available for transmitting serial data over a Modbus network RTU and ASCII. The two transmission modes differ in a number of ways: the way that the bit contents of the messages are defined, the way that information is packed in to the message fields, the way it will be decoded and the speed of operation at a given baud rate. They cannot be used together, and the mode used must be adopted by all the Modbus components throughout the network.

The RTU (Remote Terminal Unit) mode is faster and more robust than ASCII and it is for these reasons that it has been adopted for the MTL8000 Series bus interfaces. None of the MTL8000 Series bus interfaces accommodate the Modbus ASCII mode and so no further space will be devoted to discussing it.

The RTU mode has the following characteristics:

- 8-bit data is encoded in the message as one 8-bit byte .
- The transmitted messages are not printable.
- The start and end of each message is signalled by 'gaps' in transmission.
- Transmission of data within the message must be continuous.
- The error checking employed is a Cyclical Redundancy Check (CRC).

The format for each byte in the RTU mode is:

Coding system:	8-bit binary, comprising two 4-bit hexadecimal characters
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 and 1 or 2 stop bits
Error check field:	Cyclical Redundancy Check (CRC)

The table below shows the encoding of a message:

	Message	RTU
start of frame	-	> 3 chars
slave address	01	0000 0001
function	04	0000 0100
data	04 01	0000 0100 0000 0001
error check	-	1111 0001 1100 1001
end of frame	-	-

The message above requests that the slave with address '01' reads a holding register (function '04') - and returns the values to the master. For simplicity, start, stop and parity bits have been omitted.)

Modbus message framing

Modbus messages must be structured (or 'framed') so that the different Modbus components can detect the start, content structure and end point of a message. It also allows any errors to be detected.

An RTU message begins with a gap in transmission of at least 3.5 character periods (T). Network components monitor the bus continuously and when a 'silent' period of more than 3.5 character periods is detected, the first character following the transmission gap is translated to determine if it corresponds to the device's own address.

START	ADDRESS	FUNCTION	DATA	CRC CHECK	END
T1-T2-T3-T4	8 BITS	8 BITS	n X 8 BITS	16 BITS	T1-T2-T3-T4

The end of the transmitted message is marked by a further interval of at least 3.5 character periods duration. A new message can only begin after this interval.

The entire message field must be transmitted as a continuous stream. If an interval of more than 1.5 character periods is detected during transmission of the message, then the message is assumed to be incomplete and the device returns to waiting for the next device address. The action taken on receipt of an incomplete message is as for receipt of an incorrect message, and it is ignored.

If a new message begins within 3.5 characters periods of the end of the previous frame, the device again ignores the message.

The message fields

The address field

Slave addresses may be in the range 1 to 247 with Modbus (1 to 255 with JBUS). A slave is addressed by the master placing the relevant address in the address field of the query message. When the slave sends its response, it places its own address in the message field to indicate to the master that the correct slave is replying.

Address '0' is used for 'broadcast' messages. All suitable slaves read them, but do not provide responses to such query messages.

The function field

Function codes may be in the range 1 - 255, though not all functions will be supported by all devices. When a message is sent from a master to a slave, the function code defines the action that is required from the addressed slave. Examples of action requested by the various function codes include: read input status; read register content; change a status within the slave; etc..

When the slave sends its response to the master, it will repeat the function code received, to indicate that the slave has understood the query and acted accordingly. If the query instruction could not be carried out by the slave, an 'exception response' is generated and the function code and data fields are used to inform the master of the reason for the exception.

The exception response is generated by returning the original function code from the master, but with its most significant bit set to '1'. Further information regarding the exception response is passed to the master via the data field of the response message. This tells the master what kind of error occurred and allows it to take the most appropriate action - either to repeat the original message, to try and diagnose what has happened to the slave, to set alarms or to take whatever action is most appropriate.

The data field

Firstly, the responses to a number of queries require the slave to inform the master of the number of data bytes that are being returned in the response, and this requires a special implementation within the data field.

A typical example of this would be when the master has requested the slave to communicate the status of a range of registers. The slave responds by repeating the function code and it's own address, followed by the data field. The first byte of the data field identifies the number of bytes that are being returned that contain the register status information.

The main data field then transmits a number of two hexadecimal values, each in the range 00 to FF represented by a single character.

Significantly, the encoding of numerical data is not defined by Modbus, and consequently a number of data formats can be employed. The choice from those available is left to the user.

The data field provides the slave with any additional information needed to perform the function requested in the query. This would typically be a register address, a register range or a value. For some functions, the data field is not required and is not included in the query.

If no errors have occurred, the data field of the response is used by the slave to pass data back to the master. If an error does occur, the data field is used to pass more information to the master relating to the nature of the fault detected.

The error check field

The error checking technique employed on the Modbus network depends on the transmission mode selected. The error check value (in RTU mode) is the result of a Cyclical Redundancy Check (CRC) calculation performed on the message contents. A fuller explanation of this error checking technique is given at the end of this manual in the Appendix.

On receipt of the message, the receiving device also calculates an error check value and then compares it with the error check value in the received message. If these two values do not match exactly, then the receiving device knows that it has not received the message correctly, and disregards it.

Parity checking can be optionally selected.

Data Encoding and Scaling

As has been mentioned, the encoding of numerical data is not defined by the Modbus protocol, and manufacturers supplying products for general use must allow the user to select the data encoding technique. The numerical data formats used include:

- signed (2 s complement) 16-bit integer,
- unsigned 16-bit integer
- 16 bits that represent 16 individual logic states

Another area associated with the encoding of data is the way in which the data is scaled details of such scaling are available from the relevant bus interface documents.

Modbus concepts and nomenclature

Modbus was originally written as a communication protocol for use with Modicon's own PLCs, and the nomenclature and concepts within Modbus reflect this early intention.

The register concept

The Modbus protocol operates on the basis that slaves hold their data in a series of defined status flags and registers, which have defined addresses. A number of flags and registers are set aside for various purposes: single and multi bit output and single and multi bit input/output.

Register and flag organisation and addressing

The registers and flags within Modbus devices are normally grouped as below:

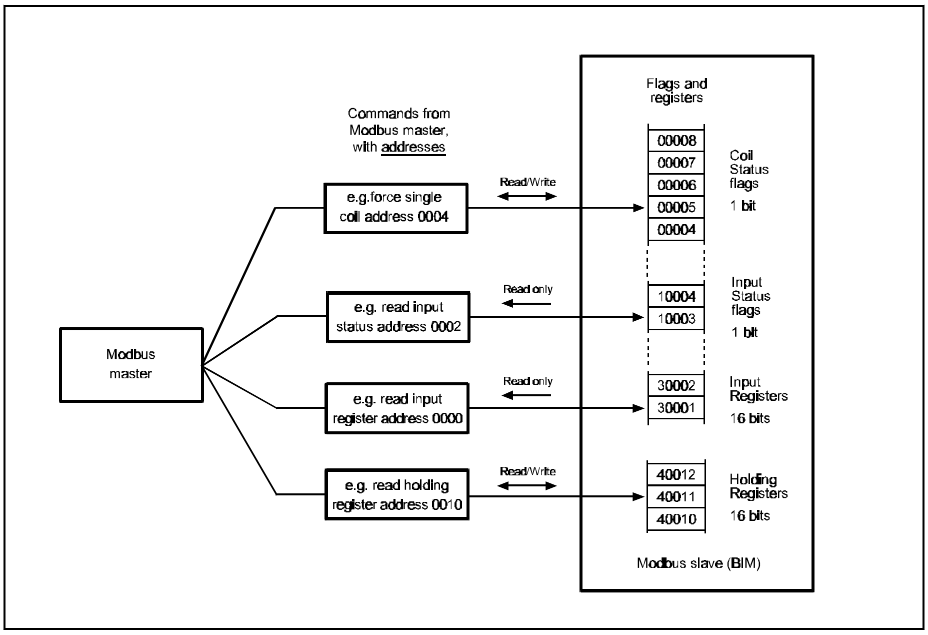


Figure 4 - Addressing Modbus registers

It is not absolutely necessary for manufacturers of Modbus devices to follow the register numbering adopted by Modicon's PLCs, but it is conventional to do so wherever possible.

Note: Commands issued by the master, relating to individual coils or registers, always use an **address** *not the number of the coil or register*, and all data addresses in Modbus messages are referenced to zero. Consequently, the first occurrence of a data item (e.g. coil 1) is addressed as item number zero. Other examples of the addressing pattern are given below.

- The coil known as coil 1 in the slave 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 that precedes the data in the message already specifies the operation as relating to a holding register; therefore, the 4xxxx reference is implicit and does not need the additional digit.
- Holding register 40108 is *addressed* as register 006B hex (107 decimal).

COIL STATUS flags (read/write)

The single bit 'COIL STATUS' flags are conventionally numbered 0xxxx, and store digital output information where an output is defined as being *from* the slave *to* the field. The Modbus master can both read from, and write to, these flags.

Note: *The term 'coil' comes from the original PLC application, in which the outputs from the PLC were set by controlling the coils of the output relays.*

INPUT STATUS flags (read only)

The single bit 'INPUT STATUS' flags are conventionally numbered 1xxxx. They store the status of digital inputs to the Modbus slave and can only be *read* by the master. There is no facility, nor any need, for the master to write to these locations.

INPUT REGISTERS (read only)

The 16-bit 'INPUT REGISTERS' are conventionally numbered 3xxxx. They store data which has been collected from the field by the Modbus slave . These registers can only be *read* by the Modbus master.

HOLDING REGISTERS (read/write)

The 16-bit 'HOLDING REGISTERS' are conventionally numbered 4xxxx. They hold general purpose data within the slave. . The master can both read from, and write to, these flags. A typical application for these registers would be to hold configuration data.

Data & Control Functions

The following table lists a range of Modbus functions some or all of which are supported by the MTL8000 bus interfaces.

FUNCTION	No.
READ COIL STATUS	01
READ INPUT STATUS	02
READ HOLDING REGISTERS	03
READ INPUT REGISTERS	04
FORCE SINGLE COIL	05
PRESET SINGLE REGISTER	06
DIAGNOSTICS	08
FETCH COMM EVENT COUNTER	11
FETCH COMM EVENT LOG	12
FORCE MULTIPLE COILS	15
PRESET MULTIPLE REGISTERS	16
REPORT SLAVE ID	17

Note: This section contains a number of detailed tables that demonstrate the construction of messages passed along the Modbus network. However, most Modbus masters have a user-interface that shelters the user from many of these details, and will only require set-up of the slave address, the function code, the initial coil or register location and the number of coils or registers to be read. High-level users will not need not concern themselves with the details presented here.

Some of the values are shown as hexadecimal (hex) some as binary. The hex values are given to describe the code or value that must be sent in the query and the response. The binary code for transmission in RTU mode is given directly, and it will be seen that this is a simple encoding of the equivalent hex value.

In the body of the text, decimal values are used, so as to be consistent with the numbering of the function codes in Revision 'G' of the Modbus specification. Where the encoding of these decimal values in to hex makes them appear differently in the table, the hex value is given in parenthesis - e.g. function code 10 ('OA' in hex). For simplicity, start, stop and parity bits are ignored throughout.

Read Coil Status (function 01)

This function requests that the slave reads the status of a specified range of it's single bit input/output flags and returns these to the master. The range of flags to be read is given in the query, by the master indicating the address of the first flag to be read and then total number of subsequent flags - including the first.

The example in the following table shows the query required to read the status of coils 00001 to 00009 of slave number 20 (14 in hex.). The start address and number of coils to be read are always transmitted as two bytes - most significant bits (MSB) first, followed by the least significant bits (LSB):

FIELD NAME	HEX. VALUE	RTU
slave address	14	0001 0100
function	01	0000 0001
starting address MSB	00	0000 0000
starting address LSB	00	0000 0000
no. of locations MSB	00	0000 0000
no. of locations LSB	09	0000 1001
error check	-	CRC

Note: Recall that, as the address is always 1 less than the least significant 4 digits of the coil number, the coil '10001' is addressed as '00 00' (hex).

The normal response to a READ COIL STATUS query contains the slave address, the repeated function code, the number of data bytes that are being transmitted in the response, the data bytes themselves and the error check.

The data bytes encode the status of the flags so that the status of the first flag to be read forms the LSB of the first data byte. Subsequent flag states form the next most significant bits of the first byte - thus if the master had requested the status of 8 flags, the data would be transmitted in a single data byte, with the LSB being the status of the first flag and the MSB being the status of the eighth. This is continued so that the status of the ninth flag requested forms the LSB of the second data byte.

If the master requests the status of a number of flags so that it is not possible to return 'complete' 8-bit data bytes (e.g. if the master requests the status of 9 flags, as above, which would require one complete 8-bit byte and a single bit), then the last data byte to be transmitted is 'packed' with '0's in it's MSBs.

The convention followed for the status is: 1 = ON; 0 = OFF.

A response to the query above would have the following format:

FIELD NAME	HEX. VALUE	RTU
slave address	14	0001 0100
function	01	0000 0001
number of data bytes returned	02	0000 0010
first data byte	XX	XXXX XXXX
second data byte	0X	0000 000X
error check	-	CRC

Note: The seven most significant bits of the second data byte in RTU mode are zero, as the query only requested the status of 9 inputs. The seven zeros were packed in to the response to allow the slave to return complete 8-bit data bytes.

Read Input Status (function 02)

This function requests that the slave reads the status of a specified range of it's single bit output flags and returns these to the master. The range of inputs to be read is given in the query, by the master indicating the address of the first input to be read and then total number of subsequent flags - including the first.

The example below shows the query required to read the status of flags 10001 to 10030 of slave number 17 (11 in hex.). The start address and number of flags to be read are always transmitted as two bytes - most significant bits (MSB) first, followed by the least significant bits (LSB):

FIELD NAME	HEX. VALUE	RTU
slave address	11	0001 0001
function	02	0000 0010
starting address MSB	00	0000 0000
starting address LSB	00	0000 0000
no. of locations MSB	00	0000 0000
no. of locations LSB	1E	0001 1110
error check	-	CRC

Note: Recall that, as the address is always 1 less than the least significant 4 digits of the input number, the input '10001' is addressed as '00 00' (hex).

The normal response to a READ INPUT STATUS comprises the slave address, the repeated function code, the number of data bytes that are being transmitted in the response, the data bytes themselves and the error check.

The data bytes encode the status of the inputs so that the status of the first input to be read forms the LSB of the first data byte. Subsequent input states form the next most significant bits of the first byte - thus if the master had requested the status of 8 inputs, the data would be transmitted in a single data byte, with the LSB being the status of the first input and the MSB being the status of the eighth. This is continued so that the status of the ninth input requested forms the LSB of the second data byte.

If the master requests the status of a number of inputs so that it is not possible to return 'complete' 8-bit data bytes (e.g. if the master requests the status of 30 inputs as in the above request, which would require three complete 8-bit bytes and 6 single bits), then the last data byte to be transmitted is 'packed' with '0's in it's MSBs.

The convention followed for the status is: 1 = ON; 0 = OFF.

A response to the query above would have the following format:

FIELD NAME	HEX. VALUE	RTU
slave address	11	0001 0001
function	02	0000 0010
number of data bytes returned	4	0000 0100
first data byte	xx	xxxx xxxx
second data byte	xx	xxxx xxxx
third data byte	xx	xxxx xxxx
fourth data byte	xx	00xx xxxx
error check	-	CRC

Note: The two most significant bits of the fourth data byte in RTU mode are zero, as the query only requested the status of 30 inputs. The two zeros were packed in to the response to allow the slave to return complete 8-bit data bytes.

Read Holding Registers (function 03)

This function requests that the slave reads the binary contents of a specified range of its 16-bit holding registers and returns the values to the master. The range of registers to be read is given in the query, by the master indicating the address of the first register and the total number of subsequent registers to be read - which includes the first register.

The example below shows the query that requests the values held in holding registers 40108 to 40110 in slave 17 (108 and 17 are 6C and 11 in hex).

FIELD NAME	HEX. VALUE	RTU
slave address	11	0001 0001
function	03	0000 0011
starting address MSB	00	0000 0000
starting address LSB	6B	0101 1011
no. of registers MSB	00	0000 0000
no. of registers LSB	03	0000 0011
error check	-	CRC

Note: Recall that, as the address is always 1 less than the least significant 4 digits of the register number, register '40108' is addressed as 0107 (hex. value '6B').

The normal response returns the slave address, the repeated function code, the number of data bytes that are being transmitted in the response, the data bytes themselves and the error check.

The data bytes encode the contents of the holding registers 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 byte the low order bits.

A response to the query above would have the following format:

FIELD NAME	HEX. VALUE	RTU
slave address	11	0001 0001
function	03	0000 0011
number of data bytes returned	06	0000 0110
first data byte (MSB 40108)	xx	xxxx xxxx
second data byte (LSB 40108)	xx	xxxx xxxx
third data byte (MSB 40109)	xx	xxxx xxxx
fourth data byte (LSB 40109)	xx	xxxx xxxx
fifth data byte (MSB 40110)	xx	xxxx xxxx
sixth data byte (LSB 40110)	xx	xxxx xxxx
error check	-	CRC

Read Input Registers (function 04)

This function requests that the slave reads the binary contents of a specified range of its 16-bit input registers and returns the values to the master. The range of inputs to be read is given in the query, by the master indicating the address of the first register and the total number of registers to be read - including the first register.

The example below shows the query required to read the values held in input register 30009 from slave 31 (1F in hex).

FIELD NAME	HEX. VALUE	RTU
slave address	1F	0001 1111
function	04	0000 0100
starting address MSB	00	0000 0000
starting address LSB	08	0000 1000
no. of points MSB	00	0000 0000
no. of points LSB	01	0000 0001
error check	-	CRC

Note: Recall that, as the address is always 1 less than the least significant 4 hex. values of the register number, register '30009' is addressed by the hex. value '00 08'.

The normal response to a READ INPUT REGISTERS query comprises the slave address, the repeated function code, the number of data bytes that are being transmitted in the response, the data bytes themselves and the error check.

The data bytes encode the contents of the input registers as two bytes per register. For each register, the first byte contains the high order bits and the second byte the low order bits.

A response to the query above would have the following format:

FIELD NAME	HEX. VALUE	RTU
slave address	1F	0001 1111
function	04	0000 0100
number of data bytes returned	02	0000 0010
first data byte (MSB)	xx	xxxx xxxx
second data byte (LSB)	xx	xxxx xxxx
error check	-	CRC

Force Single Coil (function 05)

The FORCE SINGLE COIL function requests that the slave sets a specified input/output flag to a particular status. The address of the flag to be set is given in the query. The status to which the flag must be set is provided by two data bytes. If the flag is to be set to '1', then the data bytes sent are FF 00. If the flag is to be set to '0' the data bytes are 00 00.

The example below shows the query required to force the status of a flag with address 10065 of slave 18 to '1'. (65 and 18 are '41' and '12' in hex.)

FIELD NAME	HEX. VALUE	RTU
slave address	12	0001 0010
function	05	0000 0101
flag address MSB	00	0000 0000
flag address LSB	40	0100 0000
force data MSB	FF	1111 1111
force data LSB	00	0000 0000
error check	-	CRC

The normal response to a FORCE SINGLE COIL query comprises the slave address, an echo of the function code, echoes of the flag address and status request, and an error check.

A response to the query above would have the following format:

FIELD NAME	HEX. VALUE	RTU
slave address	12	0001 0010
function	05	0000 0101
flag address MSB	00	0000 0000
flag address LSB	40	0100 0000
force data MSB	FF	1111 1111
force data LSB	00	0000 0000
error check	-	CRC

Preset Single Register (function 06)

The Preset Single Register function requests the slave writes specified data in a particular register. The address of the register to be written to is given in the query. The data to be written is provided by two data bytes.

The example below shows the query required to pre-set or write a register so that it holds the value FF FF . The register location is 40003 of slave 1.

FIELD NAME	HEX. VALUE	RTU
slave address	01	0000 0001
function	06	0000 0110
register address MSB	00	0000 0000
register address LSB	02	0000 0010
pre-set data MSB	FF	1111 1111
pre-set data LSB	FF	1111 1111
error check	-	CRC

The normal response to a Preset Single Register query comprises the slave address, and echo of the function code, echoes of the register address and the pre-set data, and an error check.

A response to the query above would have the following format:

FIELD NAME	HEX. VALUE	RTU
slave address	01	0000 0001
function	06	0000 0110
register address MSB	00	0000 0000
register address LSB	02	0000 0010
pre-set data MSB	FF	1111 1111
pre-set data LSB	FF	1111 1111
error check	-	CRC

Diagnostics (function 08)

The diagnostics function has a number of tests to check the communications link between the master and the slave. The type of test is identified by a sub-function code in the first two bytes of the data field.

Some of the tests specified by the subfunctions require the slave to return data in the response to the query, others only require the slave to acknowledge receipt of the response in the normal way. Responses to diagnostic functions will return the subfunction code as well as the 08 function code.

There are a number of sub-function codes and the user is referred to the instruction manual appropriate to the MTL8000/8800L equipment of interest for information on the particular codes in use. The following lists and then describes some of the most used sub-functions.

SUB-FUNCTION	No.
RETURN QUERY DATA	00
RESTART COMM OPTION	01
RETURN DIAGNOSTIC REGISTER	02
CHANGE ASCII I/P DELIMITER	03
FORCE LISTEN ONLY MODE	04

Return Query Data (sub-function 00 00)

The diagnostic subfunction Return Query Data requests the addressed slave to return (loop-back) to the master an exact copy of the data contained in the query. This is a useful method for confirming correct communication.

An example of a query and response with this subfunction is given below. The master requests slave number 4 to return the hexadecimal data 'AA BB'.

The query:

FIELD NAME	HEX. VALUE	RTU
slave address	04	0000 0100
function	08	0000 1000
subfunction MSB	00	0000 0000
subfunction LSB	00	0000 0000
data MSB	AA	1010 1010
data LSB	BB	1011 1011
error check	-	CRC

The response echoes the original message:

FIELD NAME	HEX. VALUE	RTU
slave address	04	0000 0100
function	08	0000 1000
subfunction MSB	00	0000 0000
subfunction LSB	00	0000 0000
data MSB	AA	1010 1010
data LSB	BB	1011 1011
error check	-	CRC

Restart Comm Option (sub-function 00 01)

This code causes the slave to be reset and all counters to be cleared. If Continue on Error (FF 00) is specified, the COMM EVENT LOG will be cleared. A normal response is sent before the restart is executed. If the slave is in Listen Only Mode when the Restart command is received, no response will be initiated.

Restart is the only command that will bring the slave out of Listen Only Mode. If this command is received, a restart is attempted, executing the power-up confidence tests. Successful completion of these tests will put the unit back in on-line mode.

An example of a query and response with this subfunction is given below. The master is attempting to restart slave number 8 and is also specifying the Continue on Error command (FF 00).

The query:

FIELD NAME	HEX. VALUE	RTU
slave address	08	0000 1000
function	08	0000 1000
subfunction MSB	00	0000 0000
subfunction LSB	01	0000 0000
data MSB	FF	1111 1111
data LSB	00	0000 0000
error check	-	CRC

The normal response to a Restart Comm Option, unless in Listen Only mode, comprises the slave address, and echo of the function code, echoes of the register address and the pre-set data, and an error check.

Return Diagnostic Register (sub-function 00 02)

The diagnostic subfunction Return Diagnostic Register requests that the slave reads the contents of its diagnostic register and returns the binary data values to the master.

The query sends two zero data bytes, following the data bytes containing the subfunction code. The response returns two 8-bit data bytes containing the register data.

The contents of the diagnostic register may be defined by the manufacturer, according to the needs of each Modbus slave.

The following example shows the query and response generated when the master requests diagnostic subfunction 00 02 from the slave with address 12 ('0C' in hex.).

The query:

FIELD NAME	HEX. VALUE	RTU
slave address	0C	0000 1100
function	08	0000 1000
subfunction MSB	00	0000 0000
subfunction LSB	02	0000 0010
data MSB	00	0000 0000
data LSB	00	0000 0000
error check	-	CRC

The response:

FIELD NAME	HEX. VALUE	RTU
slave address	0C	0000 1100
function	08	0000 1000
subfunction MSB	00	0000 0000
subfunction LSB	02	0000 0010
data MSB	XX	XXXX XXXX
data LSB	XX	XXXX XXXX
error check	-	CRC

Change Input Delimiter Character (sub-function 00 03)

The character passed by the master in the data field provides a means to define, or re-define, the line feed (LF) character used to delimit an ASCII message.

An example of a query using this subfunction is given below. The master is telling slave number 4 to use the character 0A as an ASCII message delimiter.

The query:

FIELD NAME	HEX. VALUE	RTU
slave address	04	0000 0100
function	08	0000 1000
subfunction MSB	00	0000 0000
subfunction LSB	03	0000 0000
data MSB	0A	0000 1010
data LSB	00	0000 0000
error check	-	CRC

A normal response, that echoes the query, is issued by the slave.

Force Slave to Listen only Mode (sub-function 00 04)

This function forces the slave into listen only mode (LOM). When in this mode the slave monitors transmitted data but no response will be issued. The only query that will be recognised is the Restart command (see Diagnostics sub-function 00 01).

When in LOM all active controls are immediately turned off and the Ready watchdog timer is allowed to expire, locking these controls off. This isolates a failed unit from other stations on the line, allowing the others to continue full communication without interference.

The following example shows the query when the master uses diagnostic subfunction 00 04 to force slave 12 ('0C' in hex.) into Listen only Mode.

The query:

FIELD NAME	HEX. VALUE	RTU
slave address	0C	0000 1100
function	08	0000 1000
subfunction MSB	00	0000 0000
subfunction LSB	04	0000 0100
data MSB	00	0000 0000
data LSB	00	0000 0000
error check	-	CRC

As stated earlier, no response is generated by the slave for this query.

Fetch Communications Event Counter (function 11)

This function requests that the slave returns a status word and an event count from the its 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 by this function.

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 subfunction 00 01 or 00 10 (decimal). See the Diagnostics section for more details.

The example below shows the query required to fetch the communications event counter in slave 17 (11 hex).

FIELD NAME	HEX. VALUE	RTU
slave address	11	0001 0001
function	0B	0000 1011
error check	-	CRC

The normal response contains the slave address, an echo of the function code, a two byte status word, a two byte event count and an error check. The status word will contain 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.

A response to the query above might look like the following, where the status word - FF FF hex - indicates that the slave is busy with another function. The event count shows that 264 (01 08 hex) events have been counted by the controller.

FIELD NAME	HEX. VALUE	RTU
slave address	11	0001 0001
function	06	0000 1011
status MSB	FF	1111 1111
status LSB	FF	1111 1111
event count data MSB	01	0000 0001
event count data LSB	08	0000 1000
error check	-	CRC

Fetch Communications Event Log (function 12)

This function requests that the slave return a status word, an event count, a message count and a field of event bytes. Broadcast is not supported by this function.

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

The message counter contains the quantity of messages processed by the slave since its last restart, clear counters operation or power-up. The count is identical to that returned by diagnostic sub-function Return Bus Message Count (code 00 11).

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 pushed into the data field as they occur. At any time, byte 0 is the most recent event. When full , a new byte causes the oldest byte to be flushed from the field.

The example below shows the query required to fetch the communications event counter in slave 17 (11 hex).

FIELD NAME	HEX. VALUE	RTU
slave address	11	0001 0001
function	0B	0000 1011
error check	-	CRC

A response to this query might be:

FIELD NAME	HEX. VALUE	RTU
slave address	11	0001 0001
function	0C	0000 1100
number of data bytes returned	46	0100 0110
status MSB	00	0000 0000
status LSB	00	0000 0000
event count data MSB	01	0000 0001
event count data LSB	08	0000 1000
message count data MSB	01	0000 0001
message count data LSB	21	0010 0001

event 0	20	0010 0000
event 1	00	0000 0000
error check	-	CRC

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

The most recent event is shown in the event 0 byte. Its contents (20 hex) show that it has recently entered the Listen Only Mode. The previous event - event 1 - shows that the slave received a Communications Restart (00 hex).

What Event Bytes Contain

The Modbus specification states that an event byte returned by the Fetch Communications Event Log function can be one of the following four types:

- Slave Modbus Receive Event
- Slave Modbus Send Event
- Slave Entered Listen Only Mode
- Slave Initiated Communication Restart

The type is defined by bit 7 (the highest bit) in each byte. It may be further defined by bit 6, as explained below.

Slave Modbus Receive Event

If a slave receives a query message, before processes the message it will store an event byte to record its arrival. This event is defined by bit 7 being set to a logic 1 . The other bits will be set to a 1 if the corresponding condition is TRUE.

BIT	MEANING
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 (by definition)

Slave Modbus Send Event

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

BIT	MEANING
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 (by definition)
7	0 (by definition)

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 marked by the byte contents being set to 04 (hex).

BIT	7	6	5	4	3	2	1	0
CONTENTS	0	0	1	0	0	0	0	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 sub-function Restart Communications option (code 00 01).

When that function is used it also places the slave into a Continue on Error or Stop on Error mode. If it is placed into a Continue on Error mode, the event byte is added to the existing event log. If the slave is placed into a Stop on Error mode, the byte is added to the log and the rest of the log is cleared to zeroes.

The event byte is marked by all the bits being set to zeroes.

Force Multiple Coils (function 15)

This function requests that the slave forces each coil in a sequence of coils to either ON or OFF. When broadcast, the function forces the same coil references in all slaves.

The requested ON/OFF states are specified by the 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.

In addition to the slave address and the function number the query will contain the starting coil number, the number of coils in the sequence and data on the individual states to which they are to be forced.

The example below shows the request to force a series of ten coils, starting at coil 20 (addressed as 19, or 13 hex) in slave device 17. Two data bytes are required to cover the settings for all ten coils: 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 the 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 data byte should be zero-filled.

FIELD NAME	HEX. VALUE	RTU
slave address	11	0001 0001
function	0F	0000 1111
coil address MSB	00	0000 0000
coil address LSB	13	0001 0011
number of coils MSB	00	0000 0000
number of coils LSB	0A	0000 1010
byte count	02	0000 0010
force data MSB	CD	1100 1101
force data LSB	01	0000 0001
error check	-	CRC

The normal response returns the slave address, function code, starting address and number of coils forced. A response to the above query might be:

FIELD NAME	HEX. VALUE	RTU
slave address	11	0001 0001
function	0F	0000 1111
coil address MSB	00	0000 0000
coil address LSB	13	0001 0011
number of coils MSB	00	0000 0000
number of coils LSB	0A	0000 1010
error check	-	CRC

Preset Multiple Registers (function 16)

This function requests that the slave writes specified data to a range of registers. The range of registers to be written is identified by the master which indicates the location of the first register and then the total number of registers to be written - including the first.

Note: The registers must be consecutive for this command to succeed. If the required registers are not consecutive then multiple commands must be issued.

The example below shows the query required to pre-set or write two registers so that they both contain the value FF FF . The first register location is 40003 of slave 1. Function 16 is

FIELD NAME	HEX. VALUE	RTU
slave address	01	0000 0001
function	10	0001 0000
register address MSB	00	0000 0000
register address LSB	02	0000 0010
register number MSB	00	0000 0000
register number LSB	02	0000 0010
byte count	04	0000 0100
1st pre-set data MSB	FF	1111 1111
1st pre-set data LSB	FF	1111 1111
2nd pre-set data MSB	FF	1111 1111
2nd pre-set data LSB	FF	1111 1111
error check	-	CRC

The normal response to a PRESET MULTIPLE REGISTERS query comprises the slave address, an echo of the function code, echoes of the register address and the pre-set data, and an error check.

A response to the query above would have the following format:

FIELD NAME	HEX. VALUE	RTU
slave address	01	0000 0001
function	10	0001 0000
register address MSB	00	0000 0000
register address LSB	02	0000 0010
register number MSB	00	0000 0000
register number LSB	02	0000 0010
error check	-	CRC

Report Slave ID (function 17)

The Report Slave ID function permits the user to obtain information on the slave type and RUN status.

The query for slave address 1 would simply state the function code 17 (11hex):

FIELD NAME	HEX. VALUE	RTU
slave address	01	0000 0001
function	11	0001 0000
error check	-	CRC

A response to the query above would have the following format:

FIELD NAME	HEX. VALUE	RTU
slave address	01	0000 0001
function	10	0001 0000
number of data bytes returned	02	0000 0010
slave ID	01	0000 0001
run status	FF	1111 1111
error check	-	CRC

The above response consists of a 2-byte reply which identifies a) the type of slave fitted which will depend upon the equipment in use and b) the RUN status 00 indicating false and

Exception Responses

If a slave receives a message correctly (i.e. it passes the error checking) but then finds it is unable to perform the required operation will issue one of five available exception responses. The following sections describe the construction of the exception responses in general, then describes in more detail the ones that are implemented in some of the MTL8000 bus interfaces..

Construction of exception responses

In a normal response, the slave exactly echoes the function code received from the master. In an exception response the slave responds similarly, but with the MSB of the function code set to '1'. The master can therefore identify that an exception response is being returned, and identify the function code that was received by the slave. This is possible as there are less than 128 (or 80 hex.) function codes defined which, as binary 8-bit numbers, must always have a '0' as their MSB.

The example below shows the first few bytes of an exception response issued after slave 9 correctly received a function code '01' that it was then unable to perform.

FIELD NAME	HEX. VALUE	RTU
slave address	09	0000 1001
function (as an exception for 01)	81	1000 0001

Further data is passed to the master via the first bytes of the response's data field. The bytes returned are referred to as the 'exception code' and these are used to provide the master with additional information regarding the nature of the exception.

The exception code that is generated by the slave for any particular event, can be determined by the manufacturer of the device. It is normal, however, to try and use the exception codes so that the code name is as near as possible, in meaning, to the event that has caused the exception.

The example below shows the full exception response for the example used earlier - with the reason for the exception being identified as exception code '02'. This is the 'ILLEGAL DATA ADDRESS', which would typically be used if the master had requested the slave to read a non-existent status location.

FIELD NAME	HEX. VALUE	RTU
slave address	09	0000 1001
function (as an exception for 01)	81	1000 0001
exception code	02	0000 0010
error check	-	CRC

Exception response types

Some or all of the following exception responses are supported by MTL8000 Series equipment:

ILLEGAL FUNCTION	response 01
ILLEGAL DATA ADDRESS	response 02
ILLEGAL DATA VALUE	response 03
SLAVE DEVICE FAILURE	response 04
ACKNOWLEDGE	response 05
SLAVE DEVICE BUSY	response 06
NEGATIVE ACKNOWLEDGE	response 07

Note that if a slave receives a message which does not pass the error checking employed, it will discard the message and will not issue a response. This prevents it from carrying out operations that have either not been translated correctly or which were intended for another slave. The master employs a 'time-out' check, and if it has not received a response after a given time period, it will re-try or take other appropriate action.

Illegal Function (exception code 01)

The ILLEGAL FUNCTION exception code is used to inform the master that the function code received by the slave is not an allowable function for that particular slave.

An example of such a request would be the function code FORCE SINGLE COILS (function 05). If the master were to send a request containing such a function code, an exception code '01' would be returned. The example below shows the exception response returned by such a slave, with address '04' (hex):

FIELD NAME	HEX. VALUE	RTU
slave address	04	0000 0100
function (as an exception for 05)	85	1000 0101
exception code	01	0000 0000
error check	-	CRC

Illegal Data Address (exception code 02)

The ILLEGAL DATA ADDRESS exception code is used to inform the master that an address used in the query is not available within the slave.

An example of such a request would be the function code READ INPUT REGISTERS ('04') with the number of registers to be read given as 61. The slave has a communication buffer that is only capable of containing sixty input registers, and a request to read more than this number could not be handled by the unit. The example below shows the exception response returned by such a slave, with address '09' (hex):

FIELD NAME	HEX. VALUE	RTU
slave address	09	0000 1001
function (as an exception for 04)	84	1000 0100
exception code	02	0000 0010
error check	-	CRC

Illegal Data Value (exception code 03)

The ILLEGAL DATA VALUE exception code is used to inform the master that a value used in the query is not valid for the function requested form that slave.

An example of such a request would be the function code for DIAGNOSTICS with a diagnostic sub-code of '01'. If the slave does not support this diagnostic code it would return the exception code above. The example below shows the exception response returned by such a slave, with address '06' (hex):

FIELD NAME	HEX. VALUE	RTU
slave address	06	0000 0110
function (as an exception for 08)	88	1000 1000
exception code	03	0000 0011
error check	-	CRC

Failure in subsystem (exception code 04)

The FAILURE IN SUBSYSTEM exception code is used when the slave is capable of responding to the master but elements of it are not responding correctly. This is in comparison to a slave that is incapable of responding whereupon the user might be given an exception code 02 (Illegal Data Address) to indicate the effective absence of the slave.

An example of such a request would be the function code READ HOLDING REGISTERS ('03'). If the slave has corrupted configuration information, so that it is unable to respond, it might return exception code 04 to indicate that it is still able to reply but is incapable of providing the requested data.

The following example shows the exception response returned by such a slave, with address '09' (hex):

FIELD NAME	HEX. VALUE	RTU
slave address	09	0000 1001
function (as an exception for 03)	83	1000 0100
exception code	04	0000 0100
error check	-	CRC

Acknowledge (exception code 05)

The ACKNOWLEDGE exception code is used to inform the master that the query has been received correctly but a delay will be involved in responding as some processing is required to produce a response. Further polling of the slave while in this condition will produce a rejected message response - see exception code 06.

An example of such a request would be the function code READ INPUT REGISTERS ('04'). If a large range of registers is required to be read then the response will be delayed. The example below shows the exception response returned by such a slave, with address '11' (hex):

FIELD NAME	HEX. VALUE	RTU
slave address	11	0001 0001
function (as an exception for 04)	84	1000 0100
exception code	05	0000 0100
error check	-	CRC

Slave Device Busy (exception code 06)

The SLAVE DEVICE BUSY exception code is used if the slave is involved in processing an earlier command, or is otherwise occupied. It acknowledges to the master that the query has been received but it is too busy to respond at present.

An example of such a request would be the function code FORCE SINGLE COIL ('05'). If a large range of registers is required to be read then the response will be delayed. The example below shows the exception response returned by such a slave, with address '17' (hex):

FIELD NAME	HEX. VALUE	RTU
slave address	17	0001 0001
function (as an exception for 05)	85	1000 0100
exception code	06	0000 0100
error check	-	CRC

Negative Acknowledge (exception code 07)

The NEGATIVE ACKNOWLEDGE exception code is used to inform the master that the slave cannot perform the requested function.

An example of such a failure would be attempting to write data in to a register that was 'write disabled'. If the MODBUS master attempts to write in to such a register, then the exception code '07' will be returned. The following table shows the exception code returned by such a slave, with address 04.

FIELD NAME	HEX. VALUE	ASCII	RTU
slave address	04	30 33	0000 0011
function (as an exception for 06)	86	38 36	1000 0110
exception code	07	30 37	0000 0111
error check	-	LRC	CRC

The Modbus physical layer

The serial interface used by most Modbus masters is RS232C. This interface does not allow the Modbus network to extend beyond 10 to 20 metres in length, hence, many manufacturers use other serial interfaces that have longer network capabilities.

RS485 allows simple parallel connection of a number of units. As explained later the RS485 interface supports both 2 and 4 wire connection (plus common), and the latter allows use with a RS422 host.

Note: when a non-RS232 interface is used with an RS232 master, a data converter must be inserted in to the network.

The RS485 serial interface standard

The RS485 standard defines the characteristics of the drivers and receivers that are connected to the bus. It does not define the cabling or connectors used, nor does it specify a particular data rate or signal format.

RS485 employs differential signalling and therefore requires at least two connectors per signal and a 'common' line connected between all devices.

RS485 specifies that the two differential signal lines should be marked 'A' and 'B', but this is not always followed. Many are marked with '+' and '-', which describes the relative voltages of the signalling lines in their quiescent state.

Note: With RS485, no damage will occur if the signalling lines are connected with the wrong polarity - but the system will not operate.

RS485 effectively limits the number of addresses supported to 32 units (unless buffers are used), which in Modbus corresponds to 31 slaves (with 1 master).

Terminations

RS485 interfaces should ideally be provided with a matched termination to prevent reflections and 'ringing' of the signal on the bus cabling. The termination will normally be a simple resistive terminator having an impedance that matches the characteristic of the bus - typically 100Ω. In practice, with low data rates and relatively short networks, it is often unnecessary to terminate the bus.

Biasing

When no communication is taking place, the bus is in an undefined, floating state and noise on the bus may be decoded as real characters. Well-written software should discard most of these characters, but the system can be further protected by *biasing* the bus to a known state and thereby reducing any noise that may be present.

2- and 4-wire interconnection

When using RS485, the user can select either two-wire or four-wire interconnection. The two systems are shown below.

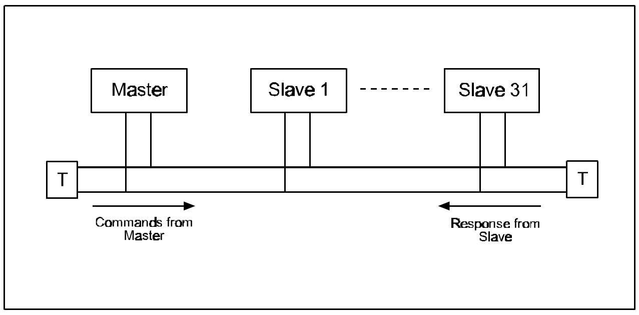


Figure 5 - 2 wire interconnection

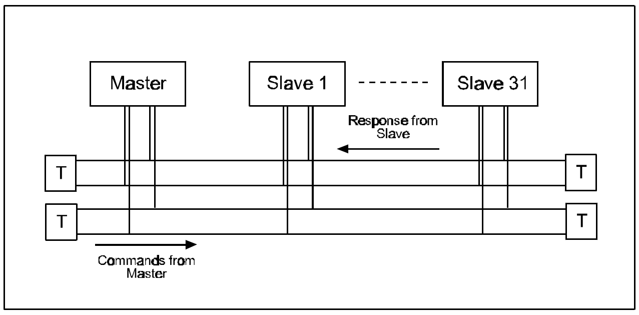


Figure 6 - 4 wire interconnection

In simple terms, two-wire uses the same pair of wires to transmit queries from the master and responses from the slave. With four-wire, queries are sent out on one pair of wires and responses are returned on another set. Note, the nomenclature 'two-' and 'four-wire' ignores the common wire, which is required to connect to all devices, but it is often not shown on interconnection diagrams.

With some protocols, with the adoption of 'four-wire' interconnection, the system can be made to run more quickly, by allowing simultaneous communication of commands and responses. This is not possible with Modbus, as the 'Query-Response' cycle ensures that simultaneous communication by both the master and the addressed slave can never occur. When using Modbus, the decision to use 'two-' or 'four-wire' interconnection is generally made according to the type of serial data interface that is made available on the host.

Point-to-point and multi-drop connection

Point-to-point and multi-drop are methods of forming the network link between the master and its slave or slaves. Point-to-point describes the connection between a master and a single slave, whereas multi-drop allows a number of slaves to be connected to one master.

Linear bus connection is a multi-drop technique for connecting a number of slaves to a single master. The method is shown below.

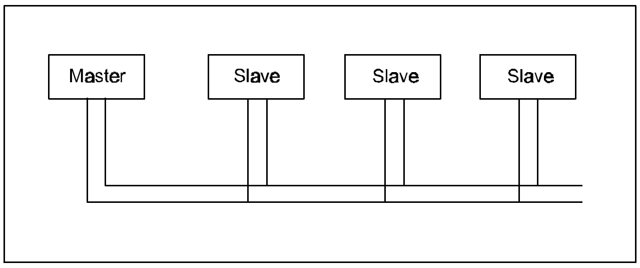


Figure 7 - Linear bus connection

Note on RS422: the only direct connection method that is theoretically allowed when using a Modbus host with an RS422 serial interface is point-to-point. The specification for RS422 indicates that it would be necessary to employ an RS422 to RS485 converter to connect more than one Modbus slave. However, in practice, RS422 is found to perform well when RS485 slaves are linear bus connected to the RS422 host.

The theory maintains that since RS422 has no tri-state facility it cannot be connected to more than one slave. However, since the Tx and Rx lines of the host are always ready to transmit to or receive from a slave, there is no need for the host to adopt a tri-state mode. This then allows linear BUS connection on an RS422 host with RS485 slaves.

Data converters

Most Modbus hosts will not provide an RS485 serial interface as standard, with RS232 and RS422 being much more common. Apart from the possibilities of using RS422 as outlined above, in many applications it will therefore be necessary to use a data converter to allow the connection of the Modbus RS485 interface to the host's RS232 or RS422 interface.

Appendix - Calculation of the CRC in RTU mode

The Cyclical Redundancy Check (CRC) used for RTU transmission mode is 16-bits long and is transmitted as two 8-bit bytes. It is calculated by the transmitting device and added to the message. It is also calculated by the receiving device, from the contents of the message, and then compared with the value contained in the message. If the two values are not equal, then an error has occurred in either the transmission or the reception of the message.

The CRC is calculated by carrying out a process of exclusive OR operations. The first byte is exclusive OR'ed with the value 1111 1111 1111 1111, and the result is used for any subsequent exclusive OR operations that are required. Only the eight bits of the message data are used for this operation. Start and stop bits and the parity bit - if one is selected - are not included in this operation.

Once the first byte has been exclusive OR'ed with the register contents, the least significant bit is removed, and the register contents are shifted towards the least significant bit position, with a zero placed in to the vacant position of the most significant bit.

The least significant bit which is extracted is examined and, depending on it's value, one of two operations is carried out. If the extracted LSB is a '1', the register is exclusive OR'ed with the hex. value 'A0 01'. If the LSB is a '0', the register contents are left as they are and shifted again. This shifting and extracting process is repeated until eight shifts have occurred.

The second data byte is exclusive OR'ed with the register contents and the process of shifting, examining and exclusive OR'ing is repeated another eight times. This process continues until all the data bytes have been through the operation. The resulting 16-bit CRC is transmitted as two 8-bit bytes, with the high order bytes transmitted first.

The MTL Instruments Group plc
Measurement Technology Limited
Power Court, Luton, Bedfordshire
England LU1 3JJ
Tel: +44 (0)1582 723633 Fax: +44 (0)1582 422283
E-mail: enquiry@mtl-inst.com

MTL Incorporated
9 Merrill Industrial Drive, Hampton, NH 03842
USA
Tel: +1 800 835 7075, +1 603 926 0090 Fax: +1 603 926 1899
Or
8576 Wellington Road
Manassas VA 20109
USA
Tel: +1 703 361 0111 Fax: +1 703 368 1029
E-mail: info@mtlnh.com

MTL Canada Safety Instrumentation
20 Regan Road, Unit 17
Brampton, Ontario L7A 1C3
Canada
Tel: +1 905 840 7850 Fax: +1 905 840 7852
E-mail: cinfo@mtlnh.com

MTL Instruments Limited
MTL Systems Pvt Limited
No.3 Old Mahabalipuram Road
Sholinganallur, Chennai (Madras) 600119
India
Tel: +91 (0)44 496 0552 Telefax: +91 (0)44 496 0477 Fax: +91 (0)44 496 1657
E-mail: mds.engg@mtlindia.sprintrpg.ems.vsnl.net.in

MTL Instruments Pty Limited
9 Vinnicombe Drive
PO Box 1441, Canning Vale
Western Australia 6155
Tel: +61 (0)8 9455 2994 Fax: +61 (0)8 9455 2805
E-mail: enquiry@mtlaus.com.au

MTL Instruments Pte Limited
150 Kampong Ampat
#06-01 KA Centre
Singapore 368324
Tel: +65 487 7887 Fax: +65 487 7997
E-mail: sales@mtlsing.com.sg

MTL Instruments KK
10th Floor, Takanawa Meiko Building
2-15-19 Takanawa, Minato-ku, Tokyo 108
Tel: +81 (0)3 5420 1281 Fax: +81 (0)3 5420 2405
E-mail: sales@mtlkk.co.jp

MTL Instruments GmbH
Hellersbergstrasse 2, D-41460 Neuss
Germany
Tel: +49 (0)2131 168016 Fax: +49 (0)2131 168652
E-mail: info@mtl.de

MTL Instruments sarl
Bâtiment SILIC 4, 1 rue des Vergers, 69760 Limonest
France
Tel: +33 (0)4 78 64 98 32 Fax: +33 (0)4 78 35 79 41
E-mail: info@mtl-inst.fr

MTL Instruments BV
MTL Systems BV
de Houtakker 33, 6681 CW Bemmel, The Netherlands
Tel: +31 (0)48 1450250 Fax: +31 (0)48 1450260
E-mail: info@mtlbenelux.com

MTL Instruments BVBA
Derbystraat 379 Blok G
9051 Sint-Denijs-Westrem, Gent
Belgium
Tel: +32 (0)9 242 8844 Fax: +32 (0)9 242 8868
E-mail: info@mtlbenelux.com

Group web site: http://www.mtl-inst.com

Members of The MTL Instruments Group plc

Printed in EU 02/99