modbus tools

For test, simulation and programming.

Protocol Description

MODBUS© Protocol is a messaging structure, widely used to establish master-slave communication between intelligent devices. A MODBUS message sent from a master to a slave contains the address of the slave, the 'command' (e.g. 'read register' or 'write register'), the data, and a check sum (LRC or CRC).
Since Modbus protocol is just a messaging structure, it is independent of the underlying physical layer. It is traditionally implemented using RS232, RS422, or RS485

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

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

Controllers can be setup to communicate on standard Modbus networks using either of two transmission modes: ASCII or RTU.

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

> **Coding System**
> Hexadecimal ASCII printable characters 0 ... 9, A ... F
> **Bits per Byte**
> 1 start bit
> 7 data bits, least significant bit sent first
> 1 bit for even / odd parity-no bit for no parity
> 1 stop bit if parity is used-2 bits if no parity
> **Error Checking**
> Longitudinal Redundancy Check (LRC)

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

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

In ASCII mode, messages start with a colon (:) character (ASCII 3A hex), and end with a carriage return-line feed (CRLF) pair (ASCII 0D and 0A hex).
The allowable characters transmitted for all other fields are hexadecimal 0 ... 9, A ... F.
Networked devices monitor the network bus continuously for the colon character. When one is received, each device decodes the next field (the address field) to find out if it is the addressed device.
Intervals of up to one second can elapse between characters within the message. If a greater

interval occurs, the receiving device assumes an error has occurred. A typical message frame is shown below.

Start	Address	Function	Data	LRC	End
:	2 Chars	2 Chars	N Chars	2 Chars	CR LF

RTU Framing
In RTU mode, messages start with a silent interval of at least 3.5 character times. This is most easily implemented as a multiple of character times at the baud rate that is being used on the network (shown as T1-T2-T3-T4 in the figure below). The first field then transmitted is the device address.
The allowable characters transmitted for all fields are hexadecimal 0 ... 9, A ... F. Networked devices monitor the network bus continuously, including during the silent intervals. When the first field (the address field) is received, each device decodes it to find out if it is the addressed device.
Following the last transmitted character, a similar interval of at least 3.5 character times marks the end of the message. A new message can begin after this interval.
The entire message frame must be transmitted as a continuous stream. If a silent interval of more than 1.5 character times occurs before completion of the frame, the receiving device flushes the incomplete message and assumes that the next byte will be the address field of a new message.
Similarly, if a new message begins earlier than 3.5 character times following a previous message, the receiving device will consider it a continuation of the previous message. This will set an error, as the value in the final CRC field will not be valid for the combined messages. A typical message frame is shown below.

Start	Address	Function	Data	CRC	End
3.5 Char time	8 Bit	8 Bit	N * 8Bit	16 Bit	3.5 Char time

Address Field
The address field of a message frame contains two characters (ASCII) or eight bits (RTU). The individual slave devices are assigned addresses in the range of 1 ... 247.

Function Field

The Function Code field tells the addressed slave what function to perform.
The following functions are supported by Modbus poll

01 READ COIL STATUS
02 READ INPUT STATUS
03 READ HOLDING REGISTERS
04 READ INPUT REGISTERS
05 WRITE SINGLE COIL
06 WRITE SINGLE REGISTER
15 WRITE MULTIPLE COILS
16 WRITE MULTIPLE REGISTERS

The data field contains the requested or send data.

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

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

RTU
When RTU mode is used for character framing, the error-checking field contains a 16-bit value implemented as two eight-bit bytes. The error check value is the result of a Cyclical Redundancy Check calculation performed on the message contents.
The CRC field is appended to the message as the last field in the message. When this is done, the low-order byte of the field is appended first, followed by the high-order byte. The CRC high-order byte is the last byte to be sent in the message.
CRC Example Code

Function 01 (01hex) Read Coils

Reads the ON/OFF status of discrete coils in the slave.

Request
The request message specifies the starting coil and quantity of coils to be read.

Example of a request to read 10...22 (Coil 11 to 23) from slave device address 4:

Field Name	RTU (hex)	ASCII Characters
Header	None	: (Colon)
Slave Address	04	0 4
Function	01	0 1
Starting Address Hi	00	0 0
Starting Address Lo	0A	0 A
Quantity of Coils Hi	00	0 0
Quantity of Coils Lo	0D	0 D
Error Check Lo	DD	LRC (E 4)
Error Check Hi	98	
Trailer	None	CR LF
Total Bytes	8	17

Response
The coil status response message is packed as one coil per bit of the data field. Status is indicated as: 1 is the value ON, and 0 is the value OFF. The LSB of the first data byte contains the coil addressed in the request. 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 zeroes (toward the high-order end of the byte). The byte count field specifies the quantity of complete bytes of data.

Example of a response to the request:

Field Name	RTU (hex)	ASCII Characters
Header	None	: (Colon)
Slave Address	04	0 4
Function	01	0 1
Byte Count	02	0 2
Data (Coils 7...10)	0A	0 A
Data (Coils 27...20)	11	1 1
Error Check Lo	B3	LRC (D E)
Error Check Hi	50	None
Trailer	None	CR LF
Total Bytes	7	15

Function 02(02hex) Read Discrete Inputs

Reads the ON/OFF status of discrete inputs in the slave.

Request
The request message specifies the starting input and quantity of inputs to be read.

Example of a request to read 10...22 (input 10011 to 10023) from slave device address 4:

Field Name	RTU (hex)	ASCII Characters
Header	None	: (Colon)
Slave Address	04	0 4
Function	02	0 2
Starting Address Hi	00	0 0
Starting Address Lo	0A	0 A
Quantity of inputs Hi	00	0 0
Quantity of inputs Lo	0D	0 D
Error Check Lo	99	LRC (E 3)
Error Check Hi	98	
Trailer	None	CR LF
Total Bytes	8	17

Response
The input status response message is packed as one input per bit of the data field. Status is indicated as: 1 is the value ON, and 0 is the value OFF. The LSB of the first data byte contains the input addressed in the request. 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 zeroes (toward the high-order end of the byte). The byte count field specifies the quantity of complete bytes of data.

Example of a response to the request:

Field Name	RTU (hex)	ASCII Characters
Header	None	: (Colon)
Slave Address	04	0 4
Function	02	0 2
Byte Count	02	0 2
Data (Inputs 17...10)	0A	0 A
Data (Inputs 27...20)	11	1 1
Error Check Lo	B3	LRC (D D)
Error Check Hi	14	None
Trailer	None	CR LF
Total Bytes	7	15

Function 03 (03hex) Read Holding Registers

Read the binary contents of holding registers in the slave.

Request
The request message specifies the starting register and quantity of registers to be read.

Example of a request to read 0...1 (register 40001 to 40002) from slave device 1:

Field Name	RTU (hex)	ASCII Characters
Header	None	: (Colon)
Slave Address	01	0 1
Function	03	0 3
Starting Address Hi	00	0 0
Starting Address Lo	00	0 0
Quantity of Registers Hi	00	0 0
Quantity of Registers Lo	02	0 2
Error Check Lo	C4	LRC (F A)
Error Check Hi	0B	
Trailer	None	CR LF
Total Bytes	8	17

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.

Example of a response to the request:

Field Name	RTU (hex)	ASCII Characters
Header	None	: (Colon)
Slave Address	01	0 1
Function	03	0 3
Byte Count	04	0 4
Data Hi	00	0 0
Data Lo	06	0 6
Data Hi	00	0 0
Data Lo	05	0 5
Error Check Lo	DA	LRC (E D)
Error Check Hi	31	None
Trailer	None	CR LF
Total Bytes	9	19

Function 04 (04hex) Read Input Registers

Read the binary contents of input registers in the slave.

Request
The request message specifies the starting register and quantity of registers to be read.

Example of a request to read 0...1 (register 30001 to 30002) from slave device 1:

Field Name	RTU (hex)	ASCII Characters
Header	None	: (Colon)
Slave Address	01	0 1
Function	04	0
Starting Address Hi	00	0 0
Starting Address Lo	00	0 0

Quantity of Registers Hi	00	0 0
Quantity of Registers Lo	02	0 2
Error Check Lo	71	LRC (F 9)
Error Check Hi	CB	
Trailer	None	CR LF
Total Bytes	8	17

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.

Example of a response to the request:

Field Name	**RTU (hex)**	**ASCII Characters**
Header	None	: (Colon)
Slave Address	01	0 1
Function	04	0 4
Byte Count	04	0 4
Data Hi	00	0 0
Data Lo	06	0 6
Data Hi	00	0 0
Data Lo	05	0 5
Error Check Lo	DB	LRC (E C)
Error Check Hi	86	None
Trailer	None	CR LF
Total Bytes	9	19

Function 05 (05hex) Write Single Coil

Writes a single coil to either ON or OFF.

Request
The request message specifies the coil reference to be written. Coils are addressed starting at zero-coil 1 is addressed as 0.

The requested ON / OFF state is specified by a constant in the request 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 write coil 173 ON in slave device 17:

Field Name	**RTU (hex)**	**ASCII Characters**
Header	None	: (Colon)
Slave Address	11	1 1
Function	05	0 5
Coil Address Hi	00	0 0
Coil Address Lo	AC	A C
Write Data Hi	FF	0 0
Write Data Lo	00	F F
Error Check Lo	4E	LRC (3 F)
Error Check Hi	8B	
Trailer	None	CR LF
Total Bytes	8	17

Response
The normal response is an echo of the request, returned after the coil state has been written.
Example of a response to the request:

Field Name	**RTU (hex)**	**ASCII Characters**
Header	None	: (Colon)
Slave Address	11	1 1
Function	05	0 5
Coil Address Hi	00	0 0
Coil Address Lo	AC	A C
Write Data Hi	FF	0 0
Write Data Lo	00	F F
Error Check Lo	4E	LRC (3 F)
Error Check Hi	8B	
Trailer	None	CR LF

Total Bytes	8	17

Function 06 (06hex) Write Single Register

Writes a value into a single holding register.

Request
The request message specifies the register reference to be Written. Registers are addressed starting at zero-register 1 is addressed as 0.

The requested Write value is specified in the request data field. Here is an example of a request to Write register 40002 to 00 03 hex in slave device 17.

Field Name	RTU (hex)	ASCII Characters
Header	None	: (Colon)
Slave Address	11	1 1
Function	06	0 6
Register Address Hi	00	0 0
Register Address Lo	01	0 1
Write Data Hi	00	0 0
Write Data Lo	03	0 3
Error Check Lo	9A	LRC (E 5)
Error Check Hi	9B	
Trailer	None	CR LF
Total Bytes	8	17

Response
The normal response is an echo of the request, returned after the register contents have been written.

Field Name	RTU (hex)	ASCII Characters
Header	None	: (Colon)
Slave Address	11	1 1
Function	06	0 6
Coil Address Hi	00	0 0
Coil Address Lo	01	0 1
Write Data Hi	00	0 0
Write Data Lo	03	0 3
Error Check Lo	9A	LRC (E 5)
Error Check Hi	9B	
Trailer	None	CR LF
Total Bytes	8	17

Function 15 (0Fhex) Write Multiple Coils

Writes each coil in a sequence of coils to either ON or OFF.

Request
The request message specifies the coil references to be written. Coils are addressed starting at zero-coil 1 is addressed as 0.

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

Below is an example of a request to write a series of ten coils starting at coil 20 (addressed as 19, or 13 hex) in slave device 17.

The request 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 and 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.

Field Name	RTU (hex)	ASCII Characters

Header	None	: (Colon)
Slave Address	11	1 1
Function	0F	0 F
Coil Address Hi	00	0 0
Coil Address Lo	13	1 3
Quantity of Coils Hi	00	0 0
Quantity of Coils Lo	0A	0 A
Byte Count	02	0 2
Write Data Hi	CD	C D
Write Data Lo	01	0 1
Error Check Lo	BF	LRC (F 3)
Error Check Hi	0B	
Trailer	None	CR LF
Total Bytes	11	23

Response
The normal response returns the slave address, function code, starting address, and number of coils written. Here is an example of a response to the request shown above

Field Name	**RTU (hex)**	**ASCII Characters**
Header	None	: (Colon)
Slave Address	11	1 1
Function	0F	0 F
Coil Address Hi	00	0 0
Coil Address Lo	13	1 3
Quantity of Coils Hi	00	0 0
Quantity of Coils Lo	0A	0 A
Error Check Lo	26	LRC (C 3)
Error Check Hi	99	
Trailer	None	CR LF
Total Bytes	8	17

Function 16 (10hex) Write Multiple Registers

Writes values into a sequence of holding registers

Request
The request message specifies the register references to be written. Registers are addressed starting at zero-register 1 is addressed as 0.

The requested write values are specified in the request data field. Data is packed as two bytes per register.

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

Field Name	**RTU (hex)**	**ASCII Characters**
Header	None	: (Colon)
Slave Address	11	1 1
Function	10	1 0
Starting Address Hi	00	0 0
Starting Address Lo	01	0 1
Quantity of Registers Hi	00	0 0
Quantity of Registers Lo	02	0 2
Byte Count	04	0 4
Data Hi	00	0 0
Data Lo	0A	0 A
Data Hi	01	0 1
Data Lo	02	0 2
Error Check Lo	C6	LRC (C B)
Error Check Hi	F0	
Trailer	None	CR LF
Total Bytes	13	23

Response
The normal response returns the slave address, function code, starting address, and quantity of registers written. Here is an example of a response to the request shown above.

Field Name	RTU (hex)	ASCII Characters
Header	None	: (Colon)
Slave Address	11	1 1
Function	10	1 0
Starting Address Hi	00	0 0
Starting Address Lo	01	0 1
Quantity of Registers Hi	00	0 0
Quantity of Registers Lo	02	0 2
Error Check Lo	12	LRC (D C)
Error Check Hi	98	
Trailer	None	CR LF
Total Bytes	8	17

LRC Example Code

This function is an example how to calculate a LRC BYTE using the C language.

```
BYTE LRC (BYTE *nData, WORD wLength)
{
BYTE nLRC = 0 ; // LRC char initialized

for (int i = 0; i < wLength; i++)
nLRC += *nData++;

return (BYTE)(-nLRC);

} // End: LRC
```

CRC Example Code

This function is an example how to calculate a CRC word using the C language.

```
WORD CRC16 (const BYTE *nData, WORD wLength)
{
static const WORD wCRCTable[] = {
   0X0000, 0XC0C1, 0XC181, 0X0140, 0XC301, 0X03C0, 0X0280, 0XC241,
   0XC601, 0X06C0, 0X0780, 0XC741, 0X0500, 0XC5C1, 0XC481, 0X0440,
   0XCC01, 0X0CC0, 0X0D80, 0XCD41, 0X0F00, 0XCFC1, 0XCE81, 0X0E40,
   0X0A00, 0XCAC1, 0XCB81, 0X0B40, 0XC901, 0X09C0, 0X0880, 0XC841,
   0XD801, 0X18C0, 0X1980, 0XD941, 0X1B00, 0XDBC1, 0XDA81, 0X1A40,
   0X1E00, 0XDEC1, 0XDF81, 0X1F40, 0XDD01, 0X1DC0, 0X1C80, 0XDC41,
   0X1400, 0XD4C1, 0XD581, 0X1540, 0XD701, 0X17C0, 0X1680, 0XD641,
   0XD201, 0X12C0, 0X1380, 0XD341, 0X1100, 0XD1C1, 0XD081, 0X1040,
   0XF001, 0X30C0, 0X3180, 0XF141, 0X3300, 0XF3C1, 0XF281, 0X3240,
   0X3600, 0XF6C1, 0XF781, 0X3740, 0XF501, 0X35C0, 0X3480, 0XF441,
   0X3C00, 0XFCC1, 0XFD81, 0X3D40, 0XFF01, 0X3FC0, 0X3E80, 0XFE41,
   0XFA01, 0X3AC0, 0X3B80, 0XFB41, 0X3900, 0XF9C1, 0XF881, 0X3840,
   0X2800, 0XE8C1, 0XE981, 0X2940, 0XEB01, 0X2BC0, 0X2A80, 0XEA41,
   0XEE01, 0X2EC0, 0X2F80, 0XEF41, 0X2D00, 0XEDC1, 0XEC81, 0X2C40,
   0XE401, 0X24C0, 0X2580, 0XE541, 0X2700, 0XE7C1, 0XE681, 0X2640,
   0X2200, 0XE2C1, 0XE381, 0X2340, 0XE101, 0X21C0, 0X2080, 0XE041,
   0XA001, 0X60C0, 0X6180, 0XA141, 0X6300, 0XA3C1, 0XA281, 0X6240,
   0X6600, 0XA6C1, 0XA781, 0X6740, 0XA501, 0X65C0, 0X6480, 0XA441,
   0X6C00, 0XACC1, 0XAD81, 0X6D40, 0XAF01, 0X6FC0, 0X6E80, 0XAE41,
   0XAA01, 0X6AC0, 0X6B80, 0XAB41, 0X6900, 0XA9C1, 0XA881, 0X6840,
   0X7800, 0XB8C1, 0XB981, 0X7940, 0XBB01, 0X7BC0, 0X7A80, 0XBA41,
   0XBE01, 0X7EC0, 0X7F80, 0XBF41, 0X7D00, 0XBDC1, 0XBC81, 0X7C40,
   0XB401, 0X74C0, 0X7580, 0XB541, 0X7700, 0XB7C1, 0XB681, 0X7640,
   0X7200, 0XB2C1, 0XB381, 0X7340, 0XB101, 0X71C0, 0X7080, 0XB041,
   0X5000, 0X90C1, 0X9181, 0X5140, 0X9301, 0X53C0, 0X5280, 0X9241,
   0X9601, 0X56C0, 0X5780, 0X9741, 0X5500, 0X95C1, 0X9481, 0X5440,
   0X9C01, 0X5CC0, 0X5D80, 0X9D41, 0X5F00, 0X9FC1, 0X9E81, 0X5E40,
   0X5A00, 0X9AC1, 0X9B81, 0X5B40, 0X9901, 0X59C0, 0X5880, 0X9841,
   0X8801, 0X48C0, 0X4980, 0X8941, 0X4B00, 0X8BC1, 0X8A81, 0X4A40,
   0X4E00, 0X8EC1, 0X8F81, 0X4F40, 0X8D01, 0X4DC0, 0X4C80, 0X8C41,
   0X4400, 0X84C1, 0X8581, 0X4540, 0X8701, 0X47C0, 0X4680, 0X8641,
   0X8201, 0X42C0, 0X4380, 0X8341, 0X4100, 0X81C1, 0X8081, 0X4040 };

BYTE nTemp;
WORD wCRCWord = 0xFFFF;

   while (wLength--)
   {
      nTemp = *nData++ ^ wCRCWord;
```

```
        wCRCWord >>= 8;
        wCRCWord  ^= wCRCTable[nTemp];
    }
    return wCRCWord;

} // End: CRC16
```