EMM-dc series

EMM hc series MULTIMETERS MODBUS-RTU COMMUNICATION PROTOCOL

MODBUS PROTOCOL

Modbus is a master-slave communication protocol able to support up to 247 slaves organized as a bus or as a star network;
The physical link layer can be RS232 for a point to point connection or RS485 for a network.
The communication is half-duplex.
The network messages can be Query-Response or Broadcast type.
The Query-Response command is transmitted from the Master to an established Slave and generally it is followed by an answering message.
The Broadcast command is transmitted from the Master to all Slaves and is never followed by an answer.

MODBUS use two modes for transmission.
A) ASCII Mode: uses a limited character set as a whole for the communication.

B) RTU Mode: binary, with time frame synchronization, faster than the ASCII Mode, uses half so long data block than the ASCII Mode.

EMM analyzers employ RTU mode.

GENERIC MESSAGE STRUCTURE:

START OF FRAME	ADDRESS FIELD	FUNCTION CODE	DATA FIELD	ERROR CHECK	END OF FRAME

START OF FRAME = Starting message marker
ADDRESS FIELD = Includes device address in which you need to communicate in Query-Response mode. In case the message is a Broadcast type it includes 00.
FUNCTION CODE = Includes the operation code that you need to perform.
DATA FIELD = Includes the data field.
ERROR CHECK = Field for the error correction code.
END OF FRAME = End message marker.

Mode RTU communication frame structure :

START OF FRAME = silence on line for time >=4 characters
ADDRES FIELD = 1 character
FUNCTION CODE = 1 character
DATA FIELD = N characters
ERROR CHECK = 16 bit CRC
END OF FRAME = silence on line for time >=4 characters

Wait time for response :
- typical : 150 mS
- worst case : 300 mS.

CRC GENERATION

Example of the CRC-16 generation with "C" language:

```
static unsigned char auchCRCHi [ ] = {
0x00,  0xC1,  0x81,  0x40,  0x01,  0xC0,  0x80,  0x41,  0x01,  0xC0,  0x80,  0x41,  0x00,  0xC1,  0x81,
0x40,  0x01,  0xC0,  0x80,  0x41,  0x00,  0xC1,  0x81,  0x40,  0x00,  0xC1,  0x81,  0x40,  0x01,  0xC0,
0x80,  0x41,  0x01,  0xC0,  0x80,  0x41,  0x00,  0xC1,  0x81,  0x40,  0x00,  0xC1,  0x81,  0x40,  0x01,
0xC0,  0x80,  0x41,  0x00,  0xC1,  0x81,  0x40,  0x01,  0xC0,  0x80,  0x41,  0x01,  0xC0,  0x80,  0x41,
0x00,  0xC1,  0x81,  0x40,  0x01,  0xC0,  0x80,  0x41,  0x00,  0xC1,  0x81,  0x40,  0x00,  0xC1,  0x81,
0x40,  0x01,  0xC0,  0x80,  0x41,  0x00,  0xC1,  0x81,  0x40,  0x01,  0xC0,  0x80,  0x41,  0x01,  0xC0,
0x80,  0x41,  0x00,  0xC1,  0x81,  0x40,  0x00,  0xC1,  0x81,  0x40,  0x01,  0xC0,  0x80,  0x41,  0x01,
0xC0,  0x80,  0x41,  0x00,  0xC1,  0x81,  0x40,  0x01,  0xC0,  0x80,  0x41,  0x00,  0xC1,  0x81,  0x40,
0x00,  0xC1,  0x81,  0x40,  0x01,  0xC0,  0x80,  0x41,  0x01,  0xC0,  0x80,  0x41,  0x00,  0xC1,  0x81,
0x40,  0x01,  0xC1,  0x81,  0x40,  0x01,  0xC0,  0x80,  0x41,  0x00,  0xC1,  0x81,  0x40,  0x01,  0xC0,
0x80,  0x41,  0x01,  0xC0,  0x80,  0x41,  0x00,  0xC1,  0x81,  0x40,  0x00,  0xC1,  0x81,  0x40,  0x01,
0xC0,  0x80,  0x41,  0x01,  0xC0,  0x80,  0x41,  0x00,  0xC1,  0x81,  0x40,  0x01,  0xC0,  0x80,  0x41,
0x00,  0xC1,  0x81,  0x40,  0x00,  0xC1,  0x81,  0x40,  0x01,  0xC0,  0x80,  0x41,  0x00,  0xC1,  0x81,
0x40,  0x01,  0xC0,  0x80,  0x41,  0x01,  0xC0,  0x80,  0x41,  0x00,  0xC1  0x81,  0x40,  0x01,  0xC0,
0x80,  0x41,  0x00,  0xC1,  0x81,  0x40,  0x00,  0xC1,  0x81,  0x40,  0x01,  0xC0,  0x80,  0x41,  0x01,
0xC0,  0x80,  0x41,  0x00,  0xC1,  0x81,  0x40,  0x00,  0xC1,  0x81,  0x40,  0x01,  0xC0,  0x80,  0x41,
0x00,  0xC1,  0x81,  0x40,  0x01,  0xC0,  0x80,  0x41,  0x01,  0xC0,  0x80,  0x41,  0x00,  0xC1,  0x81,
0x40
} ;

static unsigned char auchCRCLo [ ] = {
0x00,  0xC0,  0xC1,  0x01,  0xC3,  0x03,  0x02,  0xC2,  0xC6,  0x06,  0x07,  0xC7,  0x05,  0xC5,  0xC4,
0x04,  0xCC,  0x0C,  0x0D,  0xCD,  0x0F,  0xCF,  0xCE,  0x0E,  0x0A,  0xCA,  0xCB,  0x0B,  0xC9,  0x09,
0x08,  0xC8,  0xD8,  0x18,  0x19,  0xD9,  0x1B,  0xDB,  0xDA,  0x1A,  0x1E,  0xDE,  0xDF,  0x1F,  0xDD,
0x1D,  0x1C,  0xDC,  0x14,  0xD4,  0xD5,  0x15,  0xD7,  0x17,  0x16,  0xD6,  0xD2,  0x12,  0x13,  0xD3,
0x11,  0xD1,  0xD0,  0x10,  0xF0,  0x30,  0x31,  0xF1,  0x33,  0xF3,  0xF2,  0x32,  0x36,  0xF6,  0xF7,
0x37,  0xF5,  0x35,  0x34,  0xF4,  0x3C,  0xFC,  0xFD,  0x3D,  0xFF,  0x3F,  0x3E,  0xFE,  0xFA,  0x3A,
0x3B,  0xFB,  0x39,  0xF9,  0xF8,  0x38,  0x28,  0xE8,  0xE9,  0x29,  0xEB,  0x2B,  0x2A  0xEA,  0xEE,
0x2E,  0x2F,  0xEF,  0x2D,  0xED,  0xEC,  0X2C,  0xE4,  0x24,  0x25,  0xE5,  0x27,  0xE7,  0xE6,  0x26,
0x22,  0xE2,  0xE3,  0x23,  0xE1,  0x21,  0x20,  0xE0,  0xA0,  0x60,  0x61,  0xA1,  0x63,  0xA3,  0xA2,
0x62,  0x66,  0xA6,  0xA7,  0x67,  0xA5,  0x65,  0x64,  0xA4,  0x6C,  0xAC,  0xAD,  0x6D,  0xAF,  0x6F,
0x6E,  0xAE,  0xAA,  0x6A,  0x6B,  0xAB,  0x69,  0xA9,  0xA8,  0x68,  0x78,  0xB8,  0xB9,  0x79,  0xBB,
0x7B,  0x7A,  0xBA,  0xBE,  0x7E,  0x7F,  0xBF,  0x7D,  0xBD,  0xBC,  0x7C,  0xB4,  0x74,  0x75,  0xB5,
0x77,  0xB7,  0xB6,  0x76,  0x72,  0xB2,  0xB3,  0x73,  0xB1,  0x71,  0x70,  0xB0,  0x50,  0x90,  0x91,
0x51,  0x93,  0x53,  0x52,  0x92,  0x96,  0x56,  0x57,  0x97,  0x55,  0x95,  0x94,  0x54,  0x9C,  0x5C,
0x5D,  0x9D,  0x5F,  0x9F,  0x9E,  0x5E,  0x5A,  0x9A,  0x9B,  0x5B,  0x99,  0x59,  0x58,  0x98,  0x88,
0x48,  0x49,  0x89,  0x4B,  0x8B,  0x8A,  0x4A,  0x4E,  0x8E,  0x8F,  0x4F,  0x8D,  0x4D,  0x4C,  0x8C,
0x44,  0x84,  0x85,  0x45,  0x87,  0x47,  0x46,  0x86,  0x82,  0x42,  0x43,  0x83,  0x41,  0x81,  0x80,
0x40
}

unsigned short CRC16 (ptMsg, usDataLen)
unsigned char *ptMsg;                    / * message to calculate CRC upon * /
unsigned short usDataLen;                  / * number of bytes in message * /
{
      unsigned char uchCRCHi = 0xFF;             / * CRC high byte * /
      unsigned char uchCRCLo =0xFF;              / * CRC low byte * /
      unsigned uIndex;

      while (usDataLen--)                  / * pass through message buffer * /
      {
         uIndex = uchCRCHi ^ *ptMsg++;           /.* calculate the CRC * /
         uchCRCHi = uchCRCLo ^ auchCRCHi [ uIndex ] ;
         uchCRCLo = auchCRCLo [ uIndex ]
      }
      return (uchCRCHi « 8 | uchCRCLo ) ;
}
```

Note: The "Error Check (CRC)" field must be computed referring to the characters from the first of ADDR to the last of DATA inclusive.

READING OF THE REGISTERS (Function Code $ 03)

Reads the binary contents of holding registers (2X references) in the slave.
Broadcast is not supported.
The Query message specified the starting register and quantity of register to be read.

QUERY:

START OF FRAME	ADDRESS FIELD	FUNCTION CODE	START ADDRESS	No. OF REGISTERS	ERROR CHECK	END OF FRAME

START OF FRAME = Starting message marker.
ADDRESS FIELD = EMM device address (01…F7 HEX) (1 byte).
FUNCTION CODE = Operation code (03 HEX) (1 byte).
START ADDRESS = First register address to be read (2 byte).
No. OF REGISTERS = Number of registers (max 32) to be read (4 bytes for 1 measure value).
ERROR CHECK = Check sum.
END OF FRAME = End message marker.

WARNING:
It is possible to read more than one variable at the same time (max 16) only if their addresses are consecutive and the variables on the same line cannot be divided.

The register data in the response message are packet 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.

RESPONSE:

START OF FRAME	ADDRESS FIELD	FUNCTION CODE	No. OF BYTES	D0, D1, …, Dn	ERROR CHECK	END OF FRAME

START OF FRAME = Starting message marker.
ADDRESS FIELD = EMM device address (01…F7 HEX) (1byte).
FUNCTION CODE = Operation code (03 HEX) (1 Byte).
No. OF SEND BYTES = Number of data bytes (00…?? HEX) (1 byte). 1 register requires 2 data bytes.
D0, D1, .., Dn = data bytes (00…?? HEX) (Nr. Of register x 2 = n. byte).
ERROR CHECK = Check sum.
END OF FRAME = End message marker .

See the TABLE OF EMM REGISTERS and the EXAMPLE.

SETUP OF THE EMM PARAMETERS (Function Code $ 10)

Write values into a sequence of holding registers (2X references).

WARNING: It is possible to write more than one variable at the same time only if their addresses are consecutive and the variables on the same line cannot be divided. (max of 4 consecutive register on the same message).

QUERY:

START OF FRAME	ADDRESS FIELD	FUNCTION CODE	START ADDRESS	No. OF REGISTERS	No. OF BYTES	D0, D1, …, Dn	ERROR CHECK	END OF FRAME

START OF FRAME = Starting message marker.
ADDRESS FIELD = EMM device address (01…F7 HEX) (1 byte).
FUNCTION CODE = Operation code (10 HEX) (1 byte).
START ADDRESS = First register address to be written (2 byte).
No. OF REGISTER = Number of registers to be written (1 to 4,…) (2 byte).
No. OF BYTES = Number of data bytes (HEX) (1 byte): 1register requires 2 data bytes.
D0,D1,..,Dn = Data bytes (00…? HEX) (1 byte) (Nr. Of register x 2 = n. byte).
ERROR CHECK = Check sum.
END OF FRAME = End message marker.

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

RESPONSE:

START OF FRAME	ADDRESS FIELD	FUNCTION CODE	START ADDRESS	No. OF REGISTERS	ERROR CHECK	END OF FRAME

START OF FRAME = Starting message marker.
ADDRESS FIELD = EMM device address (01…F7 HEX) (1 byte).
FUNCTION CODE = Operation code (10 HEX) (1 byte).
START ADDRESS = First register address to be written (2 byte).
No. OF REGISTER = Number of registers to be written (2 byte).
ERROR CHECK = Check sum.
END OF FRAME = End message marker.

See the TABLE OF EMM REGISTERS and the EXAMPLE.

DIAGNOSTIC (Function Code $ 08)

This function provides a test for checking the communication system.
Broadcast is not supported.
The instrument's protocol has only the sub-function 0 of the diagnostics sub-functions set of the standard modbus protocol.
The Query and the Response messages are the following:

QUERY:

START OF FRAME	ADDRESS FIELD	FUNCTION CODE	SUB FUNCTION	DATA	ERROR CHECK	END OF FRAME

START OF FRAME = Starting message marker.
ADDRESS FIELD = EMM device address (01…F7 HEX) (1 byte).
FUNCTION CODE = Operation code (08 HEX) (1 byte).
SUB FUNCTION = Sub-function 0 (00 00 hex) (2 byte).
DATA = Max 10 data bytes.
ERROR CHECK = Check sum.
END OF FRAME = End message marker.

RESPONSE:
The response must be the loopback of the same data.

START OF FRAME	ADDRESS FIELD	FUNCTION CODE	SUB FUNCTION	DATA	ERROR CHECK	END OF FRAME

START OF FRAME = Starting message marker.
ADDRESS FIELD = EMM device address (01…F7 HEX) (1 byte).
FUNCTION CODE = Operation code (08 HEX) (1 byte).
SUB FUNCTION = Sub-function 0 (00 00 hex) (2 byte).
DATA = Data bytes.
ERROR CHECK = Check sum.
END OF FRAME = End message marker.

DIAGNOSTIC EXAMPLE

QUERY

Field Name	Example (Hex)
Slave Address	01
Function Code	08
Sub-function Hi	00
Sub-function Lo	00
Data Hi	F1
Data Lo	A7
Error Check (CRC)	??
	??

RESPONSE

Field Name	Example (Hex)
Slave Address	01
Function Code	08
Sub-function Hi	00
Sub-function Lo	00
Data Hi	F1
Data Lo	A7
Error Check (CRC)	??
	??

REPORT SLAVE ID (Function Code $ 11)

This function returns the type of the instrument and the current status of the slave run indicator.
Broadcast is not supported.
The Query and the Response messages are the following:

QUERY:

START OF FRAME	ADDRESS FIELD	FUNCTION CODE	ERROR CHECK	END OF FRAME

START OF FRAME = Starting message marker.
ADDRESS FIELD = EMM device address (01…F7 HEX) (1 byte).
FUNCTION CODE = Operation code (11 HEX) (1 byte).
ERROR CHECK = Check sum.
END OF FRAME = End message marker.

RESPONSE:

START OF FRAME	ADDRESS FIELD	FUNCTION CODE	BYTE COUNT	SLAVE ID	RUN INDICATOR STATUS	DATA	ERROR CHECK	END OF FRAME

START OF FRAME = Starting message marker.
ADDRESS FIELD = EMM device address (01…F7 HEX) (1 byte).
FUNCTION CODE = Operation code (11 HEX) (1 byte).
BYTE COUNT = Number of data bytes (16 HEX) (1 byte).
SLAVE ID = Slave ID identifier (50 HEX) (1 byte).
RUN INDICATOR STATUS = Run indicator status (FF HEX) (1 byte).
DATA = Data bytes.
ERROR CHECK = Check sum.
END OF FRAME = End message marker.

The normal response has the slave ID identifier (50 HEX) and the run indicator status (FF HEX) plus 20 data bytes (byte count is 22, 16 Hex). Last four data bytes carry firmware version (bytes 19 ,20) and bit-mapped options installed on EMM (bytes 17,18).

Byte 17 mapped bit (Value = 1: -> option installed):
Bit 0 Pulse output (Energy)
Bit 1 Neutral Current Input
Bit 5 Digital Output for Alarm function.
Bit 7 Double tariff function (Time Bands)
Bit 2,3,4,6 No meaning

Byte 18 mapped bit (Value = 1: -> option installed):
Bit 1 Analog output
Other bits: No meaning

REPORT SLAVE ID EXAMPLE

QUERY

Field Name	Example (Hex)
Slave Address	01
Function Code	11
Error Check (CRC)	??
	??

RESPONSE

Field Name	Example (Hex)
Slave Address	01
Function Code	11
Byte count	02
Slave ID	50
Run indicator status	FF
Data	20 data bytes
Error Check (CRC)	??
	??

ERROR MESSAGE FROM SLAVE TO MASTER

When a slave device receives a not valid query, it does transmit an error message.

RESPONSE:

START OF FRAME	ADDRESS FIELD	FUNCTION CODE	ERROR CODE	ERROR CHECK	END OF FRAME

START OF FRAME = Starting message marker.
ADDRESS FIELD = EMM device address (01…F7 HEX) (1 byte).
FUNCTION CODE = Operation code with bit 7 high (1 byte).
ERROR CODE = Message containing communication failure (1 byte).
ERROR CHECK = Check sum.
END OF FRAME = End message marker.

ERROR EXAMPLE

QUERY

Field Name	Example (Hex)
Slave Address	01
Function Code	03
Starting Address Hi	00
Starting Address Lo	00
Number Of Word Hi	00
Number Of Word Lo	05
Error Check (CRC)	??
	??

RESPONSE

Field Name	Example (Hex)
Slave Address	01
Function Code	83 (1)
Error Code	02 (2)
Error Check (CRC)	??
	??

(1): Function Code transmitted by master with bit 7 high.
(2): Error type:
01 = Illegal Function
02 = Illegal data address
03 = Illegal data value

TABLE OF EMM-dc REGISTERS

The following table shown EMM-dc registers. All registers are 16-bit integer type (signed or unsigned).
Note 1: To correctly reading values, always reads both registers which compose the right value of electrical parameter.
Note 2: Voltages are expressed in *10 units. Phase Voltage [VOLT]=Value($1000 reg)/10
Powers are expressed in *10 units. I.E.: Line Power L1 [WATT]=Value($1008 reg)/10
Energies are expressed in Wh*100 units. Energy [kWh] =Value($102x reg)*10 (only for EMM-4dc)
The hours counter is expressed in *10 units. Hours counted [h] =Value($1046 reg)/10

MEASURED VALUES (Function code $ 03)

Register HEX		Word	Description	M. U.	Type
EMM-4dc	EMM-4d2c				
$1000	$1000	2	PHASE VOLTAGE V_L	[V*10]	(Signed)
$1002	$1002	2	PHASE VOLTAGE V_L	[V*10]	(Signed)
$1004	$1004	2	LINE CURRENT L_1	[mA]	(Signed)
$1006	$1006	2	LINE CURRENT L_2	[mA]	(Signed)
$1008	…	2	LINE POWER L_1	[W*10]	(Signed)
$100A	…	2	LINE POWER L_2	[W*10]	(Signed)
$100C	…	2	CURRENT SUM L_1+L_2	[mA]	(Signed)
$100E	…	2	POWER SUM L_1+L_2	[W*10]	(Signed)
…	…	…	…	…	…
$1044	$1044	2	TEMPERATURE	[°C]	(Unsigned)

ENERGY AND HOURS COUNTERS

MODBUS DATA REGISTERS FOR ENERGY AND HOURS COUNTERS

Register HEX		Word	Description	M. U.	Type
EMM-4dc	EMM-4d2c				
$1020	…	2	LINE POSITIVE / IMPORTED ENERGY L_1	[100*Wh]	(Unsigned)
$1022	…	2	LINE NEGATIVE / EXPORTED ENERGY L_1	[100*Wh]	(Unsigned)
$1024	…	2	LINE POSITIVE / IMPORTED ENERGY L_2	[100*Wh]	(Unsigned)
$1026	…	2	LINE NEGATIVE / EXPORTED ENERGY L_2	[100*Wh]	(Unsigned)
$1028	…	2	LINE POSITIVE / IMPORTED ENERGY SUM L_1+L_2	[100*Wh]	(Unsigned)
$102A	…	2	LINE NEGATIVE / EXPORTED ENERGY SUM L_1+L_2	[100*Wh]	(Unsigned)
…	…	…	…	…	…
$1046	$1046	2	HOURS COUNTER	[hr*10]	(Unsigned)

VALUES STORED IN EEPROM (Function code $03)

Register HEX		Word	Description	M. U.	Type
EMM-4dc	EMM-4d2c				
$1060	$1060	2	MAX INSTANTANEOUS VOLTAGE V_L	[V*10]	(Signed)
$1062	$1062	2	MAX INSTANTANEOUS VOLTAGE V_L	[V*10]	(Signed)
$1064	$1064	2	MAX INSTANTANEOUS CURRENT L_1	[mA]	(Signed)
$1066	…	2	MAX INSTANTANEOUS POWER L_1	[W*10]	(Signed)
$1068	$1068	2	MAX INSTANTANEOUS CURRENT L_2	[mA]	(Signed)
$106A	…	2	MAX INSTANTANEOUS POWER L_2	[W*10]	(Signed)
$106C	…	2	MAX INSTANTANEOUS CURRENT L_1+L_2	[mA]	(Signed)
$106E	…	2	MAX INSTANTANEOUS POWER L_1+L_2	[W*10]	(Signed)
$1070	$1070	2	MAX AVG CURRENT L_1	[mA]	(Signed)
$1072	…	2	MAX AVG POWER L_1	[W*10]	(Signed)
$1074	$1074	2	MAX AVG CURRENT L_2	[mA]	(Signed)
$1076	…	2	MAX AVG POWER L_2	[W*10]	(Signed)
$1078	$1078	2	MAX AVG CURRENT L_1+L_2	[mA]	(Signed)
$107A	…	2	MAX AVG POWER L_1+L_2	[W*10]	(Signed)
$107C	$107C	2	LAST AVG MAX INSTANTANEOUS CURRENT L_1	[mA]	(Signed)
$107E	…	2	LAST AVG MAX INSTANTANEOUS POWER L_1	[W*10]	(Signed)
$1080	$1080	2	LAST AVG MAX INSTANTANEOUS CURRENT L_2	[mA]	(Signed)
$1082	…	2	LAST AVG MAX INSTANTANEOUS POWER L_2	[W*10]	(Signed)
$1084	…	2	LAST AVG MAX INSTANTANEOUS CURRENT L_1+L_2	[mA]	(Signed)
$1086	…	2	LAST AVG POWER L_1+L_2	[W*10]	(Signed)
$1088	$1088	2	MAX PEAK TEMPERATURE	[°C]	(Unsigned)
$108A	$108A	2	MAX AVG TEMPERATURE	[°C]	(Unsigned)
$108C	$108C	2	LAST AVG TEMPERATURE	[°C]	(Unsigned)

NOTE:
- WHEN THE INSTRUMENT CAN'T MEASURE IT SEND 0000 AS VALUE.
- …. means that there are registers not consecutive

TROUBLESHOOTING

If response from EMM doesn't happen:

- check connection from EMM and RS232/RS485 converter ;
- check if data outgoing from the RS232 serial port of the PC come in the RS232/485 converter
- try to increase the wait time for response (300 mS is good);
- check if the transmitted data stream is **EXACTLY** as in example, monitoring the data on the RS485 serial line with a terminal (i.e. Hyperterminal or other emulator);
- if the RS232/485 converter is not our model EMI-1, be sure the turnaround-time is set in range 1 to 2 mS

contrel elettronica srl

ISO 9001:2000 CERTIFICATE N.9105.C035

Contrel elettronica srl - 26900 LODI - ITALY - via S. Fereolo, 9
Tel. +39 0371 30207/30761/35386 Fax. +39 0371 32819 E-mail: contrel@contrel.it
www.contrel.it