BMS_Modbus_RTU communication description

Version: March 2021 Version 1.0

Communication data format

During communication, the data is returned in the form of words (WORD— 2 bytes). In each word returned, the high byte first and the low byte after. If two words are returned continuously (such as long integer), the high word first and the lower word after.

Data type	Number of registers	Number of bytes	Description
Character type	1	1	Send back two characters at a time, if there are less than two, use 0 to supplement
Integer	1	2	Send back at one time, high byte first and low byte last
Long integer	2	4	sent back in two words, the high word first and the low word last

Frame format

Register content query (function code 03H)
Query the frame format sent by the device

Byte Order	Code	Example	Description
0	Device address	01H	Bms address range(1~4),inverter address range(1~247)
1	03H	03H	Function code
2	Start register address high byte	00H	Register address high 8 bits
3	Starting register address low byte	10H	Register address low 8 bits
4	High byte of the number of registers	00H	High 8 bits of the number of registers
5	Low byte of the number of	02H	Low 8 bits of the number of registers

	registers		
6	CRC16 check high byte	C0H	CRC16 check high 8 bits
7	CRC16 check low byte	CBH	CRC16 check low 8 bit

BMS or inverter parsed successfully sent back frame format

Byte order	Code	Description
0	Device address	BMS address range (1~4), inverter address range (1~247)
1	03H	Function code
2	Number of returned data bytes(N)	N = number of registers requested *2
3	High byte of the first register data	
4	Low byte of the first register data	
.....		
.....		
	Nth register data high byte	
	Nth register data low byte	
N+3	CRC16 check high byte	
N+4	CRC16 check low byte	

BMS or inverter parsing error sent back frame format

Byte order	Code	Description
0	Device address	BMS address range (1~4), inverter address range (1~247)
1	03H	Function code
2	Number of returned data bytes(N)	N = number of registers requested *2
3	The first 0	a total of N 0s are returned
4	2nd 0	
.....		
.....		
	N-1th 0	
	Nth 0	
N+3	CRC16 check high byte	
N+4	CRC16 check low byte	

Register read example:

Inverter to read BMS battery voltage, BMS reply 12.0v
Inverter send : 01 03 00 64 00 01 C5 D5
BMS return: 01 03 00 02 00 78 E4 28

BMS register address

R: Indicates that it is read only and supports 03H commands
W: Indicates that it is write only and supports 10H commands
Int: integer; **Long**: long integer; **UInt**: unsigned integer; **ULong**: unsigned long integer; **ASC:** ASCII code
Max: take the maximum value; **Min**: take the minimum value
Support up to 4 BMS modules to connect to the system at the same time, and the modbus address is 1 to 4

The addresses in the following table are all expressed in decimal

Data name	**Unit**	**Format**	**Start address**	**Register number**	**Read**	**Remarks**
Battery Voltage	0.1v	int	100	1	R	
Battery Charge Current	0.1A	int	101	1	R	
Battery discharge Current	0.1A	int	102	1	R	
Total battery capacity	0.1Ah	int	103	1	R	
Battery remaining capacity	0.1Ah	int	104	1	R	
Battery percentage	1%	int	105	1	R	
Battery temperature	0.1℃	Int	106	1	R	
Reserved			107	1	R	
Reserved			108	1	R	
Reserved			109	1	R	
Maximum charging voltage	0.1V	int	110	1	R	
Minimum discharge voltage	0.1V	int	111	1	R	
Maximum charging current	0.1A	Int	112	1	R	
Maximum discharging current	0.1A	int	113	1	R	
Forced charging	0/1	int	114	1	R	
Discharge prohibited	0/1	int	115	1	R	

Charging prohibited	0/1	int	116	1	R	

Inverter register address

R: Indicates that it is read only and supports 03H commands

W: Indicates that it is write only and supports 10H commands

Int: integer; **Long**: long integer; **UInt**: unsigned integer; **ULong**: unsigned long integer; ASC: ASCII code

Max: take the maximum value; Min: take the minimum value

The addresses in the following table are all expressed in decimal

Data name	**Unit**	**Format**	**Start address**	**Register number**	**Read**	**Remarks**
Battery Status		int	760	R	R	See the battery status description for details
Maximum charging voltage	0.1V	int	761	1	R	
Minimum discharge voltage	0.1V	int	762	1	R	
Maximum charging current	0.1A	Int	763	1	R	
Maximum discharging current	0.1A	int	764	1	R	
Reserved			**765**			
Reserved			**767**			
Reserved			**768**			
Reserved			**769**			
Battery Voltage	0.1v	int	770	1	R	
Total battery capacity	0.1Ah	int	771	1	R	
Battery remaining capacity	0.1Ah	int	772	1	R	
Battery percentage	1%	int	773	1	R	
Battery Charge Current	0.1A	Int	774	1	R	
Battery Discharge Current	0.1A	int	775	1	R	

Battery status description

Status bit	Description
bit 0~11	reserved
bit 12	Discharge prohibited

Bit13	Charging prohibited
Bit14	Forced charging
Bit15	Establish connection with BMS

CRC check algorithm

Parameter model: CRC-16/MODBUS ×16+×15+×2+1

C language code

```
const 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, 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, 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
} ;

const 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 sModbusCrc16(INT8U *chMsg , INT16U dataLen)
{
    unsigned char ubCRCHi = 0xFF;
    unsigned char ubCRCLo = 0xFF;
    unsigned char duwIndex;
    while (dataLen --)
    {
        duwIndex = 0xff&(ubCRCHi ^ *chMsg++);
        ubCRCHi = 0xff&(ubCRCLo ^ auchCRCHi[duwIndex]);
        ubCRCLo = auchCRCLo[duwIndex];
    }
    return (ubCRCHi << 8 | ubCRCLo);
}
```