Gas Detector Communication Protocol

This protocol is fully compatible with the MODBUS RTU standard protocol. It is also suitable for other instruments that use serial ports (UART, RS232, RS485) and USB-to-serial function.
Change the gas type table based on V0.0.

1. Serial port settings:

Baud rate: 2400/4800/9600/19200/38400/115200 (generally the default is 9600)
Start bit: 1 bit
Data bits: 8 bits
Parity: 1 bit even parity/no parity
Stop bit: 1 bit

2. Communication protocol:

The communication protocol follows the MODBUS-RTU standard protocol and adopts a question-and-answer communication method. When the communication data of the master (host computer or PC, the same below) is sent to the slave, the slave (detector, the same below) first determines whether the address of the local machine is It is consistent with the address code, and the data is received only when the address code matches (except the broadcast address), until the frame data is stopped, and the check code is checked. If it is correct, the corresponding task is executed, and then the execution result is returned to the host; If an error occurs, an abnormal response will be returned to remind the host to send again;

3. Data format

The data format follows the MODBUS-RTU standard format. Each frame of data is divided into address code, function code, data area and check code, as shown below:

Start	Address code	Function code	Data area	Check code	End
4 byte periods	1 byte	1 byte	N bytes	2 bytes	4 byte periods

1) **Start and end**:

Follow the MODBUS-RTU standard format, there is no specially established start and end bytes, the instrument (including master and slave, the same below) detects the gap time of 4 byte cycles, it is the end of the previous frame of data and the next The start of frame data; for example, when the baud rate is 9600, the period of each byte is about 1 millisecond, and the period of 4 bytes is 4 milliseconds. Taking into account the timing error of the instrument, to ensure normal communication, it is recommended to send the frame interval is more than 5 milliseconds, and the byte interval in the same frame is within 2 milliseconds;

2) Address code:

The address code is one byte, which is the first byte of each data frame, from 0 to 255. The address byte indicates that the slave with the address set by the user will receive the information sent by the master. Each slave must have a unique address code, and only the slaves that match the address code can respond to the echo. When the slave sends back information, the equivalent address code indicates where the information comes from. 0 means the broadcast address, all slaves can receive data.

3) Function code:

The function code is one byte, which is the second byte of each data frame. It will indicate that the master tells the slave to perform the corresponding task; when the read operation (function code 03 or 65), the slave will return a band Message with specified information; when writing operation (function code 06), a successful operation will return a null data (that is, the data length byte is 0) information, if the operation fails, an exception message will be returned.

The function codes supported by this protocol are as follows:

Code	Meaning	Operation
03	Read data register	Read the contents of the specified register
06	Write data register	Modify the contents of the specified register
65	Read history record	Read the history record of the specified number
255	Abnormal response	Only when the slave returns an exception code when it is abnormal

4) Data area:

The data area contains actions to be performed by the slave or information sent back by the slave. For master sending and slave return, the length is not equal. When the master sends, the length of the data area is fixed at 4 bytes. When the slave returns, the length of the data area is variable; the format of the data area is as follows:

Host send (function code 03)

1 byte	2 bytes	3 bytes	4 bytes
Address high byte	Address low byte	Quantity high byte	Quantity low byte

Note: This protocol only supports reading and writing of a single register, so bytes 3 and 4 can only be 0x0001, and other data is invalid.

Host send (function code 06)

1 byte	2 bytes	3-4 bytes	5-6 bytes	7-8 bytes	9-10 bytes
Address high byte	Address low byte	Channel 1 data	Channel 2 data	Channel 3 data	Channel 4 data

Note: The channel data is high byte first and low byte last; character data is filled with 0 in the high byte.

Host send (function code 65)

1 byte	2 bytes	3 bytes	4 bytes
Numbering highest byte	Numbering second highest byte	Numbering second lowest byte	Numbering lowest byte

Return from the machine (general)

1 byte	2 bytes	3 bytes	。。。。。。	N+1 bytes
Data length byte Data1	Data 1	Data 2	。。。。。。	Data N

Return from the machine (function code 65)

Byte	1	2	3	4	5	6	7	8	9	10	11	。。。。。。	N*2+6	N*2+7
Data	Data length	Year	month	day	hour	minute	second	Channel 1		Channel 2		。。。。。。	Channel N	

Note: The data length and time data are of unsigned character type, and the data of all channels are of unsigned integer type, with the high byte first.

The slave returns abnormally (function code 255)

1 byte	2 bytes
Data length (fixed to 0x01)	Exception code (see definition of exception code)

5) Check code:

The master or slave can use the check code to judge whether the received information is wrong. Due to electronic noise or some other interference, the information may change during the transmission process. The error check code ensures that the master or slave does not work on the error information during the transmission. This increases the safety and efficiency of the system. This protocol complies with the MODBUS-RTU standard, and the error check uses 16-bit redundant cyclic code (CRC-16), and the high byte of the data in the CRC code is sent and received first.

4. The attached table

1) Definition of data register address:

Address	Name	Return data type	Data length	Return data content	Remarks
0x02	Total number of channels	Unsigned character type	1 byte	Total number of valid channels of the slave	Read only
0x03	Total number of records	Unsigned long integer	4 bytes	Total number of stored history records	Read only
0x10	Gas type	Unsigned character type	Number of channels* 1	Gas type of all channels	See definition of gas type
0x11	Data unit	Unsigned character type	Number of channels* 1	Data unit of all channels	See gas unit definition
0x12	Data decimal places	Unsigned character type	Number of channels* 1	Data decimal places of all channels	
0x13	Gas range	Unsigned	Number	Gas range of all channels	

		integer	of channels* 2		
0x14	Gas ADC value	Unsigned integer	Number of channels* 2	ADC value of all channels	Read only
0x15	Gas concentration value	Unsigned integer	Number of channels* 2	Concentration values of all channels	Read only
0x16	Low gas value	Unsigned integer	Number of channels* 2	Underreported value of all channels	
0x17	High gas value	Unsigned integer	Number of channels* 2	High value of all channels	

Note: All channel data is channel 1 first, and the high byte is first when the data type is integer.

All data related to gas concentration (including gas range, concentration and alarm value) must be combined with the data unit and data decimal places to get the final value; for example, the concentration value is 1234, the unit value is 4, and the decimal place value is 1. Together it means that the gas concentration value is 123.4mg/m3.

2) Gas type definition:

```
Unsigned char  GASTY[65][10]=
{
      "GAS    ","CO     ","H2S    ","O2     ","EX     ",  //0-4
      "SO2    ","NH3    ","H2     ","N2     ","O3     ",  //5-9
      "TVOC   ","CL2    ","HCL    ","NO     ","NO2    ",  //10-14
      "PH3    ","AsH3   ","HCN    ","CO2    ","SF6    ",  //15-19
      "Br2    ","HBr    ","F2     ","HF     ","N2O    ",  //20-24
      "H2O2   ","NOX    ","SOX    ","Odor   ","VOC    ",  //25-29
      "CH4    ","C2H6   ","C3H8   ","C4H10  ","iC4H10 ",  //30-34
      "C5H12  ","C2H4   ","C3H6   ","C4H8   ","iC4H8  ",  //35-39
      "CH4O   ","C2H6O  ","C3H8O  ","iC3H8O ","C4H10O ",  //40-44
      "CH2O   ","C2H4O  ","C3H6O  ","C3H4O  ","C2H2   ",  //45-49
      "C6H6   ","C7H8   ","C8H10  ","C8H8   ","C6H6O  ",  //50-54
      "ETO    ","C2H8O2 ","NMHC   ","GAS    ","GAS    ",  //55-59
      "U-DEF1 ","U-DEF2 ","U-DEF3 ","U-DEF4 ","GAS    ",  //60-64
};
```

Note: U-DEF is user-defined, which means a self-defined type.

3) Definition of gas unit:

Value	0	1	2	3	4	5+
symbol	PPM	PPB	%VOL	%LEL	Mg/m3	Reserved

4) Exception code definition：

Value	0	1	2	3	4	5	6+
significance	Unknown	Calibration error	Data error	Command error	Instrument busy	Instrument failure	Reserved

Note: Data errors include errors in the data area such as register address error, register number out of range, historical record number out of range, etc.
Command error means that the function code is not within the error range of the slave.

5) CRC code calculation method:
The steps to calculate the CRC code are:
Step 1: Preset the 16-bit register as hexadecimal FFFF (that is, all 1). Call this register CRC register;
Step 2: XOR the first 8-bit data with the low bit of the 16-bit CRC register, and put the result in the CRC register;
Step 3: Shift the contents of the register one bit to the right (toward the low bit), fill the highest bit with 0, and check the lowest bit;
Step 4: If the lowest bit is 0: repeat step 3 (shift again);
If the lowest bit is 1: CRC register is XORed with polynomial 0xA001;
Step 5: Repeat steps 3 and 4 until the right shift is 8 times, so that the entire 8-bit data is processed;
Step 6: Repeat steps 2 to 5 to process the next 8-bit data;
When all the data information is processed, the CRC register finally obtained is the CRC code.
The C language example of CRC code calculation is as follows:

```
//--------------------------------//
// Function name：Get CRC check code
// Function function: calculate the CRC checksum,
//Input variable: pSendBuf-data area address nEnd-data length
//Output variable: wCrc-checksum
//--------------------------------//
unsigned int GetCRC(unsigned char *pSendBuf, unsigned char nEnd)
{
    unsigned char i,j;
    unsigned int wCrc = (unsigned int)(0xffff);
    for(i = 0;i<nEnd;i++)
    {
        wCrc ^= (unsigned int)(pSendBuf[i]);
        for(j = 0;j<8;j++)
        {
            if(wCrc&0x01)
            {
                wCrc >>=1;
```

```
                    wCrc ^= 0xA001;
                }
                else    wCrc >>=1;
            }
        }
        return wCrc;
}
```