GCAN-GT-412

Programmable Smart Gateway

User manual

Document version: V2.0 (2020/06/18)

revise history

version	date	the reason
V1.00	2019/09/11	Create document
V2.00	2020/06/18	Add some parameters

catalogue

1. Function Introduction

1.1 Function Overview

GCAN-GT-412 is a programmable bus gateway/converter. The device integrates 2 CAN bus interfaces, 1 Ethernet bus interface, and 1 RS232 bus interface. Before actual use, users need to write application programs for the device through openPCS software to achieve data between different bus interfaces Of each other.

The GCAN-GT-412 module supports multiple standard communication protocols, such as CANopen, SAE J1939, Modbus TCP, Modbus RTU, etc. In actual use, the user can directly select the function block of the corresponding protocol to load it. The addition of the function block makes the user Programming work becomes simple, users only need to understand the basic PLC programming instructions and the characteristics and parameters of the corresponding bus corresponding protocol to complete the programming work.

GCAN-GT-412 can be programmed using OpenPCS software, which supports five standard PLC programming languages that comply with the IEC-61131-3 standard, such as: SFC (sequential function diagram), LD (ladder diagram), FBD (Function block), ST (structured text), IL (instruction list), which makes the program very portable and reusable, and the software also has a variety of debugging functions (such as power off, single step, monitoring, etc.)) To make the debugging process more convenient.

1.2 Performance characteristics

- High-speed 32-bit industrial-grade processor;
- Embedded hardware watchdog timer;
- Use external power supply (DC+24V, 40mA);
- Electrostatic discharge immunity level: contact discharge ±2KV, air discharge ± 15KV;
- Electrical fast transient pulse group immunity level: ±1KV;
- Surge immunity level: ±1KV;
- Working humidity range: 5%~95% RH without condensation;
- 2 CAN bus interfaces, 1 Ethernet interface, 1 RS232 serial interface;
- Programming software: OpenPCS (in accordance with IEC 61131-3 standard);
- Support CANopen protocol master/slave function;
- Support Modbus RTU/TCP master/slave function;
- Standard DIN rail installation method, specially designed for industrial use.
- The electrical isolation is 1500 Vrms;
- Working temperature range: -40℃~+85℃;
- Protection level: IP20;

1.3 Typical applications

- Industrial Ethernet and CAN bus data conversion
- Interconnection of industrial Ethernet equipment and CAN network equipment
- Electric power communication network
- Industrial control equipment
- High-speed, large data communication

2. Equipment installation

2.1 Module size

Equipment dimensions: (Long, including wiring terminals) 112mm * (Width) 99mm * (Height) 22mm, the schematic diagram is shown in Figure 2.1

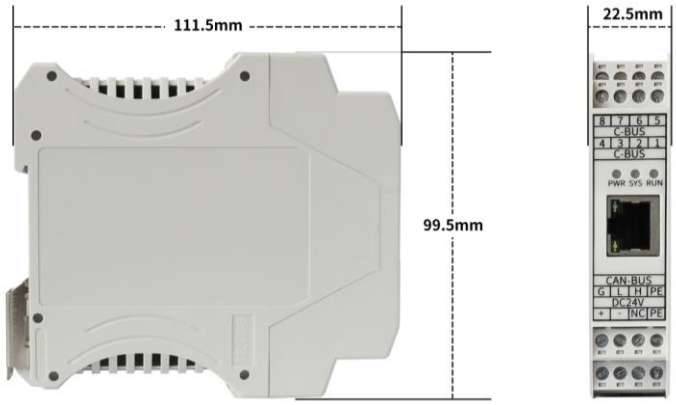


Figure 2.1 GCAN-GT-412 module dimensions

2.2 Fixation of equipment

The installation method of the GCAN-GT-412 module is shown in Figure 2.2. A flat-blade screwdriver can be used to assist in installing the module on the DIN rail.

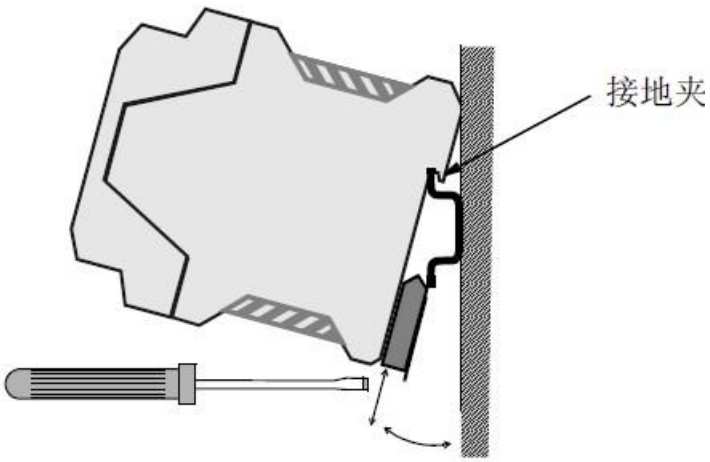


Figure 2.2 GCAN-GT-412 module installation

The GCAN-GT-412 module ground is connected to the guide rail of the mounting module. If the rail is fixed to a grounded metal component board, the module will be automatically grounded and no external ground wire is required. If the rail is fixed to an ungrounded base, the rail must be connected to the nearest ground terminal.

2.3 Interface definition and function

The GCAN-GT-412 module integrates 1 DC 24V power interface, 2 standard CAN-Bus interfaces, 1 standard Ethernet interface, and 1 RS232 serial interface. The wiring terminal block of the GCAN-GT-412 module is shown in Figure 2.3 .

Figure 2.3 GCAN-GT-412 module terminal block

The power interface of the GCAN-GT-412 module is led out by a 4 Pin plug-in terminal block. The interface definition is shown in Table 2.1.

Pin (from left to right)	port	name	Features
1	DC 24V	+	24V DC power input is positive
2		-	24V DC power input negative
3		NC	Unused
4		PE	shield

Table 2.1 Definition of power interface of GCAN-GT-412 module

The CAN-bus interface of the GCAN-GT-412 module is led out by two 4 Pin terminal blocks and can be used to connect two CAN-bus network or CAN-bus interface devices. The interface definitions are shown in Table 2.2.

Pin (from left to right)	port	name	Features
G	CAN-BUS	CAN2_G	CAN_GND ground
L		CAN2_L	CAN_L signal line (CAN low)
H		CAN2_H	CAN_H signal line (CAN high)
PE		PE	shield
2	C-BUS	CAN1_L	CAN2_L signal line (CAN low)

3		CAN1_H	CAN2_H signal line (CAN high)

Table 2.2 CAN-bus signal distribution of GCAN-GT-412 module

The serial interface of the GCAN-GT-412 module is led out by a 4 Pin terminal and can be used to connect an RS232 device. The interface definition is shown in Table 2.3.

Pin (from left to right)	**port**	**name**	**Features**
5	C-BUS	RS232-RX	RS232 data receiving
6		RS232-TX	RS232 data transmission
8		GND	Signal ground
the remaining		NC	Duty

Table 2.3 RS232 interface definition of GCAN-GT-412 module

3. Communication connection

3.1 Serial port connection

The RS232 interface of GCAN-GT-412 uses the standard serial port level, so the module can be directly connected to the equipment with RS232.

3.2 CAN connection

GCAN-GT-412 only needs to connect CAN_H to CAN_H and CAN_L to CAN_L to establish communication when accessing the CAN bus.

The CAN-bus network adopts a straight-line topology structure, and the two furthest terminals of the bus need to install 120Ω terminal resistance; if the number of nodes is greater than 2, the intermediate nodes do not need to install 120Ω terminal resistance. For branch connections, the length should not exceed 3 meters. The connection of CAN-bus bus is shown in Figure 3.1.

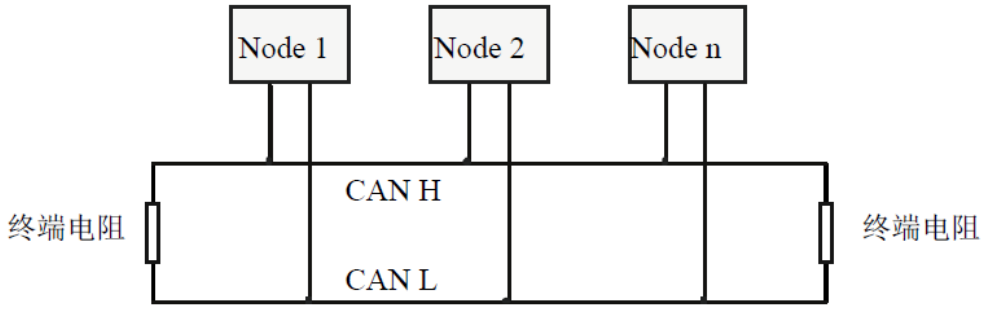


Figure 3.1 Topology of CAN-bus network

Please note: CAN-bus cable can use ordinary twisted pair and shielded twisted pair. The theoretical maximum communication distance mainly depends on the bus baud rate, and the relationship between the maximum bus length and baud rate is shown in Table 3.1. If the communication distance exceeds 1km, the cross-sectional area of the line should be greater than Φ1.0mm2, the specific specifications should be determined according to the distance, and the conventional is to increase appropriately with the length of the distance.

Baud rate	Bus length
1 Mbit/s	40m
500 kbit/s	110m
250 kbit/s	240m
125 kbit/s	500m
50 kbit/s	1.3km
20 kbit/s	3.3km
10 kbit/s	6.6km
5 kbit/s	13km

Table 3.1 Baud rate and maximum bus length reference table

3.3 CAN bus termination resistance

In order to enhance the reliability of CAN communication and eliminate CAN bus terminal signal reflection interference, the two farthest endpoints of the CAN bus network usually need to add terminal matching resistors, as shown in Figure 3.2. The value of the termination matching resistance is determined by the characteristic impedance of the transmission cable. For example, the characteristic impedance of the twisted pair is 120Ω, then the two end points on the bus should also integrate 120Ω termination resistance. If other nodes on the network use different transceivers, the termination resistance must be calculated separately.

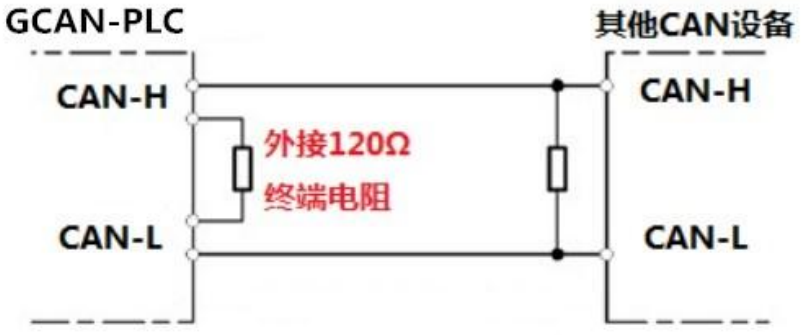


Figure 3.2 GCAN-GT-412 connected with other CAN node devices

Please note: GCAN-GT-412's CAN bus does not integrate a 120Ω termination resistor. If the number of nodes is greater than 2, the intermediate node does not need to install a 120Ω termination resistor. When needed, connect the two ends of the resistor to CAN_H and CAN_L, as shown in Figure 3.2.

4. Use of OpenPCS programming software

4.1 Software installation

OpenPCS 2008 programming software (this software is included in the CD-ROM, or you can download and install the software online)

4.2 Introduction to PLC programming interface

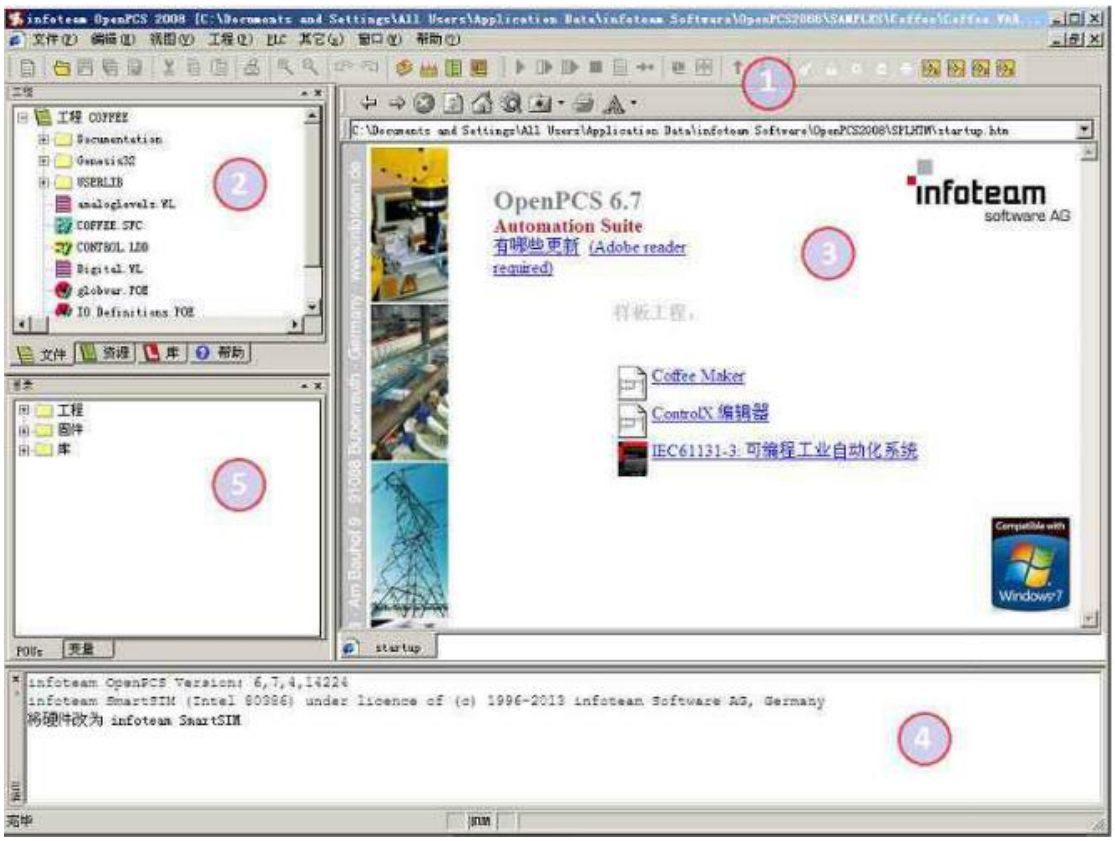


Figure 4.1 OpenPCS programming interface

The OpenPCS programming interface mainly includes:
1) Menu toolbar
2) Project Browser
3) Edit window
4) Output window
5) Directory window

4.3 Create Project

4.3.1 Project creation

Click Project->new to create a new project, as shown in Figure 5.2 below.

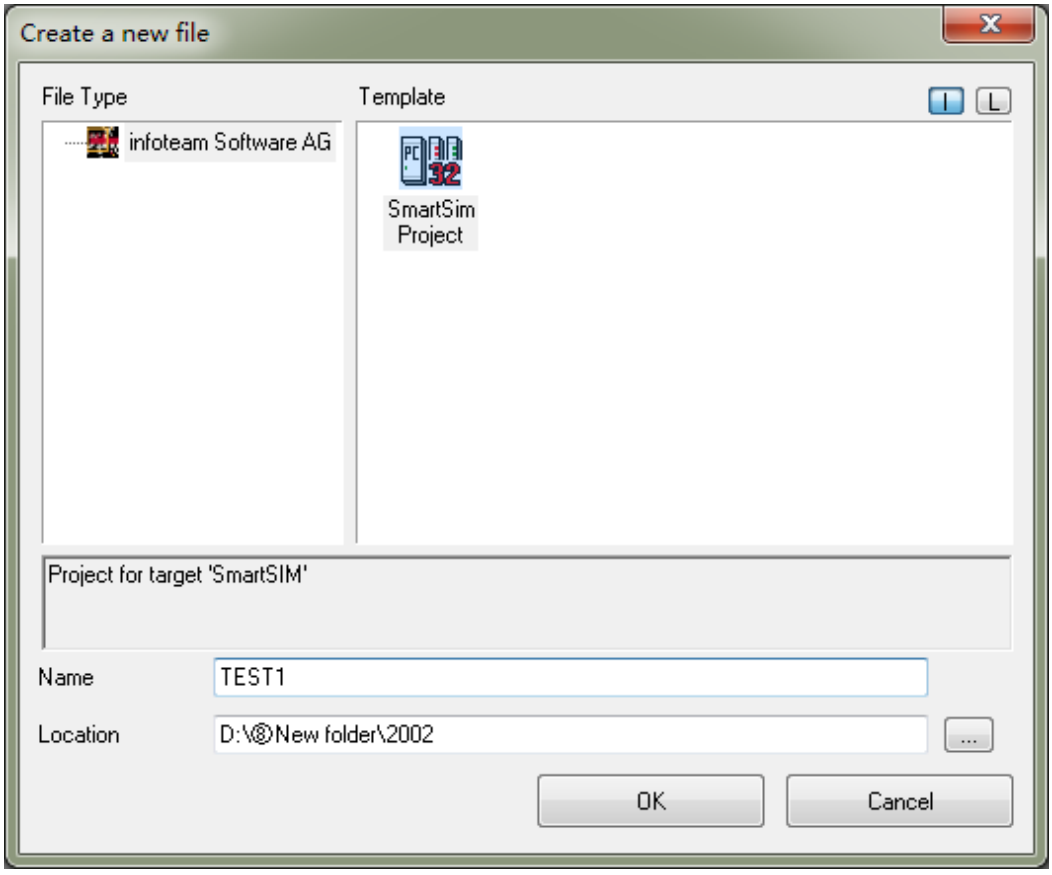


Figure 4.2 Create a project

4.3.2 Add program page file

Add files to the project (for example: add the program page ST, Program written in ST language), as shown in Figure 5.3.

Please note that the string entered in the Name column cannot begin with a number.

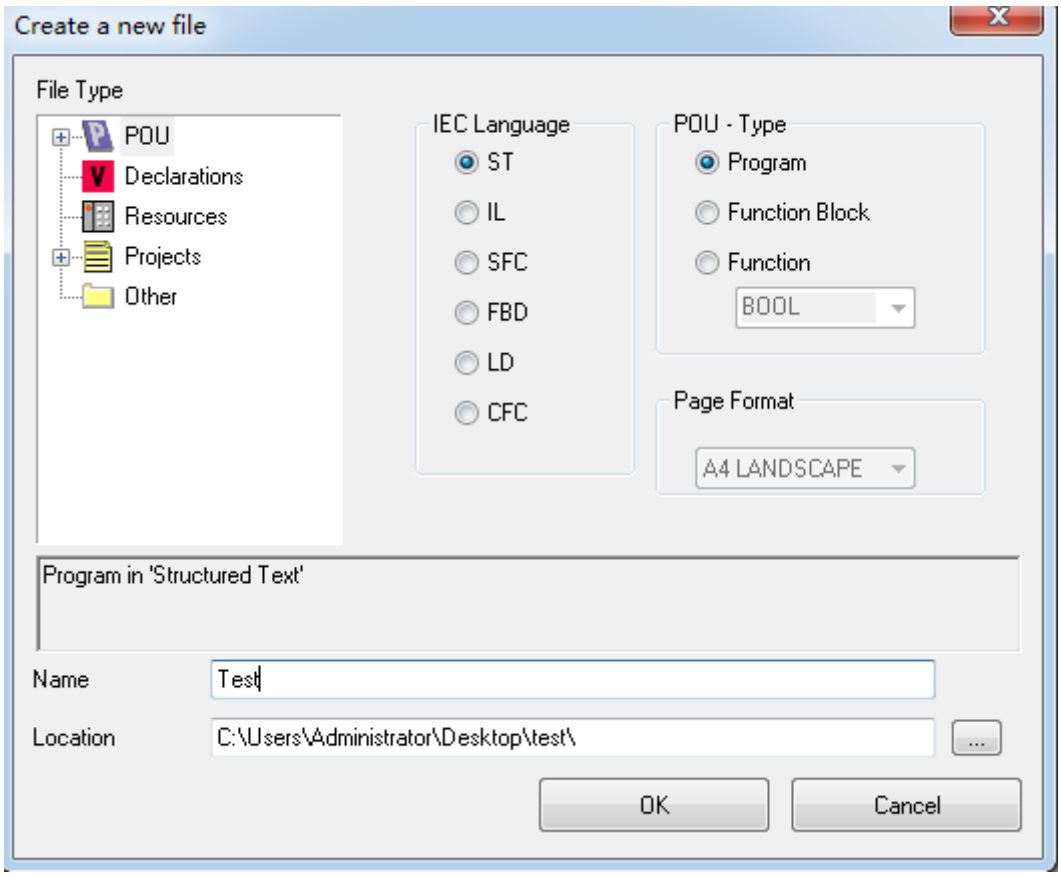


Figure 4.3 Create a program page in the project

4.3.3 Programming

First, you need to define variables (VAR to END_VAR) in the variable area.

```
VAR                             (*内部变量段开始                                    *)
    v1:INT:=0;                  (*:为变量/类型分隔符，:=为初始化操作符              *)
    v2:INT:=0;
    oled at%Q0.0:Byte;          (*   %Q0.0表示输出0单元第0位，:为变量/类型分割符    *)
                                (*符号变量地址声明。分配Q0.0到字节 OLED             *)
                                (* 如果对变量声明不理解，可参考电子书第49页，变量声明的示例*)
END_VAR
```

After completing the variable definition, you can start programming in the following programming interface. The following is a simple routine statement written in ST:
LED marquee routine:

```
IF v1<100 THEN                  (*v1自加到100时，v1归零。v1越大，闪灯的变化频率越慢   *)
    v1:=v1+1;                   (* := 可表示初始化、输入连接或者赋值                 *)
                                (* 如果对st语言的各种符号不理解可参考电子书第25页，分界符 *)
ELSE
    v1:=0;
    v2:=v2+1;
    if v2>=255 then             (*v2自加到255时，v2归零                             *)
        v2:=0;
    end_if;
    oled:=int_to_byte(v2);      (* int_to_byte 整型转字节。类型转换类函数，电子书57页 *)
                                (* :=这里表示赋值                                    *)
end_if;
```

4.3.4 Set up debug connection

1. Click
PLC->Connections...

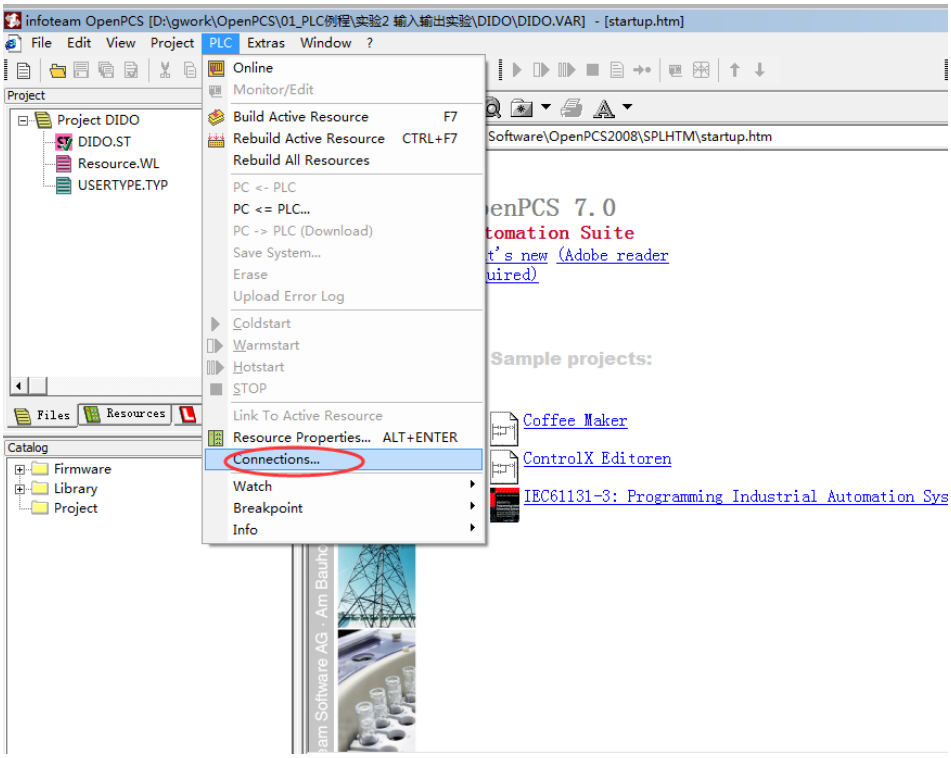


Figure 4.4 Debug connection

2. Create a new connection in the Connection Setup window and set the parameters. Click the "New" button.

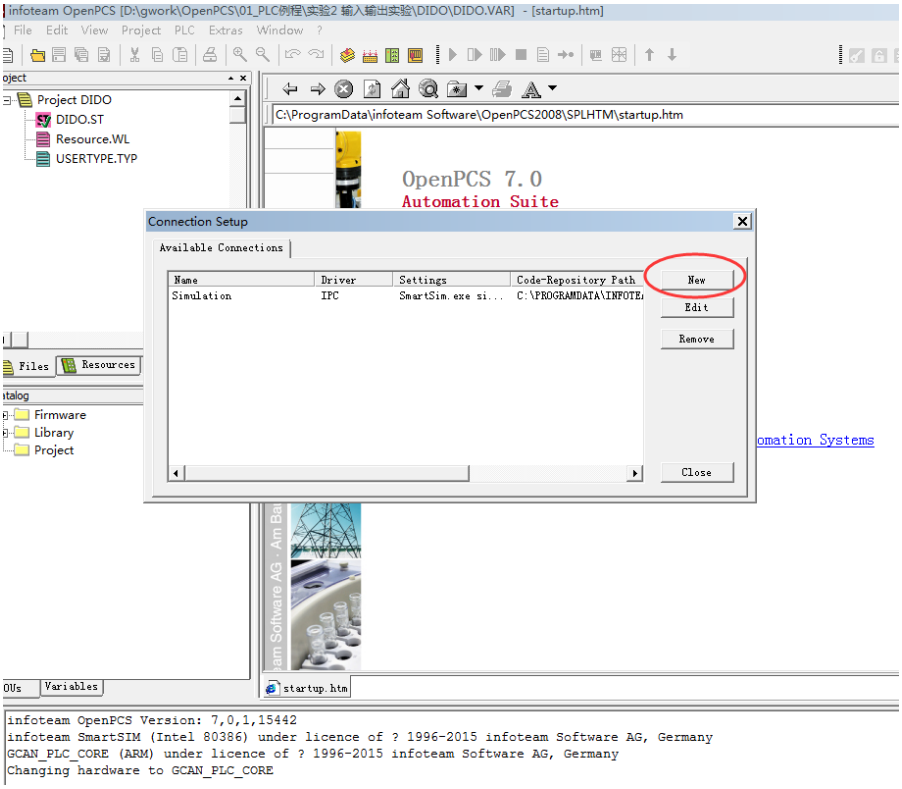


Figure 4.5 Click "New"

3. Enter TCP in Name and click the Select button.

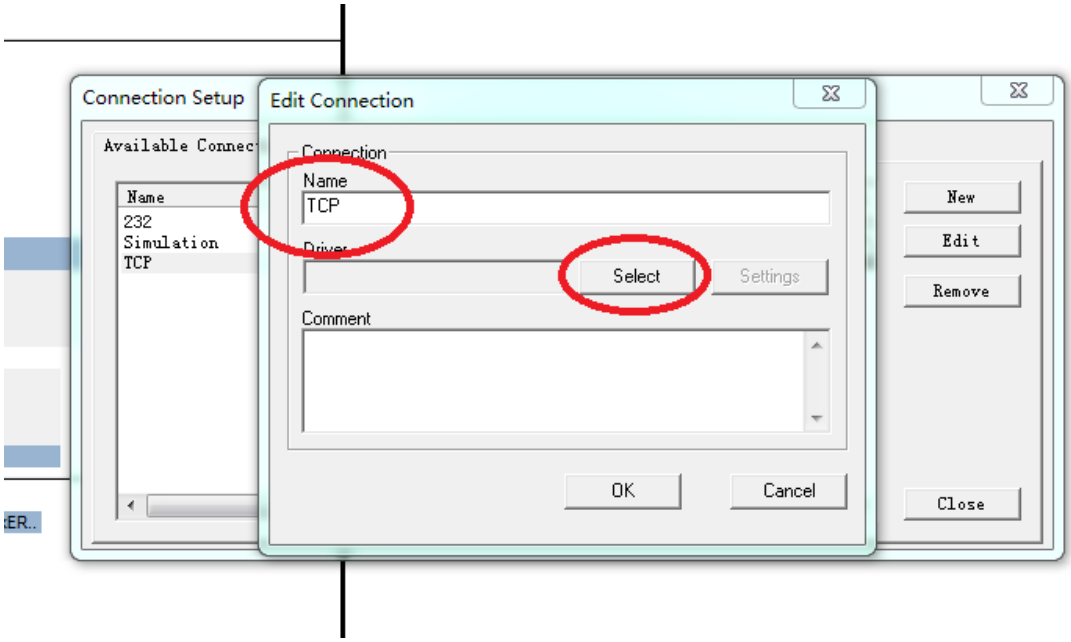


Figure 4.6 Click the "Select" button

4、4. Click the TCP432 icon 

，then click OK.

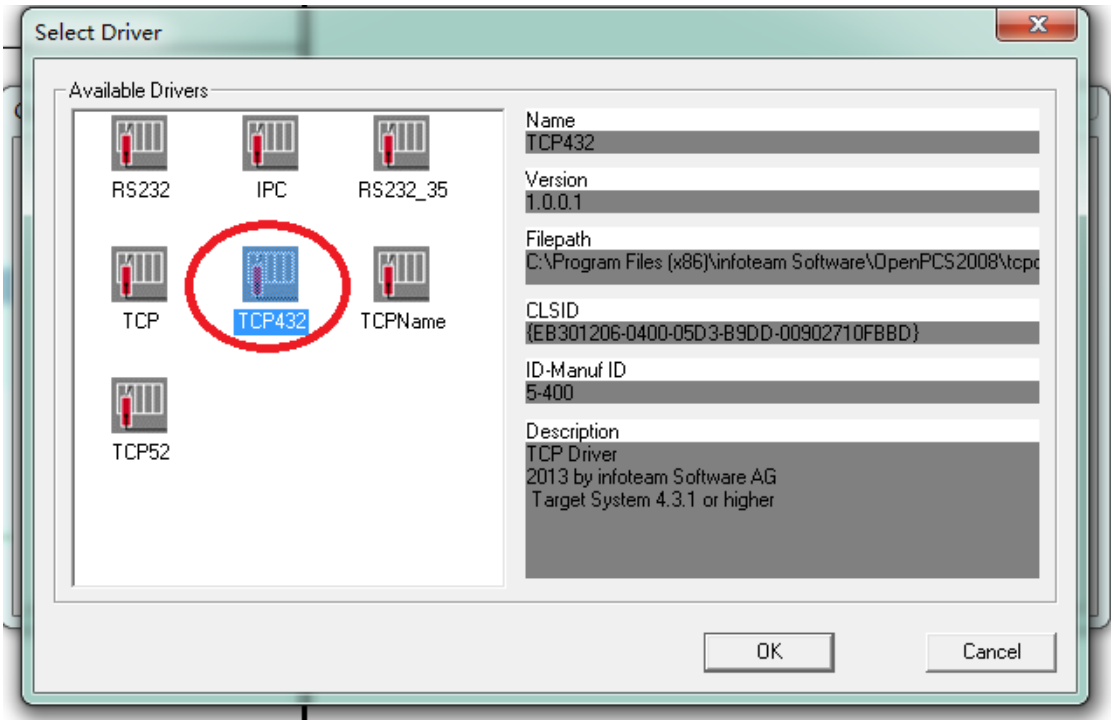


Figure 4.7 Select TCP432

5. The word "TCP432" will be displayed in the Driver. Click the "Settings" button.

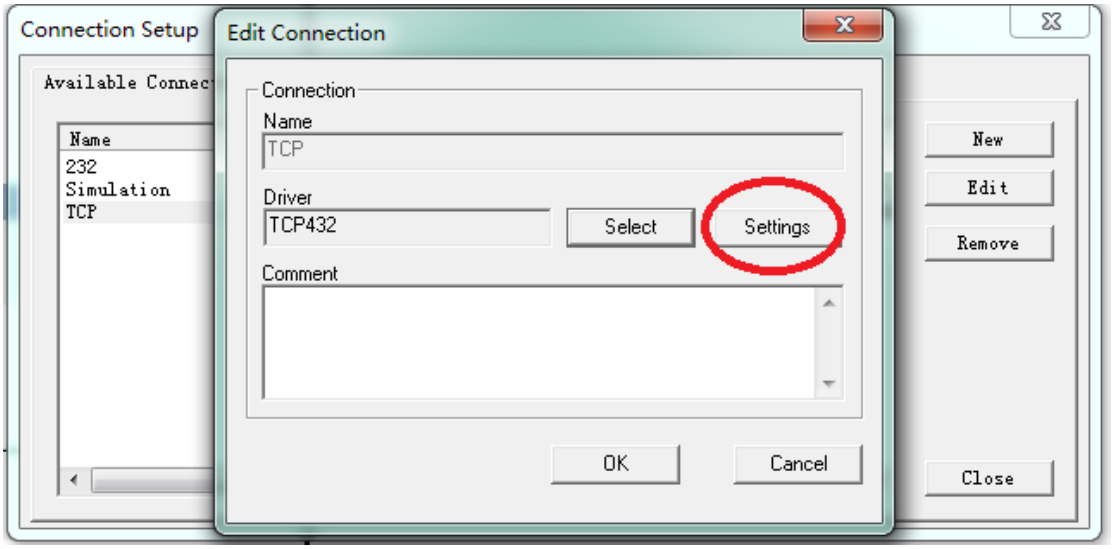


Figure 4.8 Click the "Settings" button

6. Please enter 23042 for Port. IP address is 192.168.1.30, click OK after setting.

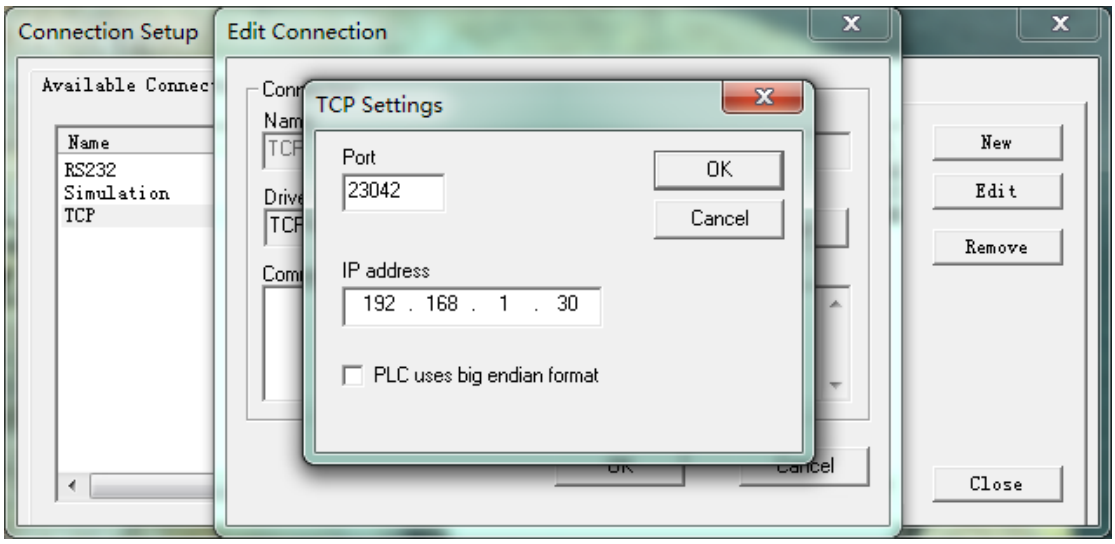


Figure 4.9 IP address and port number settings

7. After setting, return to the Connection Setup interface and click "Close".

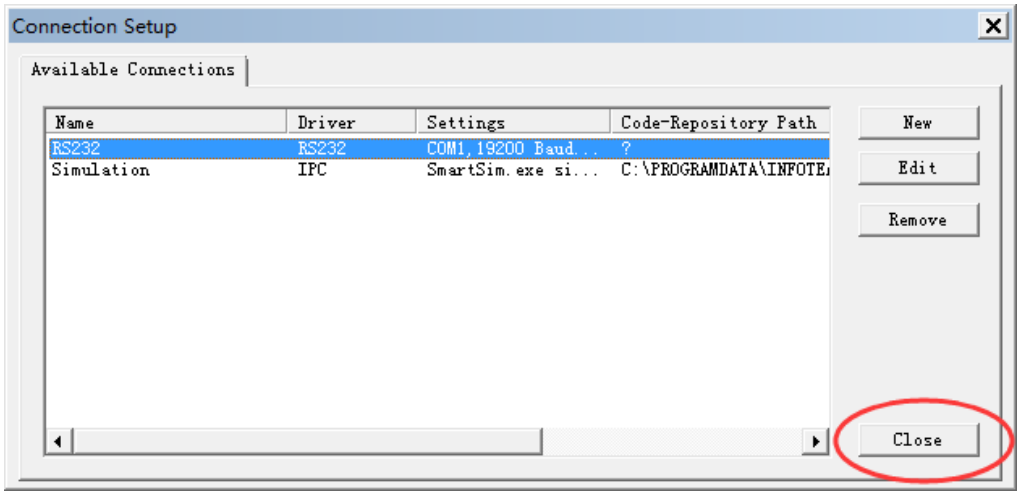


Figure 4.10 Click "Close"

8. Set Resource Properties, as shown in the figure below.

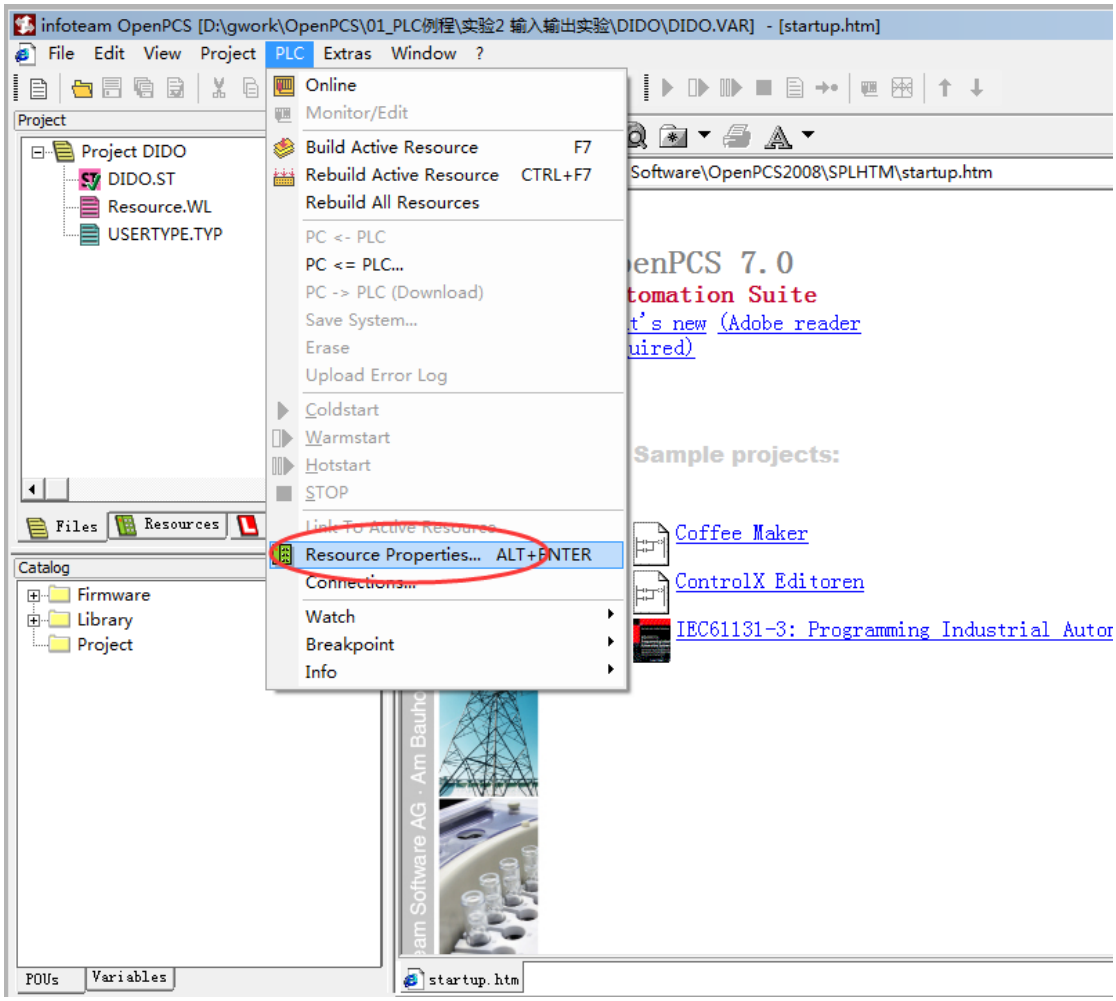


Figure 4.11 Set resource properties

9. Choose GCAN_PLC and TCP.

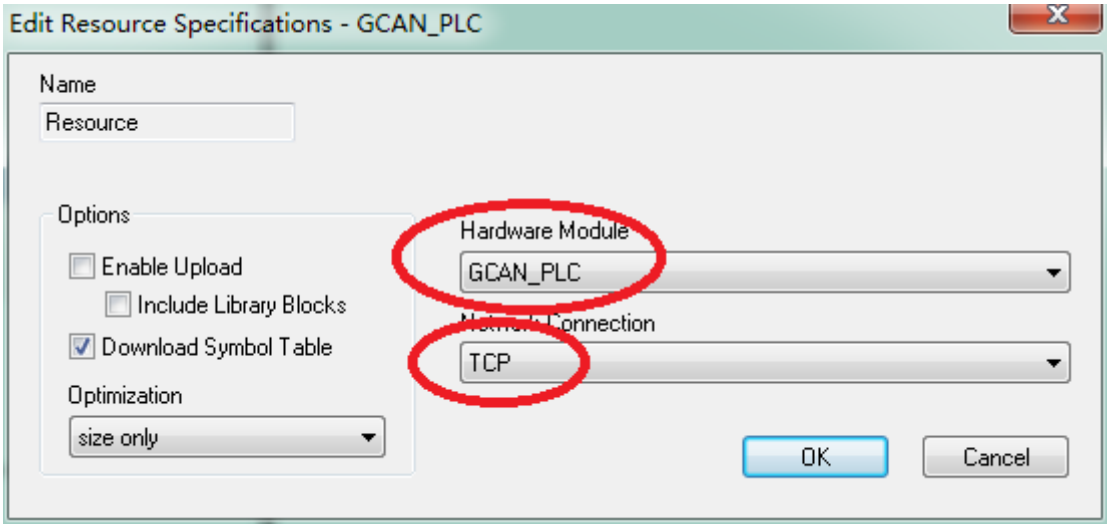


Figure 4.12 Select GCAN_PLC and TCP

4.3.5 Download and debug the program

1. After completing the program, you need to click the Build Active Resource button, as shown in Figure 5.13.

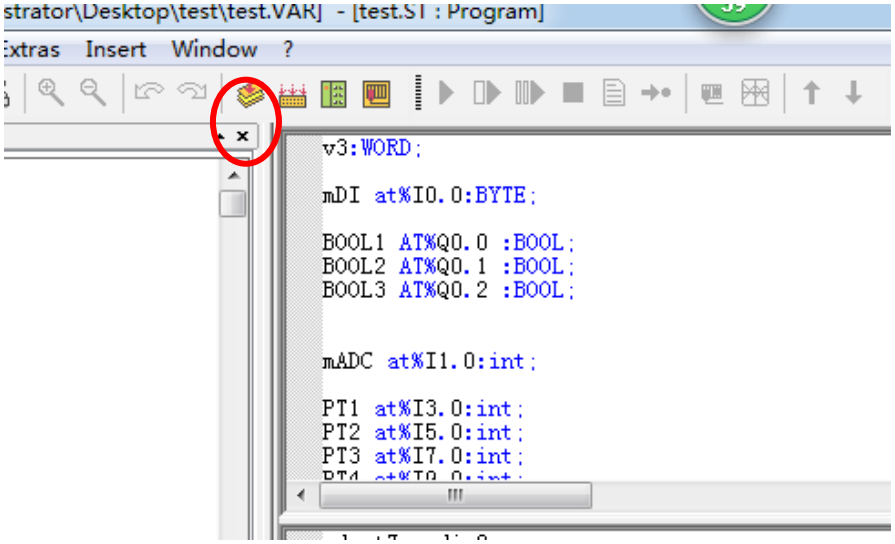


Figure 4.13 Click the Build Active Resource button

2. After the compilation is completed, there is no error. As shown below.

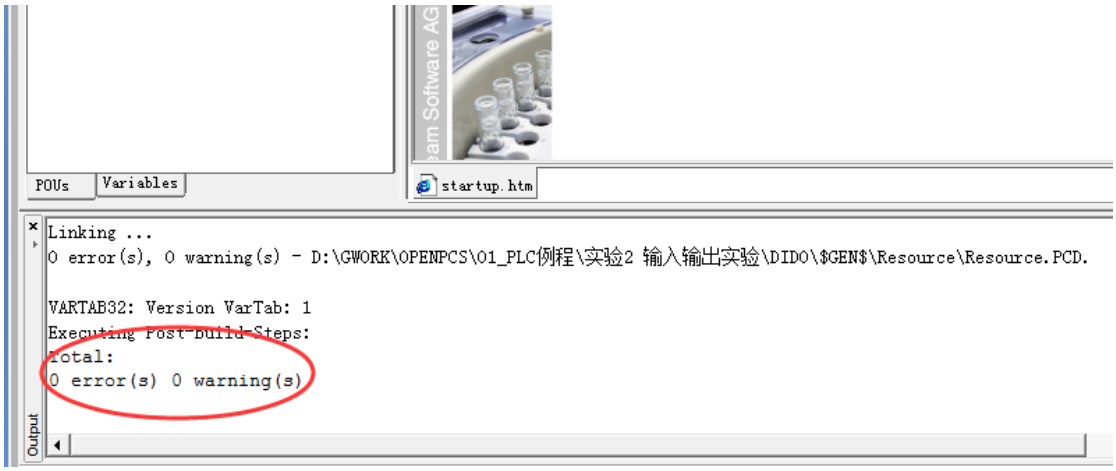


Figure 4.14 Compile completed

3. Click the Online button.

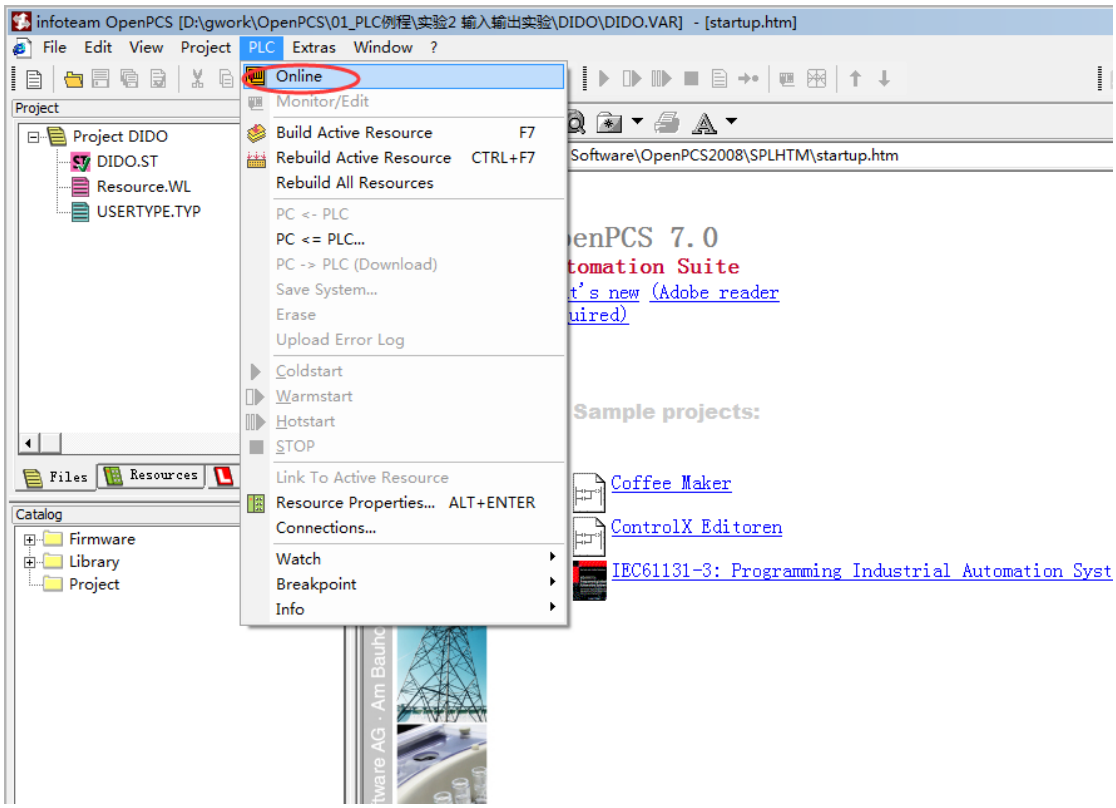


Figure 4.15 Click the Online button

4. Click PC->PLC(Download) in the drop-down menu to download the program.

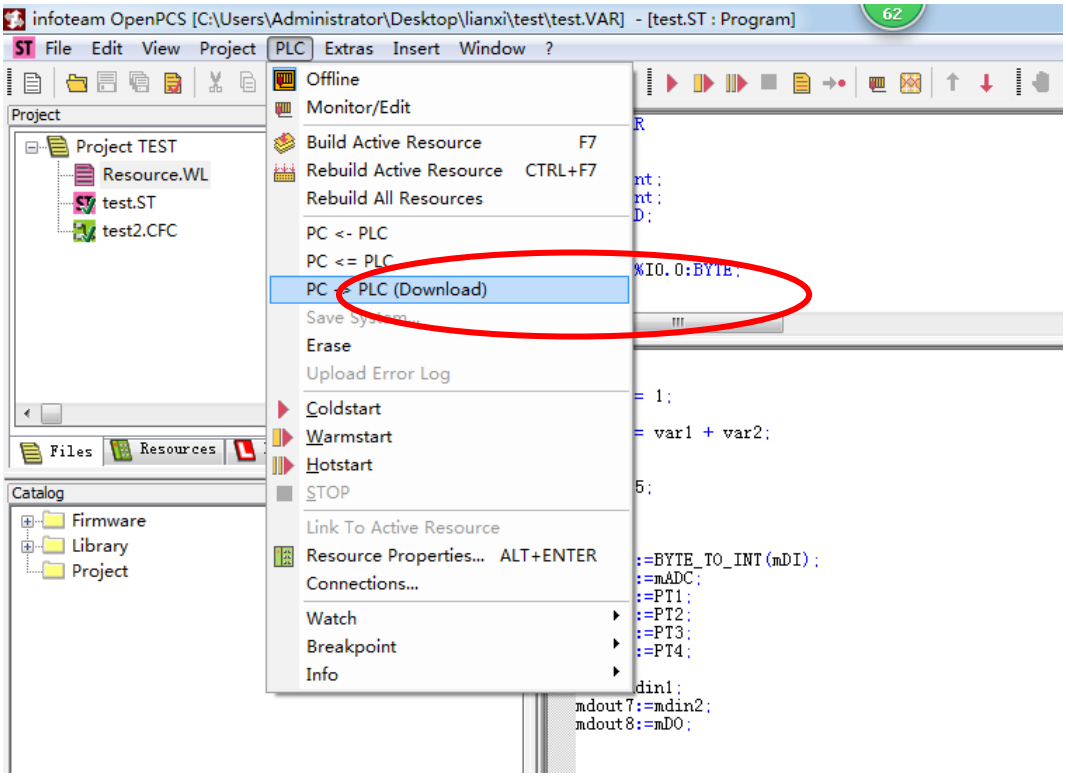


Figure 4.16 Download program

5. Technical specifications

PLC parameters	
Programming environment	OpenPCS software
Flash (program memory)	32M bytes
SRAM (data memory)	16M bytes
User data storage area	2k bytes
Run-Time system	1 PLC task
PLC cycle time	1000 instructions take about 3ms
Program online modification	not support
Programming language implementation standards	IEC 61131-3
Types of programming languages	SFC (sequential function diagram), LD (ladder diagram), FBD (function block), ST (structured text), IL (instruction list)
Desk I/O	No.
Floating point arithmetic	stand by
Communication interface features	
Communication format	1 CAN interface, 1 Ethernet interface, 1 RS232, 1 RS485 interface
CANopen master/slave	stand by
Modbus RTU/TCP master/slave	stand by
Electrical parameters	
power supply	24V DC（-15%/+20%）
Starting current	2.5 times continuous current
Recommended fuse capacity	≤10A
Power shock	24V DC max / 10A max
Electrical isolation	1500 Vrms
Environmental test	
Operating temperature	-40℃~+85℃
Working humidity	95%RH, no condensation
EMC test	EN 55024:2011-09 ； EN 55022:2011-12
Anti-vibration/impact resistance	EN 60068-2-6 / EN 60068-2-27/29
Anti-electromagnetic interference/anti-electromagnetic radiation performance	EN 61000-6-2 / EN 61000-6-4
Protection class	IP 20
Connection method	
Ethernet	RJ45
CAN	OPEN4 terminal
RS232	OPEN4 terminal
RS485	OPEN4 terminal

Appendix A: Introduction to CANopen Protocol

The CANopen protocol was developed in the late 1990s by the CiA organization (CAN-in-Automation) on the basis of CAL (CAN Application Layer), and once it was launched, it was widely recognized and applied in Europe. After several revisions to the CANopen protocol specification text, the stability, real-time, and anti-interference properties of the CANopen protocol have been further improved. And CiA has continuously introduced device sub-protocols in various industries, so that the CANopen protocol can be developed and promoted faster in various industries. At present, the CANopen protocol has been widely used in motion control, vehicle industry, motor drive, engineering machinery, marine shipping and other industries.

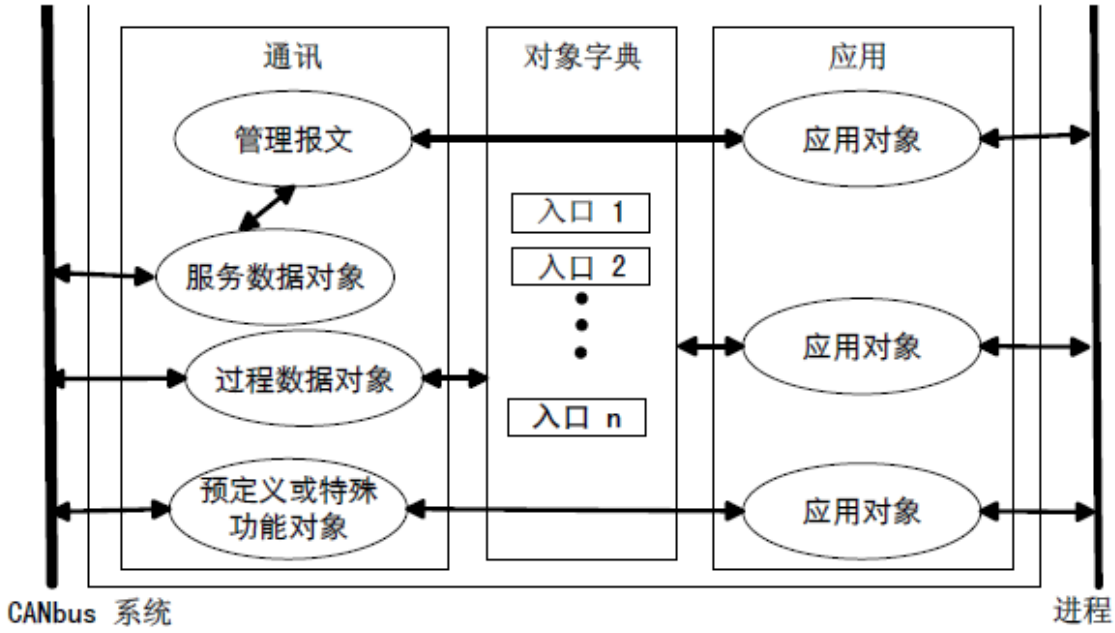


Figure A1 CANopen device structure

Figure A1 shows the CANopen device structure. The CANopen protocol is usually divided into three parts: user application layer, object dictionary, and communication.

A.1 Explanation of related terms and writing rules

1. Explanation of terms:

PDO：Process Data Object
TPDO：Transmit Process Data Object
RPDO：Receive Process Data Object
SDO：Service Data Object
NMT：Network Management
SYNC: Synchronization Objects

EMCY: Emergency Objects
OD：Object Dictionary
EDS：Electronic Data Sheet
CAN-ID: Controller Area Network-Identify
COB-ID: Communication Object-Identify
SSDO：Servers Service Data Object
DS：Draft Standard

2. Writing rules

In this manual, the writing of the object dictionary index and sub-index follow the rules shown in Figure A2 below, where the index is expressed in hexadecimal, the sub-index is expressed in decimal, and the index and sub-index are separated by spaces.

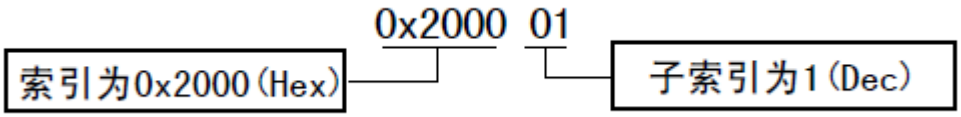


Figure A2 Index/sub-index writing rules

A.2 Predefined CAN identifier

Object function code CAN	ID range	Object function code CAN
NMT network management commands	0000b	000h
Sync message	0001b	080h
Time Stamp message	0010b	100h
Emergency message	0001b	081h-0FFh
TPDO1 send process data object 1	0011b	181h-1FFh
RPDO1 Receive process data object 1	0100b	201h-27Fh
TPDO2 send process data object 2	0101b	281h-2FFh
RPDO2 Receive Process Data Object 2	0110b	301h-37Fh
TPDO3 send process data object3	0111b	381h-3FFh
RPDO3 Receive Process Data Object 3	1000b	401h-47Fh
TPDO4 send process data object 4	1001b	481h-4FFh
RPDO4 receiving process data object 4	1010b	501h-57Fh
SDO Server-to-Client Service Data Object (Answer)	1011b	581h-5FFh
SDO Client-to-Server Service Data Object (Ask)	1100b	601h-67Fh
NMT error control Network management error control	1110b	701h-77Fh

A.3 CANopen object dictionary

CANopen object dictionary (OD: Object Dictionary) is the core concept of CANopen protocol. The so-called object dictionary is an ordered group of objects. Each object is addressed with a 16-bit index value. This index value is usually called an index, and its effective range is between 0x1000 and 0x9FFF. To allow access to a single element in the data structure, an 8-bit index value is also defined. This index value is often referred to as a sub-index. Each CANopen device has an object dictionary. The object dictionary contains all the parameters describing this device and its network behavior. The object dictionary usually records these parameters in an electronic data file (EDS: Electronic Data Sheet), without the need to keep these The parameters are recorded on paper. For the master node in the CANopen network, there is no need to access every object dictionary entry of the CANopen slave node.

The items in the CANopen object dictionary are described by a series of sub-protocols. The sub-protocol describes each object in the object dictionary its function, name, index, sub-index, data type, and whether this object is necessary, read and write attributes, etc., so as to ensure the compatibility of the same type of equipment from different manufacturers. The core description sub-protocol of the CANopen protocol is DS301, which includes the application layer and communication structure description of the CANopen protocol. Other sub-protocols are supplements and extensions to the description text of the DS301 protocol. The CANopen protocol contains many sub-protocols, which are mainly divided into the following types.

1. Communication Profile

The communication sub-protocol describes the main form of the object dictionary and the communication objects and parameters in the object dictionary. This sub-protocol applies to all CANopen devices, and its index value ranges from 0x1000 to 0x1FFF.

2. Manufacturer-specific profile

Manufacturer-defined sub-protocols. For special functions not defined in device sub-protocols, manufacturers can define object dictionary objects according to requirements in this area. Therefore, for different manufacturers, the definition of the object dictionary entries with the same index may not be the same, and the index value range is 0x2000~0x5FFF.

3. Device profile

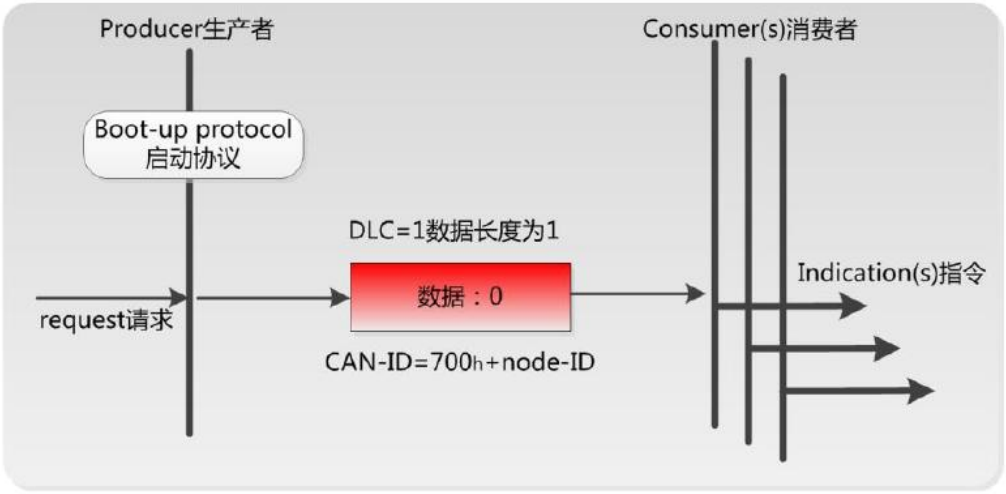
The device sub-protocol defines objects in the object dictionary for various types of devices. There are currently dozens of sub-protocols defined for different types of devices, such as DS401, DS402, DS406, etc., with index values ranging from 0x6000 to 0x9FFF.

A.4 CANopen communication

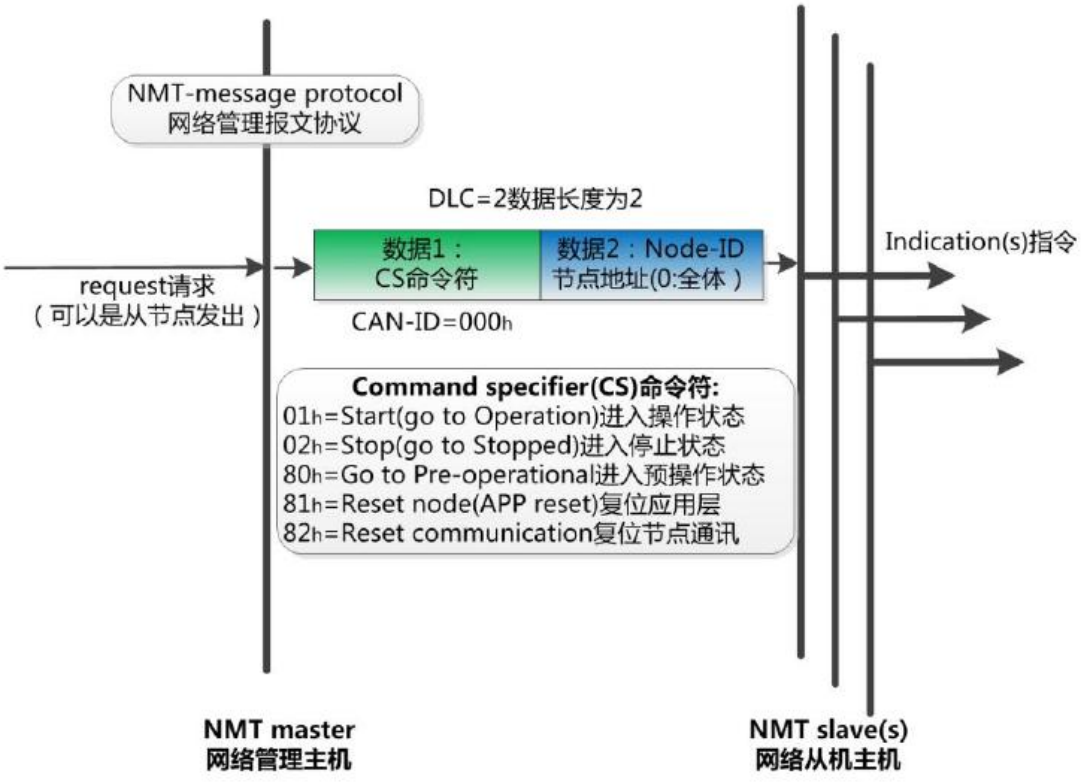
The CANopen protocol mainly defines four objects: management message object NMT (Network Management), service data object SDO (Service Data Object), process data object PDO (Process Data Object), predefined message or special function object.

1. Network Management (NMT)

Management messages are responsible for layer management, network management, and ID assignment services, such as initialization, configuration, and network management (including node protection). In network management, only one master node, one or more slave nodes are allowed in the same network, and follow the master-slave model. Through the NMT service, we can initialize, run, monitor, reset and stop the node. All nodes are considered as NMT slaves.



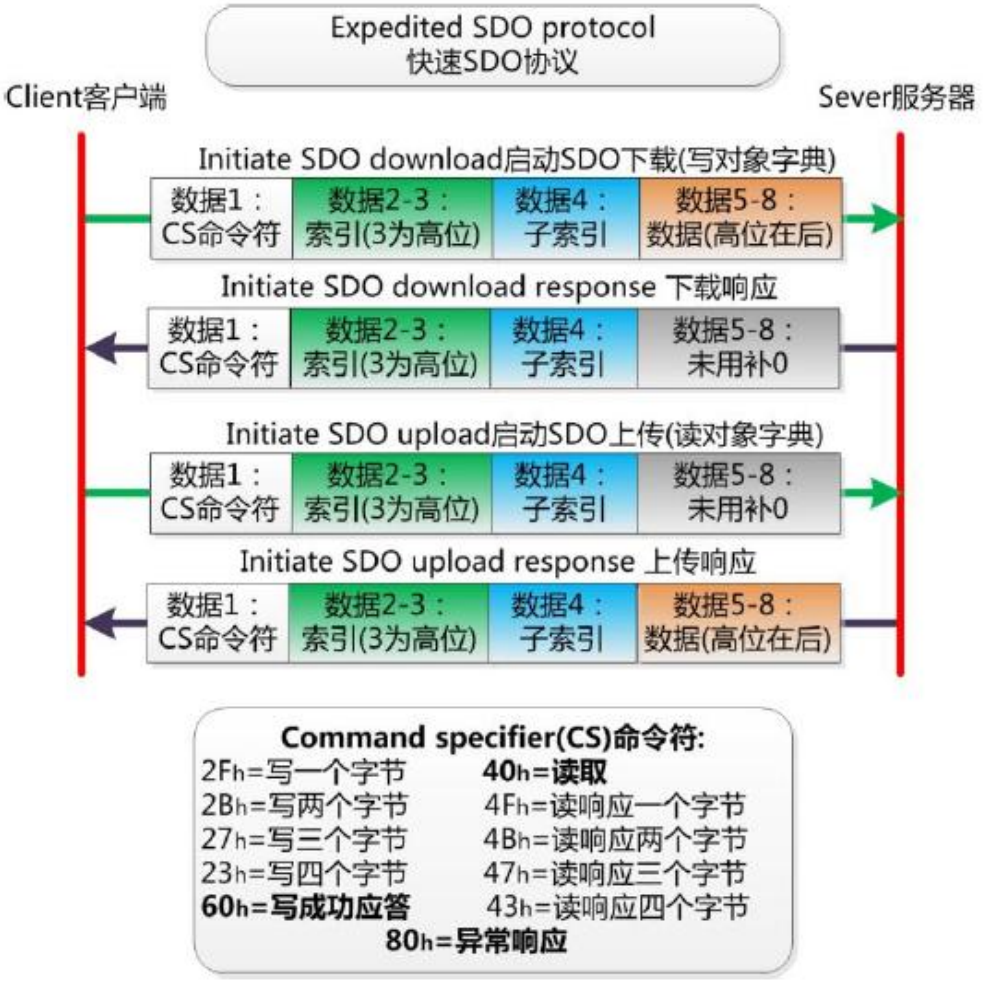
As shown in the figure above, for example, a CANopen slave device will send a data with a frame ID of 0x702 and data of 0x00 after power-up; it means that the device has been started and the node number is 2.



As shown in the figure above, for example, a CANopen master station sends a frame of data to the slave station, the frame ID is 0x000, the frame data is 0x01, 0x02, then this command can make the CANopen slave device with node number 2 enter the operating state .

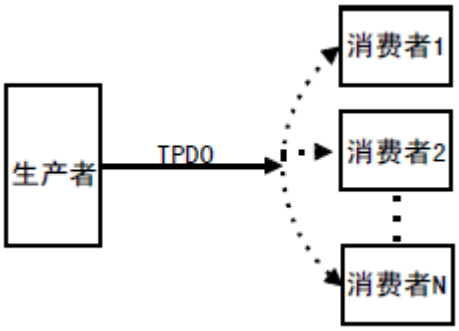
2. Service Data Object (SDO)

SDO is mainly used by the master node to configure the parameter of the slave node. Service confirmation is the biggest feature of SDO. It generates a response for each message to ensure the accuracy of data transmission. In a CANopen system, usually the CANopen slave node serves as the SDO server, and the CANopen master node serves as the client. The client can access the object dictionary on the data server through the index and sub-index. In this way, the CANopen master node can access the parameters of any object dictionary items of the slave node, and SDO can also transmit data of any length (when the data length exceeds 4 bytes, it is split into multiple messages for transmission).



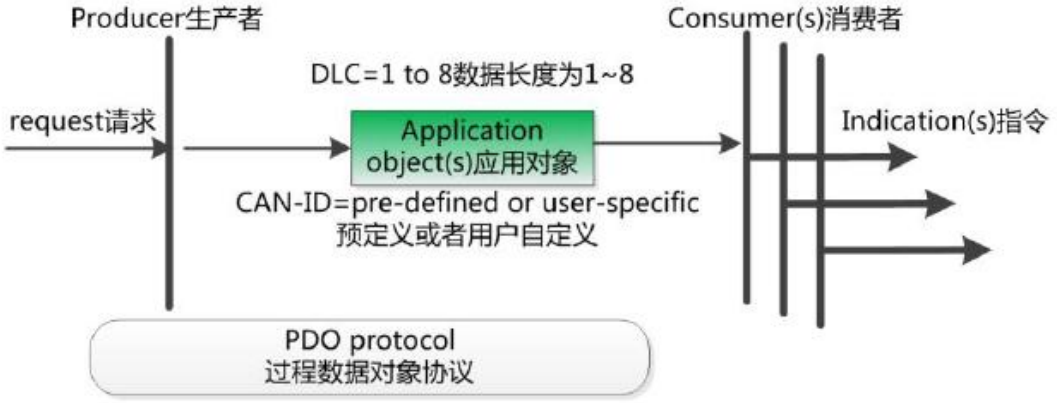
3. Process Data Object (PDO)

PDO is used to transmit real-time data, and its transmission model is the producer-consumer model, as shown in Figure A3. The data length is limited to 1~8 bytes. PDO communication objects have the following characteristics:



图A3 生产者消费者模型

- There is no protocol stipulation for PDO communication, and the content of PDO data is defined by its CAN-ID (also known as COB-ID);
- Each PDO is described by 2 objects in the object dictionary:
  - ◆ PDO communication parameters, which define the COB-ID, transmission type and timing period used by the device;
  - ◆ PDO mapping parameter, the mapping parameter contains a list of objects in the object dictionary, these objects are mapped to the corresponding PDO, including the length of the data (unit: bits), both producers and consumers must know this mapping parameter, Only then can the PDO content be correctly interpreted.
- The content of the PDO message is predefined. If the PDO supports variable PDO mapping, then the PDO can be configured through SDO;
- PDO can have multiple transmission methods:
- ◆ Synchronous transmission (to achieve synchronization by receiving synchronization objects), synchronous transmission can be divided into aperiodic and periodic transmission. Acyclic transmission is pre-triggered by remote frames or pre-triggered by object-specific events specified in the device sub-protocol. Periodic transmission is achieved by receiving synchronization objects (SYNC), which can set 1~240 synchronization objects to trigger;
- ◆ Asynchronous transmission (triggered by a specific event), which can be triggered in two ways, the first is to trigger the transmission of PDO by sending a remote frame with the same COB-ID of PDO, and the second is specified in the device sub-protocol Object specific events to trigger (for example, timing transmission, data state change transmission, etc.).



4. Predefined messages or special function objects

Predefined messages or special function objects provide specific functions for CANopen devices, which is convenient for CANopen master to manage slaves. In the CANopen protocol, the COB-ID has been predefined for special functions, which mainly have the following special messages:

- Synchronization (SYNC), the message object mainly realizes the synchronous transmission of the entire network, each node uses the synchronization message as the PDO synchronization trigger parameter, so the COB-ID of the synchronization message has a higher priority and the shortest Transmission time
- Time stamp object (Time Stamp), providing a common time reference for each node;
- Emergency object (Emergency), when an error occurs in the device, the object is triggered, that is, the device internal error code is sent;
- Node/Life Guarding, the master node can obtain the status of the slave node through node protection. The slave node can obtain the status of the master node through life protection;
- Start the message object (Boot-up), send the object to the network after the initialization of the node is completed, and enter the pre-operation state.

A.5 CANopen network configuration

In the CANopen protocol description text DS305, a network configuration protocol is defined, that is, the network configuration service LSS (Layer Setting Service), which uses the CANOpen module with the LSS master function to query or modify the CANOpen module with the LSS slave through the CAN bus. Certain parameters.

By using LSS, you can query or modify the following parameters:

- Node-ID of CANopen slave station;
- Bit timing parameters (baud rate) of the physical layer;
- LSS address (feature object 1018h).

Appendix B: Introduction to Modbus Protocol

Modbus communication protocol is a general language developed by Modicon and applied on PLC or other industrial controllers. Through this protocol, serial communication can be realized between the controllers. The Modbus communication protocol defines a message structure that the controller can recognize and use, and describes the process of the master controller accessing the slave device, such as specifying how the slave station responds Respond, check and report transmission errors, etc. The communication method of Modbus protocol is the master-slave method. The master station first sends a communication request instruction to the slave station device, and the slave node sends back the reply data to the master station according to the function code in the request instruction. Each slave device in the network must be assigned a unique address, up to 31 slave devices. Through up to 24 kinds of bus commands to achieve the exchange of information between the master controller and the slave device. The slave device only executes the instructions sent to itself, and does not respond to the messages at the beginning of other slave addresses. This question-and-answer communication mode greatly improves the accuracy of communication. Because of its advantages of simple operation, high efficiency, and reliable communication, Modbus protocol has become an international communication standard, and has been supported by most international manufacturers of industrial control products. The communication protocol has been widely used in machinery, water conservancy, electric power, environmental protection and other industrial equipment.

Modbus TCP communication protocol can be used for the monitoring of automation equipment. A common application is to develop a gateway based on this protocol, through which PLCs, I/O modules and other buses can be connected to Ethernet. Modbus TCP does not change the original Modbus protocol, but simply transplants it to the TCP/IP protocol as an application layer protocol. Modbus TCP protocol requires a response for every call. Using TCP/IP protocol, the user interface can be made more friendly through the form of webpage. Using a web browser, you can view the operation of the equipment within the corporate network. Schneider has registered port 502 for Modbus, so that real-time data can be embedded in web pages, and by embedding a web server in the device, a web browser can be used as the device's operating terminal. However, the Modbus protocol itself has some shortcomings. It does not support some new network technologies that are widely adopted, such as object-based communication models. When users use it, they have to manually configure some parameters, such as information data type, register number, etc.

B.1 Modbus RTU protocol data format

Modbus protocol has ASCII (American Standard Information Exchange Code) and RTU (remote terminal unit) two data transmission methods can be selected by the user, but all devices on a Modbus network must select the same transmission mode and

serial port parameters. Among them, the 8-bit data in the RTU mode information frame includes two 4-bit hexadecimal characters. Compared with the ASCII mode, it only requires fewer digits to express the same information, and it has a larger data flow than the ASCII mode at the same rate. . Therefore, RTU mode is often used under normal circumstances. GCAN-204 equipment also uses RTU mode.

RTU mode message transmission starts and ends with at least 3.5 character intervals (such as T1-T2-T3-T4 in Table B.1). The information frame consists of the address field, function field, data field, and CRC check field. The character bits are composed of hexadecimal 0-9, AF. The entire message frame must be transmitted as a continuous stream. If there is a pause time of more than 1.5 characters before the frame is completed, the receiving device will refresh the incomplete message and assume that the next byte is the address field of a new message. Similarly, if a new message starts after the previous message in less than 3.5 characters, the receiving device will consider it as a continuation of the previous message. This will cause an error because the value in the last CRC field cannot be correct.

Start bit	Device address	Function code	Data	CRC check	End character
T1-T2-T3-T4	8Bit	8Bit	N 个 8Bit	16Bit	T1-T2-T3-T4

Table B.1 RTU message frame format

(1) Address field

Specify the destination address of the message, including 8 bits. The address range of a single device is 1~247. The master device gates the slave device by putting the address of the slave device to be contacted into the address field in the message. When the slave device sends a response message, it puts its address in the address field of the response, so that the master device knows which device responded. Address 0 is used as a broadcast address so that all slave devices can recognize it.

(2) Functional domain

When a message is sent from the master device to the slave device, the function code field will tell the slave device what actions need to be performed. For example, to read the input switch status, read the data content of a set of registers, read the diagnostic status of the slave device, allow to call in, record, and verify the program in the slave device. When the slave responds, it uses the function code field to indicate whether it responds normally (without errors) or if some kind of error occurs (called a dissenting response). For normal response, the slave device only responds to the corresponding function code. After the master device application receives a response to the objection, the typical process is to resend the message, or diagnose the message sent to the slave device and report it to the operator.

(3) Data field

The data field is composed of two sets of hexadecimal numbers, ranging from 00 to FF. The data field sent from the master device to the slave device contains the parameters required by the slave to execute the host function code, such as the address

of the register of the processing object, the number of items to be processed, and the actual number of data bytes in the field. For example, if the master device needs to read a set of holding registers (function code 03) from the device, the data field specifies the starting register and the number of registers to be read. If the master device writes a group of slave device registers (function code 16, namely 10H), the data field specifies the starting register to be written and the number of registers to be written, the number of data bytes in the data field, and the register to be written data. If no error occurs, the data field returned from the device contains the requested data. If an error occurs, this field contains an objection code that the main device application can use to determine the next step. The data field may not exist in a certain message (0 length). For example, the master device requires the slave device to respond to the communication event record (function code 0B H), and the slave device does not need any additional information.

When transferring a 2-byte data, the high byte (MSB) will be transferred first, and then the low byte (LSB). This is just the opposite of DeviceNet's transmission method.

(4) CRC check field

The CRC field detects the content of the entire message, including two bytes, containing a 16-bit binary value. It is calculated by the transmission device and added to the message. The receiving device will recalculate the CRC of the received message and compare it with the value in the received CRC field. If the two values are different, there is an error. When CRC is added to the message, the low byte is added first, then the high byte.

B.2 Modbus TCP protocol data format

The link layer check mechanism of TCP/IP protocol and Ethernet can guarantee the correctness of data packet transmission. Therefore, the CRC-16 or LRC check field no longer exists in Modbus TCP packets, but a Modbus application frame header needs to be added (MBAP). It can explain the parameters and functions of Modbus. Each TCP/IP message can contain only one Modbus frame.

In Modbus TCP ADU, the MBAP header occupies 7 bytes (including 4 subfields), and the transaction identifier TI (Transaction Identifier), protocol identifier PI (Protocol Identifier), length identifier L (Length) (occupies 2 Byte, indicating the total length of the Protocol Identifier and Data fields) and the unit identifier UI (Unit Identifier). TI occupies 2 bytes and is used to identify the order of Modbus frames. PI occupies 2 bytes and is used to confirm the application layer protocol. The UI occupies 1 byte and is used to identify the Modbus device unit. The function code occupies 1 byte and can be divided into two types: bit operation and 16-bit word operation. The function code indicates the operation to be performed. For example, the function code 15 represents writing multiple bit registers, and the function code 06 represents writing to an independent 16-bit word register. The data field can be up to 248 bytes, and its specific format is related to the function code. When the client sends the request data, the data field gives the starting address (2 bytes) and the number (1 byte) of the register to be operated; when the server sends the response data, the data field gives

the operated register Number (1 byte) and the status value of each register. Figure B.1 shows the comparison of Modbus and Modbus TCP data frame format.

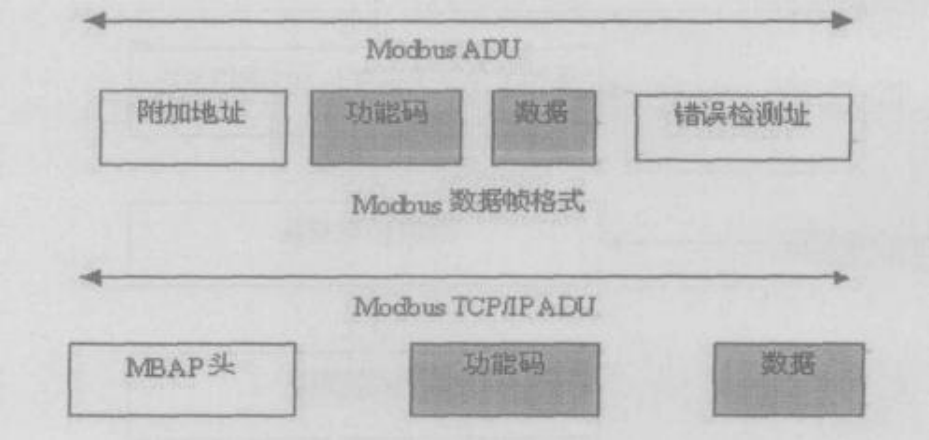


Figure B.1 Modbus and Modbus TCP/IP frame format

The ADU data unit specification of Modbus TCP is shown in Table B.1.

	description	Occupied bytes
MBAP header	Transmission identification code high bit Hi	1
	Transmission identification code low bit Lo	1
	Protocol identifier	2
	Length identifier	2
	Unit identifier	1
Modbus request	function code	1
	Start address	2
	Number of registers	2

Table B.2 Modbus TCP ADU data unit specification

Before transmitting data via Modbus TCP, a TCP/IP connection needs to be established between the client and the server. The server uses port 502 as the Modbus TCP connection port. The establishment of Modbus TCP connection is usually realized automatically by the software protocol of TCP/IP Socket interface, so it is completely transparent to the application. Once the TCP/IP connection between the client and server is established, the same connection can be used to transmit any amount of user data in the required direction. The client and server can also establish multiple TCP/IP connections at the same time. The maximum number of connections depends on the specifications of the TCP/IP interface.

When a device sends a request, its corresponding device will respond. The data format of the response is shown in Table B.2.

byte	Response data
Byte0、Byte1	Transmission identification code=0 (copy the data in response)
Byte2、Byte3	Protocol identifier
Byte4	Length identifier high byte = 0
Byte5	Length identifier low byte (identifies how many bytes follow)
Byte6	Unit identifier (slave device address)
Byte7	Modbus function code
Byte8	data

Table B.3 Modbus TCP response data format

B.3 Modbus common function codes

Among the function codes of Modbus message frames, 01, 02, 03, 04, 05, 06, and 16 function codes are more commonly used, and they can be used to read and write the digital and analog quantities of the slave.

The correspondence between the Modbus standard address and each function code is shown below.

Modbus 标准地址	数据	功能码
00001-0xxxx	DO	01、05、15
10001-1xxxx	DI	02
30001-3xxxx	AI	04
40001-4xxxx	保持寄存器	03、06、16

The following takes communication in RTU transmission mode as an example to introduce these function codes in
detail.

功能码	名称	功能说明
01	读取线圈状态	取得一组线圈的当前状态(ON/OFF)
02	读取输入状态	取得一组开关输入的当前状态(ON/OFF)
03	读取保持寄存器	在一个或多个保持寄存器中取得当前的二进制值
04	读取输入寄存器	在一个或多个输入寄存器中取得当前的二进制值
05	强置单线圈	强置一个逻辑线圈的通断状态
06	预置单寄存器	把具体二进制值装入一个保持寄存器
07	读取异常状态	取得 8 个内部线圈的通断状态
08	回送诊断校验	把诊断校验报文送从机，通信诊断
16	预置多寄存器	把具体二进制值装入一串连续的保持寄存器
128~255	保留	用于异常应答

The following is the data packet format sent and received by the master and slave of the 7 Modbus RTU commands. The rest of the commands can refer to the format.

(1) Function code: 01H

Code function: read coil status (DO)

Note: Read the ON/OFF status of the slave DO, it does not support broadcasting.

Query: The query information specifies the starting coil address and coil quantity to

be read. The starting address of the coil is 0000H, and the addressing addresses of 1-16 coils are divided into 0000H-0015H.

Host sends	Bytes	Example (Hex)	Annotate
Slave address	1Byte	01	Communicate with No. 01 slave
function code	1 Byte	01	Read coil status
Coil first address	2 Bytes	00 00	The first address of the coil is 0000H
Number of coils	2 Bytes	00 08	Read 8 coils in a row
CRC	2 Bytes	3D CC	CRC check code of the first 6 bytes

Response: The status of each coil in the response message corresponds to the value of each bit in the data area, that is, each DO occupies one bit (1 = ON, 0 = OFF). The data area is DO7, DO6...DO0 from high to low.

Slave loopback	Bytes	Example (Hex)	Annotate
Slave address	1Byte	01	Communicate with No. 01 slave
function code	1Byte	01	Read coil status
Number of data bytes	1Byte	01	1 byte
data	1Byte	02	Binary is 0000 0010, DO1 is ON
CRC	2Bytes	D0 49	CRC check code of the first 4 bytes

(2) Function code: 02H

Code function: read input status (DI)

Note: Read the ON/OFF status of the slave DI, it does not support broadcasting.

Query: The query information specifies the input start address and the number of input signals to be read. The input addressing start address is 0000H, and the addresses corresponding to inputs 1-16 are 0-15.

Host sends	Bytes	Example (Hex)	Annotate
Slave address	1Byte	01	Communicate with No. 01 slave
function code	1Byte	02	Read input status
Enter the first address	2Bytes	00 00	Enter the first address as 0000H
Number of registers	2Bytes	00 08	Continuously read 8 input ports
CRC	2Bytes	79 CC	CRC check code of the first 6 bytes

Response: The status of each input port in the response message corresponds to the value of each bit in the data area, that is, each DI occupies one bit (1 = ON, 0 = OFF). The data area from high to low is DI7, DI6...DI0.

Slave loopback	Bytes	Example (Hex)	Annotate
Slave address	1Byte	01	Communicate with No. 01 slave
function code	1Byte	02	Read input status
Number of data bytes	1Byte	01	1 byte
data	1Byte	81	Binary is 1000 0001, DI7 and DI0 are ON

CRC	2Bytes	61 E8	CRC check code of the first 4 bytes

(3) Function code: 03H

Code function: read holding register

Note: Reading the binary data of the slave holding register does not support broadcasting.

Query: The query information specifies the starting address of the register to be read and the number of registers. The starting address for register addressing is 0000H, and the addresses corresponding to registers 1-16 are 0-15.

Host sends	Bytes	Example (Hex)	Annotate
Slave address	1Byte	01	Communicate with No. 01 slave
function code	1Byte	03	Read holding register data
Register first address	2Bytes	00 01	The first address of the register is 0001H
Number of registers	2Bytes	00 03	Read 3 registers continuously
CRC	2Bytes	54 0B	CRC check code of the first 6 bytes

Response: The register data in the response message is binary data, each register corresponds to 2 bytes, the first byte is high-bit value data, and the second byte is low-bit data.

Slave loopback	Bytes	Example (Hex)	Annotate
Slave address	1Byte	01	Communicate with No. 01 slave
function code	1Byte	03	Read holding register data
Number of data bytes	1Byte	06	3 registers occupy 6 bytes
Data 1	2Bytes	02 0B	Data in 0001H register
Data 2	2Bytes	00 00	Data in 0002H register
Data 3	2Bytes	00 64	Data in 0003H register
CRC	2Bytes	84 BD	CRC check code of the first 9 bytes

(4) Function code: 04H

Code function: read input register (AI)

Description: Read the binary data in the slave input register (type 3X), it does not support broadcasting.

Query: The query information specifies the starting address of the register to be read and the number of registers. The starting address for register addressing is 0000H, and the addresses corresponding to registers 1-16 are 0-15.

Host sends	Bytes	Example (Hex)	Annotate
Slave address	1Byte	01	Communicate with No. 01 slave
function code	1Byte	04	Read input register data
Enter the first address	2Bytes	00 00	The first address of the register is 0000H
Number of registers	2Bytes	00 01	Read 1 register continuously
CRC	2Bytes	31 CA	CRC check code of the first 6 bytes

Response: The register data in the response message is binary data, each register corresponds to 2 bytes, the first byte is high-bit value data, and the second byte is low-bit data.

Slave loopback	Bytes	Example (Hex)	Annotate
Slave address	1Byte	01	Communicate with No. 01 slave
function code	1Byte	04	Read input register data
Number of data bytes	1Byte	02	1 register occupies 2 bytes
Data 1	2Bytes	0F FB	Data in 0000H register
CRC	2Bytes	FD 43	CRC check code of the first 5 bytes

(5) Function code: 05H

Code function: Forced single coil (DO)

Description: Force a single coil (DO, 0X type) to be ON or OFF. When broadcasting, this function can force all the coils of the same type in the slave to be in ON or OFF state.

Query: The query information specifies the address and status of the coil that needs to be forced. The starting address of the coil is 0000H, and the addresses corresponding to registers 1-16 are 0-15. During query, a constant in the query data area specifies the ON/OFF state of the requested coil. The FF00H value requests the coil to be in the ON state, and the 0000H value requests the coil to be in the OFF state. Other values have no effect on the coil and have no effect.

Host sends	Bytes	Example (Hex)	Annotate
Slave address	1Byte	01	Communicate with No. 01 slave
function code	1Byte	05	Forced single coil
Coil address	2Bytes	00 01	Coil address is 0001H
Coil state value	2Bytes	FF 00	ON state
CRC	2Bytes	DD FA	CRC check code of the first 6 bytes

Response: The normal response to this command request is to transfer the received data as-is after the DO state changes.

Slave loopback	Bytes	Example (Hex)	Annotate
Slave address	1Byte	01	Communicate with No. 01 slave
function code	1Byte	05	Forced single coil
Coil address	2Bytes	00 01	Coil address is 0001H
Coil state value	2Bytes	FF 00	ON state
CRC	2Bytes	DD FA	CRC check code of the first 6 bytes

(6) Function code: 06H

Code function: preset single register

Note: Preset a value to a holding register (type 4X). During broadcasting, this function presets the value to all slaves of the same type register. This function can bypass the memory protection of the controller. Keep the preset value in the register valid. The preset value can only be processed by the next logic signal of the controller. If there is no register program in the control logic, the value in the register remains

unchanged.

Query: The query information specifies the type of register to be preset. The starting address for register addressing is 0000H, and the addresses corresponding to registers 1-16 are 0-15.

Host sends	Bytes	Example (Hex)	Annotate
Slave address	1Byte	01	Communicate with No. 01 slave
function code	1Byte	06	Read register data
Register address	2Bytes	00 03	The preset register address is 0003H
Register value	2Bytes	AB CD	Preset the value into the register
CRC	2Bytes	C7 6F	CRC check code of the first 6 bytes

Response: The normal response to this command request is to transfer the received data as-is after the register value state changes.

Slave loopback	Bytes	Example (Hex)	Annotate
Slave address	1Byte	01	Communicate with No. 01 slave
function code	1Byte	06	Read register data
Register address	2Bytes	00 03	The preset register address is 0003H
Register value	2Bytes	AB CD	Preset the value into the register
CRC	2Bytes	C7 6F	CRC check code of the first 6 bytes

(7) Function code: 10H (decimal is 16)

Code function: preset multiple registers

Note: Preset the data to each (4x type) register in sequence. This function code can preset the data to the same type of register in all slaves when broadcasting. It should be noted that the function code can override the memory protection of the controller, the preset value in the register has always been valid, and the contents of the register can only be processed by the next logic of the controller. When the register program is not in the control logic, The value in the register remains unchanged.

Query: The type of register to be preset is specified in the information. The starting address for register addressing is 0. The preset value of the register is specified in the query data area. The M84 and 484 controllers use 10-bit binary data, 2 bytes, and the remaining high 6 positions are 0. Other types of controllers use a 16-bit binary data with 2 bytes per register.

Host sends	Bytes	Example (Hex)	Annotate
Slave address	1Byte	01	Communicate with No. 01 slave
function code	1Byte	10	Preset multiple registers
Register first address	2Bytes	10 20	The first address of the write register is 1020H
Number of registers	2Bytes	00 03	3 consecutive registers
Number of registers	1Byte	06	3 registers occupy 6 bytes
Data 1	2Bytes	02 01	Data in register 1020H
Data 2	2Bytes	04 03	Data in register 1021H
Data 3	2Bytes	06 05	Data in register 1022H
CRC	2Bytes	BD 9B	CRC check code of the first 13 bytes

Response: Normal response returns slave address, function code, start address and preset register number.。

Slave loopback	Bytes	Example (Hex)	Annotate
Slave address	1Byte	01	Communicate with No. 01 slave
function code	1Byte	10	Preset multiple registers
Coil address	2Bytes	10 20	The first address of the write register is 1020H
Coil state value	2Bytes	00 03	3 consecutive registers
CRC	2Bytes	85 02	CRC check code of the first 6 bytes

The following is the data packet format sent and received by the master and slave of 7 Modbus TCP commands. The rest of the commands can refer to the format. The code function and description are omitted in this section, please refer to the Modbus RTU section for related content.

(1) Function code: 01H

Host sends	Bytes	Example (Hex)	Annotate
Transmission identification	2Bytes	00 00	
Agreement Logo	2Bytes	00 00	
Data length	2Bytes	00 06	There are 6 bytes after
Slave address	1Byte	01	Communicate with No. 01 slave
function code	1Byte	01	Read coil status
Coil first address	2Bytes	00 00	The first address of the coil is 0000H
Number of coils	2Bytes	00 08	Read 8 coils in a row

Slave loopback	Bytes	Example (Hex)	Annotate
Transmission identification	2Bytes	00 00	
Agreement Logo	2Bytes	00 00	
Data length	2Bytes	00 04	There are 4 bytes
Slave address	1Byte	01	Communicate with No. 01 slave
function code	1Byte	01	Read coil status
Number of data bytes	1Byte	01	1 byte
Data	1Byte	02	Binary is 0000 0010, DO1 is ON

(2) Function code: 02H

Host sends	Bytes	Example (Hex)	Annotate
Transmission identification	2Bytes	00 00	
Agreement Logo	2Bytes	00 00	

Data length	2Bytes	00 06	
Slave address	1Byte	01	Communicate with No. 01 slave
function code	1Byte	02	Read input status
Enter the first address	2Bytes	00 00	Enter the first address as 0000H
Number of registers	2Bytes	00 08	Continuously read 8 input ports

Slave loopback	Bytes	Example (Hex)	Annotate
Transmission identification	2Bytes	00 00	
Agreement Logo	2Bytes	00 00	
Data length	2Bytes	00 04	
Slave address	1Byte	01	Communicate with No. 01 slave
function code	1Byte	02	Read input status
Number of data bytes	1Byte	01	1 byte
Data	1Byte	81	Binary is 1000 0001, DI7 and DI0 are ON

(3) Function code: 03H

Host sends	Bytes	Example (Hex)	Annotate
Transmission identification	2Bytes	00 00	
Agreement Logo	2Bytes	00 00	
Data length	2Bytes	00 06	
Slave address	1Byte	01	Communicate with No. 01 slave
function code	1Byte	03	Read holding register data
Enter the first address	2Bytes	00 01	The first address of the register is 0001H
Number of registers	2Bytes	00 03	Read 3 registers continuously

Slave loopback	Bytes	Example (Hex)	Annotate
Transmission identification	2Bytes	00 00	
Agreement Logo	2Bytes	00 00	
Data length	2Bytes	00 09	
Slave address	1Byte	01	Communicate with No. 01 slave
function code	1Byte	03	Read holding register data
Number of data bytes	1Byte	06	Number of data bytes 1Byte 06 3 registers occupy 6 bytes
Data 1	2Bytes	02 0B	Data 1 2Bytes 02 0B 0001H Register data
Data 2	2Bytes	00 00	Data 1 2Bytes 02 0B 0002H Register data
Data 3	2Bytes	00 64	Data 1 2Bytes 02 0B 0003H Register data

(4) Function code: 04H

Host sends	Bytes	Example (Hex)	Annotate
Transmission identification	2Bytes	00 00	
Agreement Logo	2Bytes	00 00	
Data length	2Bytes	00 06	
Slave address	1Byte	01	Communicate with No. 01 slave
function code	1Byte	04	Read input register data
Register first address	2Bytes	00 00	The first address of the register is 0000H
Number of registers	2Bytes	00 01	Read 1 register continuously

Slave loopback	Bytes	Example (Hex)	Annotate
Transmission identification	2Bytes	00 00	
Agreement Logo	2Bytes	00 00	
Data length	2Bytes	00 05	
Slave address	1Byte	01	Communicate with No. 01 slave
function code	1Byte	04	Read input register data
Number of data bytes	1Byte	02	1 register occupies 2 bytes
Data	2Bytes	0F FB	Data in 0000H register

(5) Function code: 05H

Host sends	Bytes	Example (Hex)	Annotate
Transmission identification	2Bytes	00 00	
Agreement Logo	2Bytes	00 00	
Data length	2Bytes	00 06	
Slave address	1Byte	01	Communicate with No. 01 slave
function code	1Byte	05	Forced single coil
Coil address	2Bytes	00 01	Coil address is 0001H
Coil state value	2Bytes	FF 00	ON state

Slave loopback	Bytes	Example (Hex)	Annotate
Transmission identification	2Bytes	00 00	
Agreement Logo	2Bytes	00 00	
Data length	2Bytes	00 06	
Slave address	1Byte	01	Communicate with No. 01 slave
function code	1Byte	05	Forced single coil
Coil address	2Bytes	00 01	Coil address is 0001H
Coil state value	2Bytes	FF 00	ON state

(6) Function code: 06H

Host sends	Bytes	Example (Hex)	Annotate
Transmission identification	2Bytes	00 00	
Agreement Logo	2Bytes	00 00	
Data length	2Bytes	00 06	
Slave address	1Byte	01	Communicate with No. 01 slave
function code	1Byte	06	Read register data
Register address	2Bytes	00 03	The preset register address is 0003H
Register value	2Bytes	AB CD	Preset the value into the register

Slave loopback	Bytes	Example (Hex)	Annotate
Transmission identification	2Bytes	00 00	
Agreement Logo	2Bytes	00 00	
Data length	2Bytes	00 06	
Slave address	1Byte	01	Communicate with No. 01 slave
function code	1Byte	06	Read register data
Register address	2Bytes	00 03	The preset register address is 0003H
Register value	2Bytes	AB CD	Preset the value into the register

(7) Function code: 10H (decimal is 16)

Host sends	Bytes	Example (Hex)	Annotate
Transmission identification	2Bytes	00 00	
Agreement Logo	2Bytes	00 00	
Data length	2Bytes	00 0D	
Slave address	1Byte	01	Communicate with No. 01 slave
function code	1Byte	10	Preset multiple registers
Register address	2Bytes	10 20	The first address of the write register is 1020H
Register value	2Bytes	00 03	3 consecutive registers
Number of data bytes	1Byte	06	3 registers occupy 6 bytes
Data 1	2Bytes	02 01	Data in register 1020H
Data 2	2Bytes	04 03	Data in register 1021H
Data 3	2Bytes	06 05	Data in register 1022H

Slave loopback	Bytes	Example (Hex)	Annotate
Transmission identification	2Bytes	00 00	
Agreement Logo	2Bytes	00 00	
Data length	2Bytes	00 06	

Slave address	1Byte	01	Communicate with No. 01 slave
function code	1Byte	10	Preset multiple registers
Register address	2Bytes	10 20	The first address of the write register is 1020H
Register value	2Bytes	00 03	3 consecutive registers

Sales and Service

Shenyang Guangcheng Technology Co., Ltd.

Address: 5th Floor, No. 135-21, Changqing South Street, Hunnan District, Shenyang City, Liaoning Province China

Wechat:gckj777

Whatsapp:+8613644001762

email:sygckj@gmail.com