# **XS** series PLCopen controller

## User manual [software] (XS Studio)

Wuxi Xinje Electric Co., Ltd.

Data No. PS06 20230906EN 1.2

## Basic description

- Thank you for purchasing XS series programmable controllers.
- This document describes the software of the XS series programmable controller.
- Before using the product, please read this manual carefully, and fully understand the contents of the manual, on the premise of programming.
- Please deliver this manual to the end user.

## User notice

- Only operators with certain electrical knowledge are allowed to connect cables and other operations on the product. If the use is not clear, please consult our company's technical department.
- The examples provided in the documents are for your understanding and reference only, and do not guarantee certain actions.
- When combining this product with other products, please confirm that it complies with the relevant specifications, principles, etc.
- When using this product, please make sure that it meets the requirements and is safe.
- Please set up your own backup and safety functions to avoid possible machine failure or loss due to the failure of this product.

## Statement of responsibility

- Although the contents in this manual have been carefully checked, errors are not avoidable and we cannot guarantee completeness.
- We will often review the contents of the manual and make corrections in subsequent editions. Your comments are welcome.
- The contents described in this manual are subject to change without notice.

## Related manual

For hardware related and advanced motion control instruction applications of XS series PLC, please refer to the following manual.

- XS series PLCopen controller user manual [hardware]
- XS series PLCopen controller user manual [instruction]



Jan. 2023

# Catalog

# 1. Product introduction

## 1-1. Overview

### 1-1-1. Product introduction

XS Studio covers XSDH, XS3, XSLH, XSA and other series, providing users with intelligent automation solutions. Adopt the international standard IEC61131-3 architecture, support ladder diagram LD, structured text ST, function block diagram FBD, sequence function flow diagram SFC, control flow diagram CFC and other programming languages. Supported buses include EtherCAT, Modbus/ModbusTCP, EtherNet/IP, OPC UA(Server), and CAN.

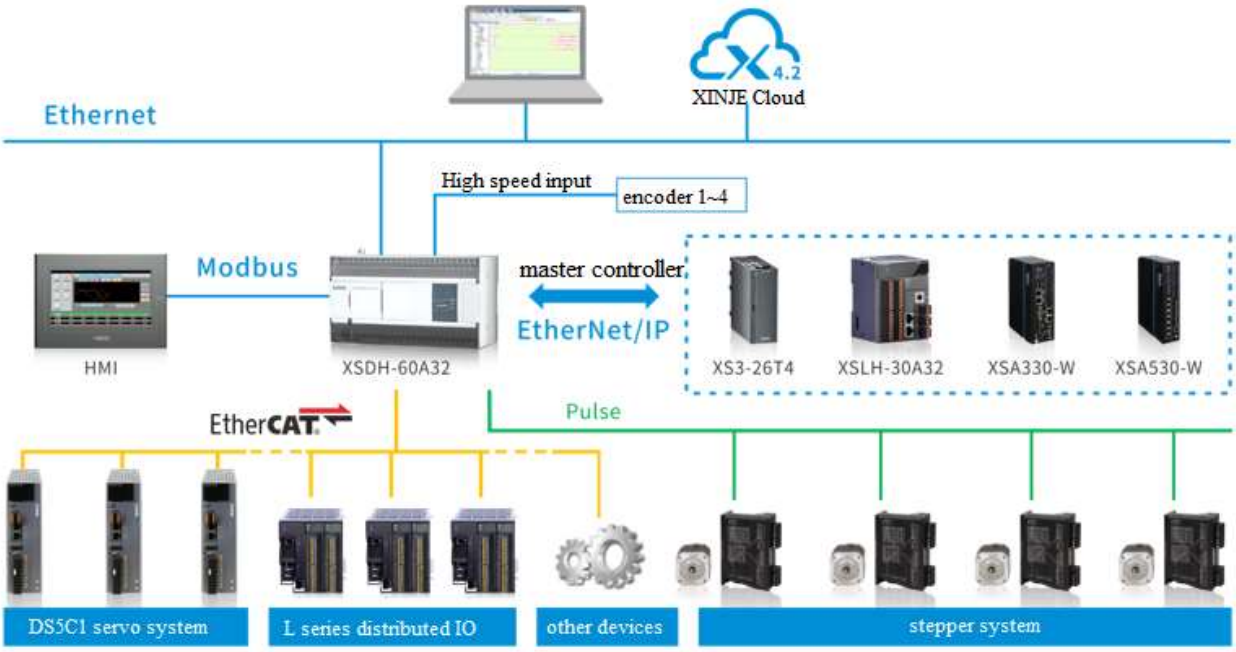
Supported extension modules:

| Model | Function |
|---|---|
| XSDH series | |
| XD-EnXmY | N-point input, M-point output, PNP/NPN type input, relay/transistor output |
| XD-E4AD | 14-Bit, 4-channel analog input (current and voltage optional), first-order coefficient adjustable, each channel can be enabled separately |
| XD-E2DA | 12-Bit, 2-channel analog output module (current and voltage optional) |
| XD-E4DA | 12-Bit, 4-channel analog output module (current and voltage optional) |
| XD-E4DA-H | 12-Bit, 4-channel analog output module (current and voltage optional); Isolation processing between channels, better anti-interference performance |
| XD-E8AD | 14-Bit, 8-channel analog input module; The first four channels are voltage (0~5V, 0~10V, -5~5V, -10~10V) input, and the last four channels are current (0~20mA, 4~20mA, -20~20mA) input. The first-order coefficient is adjustable, and each channel can be enabled separately. (Note: Hardware version H2.2 and above support bipolar) |
| XD-E8AD-A | 14-Bit, 8-channel analog current (0~20mA, 4~20mA, -20~20mA) input, first-order coefficient is adjustable, each channel can be enabled separately; (Note: Hardware version H2.2 and above support bipolar) |
| XD-E8AD-V | 14-Bit, 8-channel analog voltage (0~5V, 0~10V, -5~5V, -10~10V) input, the first-order coefficient is adjustable, each channel can be enabled separately; (Note: Hardware version H2.2 and above support bipolar) |
| XD-E12AD-V | 14-Bit, 12-channel analog voltage (0~5V, 0~10V, -5~5V, -10~10V) input, the first-order coefficient is adjustable, each channel can be enabled separately; |
| XD-E4AD2DA | 14-Bit, 4-channel analog input (current and voltage optional), current 0~20mA, 4~20mA, -20~20mA optional, voltage 0~5V, 0~10V, -5~5V, -10~10V optional; 12-Bit 2-channel analog output module (current and voltage optional), voltage 0~5V, 0~10V, -5~5V, -10~10V optional, current 0~20mA, 4~20mA optional, current first-order coefficient can be adjusted, each channel can be enabled separately; (Note: V6 and later versions of the XD-E4AD2DA module do not support -5~5V, -10~10V, -20~20mA range) |
| XD-E2AD2PT2DA | 2-channel PT100 temperature acquisition (resolution 0.1℃); 16-Bit, 2-channel analog input (current, voltage optional); 10-Bit, 2-channel analog output (voltage and current optional); Each channel can be enabled individually; |
| XD-E3AD4PT2DA | 4-channel PT100 temperature acquisition (resolution 0.1℃); 14-Bit, 3-channel analog input (0~20mA, 4~20mA optional); 10-Bit, 2-channel analog output (0~5V, 0~10V optional); Each channel can be enabled individually; |
| XD-E2TC-P | 2 channel thermocouple, support a variety of thermocouple temperature sensor analog |

| Model | Function |
|---|---|
| | input, resolution 0.1℃, 2 channels independent output PID parameters; |
| XD-E6TC-P | 6-channel thermocouple, support a variety of thermocouple temperature sensor analog input, resolution 0.1℃, 6-channel independent output PID parameters; |
| XD-E6TC-P-H | 6-channel thermocouple, support a variety of thermocouple temperature sensor input, isolation between channels, resolution 0.1°C, 6-channel transistor output, 6 groups of independent PID parameters, support self-tuning function, built-in cold end compensation; |
| XD-E6PT-P | -100~500℃, 6-channel PT100 temperature acquisition module, resolution 0.1℃, PID output; |
| XD-E4PT3-P | 4-channel PT100 (three-wire system) temperature acquisition module, resolution 0.1℃, 4-channel independent PID output; |
| XD-E1WT-D | It can collect the analog voltage signal of one pressure sensor (-20 ~ 20mV), 22-bit high-precision A/D conversion, using the A/D conversion mode of △-ΣADC, higher and faster CPU processing speed, more optimized algorithm, better anti-resonance performance, and DC24V power supply; |
| XSDH series | |
| XD-E2WT-D | It can collect the analog voltage signal of two pressure sensors (-20 ~ 20mV), 22-bit high-precision A/D conversion, using A/D conversion mode of △-ΣADC, higher and faster CPU processing speed, more optimized algorithm, better anti-resonance performance, and DC24V power supply; |
| XD-E4WT-D | Four-channel sensor analog voltage signal can be collected (-20 ~ 20mV), 22-bit high-precision AD conversion, using △-ΣADC A/D conversion mode, higher and faster CPU processing speed, more optimized algorithm, good anti-resonance performance, power supply DC24V; |
| XD-E4SSI | XD series is connected with SSI signal encoder special expansion module, one module can connect up to 4 channels at the same time, the communication speed can reach 400us/ channel; |
| XD-NES-ED | XD series PLC extended ED module, can expand 1 RS232 or RS485 communication port; (Note: Only one can be used) |
| XD-NS-BD | XD series PLC expansion BD board, RS-232 communication function; |
| XD-NE-BD | XD series PLC extended BD, bus communication function, X-NET standard interface, this BD board can also be used as RS485 communication expansion board; |
| XSLH series | |
| XL-EnXmY | N-point input, M-point output, PNP/NPN type input, input filter time adjustable, relay/transistor output (Note: -A type expansion module is horn terminal, need to be used with terminal block and special expansion cable) |
| XL-E4AD | 14-Bit 4-channel analog input (optional voltage 0~10V, 0~5V, -5~5V, -10~10V; The current can be 0~20mA, 4~20mA, -20~20mA), the first-order coefficient adjustable, each channel can be enabled separately, and the power supply is DC24V; |
| XL-E4AD2DA | 14-Bit 4-channel analog input (optional voltage 0~10V, 0~5V, -5~5V, -10~10V; Current optional 0~20mA, 4~20mA, -20~20mA); 12-Bit 2-channel analog output module (voltage and current optional 0~10V, 0~5V, -5~5V, -10~10V, 0~20mA, 4~20mA), first-order coefficient adjusted, each channel can be enabled separately, power supply DC24V; |
| XL-E4DA | 12-Bit 4-channel analog output module (optional voltage 0~10V, 0~5V, -5~5V, -10~10V; Current optional 0~20mA, 4~20mA), the first-order coefficient adjusted, each channel can be enabled separately, the power supply DC24V; |
| XL-E8AD-A | 14-Bit, 8-channel analog input (current optional 0~20mA, 4~20mA, -20~20mA), power |

| Model | Function |
|---|---|
| | supply DC24V; |
| XL-E8AD-A-S | 16-Bit, 8-channel analog input (current optional 0~20mA, 4~20mA, -20~20mA), power supply DC24V; |
| XL-E8AD-V | 14-Bit, 8-channel analog input (voltage optional 0~10V, 0~5V, -10~10V, -5~5V), power supply DC24V; |
| XL-E8AD-V-S | 16-Bit, 8-channel analog input (voltage optional 0~5V, 0~10V, -5~5V, -10~10V), power supply DC24V; |
| XL-E4TC-P | 4 channel thermocouple, support a variety of thermocouple temperature sensor analog input, resolution 0.1℃, 4 channel independent output PID parameters, power supply DC24V; |
| XL-E4PT3-P | -100~500℃, 4 channels PT100 (three-wire system) temperature acquisition, resolution 0.1℃, the module comes with PID control output function, power supply DC24V; |
| XL-E1WT-D | It can collect the analog voltage signal of one pressure sensor (-20 ~ 20mV), 24-bit high-precision A/D conversion, using the A/D conversion mode of △-ΣADC, higher and faster CPU processing speed, more optimized algorithm, better anti-resonance performance, and DC24V power supply; |
| XL-E2WT-D | It can collect the analog voltage signal of two pressure sensors (-20 ~ 20mV), 24-bit high-precision A/D conversion, using A/D conversion mode of △-ΣADC, higher and faster CPU processing speed, more optimized algorithm, better anti-resonance performance, and DC24V power supply; |
| XL-E4WT-D | It can collect the analog voltage signal of four pressure sensors (-20 ~ 20mV), 24-bit high-precision AD conversion, using A/D conversion mode of △-ΣADC, higher and faster CPU processing speed, more optimized algorithm, better anti-resonance performance, and DC24V power supply; |
| XL-ETR | This terminal resistance module is added when the number of XL series expansion modules exceeds 5 or more; |
| XL-P50-E | XL series power module, AC220V input, DC24V output, output power 50W; |
| XSLH series | |
| XL-NES-ED | XL series PLC extended ED module, can expand 1 RS232 or RS485 communication port; (Note: Only one can be used) |
| XS3 series | |
| XG-EnXmY | N-point input, M-point output, positive and negative logic can be set, input filtering time can be adjusted; The module does not require power supply, NPN&PNP input compatible; (Note: 64-point module needs to be equipped with special extension cable and terminal) |
| XG-E8TC-P | 8 channels thermocouple TC temperature acquisition, resolution 0.1℃, support a variety of thermocouple temperature sensor with analog input, the module comes with PID control output function, power supply DC24V; |
| XG-E8PT3-P | -100~500℃, 8 channels PT100 (three-wire system) temperature acquisition, resolution 0.1℃, the module comes with PID control output function, power supply DC24V; |

1-1-2. System composition



# 1-2. XS Studio overview

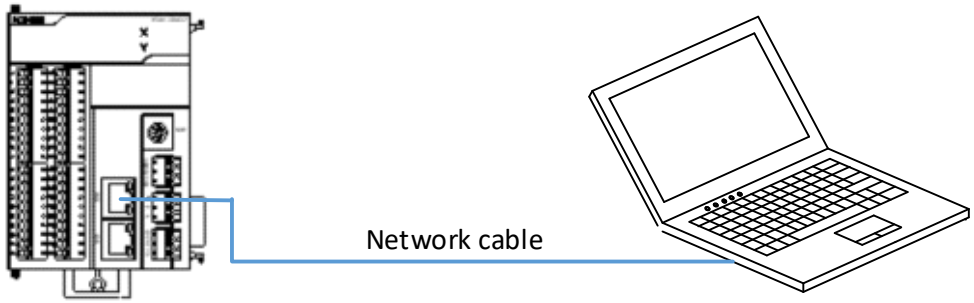
## 1-2-1. XS Studio introduction

XS Studio is a programming configuration software for the XS series based on CODESYS. Integrated PLC programming, visual HMI, safety PLC, controller real-time core, fieldbus and motion control, can provide a complete set of configuration, programming, debugging, monitoring environment, can be flexible and free to handle the powerful IEC language.

- Powerful software simulation, online debugging and program inspection capabilities, do not need to connect PLC hardware, you can complete the program debugging simulation.
- Convenient product configuration functions, which can be easily and quickly realized, including CPU configuration, IO module configuration and high-speed IO.
- Intelligent debugging function. When the user enters the wrong application code, it immediately receives a syntax error warning and error message from the compiler, so that the programmer can quickly correct it.
- Powerful motion control module. The tool kit based on PLCopen can realize single axis, multi-axis motion, electronic CAM drive, electronic gear drive, complex multi-axis CNC control, etc.

## 1-2-2. XS Studio connect to the hardware

The programming device can be connected to the PLC through the network cable, and the XS Studio software can be used to write user programs, which can be downloaded to the PLC for program monitoring and control.




## 1-2-3. Software acquisition and installation

1. System configuration requirements

Hardware and software requirements:

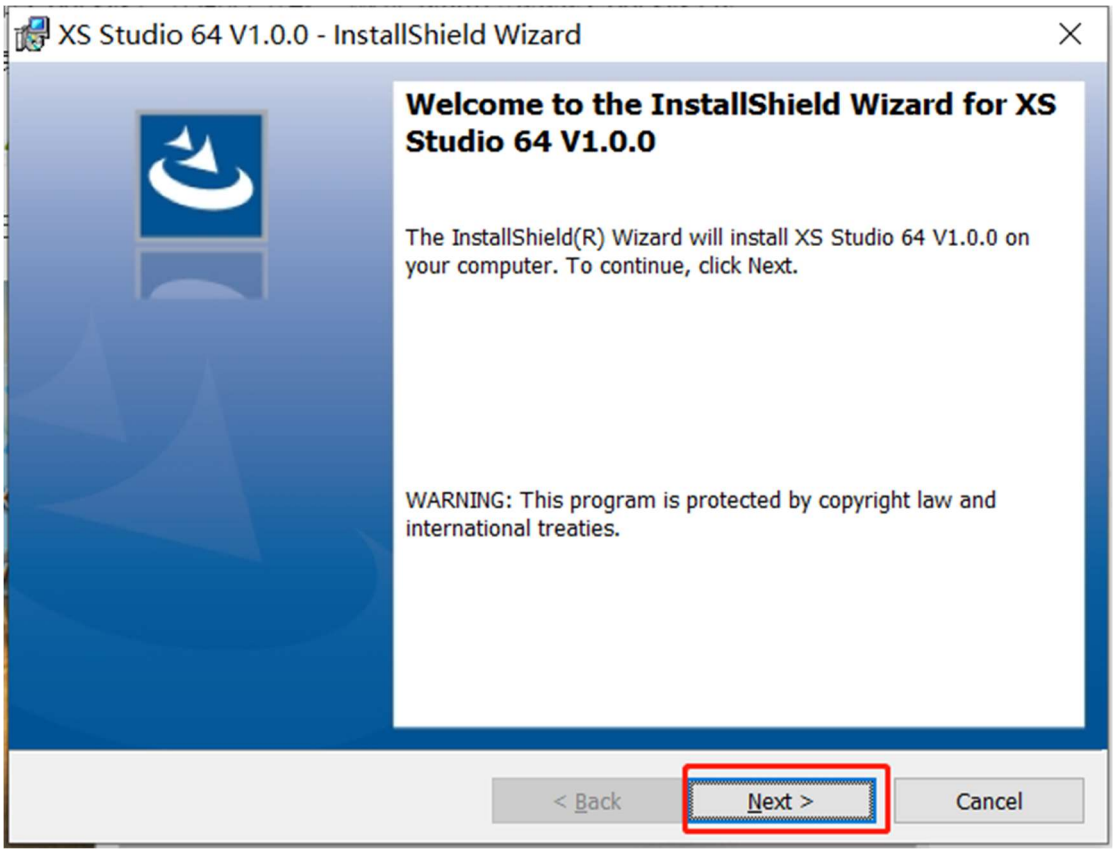
- windows 7, windows 8, or windows 10. 64-bit operating systems are recommended.
- 4GB or more memory.
- The hard disk space is greater than 12GB.

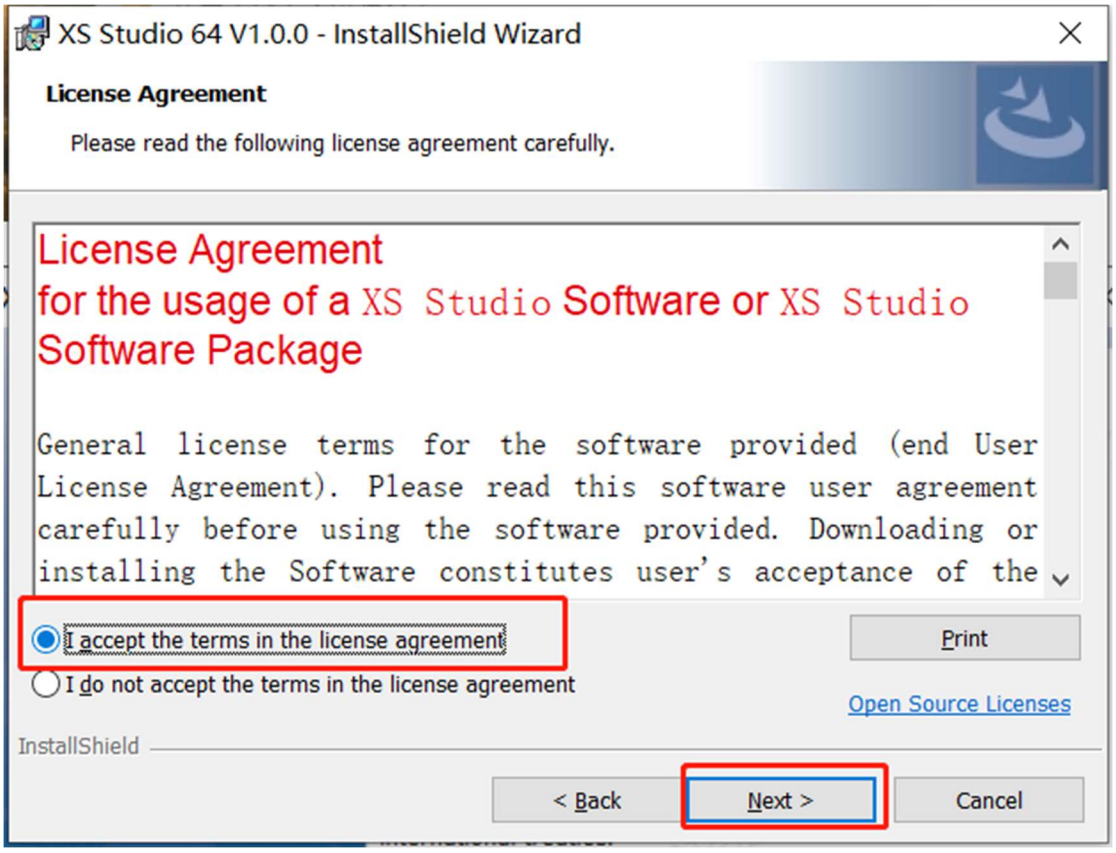
2. Software acquisition

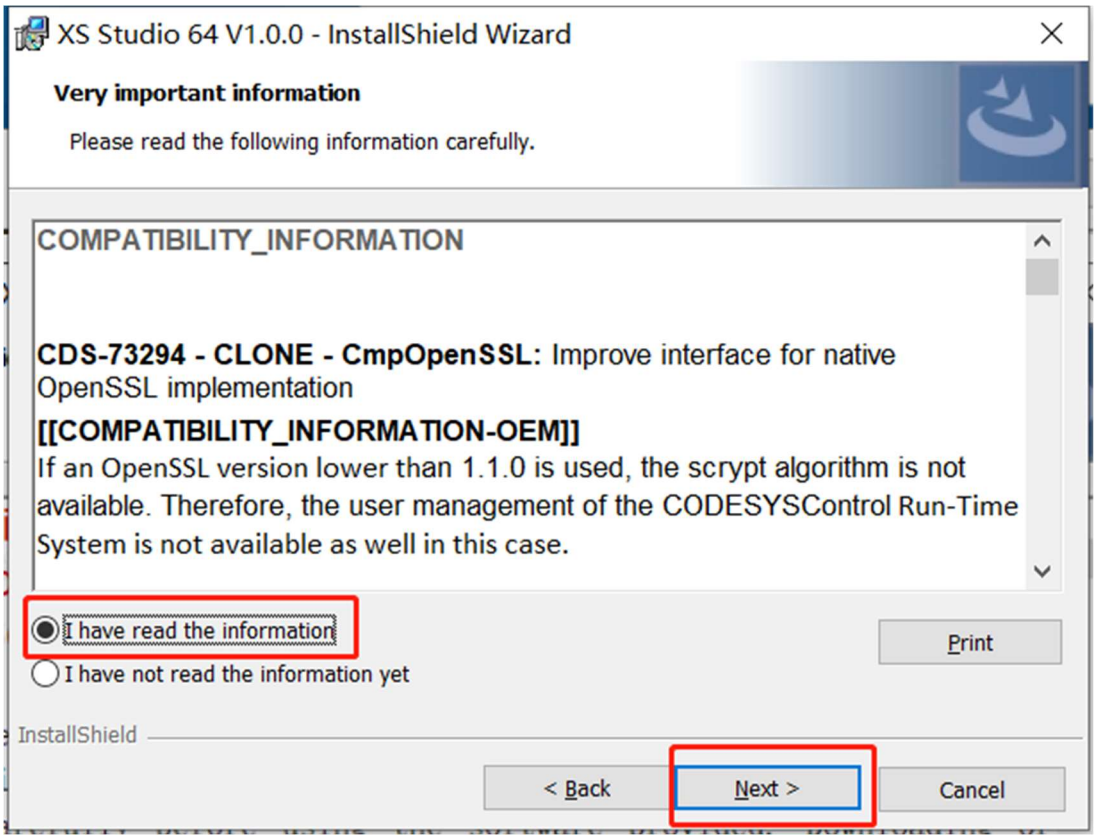
Xinje official website service and support - Download center, download website: www.xinje.com.

## 1-2-4. Software Installation Procedure

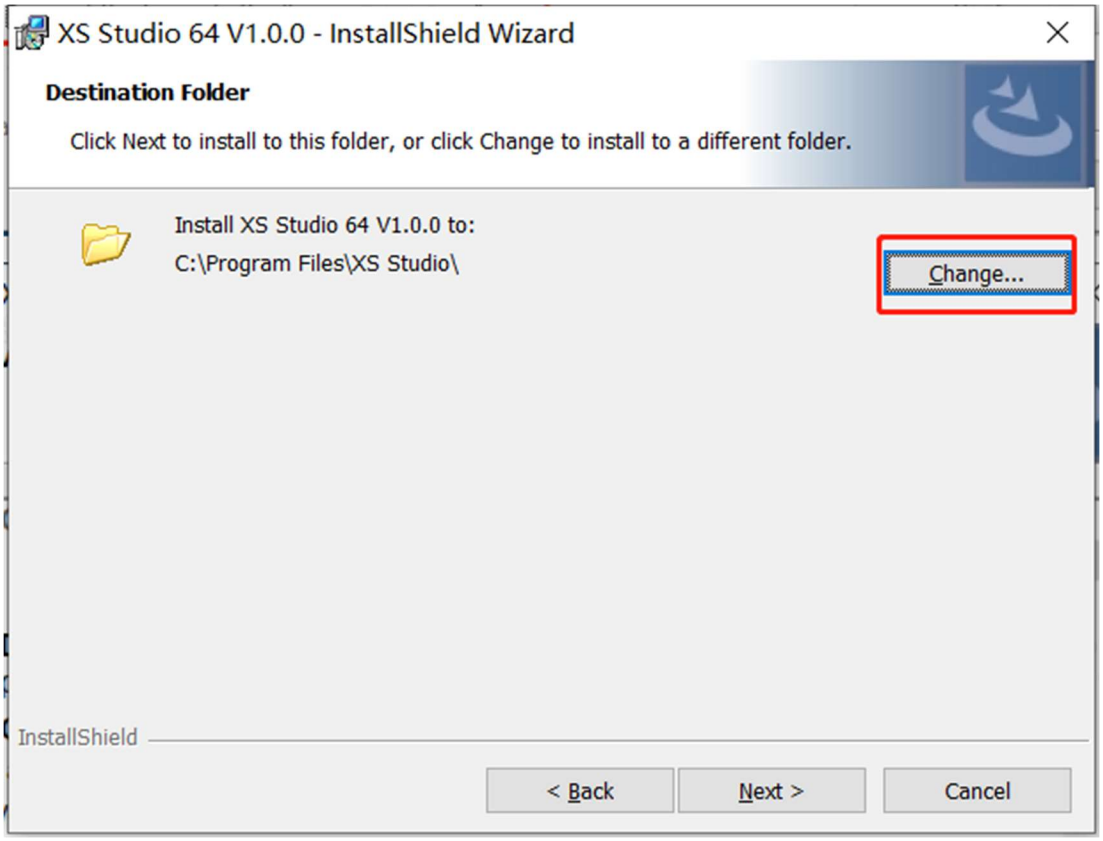
1. right-click to run as an administrator.

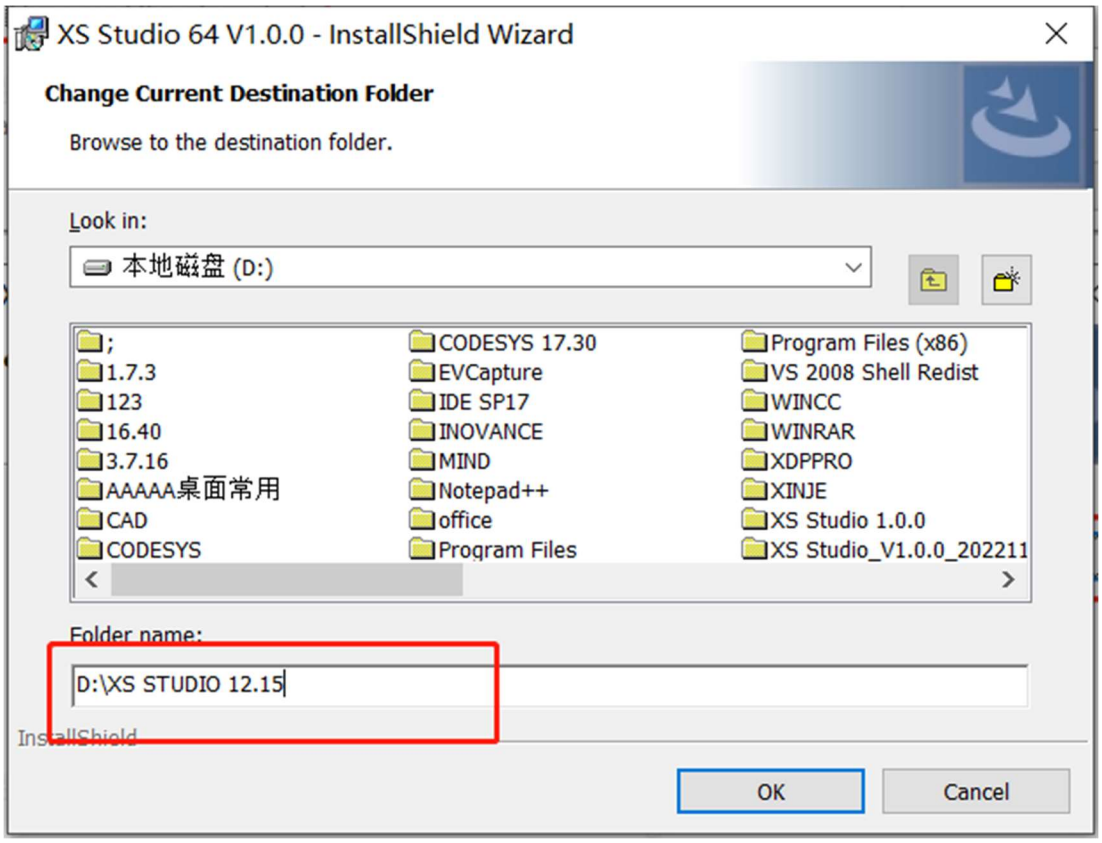

XS Studio 64 V1.0.0 - InstallShield Wizard
Welcome to the InstallShield Wizard for XS Studio 64 V1.0.0
The InstallShield(R) Wizard will install XS Studio 64 V1.0.0 on your computer. To continue, click Next.
WARNING: This program is protected by copyright law and international treaties.
< Back
Next >
Cancel


XS Studio 64 V1.0.0 - InstallShield Wizard
License Agreement
Please read the following license agreement carefully.
License Agreement
for the usage of a XS Studio Software or XS Studio Software Package
General license terms for the software provided (end User License Agreement). Please read this software user agreement carefully before using the software provided. Downloading or installing the Software constitutes user's acceptance of the
I accept the terms in the license agreement
I do not accept the terms in the license agreement
Print
Open Source Licenses
InstallShield
< Back
Next >
Cancel

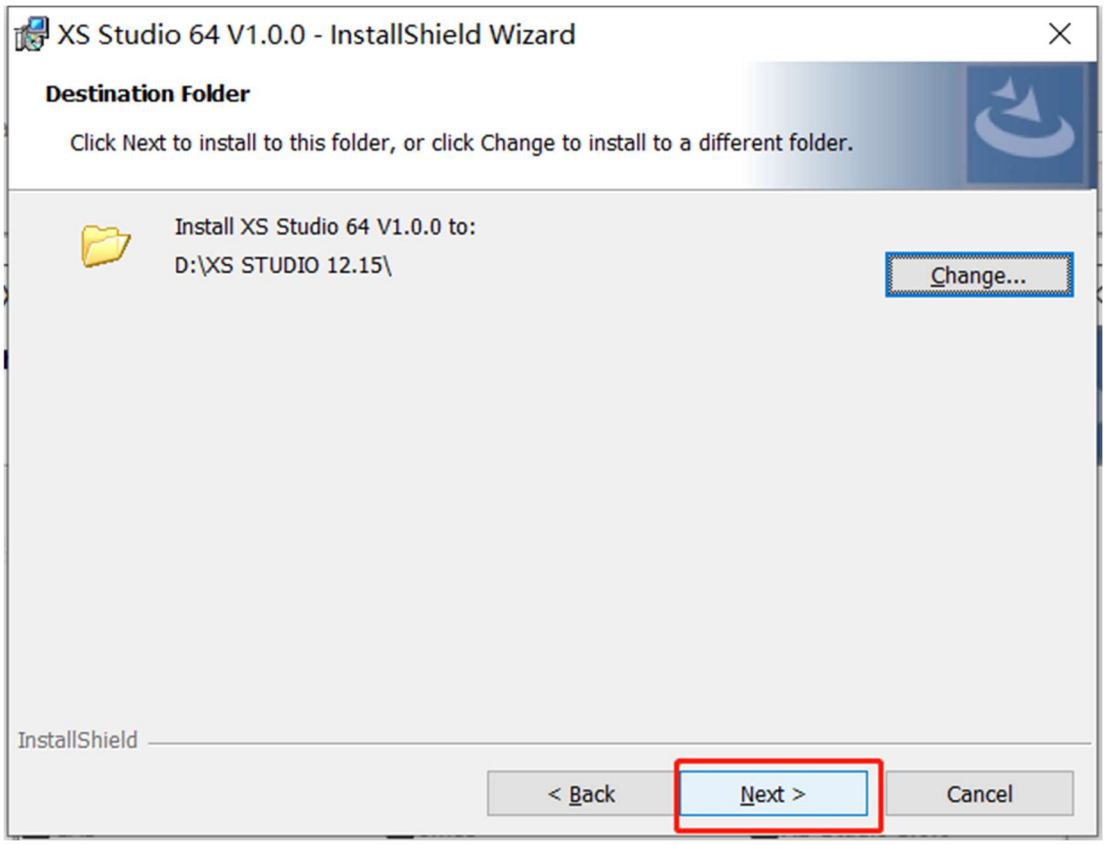



2. You are advised to install the software on a disk other than the system disk.

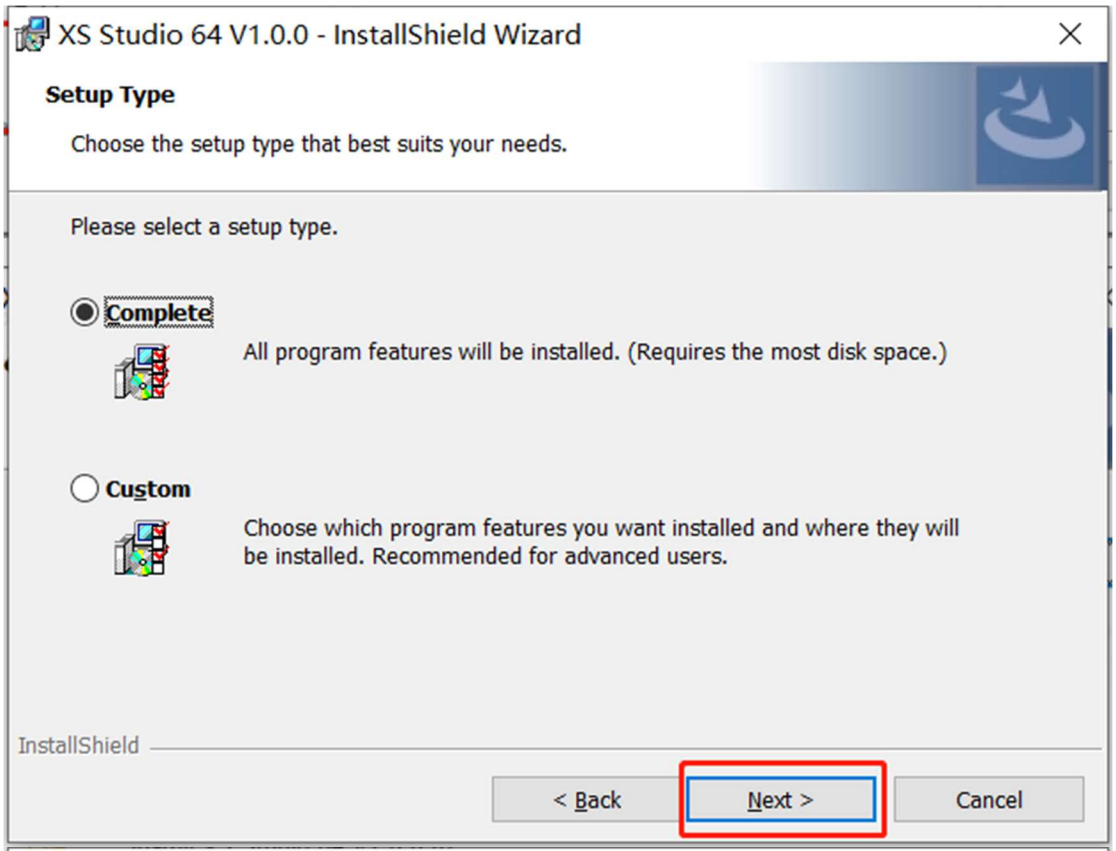


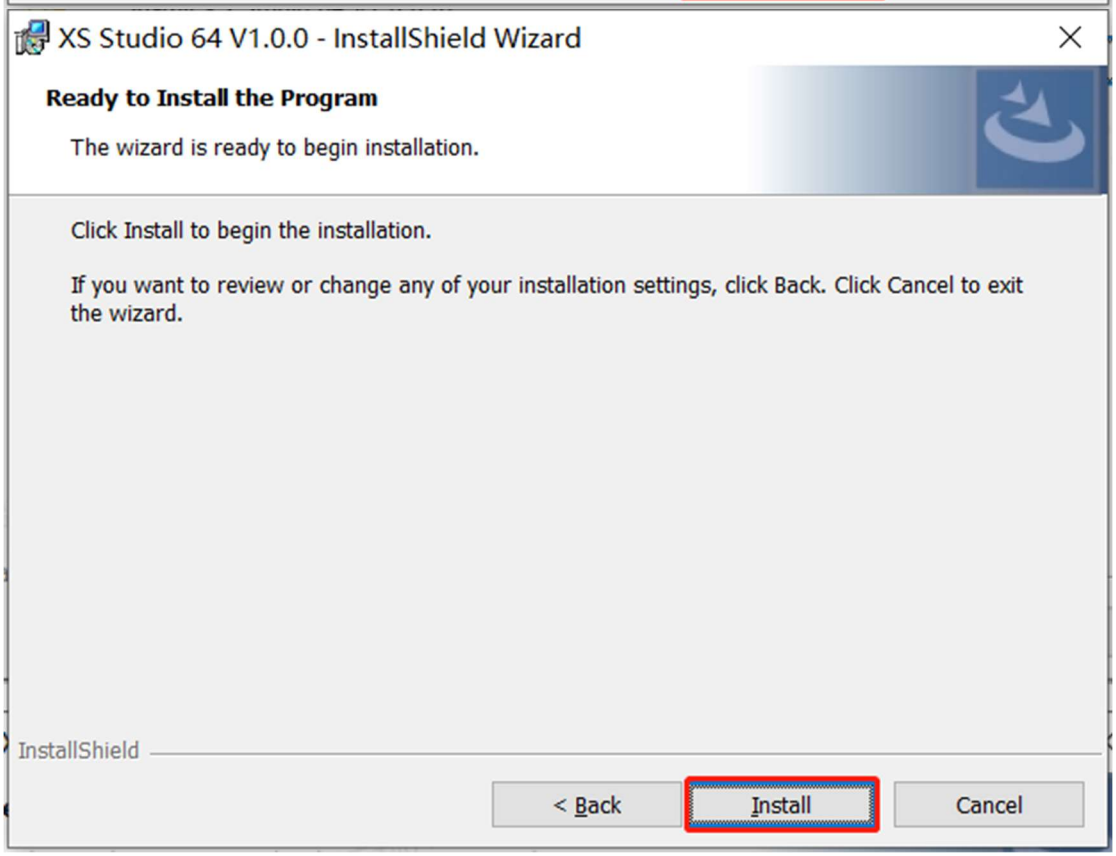



Note: The recommended installation path contains only English characters.

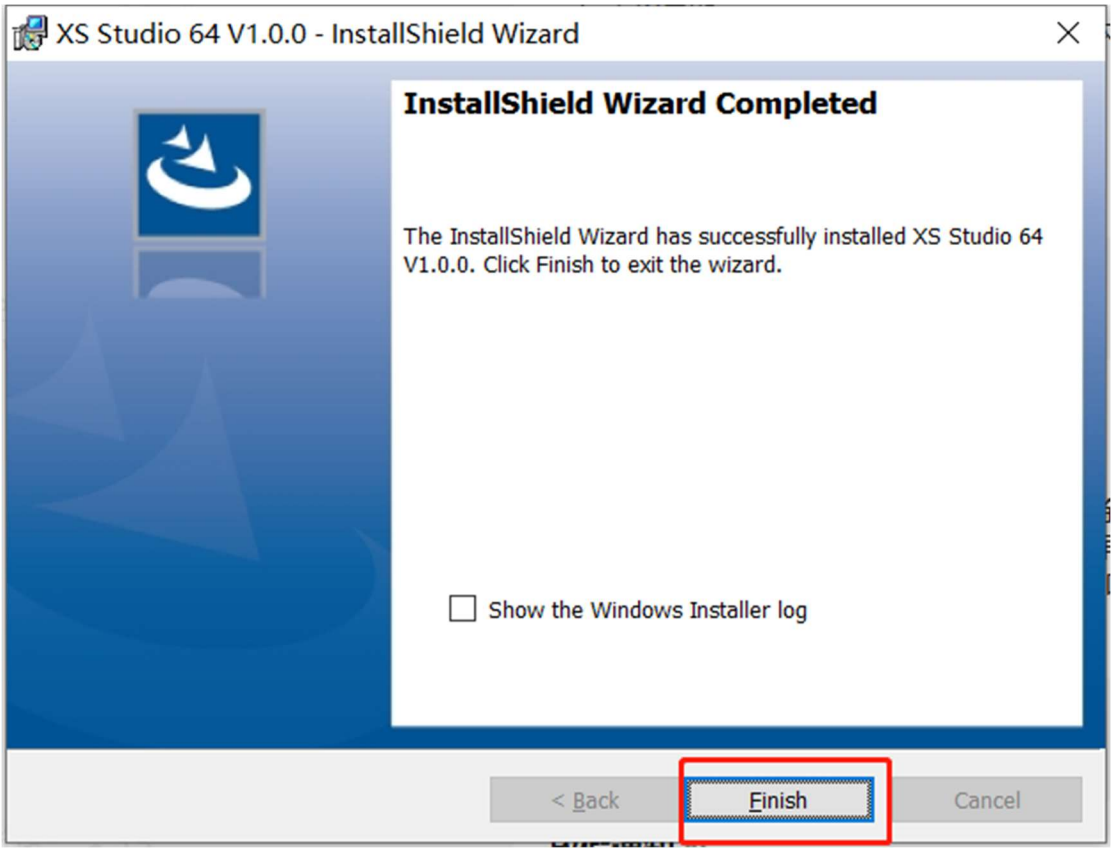



3. Complete installation




4. Installation completed


XS Studio 64 V1.0.0 - InstallShield Wizard
InstallShield Wizard Completed
The InstallShield Wizard has successfully installed XS Studio 64 V1.0.0. Click Finish to exit the wizard.
Show the Windows Installer log
< Back
Finish
Cancel

# 2. Quick start

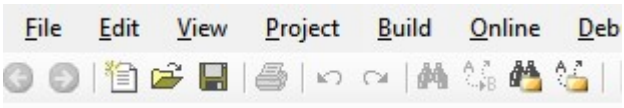
## 2-1. Start the software

Double click 

to start XS Studio software.

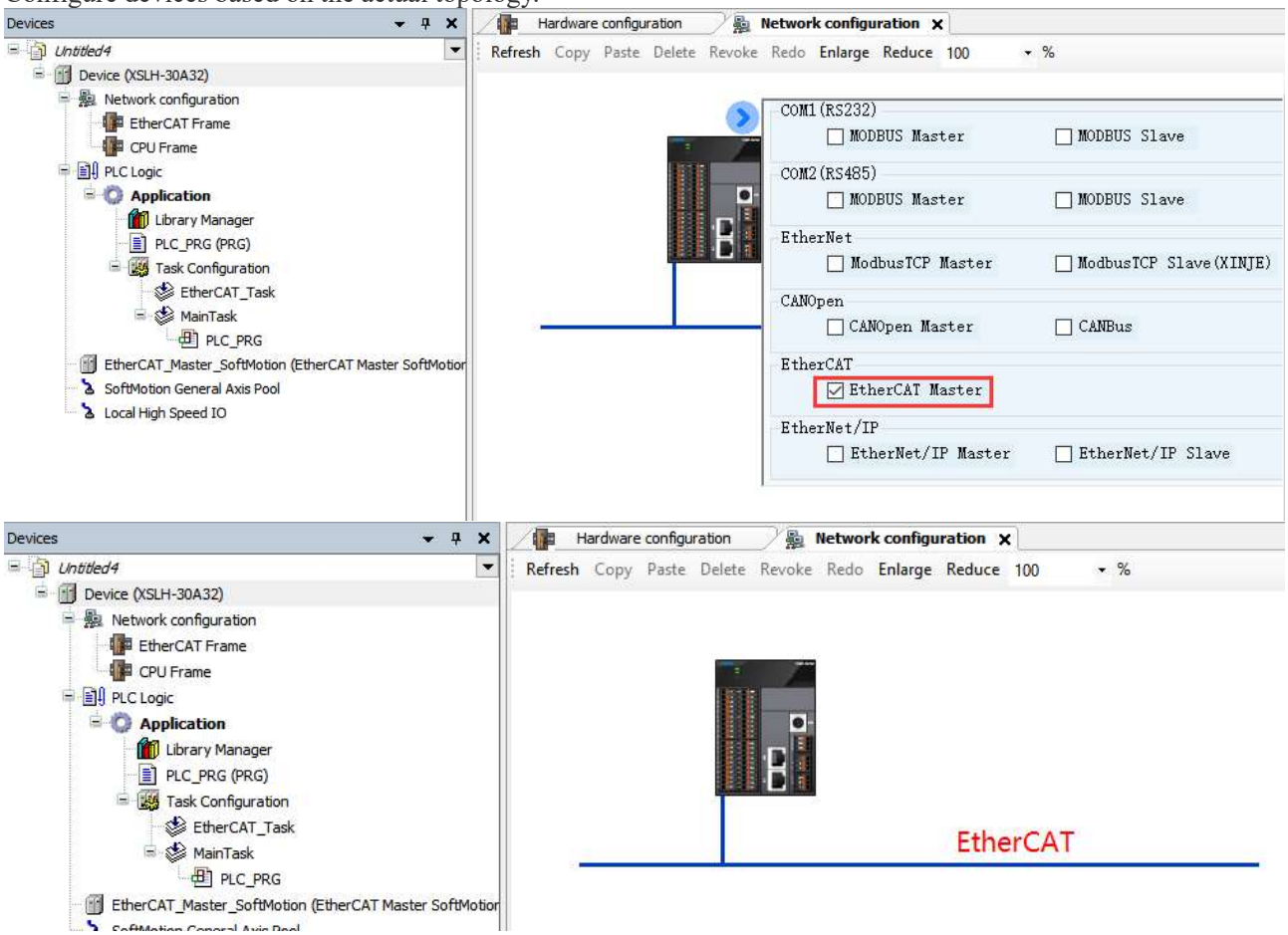
## 2-2. Interface navigation

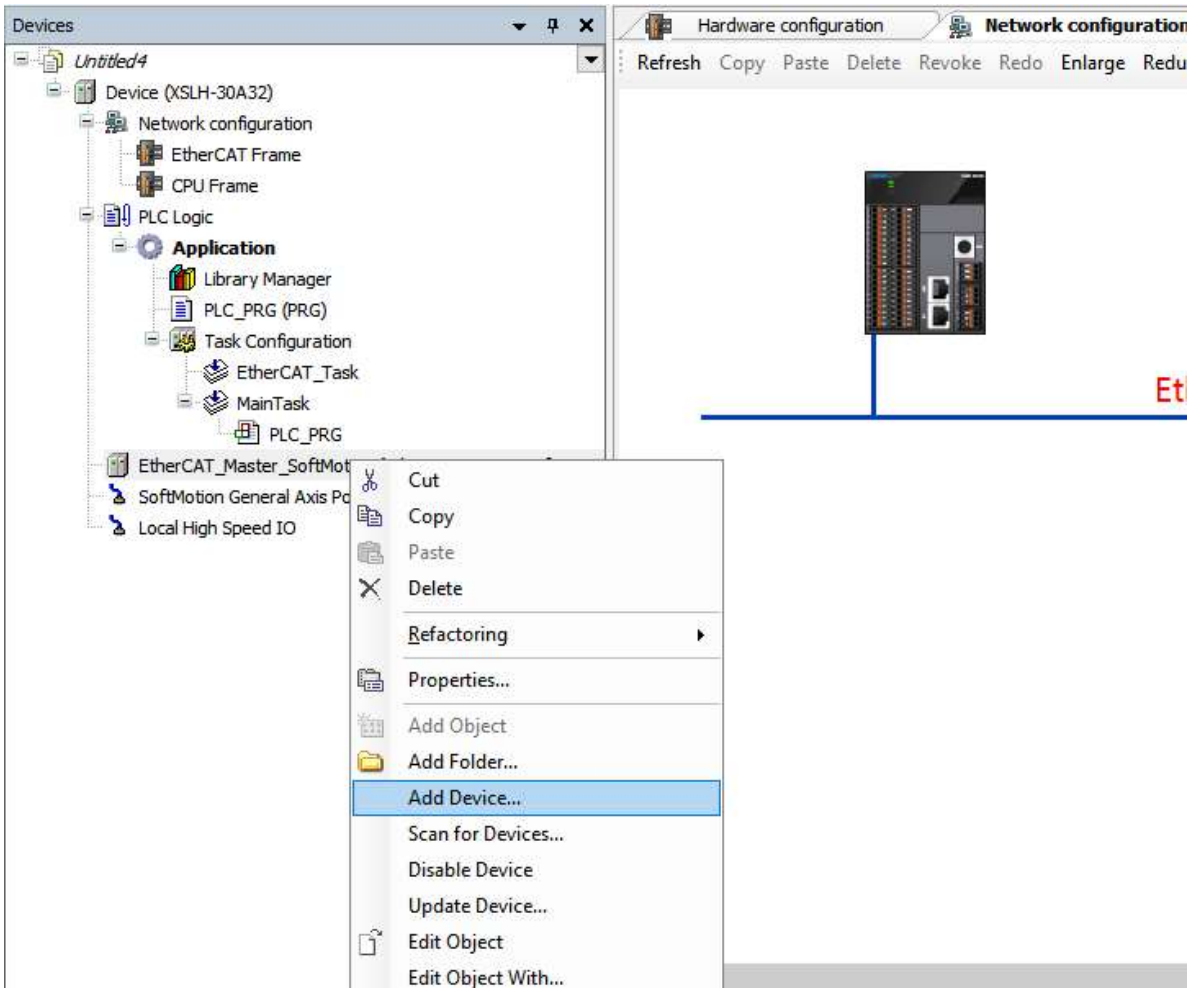
The left and right buttons represent return to the last edit position and restore to the next edit position, respectively. After the mouse click, it can help the user to locate and modify the user program position faster.



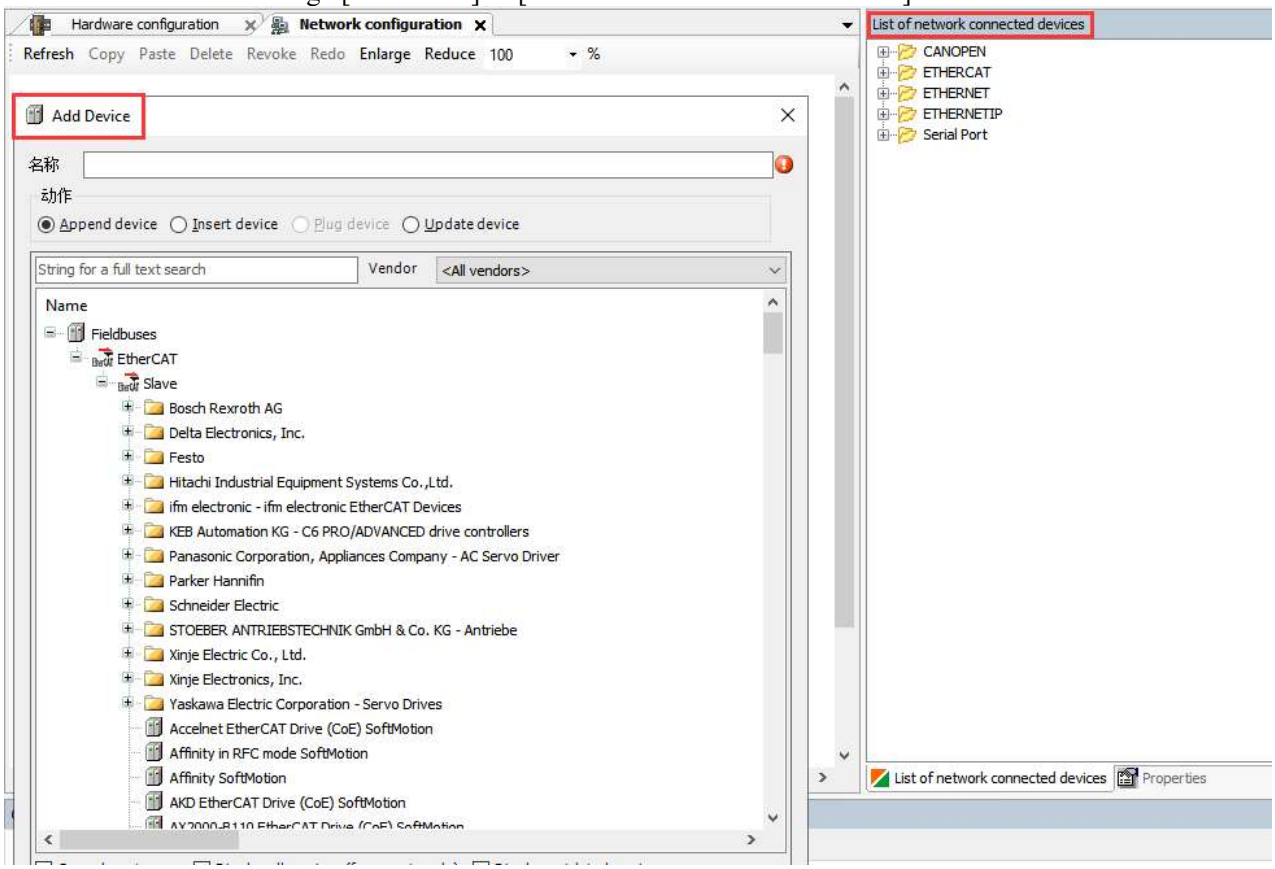

## 2-3. XS Studio programming example

Configure devices based on the actual topology.






Add the slave station through [add device] or [list of network connected devices].

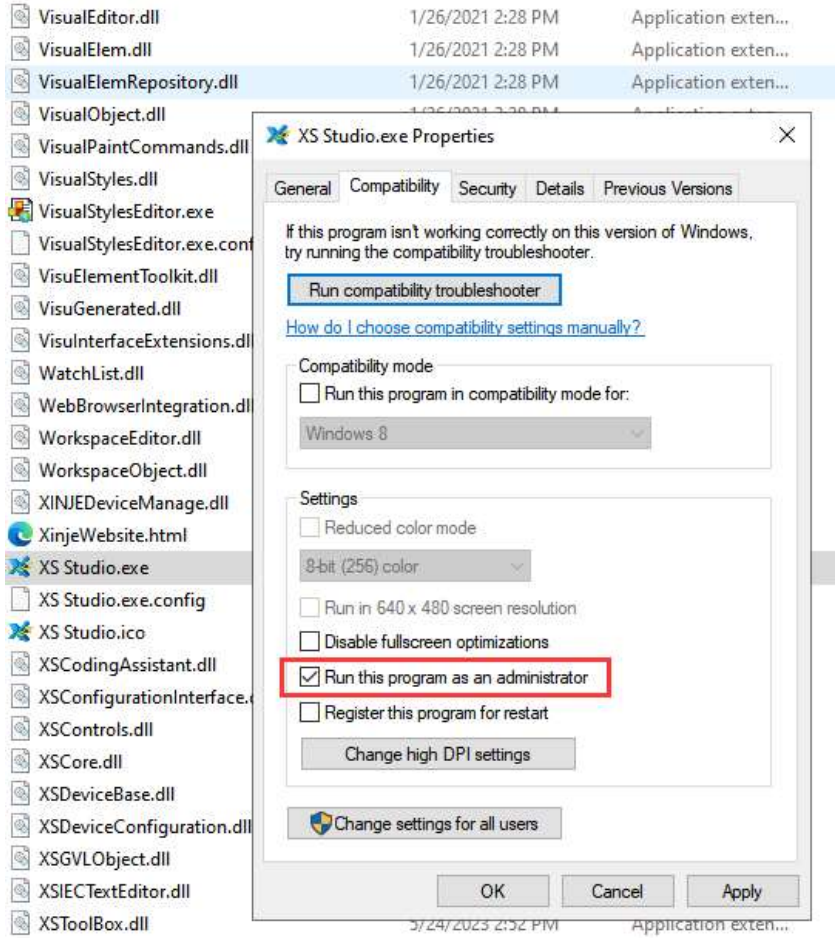


## 2-3-1. Basic programming operations

1. Start XS Studio

(1) Set administrator rights

In the Win7 system, you need to open the software with the administrator permission. Find the XS Studio.EXE file in the default installation path of XS Studio, select the file, right-click the file, and select Properties. Check the box of "Run This Program as an administrator" or "Run this program as an administrator" and click "OK" to confirm, as shown in the figure. After confirmation, the XS Studio system will automatically enter XS Studio with administrator permission by default every time XS Studio is run.

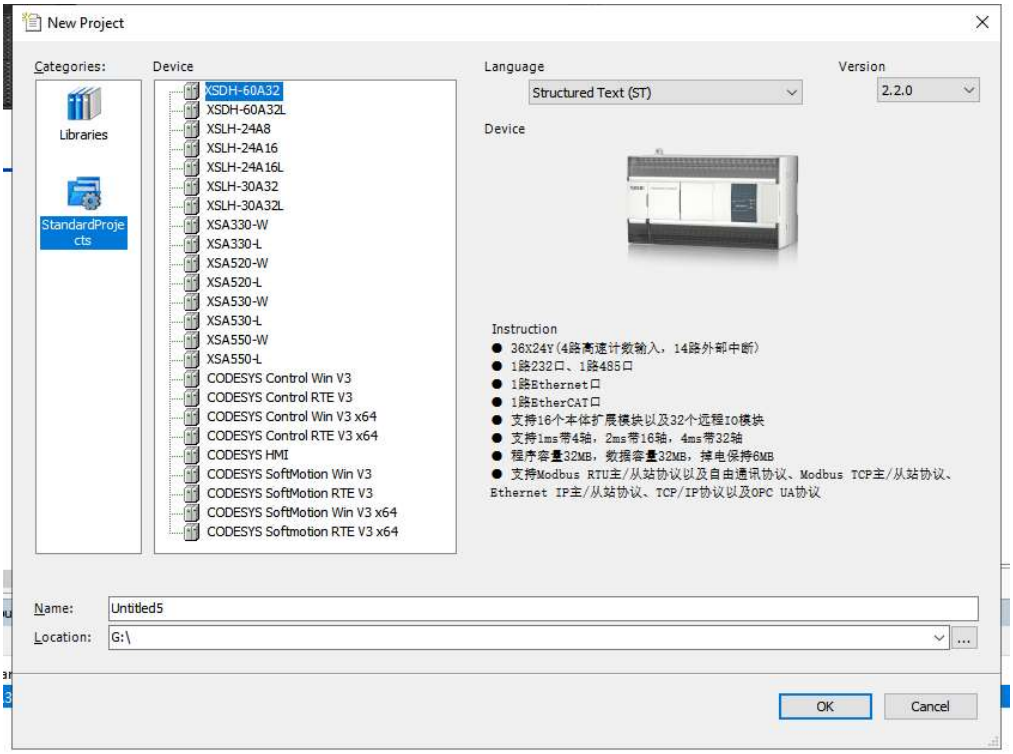



(2) Start XS Studio

Choose XS Studio >XS Studio from the Start menu or double-click the icon on the desktop to launch XS Studio.

(3) Build a new project

Click  to build a new project. Select Standard Project, select the corresponding model, select a familiar programming language, enter the project name, and select a file saving location.

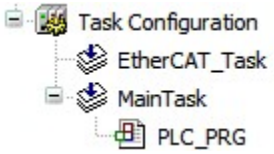



2. The establishment of PLC program file

The establishment of the PLC program file is the establishment of the running order of the running structure, the establishment of the programming mode, and even the segmentation of the data area. Before establishing the program file, the operation structure should be divided in detail, the continuous, periodic and event-triggered tasks should be determined, and the priority of periodic and event-triggered tasks should be arranged. After creating an XS Studio project, a default continuous task is automatically generated with a default program and PLC_PRG in the task.

(1) Build the task

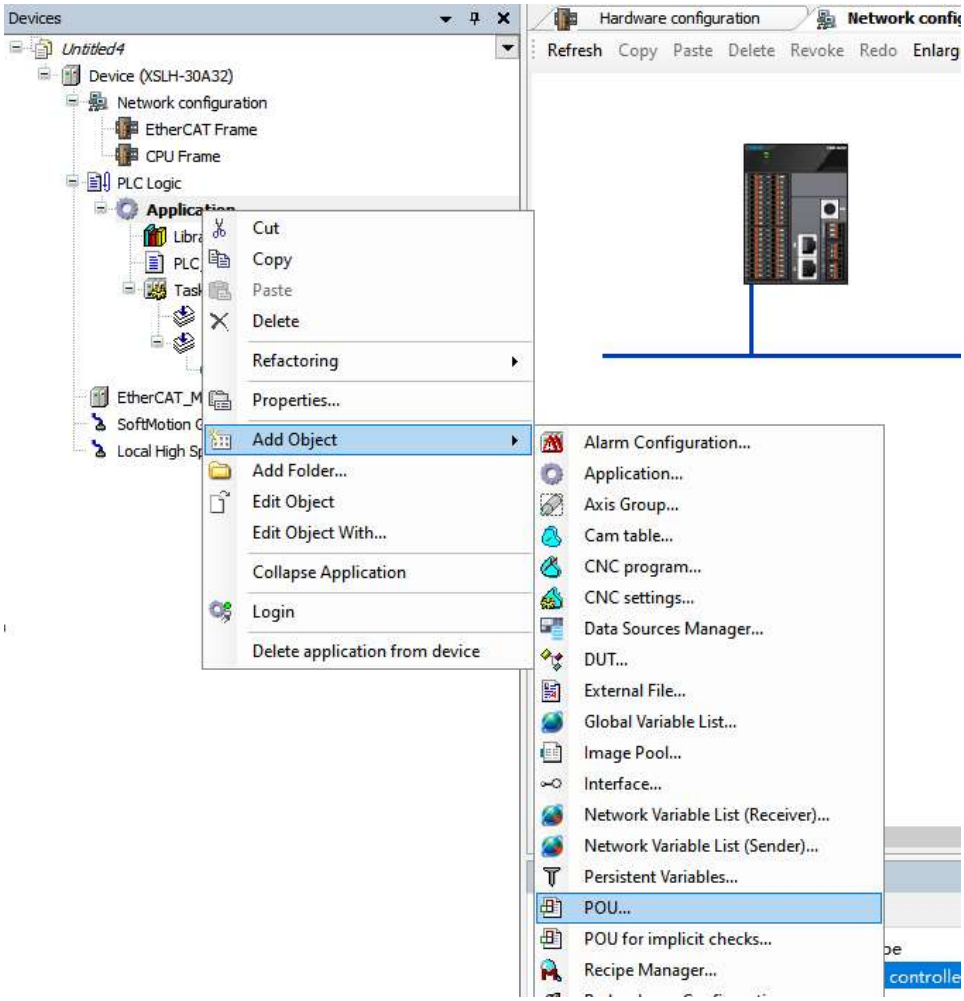
First of all, in the "task configuration" to manage the task, the usual project application can be divided into the main task, communication task. As the communication task needs to update the data source, it will be placed in a relatively high task priority level and short cycle time. In addition, if motion control is involved in the project, it will also be separated from a task and placed at the highest task priority level.

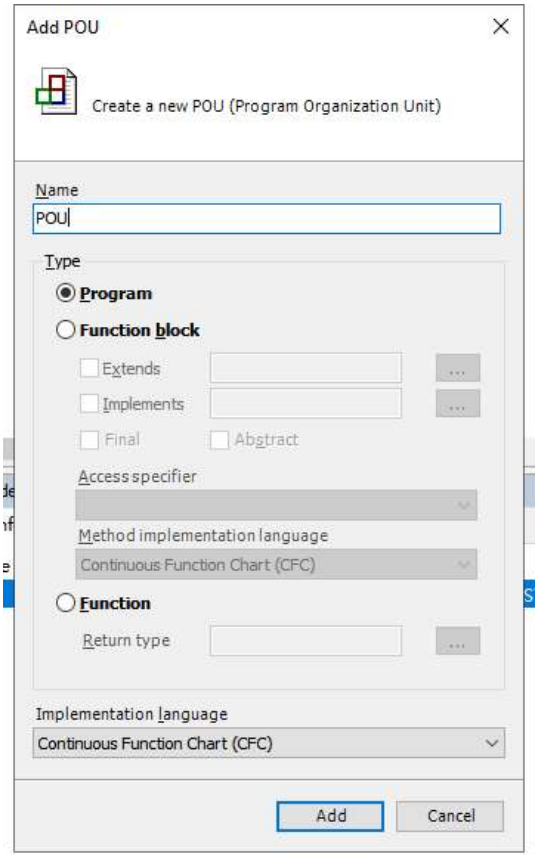


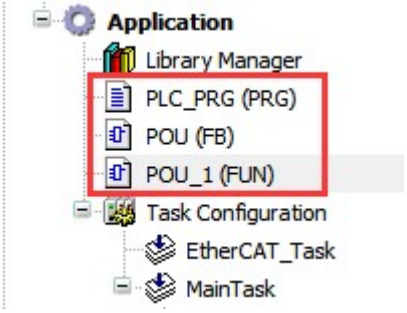

(2) Add POU

a. Custom programs/function blocks/functions

The user can use the command "Add object" from the right-click menu in the project to select "POU" program organization unit, and the dialog box as shown in the following figure will pop up. The user can choose to add programs, function blocks or functions, and the corresponding programming language can be selected from the drop-down menu. After adding, you can view the corresponding properties in the program organization unit parentheses in the project device tree on the left, FB as a function block, FUN as a function, and PRG as a program.


Devices
Untitled4
Device (XSLH-30A32)
Network configuration
EtherCAT Frame
CPU Frame
PLC Logic
Application
Cut
Copy
Paste
Delete
Refactoring
Properties...
Add Object
Add Folder...
Edit Object
Edit Object With...
Collapse Application
Login
Delete application from device
Hardware configuration
Network confi
Refresh
Copy
Paste
Delete
Revoke
Redo
Enlarg
Alarm Configuration...
Application...
Axis Group...
Cam table...
CNC program...
CNC settings...
Data Sources Manager...
DUT...
External File...
Global Variable List...
Image Pool...
Interface...
Network Variable List (Receiver)...
Network Variable List (Sender)...
Persistent Variables...
POU...
POU for implicit checks...
Recipe Manager...

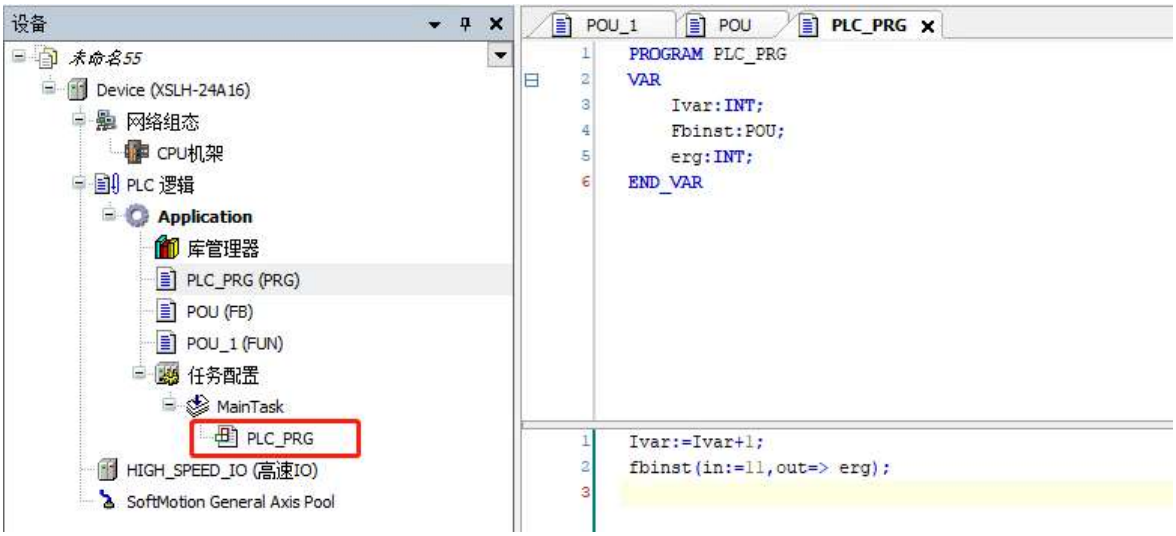

Add POU
Create a new POU (Program Organization Unit)
Name
POU
Type
Program
Function block
Extends
Implements
Final
Abstract
Access specifier
Method implementation language
Continuous Function Chart (CFC)
Function
Return type
Implementation language
Continuous Function Chart (CFC)
Add
Cancel


Application
Library Manager
PLC_PRG (PRG)
POU (FB)
POU_1 (FUN)
Task Configuration
EtherCAT_Task
MainTask

b. Declare variable

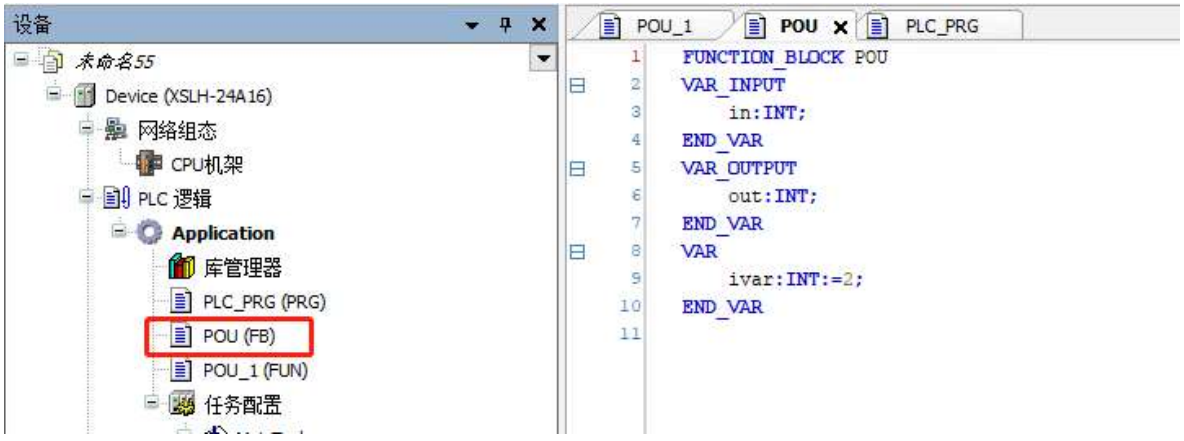
- Declare the variable in “PLC_PRG”

Double-click PLC_PRG in the device tree to automatically open it in the ST language editor of the XS Studio user interface. The language editor consists of a declaration section (upper) and an implementation section (lower), separated by an adjustable divider. The declaration section includes the line number displayed in the left border, the POU type and name (such as "PROGRAM PLC_PRG"), and the variable declaration between the keywords "VAR" and "END_VAR". As shown in the picture below:



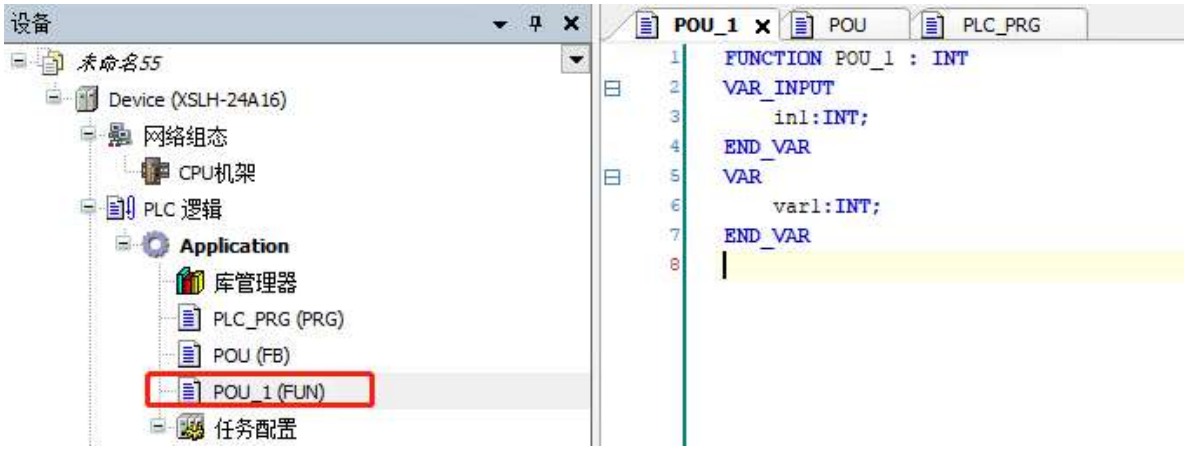

- Declares variables in function block FB

The function block language editor interface is similar to the editor interface of the program, and also includes a declaration section and a code section. All variables declared by the user are ultimately used by the program organization unit. In the variable declaration, interface variables, static variables and local variables can be declared, as shown in the following figure:




- Declare the variable inside the function FUN

A function is a basic algorithmic unit that has at least one input variable, no private data, and only one return value. A function is an organizational unit of a program without static variables. When a function is called with the same input parameters, the function always produces the same result as the function value (return value). An important feature of functions is that they cannot use internal variables to store values, unlike function blocks. The details are shown in the following figure:

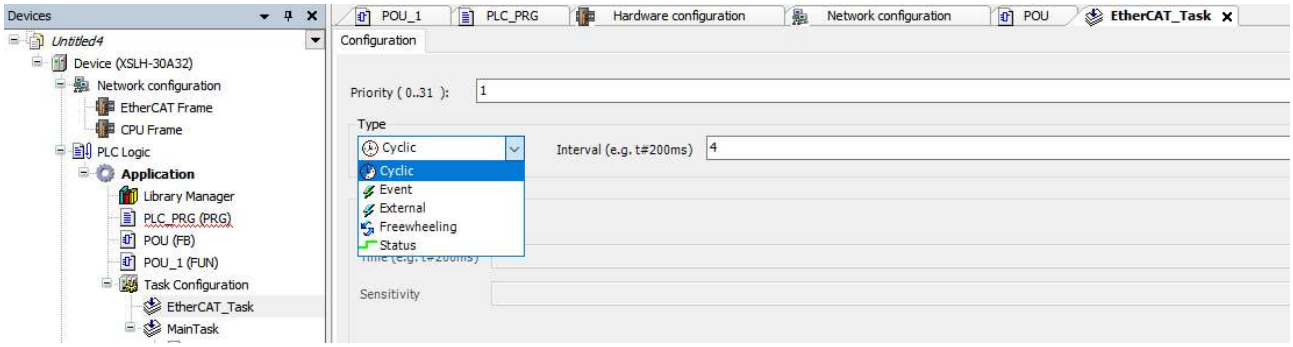



## 2-3-2. Task configuration

1. Overview

A program can be written in different programming languages. A typical program consists of many interconnected functional blocks that exchange data with each other. The execution of different parts of a program is controlled by "tasks". A "task" can be configured so that a series of programs or blocks of function execute periodically or are triggered by a specific event to begin the execution of the program. In the device tree, there is the Task Manager TAB, which in addition to declaring a specific PLC_PRG program, you can also control the execution of other subroutines within the project. A task is a property used to specify a program organizational unit at run time. It is an execution control element with the ability to invoke. Multiple tasks can be created in a task configuration, and multiple program organizational units can be invoked in a task, which can control the program execution cycle or start execution by triggering specific events once the task is set up.

In the task configuration, it is defined by name, priority, and the start type of the task. This start type can be defined by time (periodic, random) or by internal or external trigger task times, such as using the rising edge of a Boolean global variable or a particular event in the system. For each task, you can set a string of programs that are started by the task. If this task is performed during the current cycle, then these programs are processed for the length of one cycle. The combination of priority and condition will determine the timing of task execution. The task setting interface is shown below:




When the task configuration has the following attributes, the programmer should follow the following rules:

- The maximum number of loop tasks is 100.
- The maximum number of freewheeling tasks is 100.
- The maximum number of event-triggered tasks is 100.
- Depending on the target system, PLC_PRG may be executed as a free program in any case without being inserted into the task configuration.

Processing and invoking programs are executed in a top-down order within the task editor.

2. Task Priority

The priority of tasks in XS Studio can be set, and a total of 32 levels can be set (a number between 0 and 31, 0 is

the highest priority, 31 is the lowest priority). When a program is executing, the task with a higher priority takes precedence over the task with a lower priority. The task with a higher priority 0 can interrupt the execution of the program with a lower priority in the same resource, so that the execution of the program with a lower priority is slowed down.

**Note: When task priority levels are assigned, do not assign tasks with the same priority. If there are other tasks trying to precede tasks with the same priority, the results can be uncertain and unpredictable.**

If the type of the task is Cyclic, the task is executed according to the time in Interval, as shown in the following figure:


For example:

Suppose there are three different tasks, each corresponding to three different priority levels, the specific allocation is as follows:

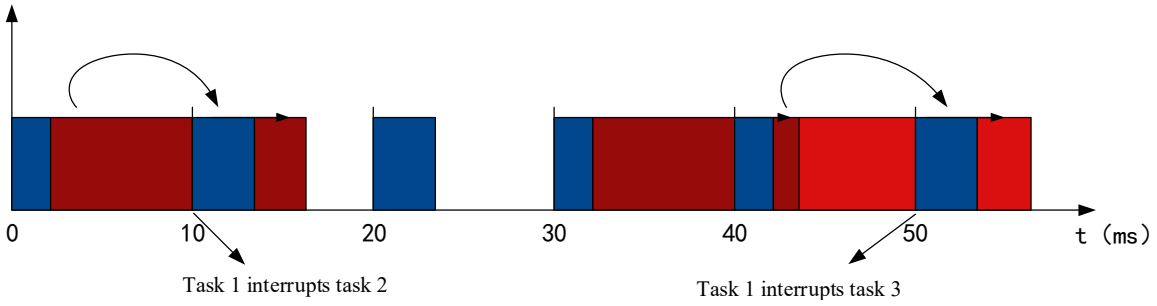
- Task 1 has priority 0 and cycle time 10ms;
- Task 2 has priority 1 and cycle time of 30ms;
- Task 3 has priority 2 and cycle time 40ms.

The sequence of each task in the controller is shown in the following figure: 0 to 10ms: Task 1 (with the highest priority) is executed first. If the program is completed within the current period, task 2 is executed within the remaining period. However, if task 2 is not completely executed at this time, but the time has reached the 10ms, because task 1 is executed every 10ms and has a higher priority, the execution of task 2 will be interrupted.

10 to 20ms: Complete the programs of Task 1. If there is any remaining time, perform Task 2 that was completed in the last period.

20 to 30ms: Task 2 is executed every 30ms. Task 2 has been executed within 10 to 20ms. In this case, you do not need to execute task 2. Perform task 1 with the highest priority only once.

30 to 40ms: same as before. 40 to 50ms: Task 3 appears. Task 3 has a lower priority. Therefore, Task 3 can be executed only after Task 2 is completely executed.




3. Execution type of the task

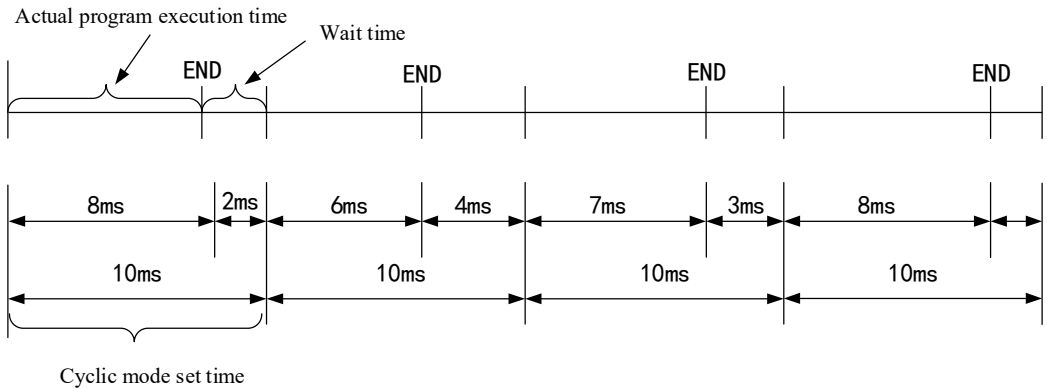
The type of editing and configuration that can be performed for each individual task. Including cyclic, event, external, freewheeling and status 5 types.

(1) Cyclic

According to whether the instruction used in the program is executed or not, the processing time of the program will be different, so the actual execution time will vary in each scan cycle, and the execution time will be long or short. By using the cyclic mode, a certain cycle time can be maintained to repeatedly execute the program. Even if the execution time of the program changes, a certain refresh interval can be maintained. Here, it is also recommended that you preferentially choose the cyclic mode.

For example, if the task corresponding to the program is set as cyclic mode and the interval is set to 10ms, the

timing diagram of the actual program execution is as shown in the figure below.

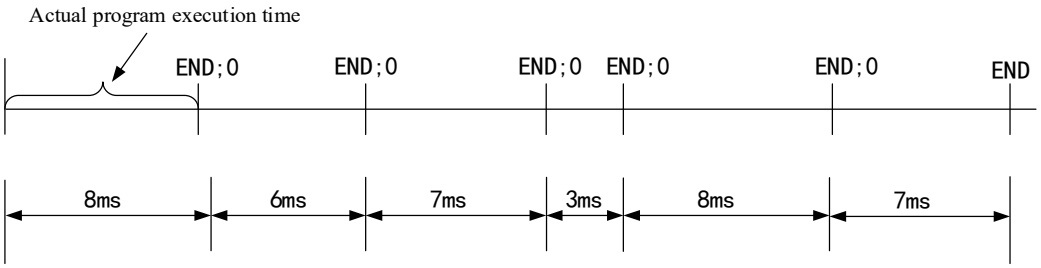



If the actual execution time of the program is completed within the specified cyclic setting time, the free time is used for waiting. If a task with a lower priority is not executed in the application, the remaining waiting time is used to execute the task with a lower priority.

(2) freewheeling

The task is processed as soon as the program starts running, and the task is automatically restarted in the next loop after the end of one run cycle.

It is not affected by the program scan cycle (interval time). That is to ensure that each time after the execution of the last instruction of the program before entering the next cycle. Otherwise, the program cycle will not end.




Because there is no fixed task time, the time of each execution may be different. Therefore, the real-time performance of the program cannot be guaranteed, and there are few occasions when this method is selected in practical applications.

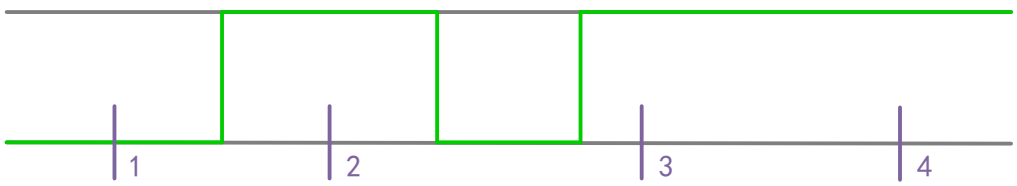
(3) Event

If the variable in the event area gets a rising edge, the task begins.

(4) Status

If the event area variable is TRUE, the task begins.

In the following figure, the event trigger and status trigger are respectively compared. The solid green line is the Boolean variable state selected by the two trigger modes. The following table is the comparison result.




Task input trigger signal

The state trigger method is similar to the event trigger function, the difference is that the program executes as long as the state trigger variable is TRUE, and does not execute if it is FALSE. The event trigger only collects the effective signal of the rising edge of the trigger variable.

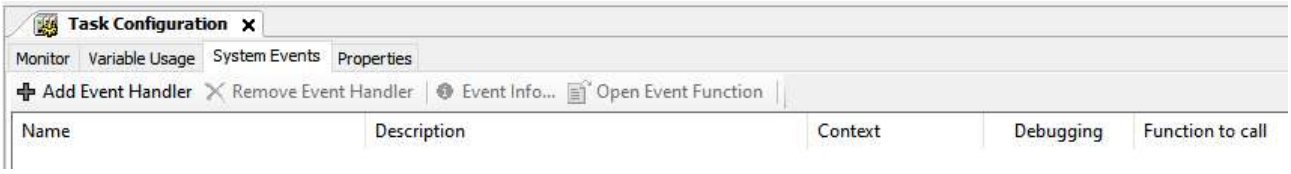
At sampling points 1-4 (purple) different types of tasks show different responses. This specific event of TRUE fulfills the condition of the state-driven task, whereas an event-driven task requires the event to change from

FALSE to TRUE. If the sampling frequency of the task plan is too low, the rising edge of the event may not be detected.

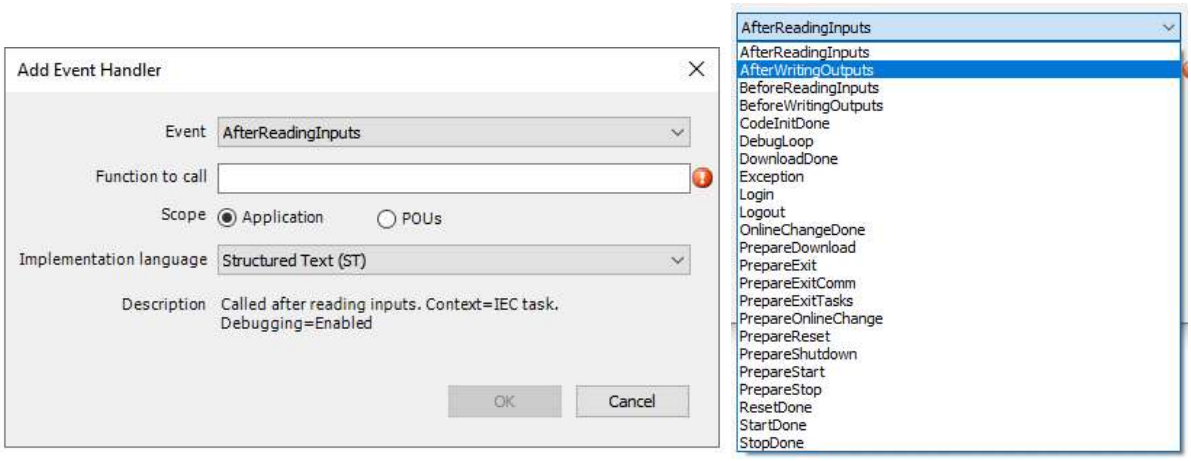
| Execution point | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Event | Not execute | Execute | Execute | Not execute |
| Status | Not execute | Execute | Execute | Execute |

(5) System events

The system events that users can select are based on the actual hardware target system, and the corresponding library files of the target system provide corresponding system events. Therefore, the system events corresponding to different target hardware devices may be different. But generally speaking, common system events are: stop, start, login, change, etc. In task configuration, you can set system events in task configuration.




You can choose Task Configuration > System Events to go to the Add Event handler page. Click Add Event handler to add system events. Users can select the time by dropping down, as shown in the following figure:




(6) External

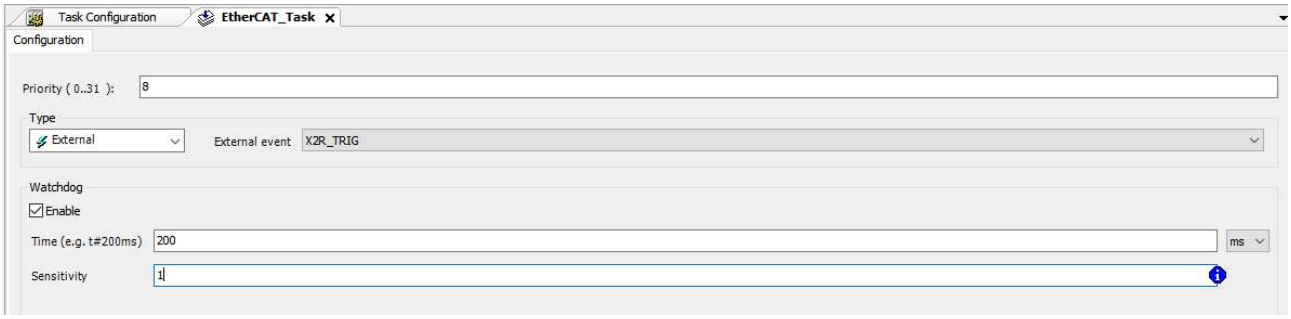
If the variable in the event area gets an external interrupt signal X with a rising or falling edge, the task begins.

The input terminal X can be used as an input to an external interrupt, each of which corresponds to an external interrupt, or a rising or falling edge or rising or falling edge can be specified as a trigger condition.

(7) Watchdog

The watchdog is a controller hardware-based timing device that can be enabled by Task Configuration in XS Studio. By default, the watchdog function is not used.

The main function of the watchdog is to monitor the exceptions that occur during the execution of the program or the failure of the internal clock. For example, when the system crashes or when the program enters a dead loop, the watchdog timer will send a reset signal to the system or stop the PLC currently running program. We can visualize it as a dog that needs its owner to feed it regularly, and if it is not fed beyond the prescribed time, it will be hungry immediately. To configure a watchdog, time and sensitivity must be defined. The configuration of a watchdog is shown below.




① Time

XS Studio can be configured with a separate watchdog for each task. If the target hardware supports long watchdog time settings, you can set the upper and lower limits. The default watchdog time unit is milliseconds (ms). If the program execution period exceeds the watchdog trigger time, the watchdog function is activated and the current task is aborted.

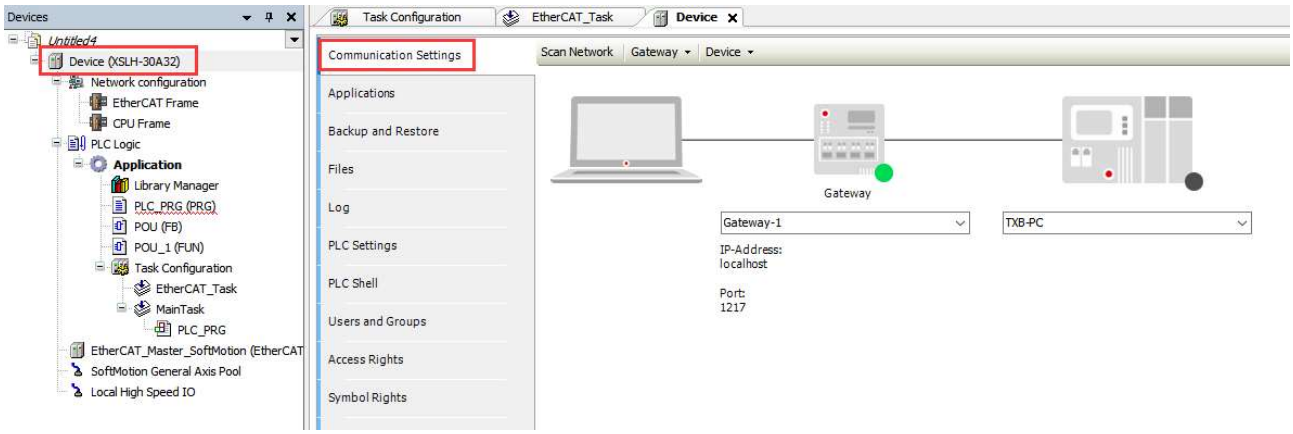
② Sensitivity

Sensitivity is used to define the number of task watchdog exceptions that must occur before the controller detects an application error. The default is 1.

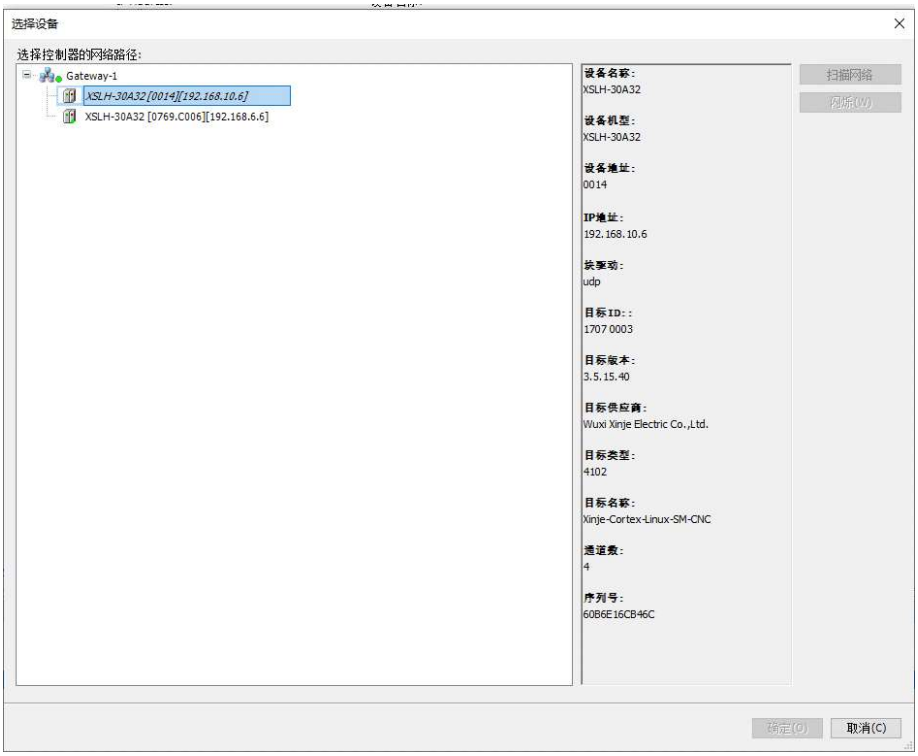
Final watchdog trigger time = time x sensitivity. If the actual execution time of the program exceeds the watchdog trigger time, the watchdog is activated. For example, if the time is 10ms and the sensitivity is set to 5, the watchdog trigger time is 50ms. Once the execution time of the task exceeds 50ms, the watchdog is activated immediately and the task is terminated.

## 2-3-3. Scan the device

Double-click the Device node in the left device tree to open the "Communication Settings" interface:

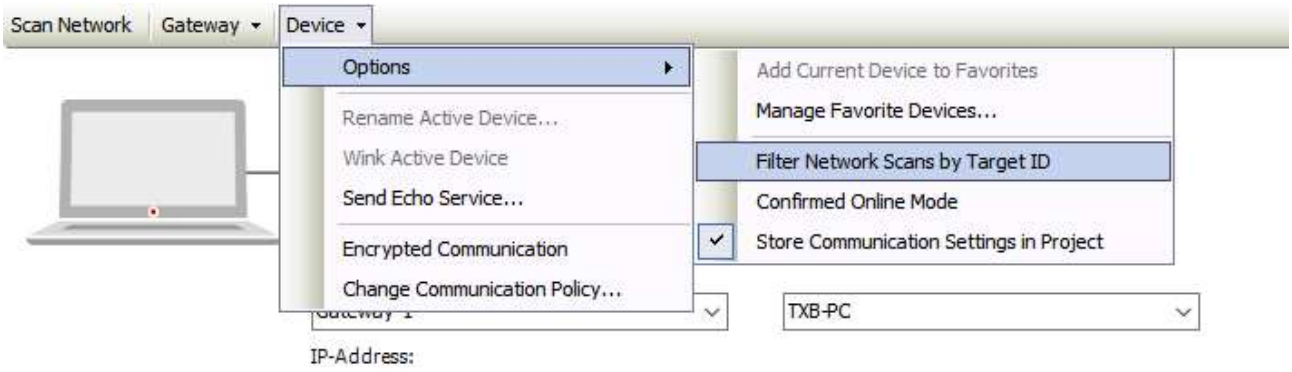


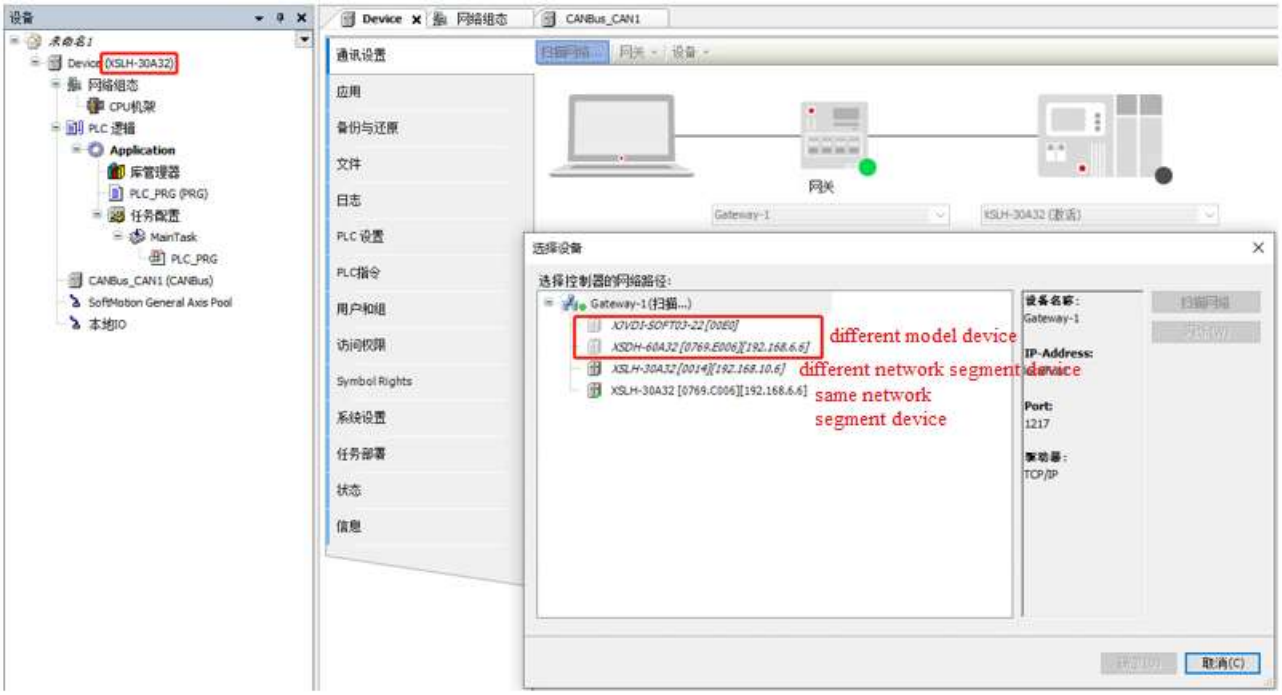

By default, the device connection configuration parameters are not modified. Click the "Scan network" option to open the "Select Devices" interface, you can start the scanning function, and the scanned devices will be displayed in the interface, as shown in the figure:



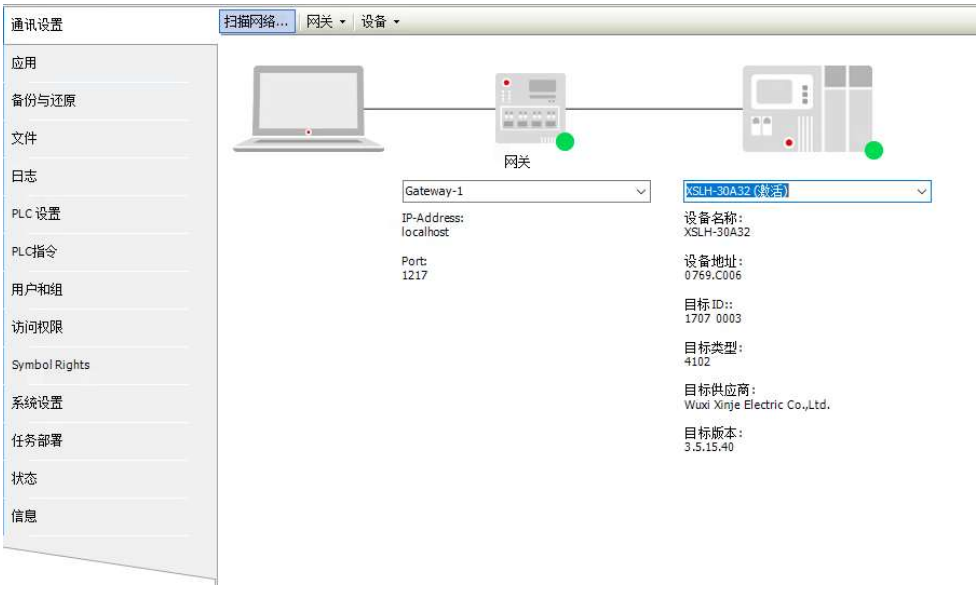

- The device 192.168.6.6 in the same network segment is displayed in green. You can select the current device and double-click the device to connect to it, or select the device and click OK to connect to it.
- Also displays cross-network segment device 192.168.10.6 without a green identifier in italics. After you select the device, you can view the device information on the right, but the connection cannot be set up.

Before scanning devices, you can open the Options menu under the Device menu on the current screen and deselect Filter network scans by target ID. Cancel and scan again. You can scan devices of the same engineering model or devices of different engineering models. As shown in the picture below:






Select the device in the same network segment that is displayed in green, for example, XSLH-30A32 (192.168.6.6). Select the device and double-click it. As shown in the picture below:

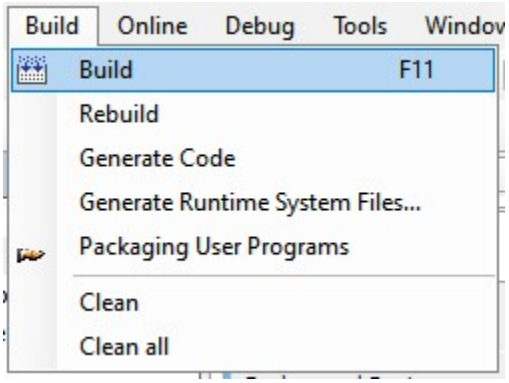



## 2-3-4. Program download/read

### 2-3-4-1. Compile

After the program is written, it needs to be compiled before it can be downloaded. The compile command performs syntax checks on the programs you write and only compiles programs that are added to the task. If the created POU is not added to the task, the compilation command does not perform syntax checks on the POU.

The compile instruction does not generate any code, but only checks the syntax of the POU. If the device login command is executed directly, the system will also execute the compilation command by default (equivalent to manually executing the compilation command first), and execute the connection login command after the compilation check is free of syntax errors. Also, no syntax check is done at compile for POU that are not added to the task. Executing the login command generates code.




(1) Build: Compile the current application.

(2) Rebuild: If you need to compile an already compiled application again, you can do so by recompiling.

(3) Generate code: After executing this command, the machine code of the current application is generated. When executing the login command, the generated code is executed by default.

(4) Clean: Delete the compilation information of the current application. If you log in to the device again, you need to generate the compilation information again.

(5) Clean all: Delete all compilation information in the project.

After the compilation command is executed, the PLC_PRG that is added to the task is displayed in blue, and the plc_prg that is not added to the task is displayed in gray. The compile instruction does not check the syntax of the gray POU because the program unit is not active, and the compile instruction only checks the syntax of the active POU. If a program unit that needs to be run appears gray during compilation, you can check whether the program unit has been successfully added to the task that needs to be run.

After the compilation command is executed, you can view the compiled information in the message bar, where you can see whether the compiled program has errors or warnings, and the number of errors and warnings. If errors and warnings are generated, you can view and search through the message window, and modify the program according to the prompt information.


### 2-3-4-2. Login download

1. Login

Login connects the application to the target device and makes it online. To log in correctly, the device's communication Settings must be configured correctly and the application must be error-free.

For logging in with the currently active application, the generated code must be error-free and the device communication Settings must be configured correctly. After login, the system will automatically select the program to download.

2. Download

Download command, valid in online mode. It consists of compiling the current application and generating object code. In addition to syntax checking (compilation processing), the application object code is generated and loaded into the PLC.

(1) Login-online change

When the user selects this option, the changed portion of the project is loaded into the controller. Log In - Online change to prevent the controller from entering the STOP state. You are advised to also select Update Automatic Startup program to prevent data program loss caused by the previous modification of program memory.

**Note:**

① The user has performed a full download at least once before.

② The pointer data is updated in the latest period. If the data type of the original variable is changed, the accuracy of the data cannot be ensured. In this case, you need to reallocate the pointer data.

(2) Login and download

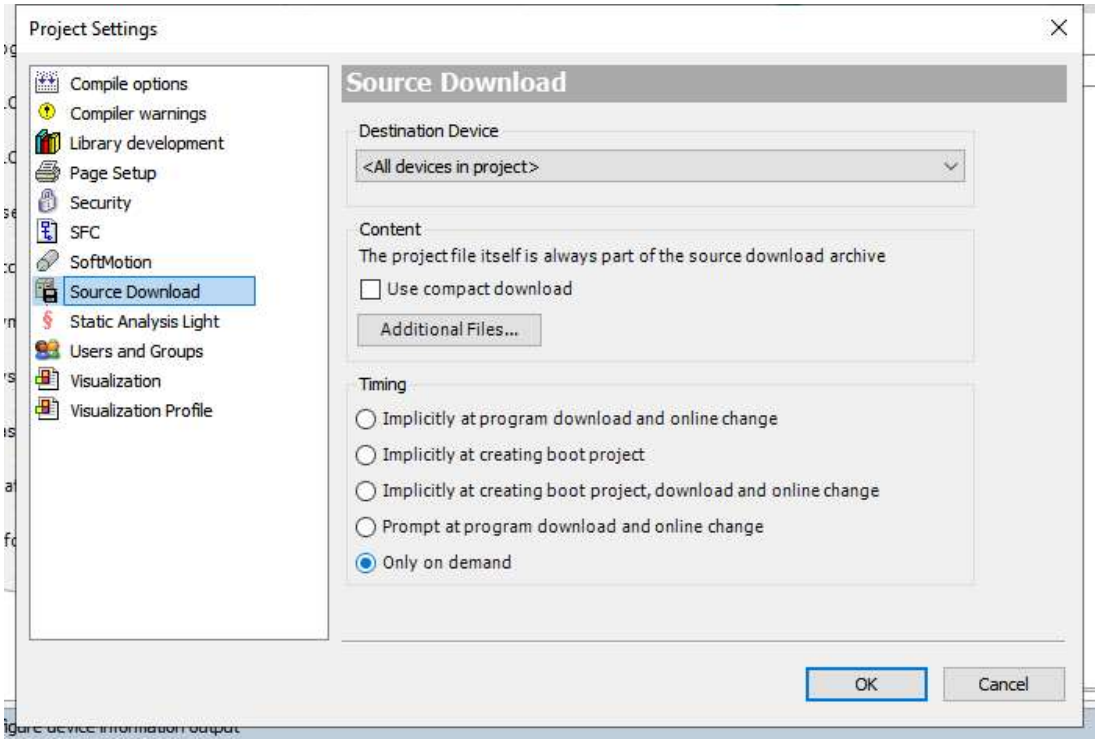
After you select "Login and Download," reload the entire project into the controller. The biggest difference with "login - online change" is that when the download is completed, the controller will stay in the STOP mode, waiting for the user to send the RUN command, or restart the controller program will run.

(3) Login-no any change

When you log in, the program that was last loaded into the controller is not changed.

### 2-3-4-3. Source code download

In order to protect the programmer's source code, the default download does not automatically download the source code, if you need to download the source code, you need to manually set, click "online" --> "source download to connected device". The user can also set this property in the "Project" --> "Project setting" --> "Source Download" --> "Timing" option.




### 2-3-4-4. Read program

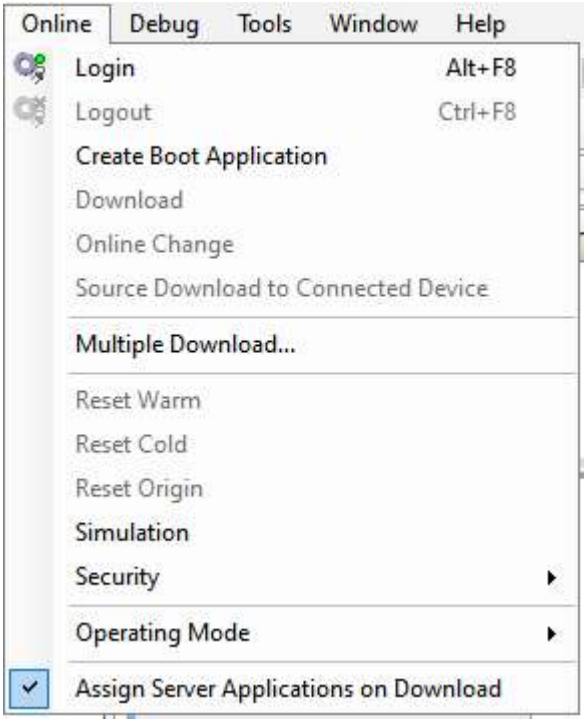
Click the "File" > "Source upload" to open a device selection dialog box, the user selects the network path to connect to the PLC, click the "OK" button. If the archive file already exists in the selected path, the system prompts you whether to overwrite the archive file.

It should be noted here that before reading the program, you need to make sure that you have done the "source download to connected device" during the previous download process. Otherwise, data in the controller cannot be read.

## 2-3-5. Program debug

### 2-3-5-1. Reset

You can reset an XS Studio program in the following three ways: Select one from the Online menu.




1. Reset warm

After hot reset, all current application variables are reinitialized except for PERSISTENT and RETAIN variables or those mapped to the M power down storage area. If variables with initial values are set, they are restored to their initial values after hot reset, otherwise variables are set to the standard initial value of 0.

2. Reset cold

Unlike "hot reset," the cold reset command not only sets the value of the common variable to the initial value of the currently active application, but also sets the value of the RETAIN variable to the initial value of 0. PERSISTENT variables, or variables mapped to M power down storage area remain unchanged.

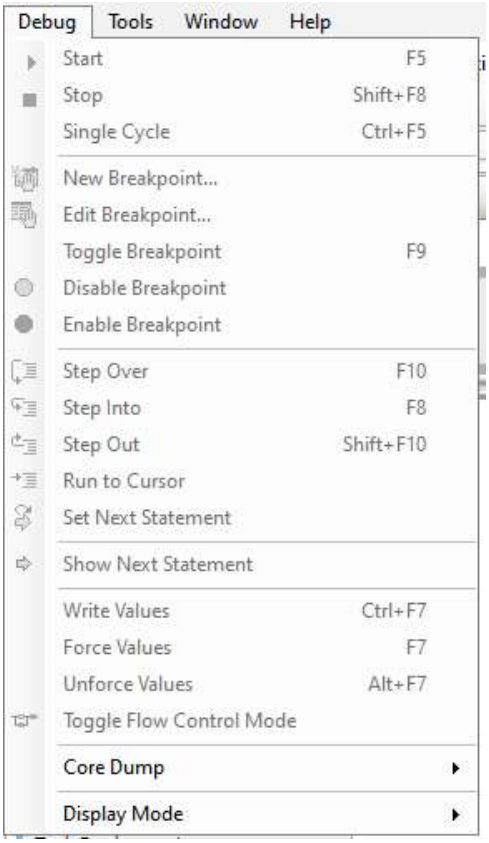
3. Reset origin

This command can be used when a programmable device is selected in the device tree, either offline or online. Using this command will reset the device to its initial state, i.e. any applications, boot projects, and remaining variables in the device will be cleared.

Because all the project information is cleared, after re-logging in, you need to re-download the program and "start" to run.

### 2-3-5-2. Program debug

The view of the Debug menu in XS Studio is shown in the figure. The main operations involve breakpoint setting and single cycle.




1. Breakpoint

Breakpoint is the function of processing stop in the program, when the program stops, the program developer can use this to observe the program to the breakpoint location of its variables and I/O and other related variables content, help to understand the mechanism of program operation, discover and eliminate program faults.

Breakpoints can be set in all programming languages in XS Studio. In the text editor ST language, breakpoints are set on the line; Set on the network number in the FBD and LD editor; In SFC, the setting is on the step.

2. Step

After the breakpoint is set, the program can be executed in a single step, which allows the program to run step by step, convenient for programmers to debug, in order to check the logic errors in the program.

(1) Step over

This command executes the current command in the program and stops after execution. Step over and step into commands have the same effect when POU is not called. However, if you call a POU, then step over does not enter the POU, but treats the POU call as a complete step, executed at once; Step into will enter the POU. If the SFC language is used, step over treats an action as a complete step and is performed at once. If you want to step into the called POU, you must use step into.

(2) Step into

When executed, the current instruction location is indicated by a yellow arrow. If the current instruction does not call POU, using this command has the same effect as using the step over command.

(3) Step out

When you are stepping in a POU, step out will execute the remaining instructions of the POU at once, and then return to the next instruction at the point where the POU was called. So, if you call POU layer by layer down, then the step out will return layer by layer up, one layer at a time. If the program does not contain any POU calls, then the step out cannot be returned to the upper level and will be returned to the beginning of the program.

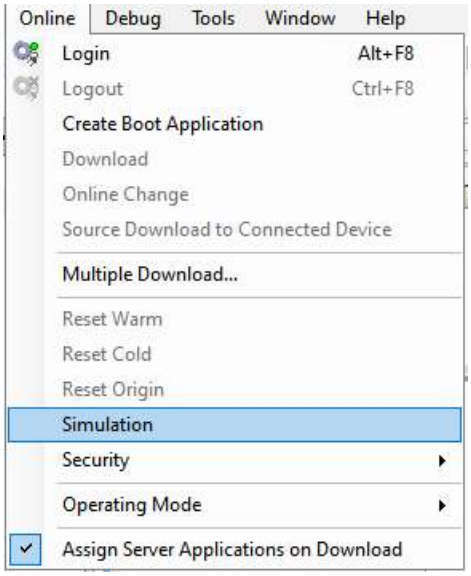
3. Single cycle

Select "Single cycle" in Debug, so that the program runs in a single step. That is, according to one run, the program executes a cycle to stop and wait for the next run instructions.

## 2-3-6. Simulation

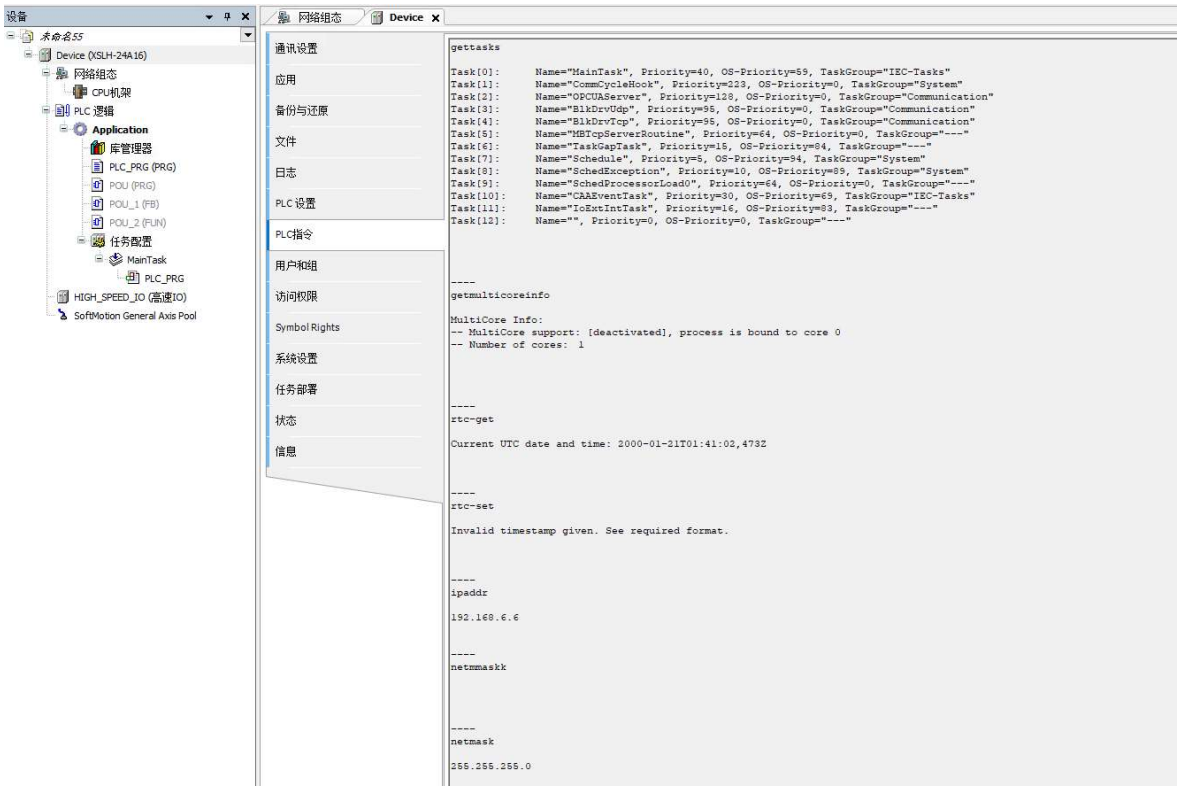
- Offline simulation

In the menu "online"→ "simulation", you will enter the simulation mode of the program running process. Verify that the "√" is marked before the "simulation" option, compile the program, and enter the simulation mode after there are no errors.



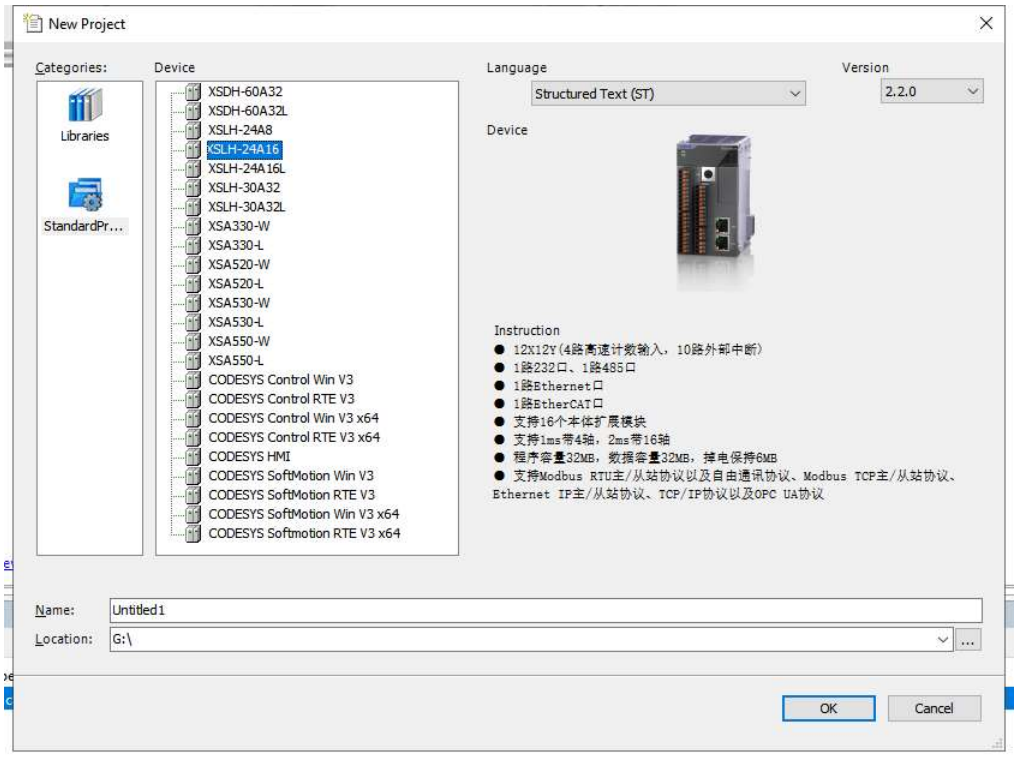

## 2-3-7. PLC script function

The PLC script is a text-based control monitor (terminal). This function takes a command with specific information from the controller, enters it as an input line and sends it to the controller as a string, returns the relevant string and displays the results in the browsing window. This function is used for diagnosis and debugging. Double-click the "Device", find "PLC shell" in the right view, and enter the corresponding command in the command input box below. Enter ? Press Enter to display all commands supported by the controller.

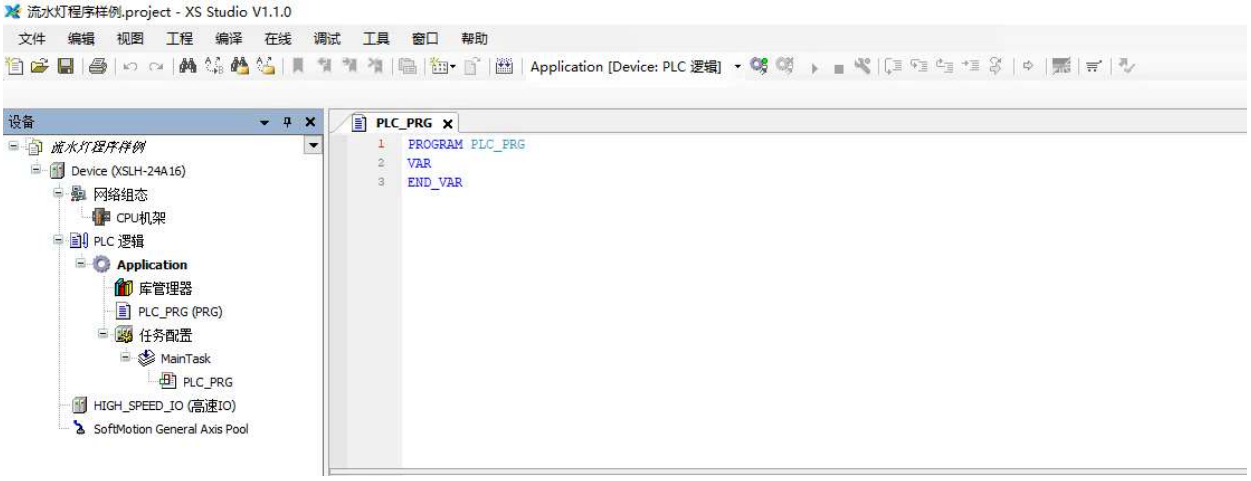


# 2-4. XS Studio write a sample flow lamp program

1. Build a new project

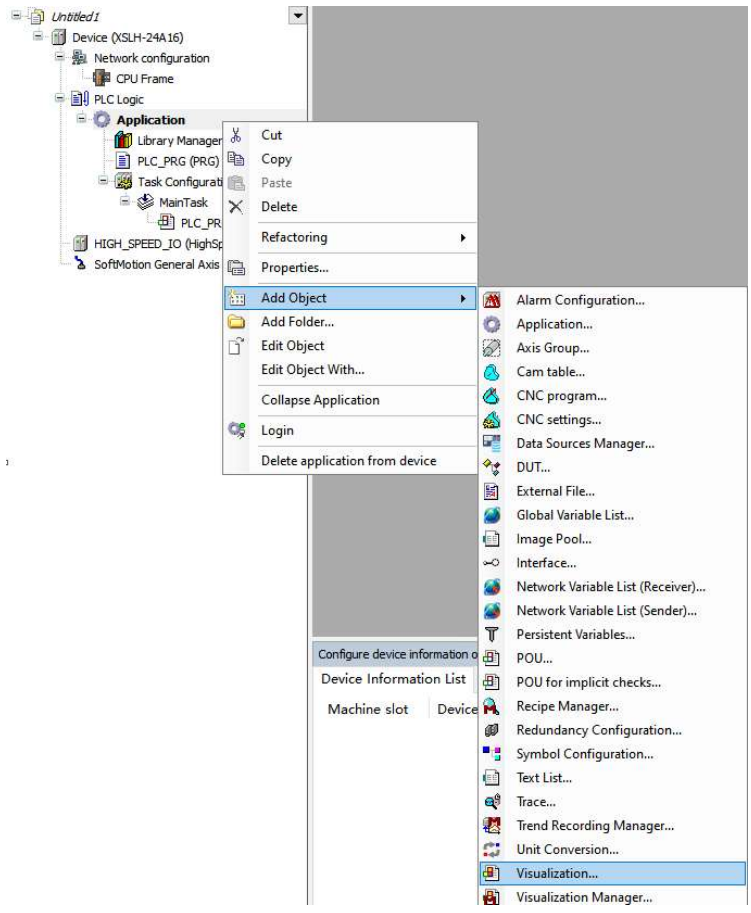


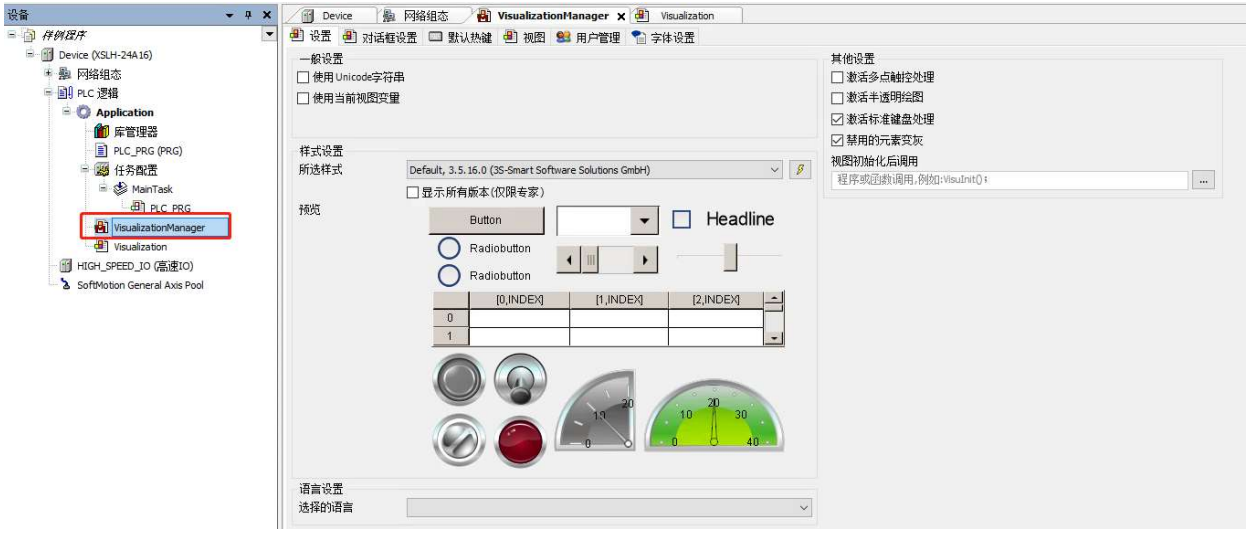

2. Make the program



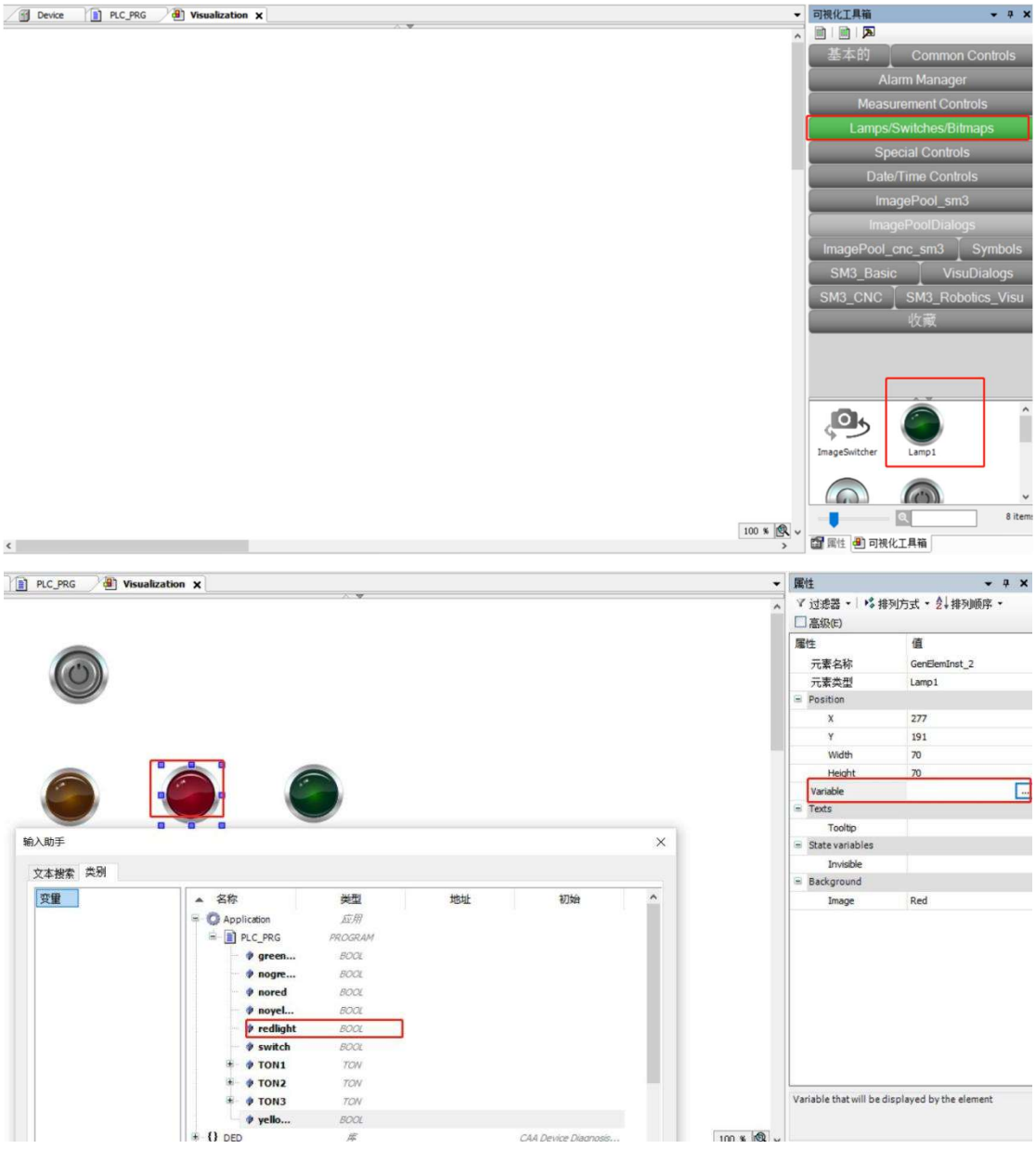
```
Device | PLC_PRG

PROGRAM PLC_PRG
VAR
    switch:BOOL;
    redlight:BOOL;
    greenlight:BOOL;
    yellowlight:BOOL;
    nored:BOOL;
    noyellow:BOOL;
    nogreen:BOOL;
    TON1:TON;
    TON2:TON;

IF switch THEN
    greenlight:=1;
    switch:=0;
END_IF
TON1(IN:=greenlight , PT:=T#1S , Q=> nogreen, ET=> );
IF nogreen THEN
    greenlight:=0;
    redlight:=1;
END_IF
TON2(IN:=redlight , PT:=T#1S , Q=>nored , ET=> );
IF nored THEN
    redlight:=0;
    yellowlight:=1;
END_IF
TON3(IN:=yellowlight , PT:=T#1S , Q=>noyellow , ET=> );
IF noyellow THEN
    switch:=1;
    yellowlight:=0;
END_IF
```

3. Click “application”-“add object”-“visualization”.

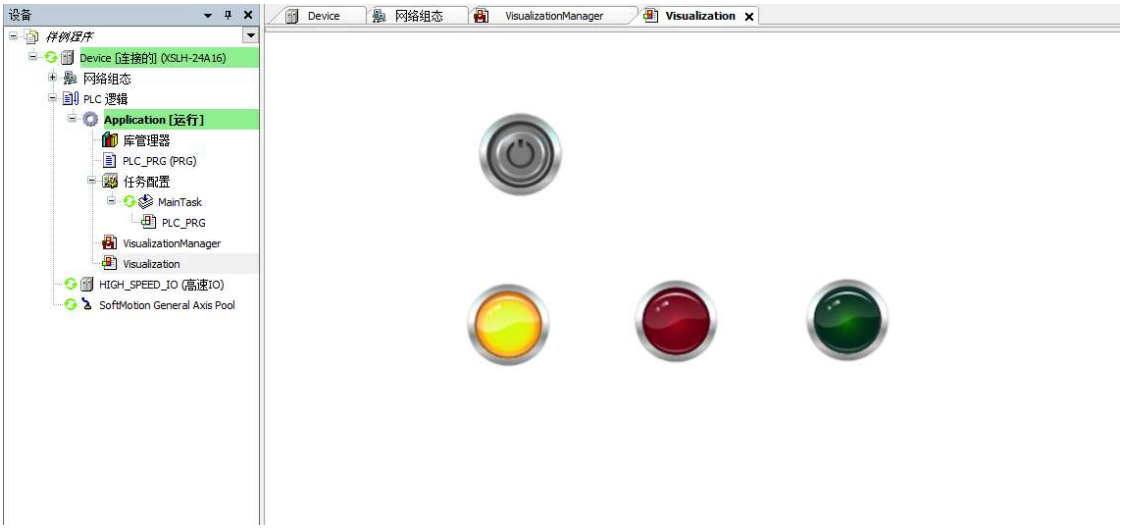





4. Add visual objects, map variables




5. Login the device, run.


设备
样例程序
Device [连接的] (XSLH-24A16)
网络组态
PLC 逻辑
Application [运行]
库管理器
PLC_PRG (PRG)
任务配置
MainTask
PLC_PRG
VisualizationManager
Visualization
HIGH_SPEED_IO (高速IO)
SoftMotion General Axis Pool
Device
网络组态
VisualizationManager
Visualization

## 2-5. How to login the device

### 2-5-1. Login operation steps and requirements

"Login device" means that XS Studio running on a PC establishes communication with XS series controllers to download user programs, monitor and debug them.

The PC can be directly connected to the XS series controller through network cables. The PLC can also be connected through a router or hub. In this case, a PC can be connected to multiple XS series controllers, and multiple PCS can also access the same XS series controller.

The IP addresses of the PC and XS series controllers must be on the same network segment and the gateways are working properly. Otherwise, XS Studio cannot scan XS series controllers. For example, the factory default IP address of XSDH is 192.168.6.6, and if the IP address of the PC is 192.168.6.xxx (xxx ranges from 1 to 254, but must not be the same as the IP address of XSDH), then XS Studio can scan XSDH and connect to it. Perform user program download, monitoring and debugging. If the IP address of the XSDH and the PC are not in the same network segment, the two cannot communicate. If the customer knows the IP address of the XSDH, the customer can change the IP address of the PC to the same network segment as the XSDH and then connect to the XSDH. If you do not know the IP address of the device, you need to restore the IP address of the XSDH to 192.168.6.6 and change the IP address of the PC to 192.168.6.xxx for connection.

### 2-5-2. Solution of cannot scan the device

- XS series

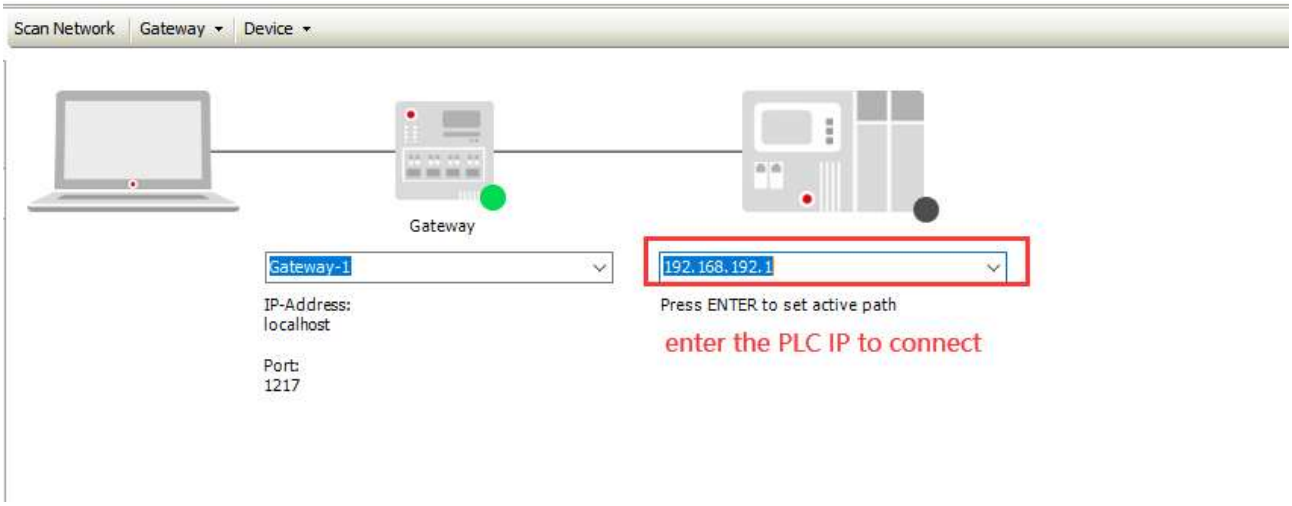
(1) Don't know the IP

Method 1: Power off the PLC, set DIP switch 1 to ON, and power it on again to restore the default IP address to 192.168.6.6.

Method 2: Starting with XS Studio V2.2.0 (PLC firmware version V2.2.0 or later), the device scanning function is supported different network segments, and the device IP address can be scanned across network segments.

(2) If the IP is confirmed correct but still cannot connect the device, it may be the PLC program crash (the program has a dead loop or exceeds the load capacity of the PLC), at this time you can set the dip 2 ON (power-on does not load the user program), and scan the connected device again; If the connection can be scanned, download an empty program, delete the abnormal program, restore the DIP switch status, and check whether the abnormal program has an excessively long cycle or the task period is too small.

(3) The IP network segment is modified, but if the PLC gateway is not set at the same time, it will not be connected. You can directly enter the IP address online as shown in the following figure, and then use the gateway command in the Device-PLC command to modify the PLC gateway.

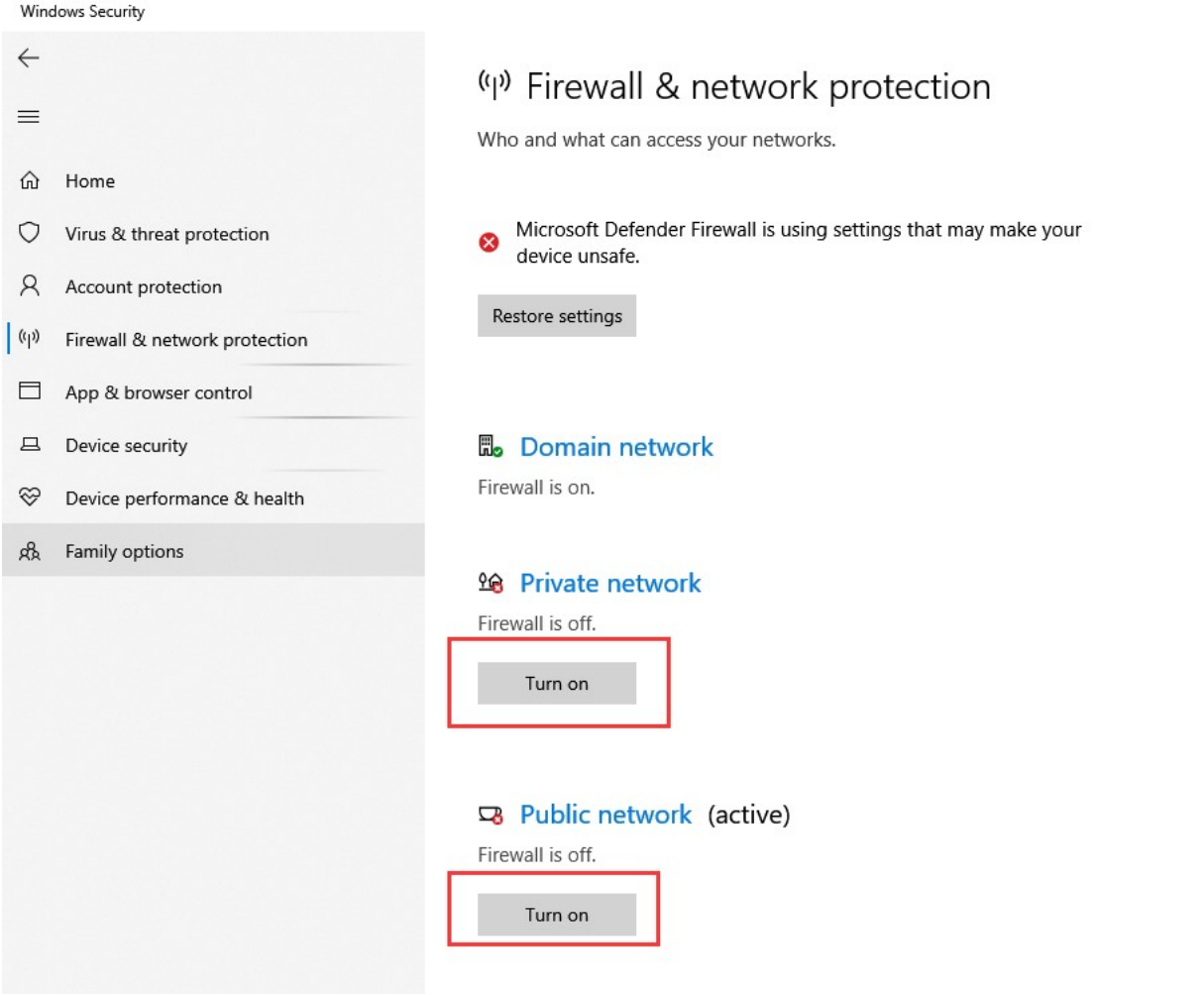



(4) If "Filter network scan through target ID" is selected, it is necessary to confirm that the engineering equipment

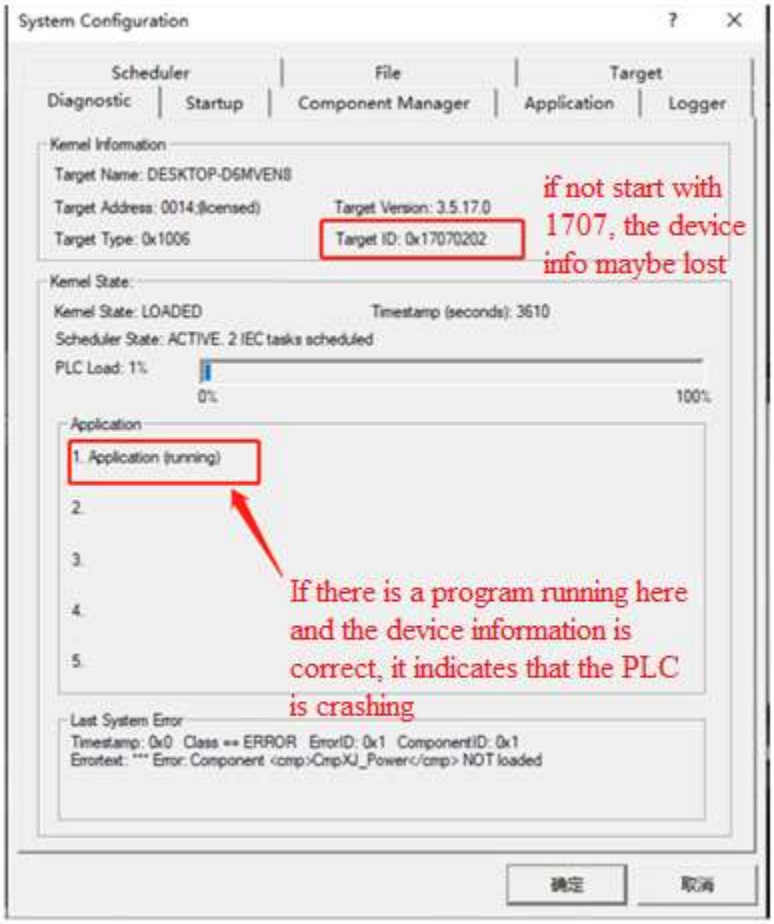
model of the upper computer is consistent with the target device, otherwise the device will not be scanned.
(5) If the above steps still fail to connect the device, please contact us and also provide information on the actions taken before the issue occurred, as well as the status of the RUN and ERR lights.

- XSA series

(1) Confirm that the upper computer engineering equipment is consistent with the target computer equipment.
(2) Connect the monitor and confirm that the IP, subnet mask, and gateway are correct. Confirm whether the IP addresses of both the PLC and the computer are in the same network segment and whether they can be pinged; If the IP address is correct but cannot be pinged, it may be a problem with the firewall, and the industrial computer firewall needs to be closed before connecting




If it can be pinged, enter the IP address directly to connect and eliminate subnet mask issues.
(3) If ping is possible and the device cannot be connected even after entering the IP address, there are two possible options:
① The PLC program crashes, delete the D:\ CODESYS \ Application folder (delete user programs), and then restart the industrial computer.
② The device information is lost, and the target ID can be viewed through the RTE configuration interface. The high 16 bits ID of this series of products is 1707. You can contact technical support for recovery processing.




Note: This step requires an external display for querying, otherwise it cannot be processed.

# 3. Network configuration

## 3-1. Device configuration

Configuration is the first step for users to program PLC, and the functions that the PLC can support can be added through the "Network Configuration" and "Hardware Configuration" interfaces.

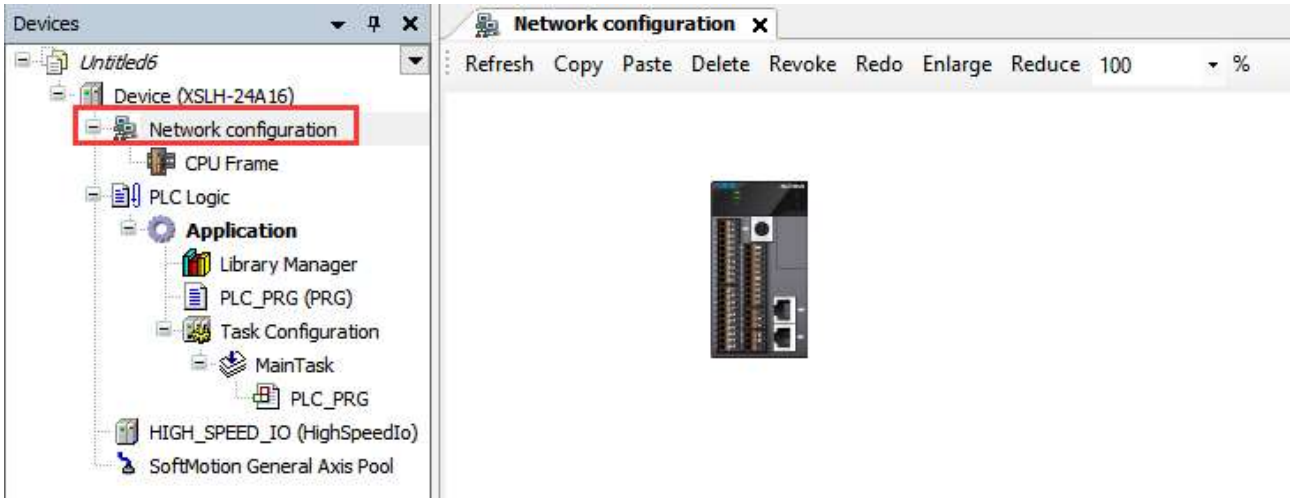
Network configuration: It is the entrance to the configuration device, which can layout the master and slave station devices through the enable window and the network device connection list, and display them in the interface of a bus type network topology.

Hardware configuration: Expansion IO modules can be added to medium-sized PLCs.

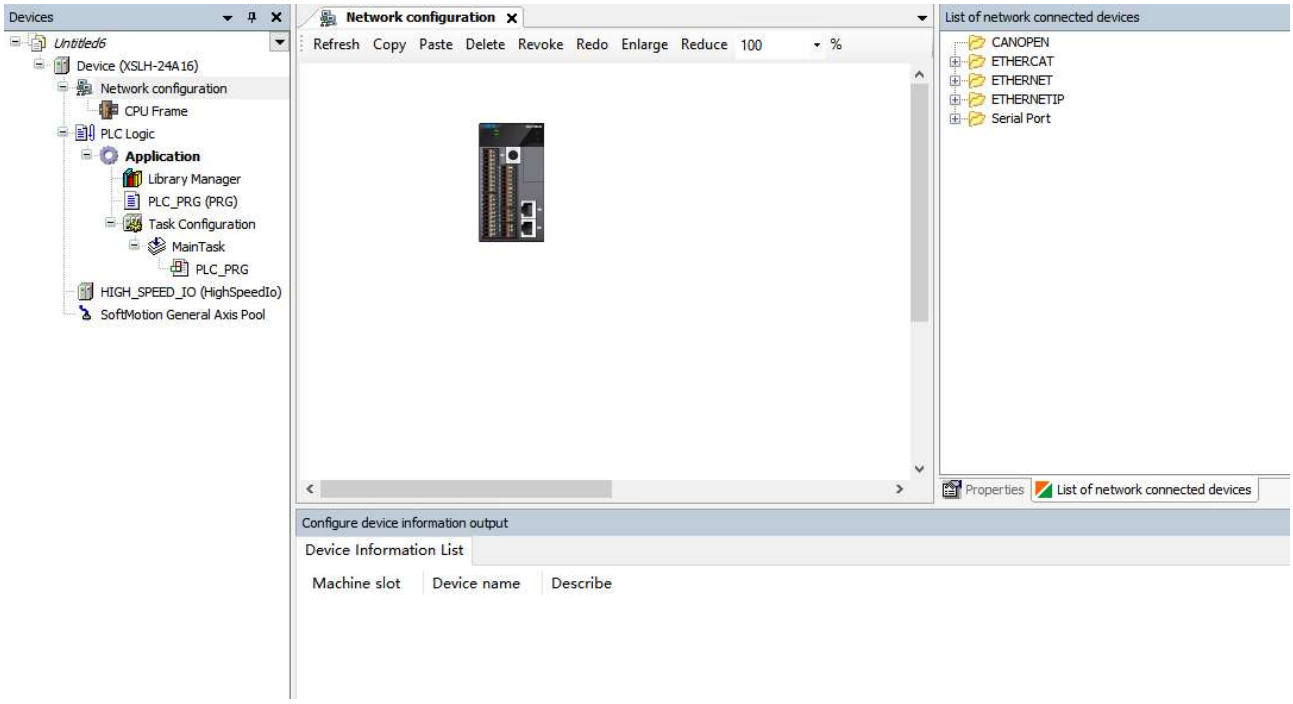
### 3-1-1. Network configuration

1. Open the configuration interface

After creating a new XS Studio project, you can open the configuration interface by double clicking on the "Network Configuration" node in the device tree on the left side of the software.




Double click on the "Network Configuration" node to open the network configuration interface, the list of network connected devices on the right, and the configuration device information output interface. The network configuration interface displays the PLC devices currently used by the user project, while the list of network connected devices displays all the devices supported by the current PLC. The device information output interface displays the name and related description information of the devices in the current configuration interface.




**Note:**

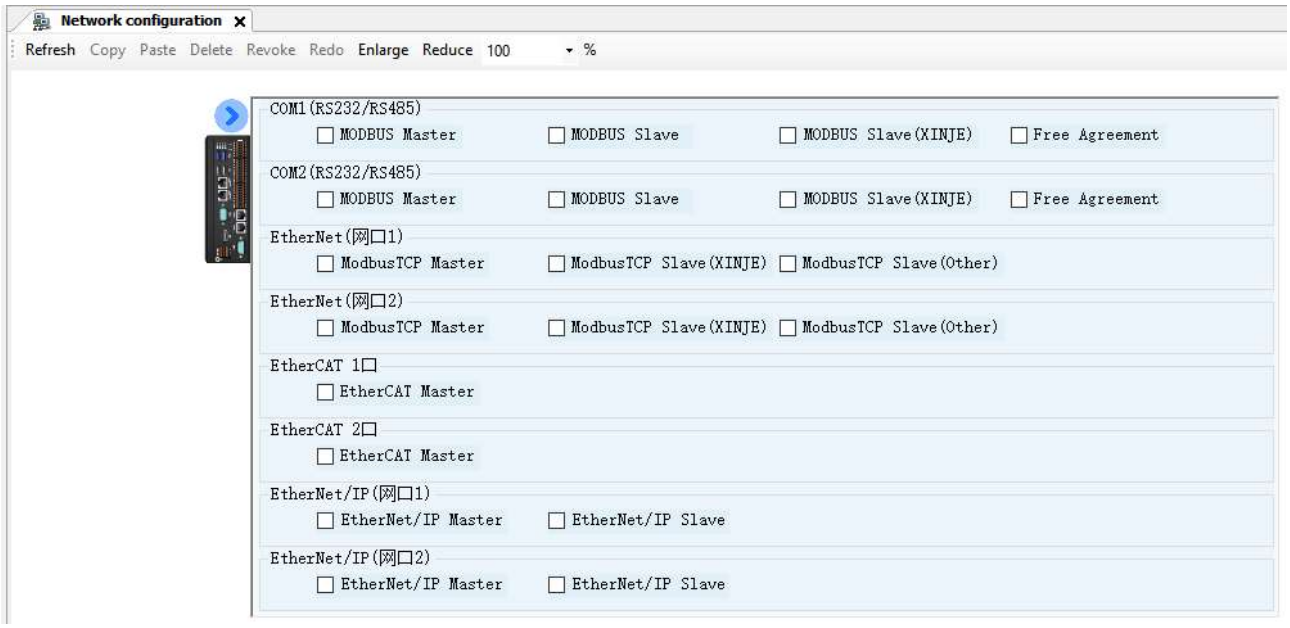
(1) The list of network connected devices defaults to a collapsed state;

(2) By default, the device information list is empty. When selecting a device in the network configuration interface, the relevant information description of the currently selected device will be dynamically displayed.

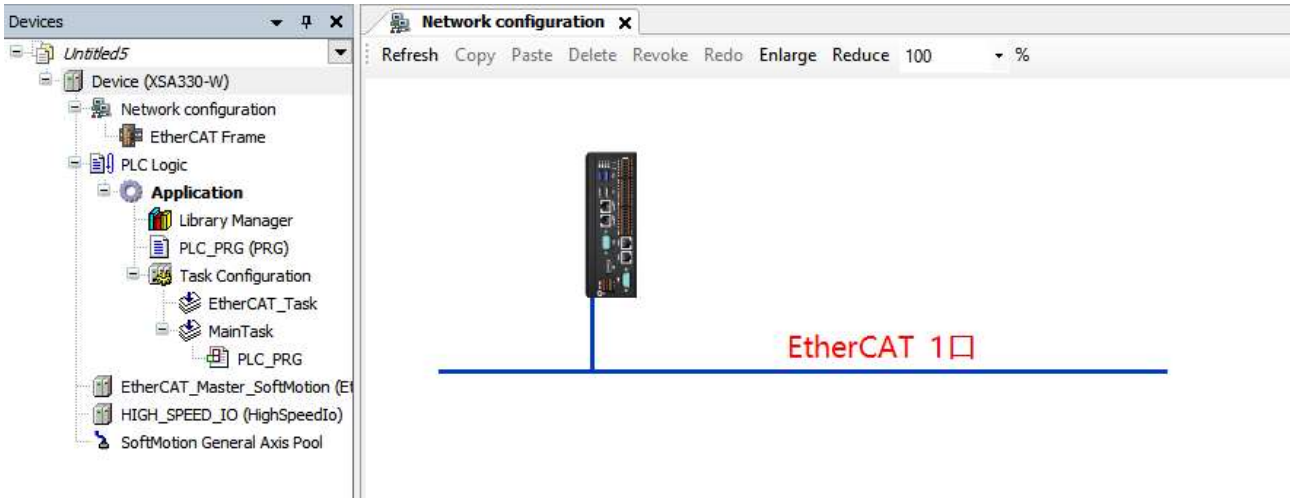
2. Set PLC as master or slave equipment

(1) Enable the master station

Clicking on the PLC device in the network configuration will display the master/slave enable window supported by the PLC. As shown in the following figure, selecting the checkbox button in the window according to the application needs can enable the master/slave functions supported by the CPU. Taking XSA330-W model as an example:




When the master station function of the CPU is enabled, a bus type topology interface will be displayed, and corresponding device nodes will be generated on the left side. The following figure shows the EtherCAT master station enabled.




- Disable device

Clicking again on the previously selected device will result in a pop-up asking if you are sure to remove the current device. Users can choose to confirm or cancel the current disabled operation.

- Mutual exclusion rule
  - COM port: When making protocol modifications to existing hardware interfaces, a pop-up prompt will appear. Click OK to replace with the newly added device, and click Cancel to cancel the current operation;
  - EtherNet: ModbusTCP Xinje slave and official slave are mutually exclusive, and selecting both will pop up a box to inquire;
  - EtherCAT: No mutual exclusion;
  - CANopen: No mutual exclusion;
  - EtherNet/IP: EtherNet/IP master/slave can be used simultaneously without mutual exclusion.

(2) Add slave station

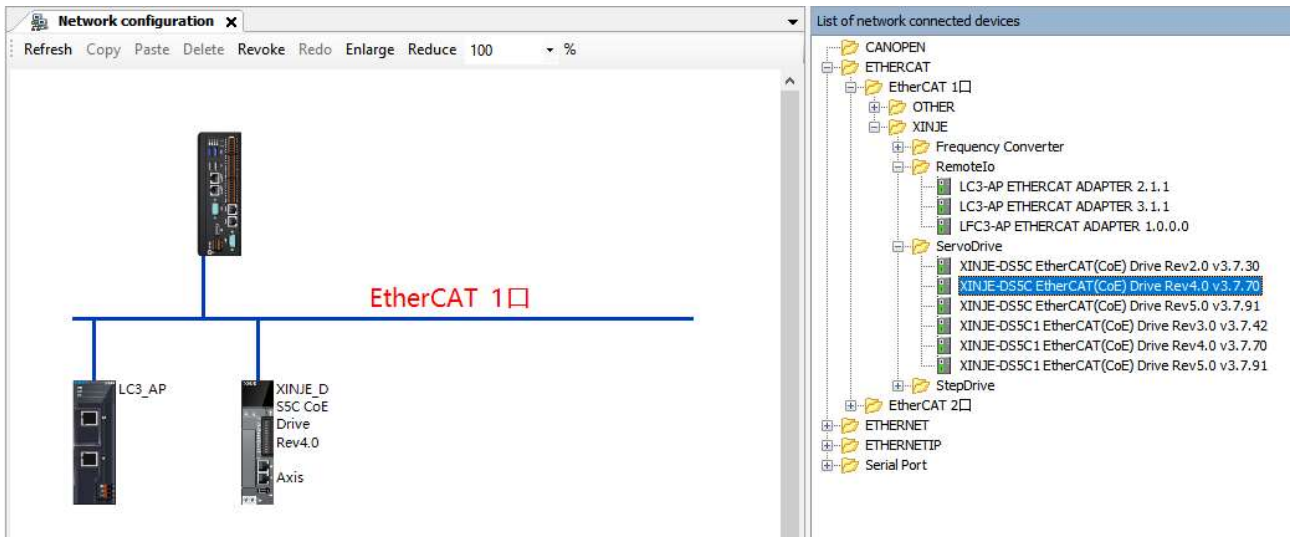
After enabling a specific master station in the CPU, you can add its corresponding slave devices under the bus. There are three ways to add slave devices (using EtherCAT bus as an example):

① First enable the EtherCAT master function, then select a slave device node from the EtherCAT port node in the network connection device list, hold down the left mouse button and drag it to the network configuration interface.

② First enable the EtherCAT master function, then double-click a slave device node under the EtherCAT port node in the network connection device list.

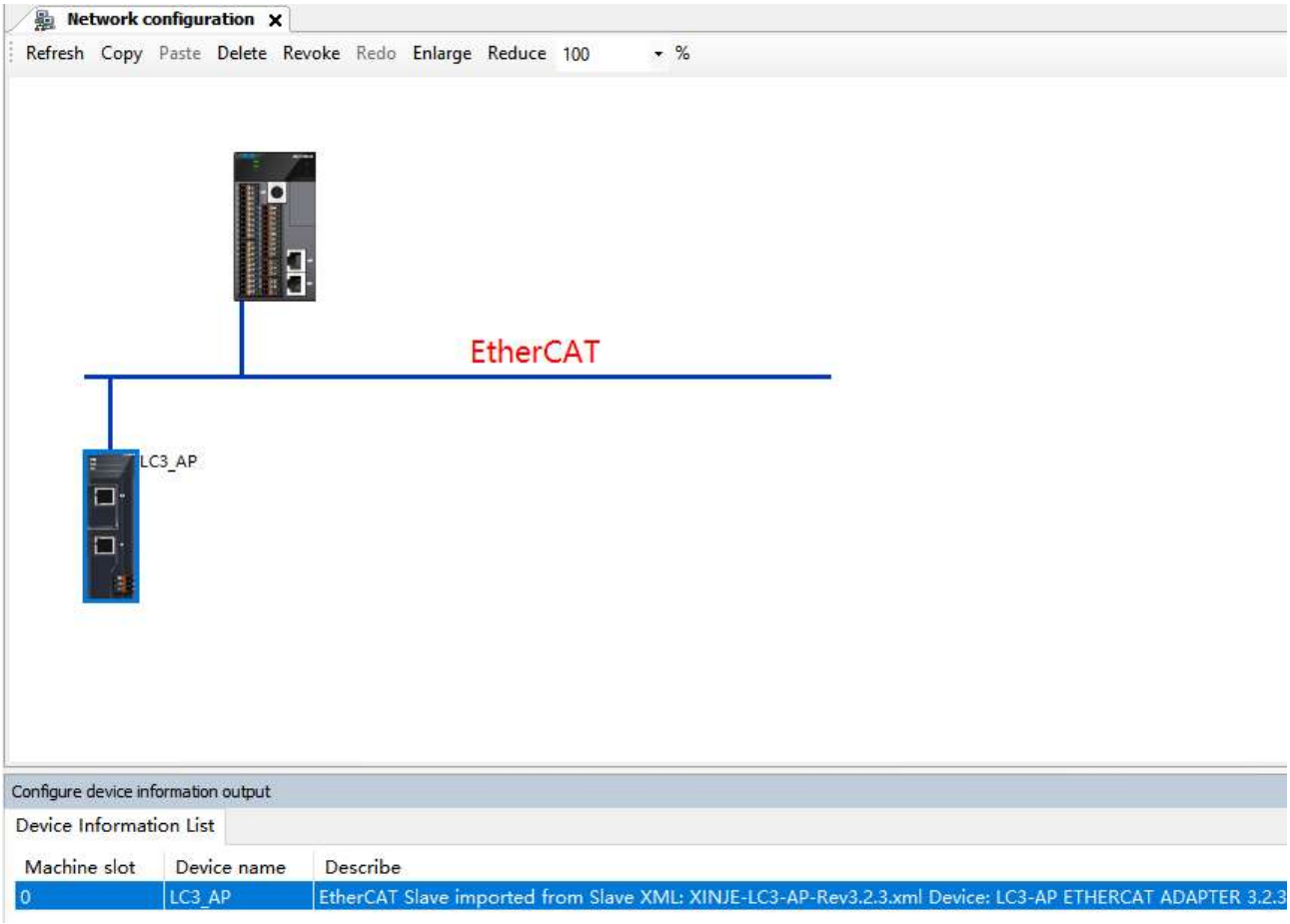
③ Double click a slave device node directly under the "EtherCAT Port" node in the network device list to add it. This method will default to enabling specific master station functions within the CPU.

If the added slave is an EtherCAT remote IO device, the IO module behind the slave needs to be configured. You can double-click the device to enter the "EtherCAT frame" interface for configuration. The network configuration interface after adding a slave station is shown below:



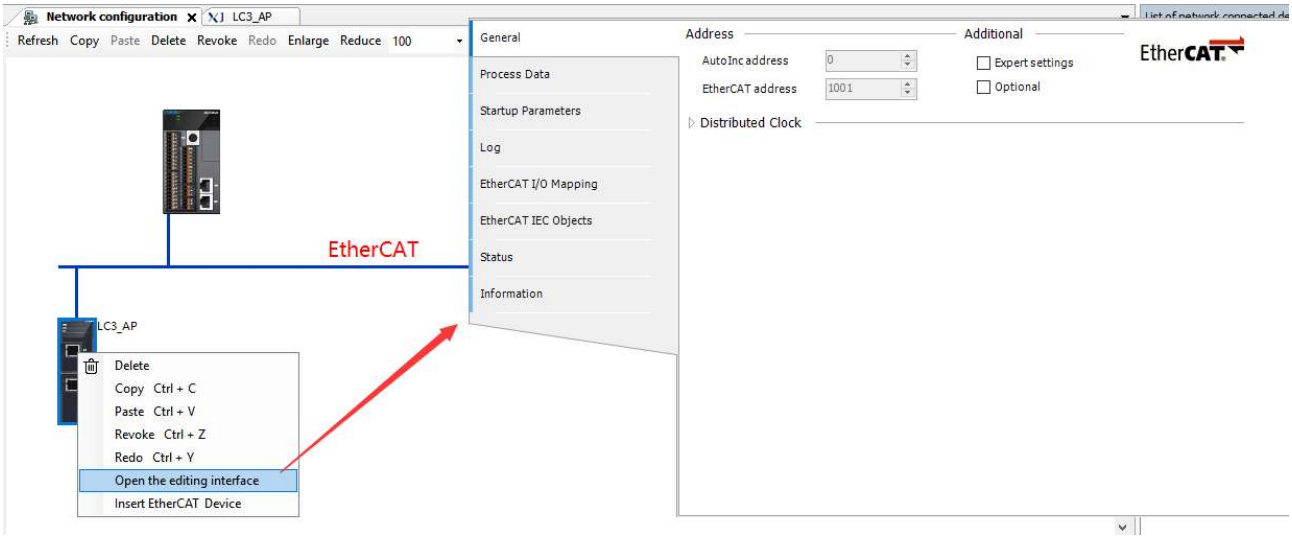
(3) View basic device information

After selecting the device in the network configuration interface, you can view the basic information corresponding to the device in the device information list in the "Configuration Device Information Output" box.




(4) Open the editing interface

Right click on the slave device in the network configuration interface, and enter the parameter configuration interface of the device through the "Open Editing Interface" menu item. Taking EtherCAT as an example, as shown in the figure:



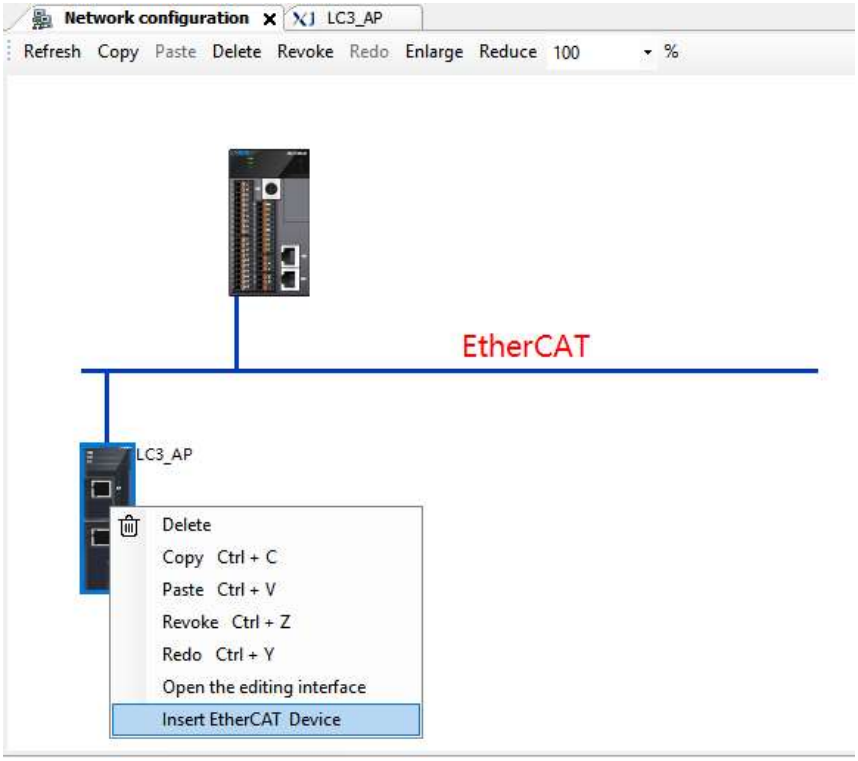

**Note:**

① Double clicking on the EtherCAT or CANopen device icon in the network configuration interface will redirect you to the hardware configuration interface corresponding to the device module. Clicking on other device icons will redirect you to the module parameter configuration interface;

② Double click on the expansion module or IO module behind the slave station to open the module configuration interface.

(5) Insert device

Right click on the slave device in the network configuration, and the "Insert XX Device" menu item can open the Insert Device pop-up to add a slave device. Taking the insertion of EtherCAT devices as an example, as shown in the following figure:



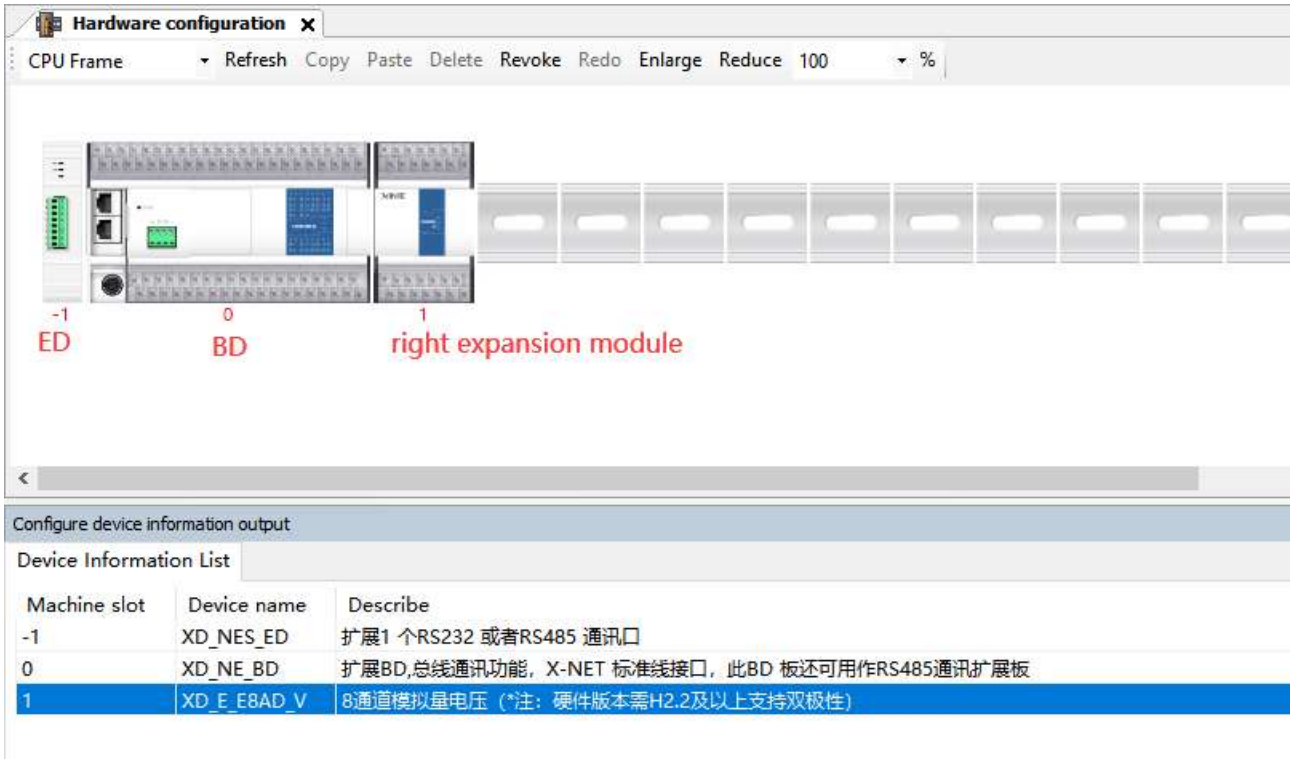

The configuration device can be operated by copying, pasting, deleting, etc. Please refer to the basic operation instructions for configuration for specific details.

(6) Device information list

The device information list is opened through the "Configuration Device Information view" under the "View" menu bar in the software. The device information list displays the basic information of the configured device, mainly including the slot number, device name, and corresponding information description. If the device information list is hidden, you need to manually click to open the list interface.


- Machine slot

The corresponding device slot in the hardware configuration, whether it is a module on the main frame CPU or a module behind the communication slave station, starts with slot number 1. Among them, the communication slave body slot number in the hardware configuration interface defaults to 0. The first slot 1 in the main frame CPU corresponds to the left expansion module, the second slot 1 corresponds to the middle expansion module, and the third slot 1 corresponds to the first right expansion module. As shown in the following figure:




Configure device information output

Device Information List

| Machine slot | Device name | Describe |
|---|---|---|
| -1 | XD_NES_ED | 扩展1 个RS232 或者RS485 通讯口 |
| 0 | XD_NE_BD | 扩展BD,总线通讯功能，X-NET 标准线接口，此BD 板还可用作RS485通讯扩展板 |
| 1 | XD_E_E8AD_V | 8通道模拟量电压（*注：硬件版本需H2.2及以上支持双极性） |

- Device name

Consistent with the device name shown in the device tree on the left side of the software.

- Describe

The basic description of the equipment, including its basic working indicators and functions.

■ Basic operation of configuration

The basic operations of configuring equipment include the functions of refreshing, copying, pasting, deleting, revoking, restoring, zoom in, and zoom out the equipment:


- Refresh

Click the refresh button. If there is an empty slot between two right expansion modules in the hardware configuration interface or CPU rack interface, the module on the right side of the empty slot can be moved left to replace the empty slot after refreshing.

Add devices with description files such as XML, EDS, DCF, etc., and update them in the list of network connected devices after refreshing.

- Copy

After clicking on the newly added device, click on the corresponding device icon with the mouse, and the device will be highlighted. Click "Copy" to copy the corresponding device. The copy button is not available when the corresponding device icon is not clicked. Support shortcut keys Ctrl+C for copying operations.

- Paste

Click copy on the added device to paste it. The paste button is not available without copying. Support shortcut keys Ctrl+V for pasting operations.

- Delete

After selecting the device, you can click "Delete" to delete the corresponding device. In the unselected state, the delete button is not available. Support the shortcut key Del for deletion operations.

- Revoke

The previous step in the interface configuration can be undone, and it can be undone multiple times in a row. Support shortcut keys Ctrl+Z for undo operations.

- Redo

For revoked content, clicking "Redo" will restore you to the interface before the previous revocation. When there is no undo operation, the redo button is not available. Support shortcut keys Ctrl+Y for recovery operations.

- Zoom in/out

The interface scaling ratio can be set by zooming in/out the dropdown menu, and the shortcut key Ctrl+mouse can also be used to zoom in or out of the current interface.

As shown in the following figure:

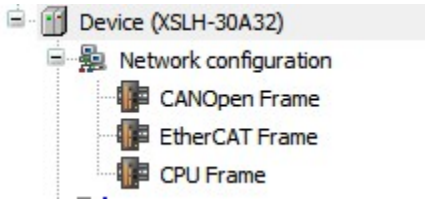

**Note:**
① The operations of copying, pasting, deleting, revoking, and restoring devices are only applicable to IO modules in the hardware configuration EtherCAT rack and CPU rack interfaces, disabled in the CANOpen rack interface, and only applicable to slave devices in the network configuration interface;
② If copying, pasting, or deleting slave devices in the network configuration interface, subsequent modules will also be operated accordingly;
③ Import device files: Support importing the required device files through the "Tools" and "Device repository" menu items in the software menu bar, and can import device description files of types such as XML, EDS, DCF, etc.

## 3-1-2. Hardware configuration

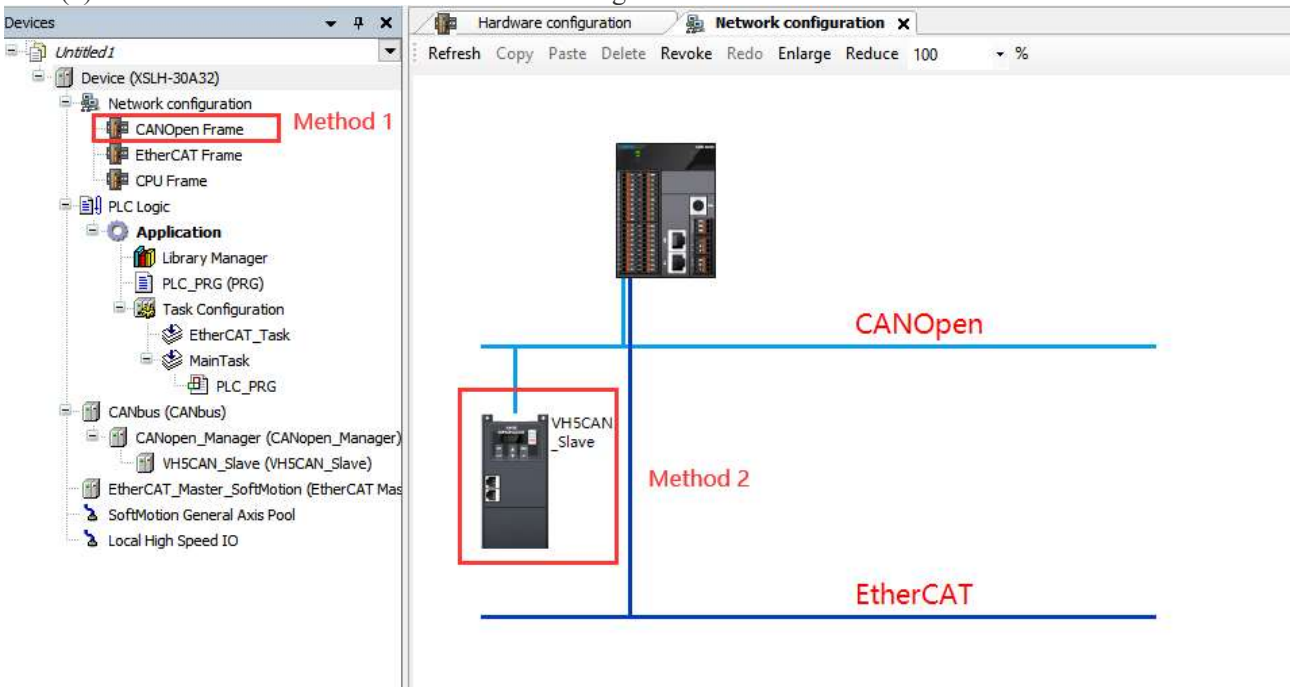
Hardware configuration introduces the concepts of racks and slots in actual device configuration to simulate modular configuration of on-site devices. The hardware configuration is mainly aimed at the IO modules of PLC series products.
In terms of configuration process, if adding a remote IO module, the communication module configuration should be completed in the network configuration before configuring the IO module in the hardware configuration.

1. Hardware configuration interface

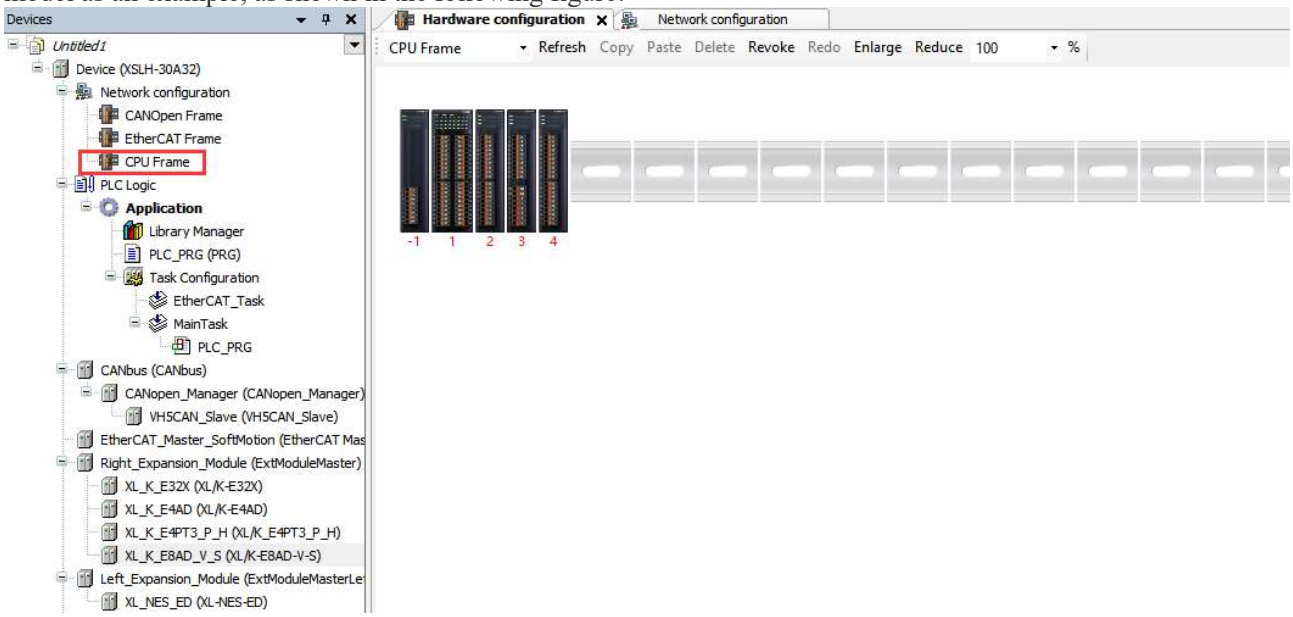
At present, bus type devices EtherCAT and CANopen have corresponding hardware configuration interfaces.




There are two ways to access the hardware configuration interface:
(1) Double click on a bus node under the network configuration node in the device tree;
(2) Double click on a device in the network configuration interface.



ARM series controllers have a default "CPU frame" bus configuration node (not supported by X86 series controllers). Double click on the "CPU frame" in the left device tree to enter the local module configuration interface, and the "I/O module list" will be displayed on the right side of the software. Taking the XSLH-30A32 model as an example, as shown in the following figure:




**Note:**

① The XSDH series supports adding 1 ED and 1 BD, as well as 16 right expansion modules;

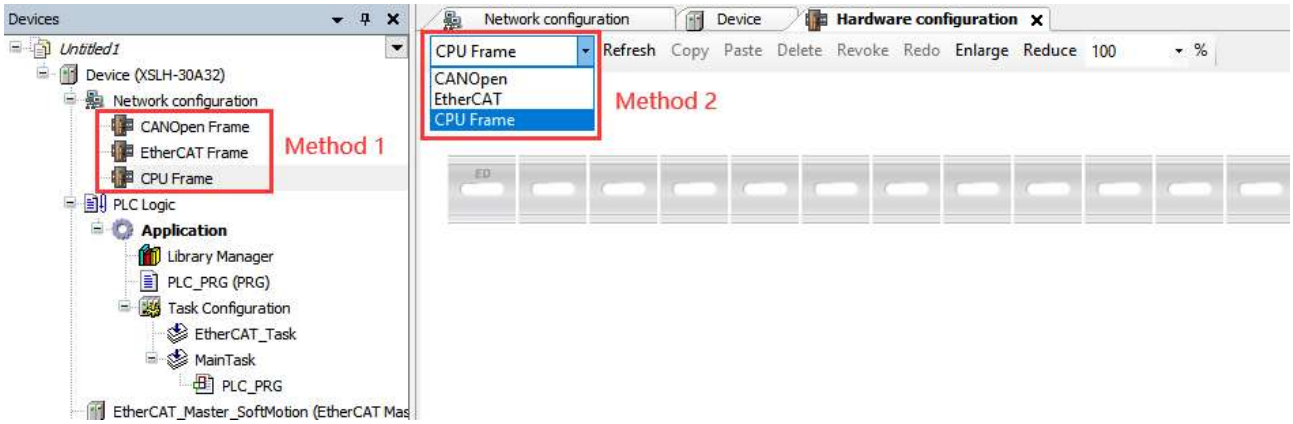
② The XSLH series supports adding 1 ED and 16 right expansion modules.

2. Bus switching

There are two ways to switch between hardware configuration buses.

(1) Double click on a bus node under the "Network Configuration" node in the device tree on the left side of the software to enter the corresponding configuration interface;

(2) Select other bus types in the dropdown menu on the current hardware configuration interface to switch.




3. Add module

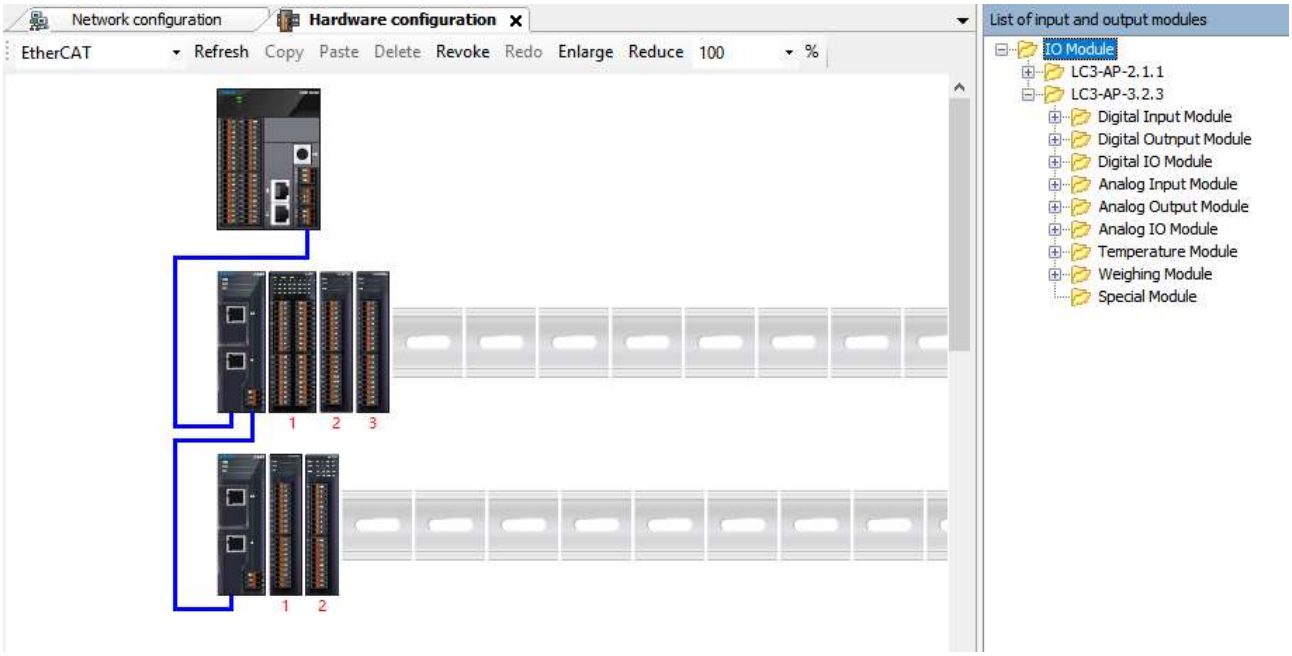
There are three ways to add IO modules.

(1) Double click the bus slave station to open an empty slot on the rack, and double-click a specific module in the pop-up "Insert Remote IO Module" to add it;

(2) In the "Input/Output Module List" in the right view, select a device node, hold down the left mouse button, and drag it onto an empty slot;

(3) To add an IO module from the back of the station, you need to select a device and double-click a device in the "I/O Module List" on the right side of the view to automatically add the devices to the empty slots on the rack in order. If a certain empty slot is selected, it will be added to that empty slot. To add an IO module to the CPU rack interface, there is no need to select the main body before adding it. Simply double-click on it.

As shown in the following figure:



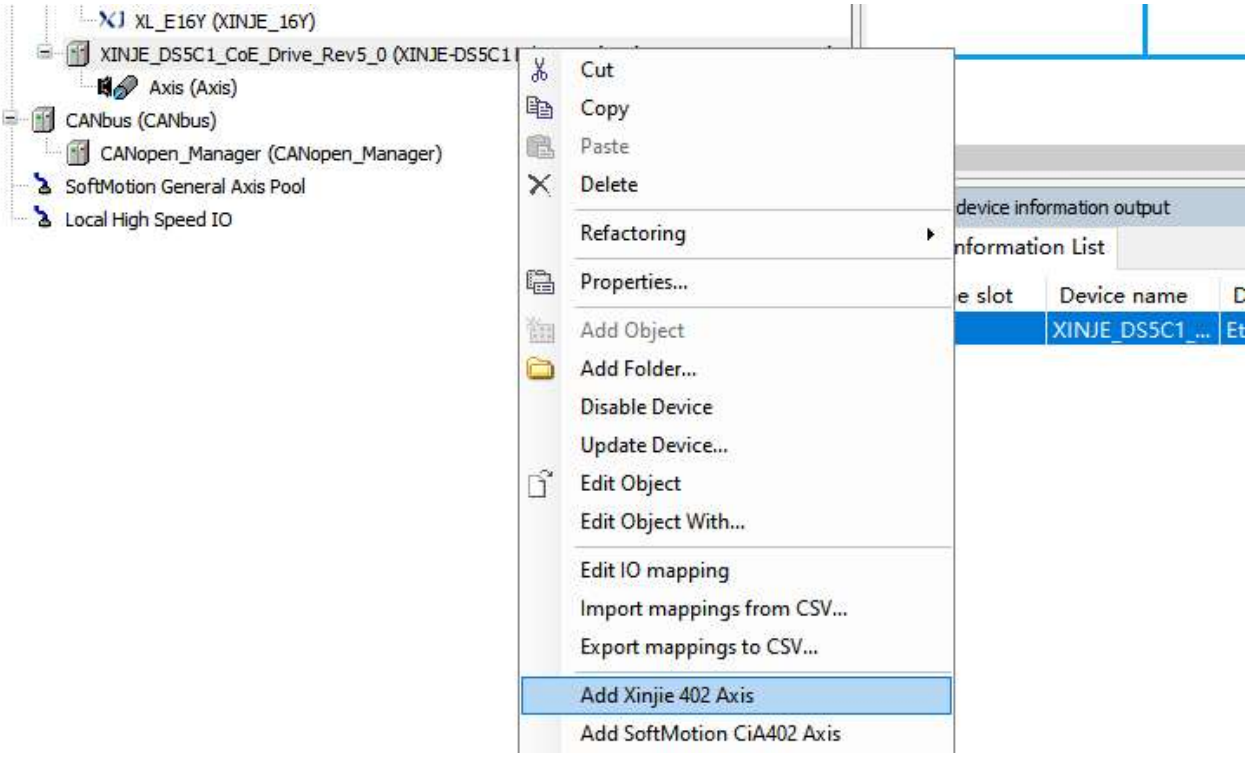

4. Drag module

By selecting a module and holding down the left mouse button, drag the module to the target slot position and release the mouse. The drag operation includes exchanging positions between two modules or dragging a module into an empty slot, but does not support the drag operation of modules between two expansion racks.

## 3-1-3. Device tree operations

1. Xinje axis 402

EtherCAT servo supported adding xinje axis 402.




Xinje axis 402 supports Homing interface configuration.

### 3-1-4. Configuration editing error localization

Configuration equipment has established some configuration rules and error detection mechanisms. For example, in network configuration, the station numbers of two MODBUS devices are the same, or the IP addresses of TCP devices are the same; The slave device on the expansion rack in the hardware configuration device is not connected to an IO module; The number of non disabled axes mounted on EtherCAT exceeding the supported range can cause configuration compilation errors.

When compiling a project, if there is a configuration error, it will be displayed in the XS Studio message output box. Double clicking on the error list can automatically locate the corresponding configuration interface.


## 3-2. MODBUS communication

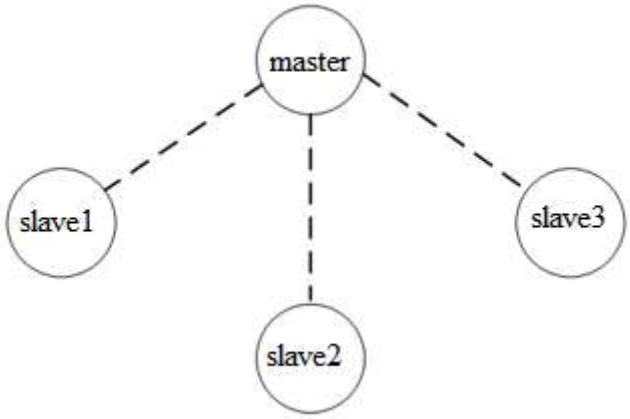
XS Studio supports Modbus protocol communication in both master and slave formats.

Main station form: When the programmable controller is used as the main station equipment, it can communicate with other slave devices using the Modbus protocol; Exchange data with other devices. Example: The Xinje XS series PLC can control the frequency converter through communication.

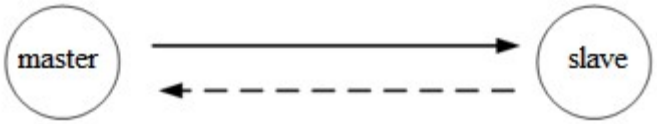
Slave form: When a programmable controller is used as a slave device, it can only respond to the requirements of other master stations.

The concept of master-slave: In the RS485 network, there can be one master-slave at a certain time (as shown in the figure below), where the master station can perform read and write operations on any of the slave stations, and data exchange between the slave stations cannot be directly carried out. The master station needs to write communication programs to read and write one of the slave stations, and the slave stations do not need to write communication programs. They only need to respond to the read and write operations of the master station.

(Wiring method: All 485+ connected together, all 485- connected together).




In the RS232 network (as shown below), only one-on-one communication is allowed, and there is only one master and one slave at a certain time.

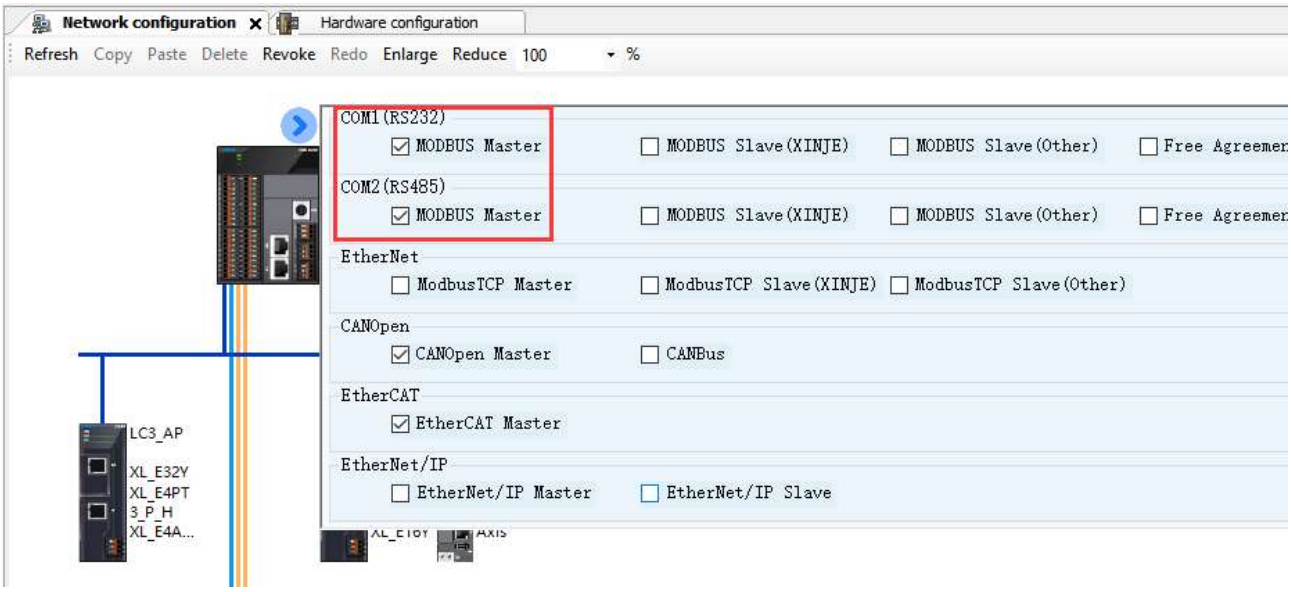


The reason why there are dashed arrows in the figure (including in RS485 networks) is because theoretically, in two networks, as long as each PLC does not send data, any PLC in the network can be used as the master station, and the other PLCs can be used as the slave stations; However, due to the lack of a unified clock reference between multiple PLCs, it is easy for multiple PLCs to send data at the same time, which can lead to communication conflicts and failures. Therefore, it is not recommended to use this method.

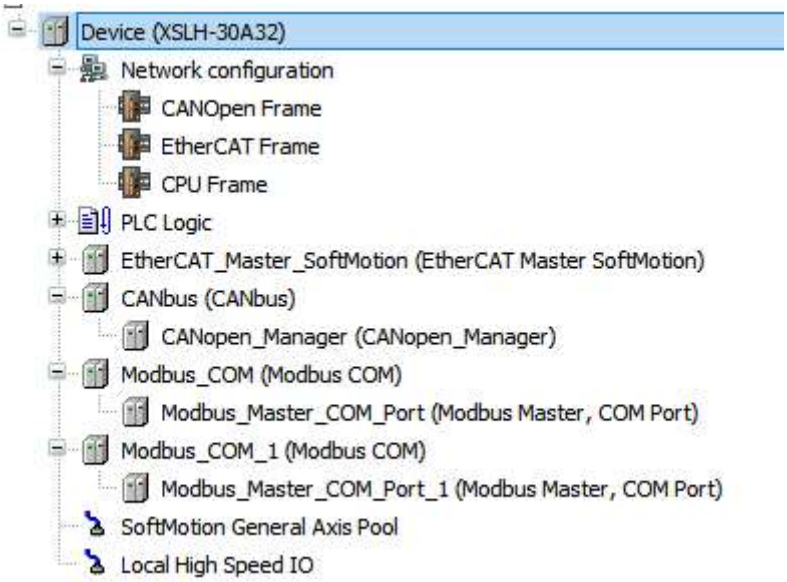
## 3-2-1. MODBUS master station configuration

1. Enable and add master station

Clicking on the PLC device in the network configuration will display the enabling window for the master/slave stations supported within the PLC. As shown in the following figure: Click the checkbox button in the window to enable the master/slave functions supported by the CPU, and then click "MODBUS" from the "Network Connection Device List" on the right side of the view to add the slave to the network diagram.

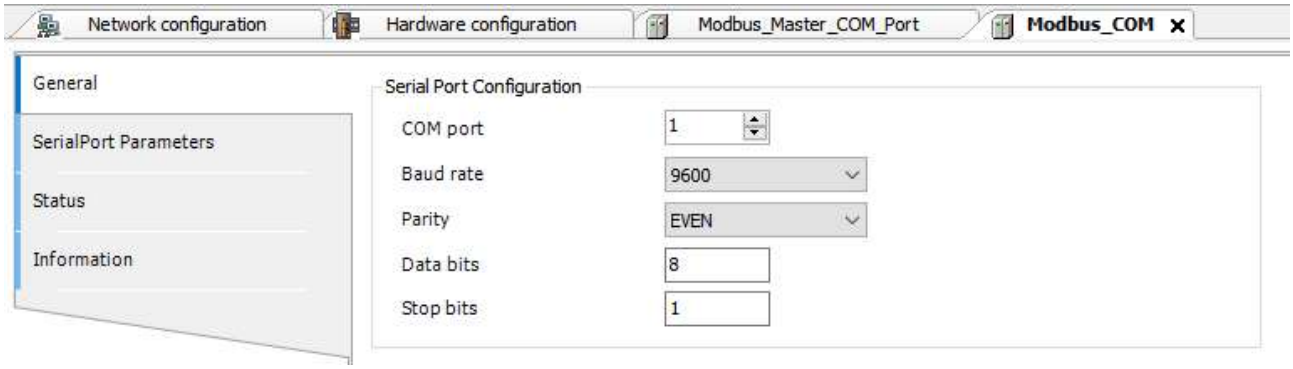



At this point, the Modbus configuration corresponding device tree will appear in the left side view of the interface, as shown in the following figure:



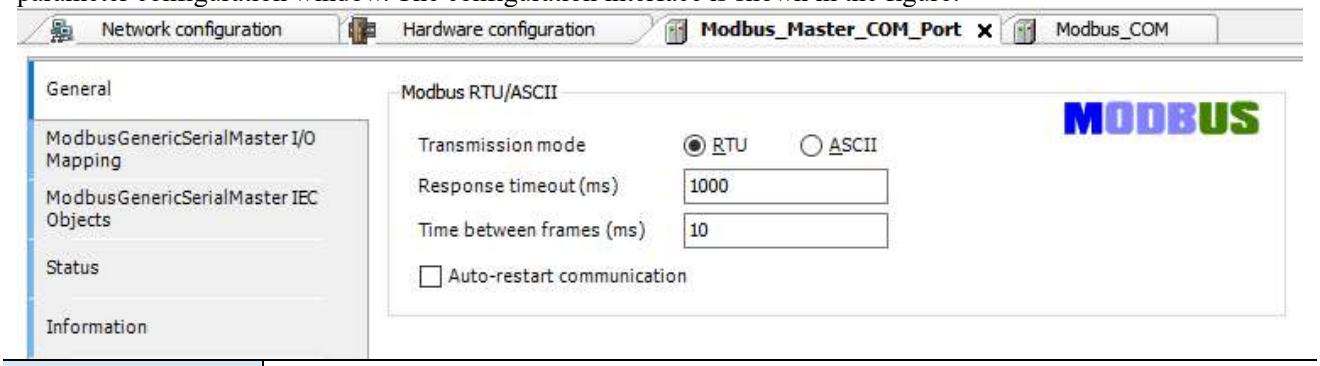

2. Master station communication configuration

When using PLC as the Modbus master station, double-click on the "Modbus COM" node in the left device tree to open the Modbus communication configuration interface. The relevant configuration interface is as follows:




| | |
|---|---|
| COM port | The physical connection of the main station is selected as either serial port 0 or serial port 1 |
| Baud rate | Rate during communication |
| Parity | Verification method for communication frames |
| Data bits | The actual data bits contained in the communication frame |
| Stop bits | Representing the last bit of a single packet during communication |

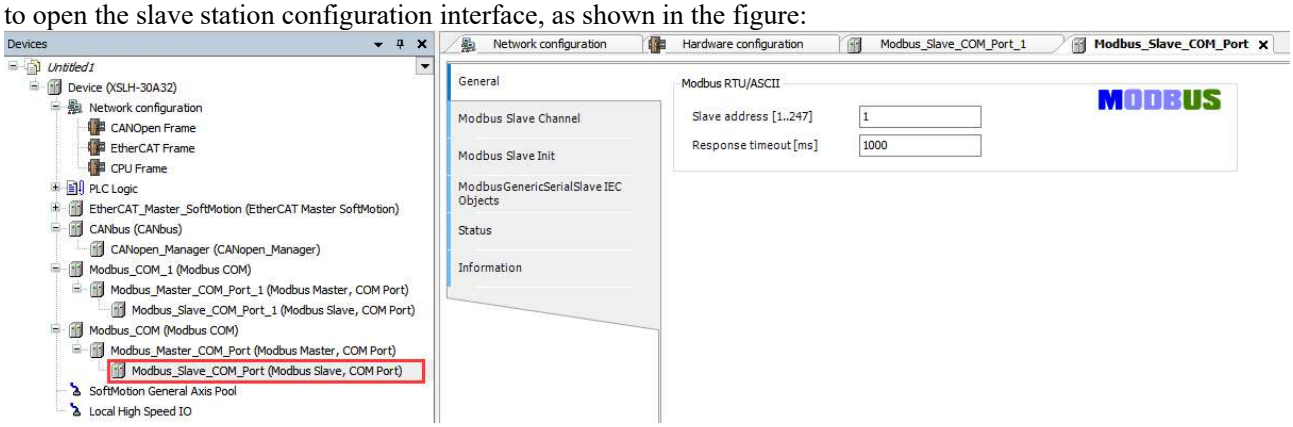
Double click on the main station device in the device tree to open the Modbus main station communication parameter configuration window. The configuration interface is shown in the figure:




| | |
|---|---|
| Transmission mode | Choose RTU or ASCII code |
| Response timeout (ms) | The time interval between the master station and the slave station for response. If there is no response from the slave station during this period, the master station will request the next slave station. At this point, the input value will be considered as the default value for each slave station. On the slave configuration page, each slave can be individually set with an appropriate time interval |
| Time between frames (ms) | The time interval between the main station receiving the previous response data frame and the next request data frame. This parameter can be used to adjust the data exchange rate |

At this point, the configuration of the master station is complete. Next, it is necessary to configure the slave stations connected to the master station accordingly.

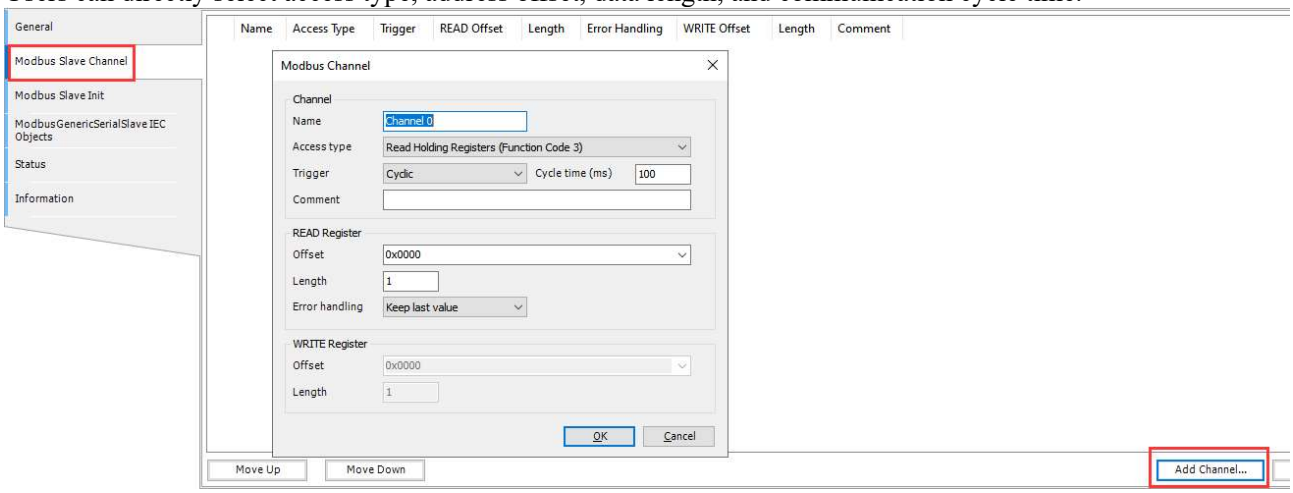
After the master station configuration is completed, double-click the MODBUS (Modbus Slave, COM Port) node to open the slave station configuration interface, as shown in the figure:



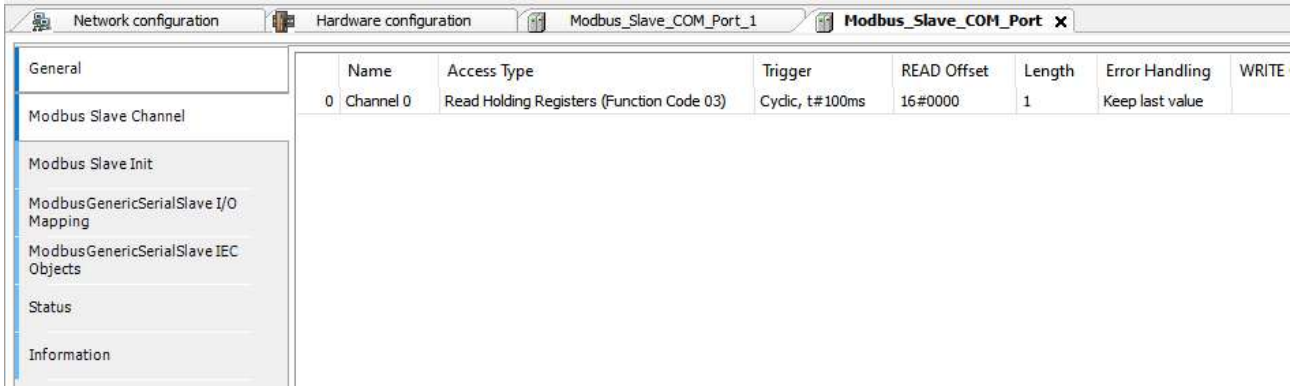

Slave address: Set the slave address, valid from 1 to 247.

Response timeout: Set the response timeout time for the slave station. If the slave station has not responded to the master station after this time, the master station considers that the slave station has a communication failure.

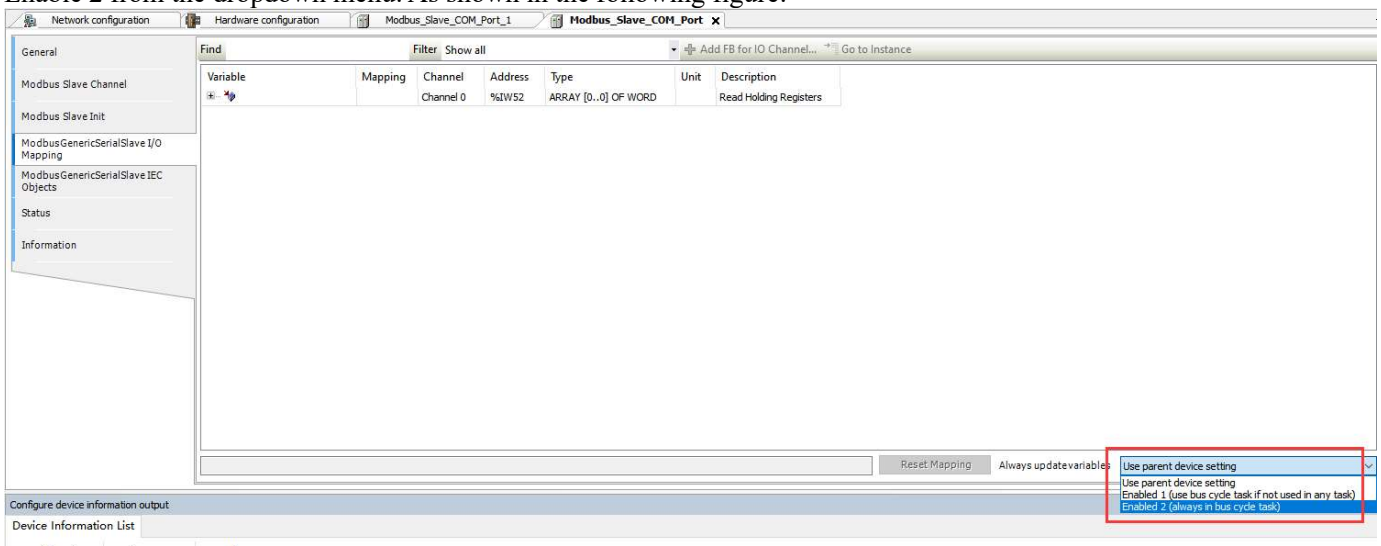
Set the communication channel of the slave station as shown in the figure. In this setting option, users can customize the Modbus communication channel of the slave station, but it must match the actual slave station hardware. After clicking "Add Channel", the system will automatically pop up the Modbus Channel dialog box. Users can directly select access type, address offset, data length, and communication cycle time.




After successfully adding the channel, as shown in the following figure:

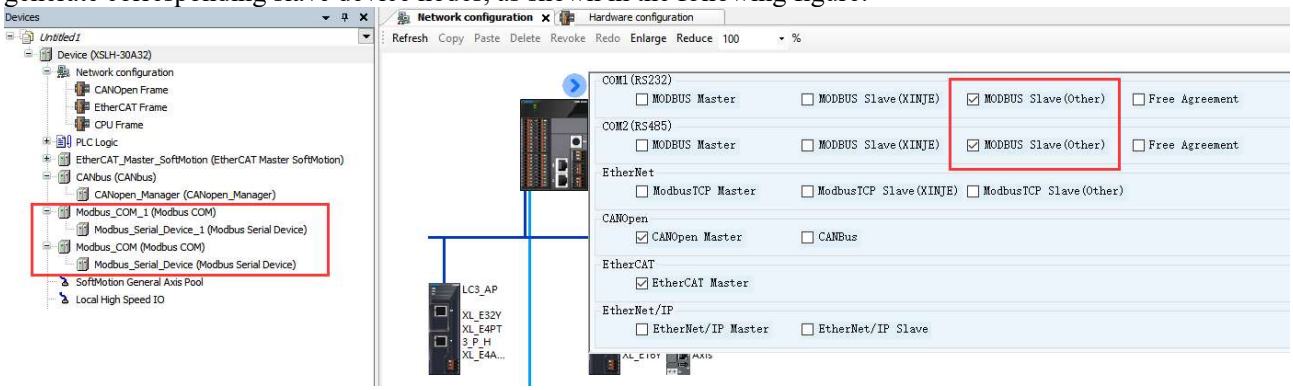



Here, users need to set "keep updating variables" according to their actual needs. They can select Enable 1 or Enable 2 from the dropdown menu. As shown in the following figure:

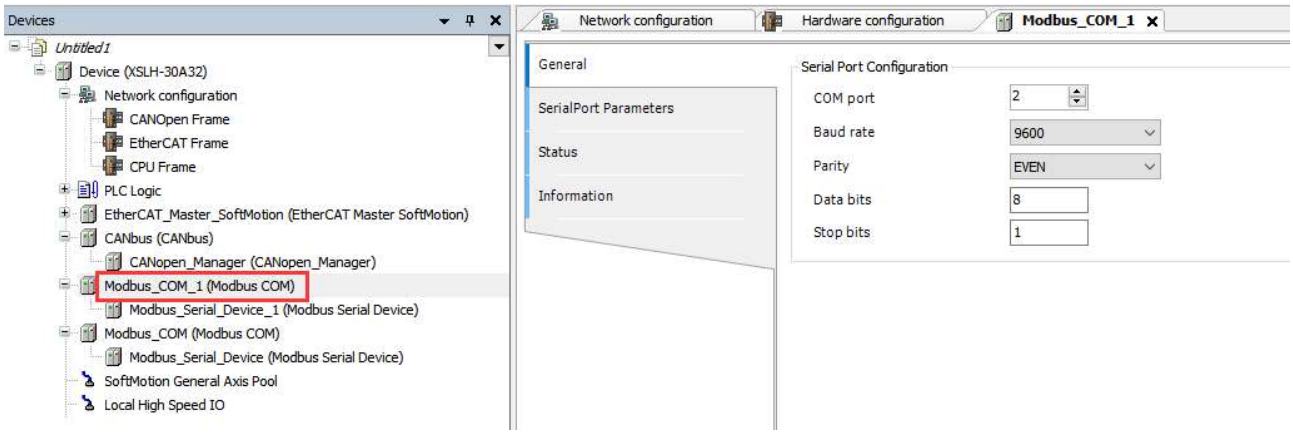


## 3-2-2. MODBUS slave station configuration

Slave devices can be enabled through the enable window in the network configuration interface. The left view will generate corresponding slave device nodes, as shown in the following figure:

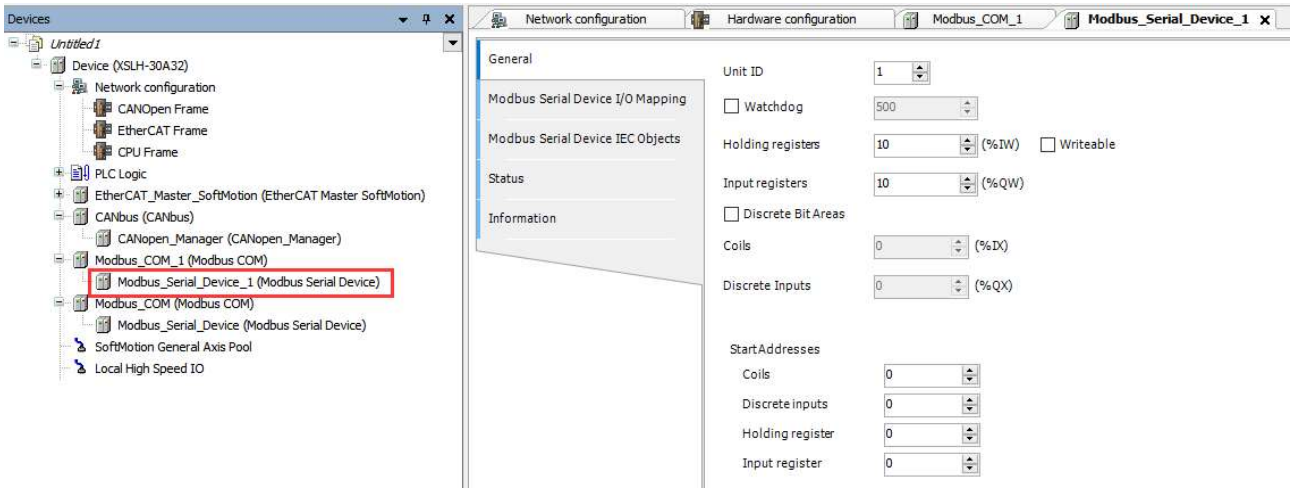



After adding the slave device, double-click MODBUS_COM_1(Modbus COM) node to open the configuration interface and can switch to the Modbus slave communication configuration interface. As shown in the following figure:

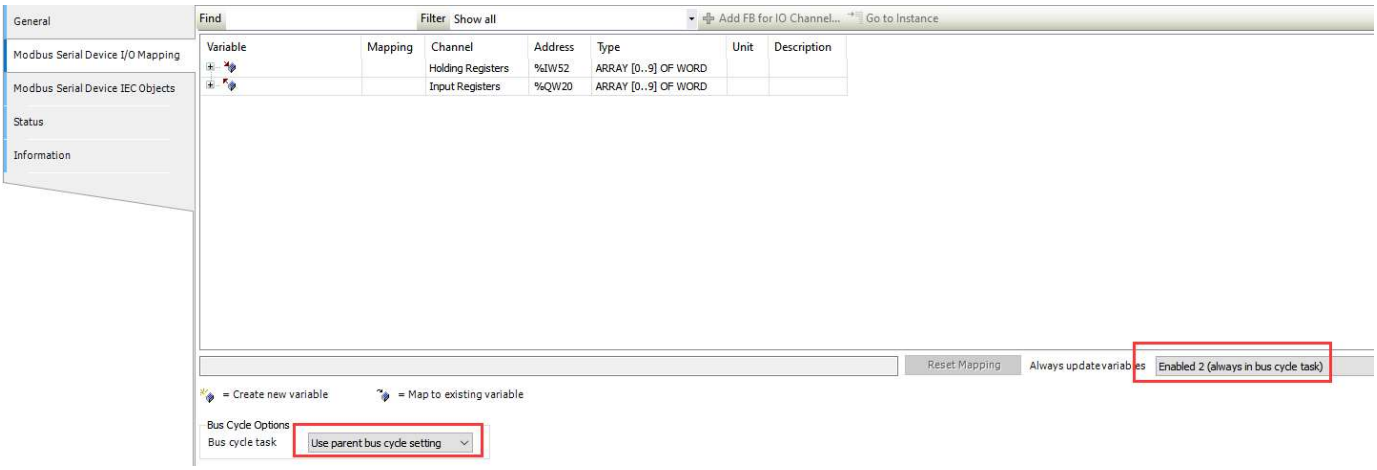



| | |
|---|---|
| COM port | The serial port number selected by the master station in the network configuration |
| Baud rate | Rate during communication |
| Parity | Verification method for communication frames |
| Data bits | The actual data bits contained in the communication frame |
| Stop bits | Representing the last bit of a single packet during communication |

Click the node “Modbus_Serial_Device(Modbus Serial Device)” in the device tree to open Modbus Slave communication data configuration interface. As shown in the following figure:

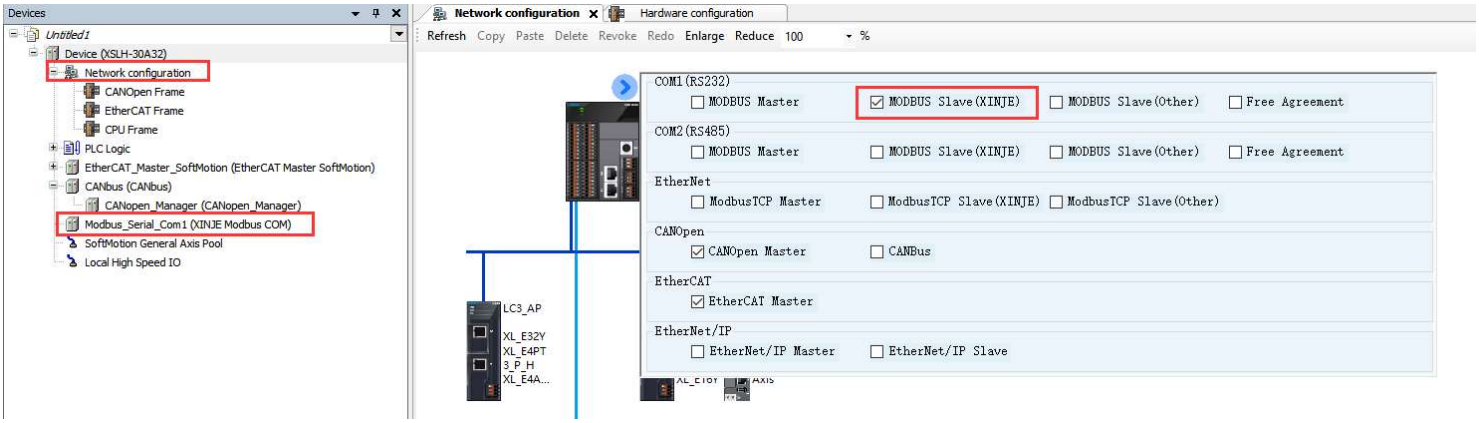



Switch to "Modbus Serial Device I/O Mapping" in this window, and the user needs to set "Bus Loop Options" and "Always Update Variables" according to actual needs, as shown in the following figure:

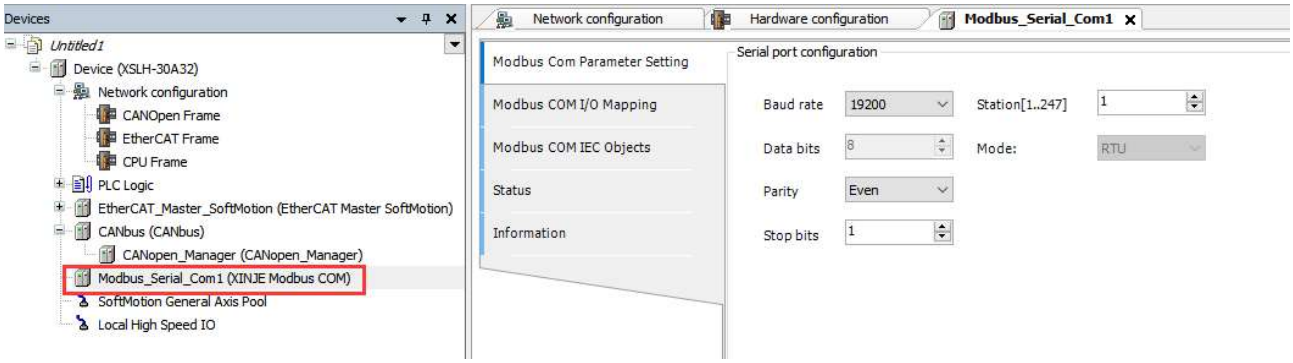



## 3-2-3. MODBUS RTU (XINJE) slave setting

1. Double click on the "Network Configuration" node from the left device tree to open the network configuration interface. Enable the Modbus slave (XINJE) device through the enable window, and a "Modbus Serial.Com1" node will be generated in the left device tree. As shown in the following figure




2. Double click on the "Modbus Serial.Com1" node in the left device tree to open the Modbus parameter configuration interface. Relevant serial port parameters can be set according to actual needs. The default situation is shown in the following figure:




■ The configuration parameters of the Modbus slave station are as follows:

| | |
|---|---|
| COM port | The serial port number selected by the master station in the network configuration |
| Baud rate | Rate during communication |

| | |
|---|---|
| Data bits | The actual data bits contained in the communication frame, when the mode is selected as RTU and the mode data is 8 bits |
| Parity | Verification method for communication frames |
| Stop bits | Representing the last bit of a single packet during communication |
| Mode | RTU |
| Station | The station number of this device, ranging from 1 to 247 |

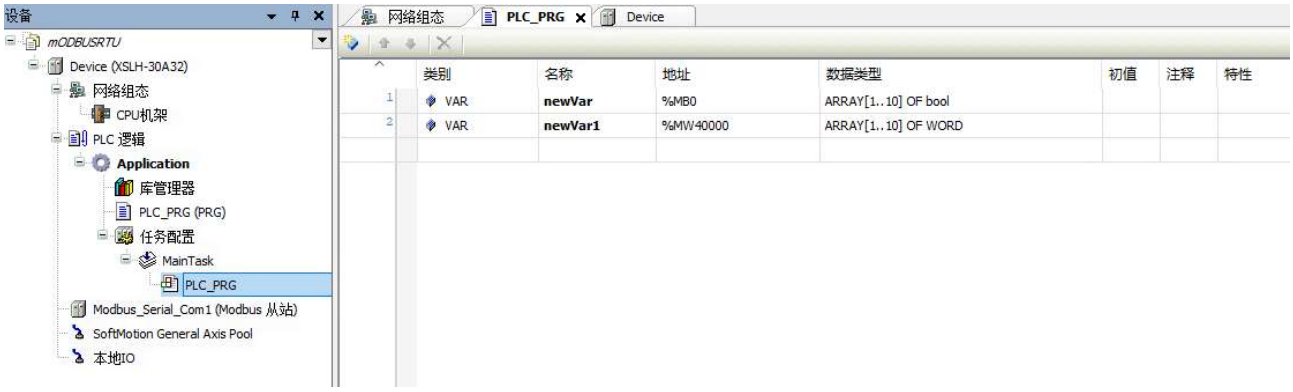
■ When using a PLC as a Modbus RTU (XINJE) slave device, the address range that can be accessed by the master device is defined as follows:

◆ All the coils (function code 0x01, 0x02, 0x05, 0x0F). The read-write address is: %MB0-%MB65534;
◆ All the registers (function code 0x03, 0x04, 0x06, 0x10). The read-write address is: %MW40000-%MW105534.

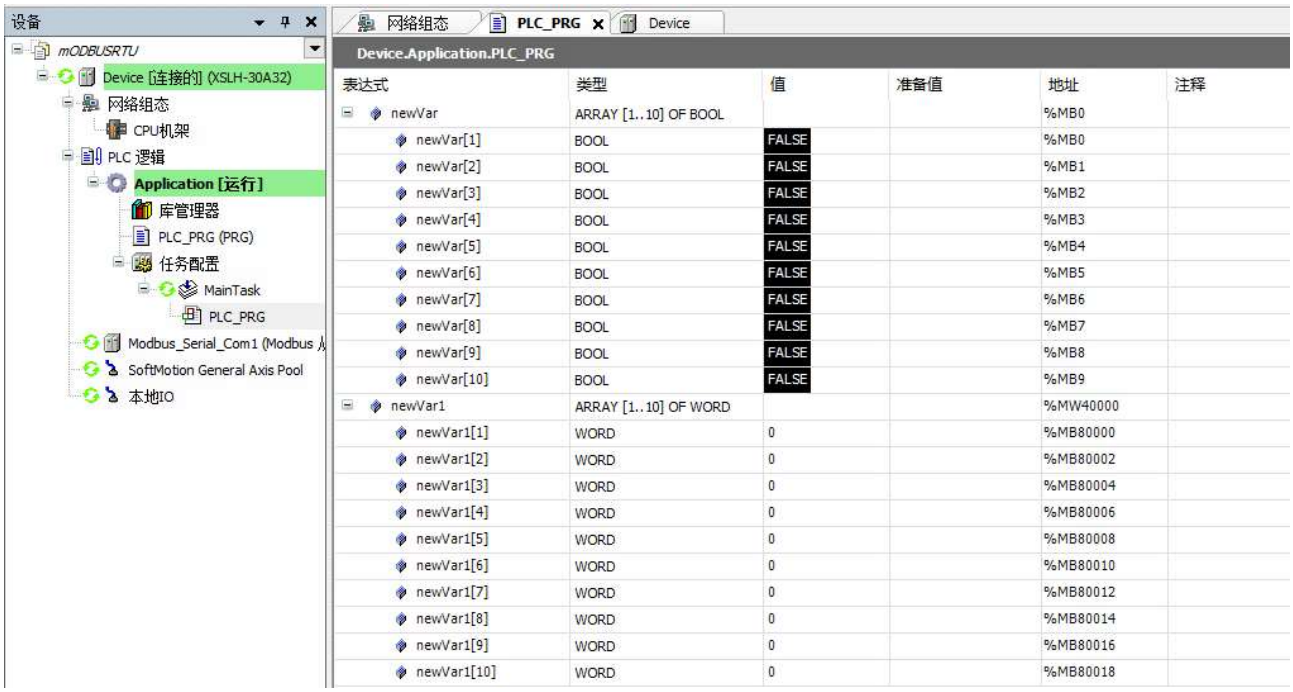
3. Application example

Here, XS Studio software serves as a slave station and uses third-party debugging tool Modbus Poll as the master station to establish connections and perform serial communication, enabling the reception or transmission of register or coil data.

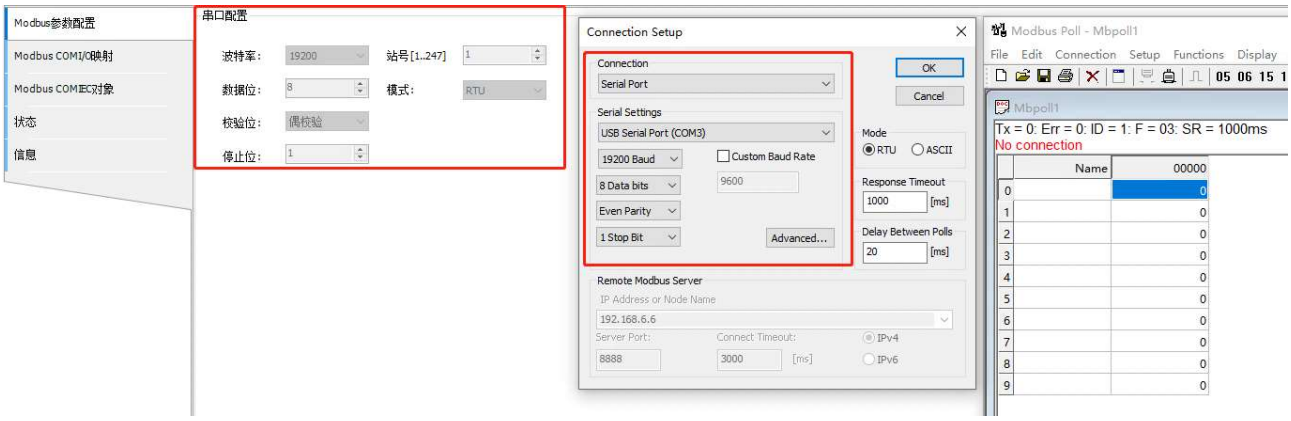
(1) Declare two variables in the "PLC-PRG" editor to receive and send register or coil data, respectively. As shown in the following figure:




(2) Establish a connection with the XSLH-30A32 device and log in to run it. As shown in the following figure:

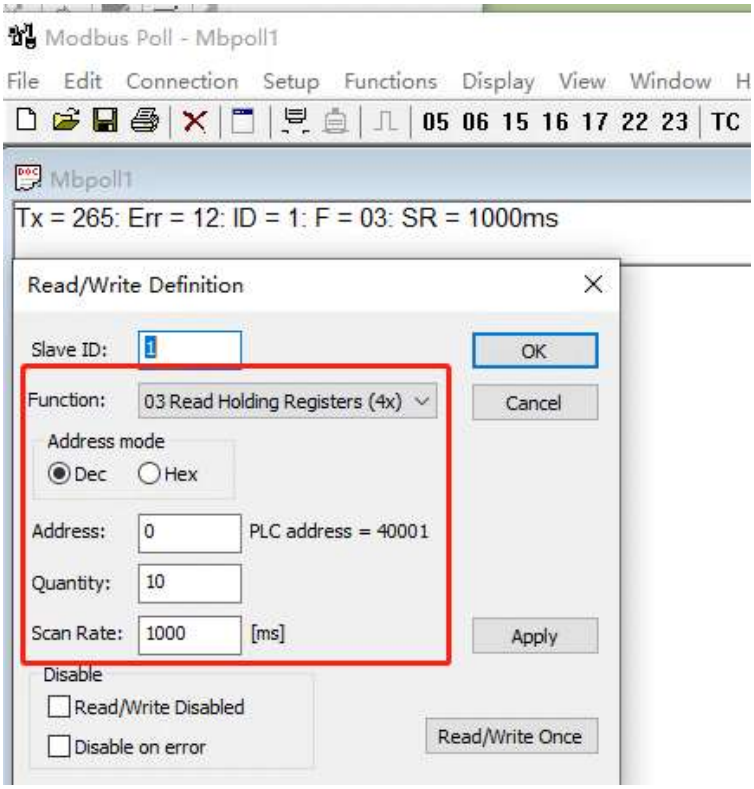


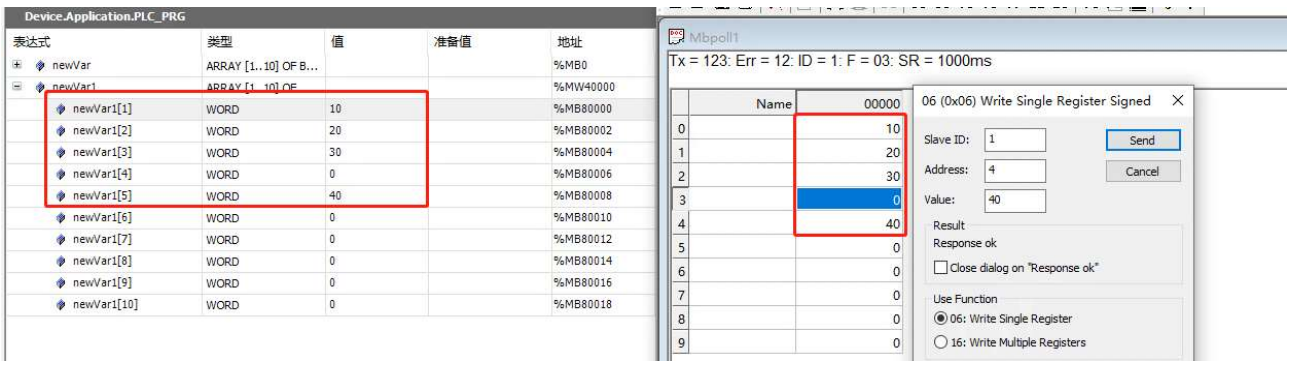

(3) Configure the relevant parameters of Modbus Poll to be consistent with the serial port configuration information of the slave station, otherwise the connection cannot be successfully established. As shown in the following figure:



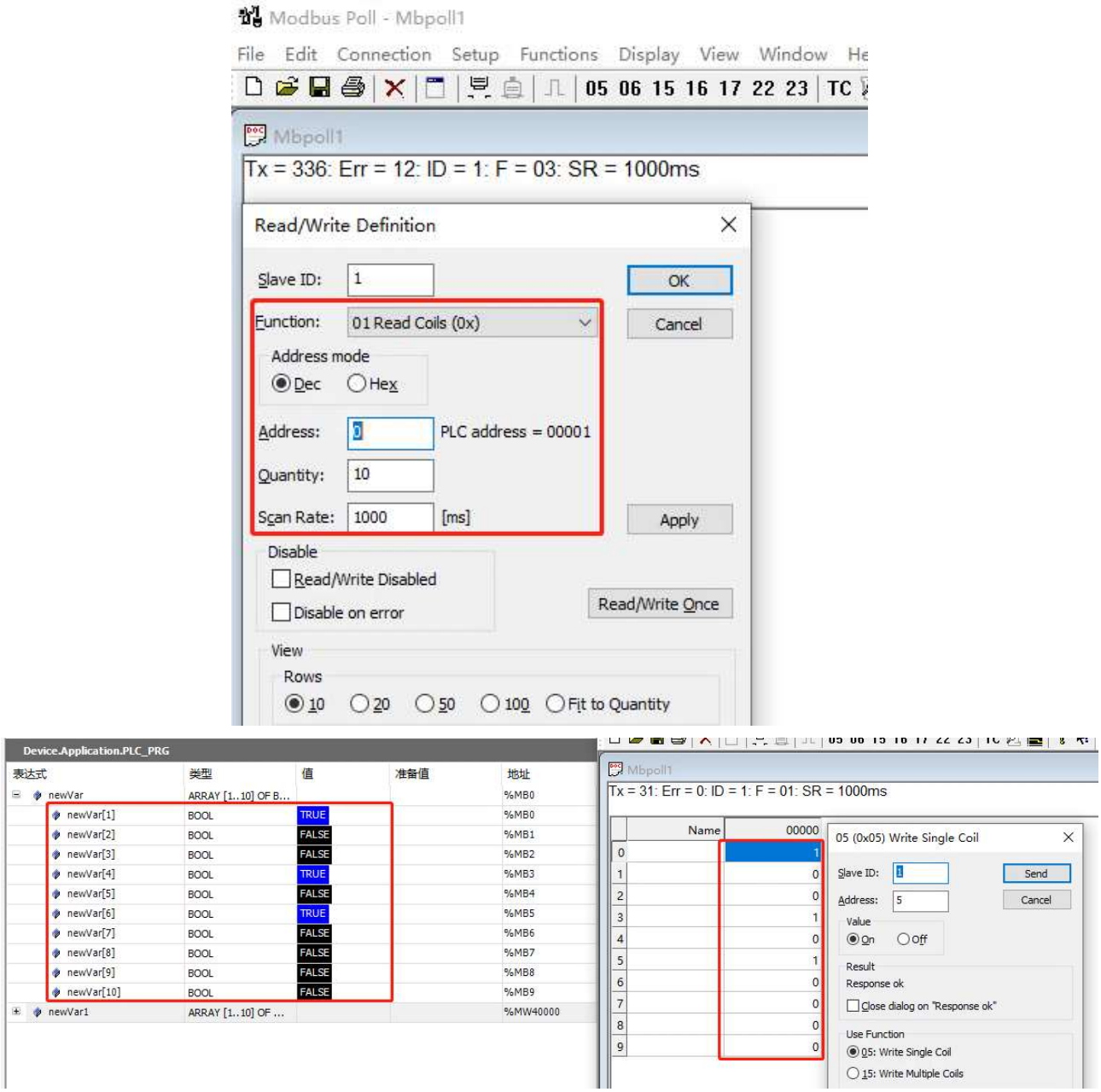

(4) Configure the relevant read and write parameters of the master station equipment to perform related read/write register or coil operations with the slave station.

- Read/write register operations, as shown in the following figure



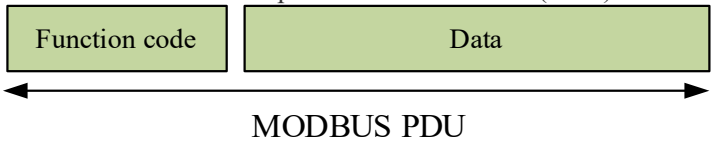
- Read/write coil operation. As shown in the following figure




At this point, the master and slave stations have successfully communicated.

## 3-2-4. MODBUS communication frame

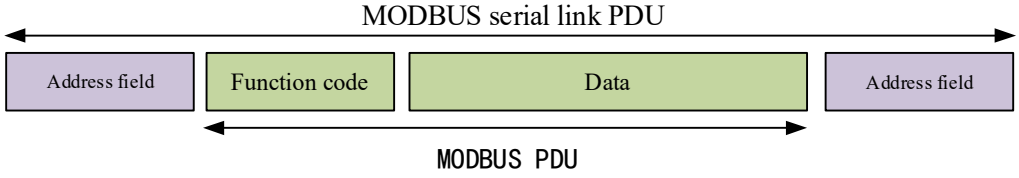
The Modbus Application Protocol defines a simple Protocol Data Unit (PDU):




MODBUS PDU

Modbus Protocol data unit

The client that initiates Modbus transactions constructs a Modbus PDU, and then adds additional domains to construct a communication PDU.




Modbus data frames on the serial link

1. In the Modbus serial link, the address field only contains sub node addresses.

As mentioned earlier, the legitimate sub node addresses are decimal 0-247, and each sub device is assigned an address within the range of 1-247. The master node addresses the sub nodes by placing their addresses in the address field of the message. When a sub node returns a reply, it places its own address in the address field of the reply message to let the master node know which sub node is answering.

2. The function code indicates the action that the server needs to perform. The function code can be followed by a data field representing both request and response parameters.

3. The error checking domain is the calculation result of performing a redundancy check on the message content.

Use two different calculation methods based on different transmission modes (RTU or ASCII).

There are two serial transmission modes defined: RTU mode and ASCII mode. All devices must implement RTU mode, and ASCII transmission mode is an option. Modbus RTUs typically use serial ports RS232C or RS485/422, while Modbus TCP typically uses Ethernet ports.

- RTU transmission mode

When the device uses RTU (Remote Terminal Unit) mode to communicate on the Modbus serial link, each 8-bit byte in the message contains two 4-bit hexadecimal characters. The main advantage of this mode is its high data density and higher throughput than ASCII mode at the same baud rate. Each message must be transmitted in a continuous character stream.

**The format of each byte (11 bits) in RTU mode is:**

Encoding system: 8-bit binary, each 8-bit byte in the message contains two 4-bit hexadecimal characters (0-9, A-F)

**Bit stream per byte:**

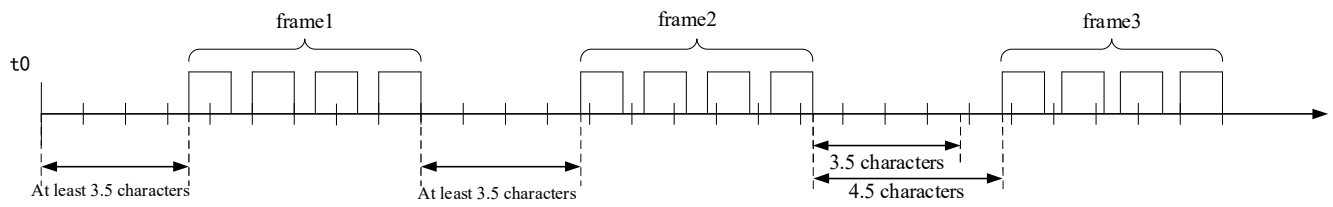
- 1 Starting bit
- 8 data bits, first send the least significant bit
- 1 bit as parity check
- 1 stop bit

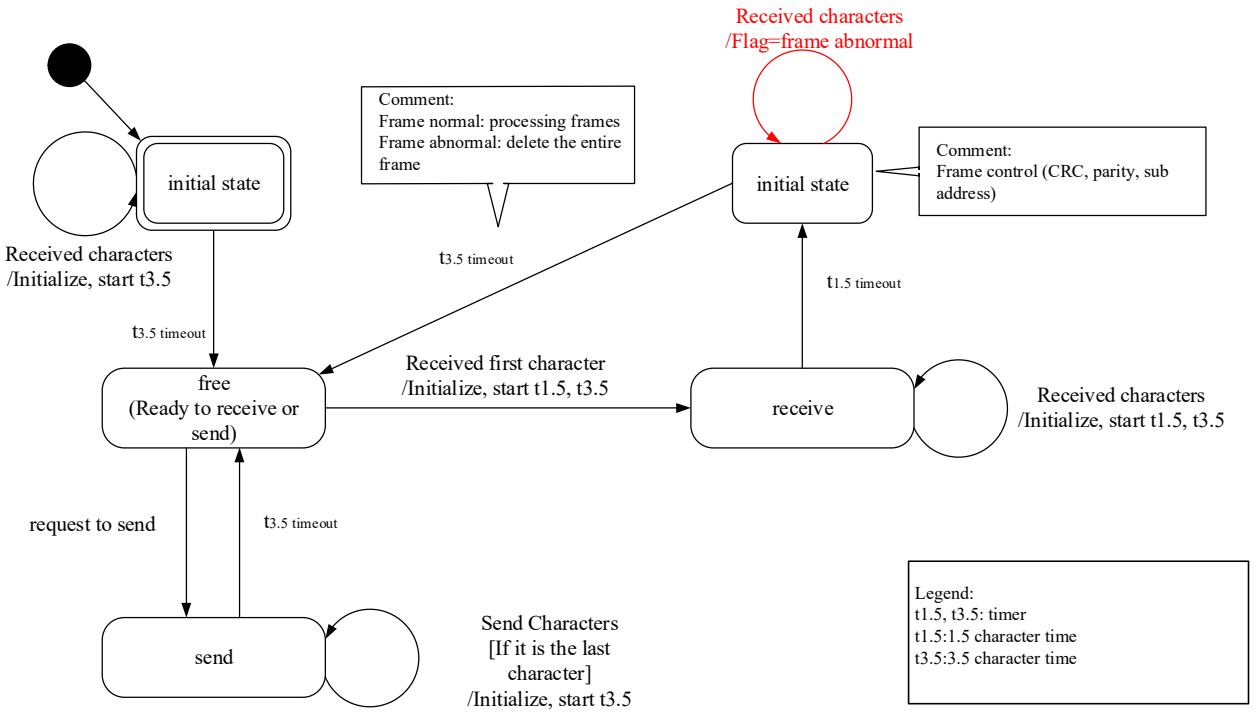
Even parity is required, and other modes (odd parity, no parity) can also be used. To ensure maximum compatibility with other products and support no verification mode, it is recommended. The default verification mode must be even verification.

Note: Using no verification requires 2 stop bits.

| Address code | Function code | Data | Parity |
|---|---|---|---|
| 1 byte | 1 byte | N byte | 2 bytes (CRC) |

In RTU mode, message frames are distinguished by idle intervals with a duration of at least 3.5 characters. In the subsequent section, this time interval is referred to as t3.5.






- The transition from "initial" state to "idle" state requires a t3.5 timed timeout: this ensures inter frame delay;
- The "idle" state is a normal state where no messages are sent or received to be processed;
- In RTU mode, when there is no active transmission with a time interval of 3.5 characters, the communication link is considered to be in an "idle" state.
- When the link is idle, any transmitted characters detected on the link are recognized as the beginning of the frame. The link becomes active. Then, when the time interval for no character transmission on the link reaches t3.5, it is recognized as the end of the frame
- After detecting the end of the frame, complete CRC calculation and verification. Then, analyze the address domain to determine whether the frame is sent to this device, and if not, discard the frame. To reduce reception processing time, the address domain can be analyzed as soon as it is received, without waiting until the end of the entire frame. In this way, CRC calculation only needs to be performed when the frame is addressed to that node (including broadcast frames).

■ ASCII transmission mode

When the devices on the Modbus serial link are configured to communicate in ASCII mode, each 8-bit byte in the message is sent as two ASCII characters. When the communication link or device cannot comply with the timing management of RTU mode, this mode is used.
Note: Due to the requirement of two characters per byte, this mode is less efficient than RTU.

The format of each byte (11 bits) in ASCII mode is:

Encoding system: hexadecimal, ASCII characters 0-9, A~F. Each ASCII character in the message contains 1 hexadecimal character

**Bit stream per byte:**

- 1 Starting bit
- 8 data bits, first send the least significant bit
- 1 bit as parity
- 1 stop bit

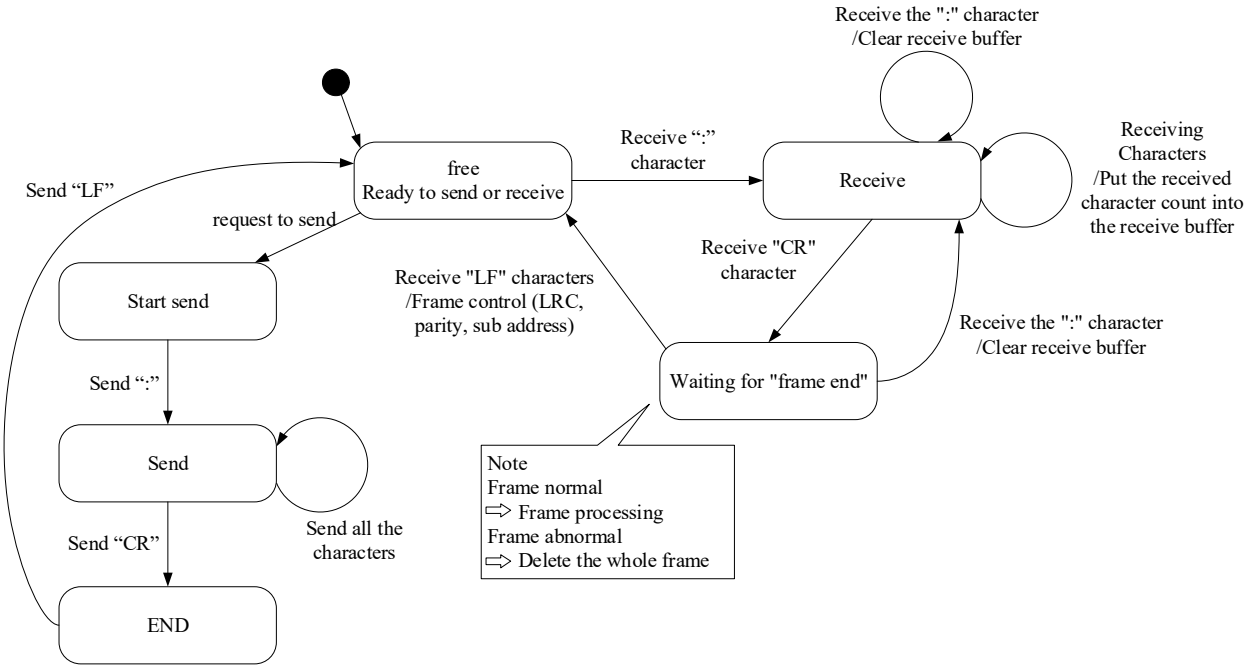
Even parity is required, and other modes (odd parity, no parity) can also be used. To ensure maximum compatibility with other products and support no verification mode, it is recommended. The default verification mode must be even verification.

Note: Using no verification requires 2 stop bits.

| Start | Address code | Function code | Data | Parity | Enter |
|---|---|---|---|---|---|
| Character ":" (colon) | 2 bytes | 2 bytes | 0 to 2*252 bytes | 2 bytes (LRC parity) | 2 bytes (CR, LF) |

The address field of the message frame contains two characters.
In ASCII mode, messages use special characters to distinguish between the beginning and end of a frame. A

message must start with a colon (:) (ASCII hexadecimal 3A) and end with a carriage return line feed corresponding to ASCII hexadecimal 0D and 0A.
For all domains, the allowed transmitted characters are hexadecimal 0-9, A~F (ASCII encoding). The device continuously monitors the colon character on the bus. After receiving this character, each device decodes the subsequent characters until the end of the frame.
The time interval between characters in the message can reach one second. If there is a larger interval, the receiving device believes that an error has occurred.




ASCII transmission mode state diagram

- The "idle" state is a normal state where no messages are sent or received to be processed.
- Each time a ":" character is received, it indicates the beginning of a new message. If the character is received during the receiving process of a message, the current message is considered incomplete and discarded. And a new receive buffer is reallocated.
- After detecting the end of the frame, complete LRC calculation and verification. Then, analyze the address domain to determine whether the frame is sent to this device, and if not, discard the frame. To reduce reception processing time, the address domain can be analyzed as soon as it is received, without waiting until the end of the entire frame.

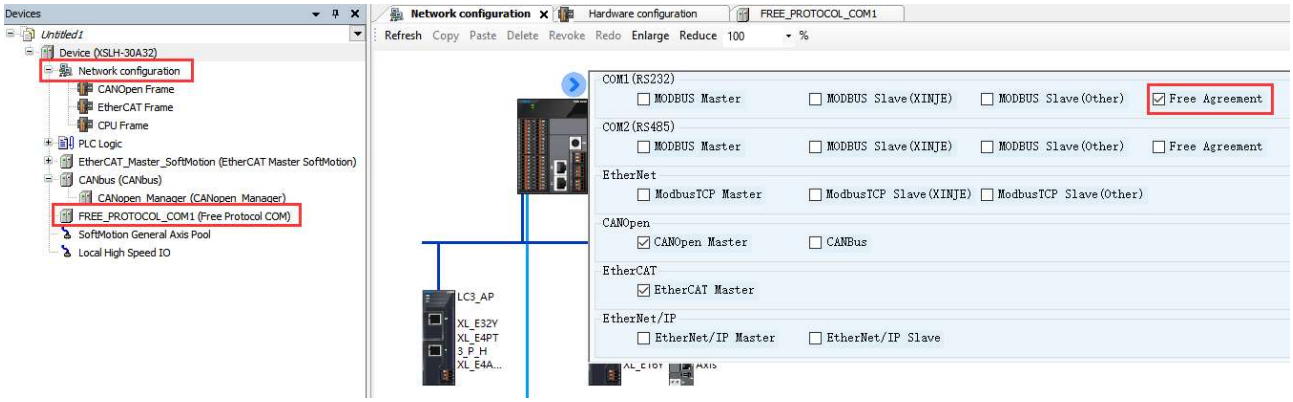
# 3-3. Serial port free format protocol communication

## 3-3-1. Overview

When communicating with other devices, if the Xinje PLC is used as a lower computer, the upper computer must exchange data with it according to the MODBUS RTU data format; If the Xinje PLC is used as the upper computer and the lower computer also supports the MODBUS RTU protocol, relevant communication instructions can be directly used for communication, making program writing simpler and more efficient. If the lower computer does not directly support the MODBUS RTU protocol, free format communication can be used.
The so-called free format refers to when the communication protocol of the lower computer does not match the PLC protocol, the PLC customizes the data format internally to send data, which can communicate with many lower computers.
Free format communication is the transmission of data in the form of data blocks, with a maximum transmission capacity of 1024 bytes per block. At the same time, each block can be set with a start and end symbol, or not set.

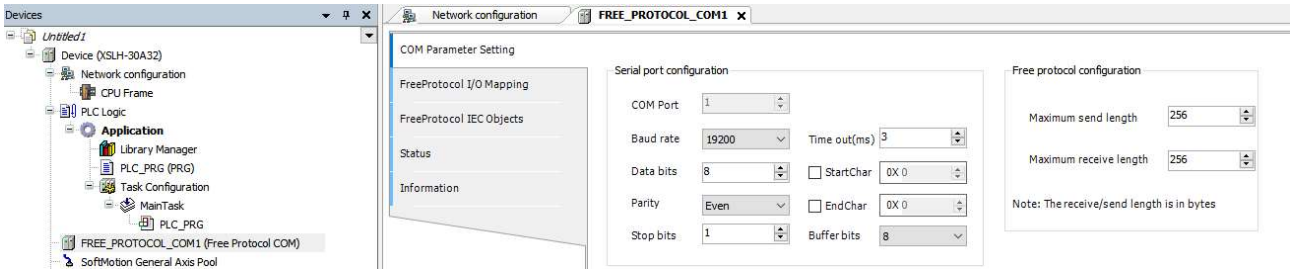
## 3-3-2. Serial port configuration

Taking XSLH-30A32 model equipment as an example, configure the serial port free protocol.
Double click on the "Network Configuration" node from the left device tree to open the network configuration interface. From the enable window, configure "COM1" as "Free Protocol". After configuration is completed, generate the "FREEPPROTOCOL-COM1 (Free Protocol COM)" node in the left device tree. As shown in the following figure.




## 3-3-3. Communication setting

(1) Double click on the "FREE.PROTOCOL-COM1 (Free Protocol COM)" node from the left device tree to open the free protocol parameter configuration interface. The parameter configuration interface is as follows:




■ Serial port parameters

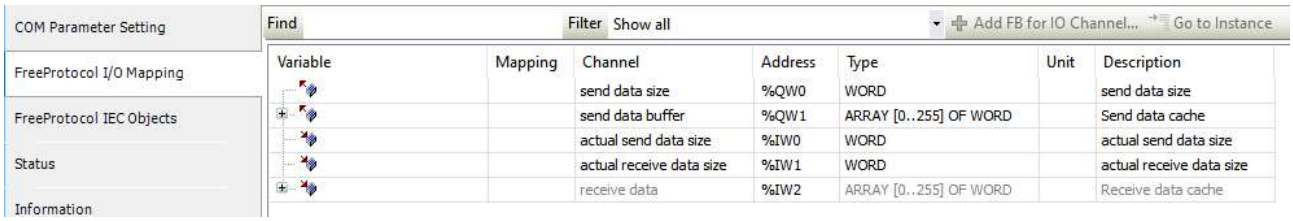
| | |
|---|---|
| COM port | The serial port number of the physical connection of the main station |
| Baud rate | Rate during communication |
| Data bit | The actual data bits contained in the communication frame |
| Parity | Verification method for communication frames |
| Stop bit | Representing the last bit of a single packet during communication |
| Timeout | The waiting time interval between the main station receiving the previous response data frame and the next request data frame |
| Start character | After setting the start symbol, the PLC automatically adds the start symbol when sending data, and automatically removes the start symbol when receiving data, which can be seen as the data frame header in the protocol |
| End character | After setting the end symbol, the PLC automatically adds a end symbol when sending data, and automatically removes the end symbol when receiving data, which can be seen as the end of the data frame in the protocol |
| Buffer bits | The cache bit can be set to 8 or 16 bits. When the cache bit is 8 bits, only the low byte data of the register is sent; When the cache bit is 16 bits, both high and low byte data of the register will be sent, with low bytes first and high bytes last |

■ The configuration of free protocol parameters is as follows

Maximum receive length: The maximum number of data bytes that can be received at a time. The default is 256 bytes, and the maximum allowable setting is 1024 bytes;
Maximum sending length: The maximum number of data bytes that can be sent at once. The default is 256 bytes,

and the maximum allowed setting is 1024 bytes.

(2) Switch from the current free protocol parameter configuration window to the FreeProtocolI/O mapping interface. The specific channels for sending or receiving data are as follows:



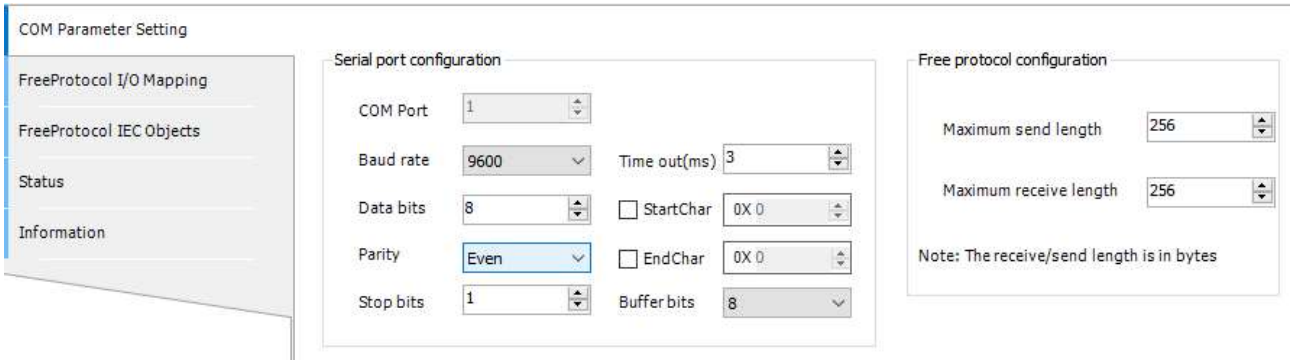

Note: The free protocol I/O mapping parameters are shown in the table below:

| Channel | Description |
|---|---|
| send data size | Send data length; Trigger sending by filling in non-zero data as the number of bytes sent. After sending the data, the register is automatically reset to 0. When sending the data, the preparation state for receiving the data needs to be interrupted, and the preparation state for receiving the data needs to be restored after the data is sent. |
| Send data buffer | Send data cache; Cache the data to be sent in the send data buffer and wait for the signal to be sent to send the data. |
| actual send data size | Actual length of data sent; Display based on the actual number of bytes sent. |
| actual recive data size | Actual received data length; Display based on the actual number of bytes of received data. |
| receive data | Receive data; Store the received data in the corresponding address according to the mode selected by the cache. The received data is in a constantly ready to receive state, except for sending data in a interrupt ready state. |

## 3-3-4. Application example

Here, XS Studio software serves as a slave station and the touch screen serves as the master station to establish a connection and perform Modbus serial port free protocol communication, achieving data reception or transmission.

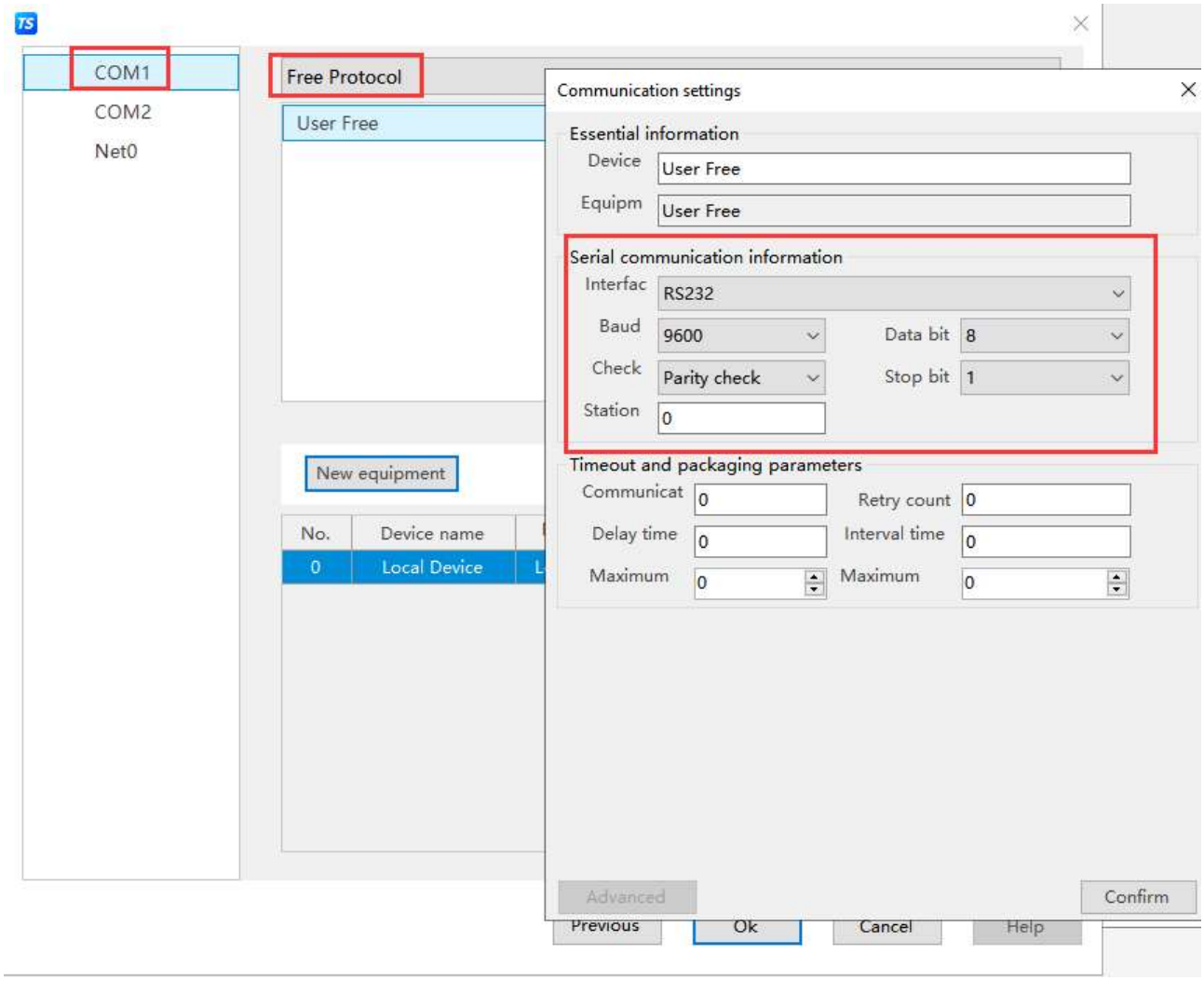
1. Configure the serial port and free protocol parameter configuration within XS Studio software. As shown in the following figure:

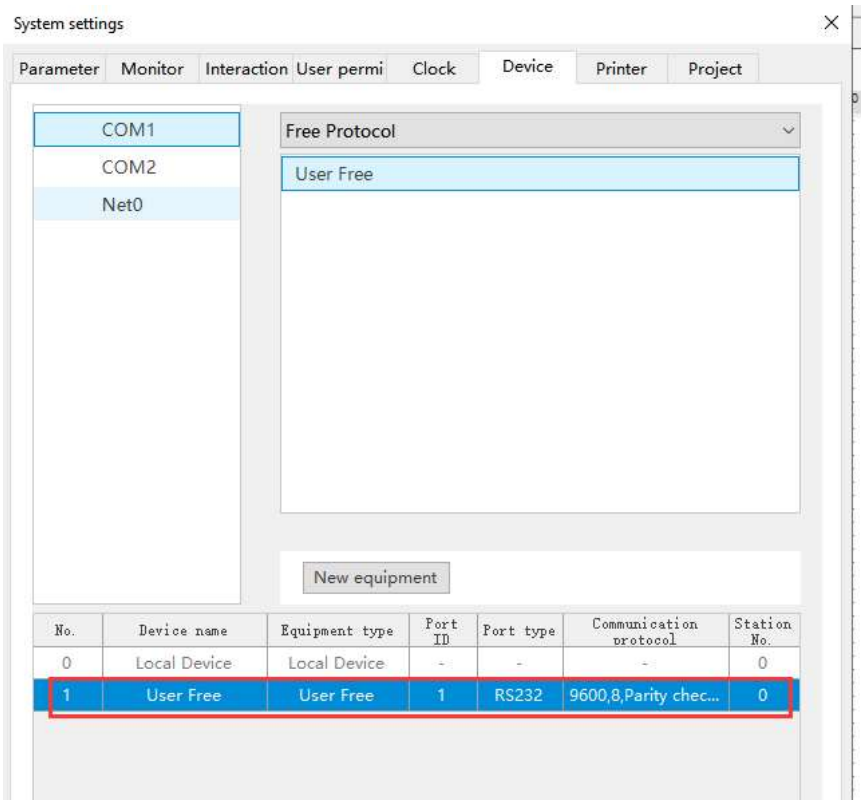



2. Establish connection with PLC, log in and run. As shown in the following figure:

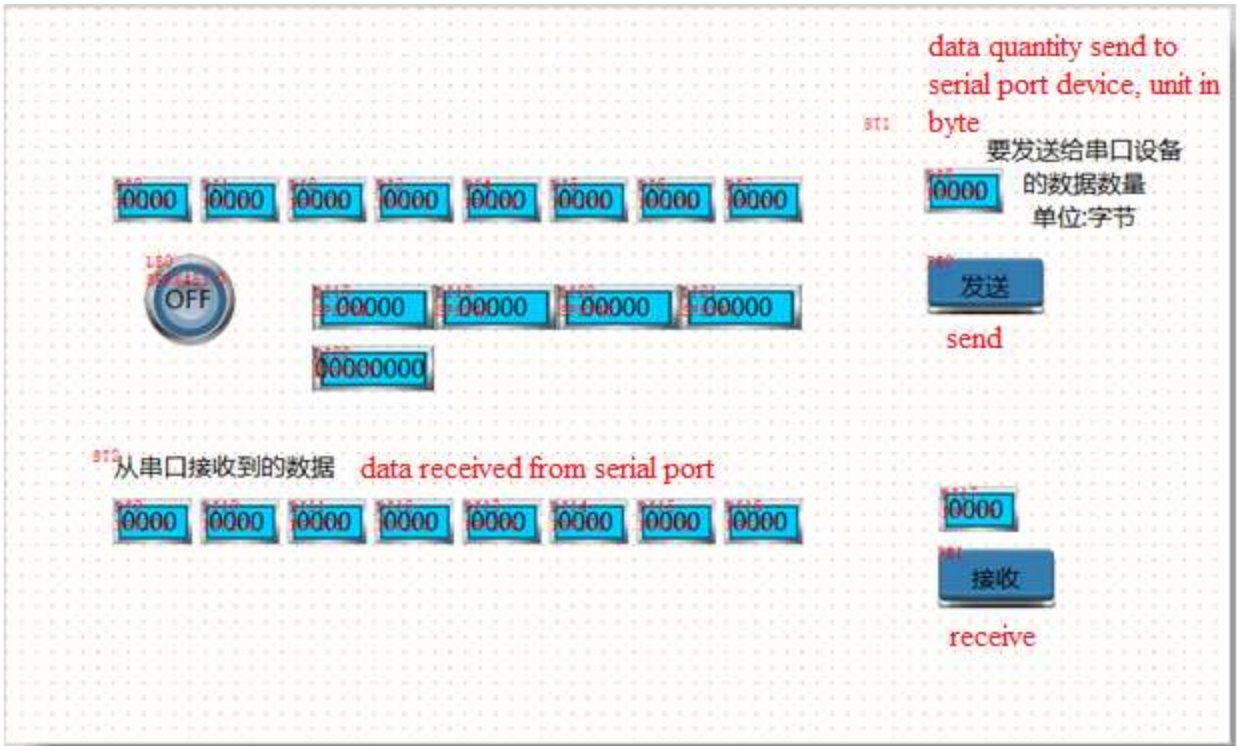

3. Set communication parameters for the touch screen.

Open the "Device" function interface from the "System Settings" window, select COM1->Free Format under Free Communication according to the actual COM port of the PLC device, then create a new device and perform communication settings. The serial port information here needs to be consistent with XS Studio, otherwise correct data exchange cannot be performed. As shown in the following figure:

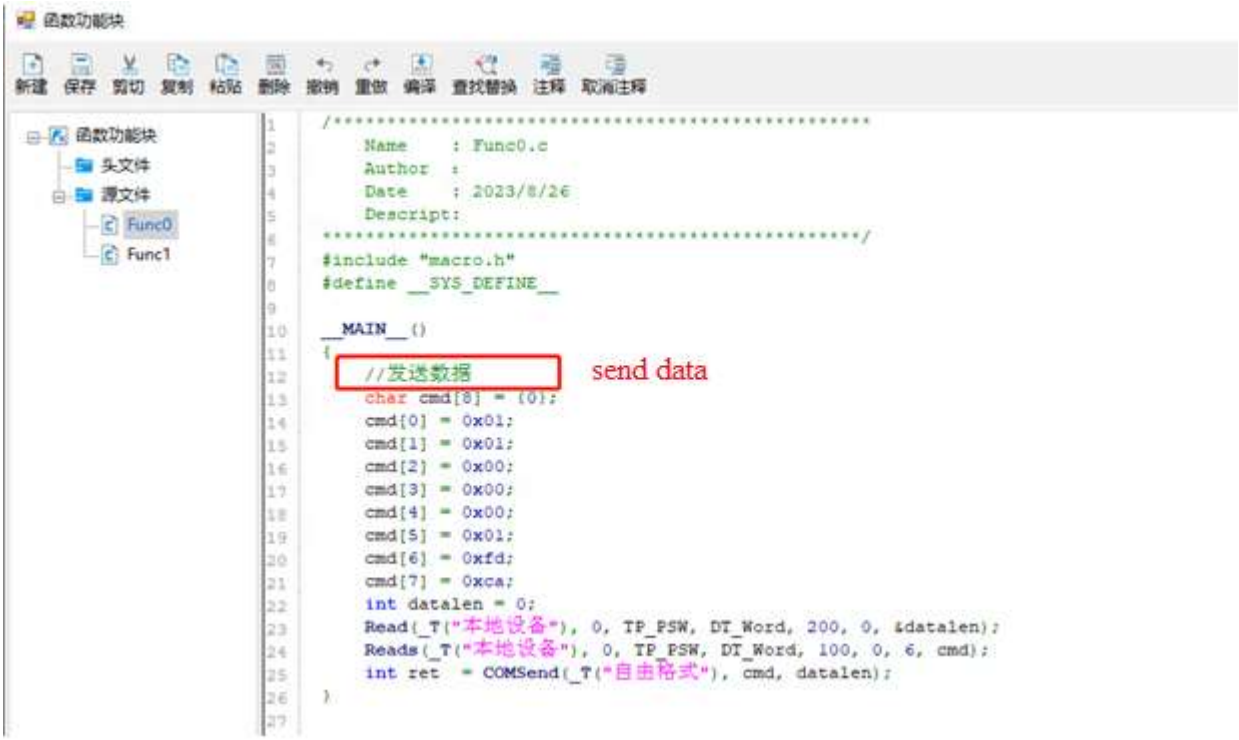


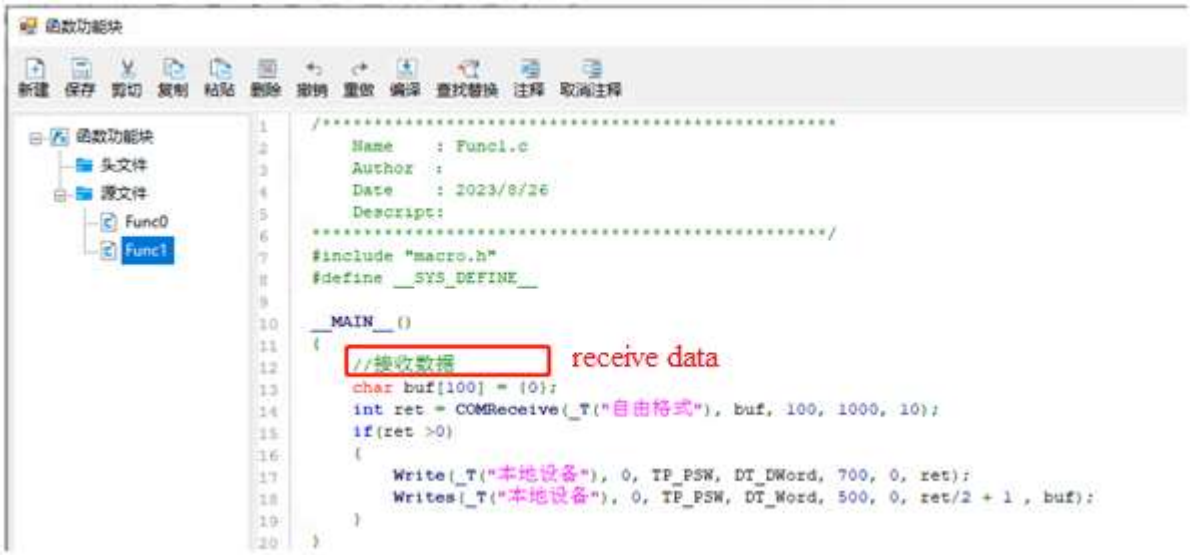



4. Create a new project and add components within the touch screen. As shown in the following figure:

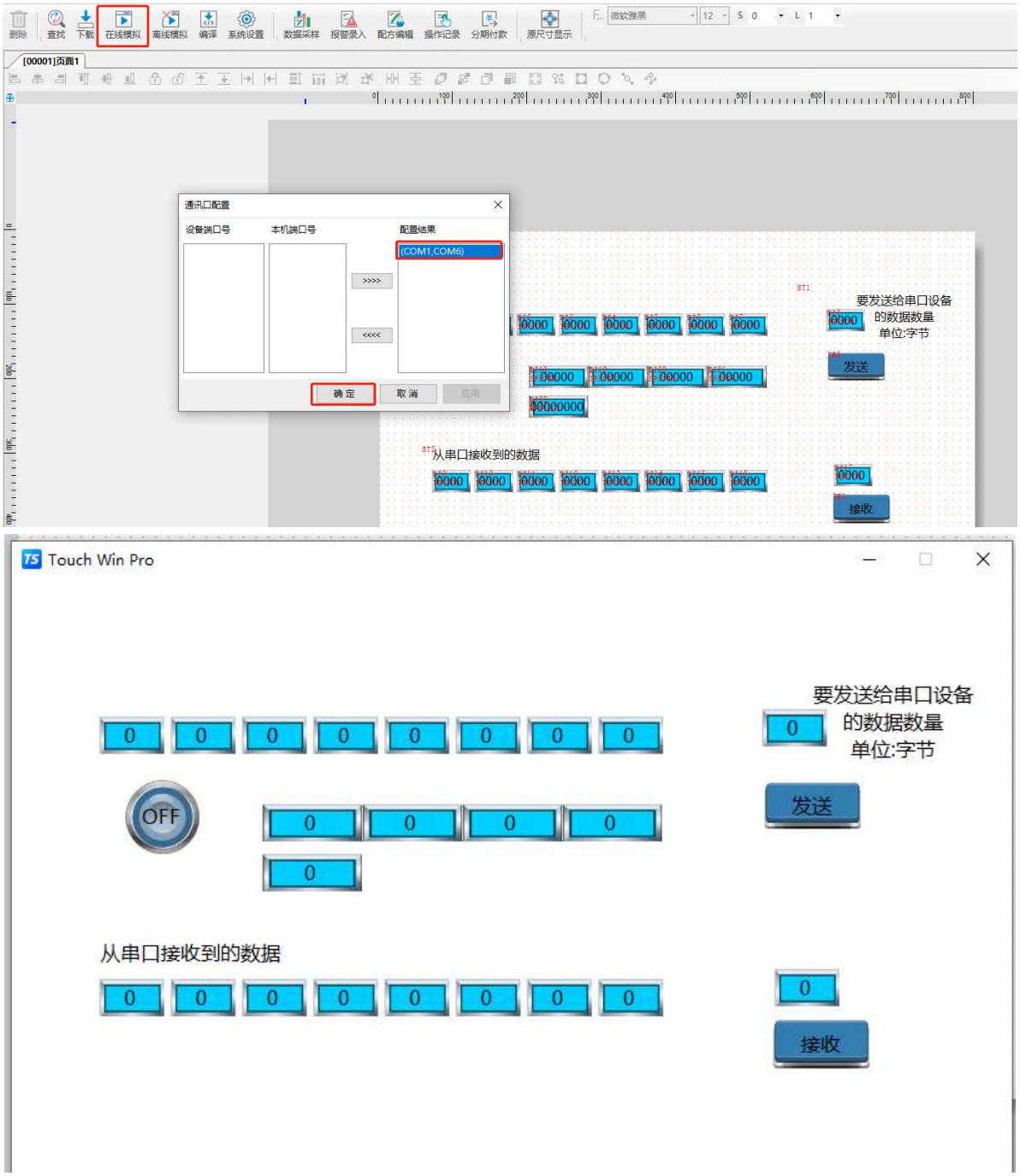



5. Establish a C function block to receive or send data. As shown in the following figure:




6. After editing the receive/send function, call the function and select online simulation to establish a connection with XS Studio. As shown in the following figure:

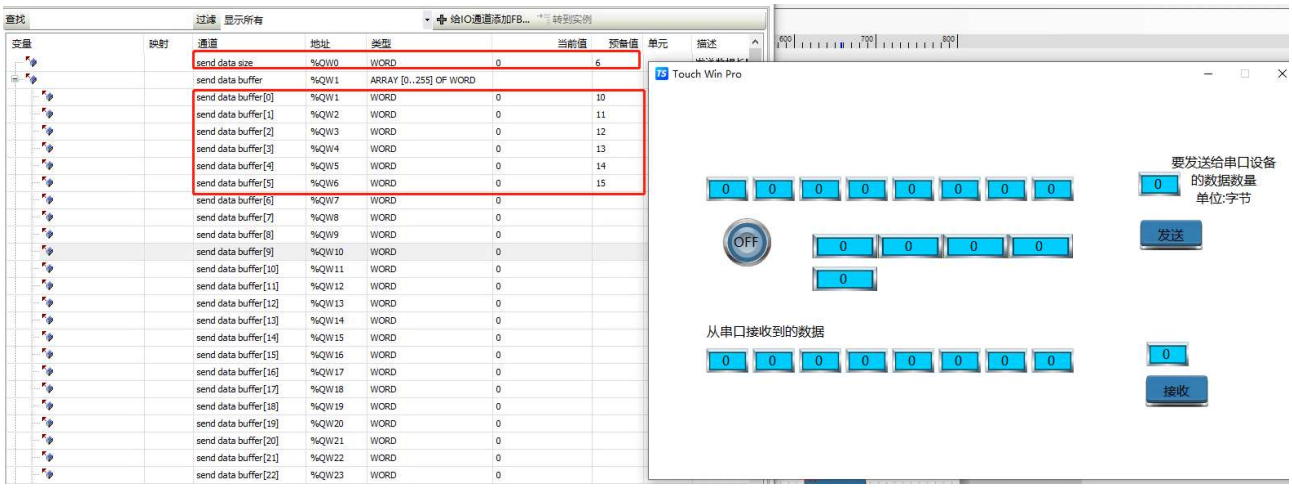



Note: As shown in the above figure, the COM1 port in the configuration result module is the actual serial port connected to the PLC device. COM6 refers to the virtual USB serial port on an actual PC.

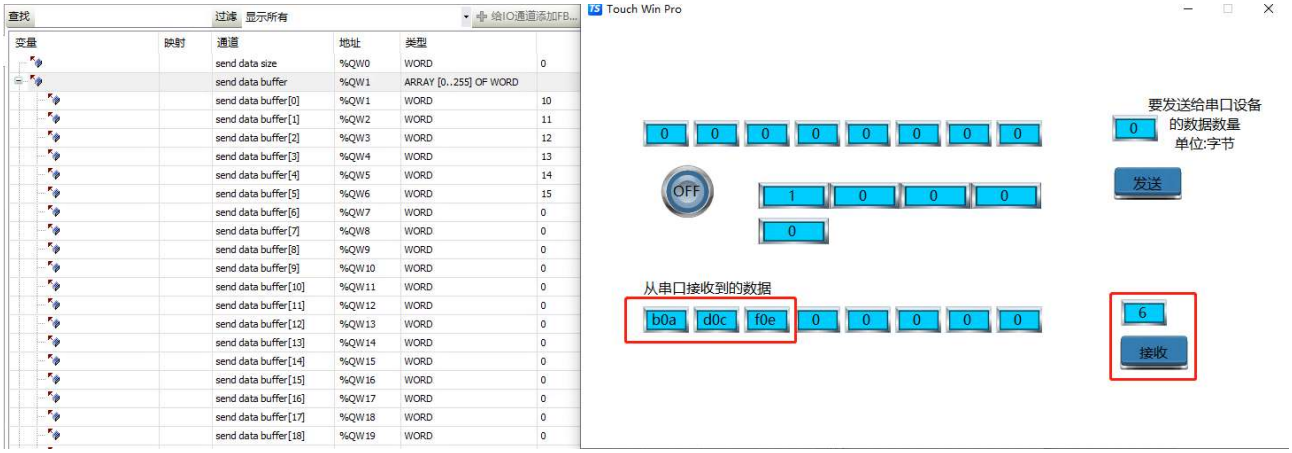
7. XS slave station interacts with touch screen master station for data exchange. As shown in the following figure:

■ XS Studio send data to touch screnn

Enter a custom preset value in the "Send Data Cache" channel and set "% QW0 Send Data Length"; If not set, data cannot be sent after writing. As shown in the following figure:



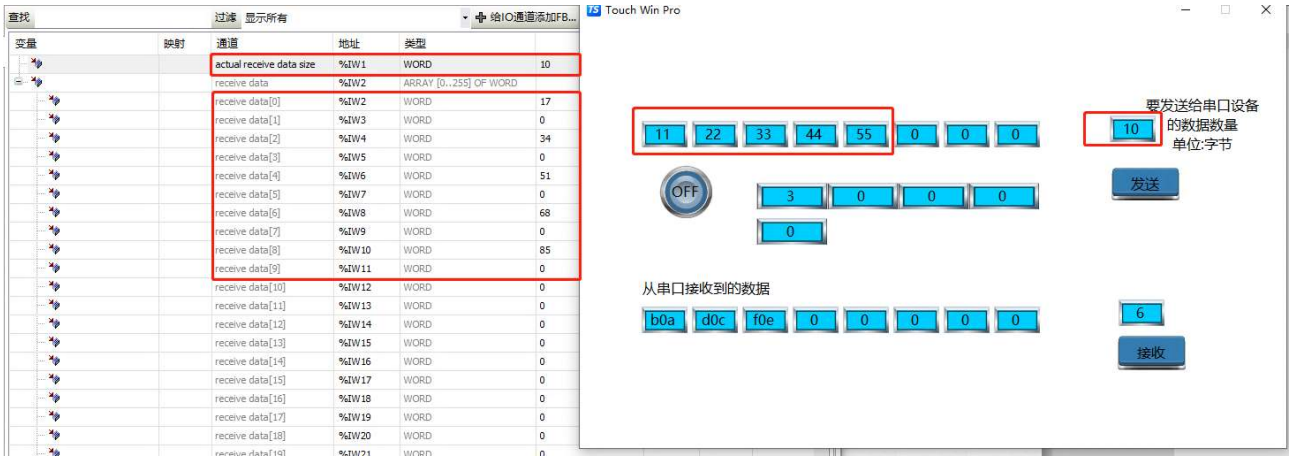

After the transmission is completed, the data within the length of "% QW0" will be automatically reset to zero. At this time, the touch screen receiving area can receive the corresponding data sent by the PLC. As shown in the following figure:




After XS Studio sends data, click the receive button on the touch screen. At this time, the length of the received data will be displayed above the button, and the left component of the button will display the received data. The hexadecimal high and low bytes will be displayed here.

Note: If the length of data written by the PLC device is greater than the set maximum sending length of 10 bytes, the data will be sent according to the set data length of 10.

- Touch screen send data to XS Studio




At this point, the actual received data length is % IW1=10, and the received data is cached in % IW2-% IW11.

Note:

(1) When the master station sends data to the PLC again, it will overwrite the original data and continue to receive cache from the first address "% IW2".

(2) When the length of data sent by the main station exceeds the set maximum receive length of 256 bytes, it is written at 256.

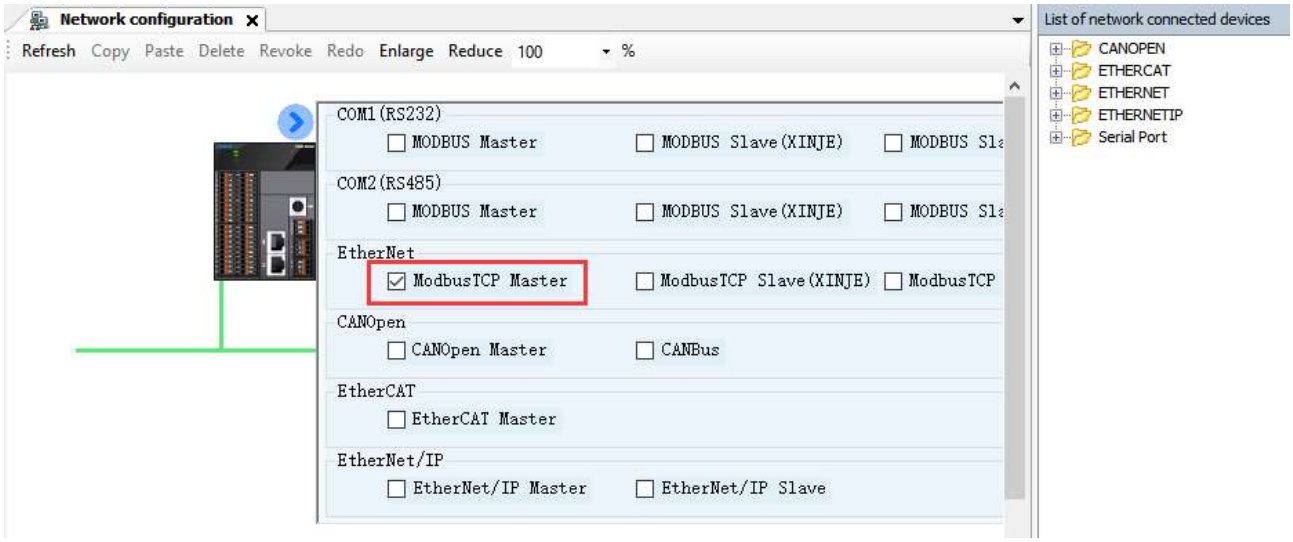
## 3-4. ModbusTCP communication

Modbus TCP uses TCP/IP to transmit Modbus messages between sites. Modbus TCP combines TCP/IP protocol with Modbus protocol as the application protocol standard for data representation. Modbus TCP communication packets are encapsulated in Ethernet TCP/IP packets. Unlike traditional serial port methods, Modbus TCP inserts a standard MODBUS packet into the TCP packet without data checksum addresses.
The XS series programmable controller body supports Modbus TCP protocol communication in both master and slave forms.
Main station form: When the programmable controller serves as the main station device, it can communicate with other slave devices using the Modbus TCP protocol. A master station can connect up to 64 slave stations.
Slave form: When a programmable controller is used as a slave device, it can only respond to the requirements of other master stations.

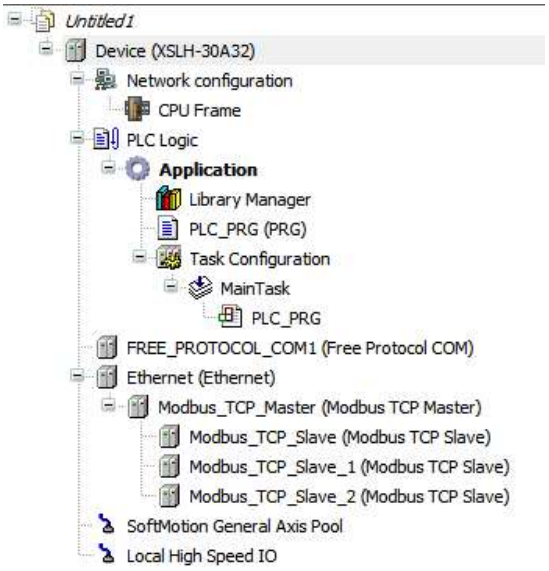
### 3-4-1. MODBUS TCP master station configuration

1. Enable master station, add slave station

Clicking on the PLC device in the network configuration will display the enabling window for the master/slave stations supported within the PLC. As shown in the following figure: Click the checkbox button in the window to enable the master/slave functions supported by the CPU, and then click "MODBUS-TCP" from the "Network Connection Device List" on the right side of the view to add the slave to the network.

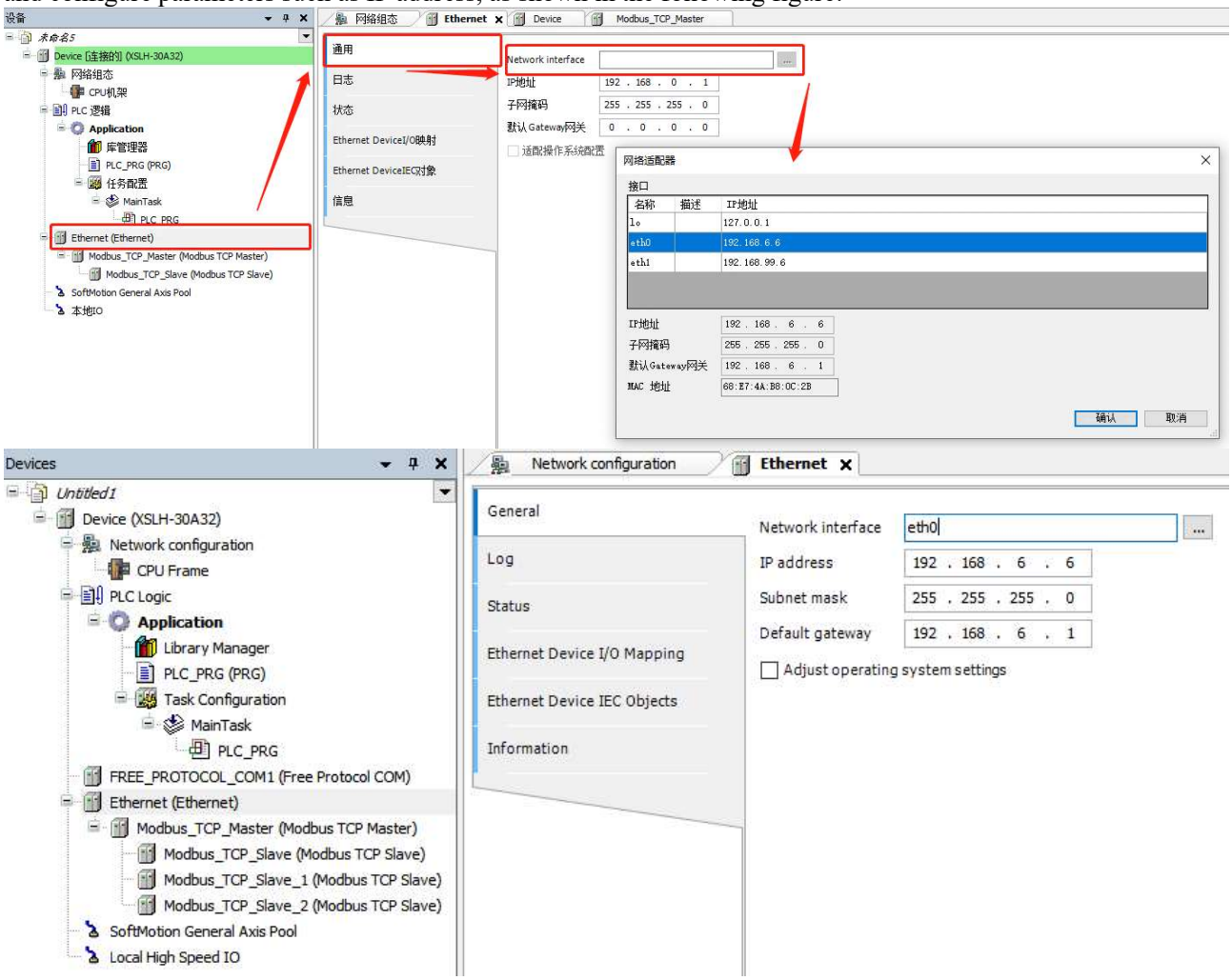



At this point, the ModbusTCP configuration corresponding device tree will appear in the left side view of the interface, as shown in the following figure:

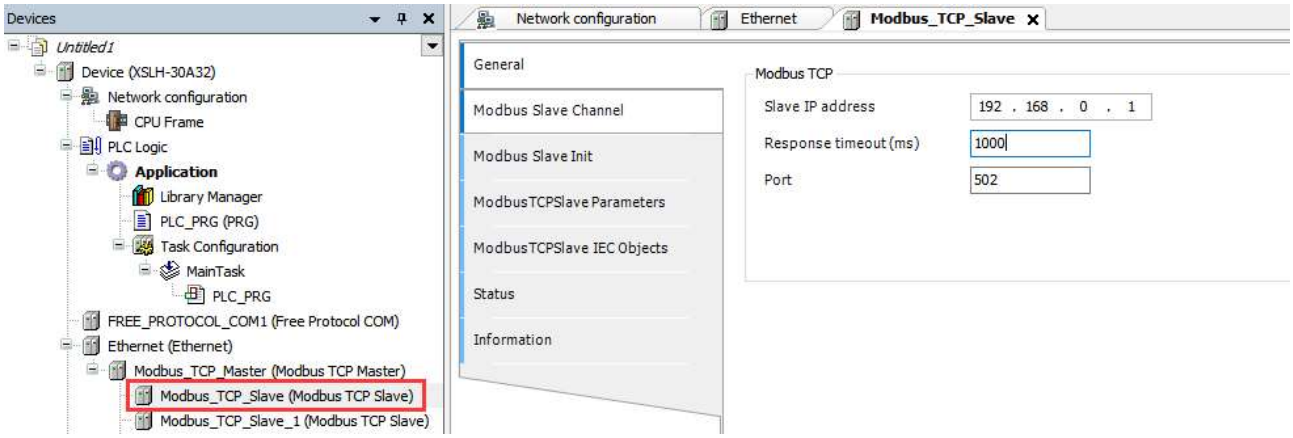



2. Master station communication configuration

When using PLC as a Modbus TCP master station, double-click "Ethernet" in the device tree to open the Ethernet master station configuration window and configure it. Click "General" to select the master station network port and configure parameters such as IP address, as shown in the following figure:

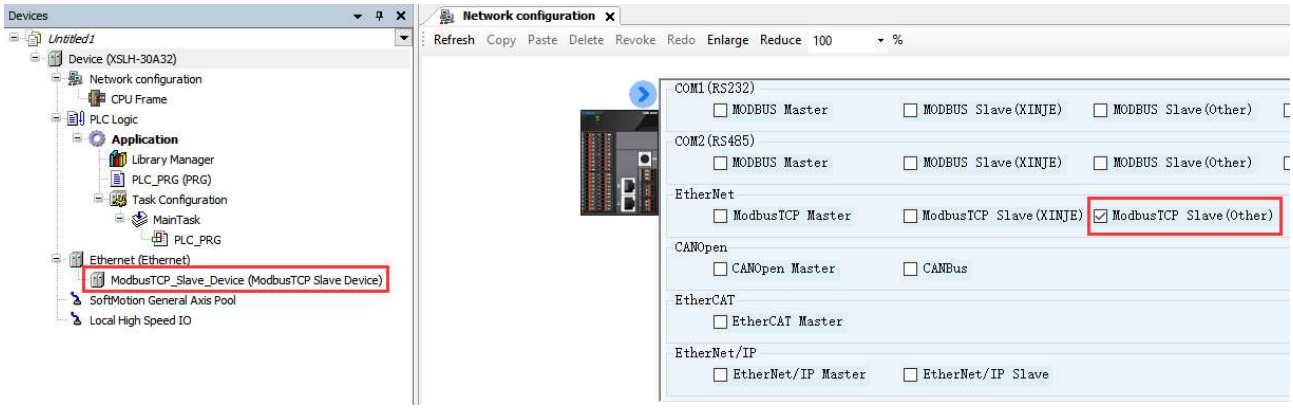



Double click on MODBUS_TCP_Slave(MODBUS TCP Slave) node opens the configuration interface to configure slave information, as shown in the figure:

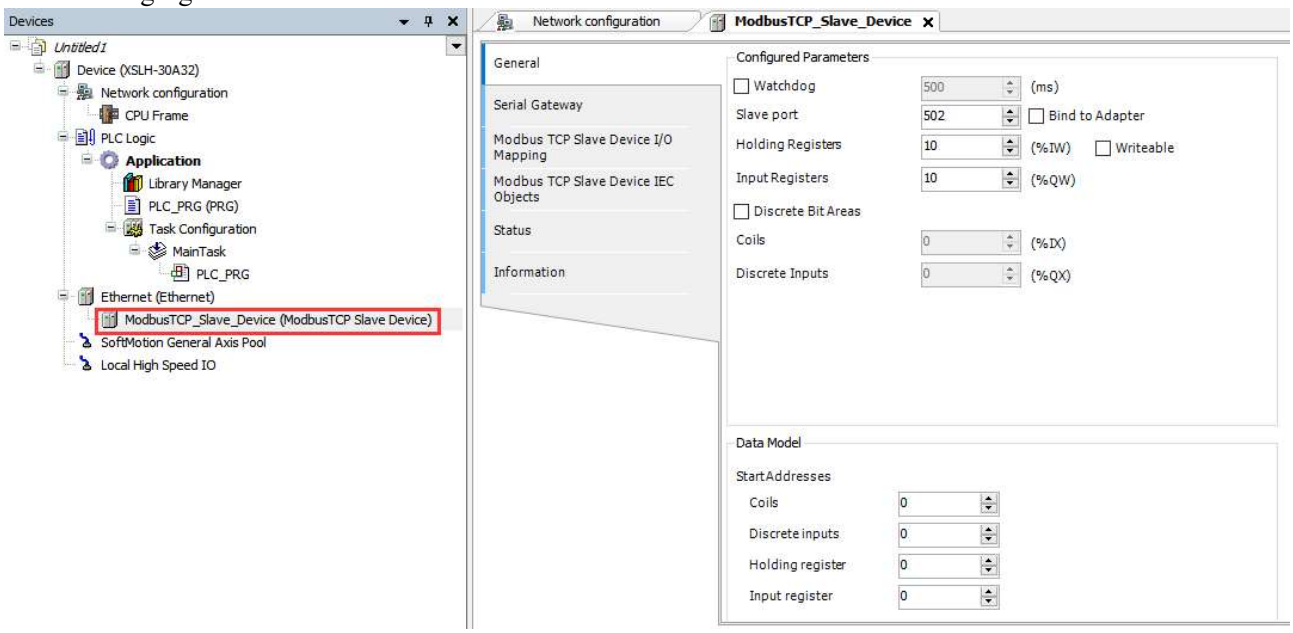



## 3-4-2. MODBUS TCP slave station configuration

Slave devices can be enabled through the enable window in the network configuration interface. The left view will generate corresponding slave device nodes, as shown in the following figure:

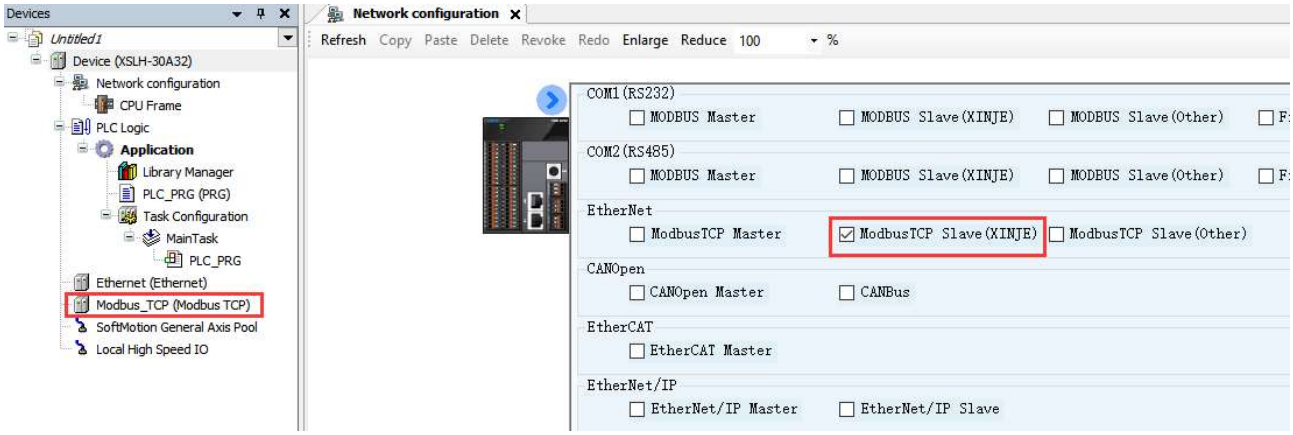



After adding a slave device, double-click on the " Modbus_TCP_Slave " node in the device tree node to open the configuration interface, which allows you to switch to the Modbus TCP slave configuration interface. As shown in the following figure:

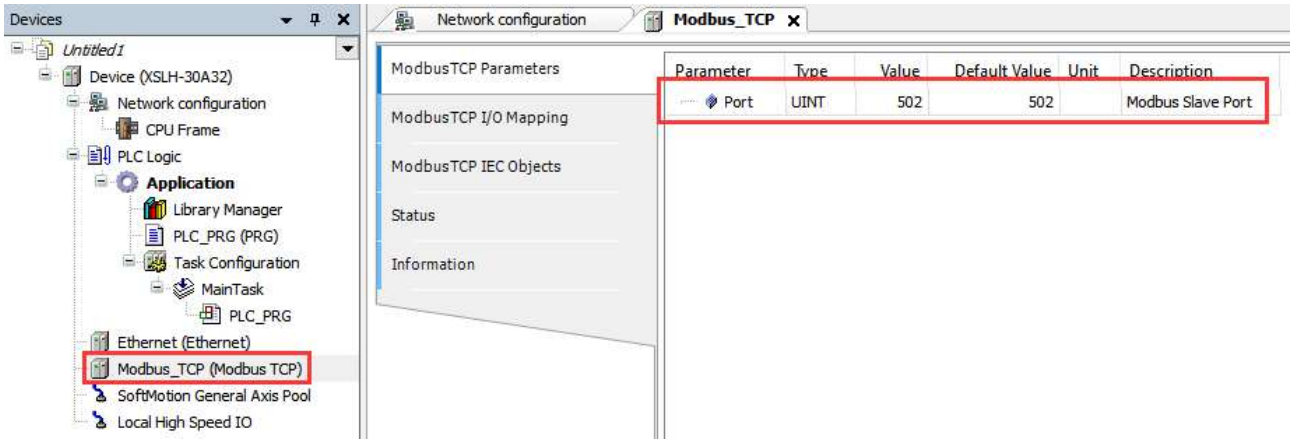


## 3-4-3. MODBUS TCP (XINJE) slave configuration

1. Double click on the "Network Configuration" node from the left device tree to open the network configuration interface. Enable the ModbusTCP slave (XINJE) device through the enable window, and a "Modbus TCP" node will be generated in the left device tree. As shown in the following figure:




2. Double click on the "Modbus TCP" node in the left device tree to open the Modbus TCP parameter configuration interface. ModbusTCP ports can be set according to actual needs. Here, according to actual needs, it is set to 8888, as shown in the following figure:




**Note:**

(1) Allow users to configure port numbers, with a range of 1-65535 and a default port number of 502.

(2) When using a PLC as a Modbus TCP (XINJE) slave device, the address range that can be accessed by the master device is defined as follows:

All coil operations (function codes 0x01, 0x02, 0x05, 0x0F) have read-write addresses of %MB0-%MB65534;

All register operations (function codes 0x03, 0x04, 0x06, 0x10) have read-write addresses of %MW40000-%MW105534.

(3) The power-off retention properties supported by different firmware versions and different Modbus TCP (XINJE) library versions are as follows:

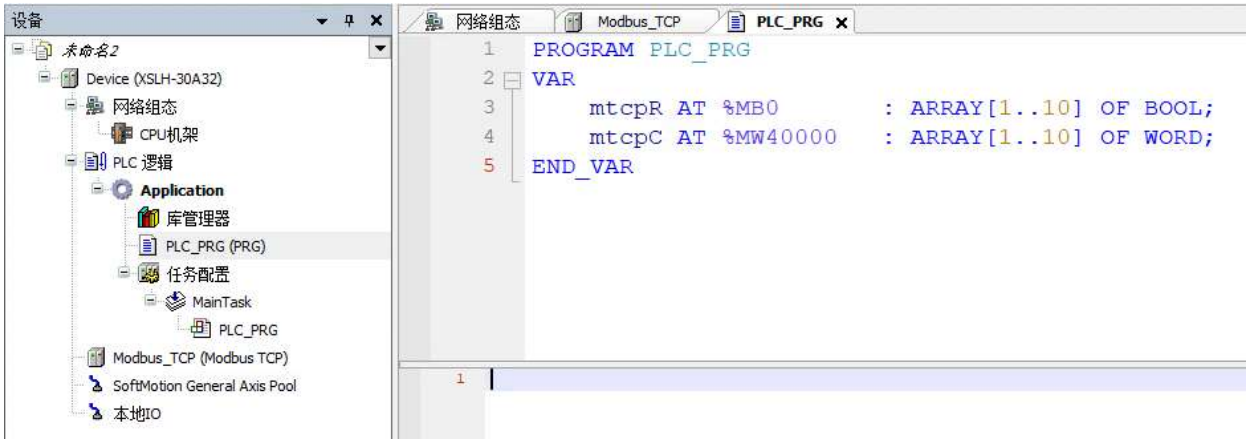
| Modbus TCP (XINJE) Library version / Firmware version | V1.0.0.0 | V2.0.0.0 | V3.0.0.0 |
|---|---|---|---|
| V1.0.2a | M area fixed power-off maintenance | M area fixed power-off maintenance | M area fixed power-off not maintenance |
| V1.1.0 | M area fixed power-off maintenance | M area fixed power-off not maintenance | M area fixed power-off not maintenance |

| | | | |
|---|---|---|---|
| V2.2.0 | M area fixed power-off maintenance | M-zone can be equipped with power-off maintenance | M area fixed power-off maintenance |

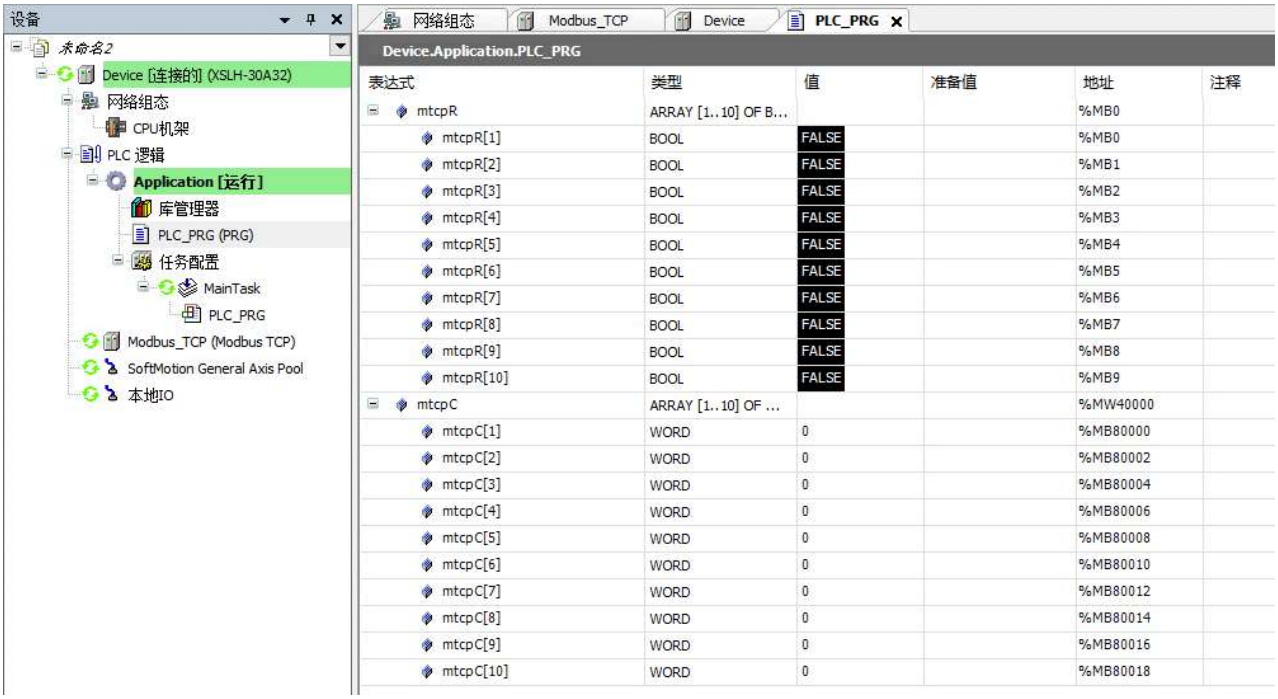
3. Application example

Example 1: Here, XS Studio software serves as a slave station and uses the third-party debugging tool Modbus Poll as the master station to establish a connection and perform Modbus TCP communication, achieving the reception or transmission of register or coil data.

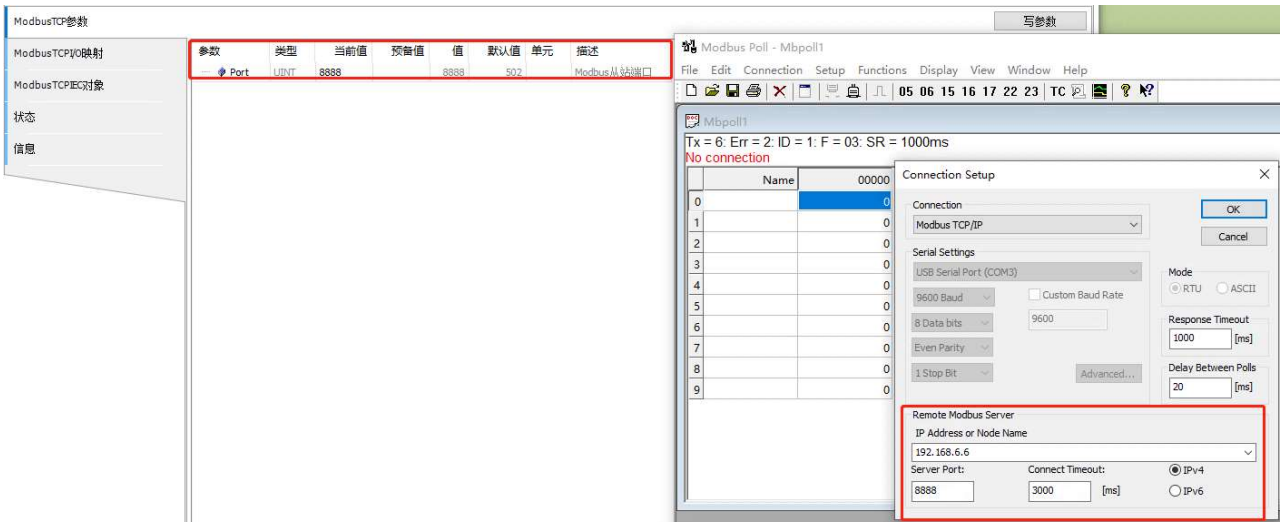
(1) Declare two variables in the "PLC-PRG" editor to receive and send register or coil data, respectively. As shown in the following figure:




(2) Establish a connection with the XSLH-30A32 device and log in to run it. As shown in the following figure:

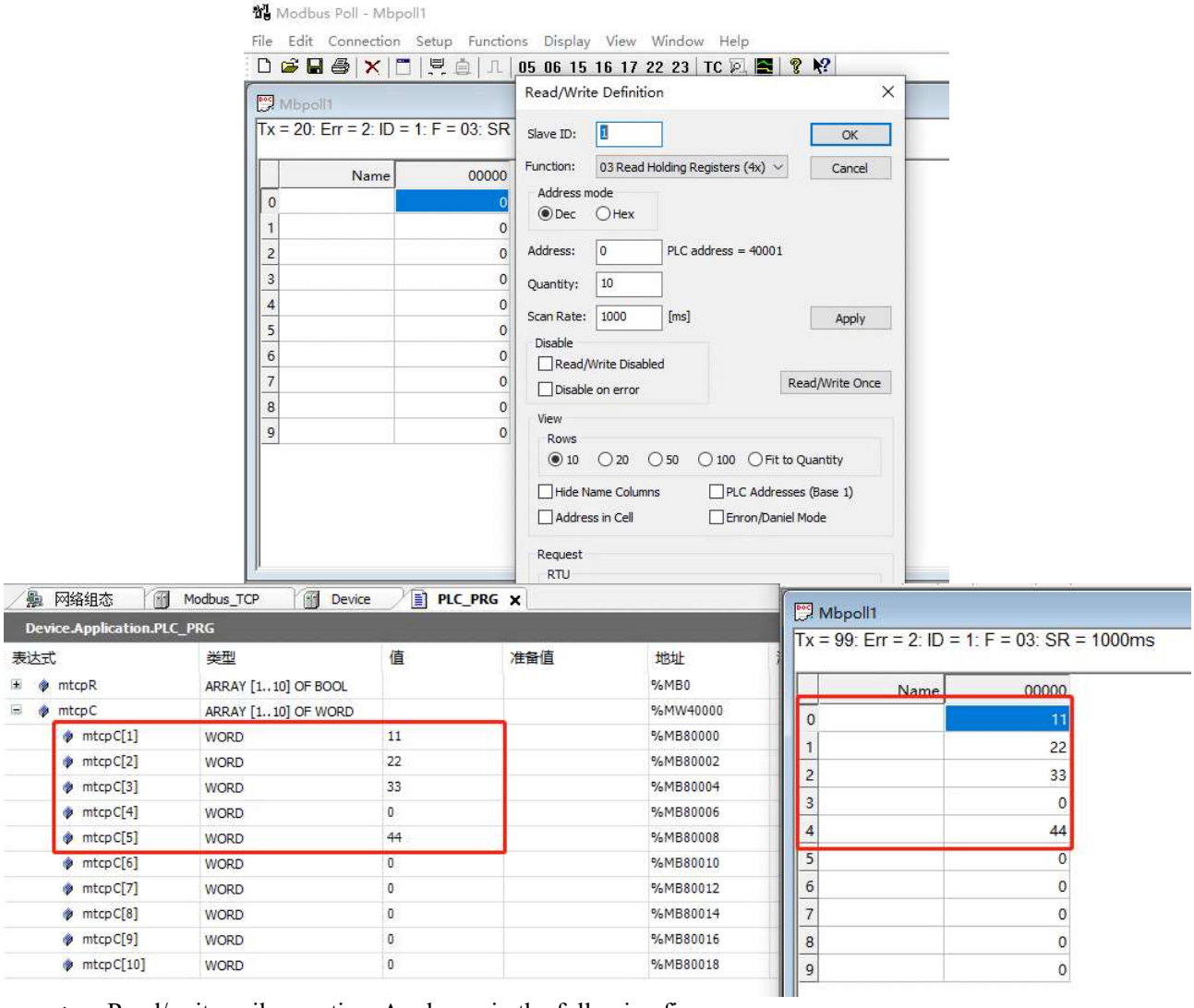



(3) Configure the relevant parameters of Modbus Poll to be consistent with the IP of the slave device and the port number in the software, otherwise the connection cannot be successfully established. As shown in the following figure:

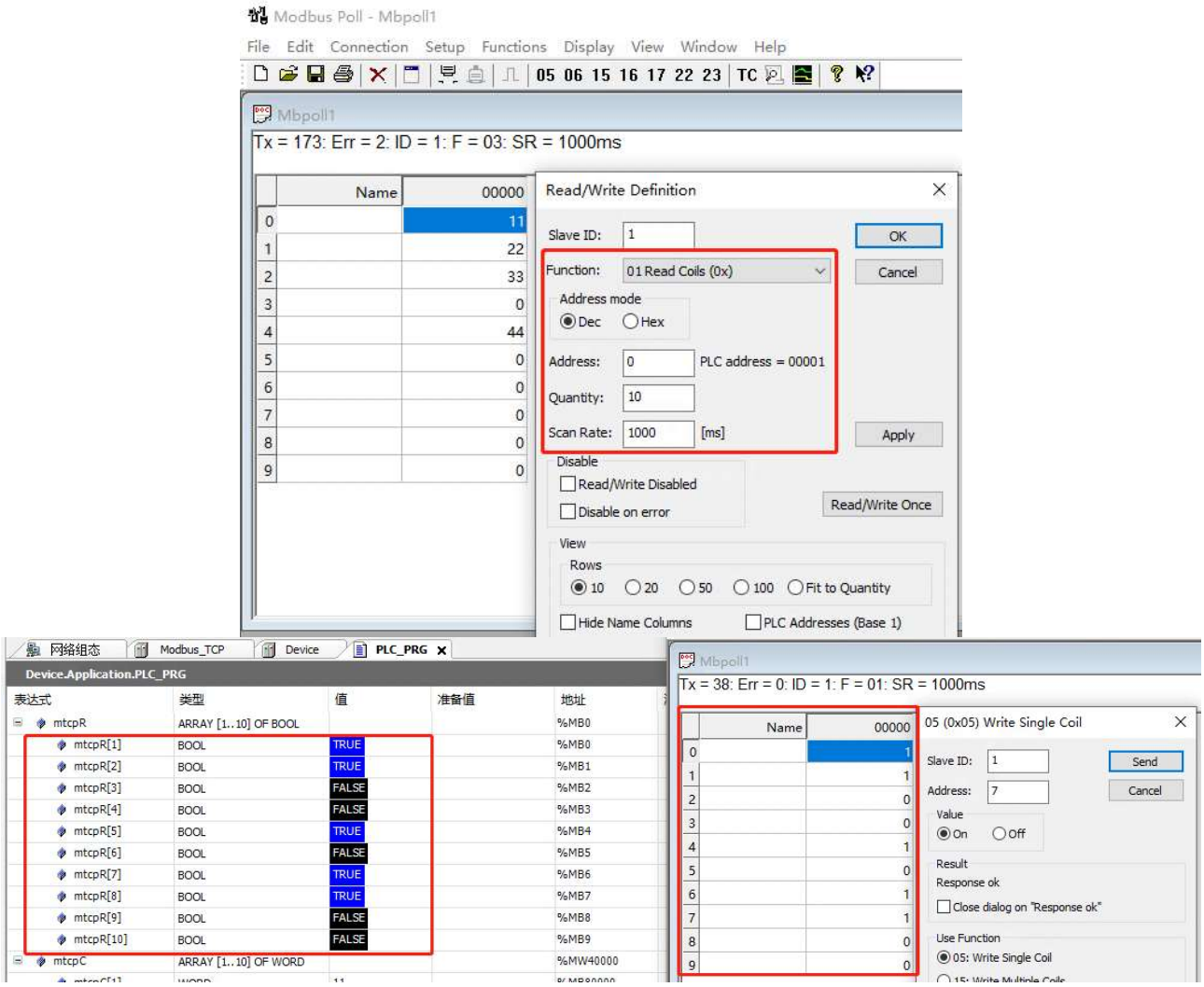



(4) Configure the relevant read and write parameters of the master station equipment to perform related read/write register or coil operations with the slave station:

- Read/write register operations. As shown in the following figure:



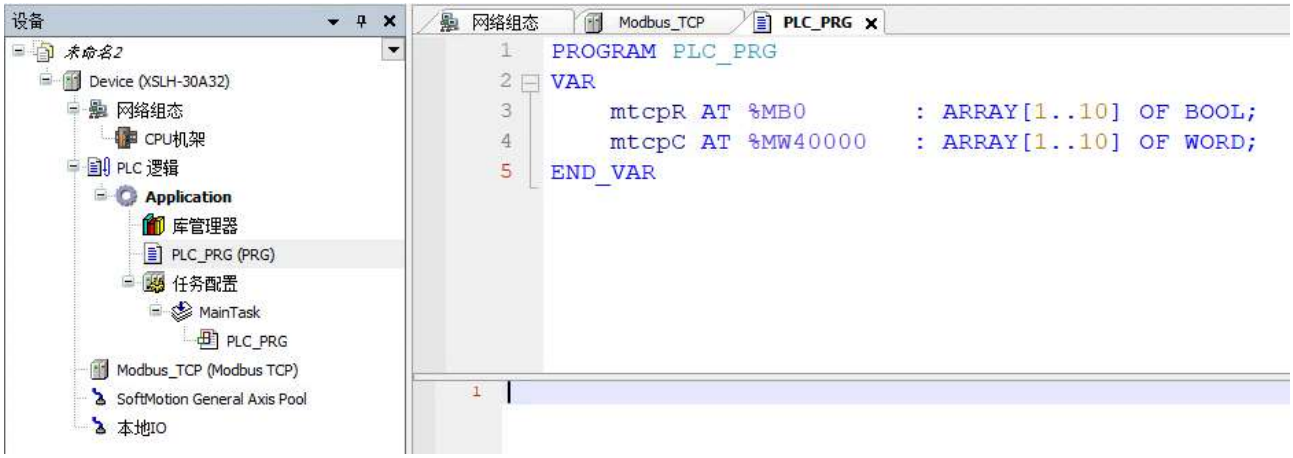

- Read/write coil operation. As shown in the following figure:



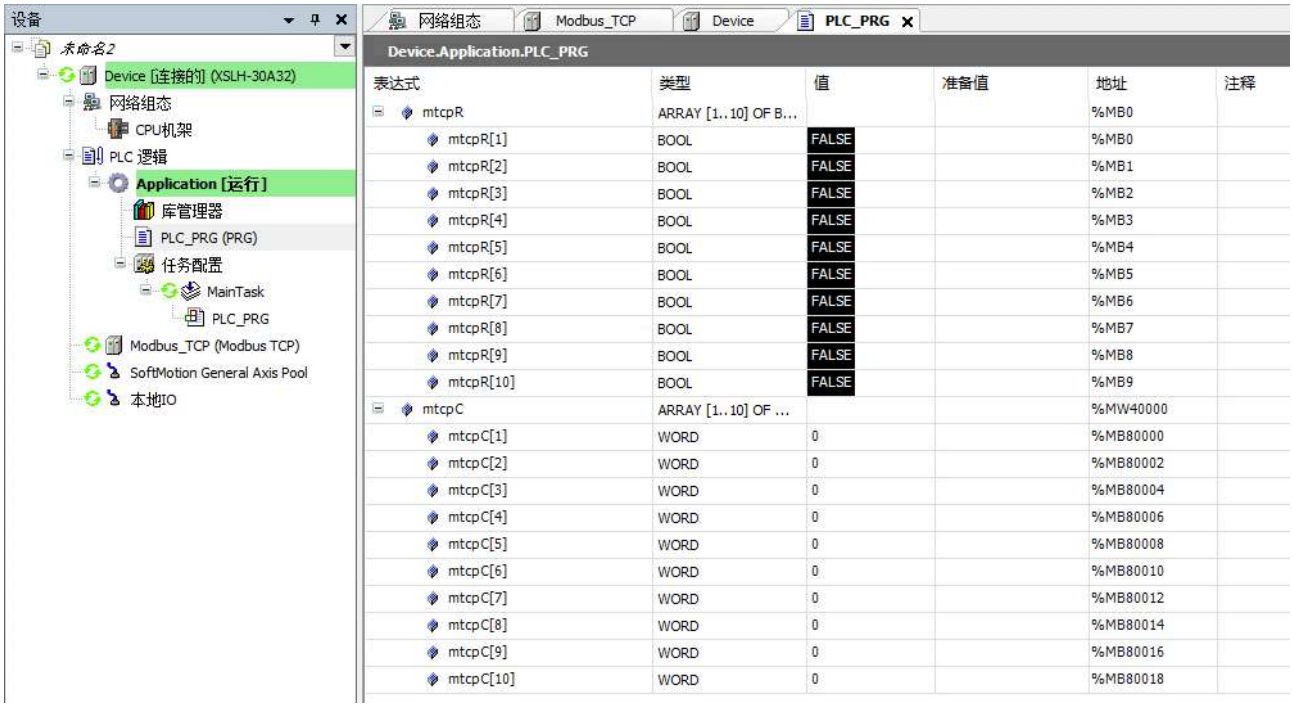

At this point, the master and slave stations have successfully communicated.

Example 2: Here, XS Studio software serves as a slave station and the touch screen serves as the master station, establishing a connection with the touch screen and conducting Modbus TCP communication to achieve data exchange.

(1) Declare two variables in the "PLC-PRG" editor to receive and send register or coil data, respectively. As shown in the following figure:



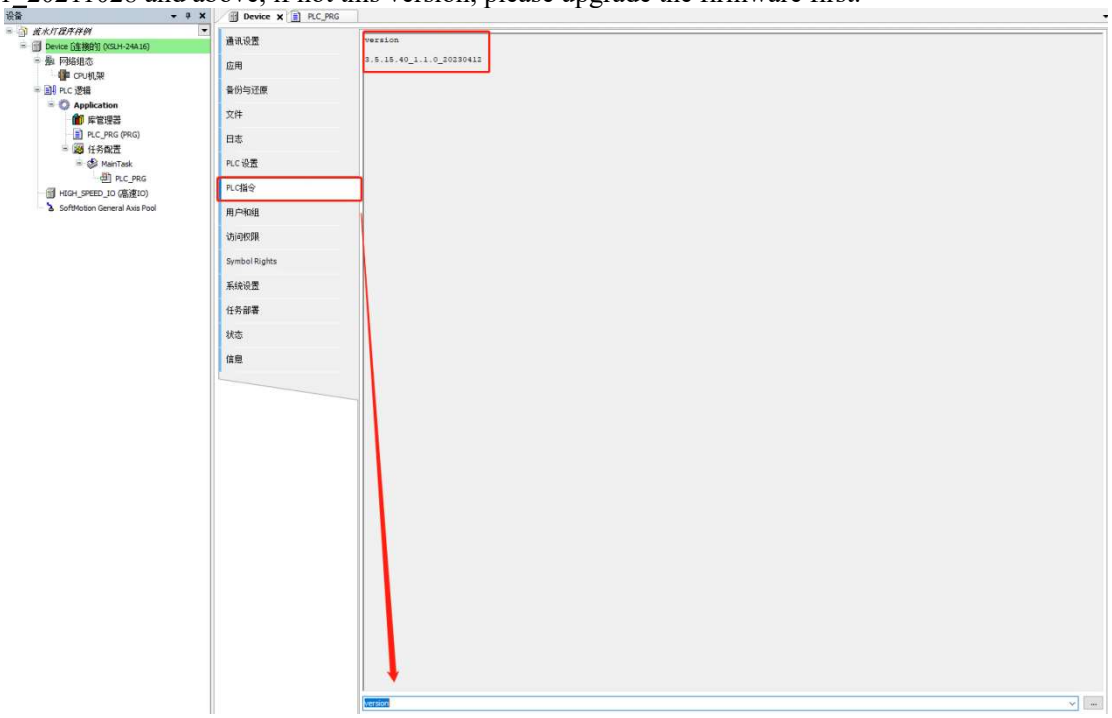

(2) Establish a connection with the XSLH-30A32 device and log in to run it. As shown in the following figure:




| 表达式 | 类型 | 值 | 准备值 | 地址 | 注释 |
|---|---|---|---|---|---|
| mtcpR | ARRAY [1..10] OF B... | | | %MB0 | |
| mtcpR[1] | BOOL | FALSE | | %MB0 | |
| mtcpR[2] | BOOL | FALSE | | %MB1 | |
| mtcpR[3] | BOOL | FALSE | | %MB2 | |
| mtcpR[4] | BOOL | FALSE | | %MB3 | |
| mtcpR[5] | BOOL | FALSE | | %MB4 | |
| mtcpR[6] | BOOL | FALSE | | %MB5 | |
| mtcpR[7] | BOOL | FALSE | | %MB6 | |
| mtcpR[8] | BOOL | FALSE | | %MB7 | |
| mtcpR[9] | BOOL | FALSE | | %MB8 | |
| mtcpR[10] | BOOL | FALSE | | %MB9 | |
| mtcpC | ARRAY [1..10] OF ... | | | %MW40000 | |
| mtcpC[1] | WORD | 0 | | %MB80000 | |
| mtcpC[2] | WORD | 0 | | %MB80002 | |
| mtcpC[3] | WORD | 0 | | %MB80004 | |
| mtcpC[4] | WORD | 0 | | %MB80006 | |
| mtcpC[5] | WORD | 0 | | %MB80008 | |
| mtcpC[6] | WORD | 0 | | %MB80010 | |
| mtcpC[7] | WORD | 0 | | %MB80012 | |
| mtcpC[8] | WORD | 0 | | %MB80014 | |
| mtcpC[9] | WORD | 0 | | %MB80016 | |
| mtcpC[10] | WORD | 0 | | %MB80018 | |

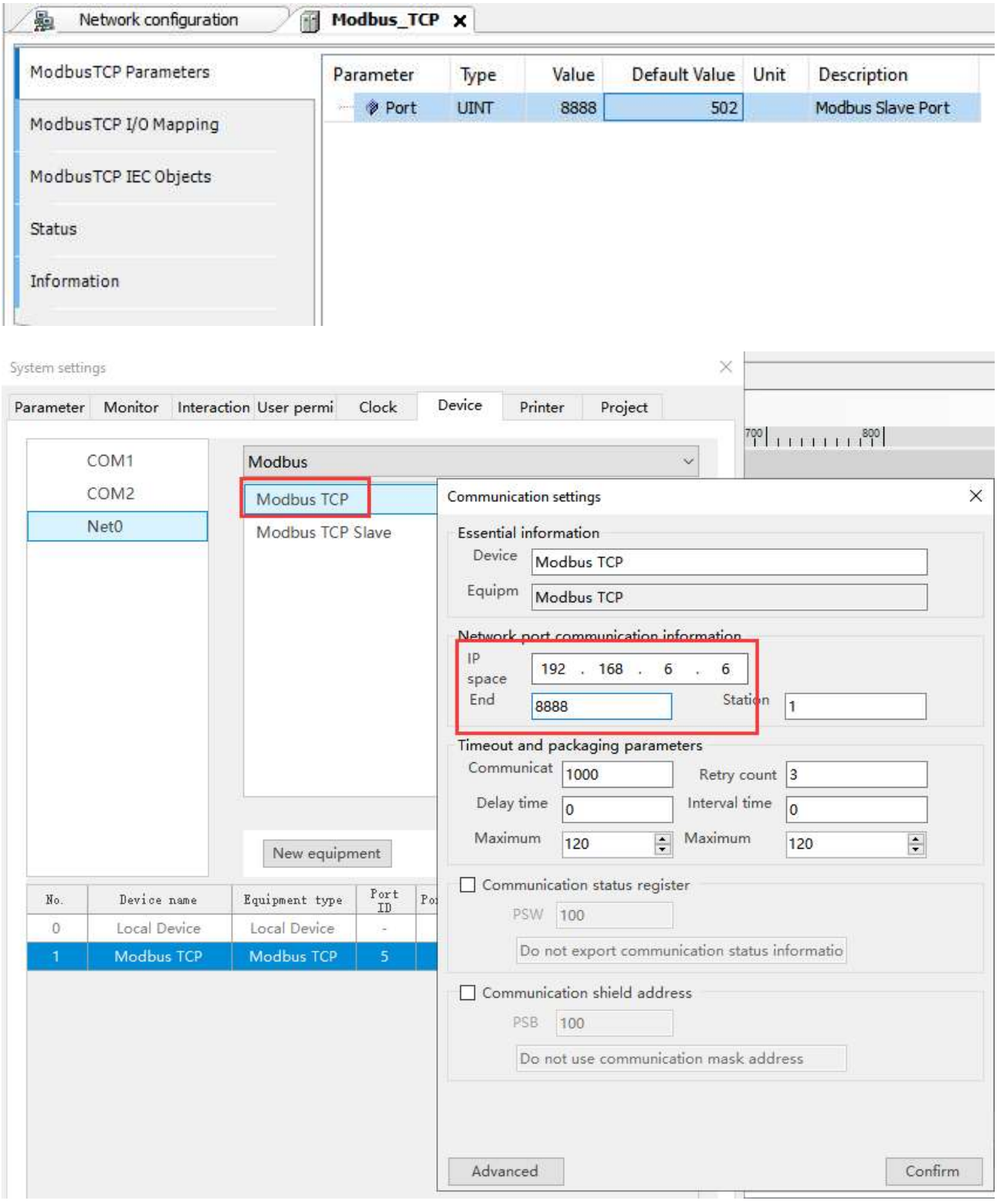
(3) Configure touch screen related parameters.

When using this communication function, please first check if the firmware version of the PLC is 3.5.15.40_1.0.0_P1_20211028 and above, if not this version, please upgrade the firmware first.

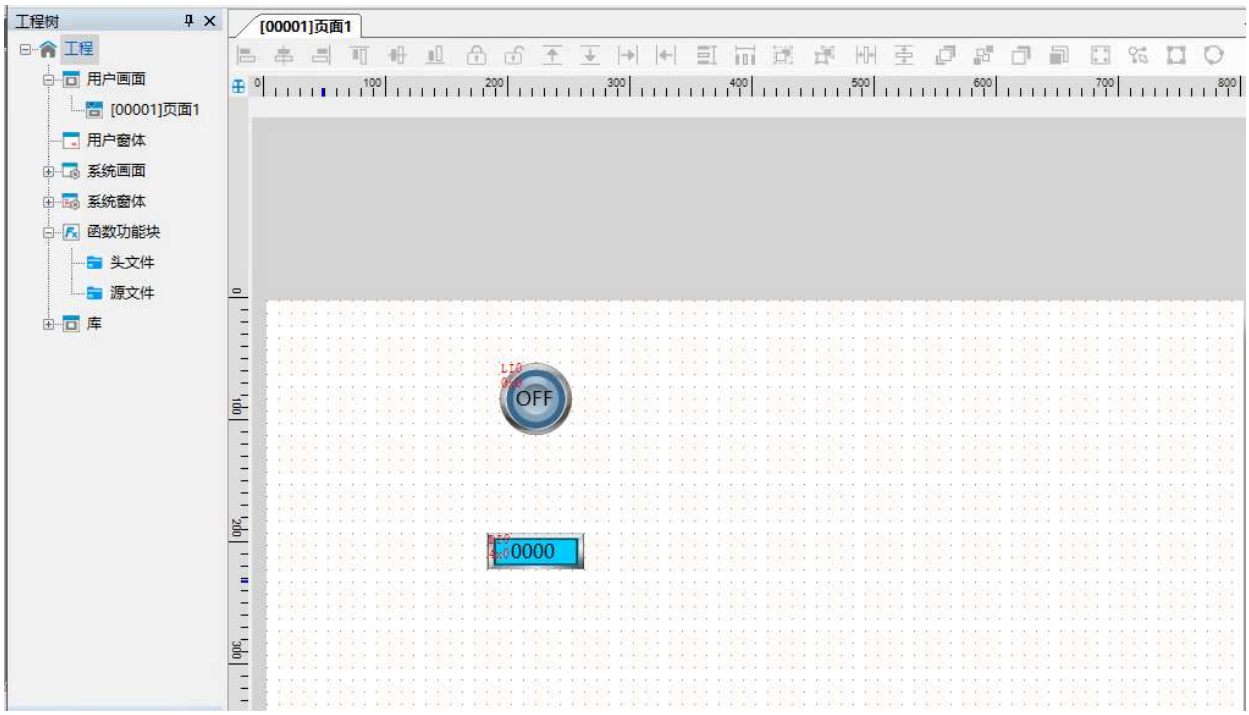



(4) Set communication parameters for the touch screen.

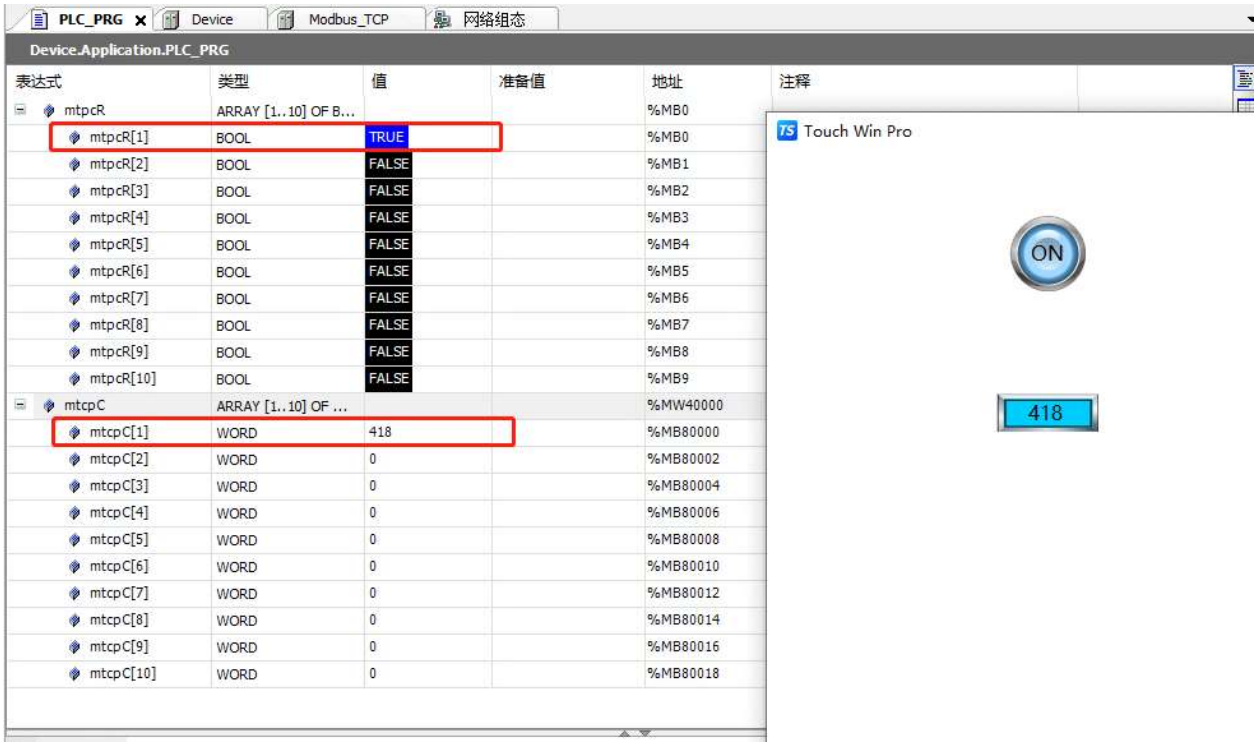
Open the "Device" function interface from the "System Settings" window and select Net0->Modbus TCP (Display Master). The IP address here needs to be consistent with the IP of the slave device and the port number in the software, otherwise the connection cannot be successfully established. As shown in the following figure:




(5) Add relevant components in the project interface, as shown in the following figure:




(6) Online simulation, establish a connection with XS Studio software to perform related read/write register or coil operations with the slave station. As shown in the following figure:




**Note:**

(1) Add the required touch screen elements and select Modbus_general, station number must be set to 0!

(2) Select 0X (readable and writable) or 1X (read-only) for the object type of the button or indicator light, where both 0X0 and 1X0 correspond to MB0, and so on;

(3) Select 3X (read-only) or 4X (readable and writable) as the object type for data input or display. Both 3X0 and 4X0 correspond to %MW40000, and so on;

(4) If the data type input or display is DWORD, then 3X0 and 4X0 occupy the %MW40000, %MW40001 registers, and so on.

## 3-4-4. MODBUS TCP common faults

1. The master station is unable to read and write to the Xinje XS controller as a slave station
Processing: When configuring slave station parameters on the master station side, the station number needs to be filled in as 0.
2. The Xinje XS controller, as the master station, cannot communicate with the slave station
Handling:
(1) To access this address for the client, the first step is to ensure that the server has this address, otherwise the client will not be able to access non-existent addresses and an error will be reported;
(2) Check if the slave station has data types, initial addresses, and communication numbers of these accesses;
(3) Please pay attention to the function code. The function code does not match and communication is not possible.

## 3-4-5. MODBUS TCP communication frame

Modbus devices can be divided into a main station (poll) and a slave station (slave). There is only one master station and multiple slave stations. The master station sends request frames to each slave station and the slave station responds. When using TCP communication, the master station is the client side and actively establishes a connection; Slave station is the server side, waiting for connection.

- Main station request: function code+data;
- Normal response of the slave station: request function code+response data;
- Abnormal response of the slave station: abnormal function code+abnormal code, where the abnormal function code is about to request the highest effective position 1 of the function code, and the abnormal code indicates the type of error;
- Attention: A timeout management mechanism is required to avoid waiting indefinitely for responses that may not occur.

IANA (Internet Assigned Numbers Authority) assigns the TCP port number 502 to the Modbus protocol, which is currently the only port number assigned in the instrumentation and automation industry.
The data frame of ModbusTCP can be divided into two parts: MBAP+PDU.

- Message header MBAP

MBAP is the header of the message, with a length of 7 bytes, and its composition is as follows:

| Transaction ID | Protocol identification | Length | Unit identifier |
|---|---|---|---|
| 2 bytes | 2 bytes | 2 bytes | 1 byte |

**Transaction identifier**: It can be understood as the sequence number of a message, and usually needs to be added with 1 after each communication to distinguish different communication data messages.
**Protocol identifier**: 00 00 represents the ModbusTCP protocol.
**Length**: represents the length of the following data, measured in bytes.
**Unit identifier**: can be understood as the device address.

- Frame structure PDU

The PDU consists of a function code and data. The function code is 1 byte, and the data length varies depending on the specific function.

**Communication process:**

1. Connect to establish a TCP connection;
2. Prepare Modbus messages;
3. Send a message using the send command;
4. Waiting for response under the same connection;
5. Use the recv command to read the message and complete a data exchange;
6. At the end of the communication task, close the TCP connection.

# 3-5. CANbus

CAN is the abbreviation for Controller Area Network (hereinafter referred to as CAN), which is an ISO internationally standardized serial communication protocol. In North America and Western Europe, the CAN bus protocol has become the standard bus for automotive computer control systems and embedded industrial control LANs, and has the J1939 protocol designed specifically for large trucks and heavy machinery vehicles based on CAN as the underlying protocol.

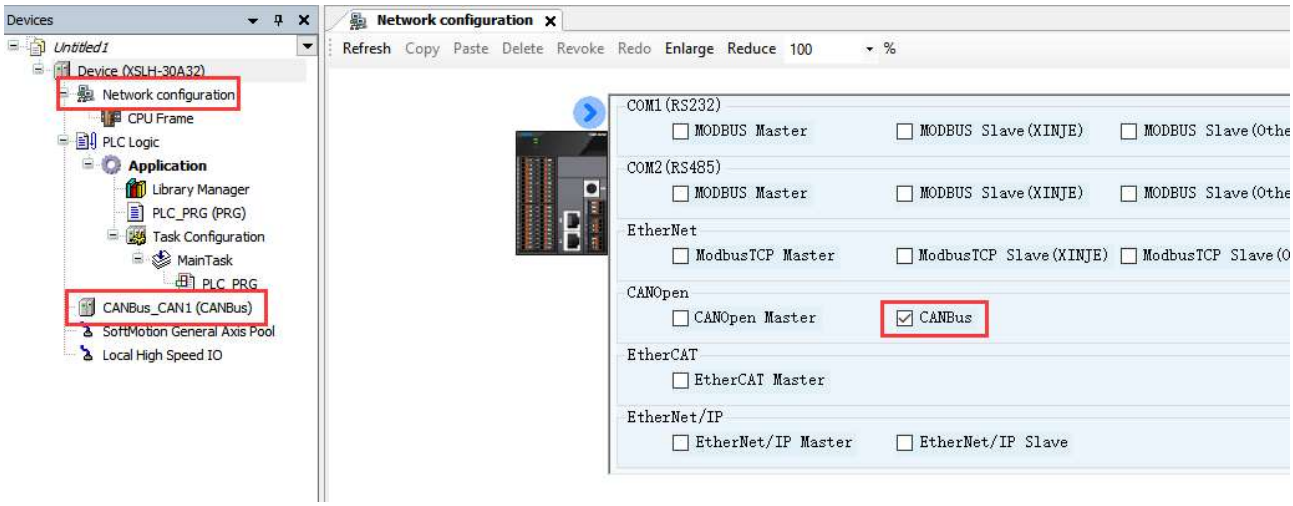
## 3-5-1. Parameter configuration

The abstract meaning of CANBus is a controller local area network. In fact, it is a twisted pair with high and low level differences. It plays a role in transmitting data, and is favored by engineers due to its real-time, reliable, and effective serial communication. Originally developed by Bosch in Germany for the application of automotive electronics, it has now been promoted to fields such as mechanical manufacturing, industrial automation, servo motor manufacturing, large-scale medical machinery, and building security monitoring. At present, CANBus has become the preferred fieldbus for industrial communication.

The PLC equipment of XSLH-30A32 model exchanges data in free format with other devices in the CAN network that support CAN2.0B or CAN2.0A protocol. (Currently, only XSLH-30A32 models support CANBus communication)

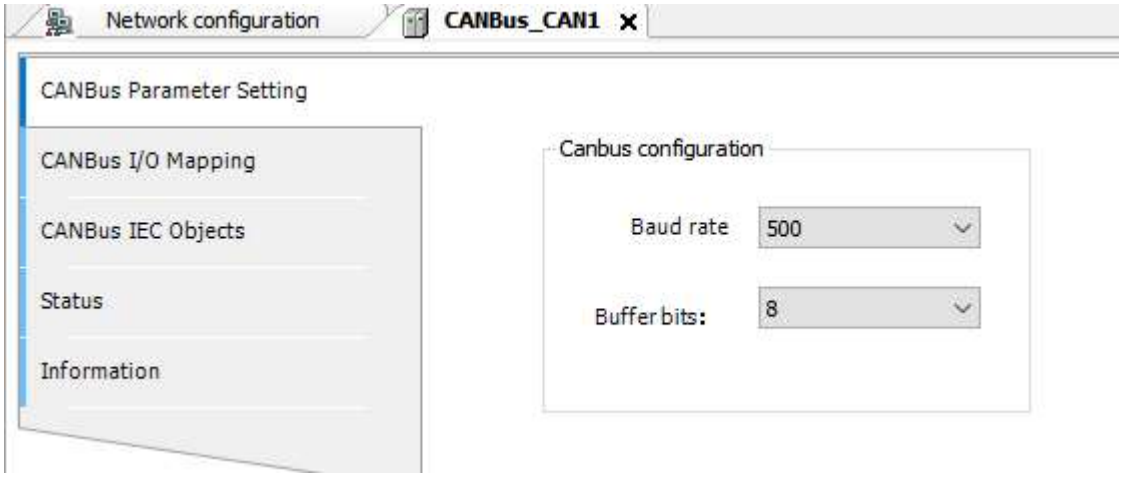
1. Enable CANBus devices

Double click on the "Network Configuration" node in the left device tree to open the network configuration interface. Enable "CANBus" through the enable window and generate the "CANBus_CAN1 (CANBus)" node on the left device. As shown in the following figure.




2. Set the CANBus parameters

Double click on the "CANBus_CAN1 (CANBus)" node in the left device tree to open the CANBus free protocol configuration interface. The baud rate and cache bits of CANBus can be set according to actual needs. As shown in the following figure.

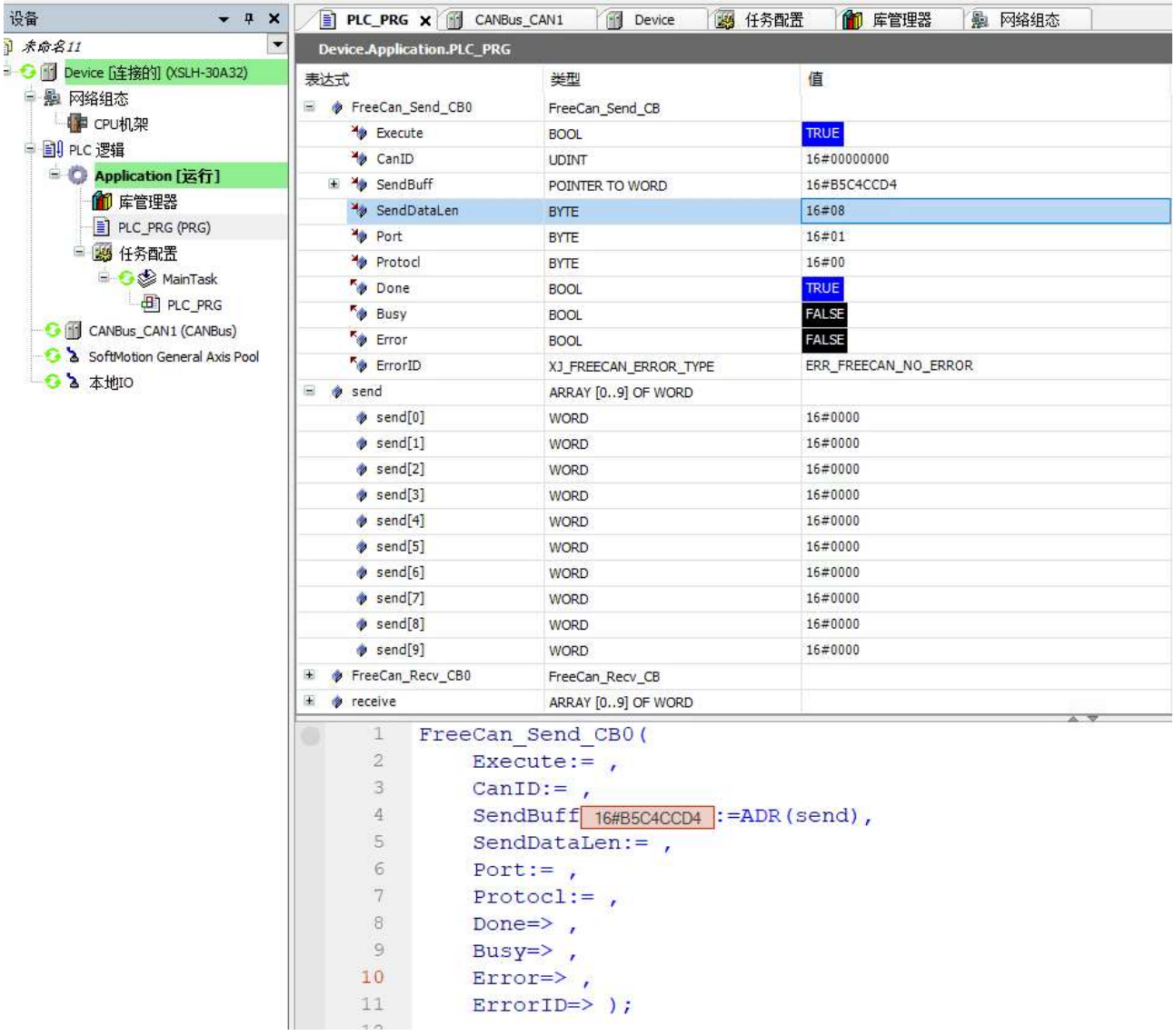



The CANbus parameter configuration is as follows:

Baud rate: The rate at which communication occurs.

Buffer bit: The cache bit can be set to 8 or 16 bits. When the cache bit is 8 bits, only the low byte data of the register is sent; When the cache bit is 16 bits, both high and low byte data of the register will be sent, with low bytes first and high bytes last.

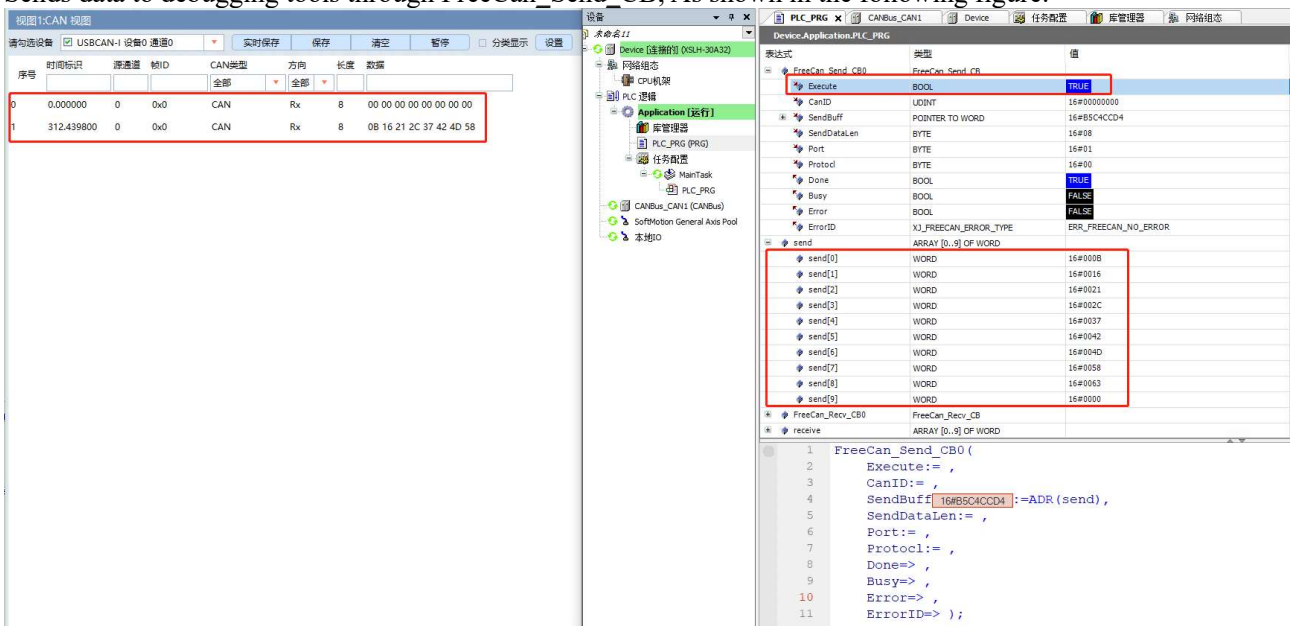
3. Write CANBus free format communication instructions and configure relevant parameters to establish a connection with PLC equipment. As shown in the following figure:



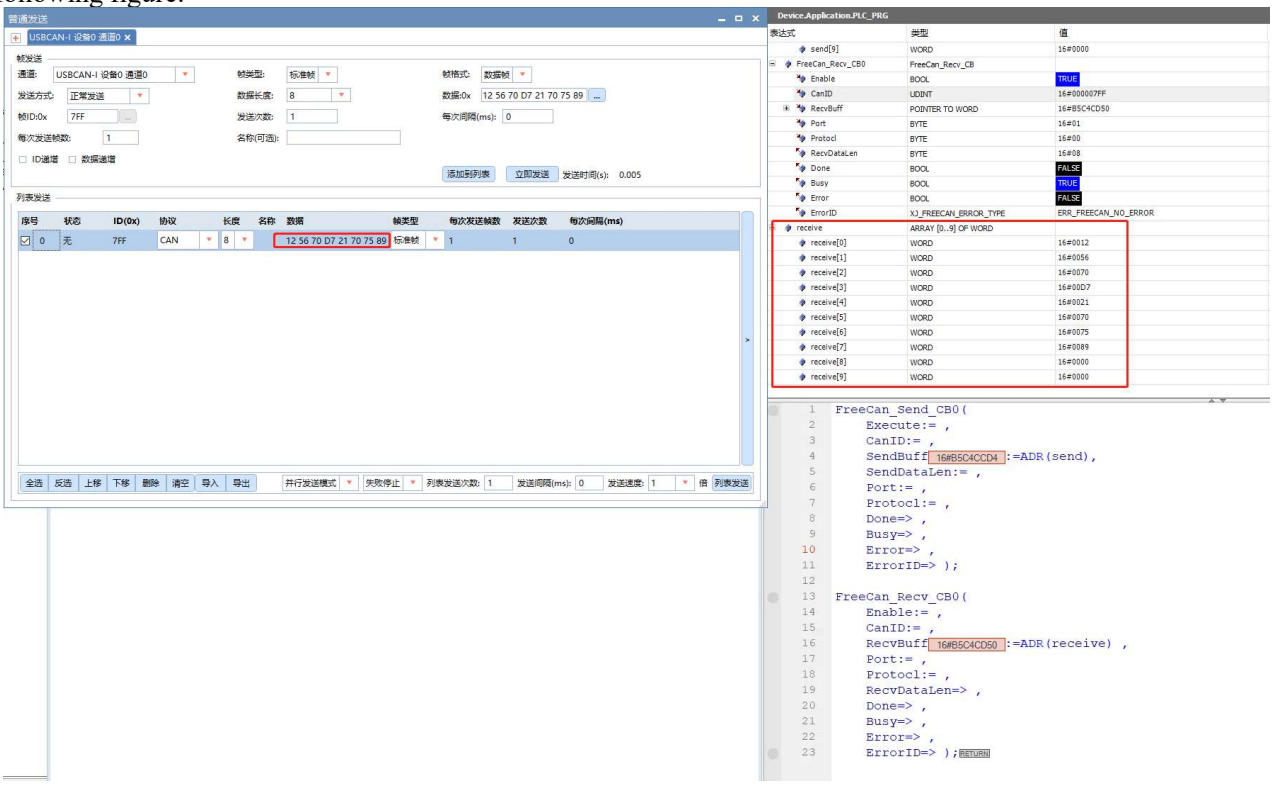
The relevant instructions are not described in this manual. Please refer to the User Manual for XS Series PLCopen Standard Controllers [Instruction Section].

4. XS Studio software interacts with third-party debugging tool ZCANPRO for data exchange. As shown in the following figure:

Sends data to debugging tools through FreeCan_Send_CB; As shown in the following figure:

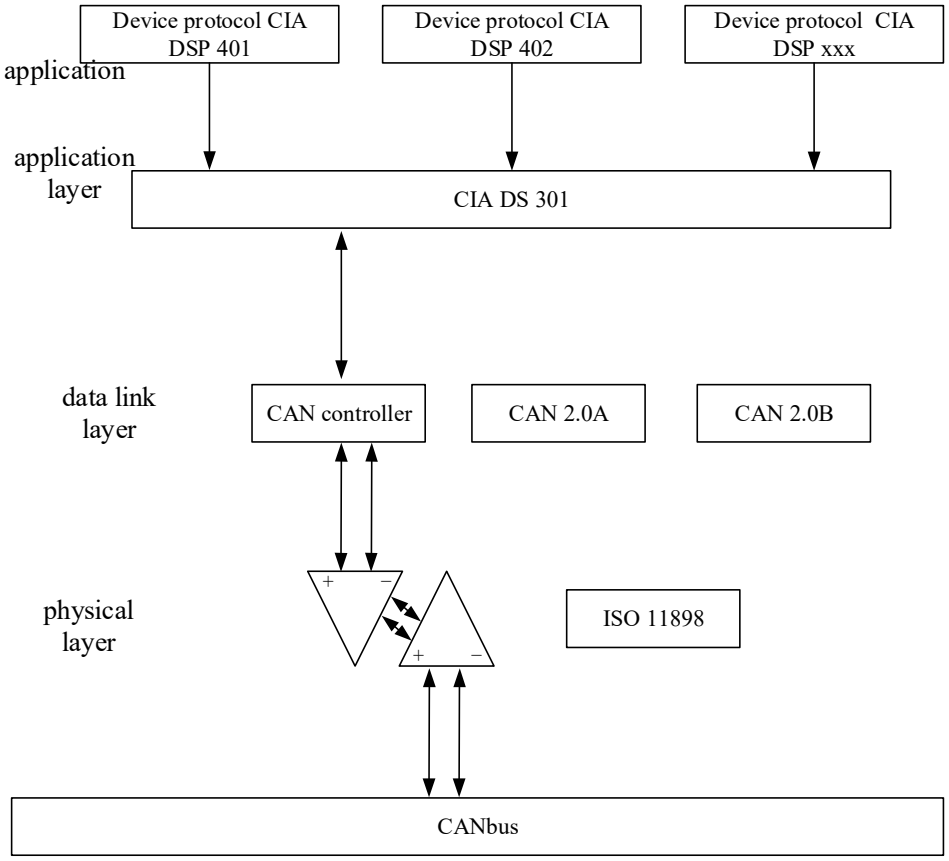



Debugging tool ZCANPRO sends data, receive data through the command FreeCan_Recv_CB. As shown in the following figure:

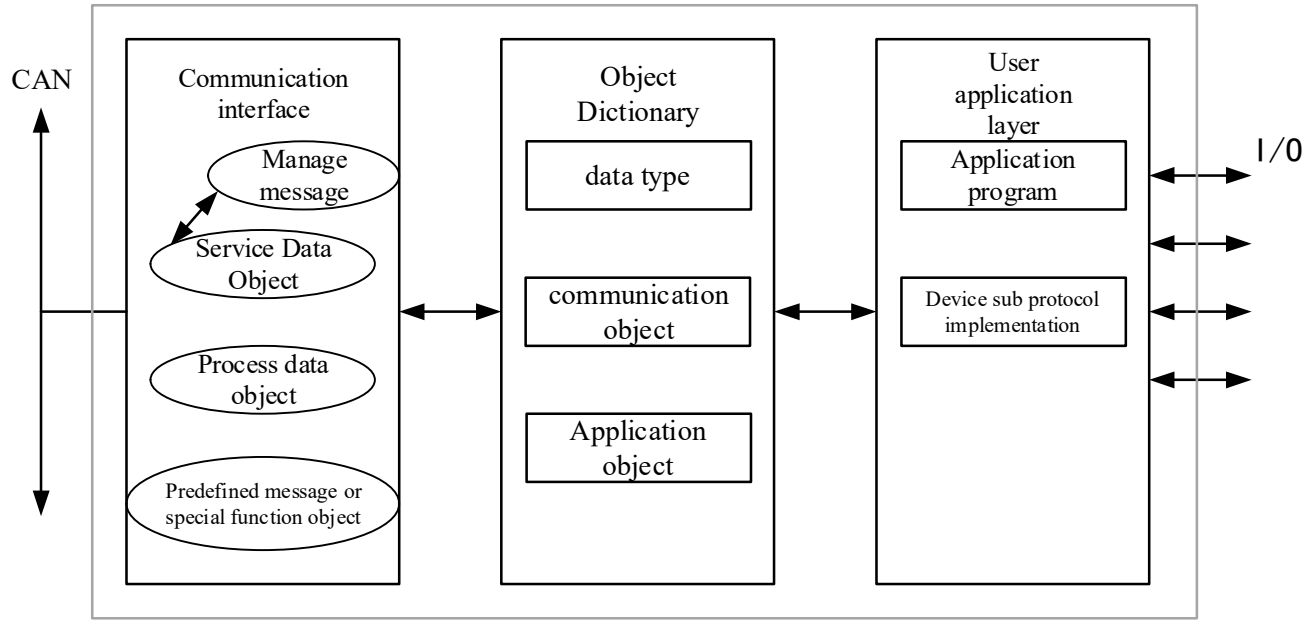


## 3-5-2. CANOpen network

The CANopen protocol was developed in the late 1990s by the CAN in Automation organization based in Nuremberg, Germany, based on the CAN Application Layer.
CANopen is an application layer protocol for a network transmission system based on the CAN serial bus, following the ISO/OSI standard model. The basic protocol is the CANopen Application Layer and Communication Profile (DS 301), which specifies the CANopen protocol layer and communication structure description. On top of the basic protocol, various industries have device sub protocols. The so-called sub protocol refers to redefining or adding new control logic to the internal data meaning of CANopen for application objects in different industries.
In the OSI model, the relationship between CAN standard and CANopen protocol is shown in the following figure:




The OSI model is a conceptual model used to standardize communication functions between various communication technologies. Lower layers describe basic communication (such as raw bitstreams), while higher layers describe things like segmentation of long messages and services such as message initiation, indication, response, and confirmation.
The CANopen protocol is usually divided into three parts: user application layer, object dictionary, and communication. The most crucial one is the object dictionary, which allows CANopen communication to access all parameters of the driver through the object dictionary (OD).
The structure of the CANopen device is shown in the following figure:




- Communication object

The commonly used communication objects in the CANopen protocol include the following:

**1. Network management objects (NMT)**

The network management objects include Boot up messages, Heartbeat protocols, and NMT messages. Based on the master-slave communication mode, NMT is used to manage and monitor various nodes in the network, mainly achieving three functions: node status control, error control, and node startup.

**2. Service Data Object (SDO)**

The service data object is mainly used for parameter configuration between the master node and the slave node. Service confirmation is the biggest feature of SDO, which generates a response for each message to ensure the accuracy of data transmission. In a CANopen system, typically 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 indexes and sub indexes, so the CANopen master node can access the parameters of any object dictionary entry from the slave node, and SDO can transmit data of any length (when the data length exceeds 4 bytes, it is split into multiple packets for transmission).

**3. Process data object (PDO)**

Used to transmit real-time data from one creator to one or more recipients. The data transmission is limited to 1 to 8 bytes. Each CANopen device contains 8 default PDO channels, 4 sending PDO channels, and 4 receiving PDO channels. PDO includes two transmission methods: synchronous and asynchronous, which are determined by the corresponding communication parameters of the PDO.

**4. Synchronization Object (SYNC)**

The synchronization object is a message periodically broadcasted by the CANopen master station to the CAN bus, used to implement basic network clock signals. Each device can decide whether to use this event to synchronize communication with other network devices based on its own configuration.

**5. Emergency message (EMCY)**

The message sent when there is an internal communication failure or application failure error within the device.

- Object Dictionary

The CANopen Object Dictionary (OD) is the most core concept of the CANopen protocol. The so-called "object dictionary" is an ordered set of objects; Each object is addressed using a 16 bits index value. In order to access elements in the data structure, an 8-bit subindex is also defined.
Each node in the CANopen network has an object dictionary. The object dictionary contains all the parameters that describe this device and its network behavior.
The items in the CANopen object dictionary are described by a series of sub protocols. The sub protocol describes the function, name, index, sub index, data type, read/write properties of each object in the object dictionary, as well as whether this object is necessary to ensure compatibility with devices of the same type from different vendors.

The core descriptive sub protocol of CANopen protocol is DS301, which includes the application layer and communication structure description of CANopen protocol. Other sub protocols are supplements and extensions to the description text of DS301 protocol.

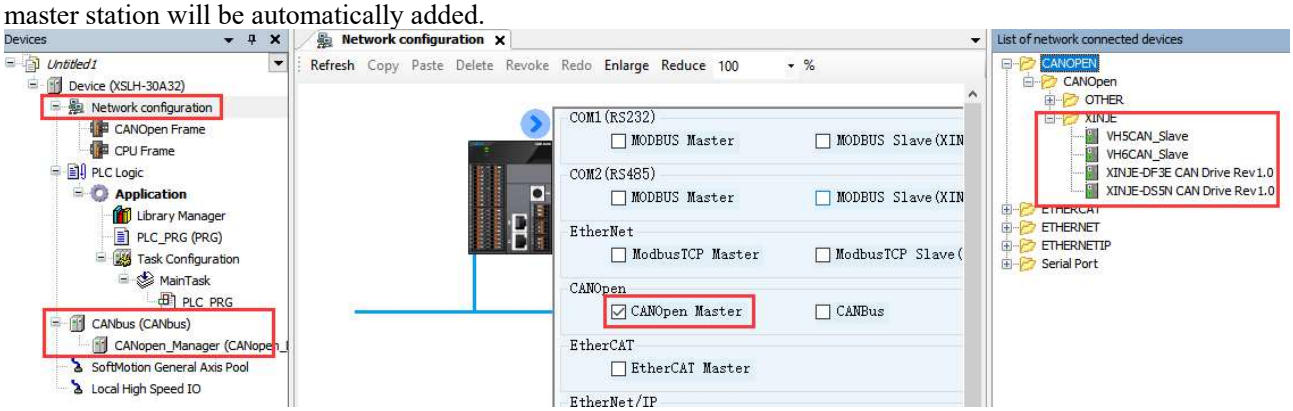
## 3-5-3. CANOpen master configuration

1. Hardware interface

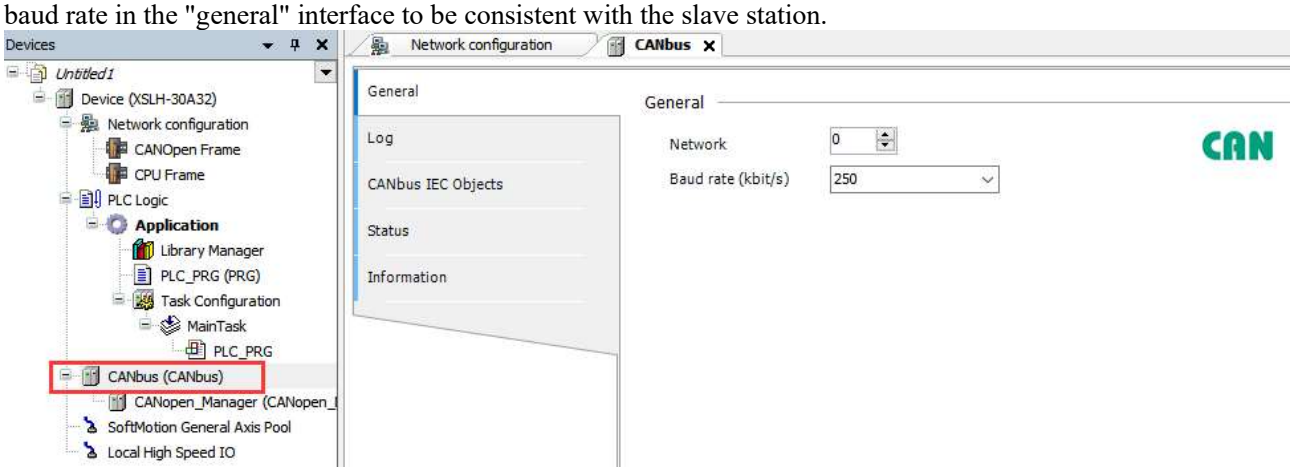
When the device is connected to the CAN bus, it is necessary to connect CAN+ to CAN+ and CAN- to CAN-. If the slave station is a servo, the first (TX+) and second (TX-) wires on one end of the network cable need to be connected to CAN+ and CAN- respectively, and the other end needs to be inserted into the network port of the servo. At the same time, dial 3 and 4 on the PLC are built-in terminal resistors that need to be set to ON. In order to enhance the reliability of CAN communication and eliminate the reflection interference of CAN bus terminal signals, terminal resistors are usually added to the farthest two endpoints of the CAN bus network. If other CANopen devices do not have built-in terminal resistors, users need to install them themselves.

2. Software setting

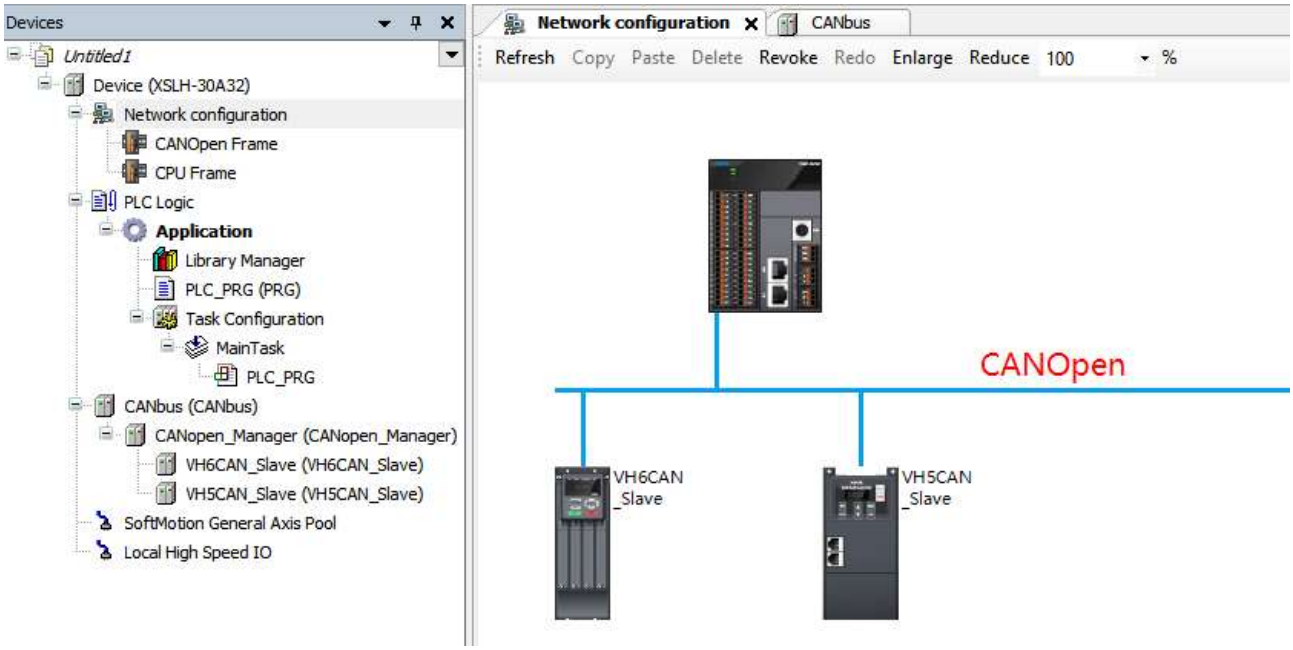
(1) Activate the CANopen bus in the network configuration. After activating the CANopen bus, the CANopen master station will be automatically added.



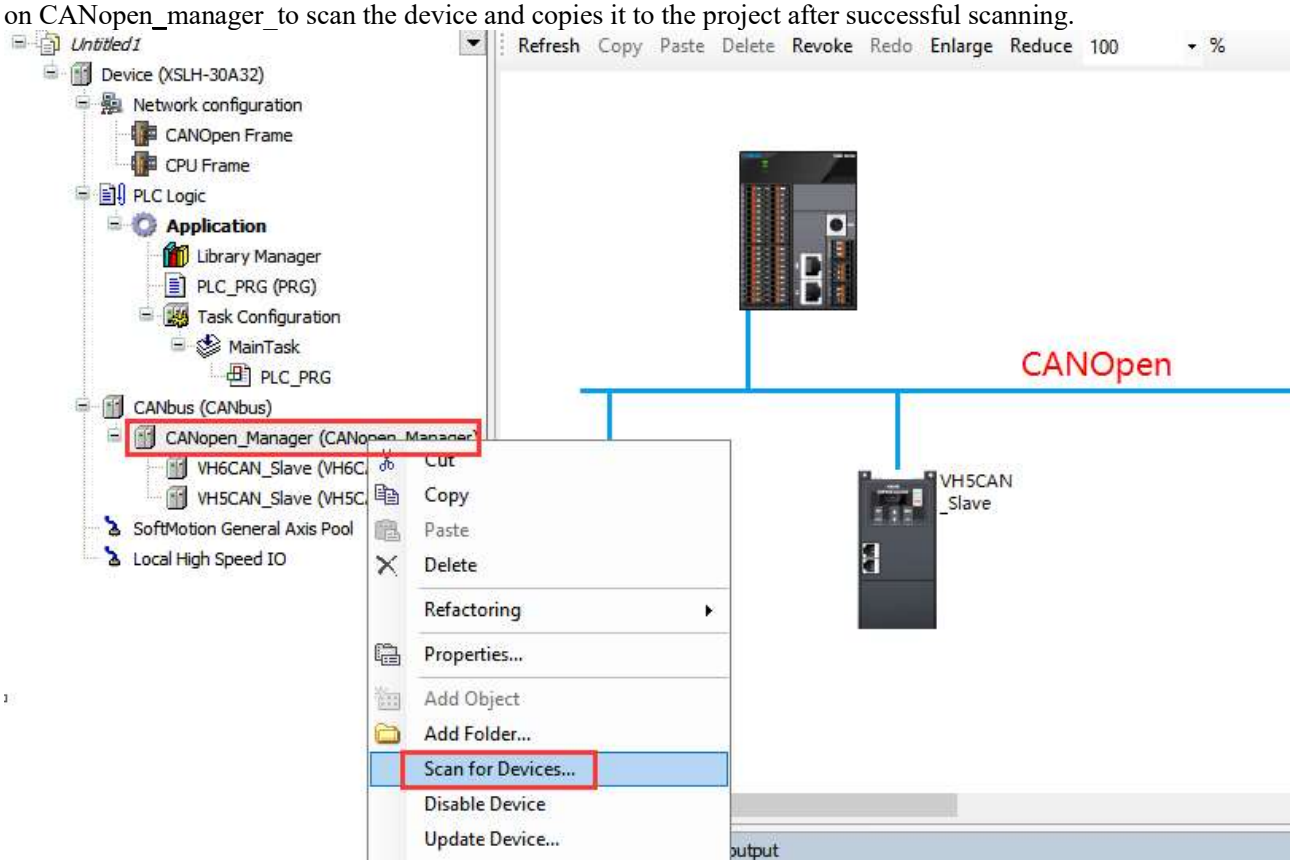

(2) After successful addition, you can see "CANBus" under the device bar. Double click "CANBus" and set the baud rate in the "general" interface to be consistent with the slave station.



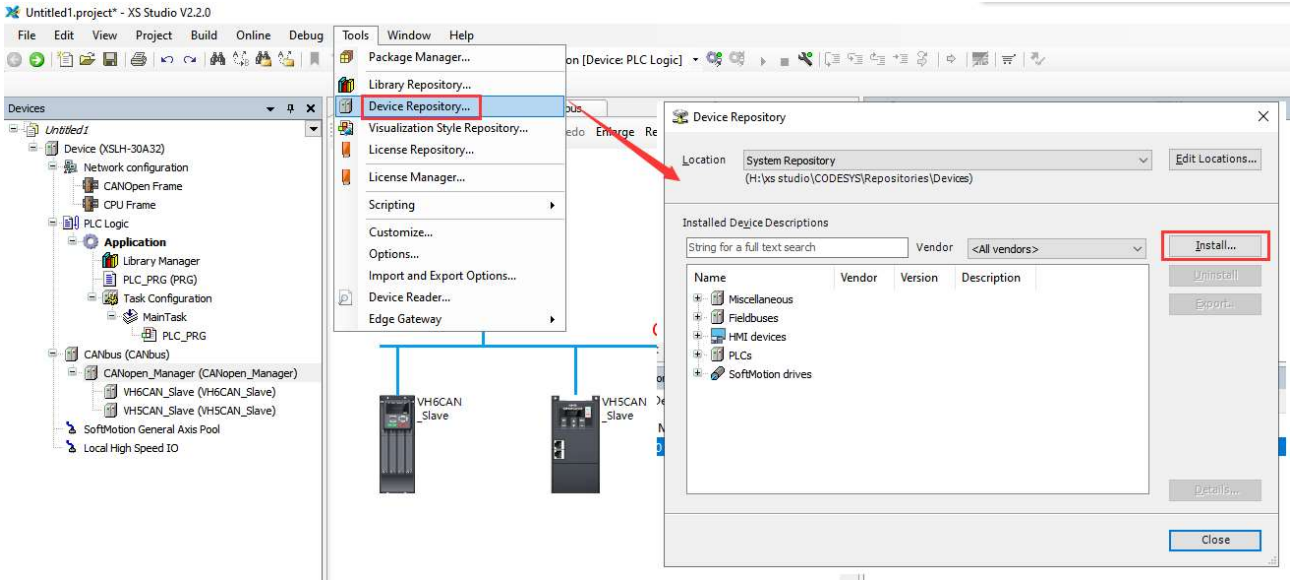

The CANopen slave module can be added through the "Network Device Connection List" on the right, and the configuration corresponding device tree will appear in the left view of the interface. As shown in the figure:



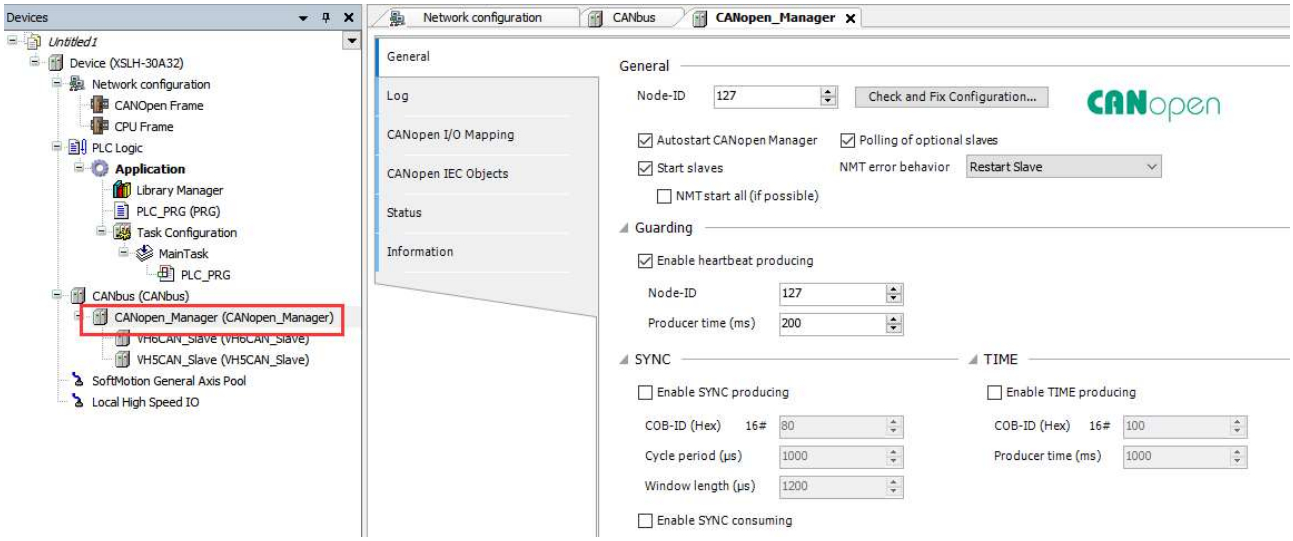

(3) After adding "CANopen_manager ", you need to download the program first. After downloading, right-click on CANopen_manager_to scan the device and copies it to the project after successful scanning.




If the scan is not successful, you can check if the EDS file has been imported. In Tools - Device Repository, import the EDS file from the slave station. After scanning, the node ID of the slave station will be automatically recognized. If you manually add a slave device, you need to manually modify the slave ID.




(4) In "CANopen_Manager", it is necessary to set up the CANopen master station.




- Node ID: The unique identification number of the master station in the CANopen network, with a default value of 127 and a range of 1-127.
- Check and fix configuration: After clicking to enter, if there are any errors, you can click "Auto Repair".
- Auto start CANopenManager (default: enabled): When checked, CANopenManager will automatically restart after all slave stations are ready.
- Polling of optional slaves (default: enabled): When the slave does not respond in the boot sequence, the CANopen manager queries it once per second until it responds. Continuously polling the slave station will increase the bus cycle time, which can interfere with applications (especially motion applications). You can disable polling to avoid this behavior. If polling is disabled, the slave server will be detected again when sending a startup message.
- Start Slaves (default: enabled): The CANopen Manager is responsible for starting the slave.
- NMT start all (if possible): If the start slave option is activated (default: disabled), the CANopen manager uses the "NMT start all" command to start all slaves. As long as the optional slave station is not ready to start, the "NMT all start" command will not be executed. In this case, the CANopen manager starts each slave separately. The "NMT all start" command can only be guaranteed in projects without optional slave devices.
- NMT error behavior: Restart Slave - If an error occurs during slave monitoring (NMT error event), the stack will automatically restart the slave (NMT reset+SDO configuration+NMT start); Stop Slave - If an error occurs during slave monitoring (NMT error event), the slave will stop. Then, you must use the CiA405 NMT function block to reset the slave from the application.

- Guarding

- Enable heartbeat producing: If this option is enabled (default: disabled), the main site will send heartbeat information.
- Node-ID: Unique identifier for sending heartbeat information, default to the master node ID, ranging from 1

to 127 (decimal).

- Producer time (ms): The time interval for sending heartbeat information, in milliseconds, ranging from 1ms to 65535ms, and is an integer multiple of the bus task time.

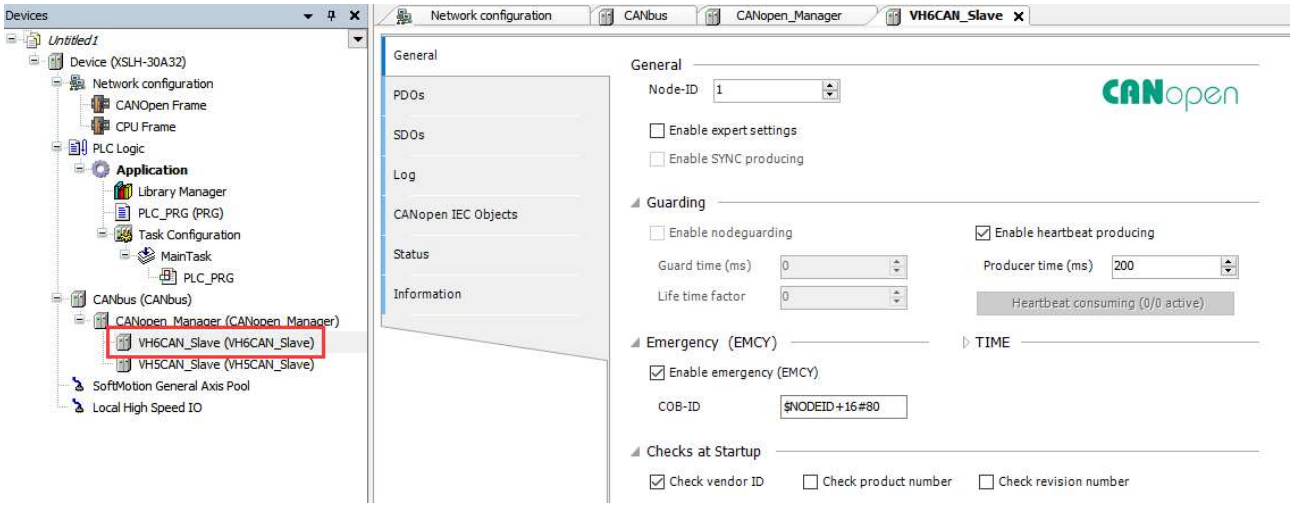
- SYNC:

- Enable SYNC producing: If this option is enabled (default: enabled), the main site will send synchronization information. A CANopen bus system can only have one station enabled for synchronous production. The synchronization type PDO sends information according to the set type after the synchronization information is sent.
- COB-ID: Communication object identification, this setting is used to identify the synchronization message ID. Value cannot be modified, it is 16#80.
- Cycle period (us): The synchronization information is sent at a time interval defined by the synchronization cycle, which is measured in microseconds and ranges from 100-4294967295us, and is an integer multiple of the bus task time.
- Window length (us): Time window length in microseconds for synchronizing PDO.
- Enable SYNC consuming: If this option is enabled (default: disabled), another device must generate SYNC messages received by the CANopen manager.

- TIME

- Enable time producing: If this option is enabled (default: disabled), the CANopen manager sends a TIME message
- COB-ID: (Communication object identifier), Identify the timestamp of the message. Default value: [0... 2047], preset 16#100.
- Producer time (ms): The time interval when sending a timestamp, which must be a multiple of the task cycle time, within the range of [0.. 65535].

(5) Double click on the slave device and configure the slave basic parameters, PDO configuration, and SDO configuration in the CANopen slave station.




- General

- Node ID: The unique identifier range of the slave station in the CANopen network is 1-127 (decimal), which needs to be consistent with the slave station itself.
- Enable Expert Settings: If this option is enabled (default: disabled), all settings predefined by the device description (EDS file) are displayed.
- Enable SYNC producing: If this option is enabled (default: disabled), this slave will send synchronization information. A CANopen bus system can only have one enabled synchronous production. Synchronize sending parameters using the synchronization configuration parameters of the main station.
- Enable SYNC producing: Only available when the "Enable SYNC producing" option is selected in "CANopen Manager". If this option is enabled (default: disabled), I/O transmission is synchronized on the bus. The slave station acts as a synchronous producer. The parameters for the synchronization interval are defined in the settings of "CANopen Manager".

- Guarding

- Enable node guarding: If this option is enabled (default: disabled), the CANopen manager sends messages to the slave station within the protection time interval (ms). If the slave station does not respond with the given protected COB-ID (communication object identifier), the CANopen manager will resend this message the

number of times defined in the lifetime factor, or until the slave station responds. If the slave station does not respond, mark it as "unavailable".

- Guard time (ms): The interval between sending messages (default: 200 milliseconds).
- Life time factor: When there is no response from the slave station, node protection errors will be established based on multiplying the life time factor by the protection time.
- Enable heartbeat producing: The module sends a detection signal at the time interval given in the producer time (ms).
- Producer time (ms): Refer to the time set in the device description file.

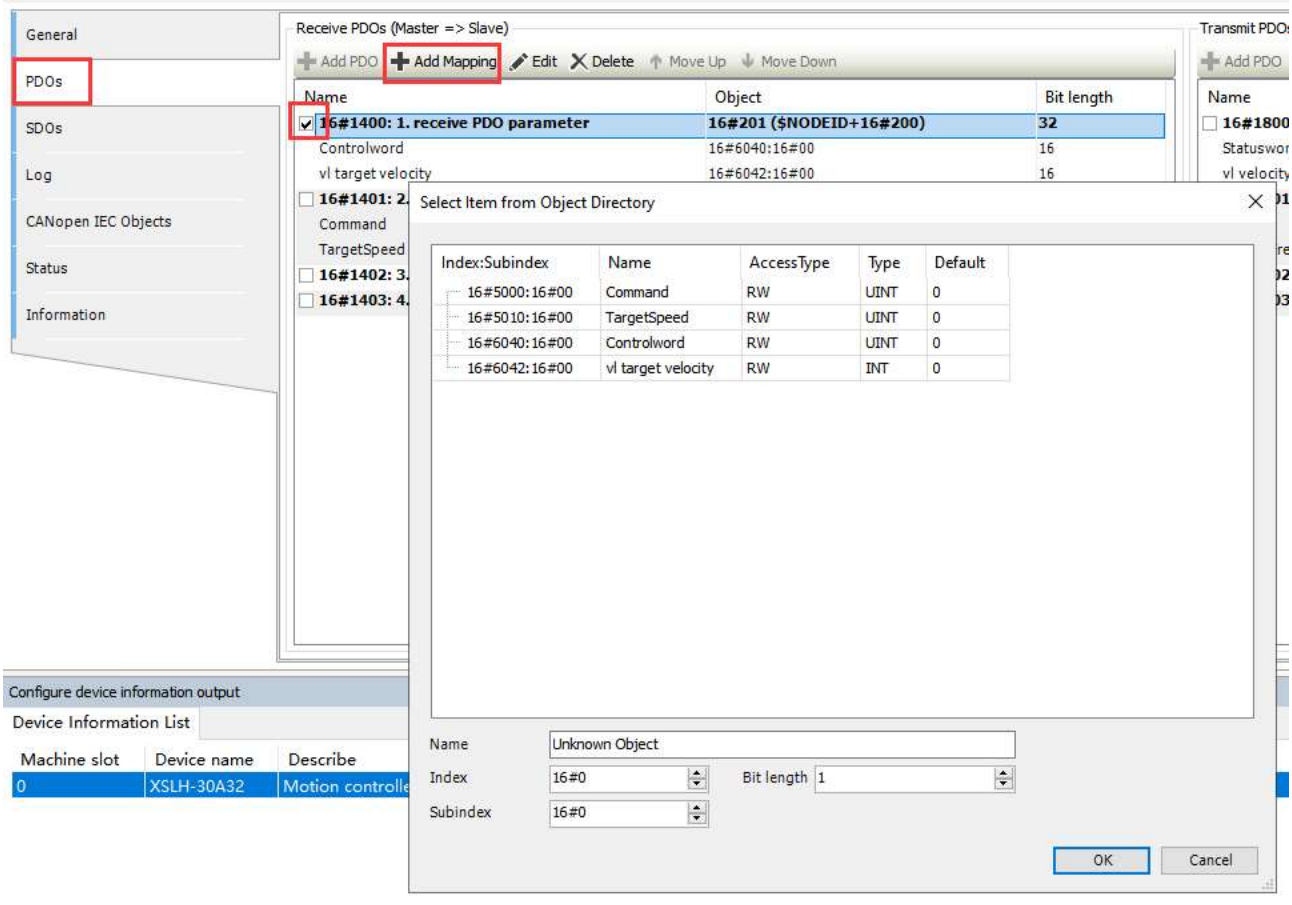
  - Emergency

- Enable emergency: When an internal error occurs, the slave station sends an emergency message with a unique COB-ID.
- COB-ID: The COB-ID of the emergency message sent by the slave station, default to $NODEID+16#80.
- Time - The availability of this feature depends on the device description
- Enable time producing: The device sends a time message.
- COB-ID (hexadecimal): (Communication object identifier): Identifies the timestamp of the message.
- Enable time consuming: Device processing time messages.

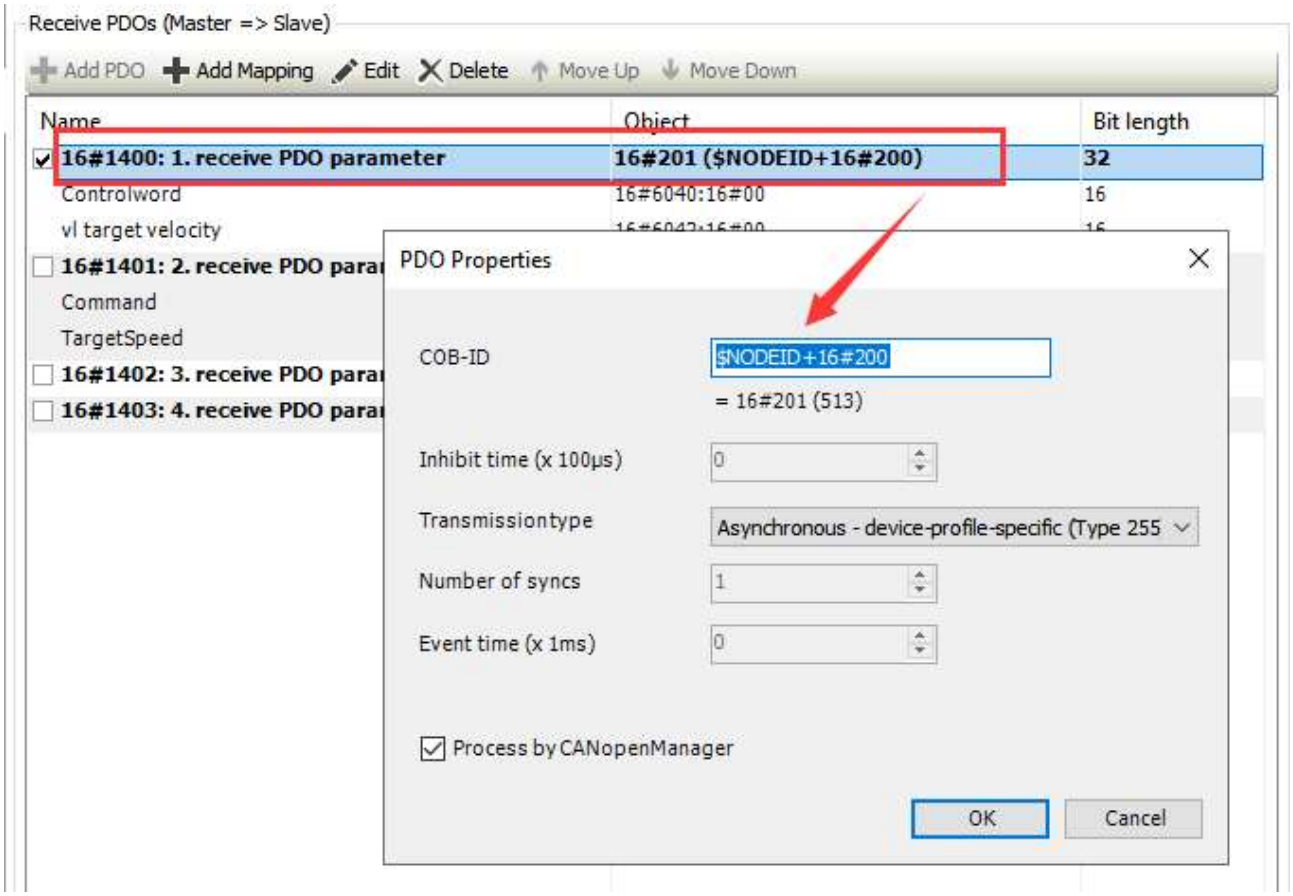
  - Checks at startup

Read the corresponding information from the firmware of the CANopen slave station (0x1018 identity object) and compare it with the information in the EDS file. If there is a difference, stop the configuration and do not start the slave station.

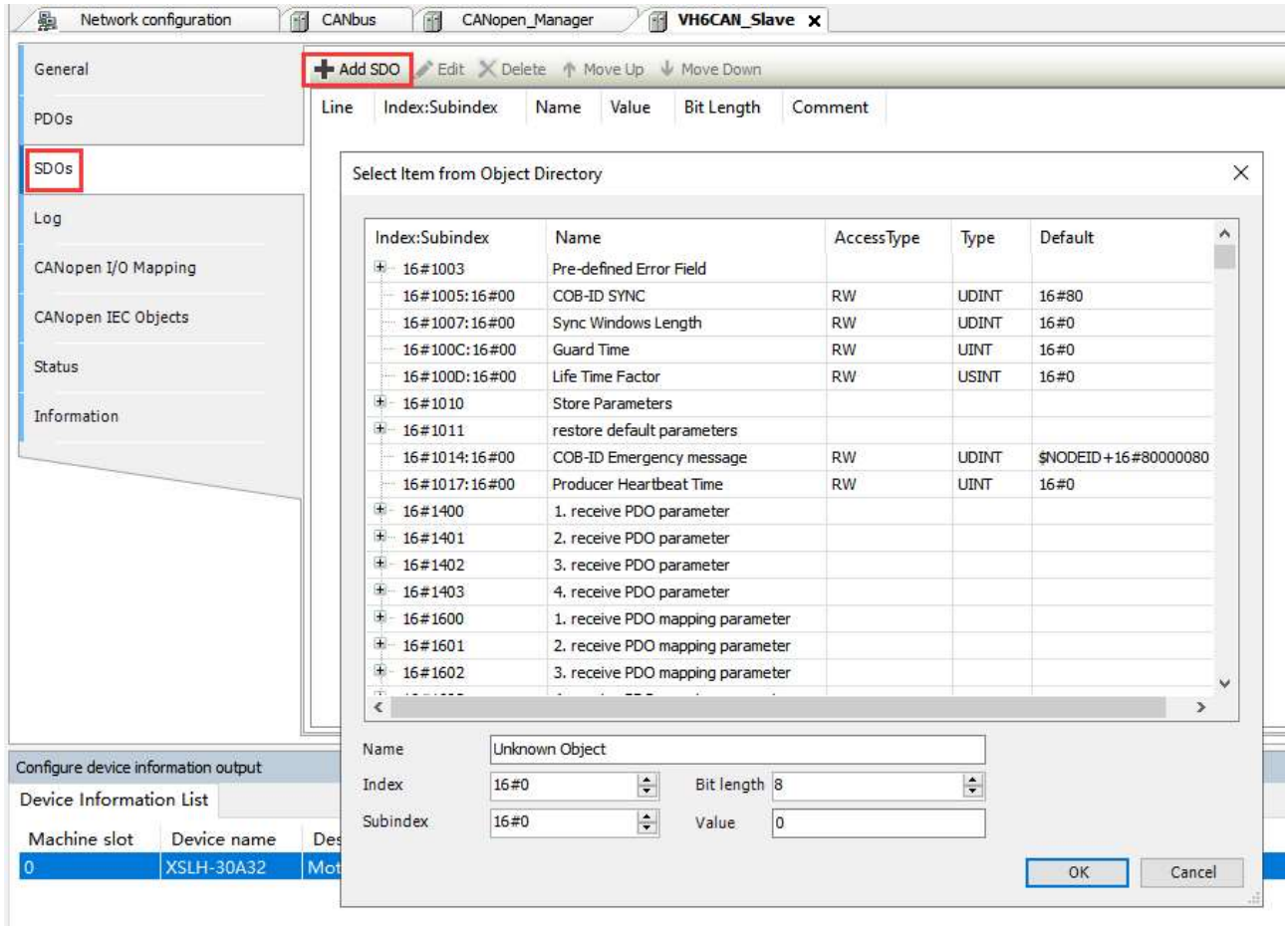
- Check vendor ID: Check supplier ID at startup.
- Check product number: Check product number at startup.
- Check revision number: Check the revision number at startup.
- PDOs:
- PDO (Process Data Object) is used for real-time data transmission between the master and slave stations, receiving PDO as the real-time data sent from the master station to the slave station.
- On the PDOs interface, receive PDO from index 1400-1403 in the object dictionary, send PDO from index 1800-1803 in the object dictionary, click on the index to add the required communication parameters, select the index and sub index, and click "OK". If users need to add/delete/modify mapping addresses, they need to set them in "Receive PDOs" and "Transmit PDOs".



Double click bold font - index to set specific PDO, COB-ID, and transmission type.

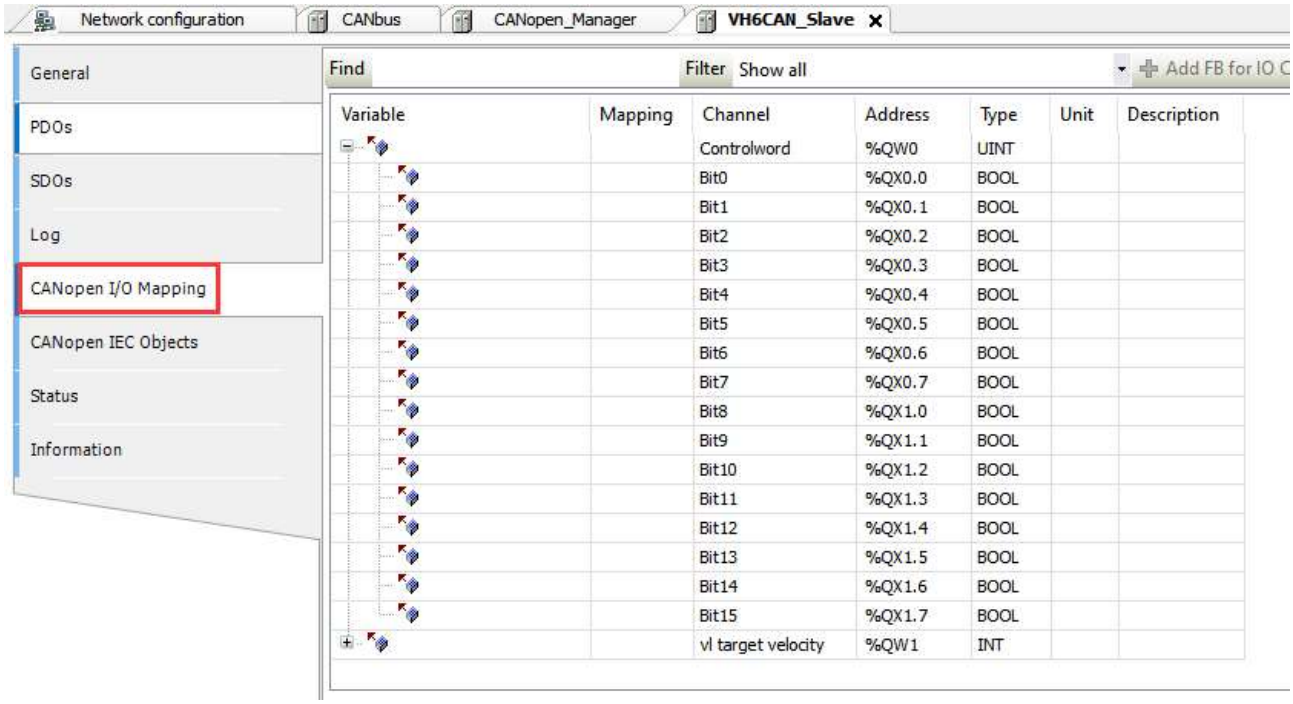



SDOs:



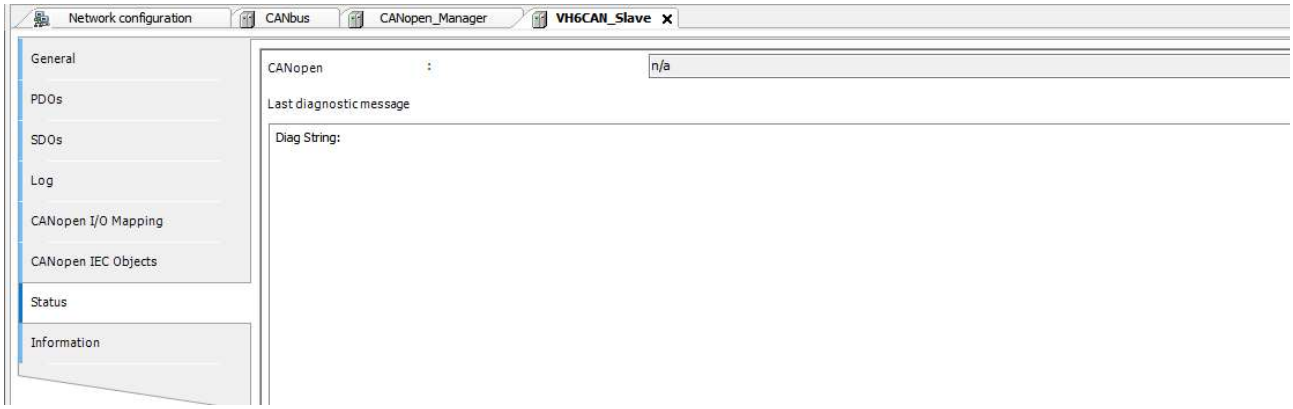
CANopen/IO mapping:

You can view CANopen/IO mapping relationships, functional descriptions, actual addresses, and types of mapping variables.




Status:

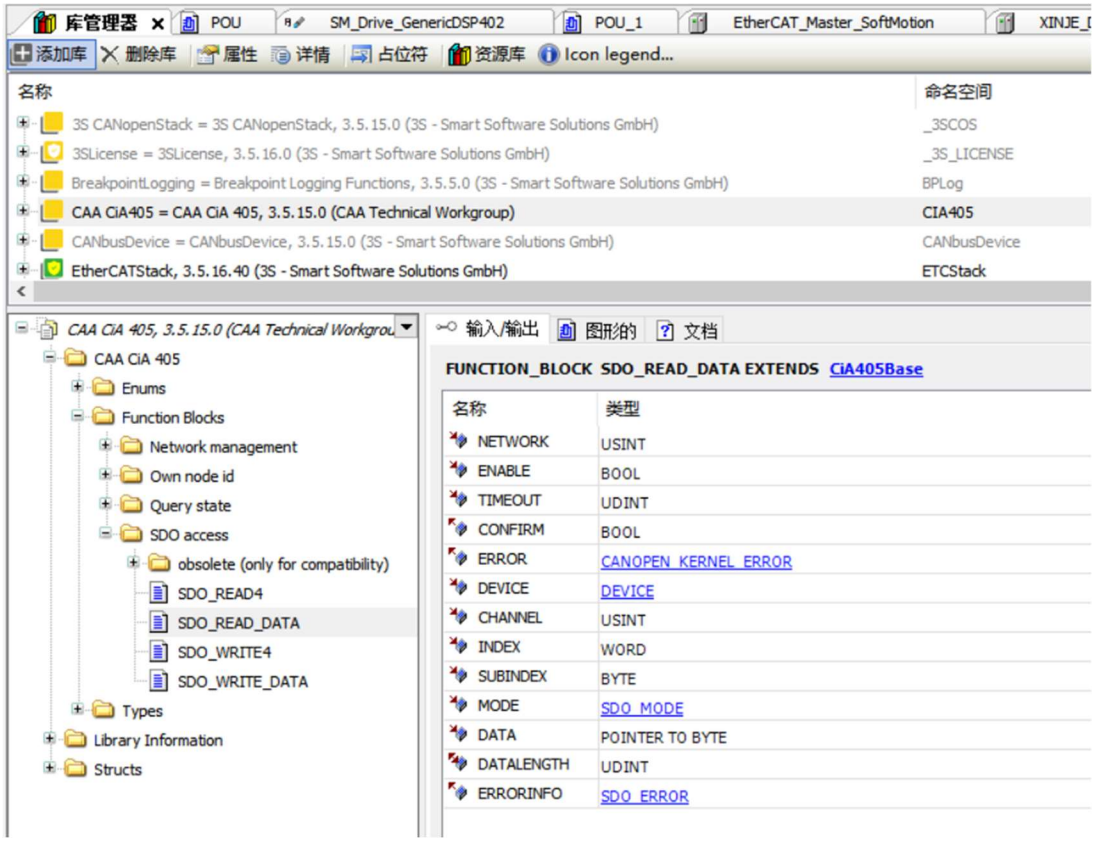
It can provide users with device status (such as "running", "not running") and diagnostic information of the device.



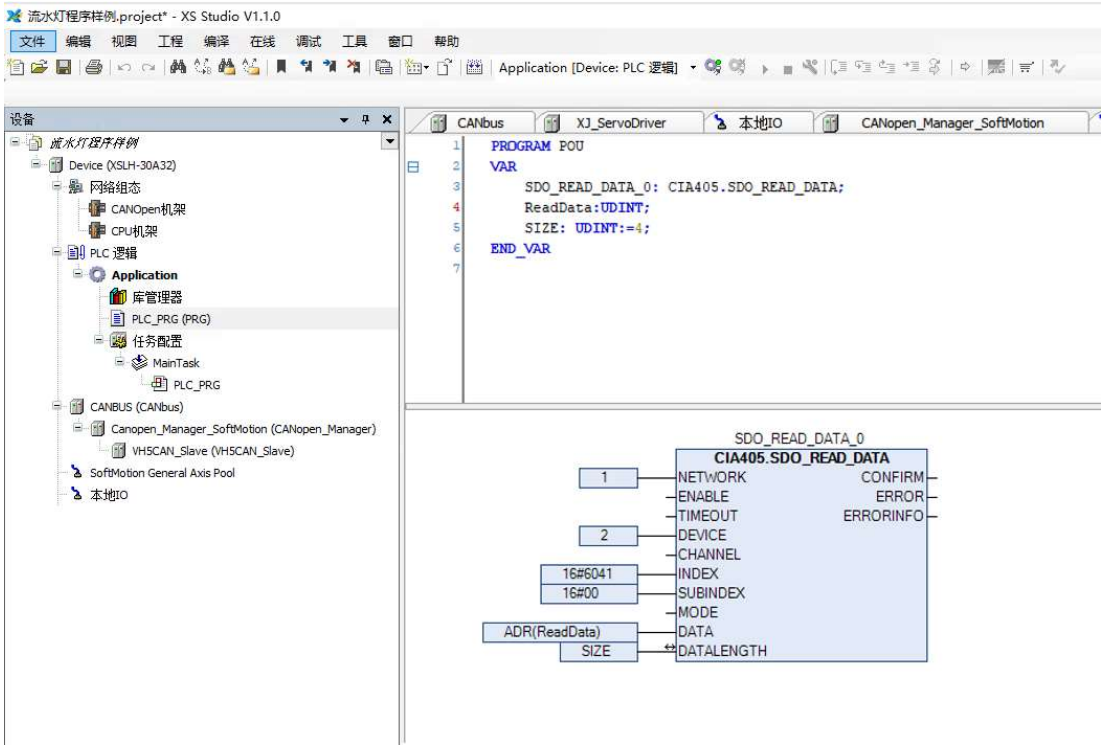

(6) SDO communication function block

Using PDO method for data exchange is simple and direct. But due to quantity limitations, and these data will occupy the bus, it will result in not being able to connect too many devices on the bus. SDO communication is mainly used for configuring parameters of master node to slave nodes, and for transmitting low priority data between devices.

- If using SDO communication method, it is necessary to add the library "CAA CIA405". After adding it, the "SDO access" folder can be seen in the library file.

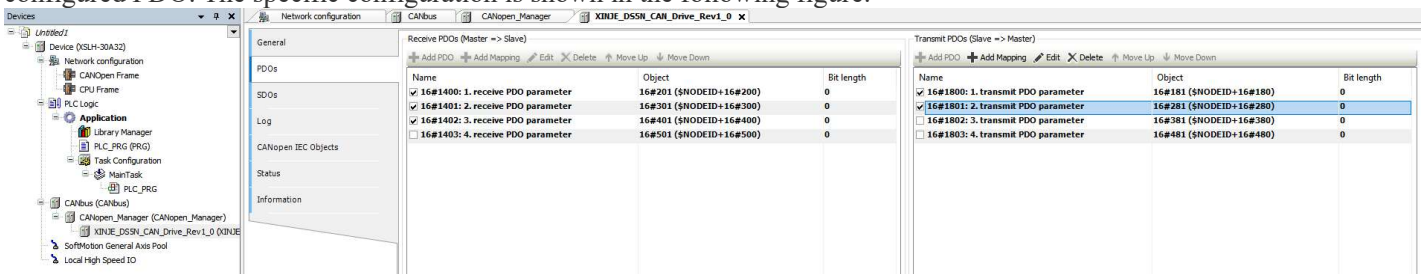



For example, by adding the function block " CIA405.SDO_READ_DATA ", the parameter can be read through the program function block.

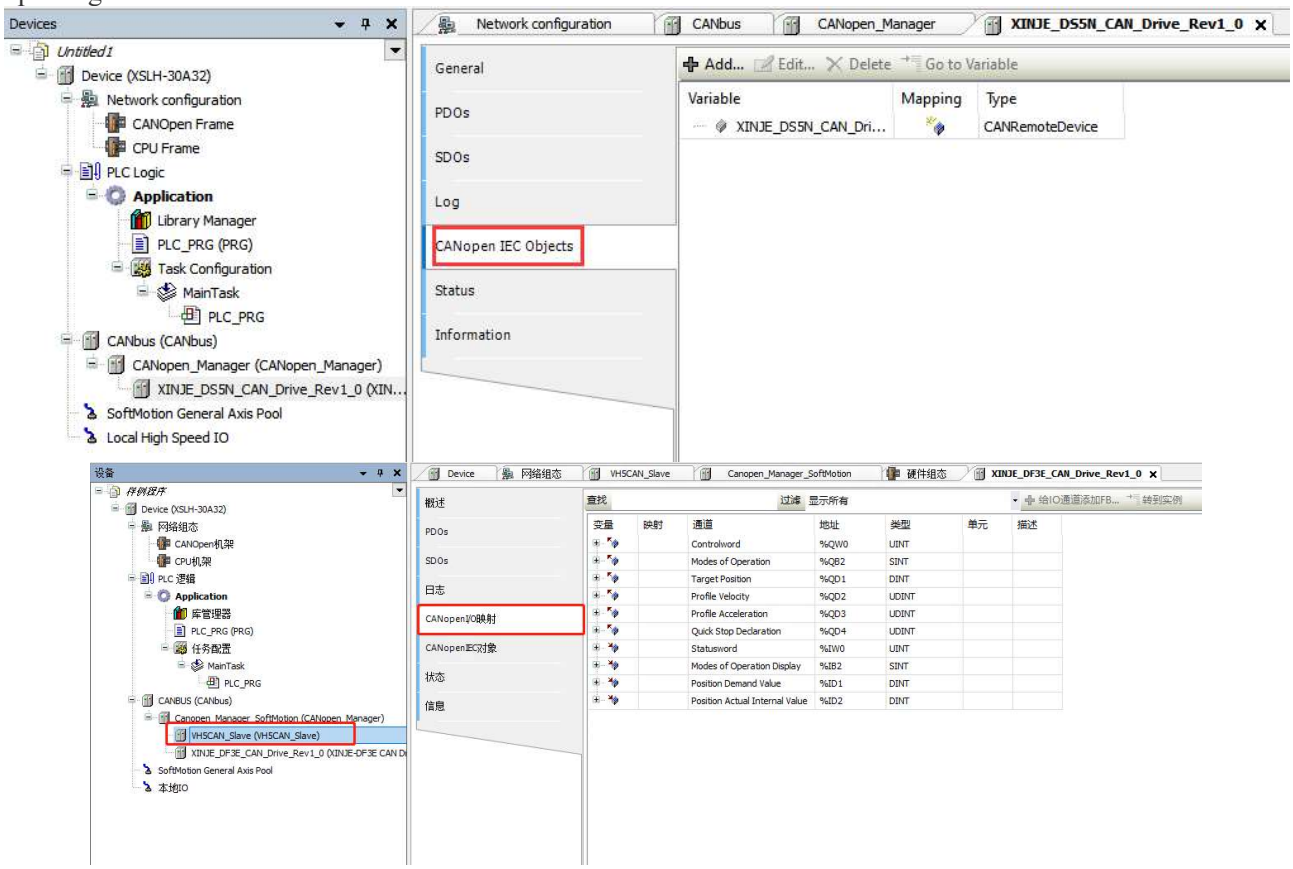


## 3-5-4. Application example

Using the Xinje DS5N1 servo as the slave station and setting it to PP mode, configure the object binding of TxPDO and RxPDO in the PDOs interface of "Xinje_DS5N_CAN_Drive". Here, bind several commonly used objects in PP mode. If there are other needs, you can add them yourself. After completing the binding, enable the configured PDO. The specific configuration is shown in the following figure:



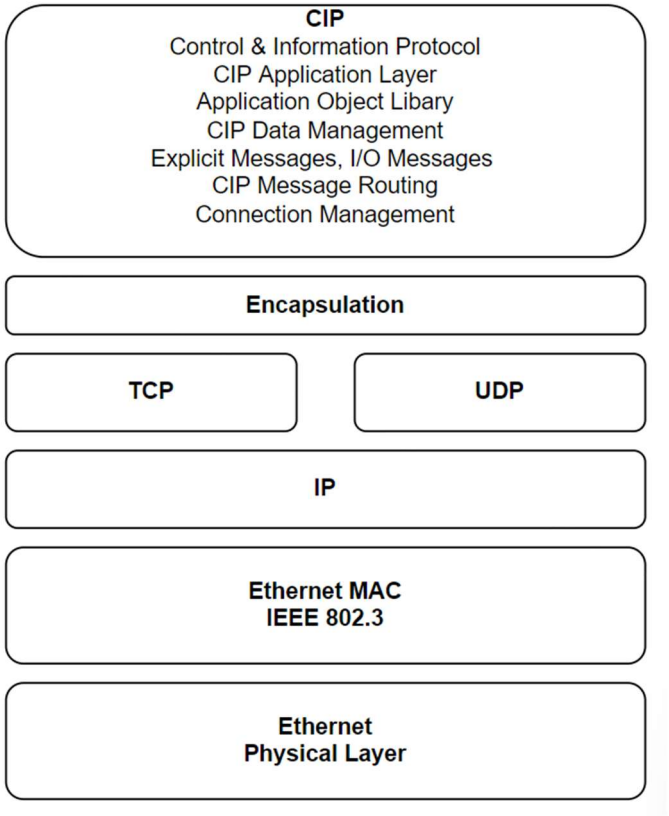

You can see the mapping address of the parameters in the CANopenI/O mapping interface. You can set "keep updating variables" as needed.




%QB2 can be set to 1 (PP mode) in this interface address, and %QW0 (control word 6040h) can be modified to 0X6-->0X7-->0XF/0X4F to enable the slave station. By setting the given position, speed, acceleration and deceleration parameters, and then modifying the control word 0XF -->0X1F to achieve absolute position motion, and 0X4F -->0X5F to achieve relative position motion. Other monitoring parameters start from %IW0.

## 3-6. EtherNet/IP communication

Ethernet/IP is an industrial application layer protocol for industrial automation applications. It is an industrial Ethernet standard jointly developed by Control Net International and ODVA (Open DevicenetVendors Association) in March 2000. It is built on top of the standard UDP/IP and TCP/IP protocols, and uses fixed Ethernet hardware and software to define an application layer protocol for configuring, accessing, and controlling industrial automation devices. The structure of each layer is shown in the figure:

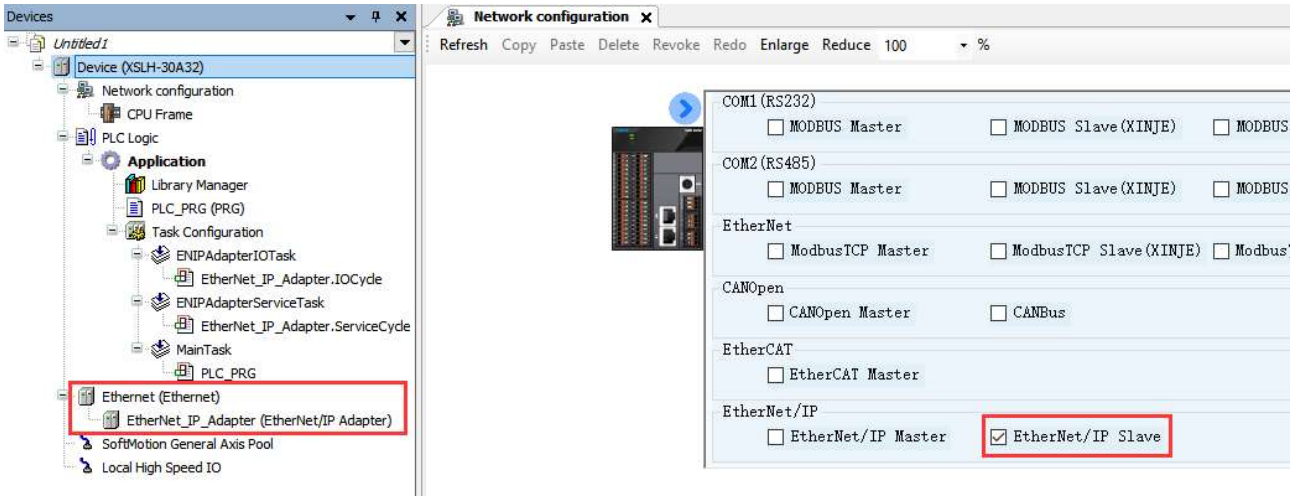



The method of implementing real-time performance through Ethernet/IP is to add the Common Industrial Protocol (CIP) protocol on top of the TCP/IP layer for real-time data exchange and running real-time applications.
The technical characteristics of EtherNet/IP protocol:

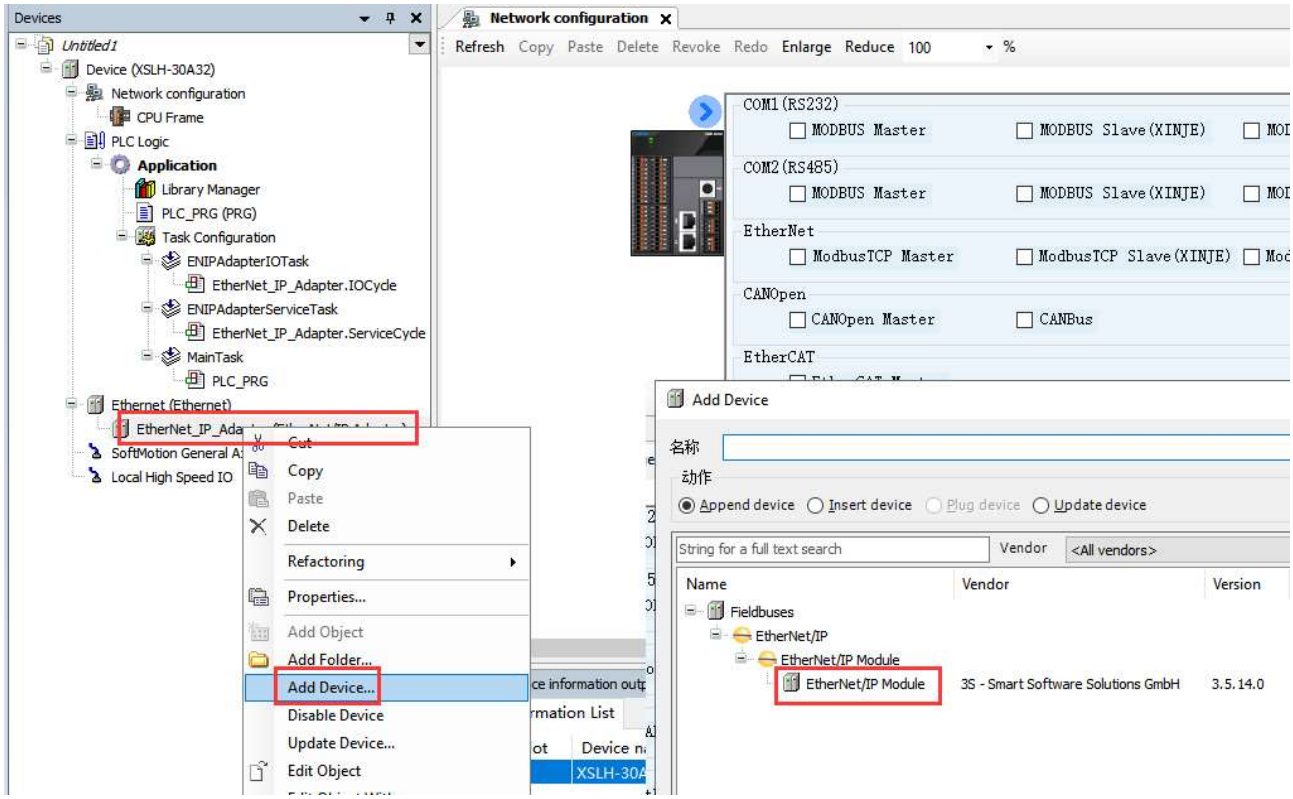
- The method of implementing real-time performance through Ethernet/IP is to add the Common Industrial Protocol (CIP) protocol on top of the TCP/IP layer for real-time data exchange and running real-time applications.
- Ethernet/IP adopts standard Ethernet technology at the physical layer and data link layer, and uses IP protocol, TCP, UDP protocol to transmit data at the network layer and transport layer. UDP is a non connection oriented protocol that can work in both unicast and multicast modes, providing only the ability to send datagrams between devices. For high real-time I/O data, motion control data, and functional safety data, use UDP/IP protocol to send. And TCP is a reliable, connection oriented protocol. For data with low real-time requirements (such as parameter settings, configuration, and diagnosis), TCP/IP protocol is used to send.
- Ethernet/IP adopts a producer/consumer data exchange model. Producers send packets with unique identifiers to the network. Consumers receive the required data from the network through identifiers as needed. In this way, the data source only needs to transmit the data to the network at once, and other nodes selectively receive the data, thereby improving communication efficiency.
- Ethernet/IP enables the transmission of non real-time data and real-time data under the control of the CIP protocol. CIP is an object-oriented protocol that provides end-to-end industrial equipment, independent of the physical layer and data link layer, allowing devices from different suppliers to interact well. In addition, in order to achieve better clock synchronization performance, ODVA introduced IEEE 15888 into Ethernet/IP in 2003 and developed the CIPsync standard to improve the clock synchronization accuracy of Ethernet/IP.

## 3-6-1. EtherNet/IP slave example

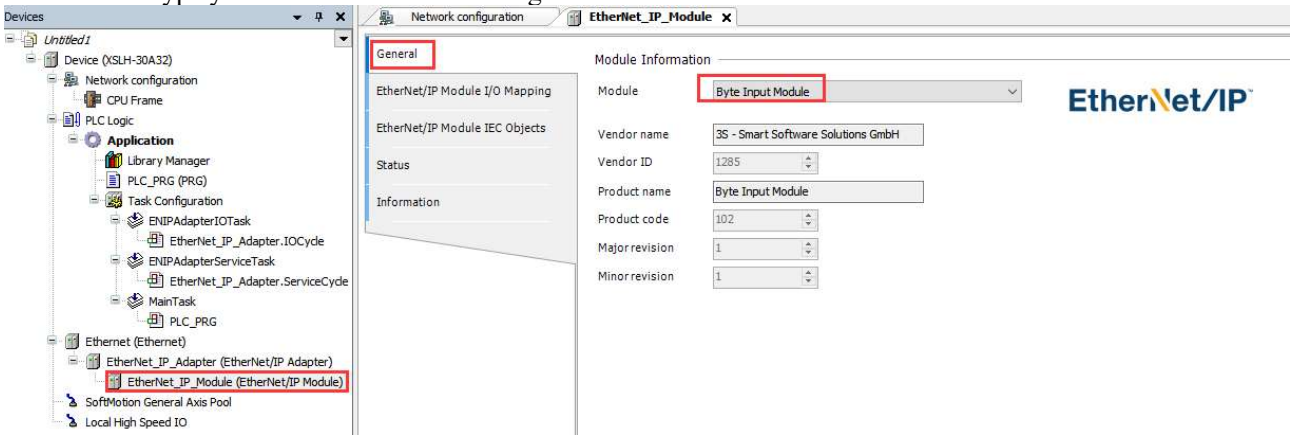
1. Click on the enable window in the network configuration and select "EtherNet IP Slave". under the left device tree node, "EtherNet/IP Adapter" will be automatically added

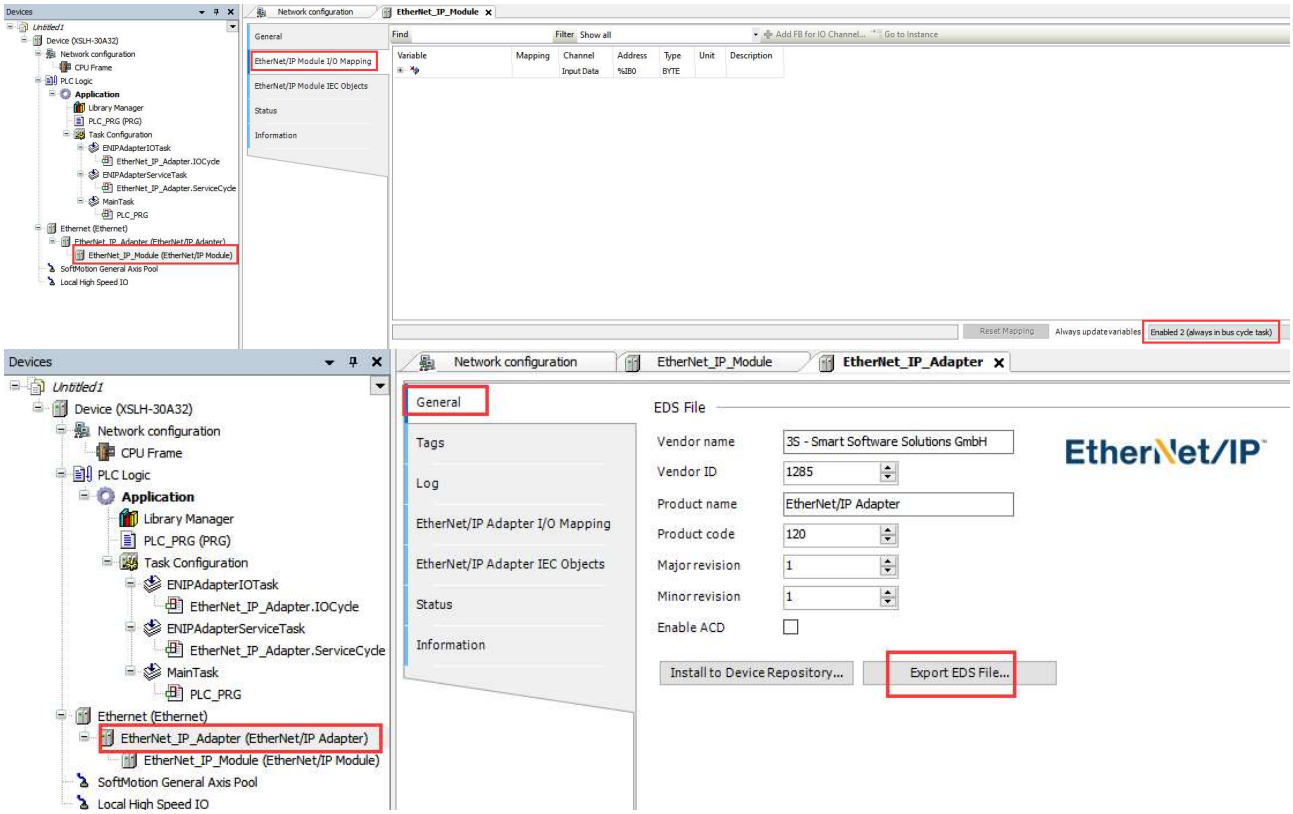



2. Right click “EtherNet/IP Adapter”→ add “EtherNet/IP Module”.

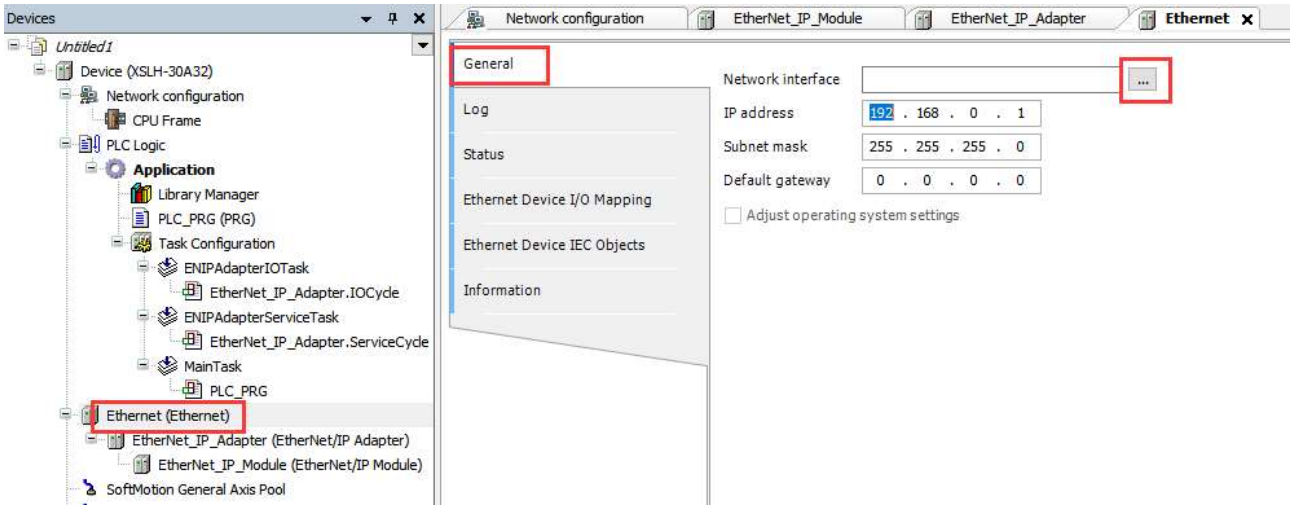



Add the data type you want to access for testing.

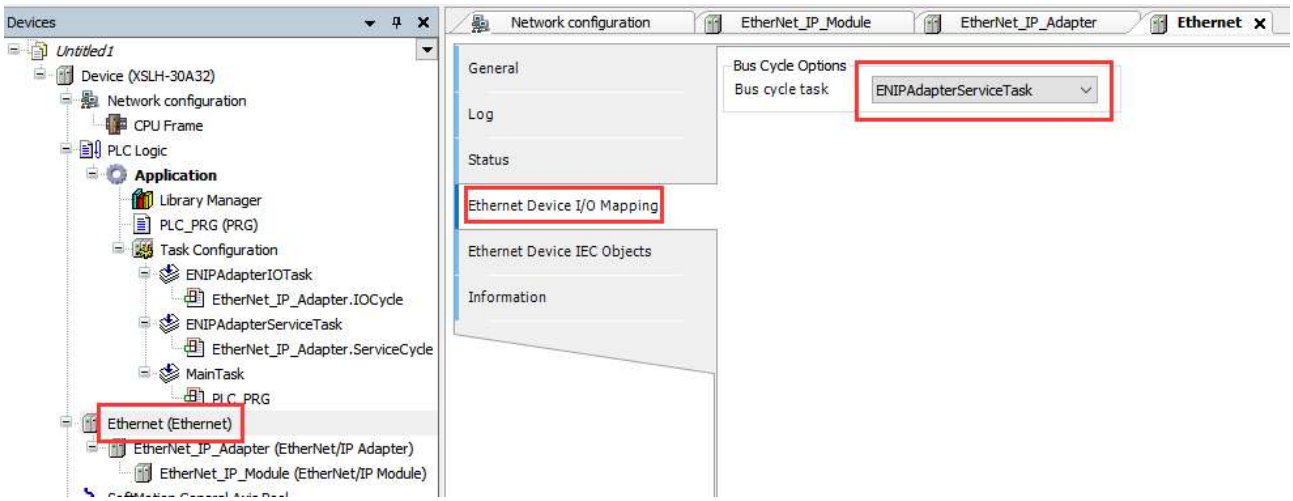





Select the network interface for Ethernet IP communication.

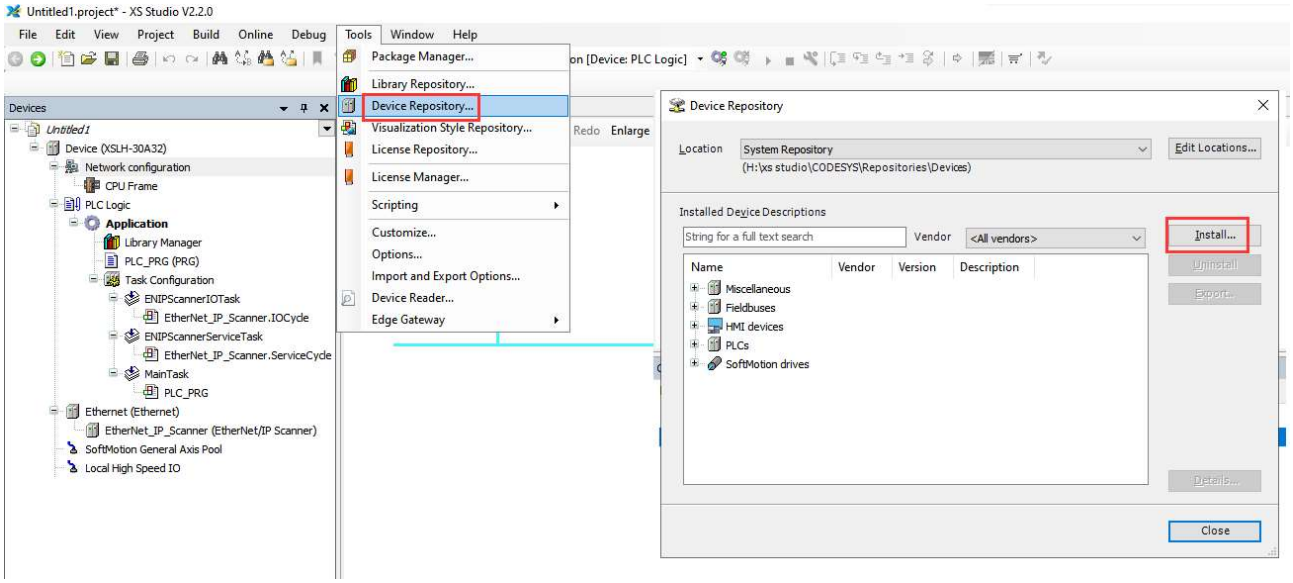


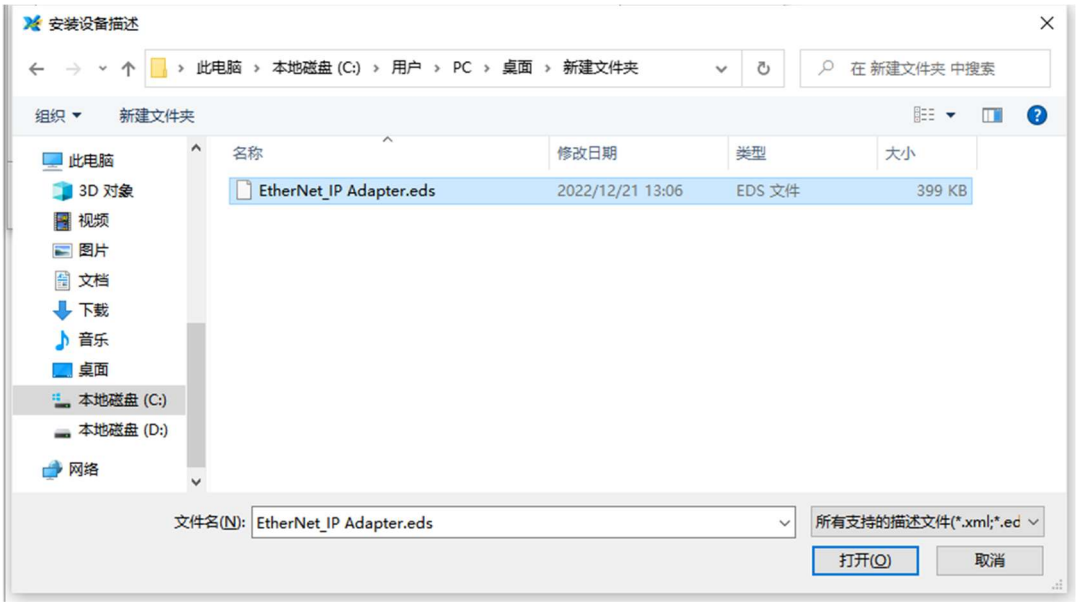

Bus cycle task set to “ENIPAdapterIOTask”:

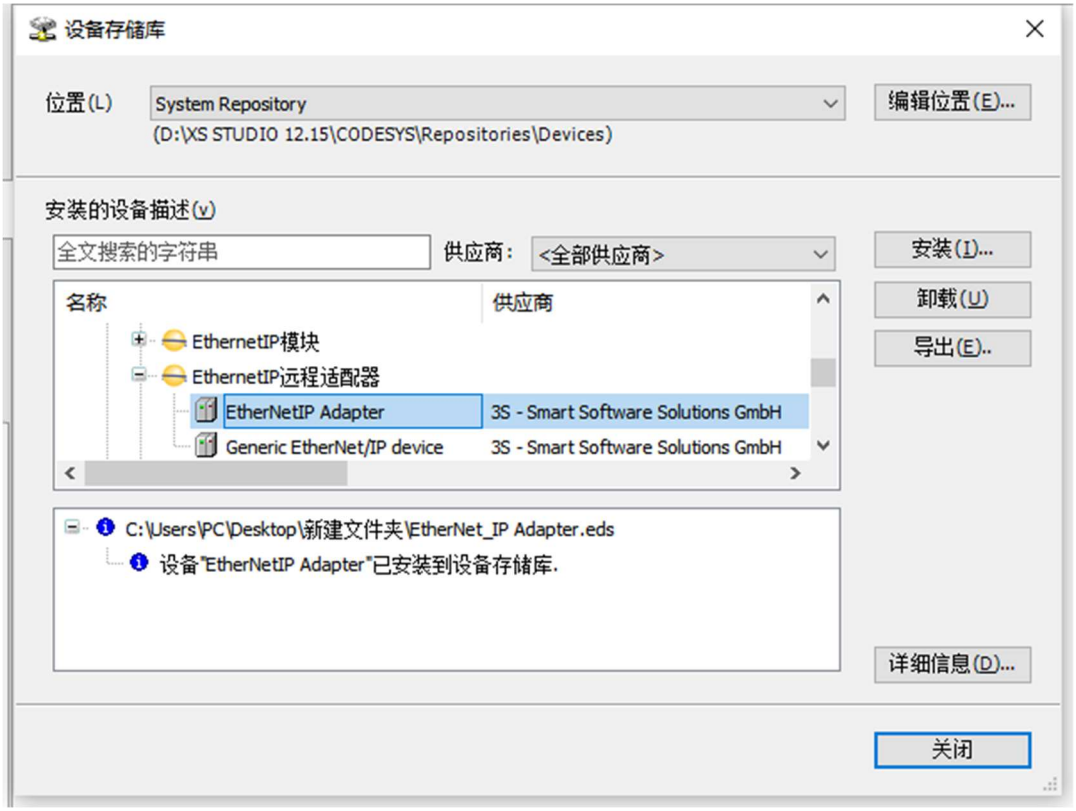


## 3-6-2. EtherNet/IP master example

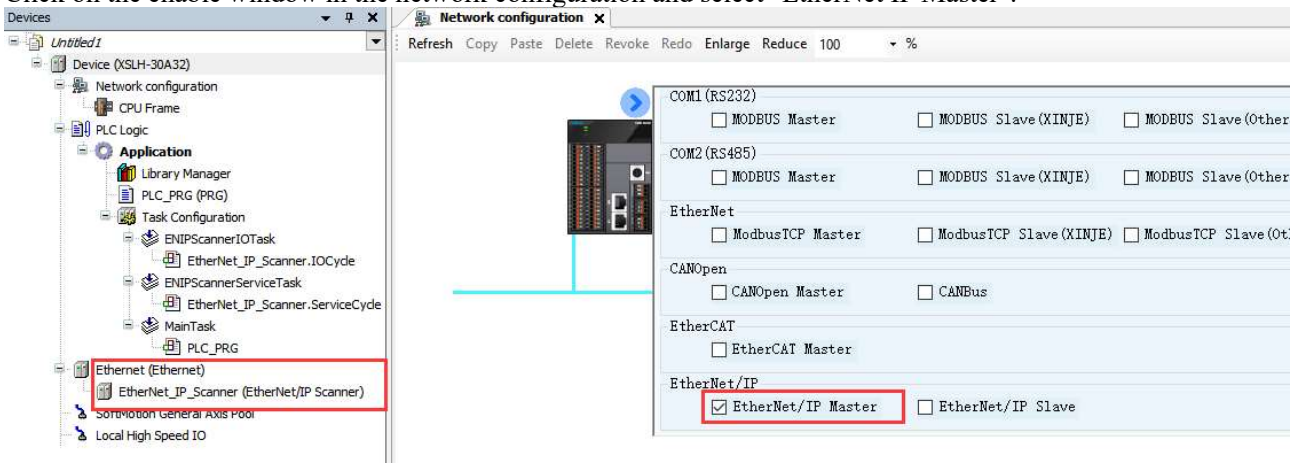
1. Tools → Device repository → Install → Open the EDS file just exported → As shown in the figure, the addition is complete.



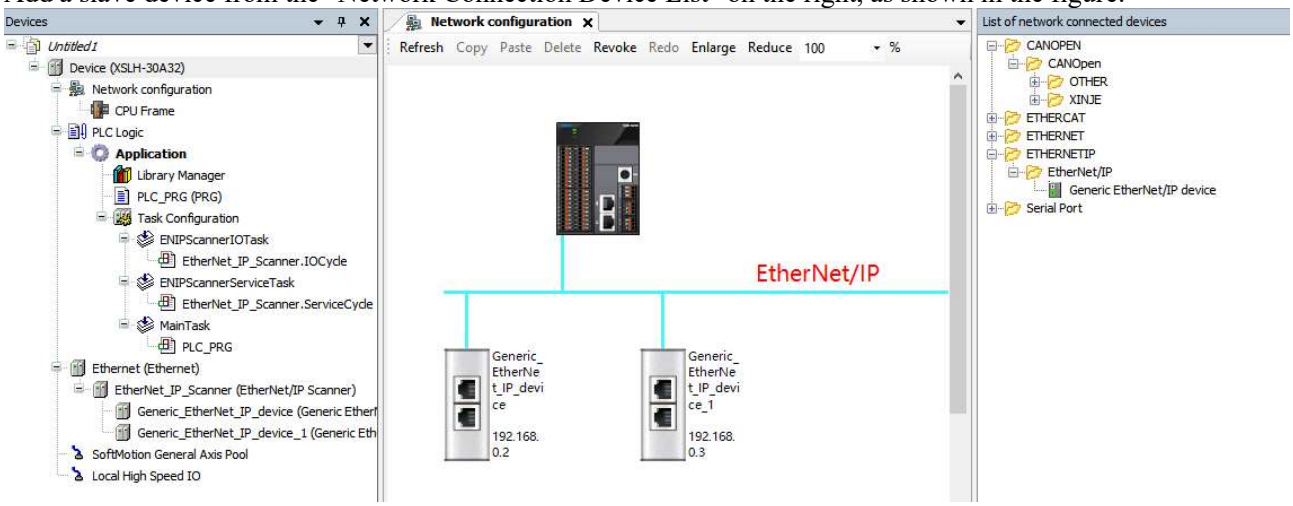





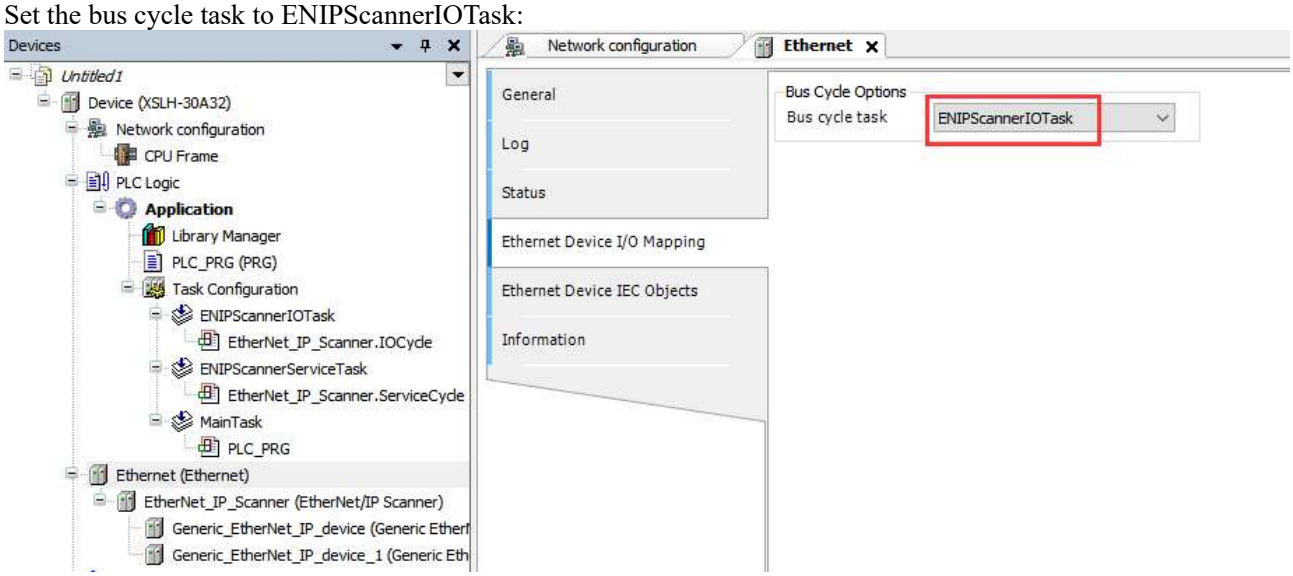

Click on the enable window in the network configuration and select "EtherNet IP Master".



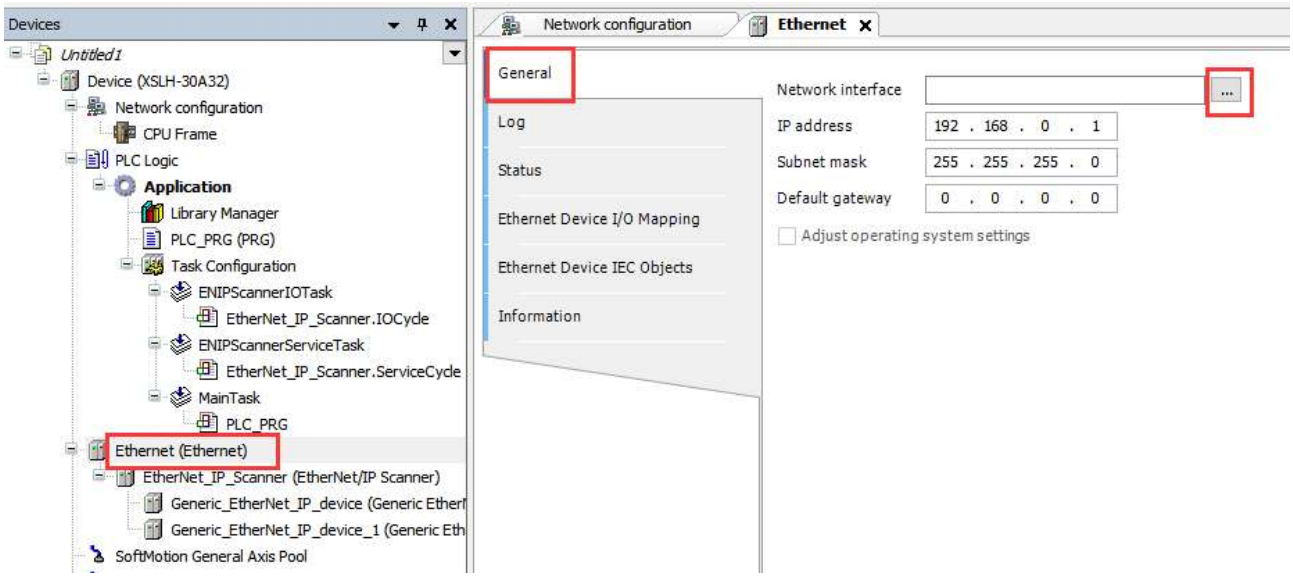

Add a slave device from the "Network Connection Device List" on the right, as shown in the figure:



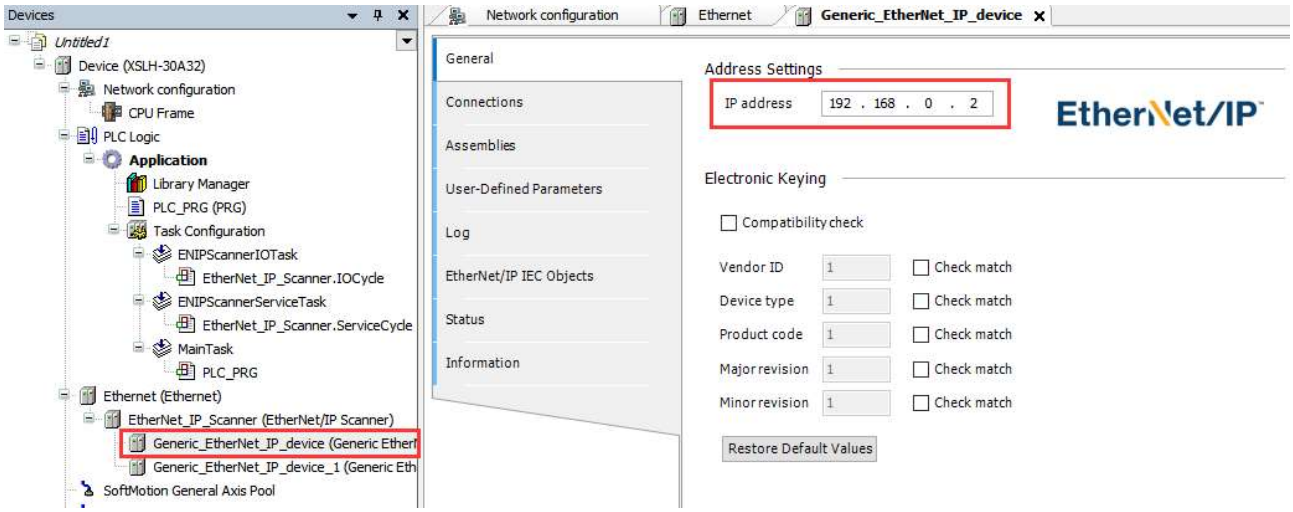
Set the bus cycle task to ENIPScannerIOTask:



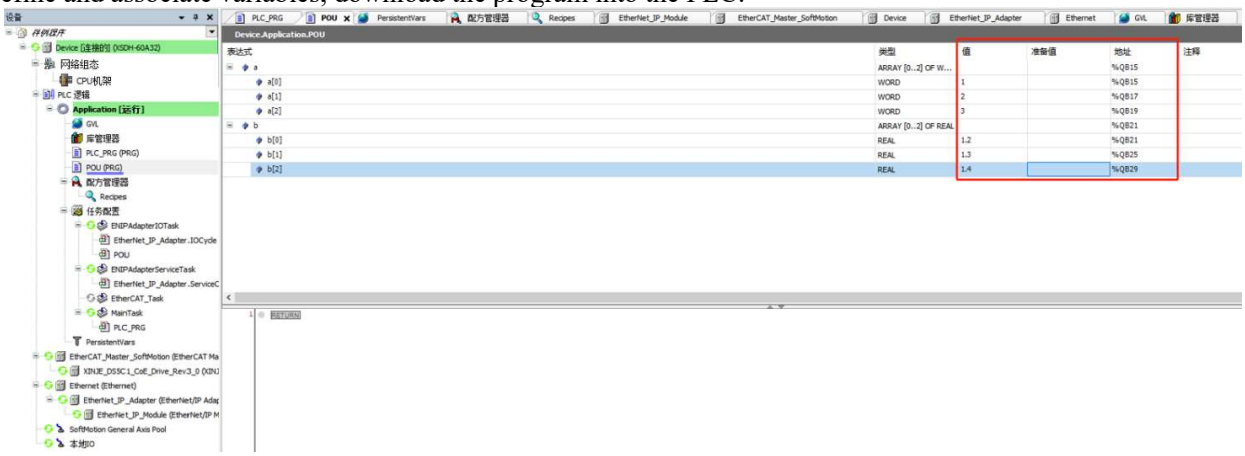

Select the master station Ethernet port:




Set the slave station IP:



Communication testing:
Define and associate variables, download the program into the PLC.



Communication successful.

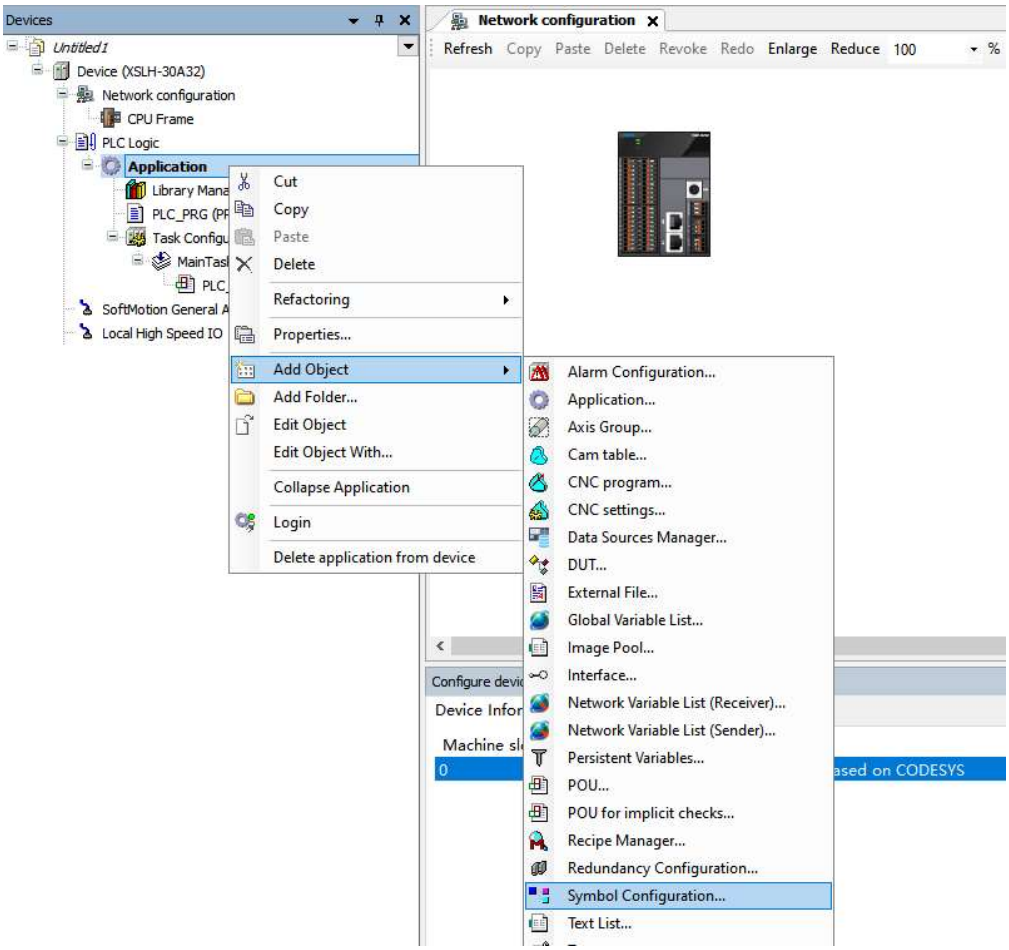
## 3-7. OPC UA communication

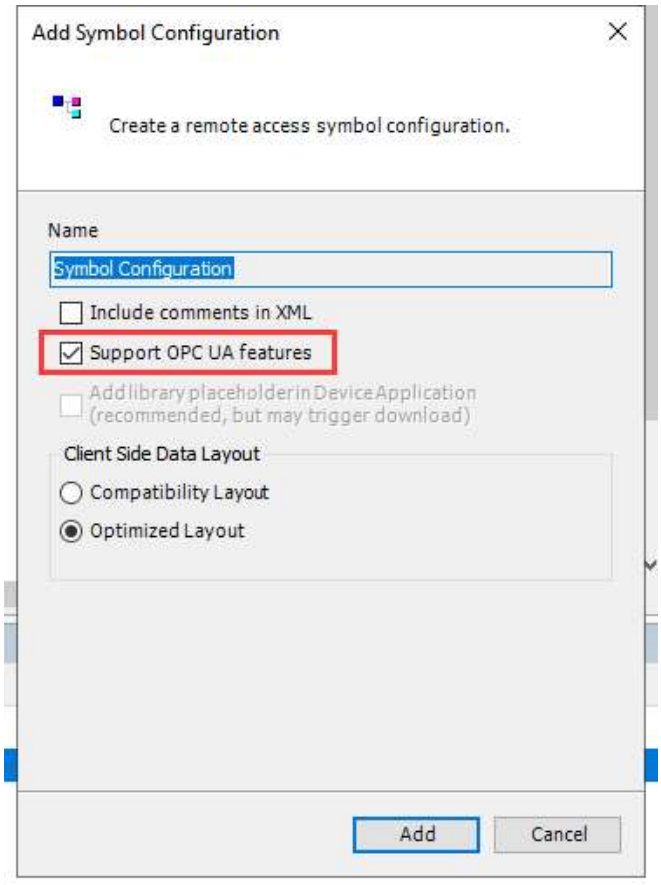
### 3-7-1. Communication overview

OPC UA (OPC Unified Architecture) is a time sensitive network technology based on OPC Unified Architecture, which establishes a time sensitive mechanism to support network interoperability and achieves a breakthrough in the comprehensive integration of information technology (IT) and operational technology (OT) at the physical layer, data link layer, network layer, transport layer, session layer, expression layer, and application layer. This technology is based on the international standards of the International Electrotechnical Commission (IEC) and the Institute of Electrical and Electronics Engineers (IEEE), and can provide standardized modules for the construction of industrial internet network systems. It is a key technology for establishing large bandwidth, high synchronization, and wide compatibility communication from sensors to the cloud.
OPC UA is essentially an abstract framework, a multi-layered architecture where each layer is completely abstracted from its adjacent layers. These layers define various communication protocols on the line and whether messages containing data, data type definitions, and other content can be safely encoded/decoded. By utilizing this core service and data type framework, people can easily add more features on top of it (inheritance).

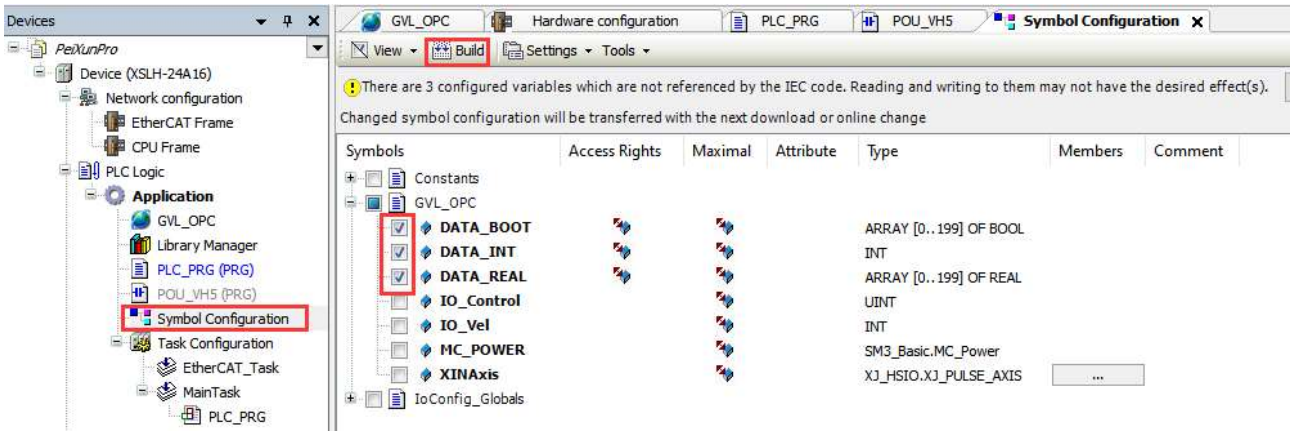
### 3-7-2. Parameter setting

① Right click “Application”, click “Add Object”-“Symbol Configuration..”, select “Support OPC UA features” in the pop-up window to enable the OPC UA function.


Devices
Untitled1
Device (XSLH-30A32)
Network configuration
CPU Frame
PLC Logic
Application
Library Mana
PLC_PRG (PF
Task Configu
MainTasl
PLC_
SoftMotion General A
Local High Speed IO
Network configuration
Refresh
Copy
Paste
Delete
Revoke
Redo
Enlarge
Reduce
100
%
Cut
Copy
Paste
Delete
Refactoring
Properties...
Add Object
Add Folder...
Edit Object
Edit Object With...
Collapse Application
Login
Delete application from device
Alarm Configuration...
Application...
Axis Group...
Cam table...
CNC program...
CNC settings...
Data Sources Manager...
DUT...
External File...
Global Variable List...
Image Pool...
Interface...
Network Variable List (Receiver)...
Network Variable List (Sender)...
Persistent Variables...
POU...
POU for implicit checks...
Recipe Manager...
Redundancy Configuration...
Symbol Configuration...
Text List...
Configure devi
Device Infor
Machine sl
0
ased on CODESYS


Add Symbol Configuration
Create a remote access symbol configuration.
Name
Symbol Configuration
Include comments in XML
Support OPC UA features
Add library placeholder in Device Application
(recommended, but may trigger download)
Client Side Data Layout
Compatibility Layout
Optimized Layout
Add
Cancel

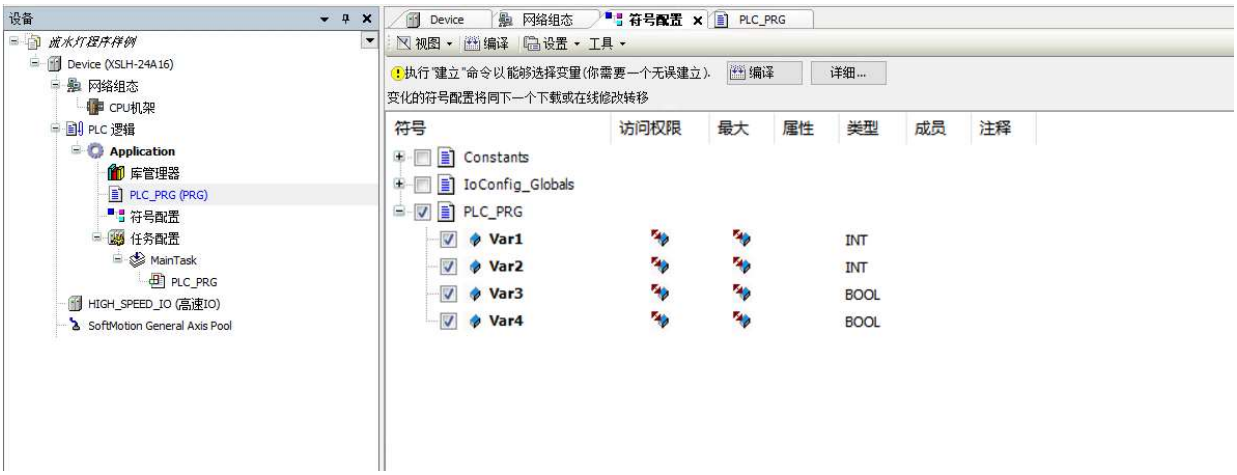
② Double click on "Symbol Configuration", click "Build" in the pop-up interface, and check the parameters that need to be monitored.




## 3-7-3. OPC UA example

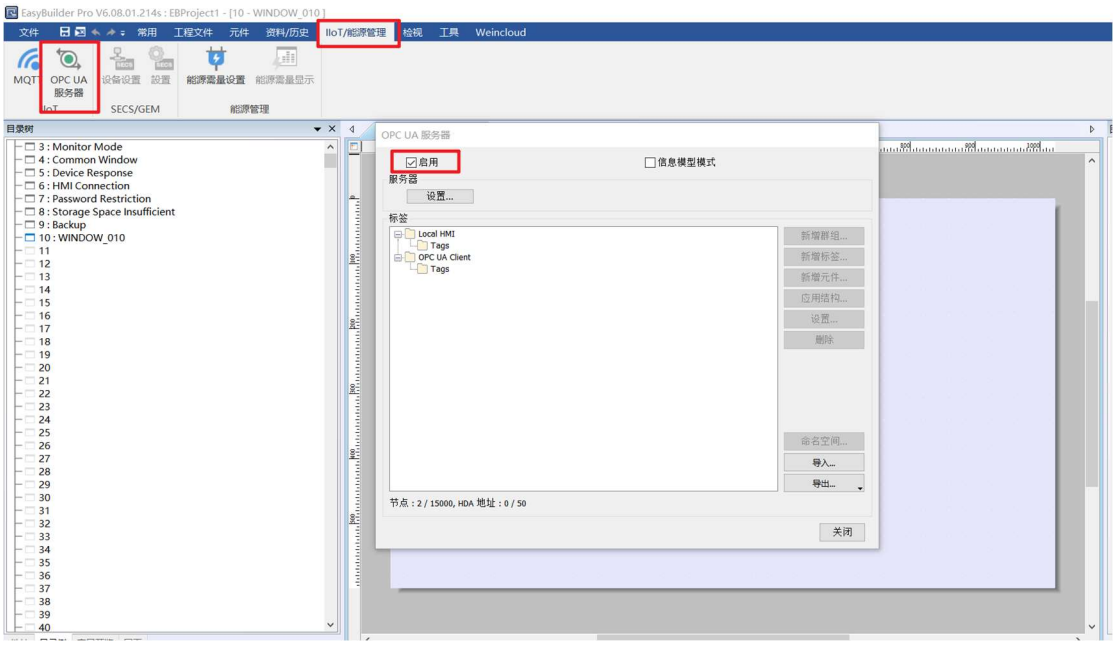
Example 1: Use the Xinje XSLH-24A16 and Weinview HMI (model CMT3105X) for OPC UA communication.
Programming:

(1) Several parameters were written in XSLH-24A16, and login download was checked in the OPC UA interface.

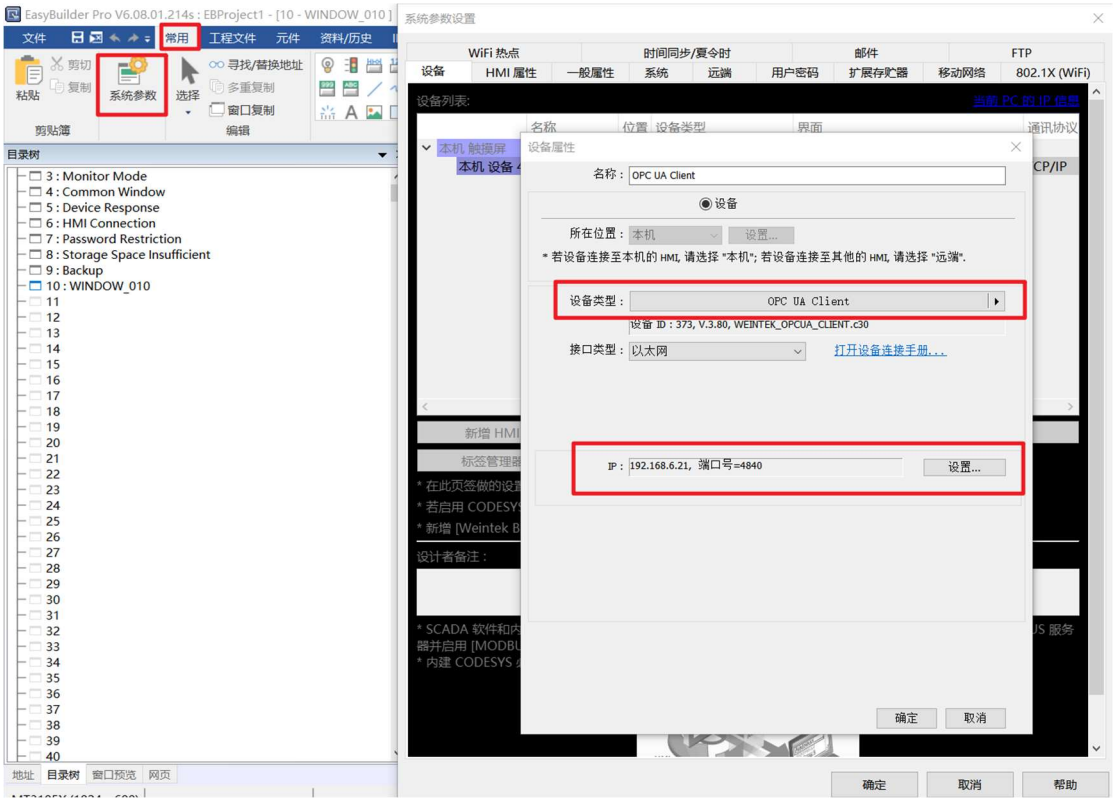



(2) Weinview HMI settings

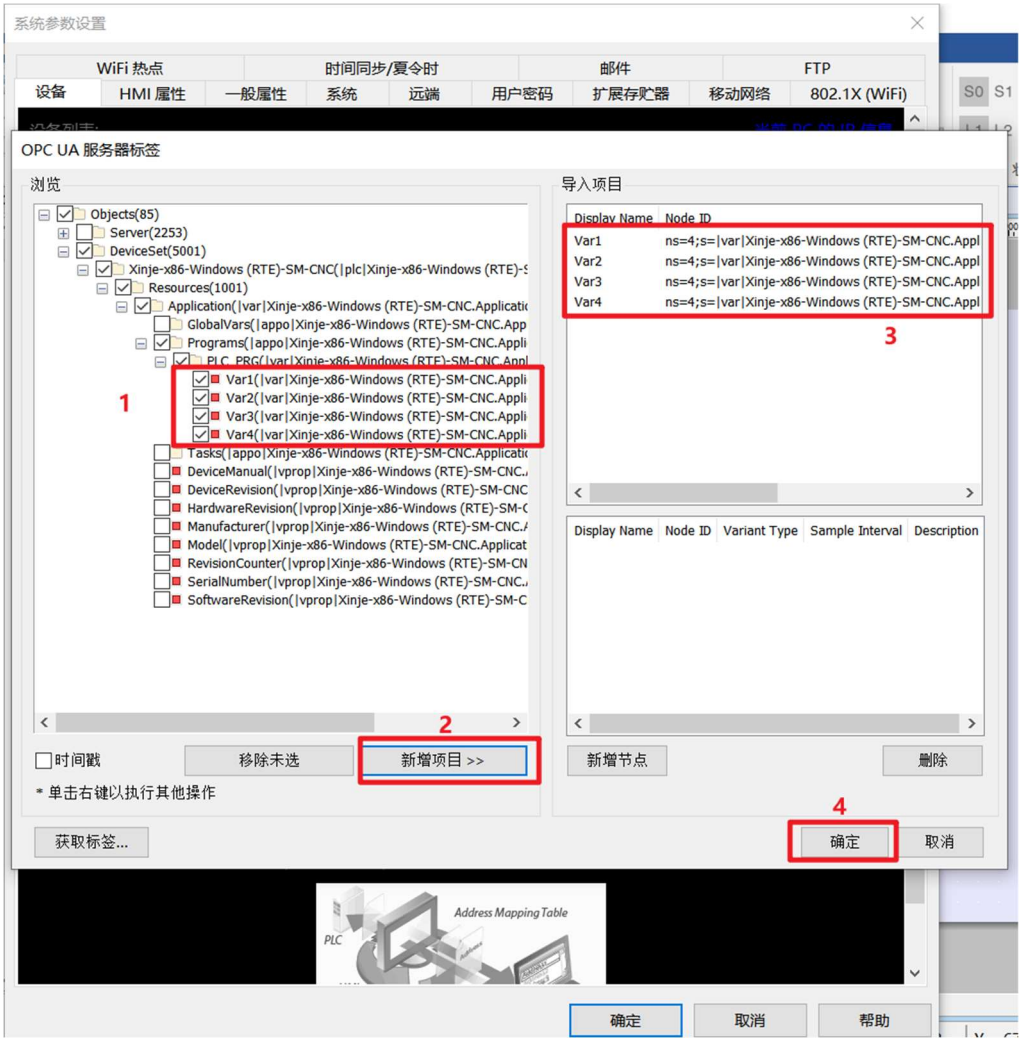
① Select "OPCUA Server" in the "IIOT Energy Management" interface, check "Enable" in the opened "OPCUA Server" interface, and click "OK" in the pop-up interface. After the relevant label pops up, close the interface.



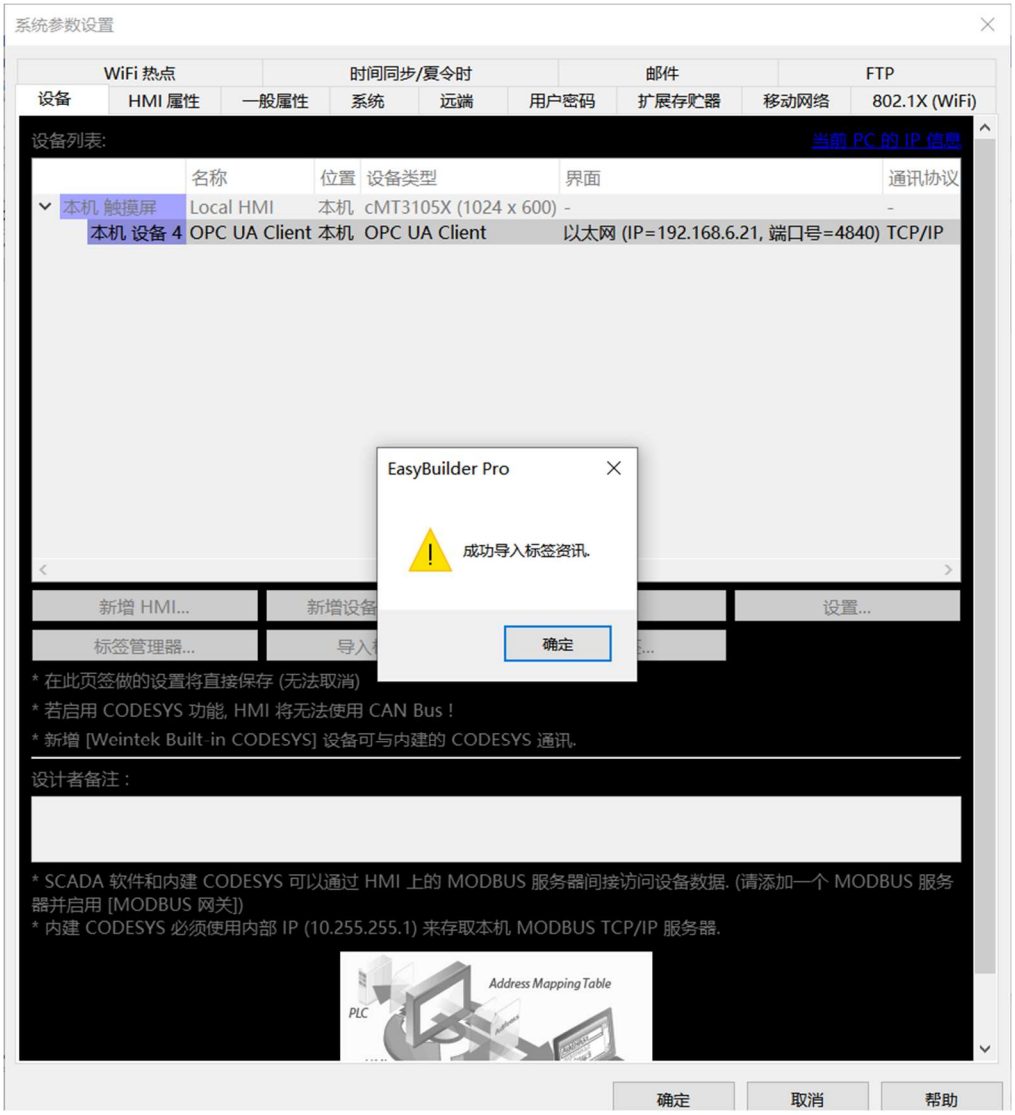

② In the "System Parameter Settings" interface, click "Add Device/Server", select the device type as "OPC UA Client" in the pop-up "Device Properties" interface, and set the IP address to XSLH-24A16. After setting it, click OK.



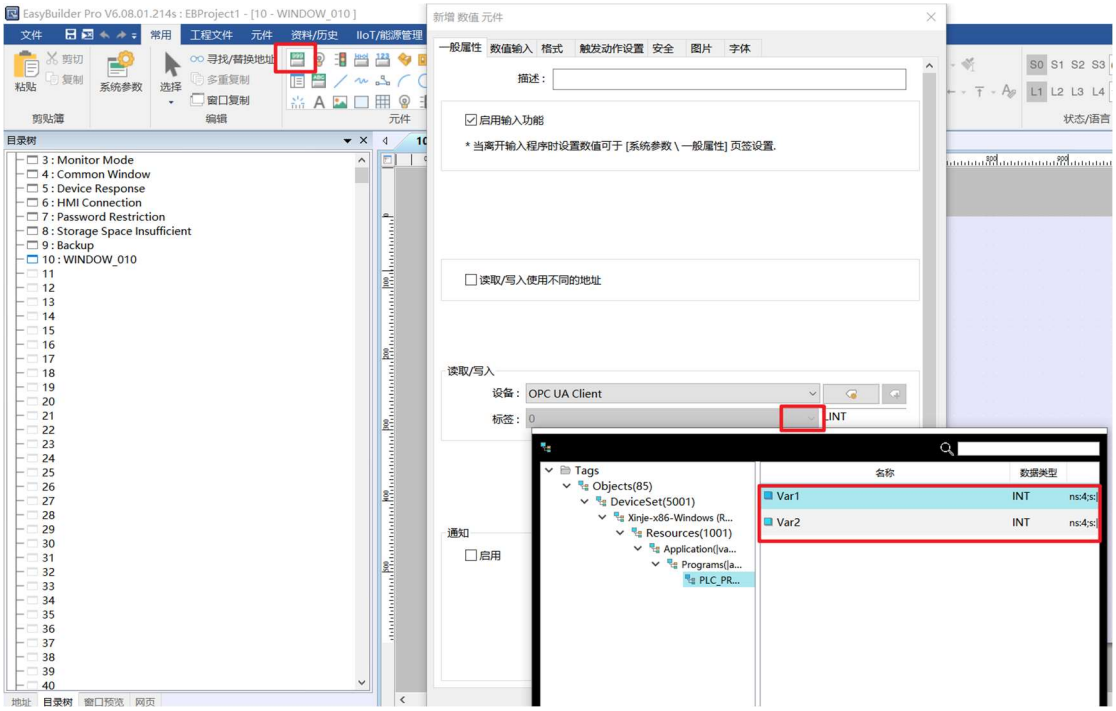

③ Click on "Import Labels" and select "OK" in the pop-up interface. The "OPC UA" server label interface will appear, where you can select the relevant data of PLC

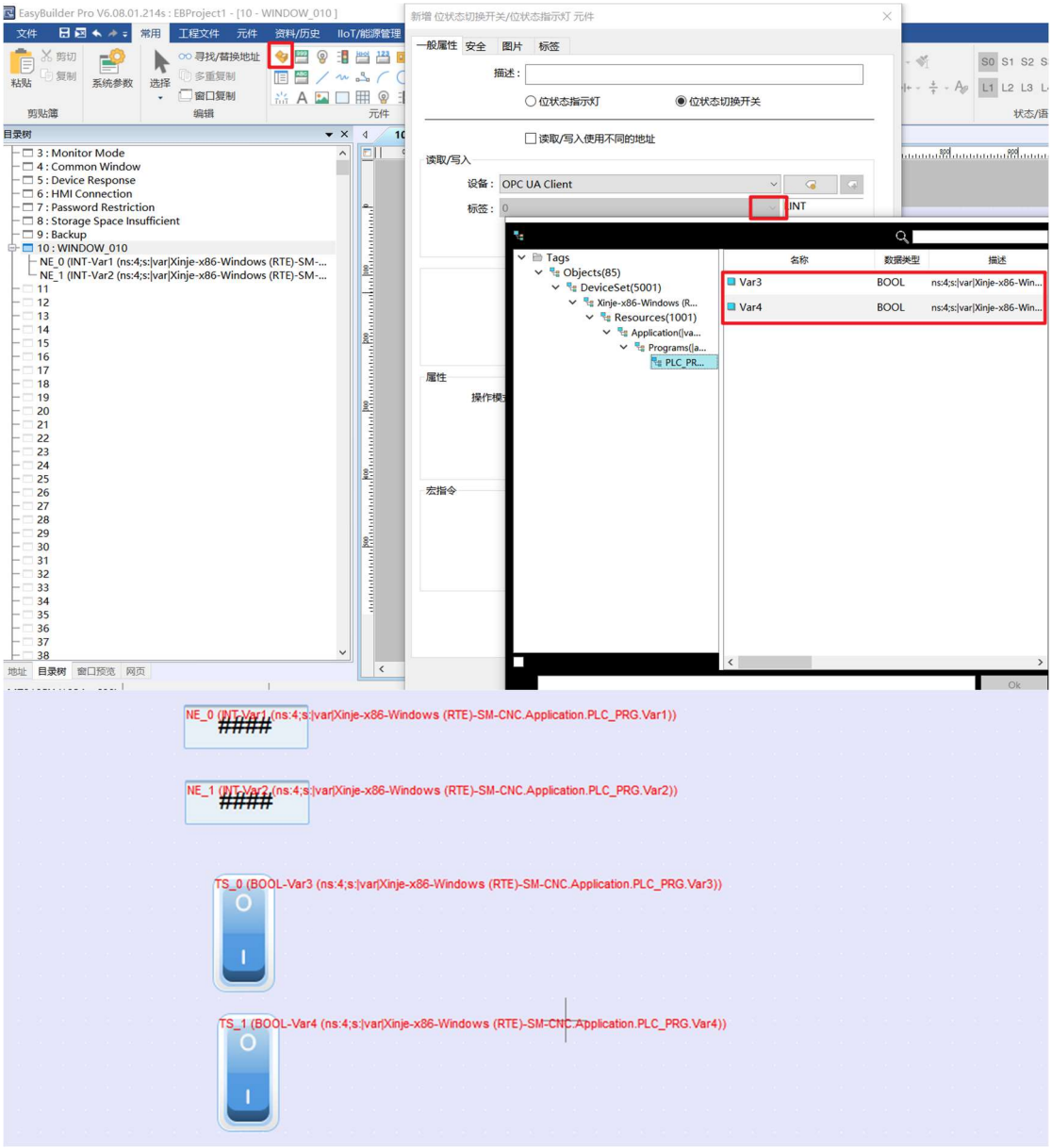



④ Click OK and the message "Successfully imported tag communication" will appear.

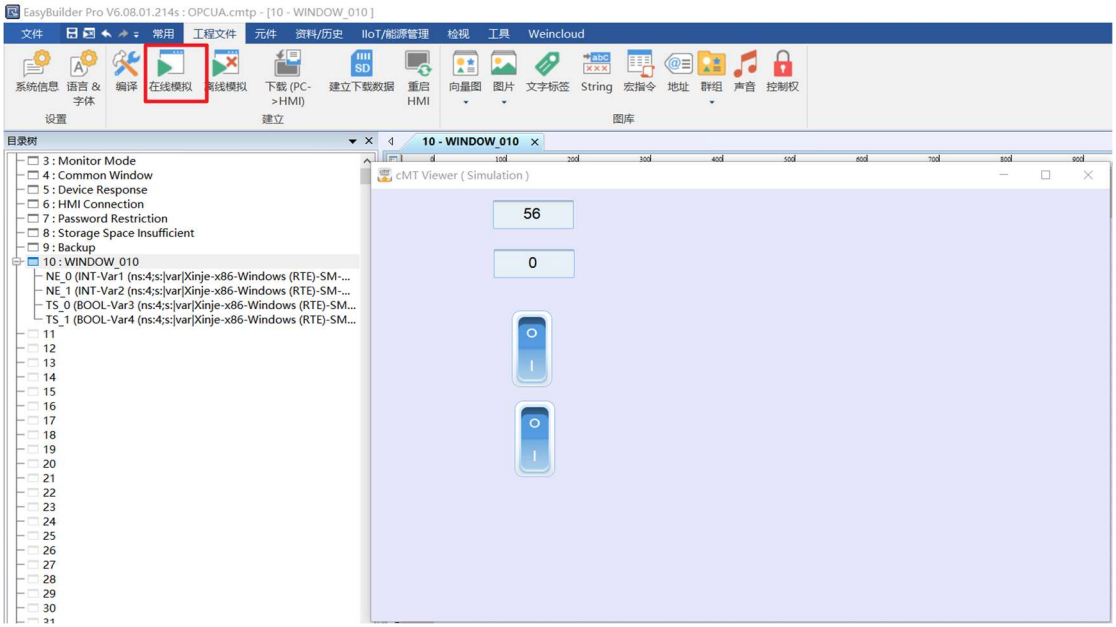


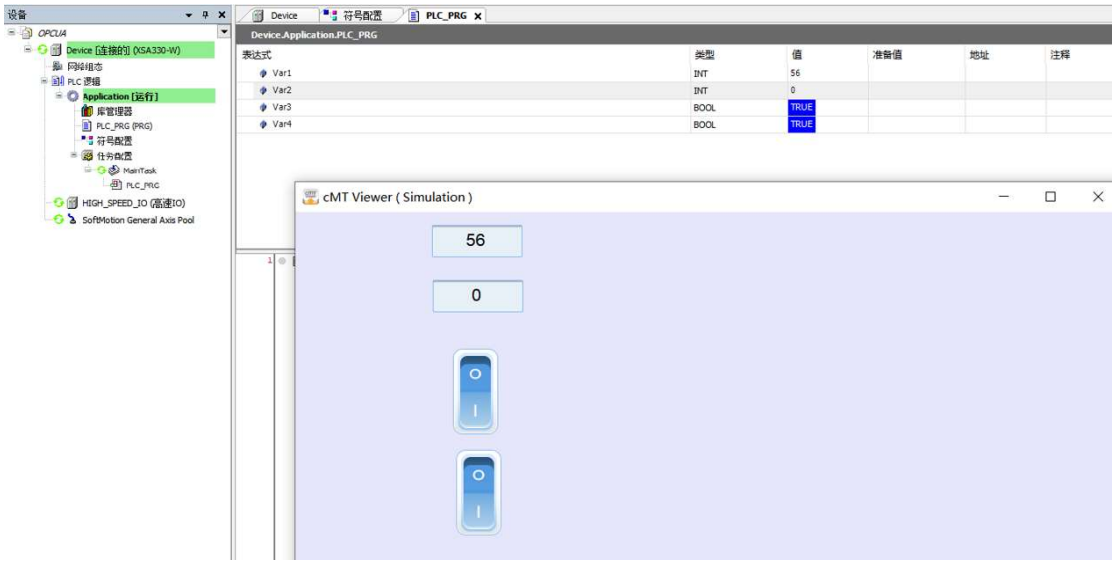

⑤ Select the relevant type of component in the "Components" section, click on the dropdown icon in the "Labels" section of the pop-up interface, and the relevant parameters will appear. Select all parameters in sequence






⑥ Select "Online Simulation" in the "Engineering Files" to achieve communication between the touch screen and PLC.

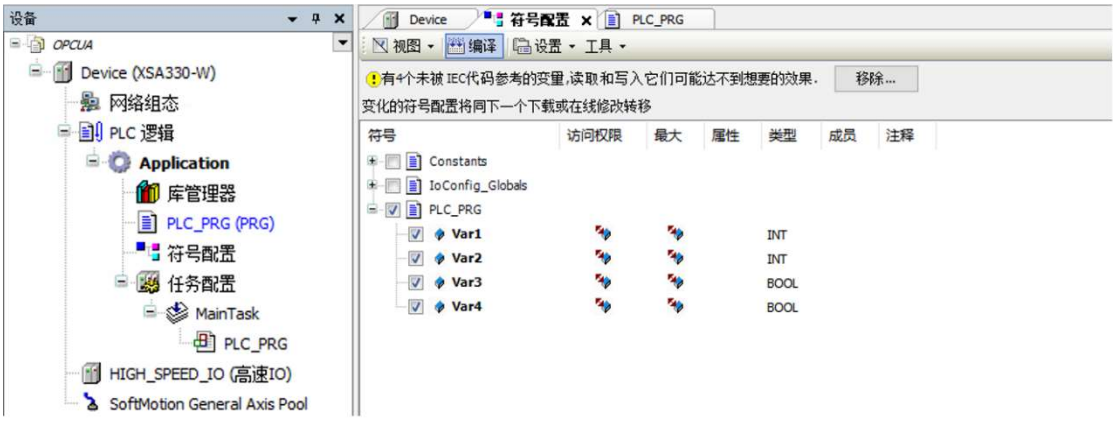





Example 2: Use the Xinje XSA330-W and Weinview HMI (model CMT3105X) for "codesys v3" communication.

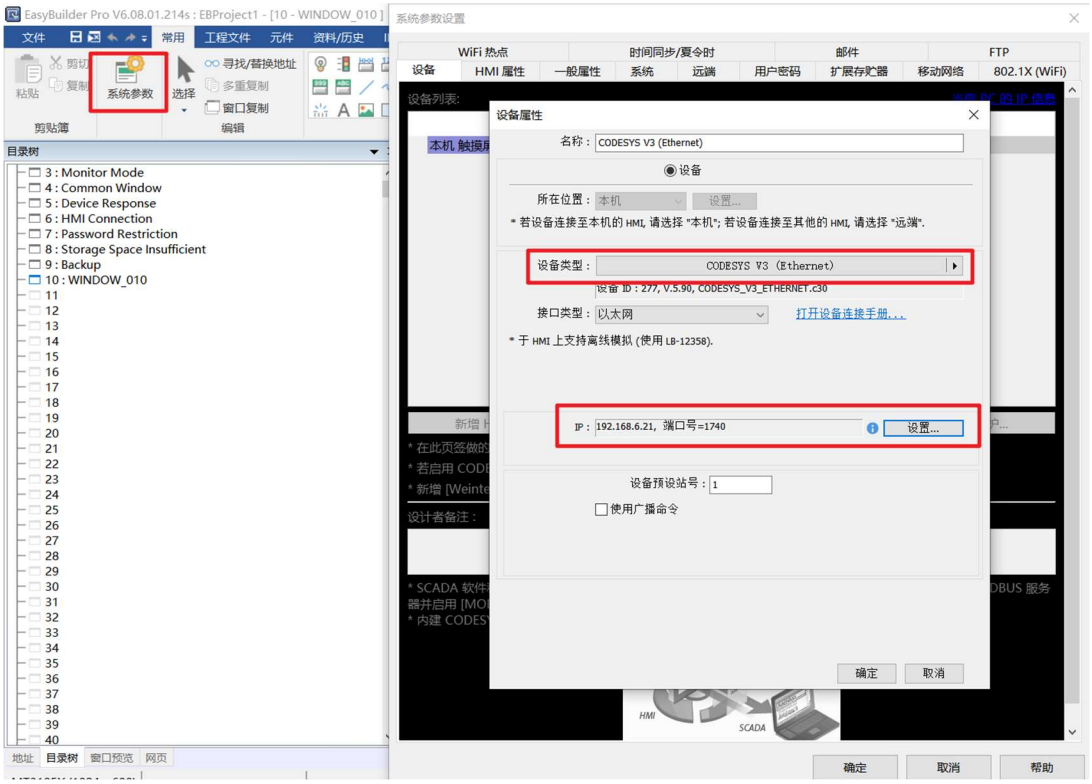
Programming:

(1) Several parameters were written in XSA330-W and checked for login and download in the OPC UA interface.



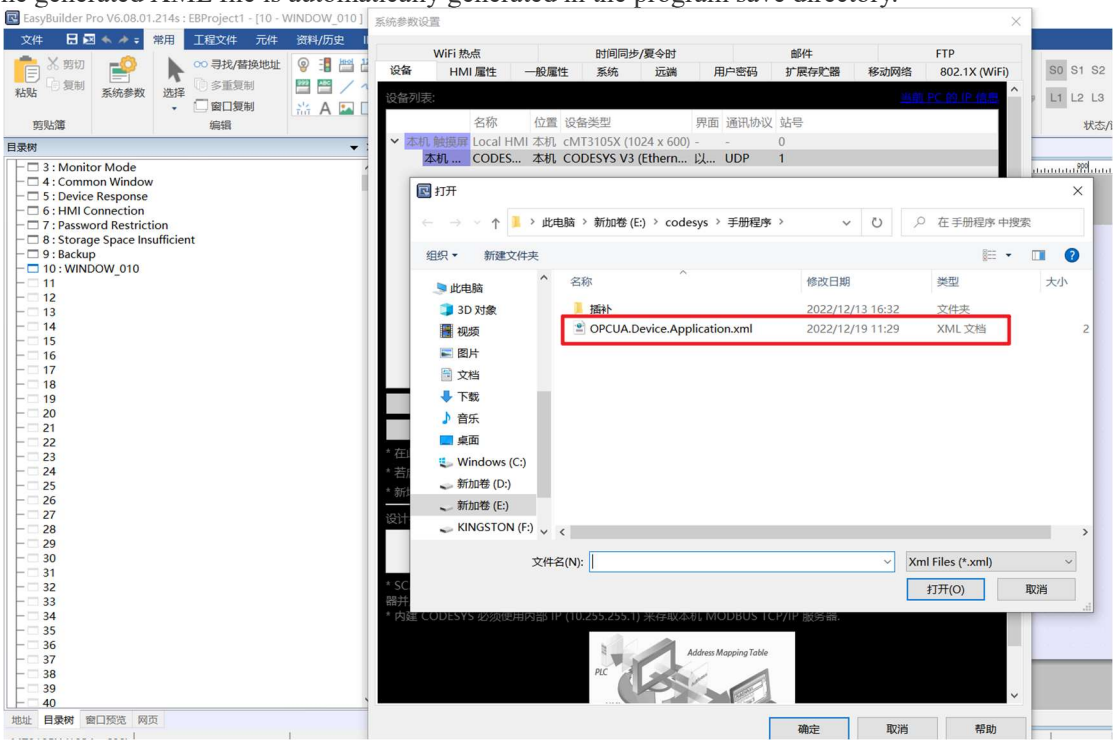

(2) Weinview HMI setting

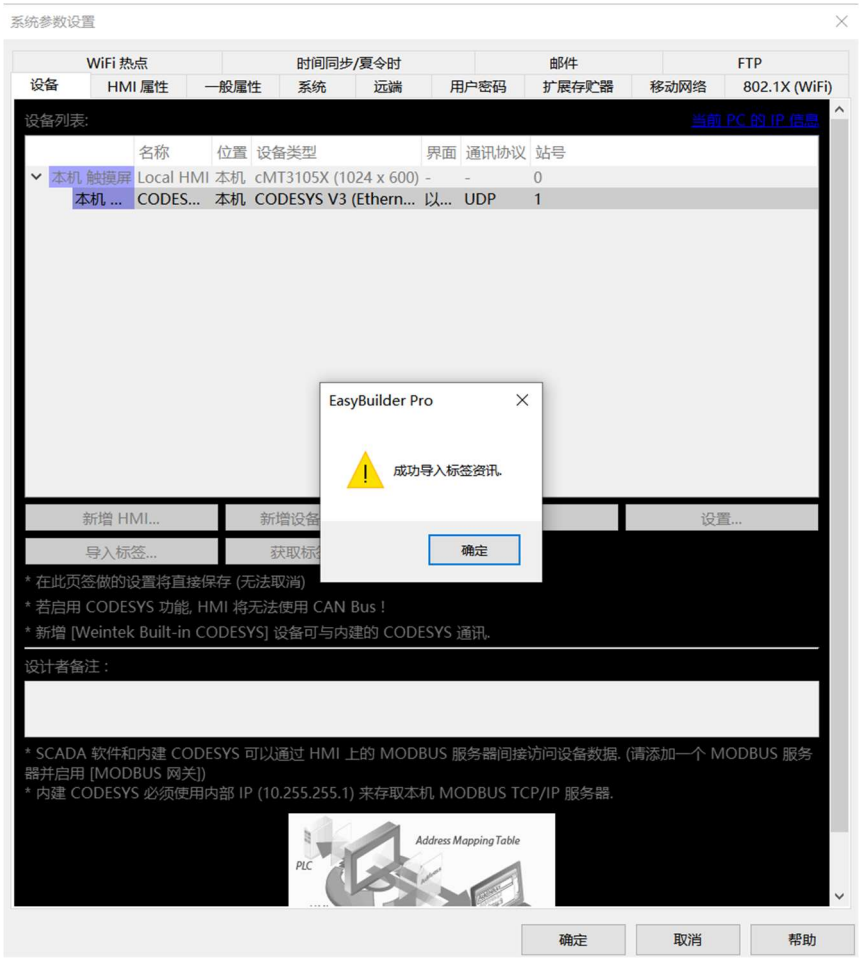
① In the "System Parameter Settings" interface, click "Add Device/Server", select the device type "CODESYS V3" in the "Device Properties" interface that pops up, and set the IP address to XSA330-W. After setting it, click confirm.



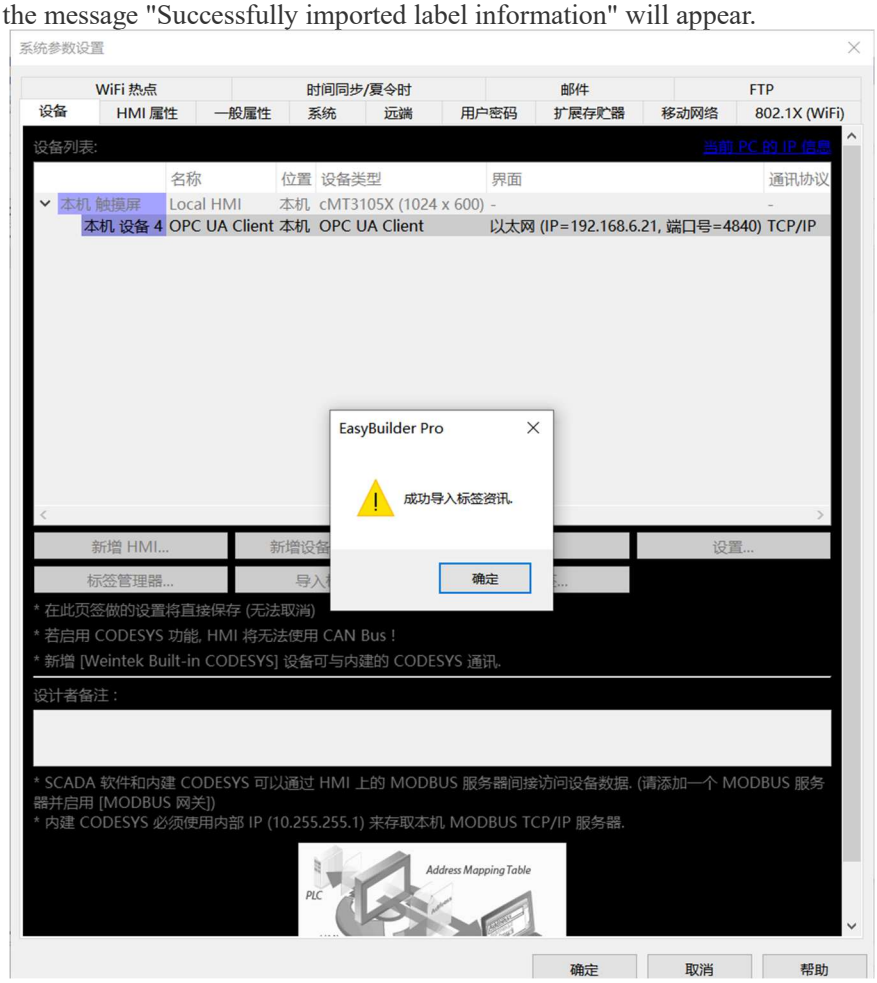

② Click "Import Tags" to import the generated XML file. After successful import, "Successfully imported tag information" will appear.

Note: The generated XML file is automatically generated in the program save directory.

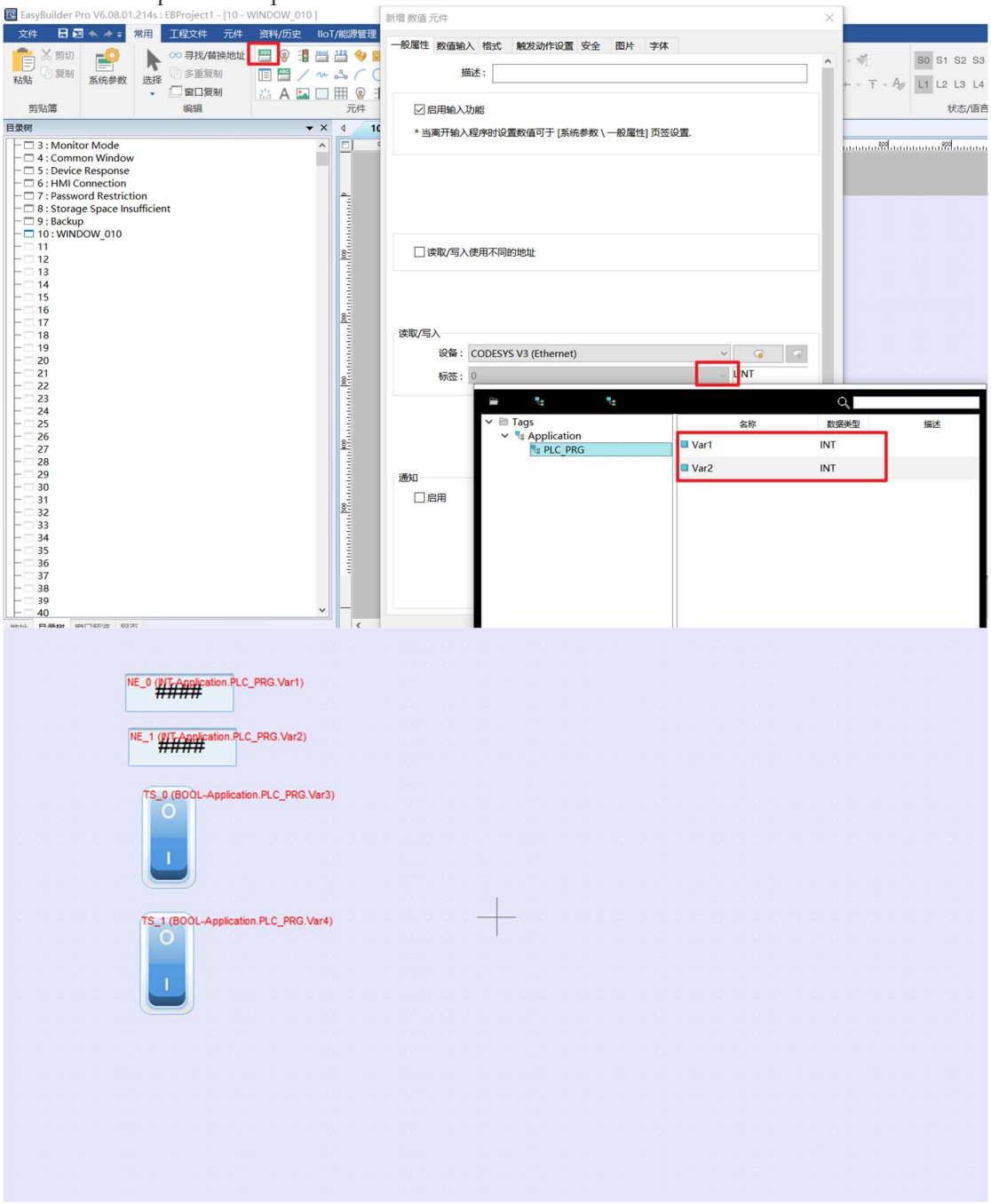




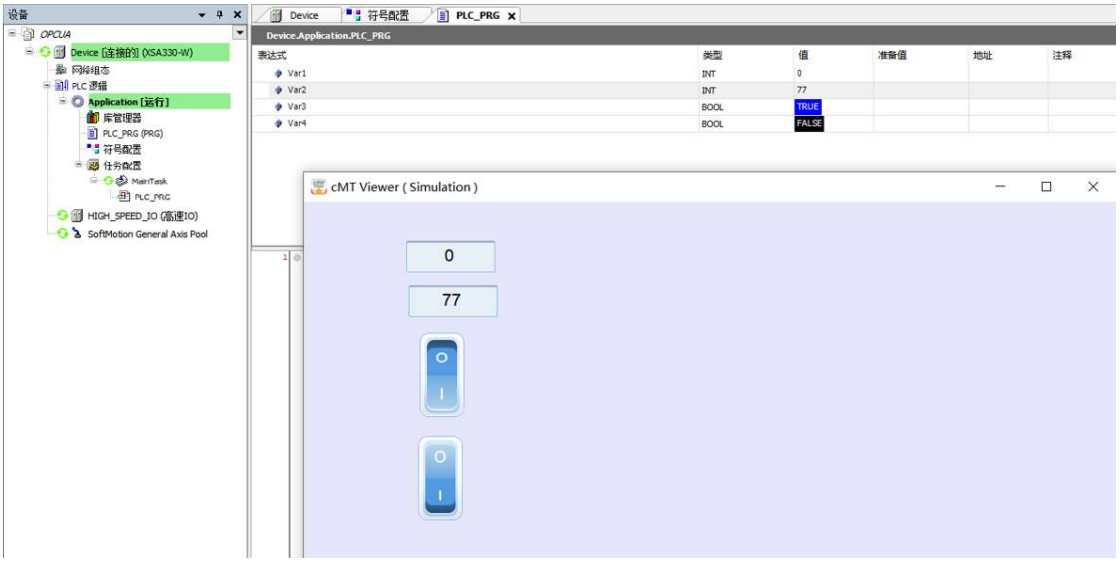

③ Click OK and the message "Successfully imported label information" will appear.



④ Select the relevant type of component in the "Components" section, click on the dropdown icon in the "Labels" section of the pop-up interface, and the relevant parameters will appear. Select all parameters in sequence. The same steps as Example 1.



⑤ Select "Online Simulation" in the "Engineering Files" to achieve communication between the touch screen and PLC.



# 4. EtherCAT configuration

## 4-1. EtherCAT overview

### 4-1-1. Overview

EtherCAT is the abbreviation for Ethernet for Control Automation Technology. It is an open network communication system developed by Beckhoff Automation GmbH for real-time Ethernet between master and slave stations, managed by ETG (EtherCAT Technology Group).

### 4-1-2. System composition

The connection form of EtherCAT is a network system that connects the main station (FA controller) and multiple slave stations in a linear manner.
The number of nodes that a slave can connect to depends on the processing or communication cycle of the master station, the number of bytes transmitted, etc.

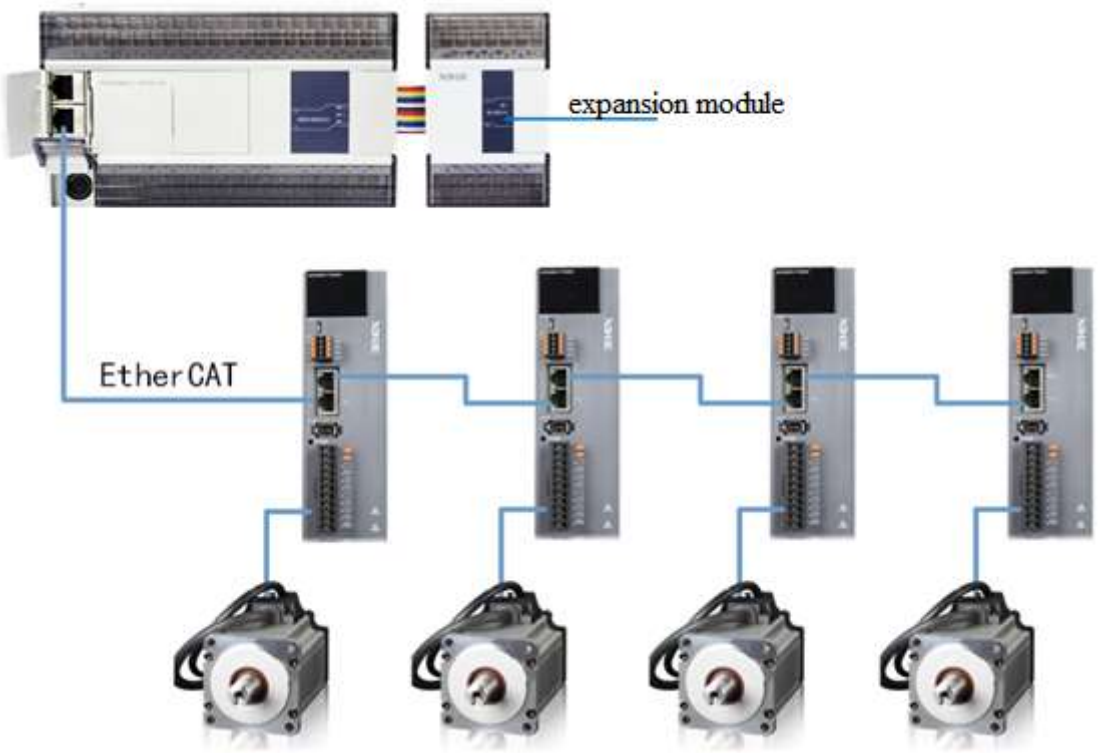
### 4-1-3. Communication specification

<table>
<tr><th>Item</th><th>Specification</th></tr>
<tr><td>Physical layer</td><td>100BASE-TX (IEEE802.3)</td></tr>
<tr><td>Baud rate</td><td>100[Mbps] (full duplex)</td></tr>
<tr><td>Topology</td><td>Line</td></tr>
<tr><td>Connecting cables</td><td>JC-CA twisted pair (shielded twisted pair)</td></tr>
<tr><td>Cable length</td><td>The longest distance between nodes 100m</td></tr>
<tr><td>Communication port</td><td>2 Port (RJ45)</td></tr>
<tr><td>EtherCAT Indicators (LED)</td><td>[Run] RUN Indicator<br>[L/A IN] Port0 Link/Activity Indicator (Green)<br>[L/A OUT] Port1 Link/Activity Indicator (Green)</td></tr>
<tr><td>Station Alias (ID)</td><td>Setting range: 0~65535<br>Setting address: 2700h</td></tr>
<tr><td>Explicit Device ID</td><td>Not supported</td></tr>
<tr><td>Mail protocol</td><td>COE (CANopen Over EtherCAT)</td></tr>
<tr><td>SyncManager</td><td>4</td></tr>
<tr><td>FMMU</td><td>3</td></tr>
<tr><td>Modes of operation</td><td>
<table>
<tr><th></th><th colspan="2">Modes of operation</th></tr>
<tr><td rowspan="3">position</td><td>csp</td><td>Cyclic synchronous position mode</td></tr>
<tr><td>PP</td><td>Profile position mode</td></tr>
<tr><td>hm</td><td>Homing mode</td></tr>
<tr><td rowspan="2">Velocity</td><td>csv</td><td>Cyclic synchronous velocity mode</td></tr>
<tr><td>pv</td><td>Profile velocity mode</td></tr>
<tr><td rowspan="2">Torque</td><td>cst</td><td>Cyclic synchronous torque mode</td></tr>
<tr><td>tq</td><td>Torque profile mode</td></tr>
</table>
</td></tr>
<tr><td>Synchronous mode</td><td>DC (SYNCO event synchronism)</td></tr>
</table>

| Item | Specification |
|---|---|
| | SM (SM event synchronization) |
| Cyclic time (DC communication period) | 500, 1000, 2000, 4000[μs] |
| Communication object | SDO[Service Data Object], PDO[Process data object] |
| Email communication interval in PreOP mode | 1ms |
| Email | SDO requests and SDO information |

## 4-1-4. EtherCAT communication connection

The wiring of the EtherCAT motion control system is very simple. Thanks to EtherCAT, the star topology of Ethernet can be replaced by a simple linear structure. Taking the Xinje DS5C series servo as an example, due to the fact that EtherCAT does not require a hub or switch, the DS5C series servo comes with an EtherCAT communication network port, which greatly reduces the amount of cables and cable trays used, and greatly reduces the workload of wiring design and joint calibration, making it easier to save installation costs.
It is recommended to use linear connection method for EtherCAT bus wiring. The wiring method of XSDH series is shown in the following figure:

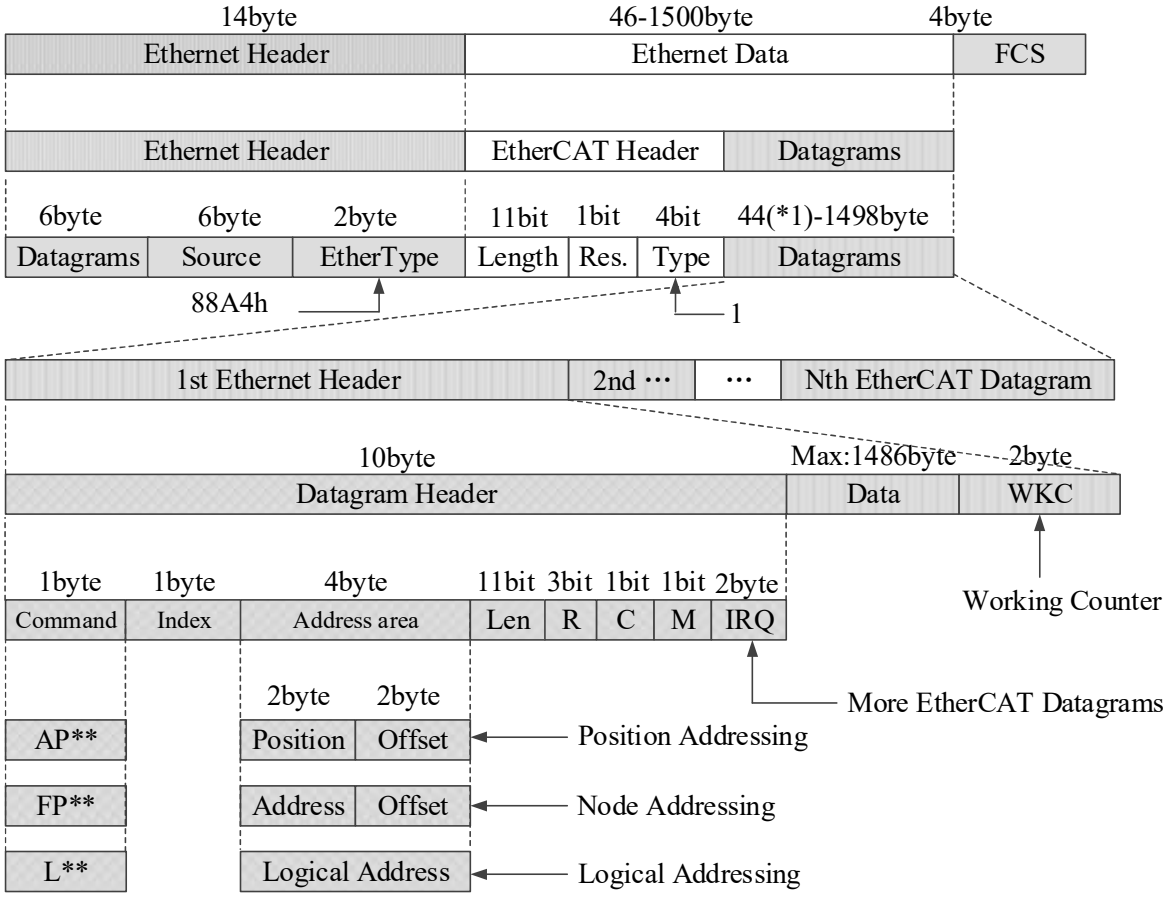



The entire bus network adopts a linear structure, with the XSDH series controller as the master station and the Xinje DS5C1 series bus controlled servo as the slave station. The XS3 series PLC has two Ethernet/IP ports, the above port are used to connect the XS Studio upper computer; The following network port is an EtherCAT connection port, used to connect the Xinje DS5C1 series servo to achieve EtherCAT communication. The two communication network ports of the Xinje DS5C1 series servo driver need to follow the principle of "bottom in and top out".

# 4-2. EtherCAT communication specification

## 4-2-1. EtherCAT frame structure

EtherCAT is an industrial communication protocol based on Ethernet that can be controlled in real-time. It only expands the IEEE 802.3 Ethernet specification without making any changes to the basic structure, so it can transmit data within standard Ethernet frames.



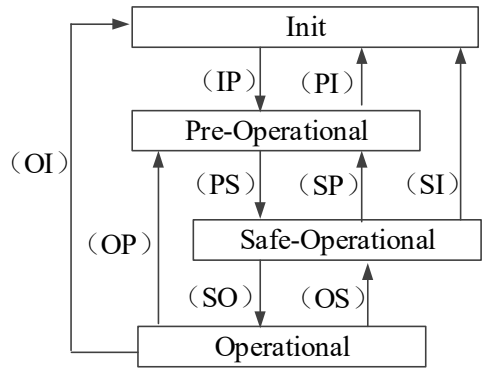

***1:** Ethernet frames are shorter than 64 bytes, adding 1 to 32 bytes.
(Ethernet Header + Ethernet Data + FCS)

Because the EtherType of the Ethernet header is 88A4h, the subsequent Ethernet data will be processed as EtherCAT frames. EtherCAT frames are defined and parsed using a certain protocol, as long as both the master and slave stations comply with this protocol, data communication can be achieved. The commonly used protocols include CANopen Over EtherCAT (CoE) and Sercos Over EtherCAT (SoE).

## 4-2-2. State machine ESM

The EtherCAT State Machine (ESM) is responsible for coordinating the state relationship between the master and slave application programs during initialization and runtime.
The state change request is executed by the master station, which makes a control request to the application layer service. The latter generates an application layer control event in the slave station, and the slave station responds to the application layer control service through the local application layer state write service after the state change request is successful or failed. If the state change fails, the slave station will remain in the state and display an error flag.

**The following diagram shows the state transition of ESM:**




※The IP in the state transition diagram is an abbreviation for state transition
(IP): Init→Pre-Operational
(PS): Pre-Operational→Safe-Operational

Init: Initialization status;
Pre-Operational: Preoperational status;
Safe-Operational: Safe operation status;
Operational: operation status;

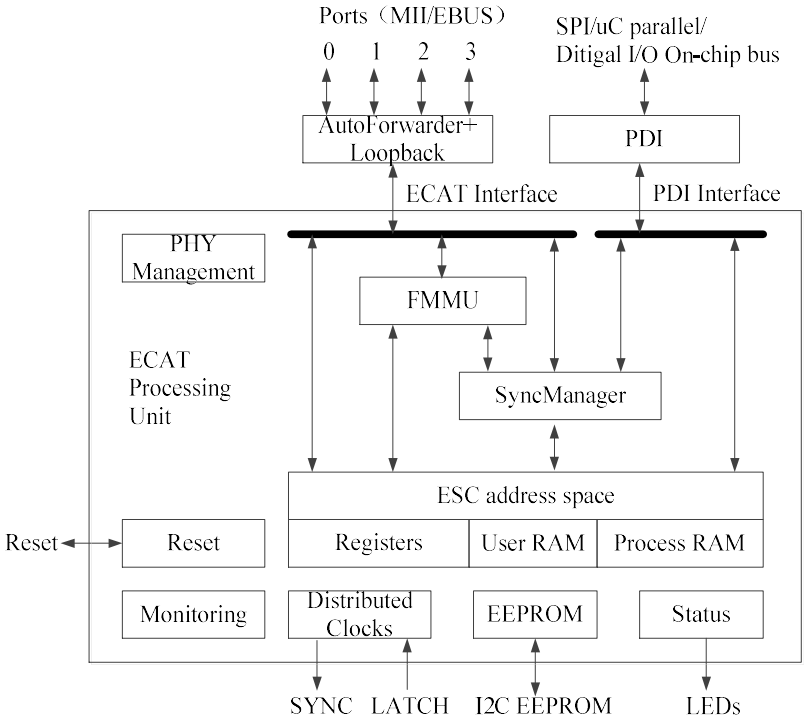
| Slave station status | Actions in each state | Communication action | | |
|---|---|---|---|---|
| | | SDO (email) Sending and receiving messages | PDO Sending message | PDO Receiving message |
| Init | Communication initialization, SDO, PDO unable to receive and send messages status | - | - | - |
| Pre-Operational (PreOP) | Only SDO sending and receiving message status | Yes | - | - |
| Safe-Operational (SafeOP) | Only SDO receiving and sending message, PDO sending message status | Yes | Yes | - |
| Operational (OP) | SDO receiving and sending message, PDO receiving and sending message all feasible state | Yes | Yes | Yes |

Note: Access from the main station to the ESC register is independent of the above table and can be accessed at any time.
PDO (Process Data Object) is a process data object used to transmit periodic communication data.
SDO (Service Data Object) is a service data object used to transmit non periodic communication data.
Instructions or interface operations during ESM state switching may cause communication anomalies and errors.

## 4-2-3. Slave station controller ESC

### 4-2-3-1. Principle overview

ESC refers to the EtherCAT Slave Controller. The communication process is entirely handled by ESC, which has four data transmission and reception ports, each with a TX and RX. Each port can send and receive Ethernet data frames, and the data flow in ESC is fixed: Port 0->Port 3->Port 1->Port 2->Port 0 is transmitted in sequence. If ESC detects that a port does not have an external PHY, it automatically closes the port and forwards it to the next port through an internal loop.




4-2-3-2. Address space

The DS5C1 series holds a physical address space of 8K bytes.

The initial 4Kbyte (0000h~0FFFh) was used as a register space, while the additional 4Kbyte (1000h~1FFFh) was used for process data PDO in the RAM domain. Please refer to the data table of IP (ET1810/ET1811/ET1812) for detailed information on registers.

| ESC Register Byte Address | Length (Byte) | Explanation | Initial value |
|---|---|---|---|
| ESC Information (Slave controller information) | | | |
| 0000h | 1 | Type | 04h |
| 0001h | 1 | Revision | 02h |
| 0002h~0003h | 2 | Build | 0040h |
| 0004h | 1 | FMMUs supported | 03h |
| 0005h | 1 | SyncManagers supported | 04h |
| 0006h | 1 | RAM Size | 08h |
| 0007h | 1 | Port Descriptor | 0Fh |
| 0008h~0009h | 2 | ESC Features supported | 0184h |
| Station Address | | | |
| 0010h~0011h | 2 | Configured Station Address | - |
| 0012h~0013h | 2 | Configured Station Alias | - |
| … | | | |
| Data Link Layer | | | |
| … | | | |
| 0100h~0103h | 4 | ESC DL Control | - |
| … | | | |
| 0110h~0111h | 2 | ESC DL Status | - |

| ESC Register Byte Address | Length (Byte) | Explanation | Initial value |
|---|---|---|---|
| Application Layer | | | |
| 0120h~0121h | 2 | AL Control | - |
| 0130h~0131h | 2 | AL Status | - |
| 0134h~0135h | 2 | AL Status Code | - |
| … | | | |
| PDI | | | |
| 0140h | 1 | PDI Control | 08h |
| 0141h | 1 | ESC Configuration | 0Ch |
| 0150h | 1 | PDI Configuration | - |
| 0151h | 1 | SYNC/LATCH PDI Configuration | 66h |
| 0152h~153h | 2 | Extend PDI Configuration | - |
| … | | | |
| Watchdogs | | | |
| 0400h~0401h | 2 | Watchdog Divider | - |
| 0410h~0411h | 2 | Watchdog Time PDI | - |
| 0420h~0421h | 2 | Watchdog Time Process Data | - |
| 0440h~0441h | 2 | Watchdog Status Process Data | - |
| 0442h | 1 | Watchdog Counter Process Data | - |
| 0443h | 1 | Watchdog Counter PDI | - |
| … | | | |
| FMMU | | | |
| 0600h~062Fh | 3x16 | FMMUs[2:0] | - |
| +0h~3h | 4 | Logical Start Address | - |
| +4h~5h | 2 | Length | - |
| +6h | 1 | Logical Start bit | - |
| +7h | 1 | Logical Stop bit | - |
| +8h~9h | 2 | Physical Start Address | - |
| +Ah | 1 | Physical Start bit | - |
| +Bh | 1 | Type | - |
| +Ch | 1 | Activate | - |
| +Dh~Fh | 3 | Reserved | - |
| … | | | |
| Distributed Clocks（DC）-SYNC Out Unit | | | |
| 0981h | 1 | Activation | - |
| … | | | |
| 0984h | 1 | Activation Status | - |
| 098Eh | 1 | SYNCO Status | - |
| … | | | |
| 0990h~0993h | 4 | Start Time Cyclic Operation/Next SYNC0 Pulse | - |
| … | | | |

| ESC Register Byte Address | Length (Byte) | Explanation | Initial value |
|---|---|---|---|
| 09A0h~09A3h | 4 | SYNC0 Cycle Time | - |
| … | | | |

### 4-2-4. SII area

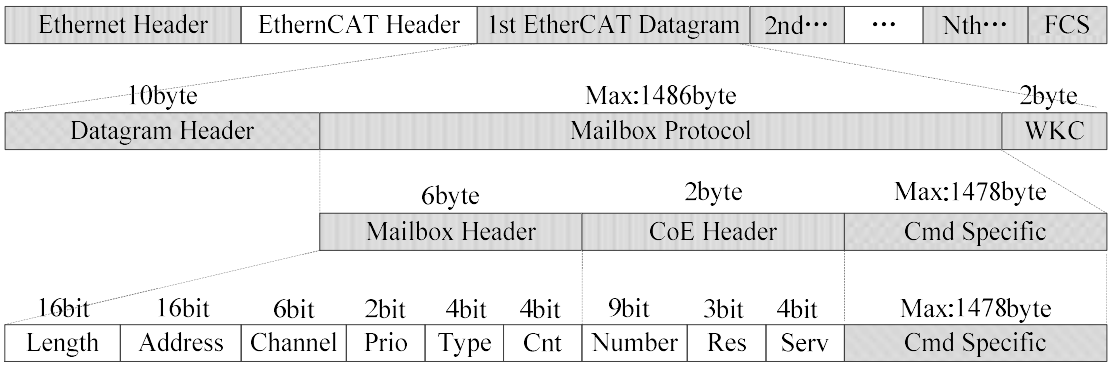
In the ESC configuration area (EEPROM word addresses 0000h to 0007h), after the driver power is turned on, the configured station alias automatically reads and writes to the ESC register based on ESC. When reflecting the updated value of SII EEPROM to the ESC register, it is necessary to restart the power supply. In addition, the initial value of the IP core (ET1810/ET1811/ET1812) is set. Please refer to the data table for IP cores (ET1810/ET1811/ET1812) for detailed information.

### 4-2-5. SDO

The DS5C1 series supports SDO (Service Data Object). The data exchange of SDO uses Mailbox communication, so the data refresh time of SDO becomes unstable.
The master station can read and write data from the records in the object dictionary, and can set objects and monitor various states of the slave station. The response to read and write actions to SDO takes time. Please do not use SDO to refresh objects that have been refreshed with PDO. Please overwrite with PDO values.

#### 4-2-5-1. Mailbox frame structure

The frame structure of Mailbox/SDO is shown below. Please refer to the ETG specification sheet (ETG1000-5 and ETG1000-6) for details.




| Frame | Data area | Data type | Function |
|---|---|---|---|
| MailBox Header | Length | WORD | Mailbox data length |
| | Address | WORD | The station address of sending source |
| | Channel | Unsigned6 | (Reserved) |
| | Prority | Unsigned2 | Priority |
| | Type | Unsigned4 | Mailbox type<br>00h：error<br>01h：(Reserved)<br>02h：EoE (Not corresponding)<br>03h：CoE<br>04h：FoE (Not corresponding)<br>05h：SoE (Not corresponding)<br>06h-0Eh：(Reserved) |

| Frame | Data area | Data type | Function |
|---|---|---|---|
| | | | 0Fh：VoE (Not corresponding) |
| | Cnt | Unsigned3 | Mailbox counter |
| | Reserved | Unsigned1 | (Reserved) |
| CoE Header | Number | Unsigned9 | Reserved |
| | Reserved | Unsigned3 | Reserved |
| | Service | Unsigned4 | Information type |
| Cmd specific | Size Indicator | Unsigned1 | Data Set Size License |
| | Transfer Type | Unsigned1 | Normal Forwarding/Expedited Forwarding |
| | Data Set Size | Unsigned2 | Specify data size |
| | Complete Access | Unsigned1 | Selection of access methods for objects (not corresponding) |
| | Command Specfier | Unsigned3 | Upload/Download<br>Selection of requirements/responses, etc |
| | Index | WORD | Object Index |
| | Subindex | BYTE | Object Subindex |
| | | | Object data or Abort message |

4-2-5-2. Mailbox timeout

This servo driver performs the following timeout settings in the mailbox communication.

Mailbox request timeout: 100ms

The master station sends a request to the slave station (driver), and if the WKC of the transmission data of the request frame is updated, the slave station is considered to receive the request normally. Until the WKC is updated, retry repeatedly. However, if the WKC is not updated by this set time, the main station will timeout.

Mailbox response timeout: 10s

The master station receives a response from the slave station (driver) request, and if this WKC is updated, it is considered a normal receiving response. Until this set time, if the response of WKC being updated cannot be received, the main station side will time out.

The maximum time required for the response of the slave station (driver) to complete.

4-2-5-3. Information during abnormal alarm

(1) Error code

Error code returns the same value as 603Fh (Error code).

0000h~FEFFh is defined according to IEC61800-7-201.

FF00h to FFFFh are defined by the manufacturer, as follows:

| Index | Sub-Index | Name/Description | Range | Date Type | Access | PDO | Op-mode |
|---|---|---|---|---|---|---|---|
| 603Fh | 00h | Error code | 0-65535 | U16 | ro | TxPDO | All |
| | | The alarm that occurs in the servo drive now (only the main number).<br>When the alarm does not occur, it displays 0000h.<br>When an alarm occurs, it displays an alarm.<br>FF**h<br>Alarm (main) number (00h~FFh)<br>(Example) FF03h... 03h=3d E-030 (overvoltage protection) occurs<br>FF55h... 55h=85d E-850 (TxPDO configuration exception protection), E-851 | | | | | |

| | | |
|---|---|---|
| | | (RxPDO configuration exception protection), any one of which occurs<br>As an exception, in the case of E-817 (SyncManager2/3 setting exception), A000h is displayed. |

(2) Error register

Error register returns the same value as 1001h (Error register).

| Index | Sub-Index | Name/Description | Range | Date Type | Access | PDO | Op-mode |
|---|---|---|---|---|---|---|---|
| 1001h | 00h | Error register | 0-65535 | U16 | ro | TxPDO | All |

Display the type (status) of alarm that is currently occurring in the servo drive.
When the alarm does not occur, it displays 0000h.
Do not display warnings.

| Bit | Content |
|---|---|
| 0 | Not support |
| 1 | |
| 2 | |
| 3 | |
| 4 | AL status code defined alarm occurrence*1 |
| Bit | Content |
| 5 | Not support |
| 6 | Reserved |
| 7 | AL status code Undefined alarm occurrence*2 |

*1: The so-called "AL status code defined alarm" refers to EtherCAT communication association abnormalities E-800-7, E-810-7, E-850-7.
*2: The so-called "AL status code undefined alarm" refers to abnormalities in EtherCAT communication association E-880-7 and anomalies outside of EtherCAT communication association.

## 4-2-6. PDO

The DS5C1 series supports PDO (Process Data Object).
Real time data forwarding based on EtherCAT is carried out through data exchange through PDO (Process Data Object).
PDO includes RxPDO for transfer from master station to slave station and TxPDO for transfer from slave station to master station.

| | Sending side | Receiving side |
|---|---|---|
| RxPDO | Master station | Slave station |
| TxPDO | Slave station | Master station |

### 4-2-6-1. PDO mapping object

PDO mapping refers to the mapping of application objects from object dictionaries to PDO.
The table used for DS5C series PDO mapping can use mapping objects ranging from 1600h to 1603h for RxPDO and 1A00h to 1A03h for TxPDO.
The maximum number of application objects that can be mapped by a mapping object is as follows:
RxPDO: 24 [byte] , TxPDO: 24 [byte]
The following is an example of setting PDO mapping.
<Setting Example>
Assign application objects 6040h, 6060h, 607Ah, 60B8h to map object 1600h (Receive PDO mapping 1:

RxPDO1).

| Index | Sub | Object contents | |
|---|---|---|---|
| 1600h | 00h | 04h | |
| | 01h | 6040 00 10 h | |
| | 02h | 6060 00 08 h | |
| | 03h | 607A 00 20 h | |
| | 04h | 60B8 00 10 h | |
| | 05h | 0000 00 00 h | |
| | … | | |
| | 18h | 0000 00 00 h | |
| 6040h | 00h | Controlword | U16 |
| 6060h | 00h | Mode of operation | I8 |
| 607Ah | 00h | Target Position | I32 |
| 60B8h | 00h | Touch probe function | U16 |

4-2-6-2. PDO allocation object

For PDO data exchange, it is necessary to allocate the tables used for PDO mapping to SyncManager. The relationship between the table used for PDO mapping and SyncManager is described to the PDO allocation object. The DS5C series, as a PDO allocation object, can use RxPDO (SyncManager2) for 1C12h and TxPDO (SyncManager3) for 1C13h.
The maximum number of application objects that can be mapped by a mapping object is as follows:
RxPDO: 4 [Table] (1600h～1603h).
RxPDO: 4 [Table] (1A00h～1A03h).
Usually, since one mapping object is sufficient, it does not need to be changed by default.

Example of setting PDO allocation objects:
Assign mapping object 1600h to allocation object 1C12h (Sync manager channel 2).

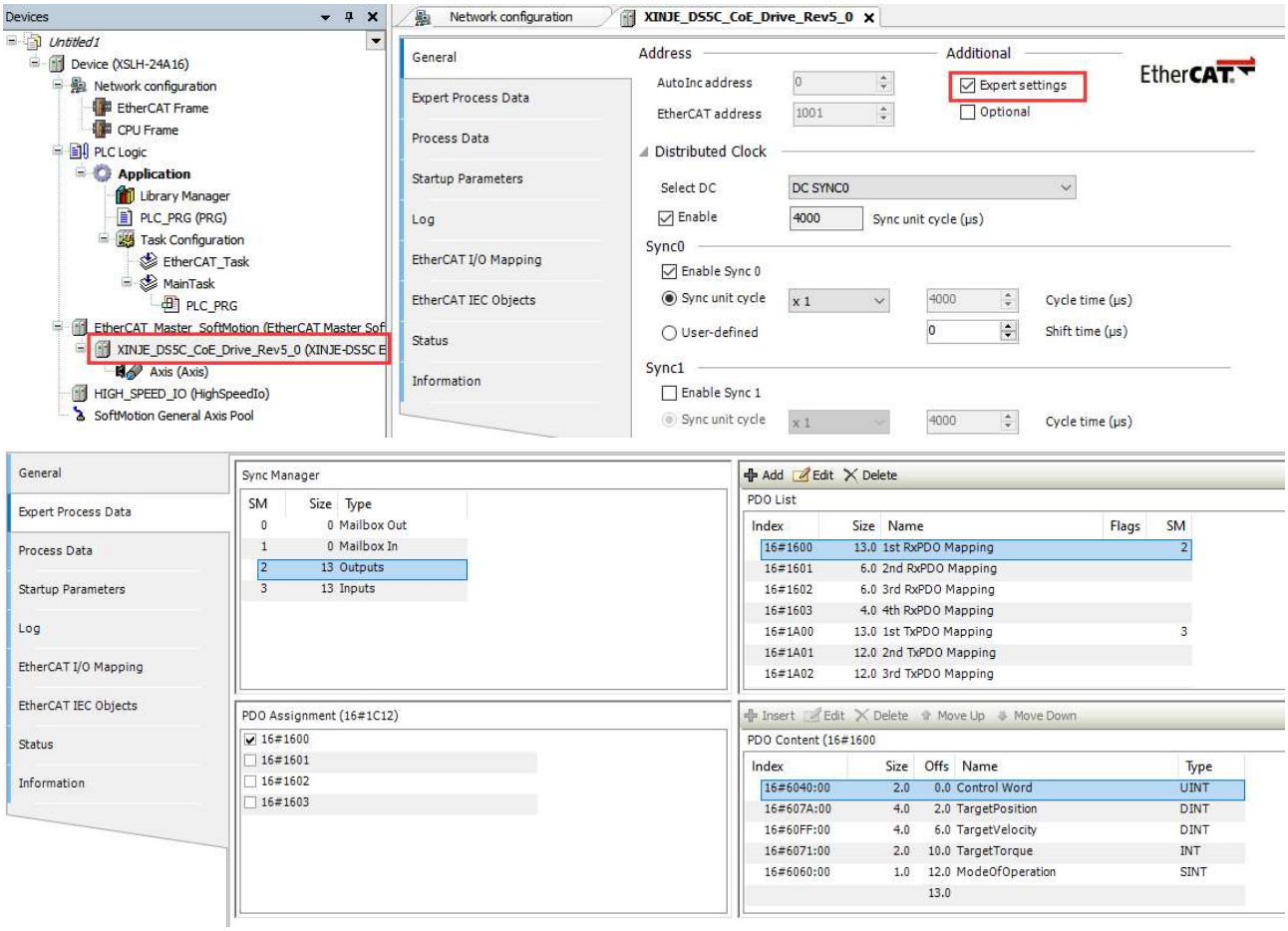
| Index | Sub | Object contents |
|---|---|---|
| 1C12h | 00h | 01h |
| | 01h | 1600h |
| | 02h | 0000h |
| | 03h | 0000h |
| | 04h | 0000h |

Assign mapping object 1600h to allocation object 1C13h (Sync manager channel 3).

| Index | Sub | Object contents |
|---|---|---|
| 1C13h | 00h | 01h |
| | 01h | 1A00h |
| | 02h | 0000h |
| | 03h | 0000h |
| | 04h | 0000h |

### 4-2-6-3. PDO configuration

Double click EtherCAT slave device - "General" - Check "Expert Settings" - "Expert Process Data"



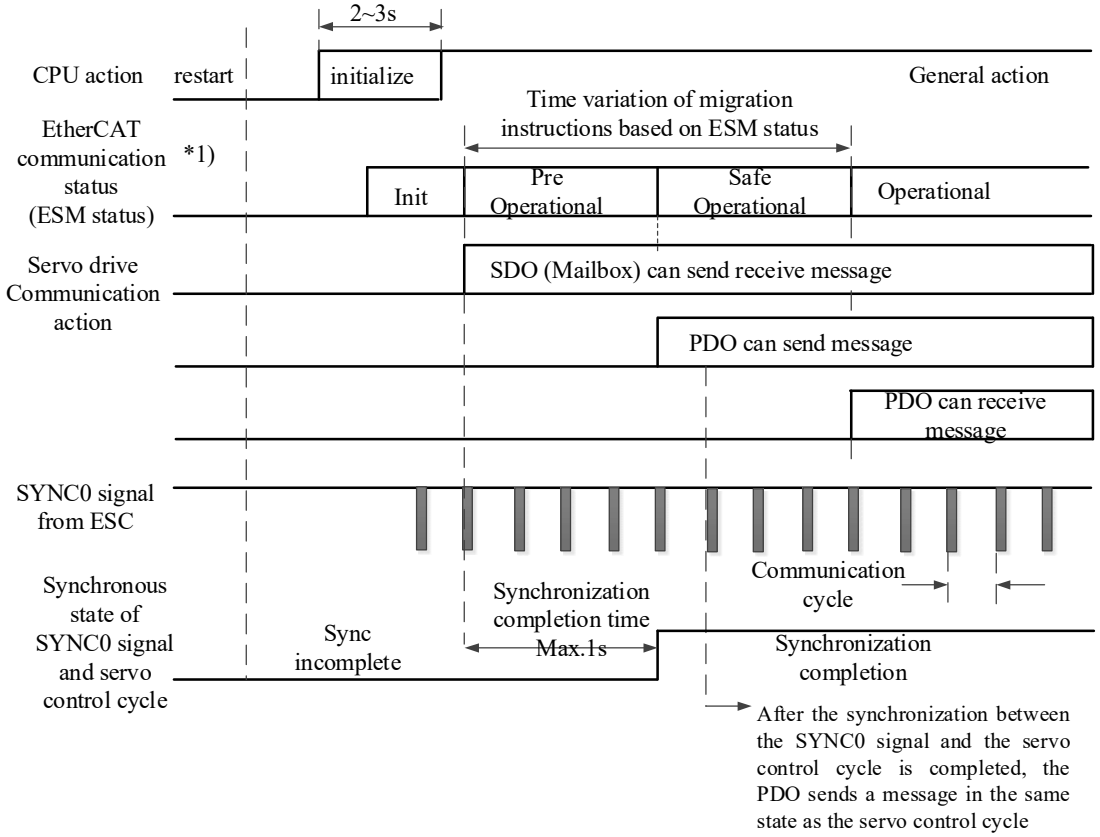

## 4-2-7. Communication synchronization mode

The DS5C series can choose from the following synchronization modes.

| Synchronization mode | Content | Synchronous method | Features |
|---|---|---|---|
| DC | SYNC0 event synchronization | Synchronize the time information of other slave stations based on the time of the first axis | High precision<br>Compensation processing needs to be carried out on the main station side |
| SM2 | SM2 event synchronization | Synchronize according to the receiving time of RxPDO | No transmission delay compensation, poor accuracy<br>Need to maintain transmission time on the controller side (dedicated hardware, etc.) |
| FreeRun | Asynchronous | Asynchronous | Easy to handle<br>Poor real-time performance |

4-2-7-1. DC (SYNC0 event synchronization)

The DS5C series has a 64 bits DC (Distributed Clock).
The synchronization of EtherCAT communication is based on this DC. According to the DC slave station, synchronization is achieved through a shared clock (System Time) with the same reference. The local cycle of the slave station starts with the SYNC0 event. Because the processing of the slave station (servo processing) starts with the SYNC0 event cycle, it is always synchronized with the SYNC0 event.
The master station needs to perform transmission delay compensation (offset compensation) and regular deviation compensation during communication initialization. The following figure shows the synchronization process from the control power input to the SYNC0 event and the processing of the slave station (servo processing).




4-2-7-2. SM2 (SM2 event synchronization)

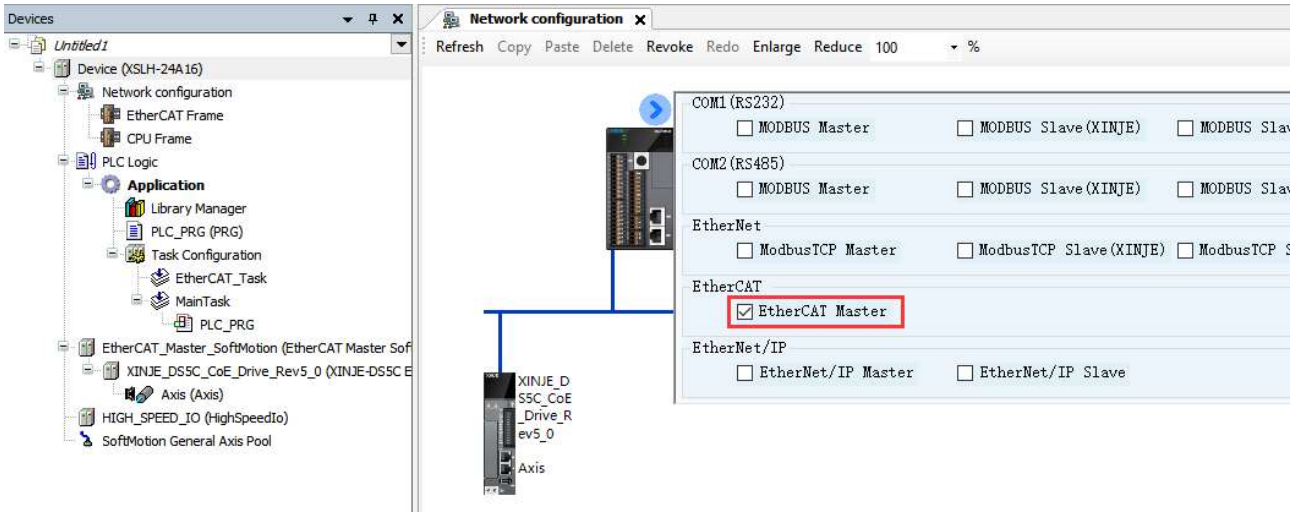
The local cycle of the slave station starts from the SM2 event.
Because the processing of the slave station starts with the SM2 event cycle, it is always synchronized with the SM2 event.
Because the SM2 incident occurred when the PDO received the message, it is important to ensure that the upper (main) side sends the message on a scheduled basis. If the fluctuation (deviation) of the delivery time is too large, synchronization cannot be completed, or an alarm occurs.
If the above problem occurs, please use DC (SYNC0 event synchronization).

# 4-3. EtherCAT parameter configuration

## 4-3-1. EtherCAT master station

### 4-3-1-1. Add master station

Click on the enable window in the network configuration interface, and add the master station device by checking "EtheCAT Master", as shown in the figure:



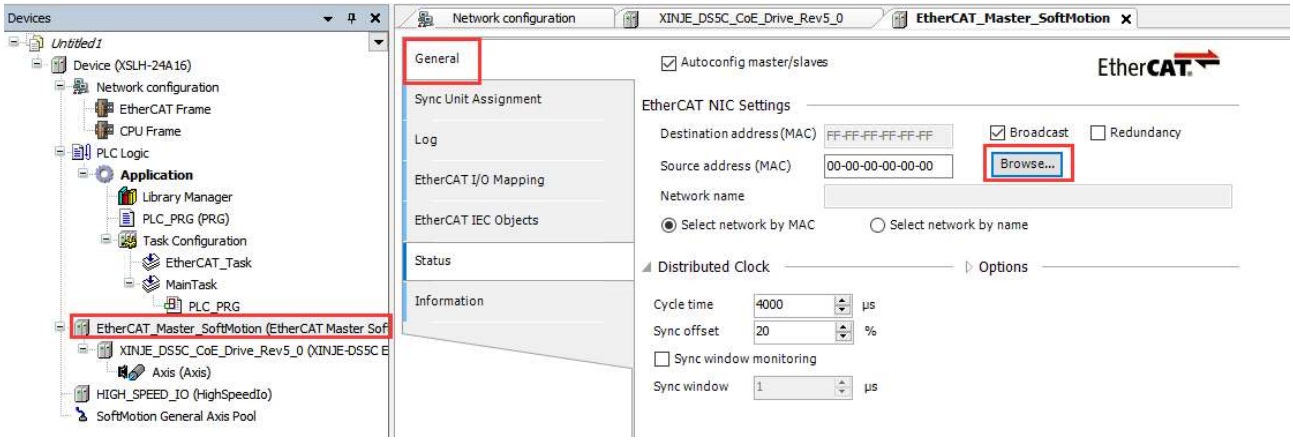

### 4-3-1-2. General

(1) EtherCAT NIC setting

Destination address (MAC): The destination address for receiving EtherCAT messages. If the "broadcast" option is activated, there is no need to enter the destination address. The system will automatically search for the destination address through broadcast.

Redundancy: When this option is enabled, EtherCAT redundancy mode is officially enabled, which supports ring topology.

Source address (MAC): The MAC address of the PLC network interface, which can be selected as "Select network by MAC" or "Select network by name". Users can select the "Browse" to select the source address they want to set.



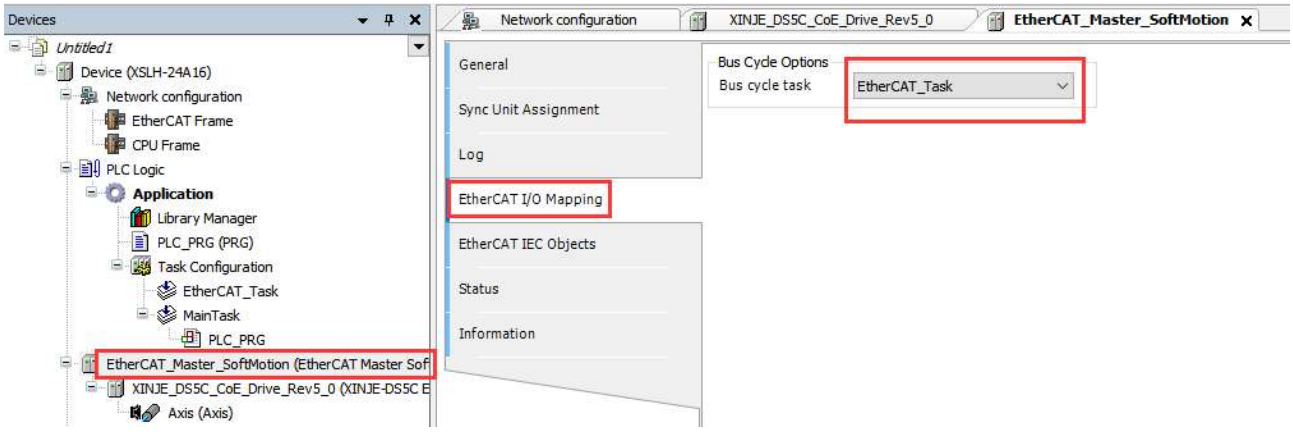

(2) Distributed clock

Cycle time: If the distributed clock function is activated, the master station will send corresponding data packets to the slave station based on the cycle time. Therefore, data exchange can achieve precise synchronization, and this function is particularly important when synchronous actions are required in distributed processes (such as

multiple servo axes executing simultaneous linkage tasks). Can provide a master clock with signal jitter less than 1 microsecond within the network range.
Sync offset: Usually, when the PLC task starts 20%, the synchronization message begins to affect the slave station, which means that the PLC task cycle can have an 80% delay, and no data will be lost within this delay.
Sync window monitoring: If this option is turned on, it can monitor the synchronization status of the slave station.
Sync window: used to monitor the time of the synchronization window. If all slave stations are within the synchronization window time, the variable xSyncInWindow (IoDrvEtherCAT) will be set to True, otherwise it will be FALSE.

#### 4-3-1-3. EtherCAT I/O mapping

When establishing an EtherCAT master station, EtherCAT_Task will be automatically established, set bus cycle task in EtherCAT I/O mapping, default to EtherCAT_Task.

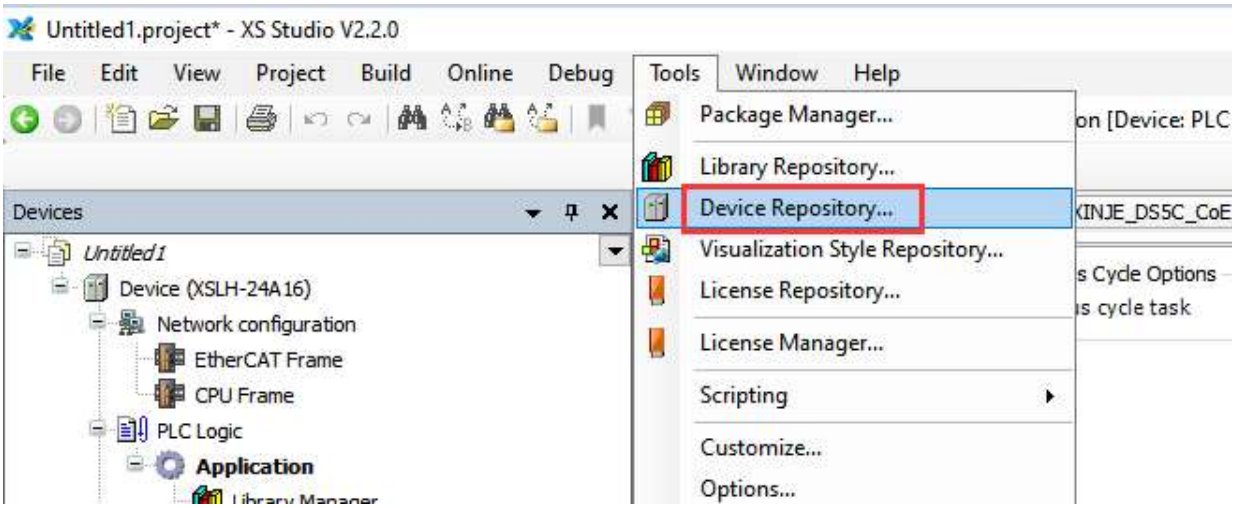


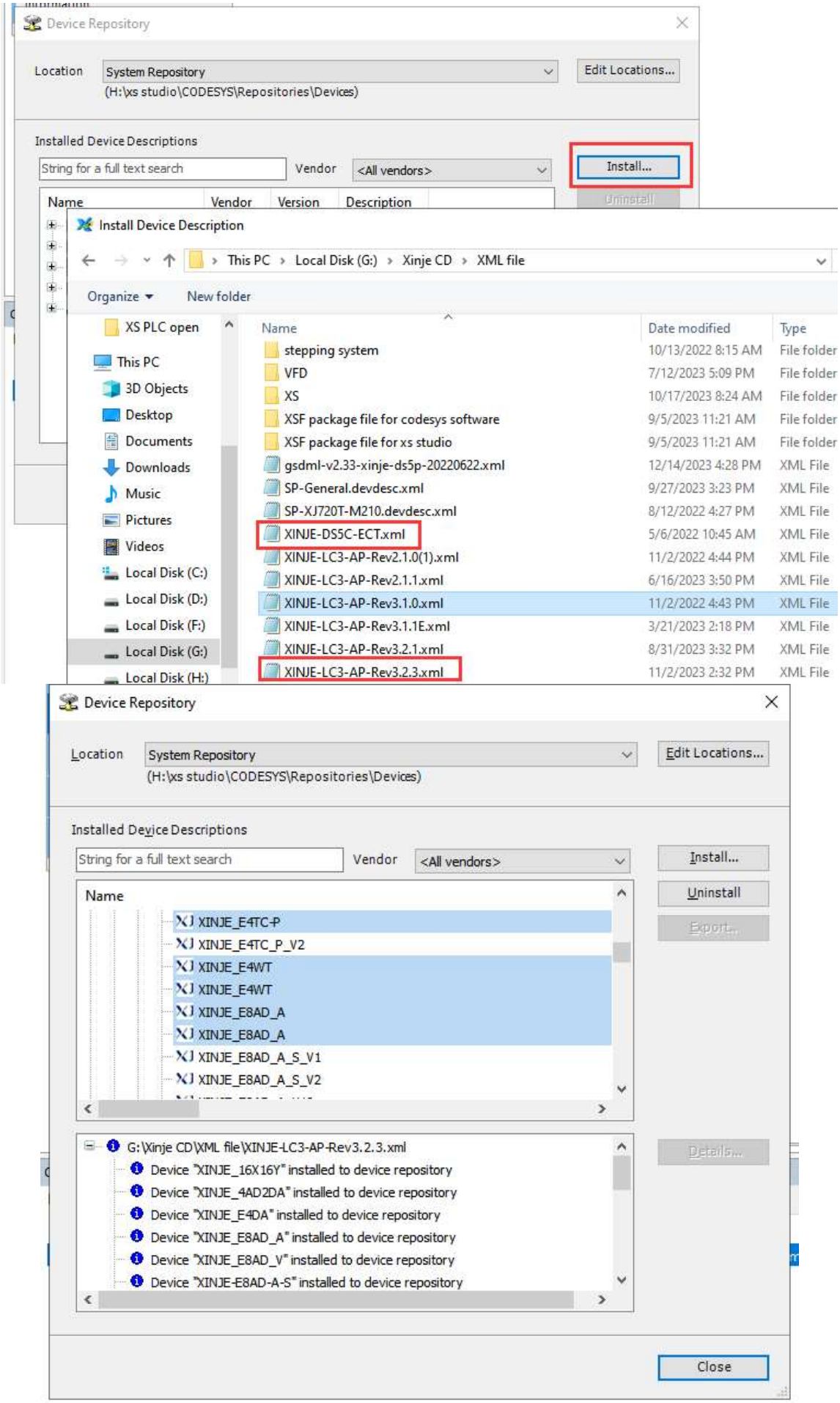

### 4-3-2. EtherCAT slave station

#### 4-3-2-1. Add slave station

(1) Add xml file

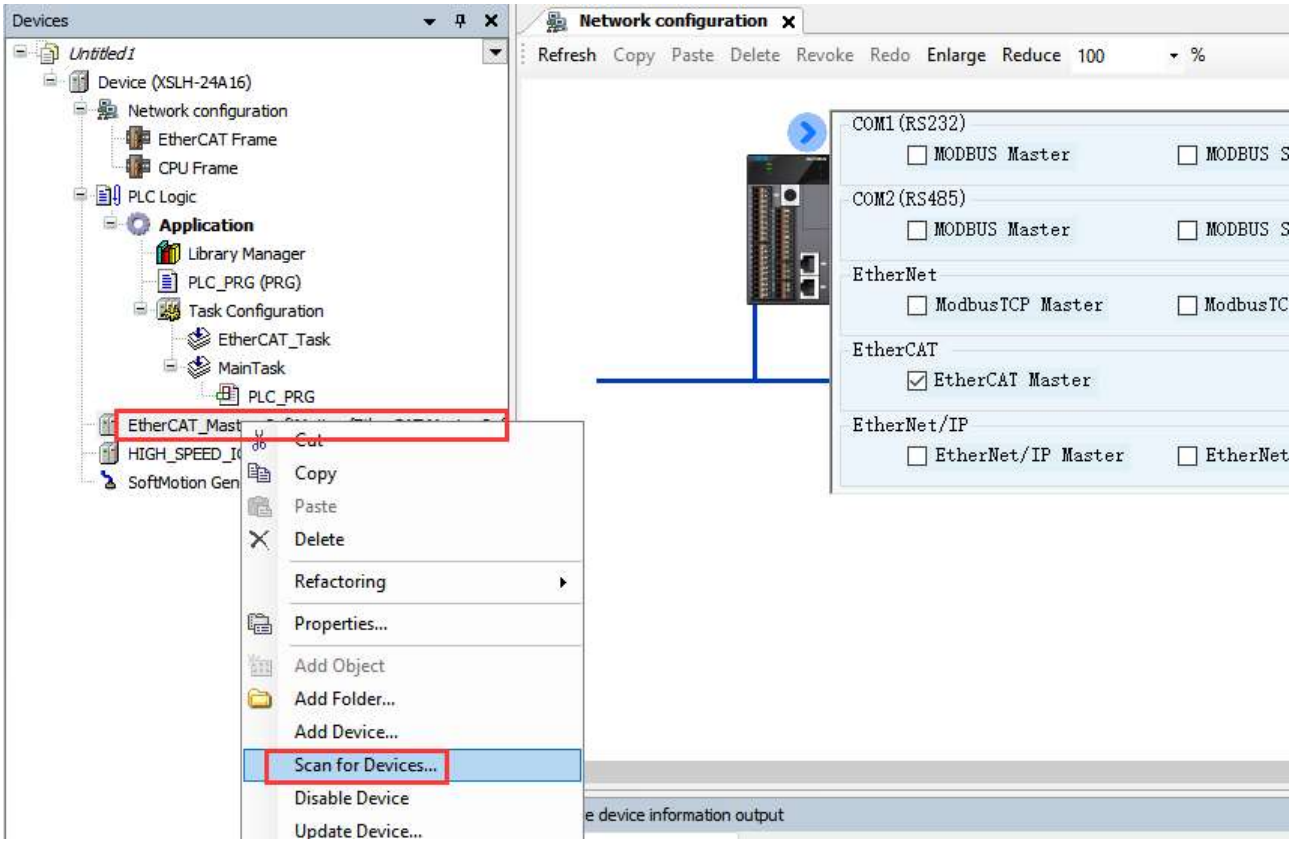
Open the tool device library and add the XML file of the slave device. Click "Tools" - "Device repository..." in sequence, click "Install" in the pop-up dialog box, select the path where the XML file is located, find the XML file, select it, and click open.



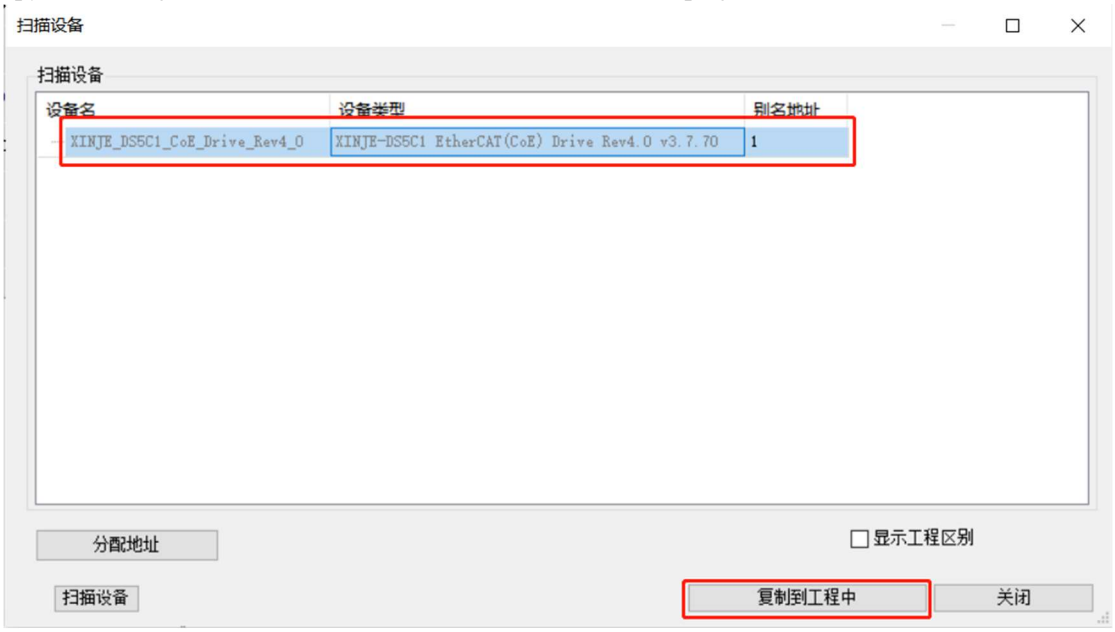

Device Repository
Location
System Repository
(H:\xs studio\CODESYS\Repositories\Devices)
Edit Locations...
Installed Device Descriptions
String for a full text search
Vendor
<All vendors>
Install...
Name
Vendor
Version
Description
Install Device Description
This PC › Local Disk (G:) › Xinje CD › XML file
Organize
New folder
XS PLC open
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
Local Disk (D:)
Local Disk (F:)
Local Disk (G:)
Local Disk (H:)
Name
Date modified
Type
stepping system 10/13/2022 8:15 AM File folder
VFD 7/12/2023 5:09 PM File folder
XS 10/17/2023 8:24 AM File folder
XSF package file for codesys software 9/5/2023 11:21 AM File folder
XSF package file for xs studio 9/5/2023 11:21 AM File folder
gsdml-v2.33-xinje-ds5p-20220622.xml 12/14/2023 4:28 PM XML File
SP-General.devdesc.xml 9/27/2023 3:23 PM XML File
SP-XJ720T-M210.devdesc.xml 8/12/2022 4:27 PM XML File
XINJE-DS5C-ECT.xml 5/6/2022 10:45 AM XML File
XINJE-LC3-AP-Rev2.1.0(1).xml 11/2/2022 4:44 PM XML File
XINJE-LC3-AP-Rev2.1.1.xml 6/16/2023 3:50 PM XML File
XINJE-LC3-AP-Rev3.1.0.xml 11/2/2022 4:43 PM XML File
XINJE-LC3-AP-Rev3.1.1E.xml 3/21/2023 2:18 PM XML File
XINJE-LC3-AP-Rev3.2.1.xml 8/31/2023 3:32 PM XML File
XINJE-LC3-AP-Rev3.2.3.xml 11/2/2023 2:32 PM XML File
Device Repository
Location
System Repository
(H:\xs studio\CODESYS\Repositories\Devices)
Edit Locations...
Installed Device Descriptions
String for a full text search
Vendor
<All vendors>
Install...
Uninstall
Export...
Name
XINJE_E4TC-P
XINJE_E4TC_P_V2
XINJE_E4WT
XINJE_E4WT
XINJE_E8AD_A
XINJE_E8AD_A
XINJE_E8AD_A_S_V1
XINJE_E8AD_A_S_V2
G:\Xinje CD\XML file\XINJE-LC3-AP-Rev3.2.3.xml
Device "XINJE_16X16Y" installed to device repository
Device "XINJE_4AD2DA" installed to device repository
Device "XINJE_E4DA" installed to device repository
Device "XINJE_E8AD_A" installed to device repository
Device "XINJE_E8AD_V" installed to device repository
Device "XINJE-E8AD-A-S" installed to device repository
Details...
Close

(2) Scan the slave station

In the Device project bar, right-click EtherCAT_ Master_ SoftMotion, click on "Scan for Device" to scan EtherCAT slave devices, or right-click on EtherCAT_ Master_ SoftMotion, click "Add Device" to manually add the device.

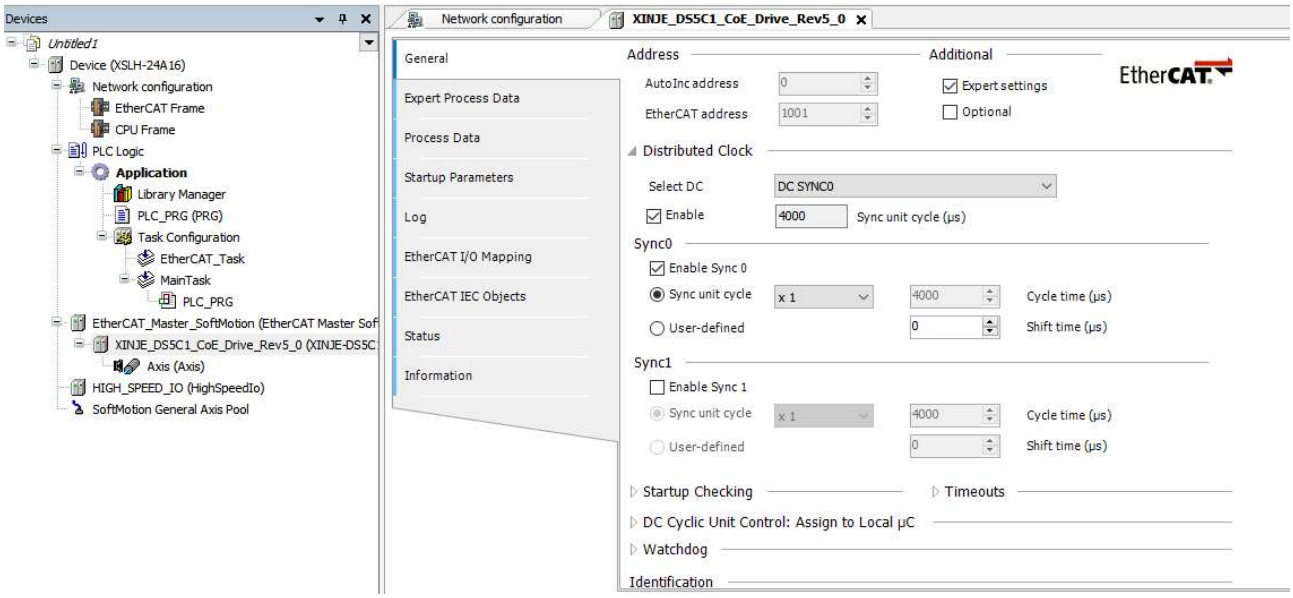



In this example, one DS5C1 series servo was connected, and the scanning result is shown in the following figure. Click Copy All to Project to add all the scanned slave stations to the project.




Note: Before using the "Scan for Device", it is necessary to ensure that the EtherCAT device description file of the slave has been installed in the XS Studio of the debugging PC, otherwise this feature cannot be used.

4-3-2-2. General



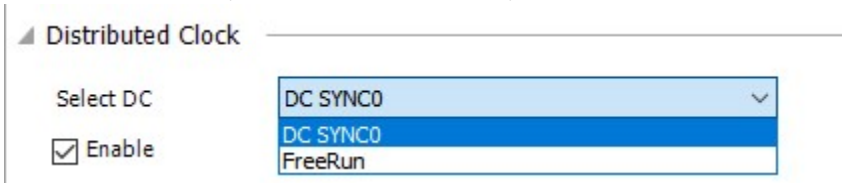

(1) Address

Automatic address configuration: determined by the location of the slave station in the network. This address is only used during startup, and the master station needs to allocate an EtherCAT address to the slave station. When the first message used for this purpose passes through a slave station, each passing slave station adds its own automatic incremental address by 1.

EtherCAT address: The final address of the slave station, allocated by the master station at startup.

(2) Distributed clock

Select DC: The dropdown menu provides all the settings related to distributed clocks provided by the device description file, and can be selected as synchronous or freerun asynchronous mode.



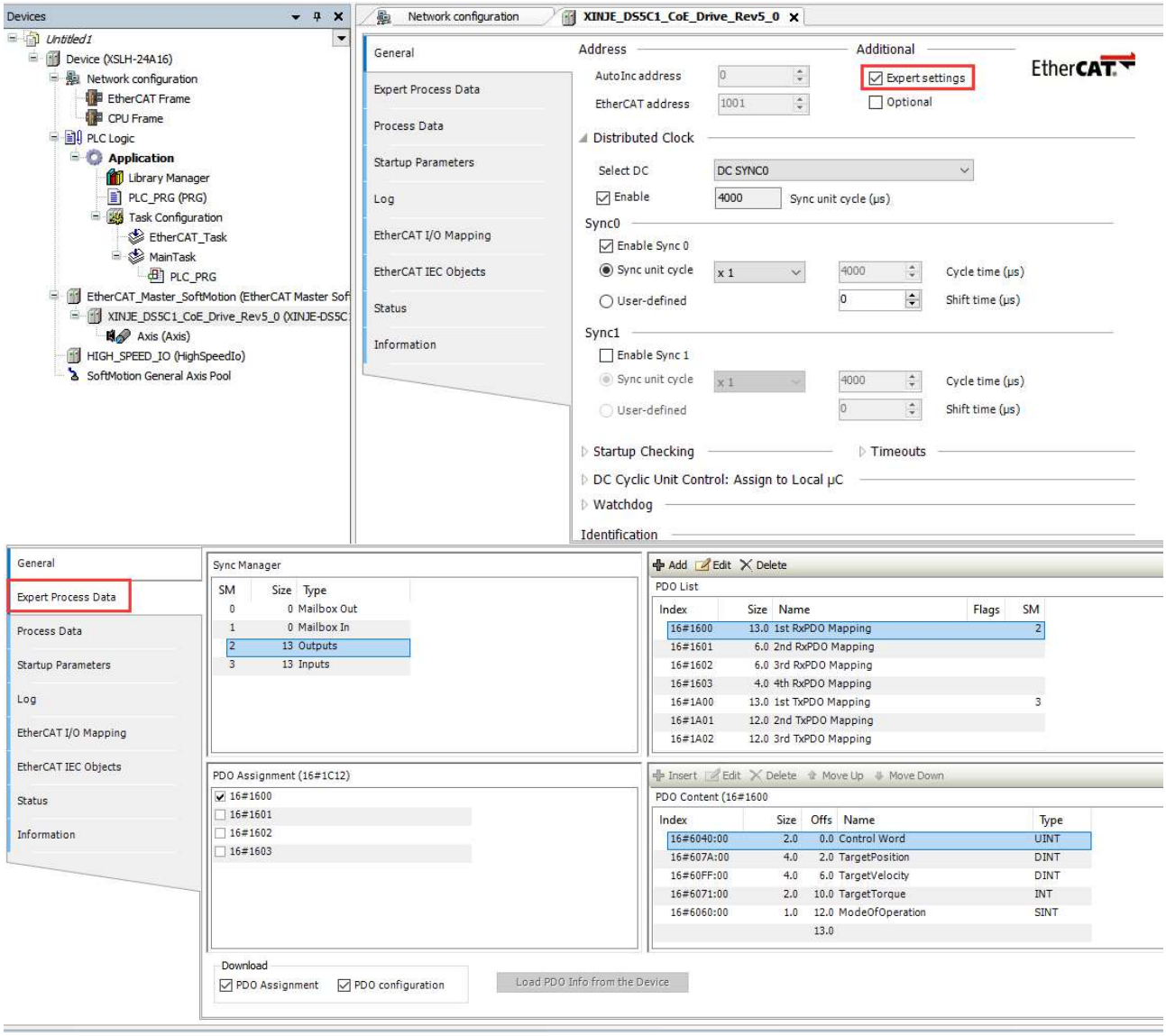

（3） Sync 0/1

Enable Sync 0/1: If this option is selected, use "sync0/1" to synchronize the unit. A synchronization unit describes a set of process data for synchronous exchange.

Sync unit cycle: The time of the master station cycle multiplied by the selected coefficient will be used as the synchronization cycle time of the slave station. The cycle time (us) displays the current set cycle time.

### 4-3-2-3. Expert settings

In the general interface, selecting Expert Settings will bring up the configuration interface for expert process data.

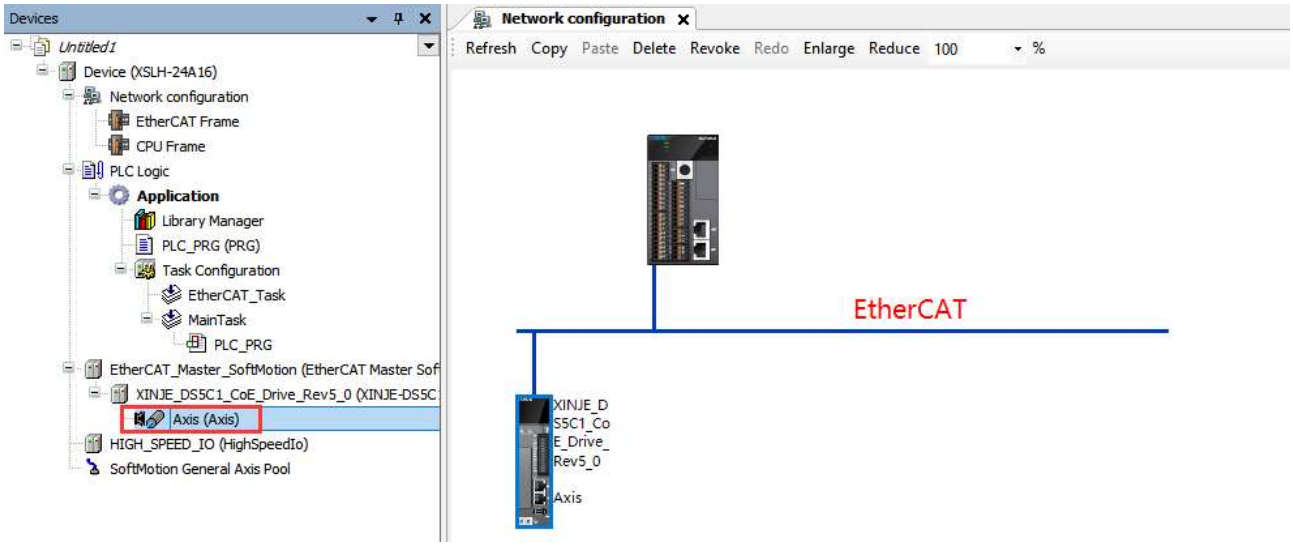



## 4-3-3. Axis configuration

### 4-3-3-1. Xinje axis 402

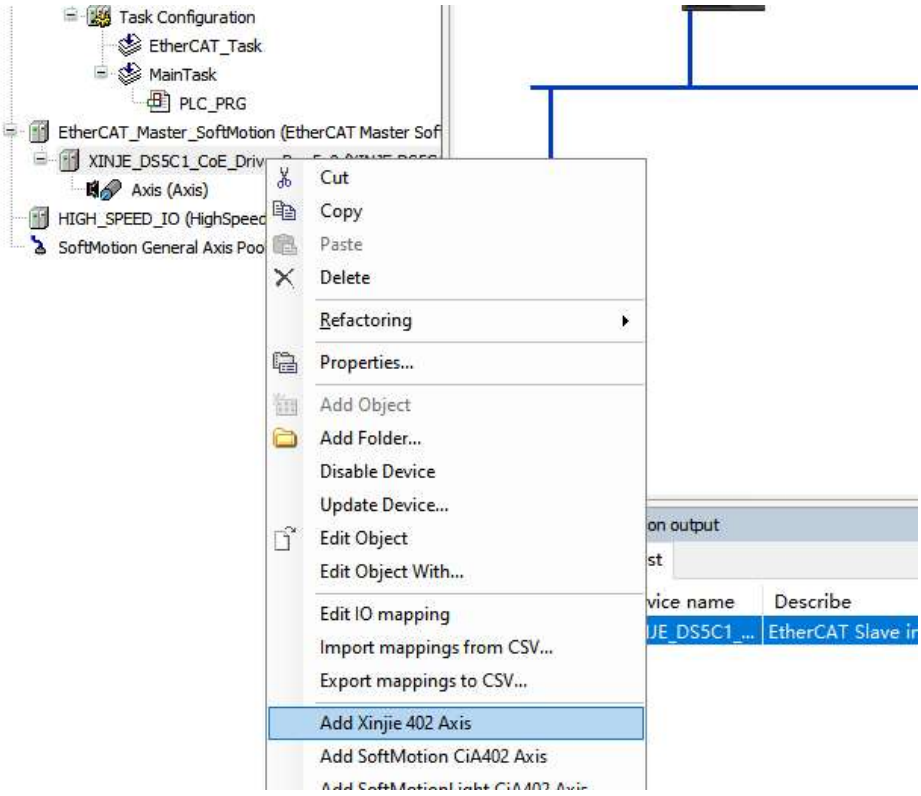
1. Add "Xinje 402 Axis"

Method 1: Enable the EtherCAT master station in the network configuration interface. When adding the servo slave station equipment of Xinje, the Xinje 402 axis will be automatically added. As shown in the following figure:

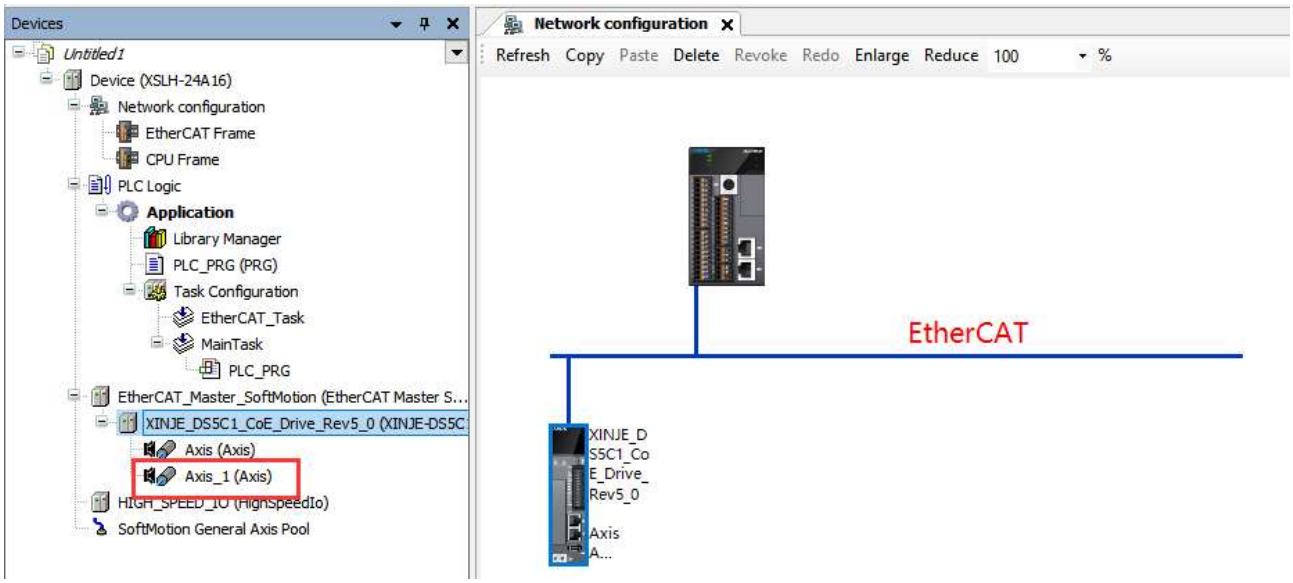



Method 2: After adding a servo slave station, right-click the menu to add "Xinje 402 Axis". As shown in the following figure:

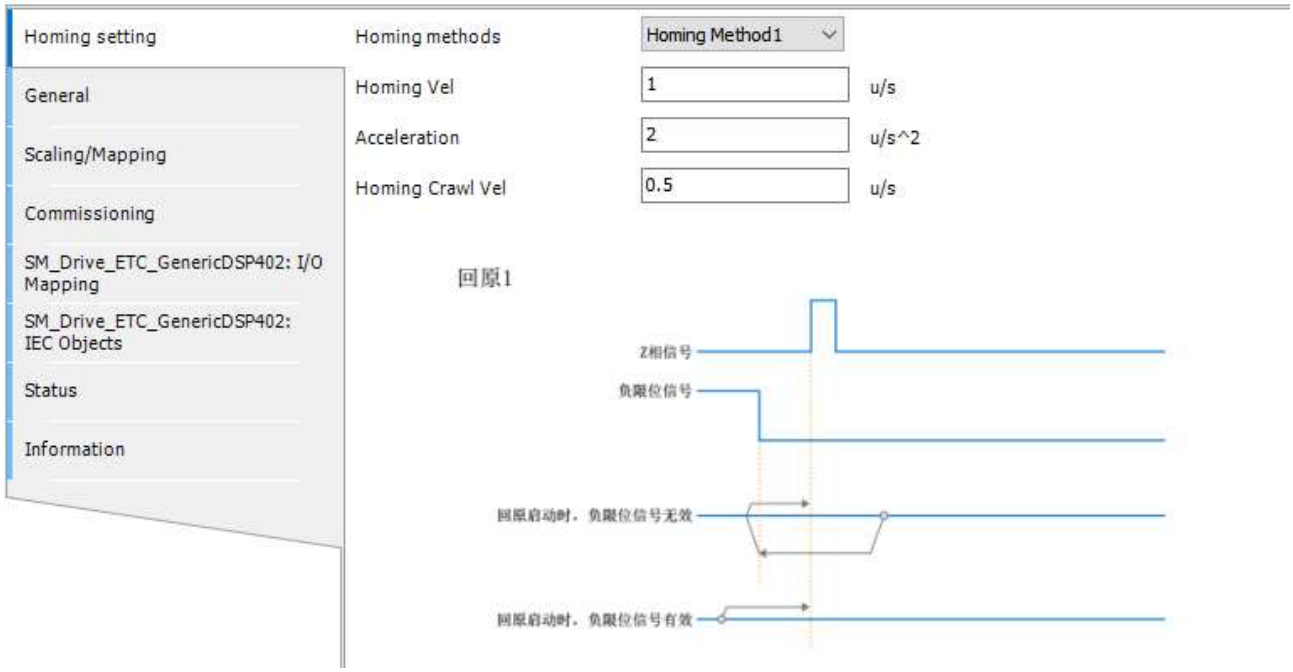
Before adding:




After adding Xinje 402 axis:

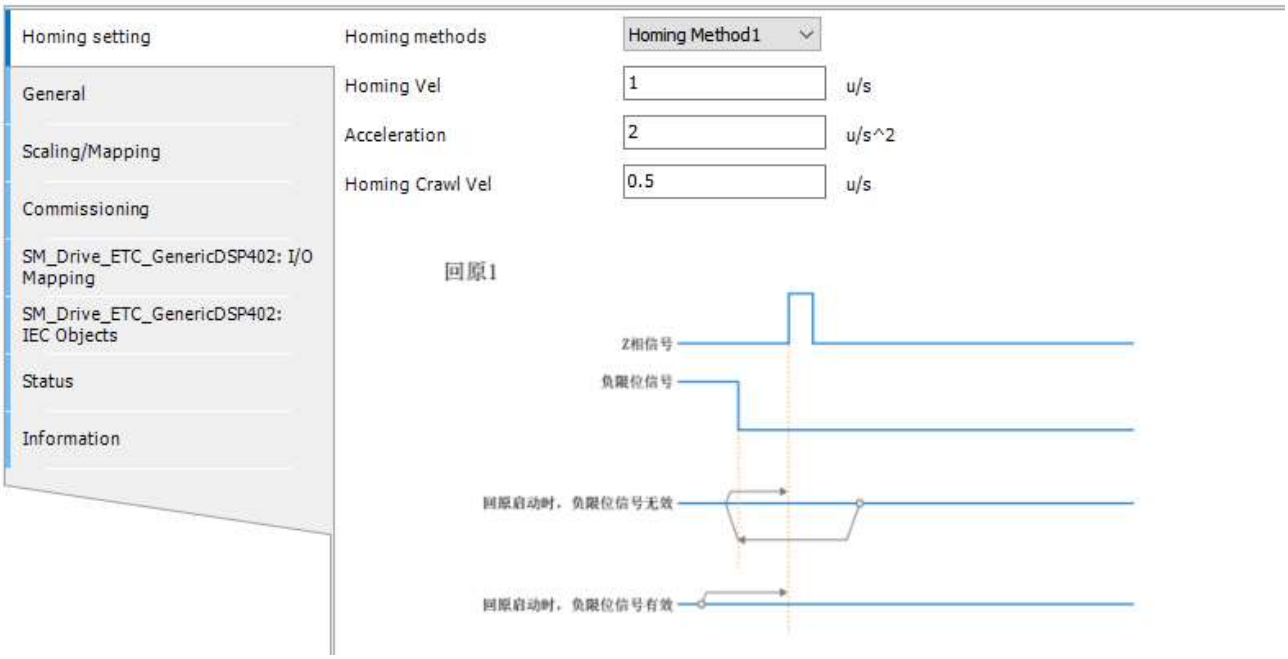



Double click on "Axis" to open the corresponding axis configuration interface, as shown in the following figure:




3. Homing configuration

HOME reset parameter settings are mainly used for graphical parameter configuration of axis homing. Provides graphical configuration guidance, allowing users to directly select the desired homing method through the drop-down menu in the configuration interface without the need to consult the servo manual separately, making it more intuitive and convenient for users to complete the parameter configuration process.




The main options and their functions in the figure are as follows:

① Homing methods (#6098h: Home method)

There are a total of 35 options supported for configuring the way the driver homing (the actual way of homing is determined by the driver). The example diagram below will vary for each different homing method (refer to the servo homing method of the DS5C series), and different homing methods can be selected according to needs.

② Homing Velocity (#6099h subindex 01h)

Set the speed of the action detected by the Switch signal.

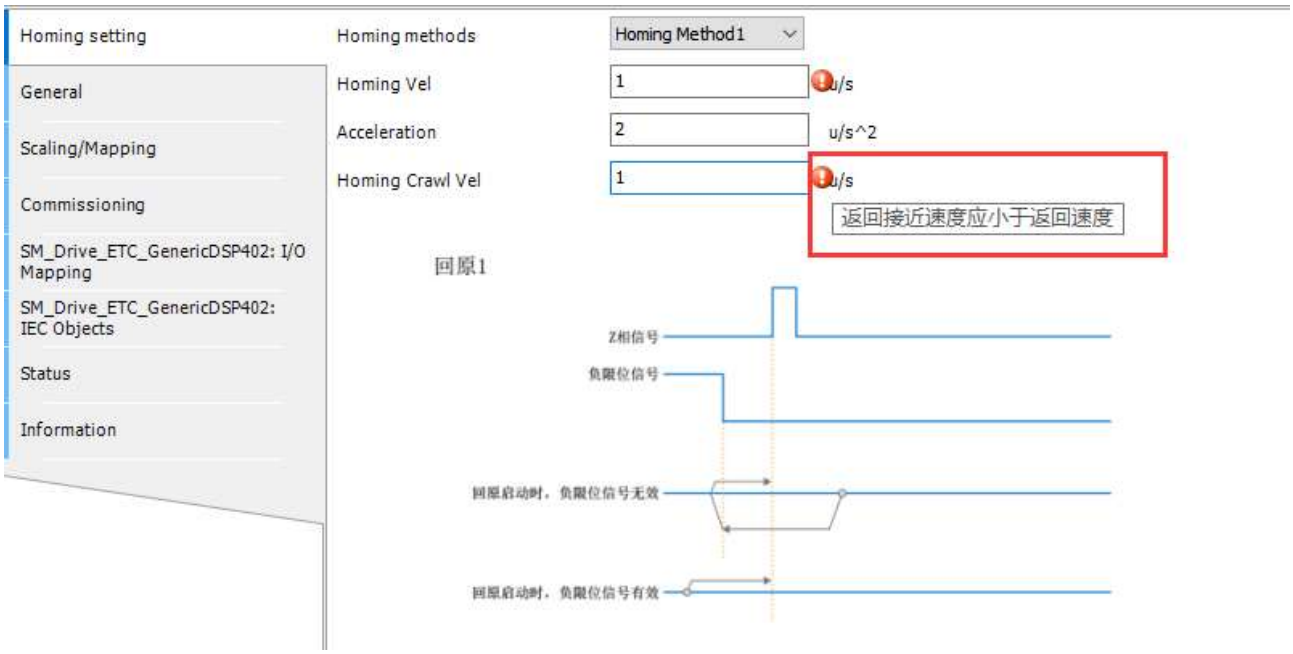
③ Acceleration (#609Ah)

Set the acceleration and deceleration when homing.

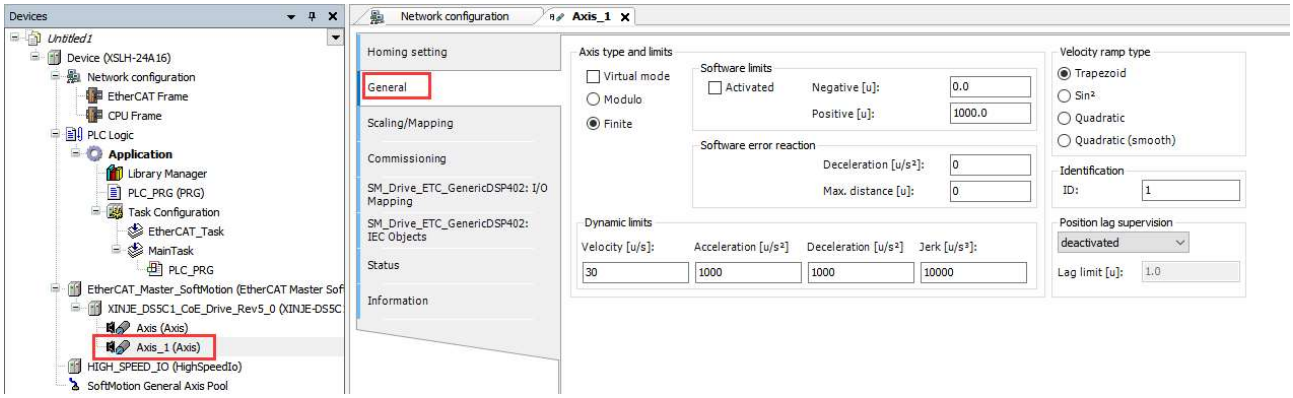
④ Homing Crawl Velocity (#6099h subindex 02h)

Set the action speed for detecting at the origin.

Note: If the homing speed is ≤ Homing crawl speed, an exclamation mark alarm and information prompt will be displayed on the right side of both input boxes. As shown in the following figure:



4-3-3-2. SoftMotion drive: general

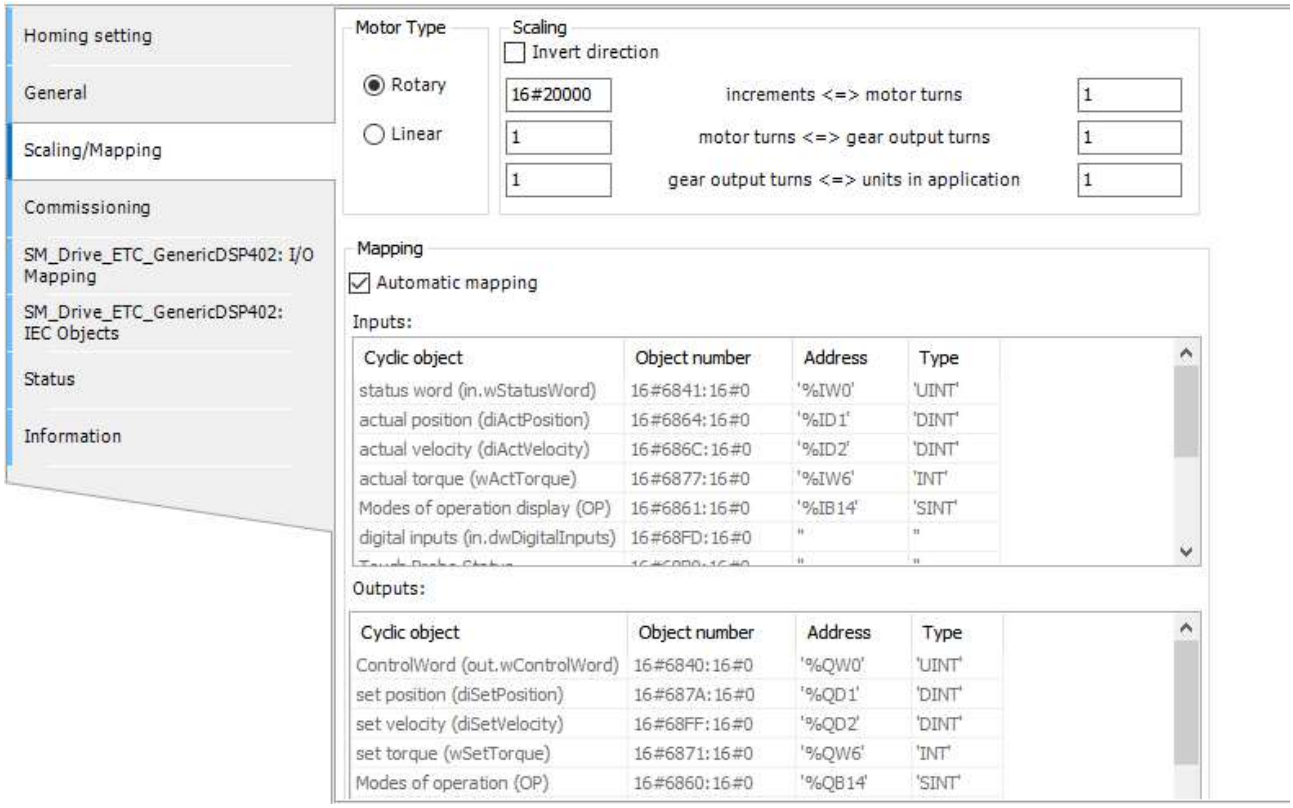



(1) Axis type

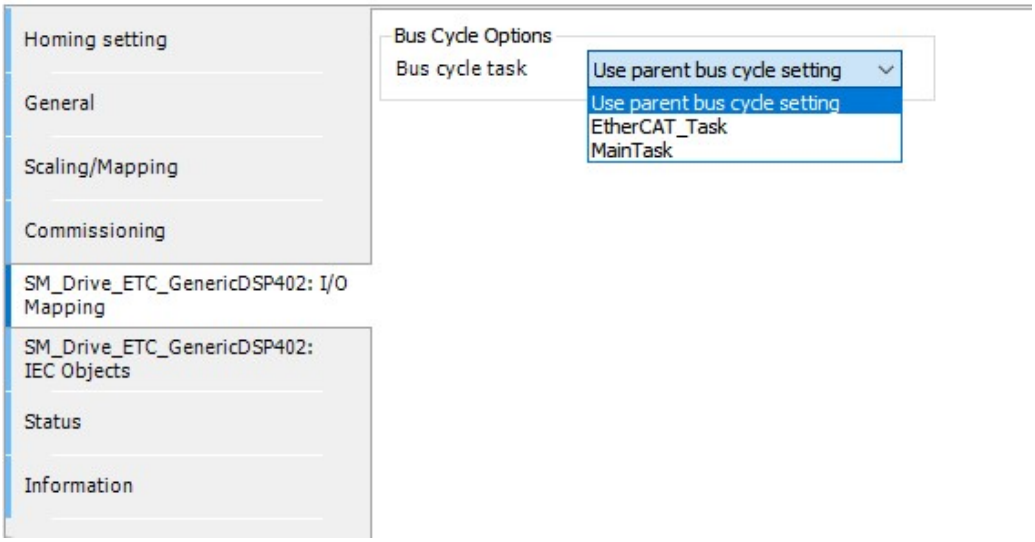
In order to accurately control the motion position, the controller must accurately calculate the position of the servo motor. Based on the operating characteristics and stroke characteristics of the application system, select the "axis type and limit", so that the controller can calculate the feedback information of the motor encoder internally, obtain accurate positions, and avoid errors caused by the accumulation and overflow of encoder pulse numbers. In situations where there is no actual servo motor connected, select "virtual mode"; For the reciprocating mechanism of the screw type, its stroke is limited, and we often need to know its absolute position within the range of screw stroke. In this case, it is better to choose "linear mode"; If the rotation axis of the unidirectional operation type is prone to position counting overflow when using linear mode, resulting in position calculation errors, then choosing "periodic mode" is better.

4-3-3-3. SoftMotion drive: scaling/mapping

The encoder parameters of the motor (such as resolution) and the mechanical reduction ratio of the application system may vary, and programming needs to be set according to the actual situation.



4-3-3-4. SM_Drive_ETC_GenericDSP402:I/O mapping




## 4-3-4. EtherCAT control project

### 4-3-4-1. Motion project control

In a project, all instructions used in the program require support from a file library. Each POU will not be executed if it is not called in a task. Users can choose to configure it directly to a certain task for execution, or choose to call the POU for the configured task from another POU that is already in the task. If the program executed in the POU needs to interact with external IO or buses, corresponding high-speed IO modules or EtherCAT buses and slave devices need to be configured separately in the program.

### 4-3-4-2. Multiple POU usage

When writing applications, program functions with different execution cycles should be placed in different POUs for writing, and configured into tasks with different priorities and cycle times for easy viewing and optimization of subsequent programs.

- Reasonably allocate CPU resources and allocate cycles according to the required cycle time for each function;
- The program structure is clear, and each function is clearly distinguished. Compared to stacking all programs together, the use of multiple POUs can be distinguished by different names to distinguish functions, which is reflected in the engineering column. The logical structure of the program is clear at a glance;
- Debugging is convenient, and during debugging, it is easy to block certain functions that need to be blocked;
- It is possible to directly reference POUs between different projects, copying POUs directly from Project 1 to Project 2;
- After planning the program clearly, it can be divided into multiple individuals for programming and development, improving the efficiency of programming;
- Different programming languages can be used in different POUs, as long as the interface is clear and there are no unified requirements for programming languages within the POU.

### 4-3-4-3. Call motion functions

In a project, in order to allocate CPU resources more reasonably, programs with different cycles are placed in different POUs and tasks during programming.

The motion function requires the highest priority task, while the logic function generally does not require such a high priority task configuration. Therefore, in practical engineering, these two blocks are usually placed between two different POUs and tasks. So, how can we achieve the ability to control the execution of a movement function even if it is separated from a logical function? Generally, input and output variables are defined in the motion function to be called by other functions. For example, in a logical POU, if the motion function needs to be adjusted, control data is written to the input variables of the motion POU. The motion POU places the motion state in the output variable and gives it to the logical POU to determine the motion state and execute the program logic.

# 5. Programming basis

## 5-1. Direct address

### 5-1-1. Defining grammar

In XS Studio applications, this declaration method is required when variable mapping with the I/O module of a programmable logic controller or network communication with external devices is required.

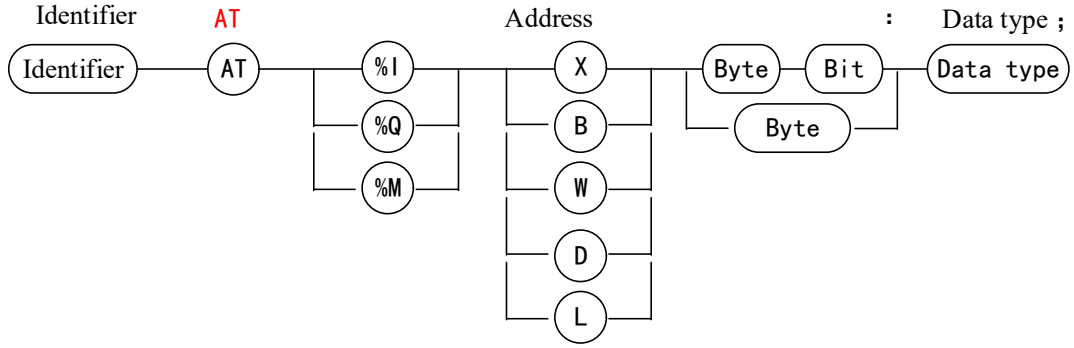
Using the keyword AT to directly link variables to a specific address, direct variables must comply with the following rules:

AT<Address>:

<identifier>AT<address>:<data type>{:=<initialization value>};

{} is an optional part.

Start with "%", followed by the position prefix symbol and size prefix symbol. If there is a classification, use an integer to represent the classification, and use the decimal symbol "." to represent it, such as %IX0.0, %QW0. The specific format of direct variable declaration is shown in the following figure:




Definition of positional prefix:

- I: Indicates input unit;
- Q: Indicates output unit;
- M: Indicates storage area unit.

The definition of size prefix is shown in the table below:

| Prefix symbol | Definition | Agreed data type |
|---|---|---|
| X | Bit | BOOL |
| B | Byte | BYTE |
| W | WORD | WORD |
| D | Double words (DWORD) | DWORD |
| L | Long words (LWORD) | LWORD |
| * | Internal variables without specific locations, automatically assigned by the system | |

This area can be resized based on actual hardware resources.

Example:

%IX3.2 Input area offset 3 bytes bit 2

%QW10 Output area offset 10 words

%MB20 Memory area offset 20 bytes

Var1 AT%ID48:DWORD; //Var1 variable is a doubleword type, mapped to the input area offset of 48 doubleword positions

### 5-1-2. PLC direct address storage area

| Area | Purpose | Size | Address range |
|---|---|---|---|
| I area (%I) 128KB | User usage area | 64KWords | %IW0-%IW65535 |
| Q area (%Q) 128KB | User usage area | 64KWords | %QW0-%QW65535 |
| M area (%M) 256KB | User usage area | 128KWords | %MW0-%MW131070 |

## 5-2. Variables

### 5-2-1. Overview

Variables can be defined in the definition section of POU or through the automatic declaration dialog box, or in the global variable list editor. Variable types can be identified through variable type keywords, such as VAR and END_VAR is used to identify variables defined between them as local variables.

Variable types include local variables (VAR), input variables (VAR_INPUT), output variables (VAR_OUTPUT), input-output variables (VAR_IN_OUT), global variables (VAR_GLOBAL), temporary variables (VAR_TEMP), static variables (VAR_STAT), constants (VAR_CONSTENT), hold variables (VAR_RETAIN), and persistent variables (VAR_PERSISTENT).

### 5-2-2. Variable definition

- Variable definition

**Text declaration**

```
{attribute 'qualified_only'}
VAR_GLOBAL PERSISTENT RETAIN
    M AT %MB0: BOOL:=TRUE;
    D AT %MW40000:ARRAY [0..5] OF WORD;
    HD AT %MW40006:ARRAY [0..1] OF LREAL;
END_VAR
```

**Table declaration**

| | 类别 | 名称 | 地址 | 数据类型 | 初值 | 注释 | 特性 |
|---|---|---|---|---|---|---|---|
| 1 | VAR_GLOBAL RETAIN PERSISTENT | M | %MB0 | BOOL | TRUE | | |
| 2 | VAR_GLOBAL RETAIN PERSISTENT | D | %MW40000 | ARRAY[0..5] OF WORD | | | |
| 3 | VAR_GLOBAL RETAIN PERSISTENT | HD | %MW40006 | ARRAY[0..5] OF LREAL | | | |

In the table declaration, various attributes of variables can be edited and set. The following table provides a specific explanation of the table declaration:

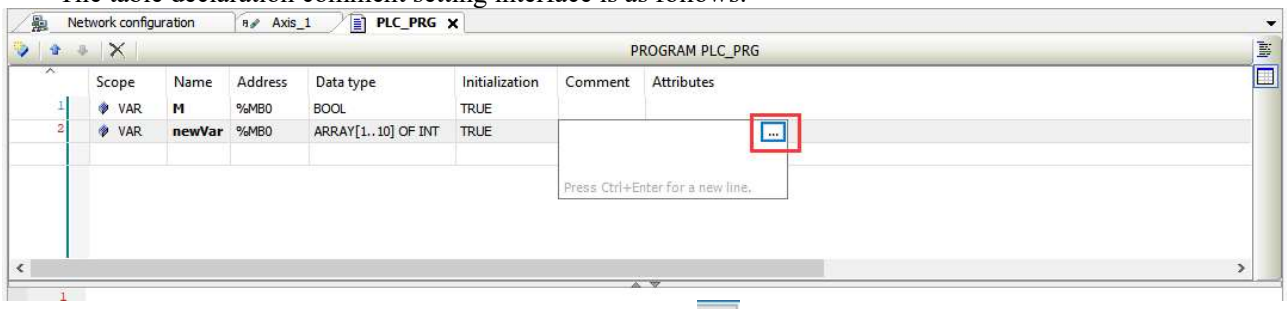
| Table declaration | Description |
|---|---|
| Type | Variable types (such as local variable (VAR), input variable (VAR_INPUT), output variable (VAR_OUTPUT) etc.) |
| Name | Variable name |
| Address | Variable mapping address |
| Data type | Variable data type (such as BOOL, INT etc.) |
| Initial value | Variable initial value |
| Comment | Variable comment |

| Feature | Variable features |
|---|---|

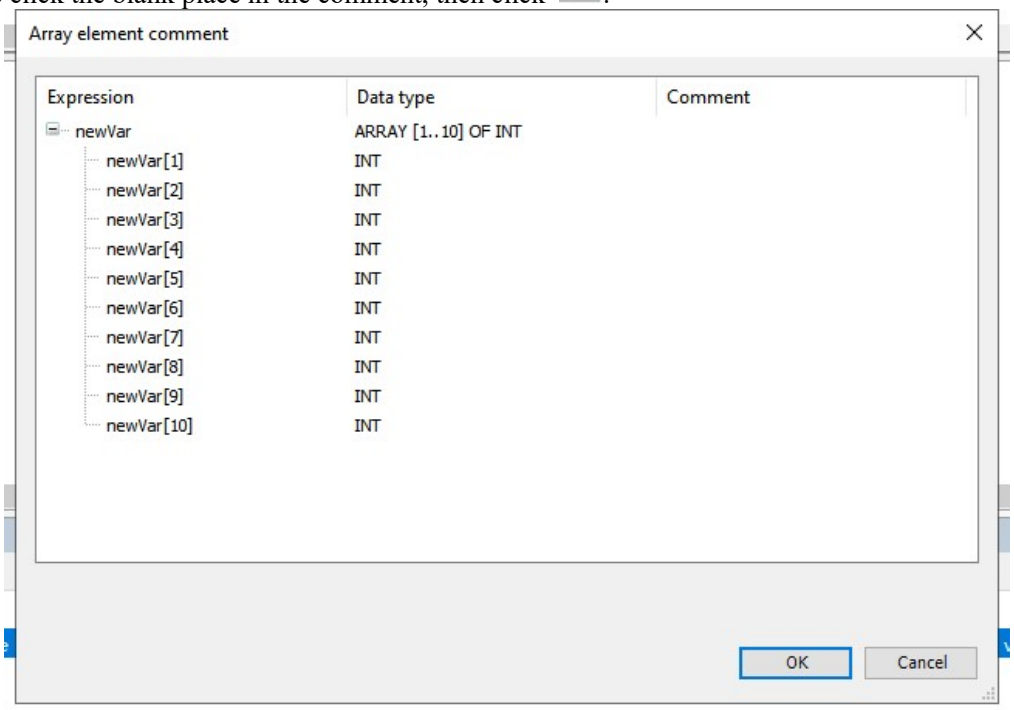
■ Variable definitions support array element comment and instance comment

1. Array element comment

The table declaration comment setting interface is as follows:



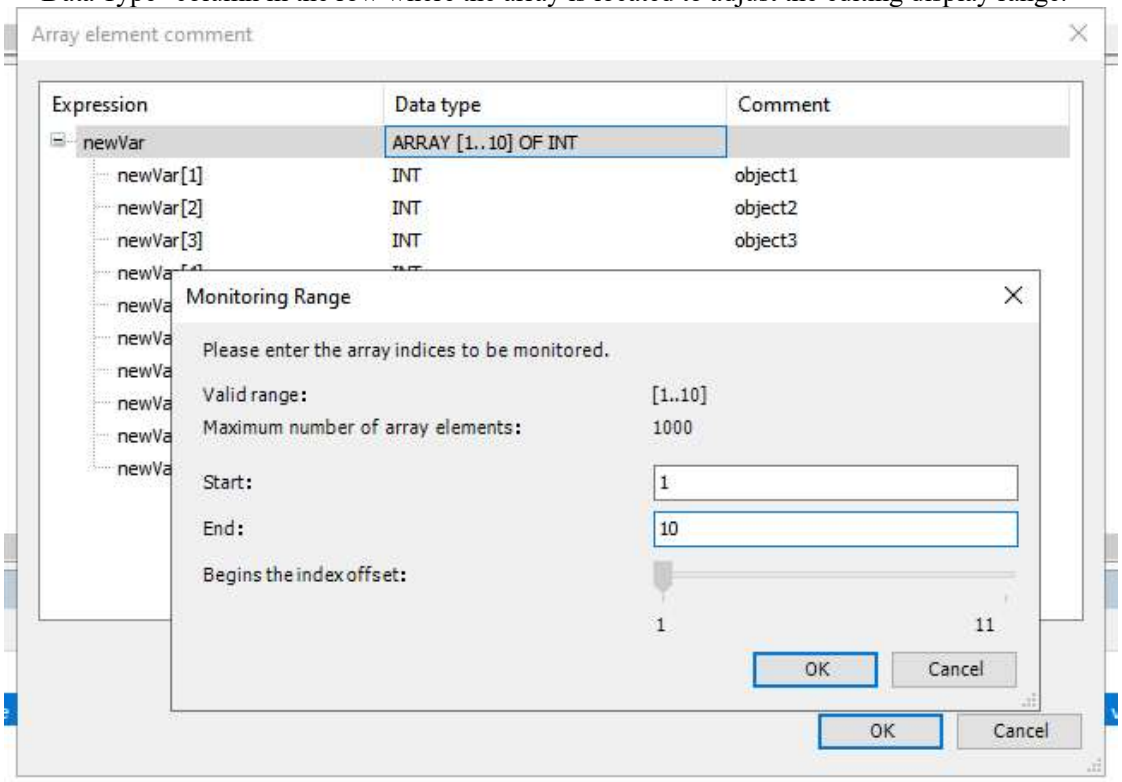

Double click the blank place in the comment, then click  .




After setting up, the text declaration effect is shown in the figure (it can also be declared directly using text):


- Array element comment editing can be done through tables and text.
- In the comment column of the table, a pop-up (similar to the initial value operation) displays the current element and sub element comment editing interface.
- The editing format of the text editor is as follows:
  - Array itself: Use standard comment editing methods.
  - Array elements: {attribute 'ElemComment':='1 (subelement 1 comment), 1 (subelement 2 comment), n (subelement comment)'}.
- If it is in table mode, when declaring array type variables, the default comment is empty (added attribute format by default).
- The array element comment in table mode only display the comment of the array itself, not the element comment.
- In the feature column, remove the element comment feature display (array element comments are implemented using attributes, which are information marked on variables).

- When the length of the table view array changes, the comments of the inventory array elements will also be saved accordingly.
- When the array dimension changes in the table view, the array element comments are translated and saved according to the minimum index of the extended dimension.
  - For example, array INT_ARRAY:ARRAY[1..2,2..3] dimension changed to ARRAY[1..2,2..3,3..4], the original array elements INT_ARRAY[1,2] comments will migrate to new array elements INT_ARRAY[1,2,3].
  - For example, INT_ARRAY:ARRAY[1..2,2..3] dimension changed to ARRAY[1..2], the original array elements INT_ARRAY[1,2] comments will migrate to new array elements INT_ARRAY[1].
- In the table view, when the data type is changed from array type to non array type, the array element comment will be cleared.
- The array element comment editing interface can display up to 1000 elements. Double click on the "Data Type" column in the row where the array is located to adjust the editing display range.




2. Instance comment

Variables declared in PRG (program) and GVL (global variable table) or declared as VAR_ STAT (static) type variables can expand internal member to edit comments without restrictions. When saving comments, all internal member comments will be marked on the variable, and this type of comment is called an instance comment of the variable.

As shown in the figure below, member comments within the data structure can be marked and saved on the variable array structure.

```
PROGRAM PLC_PRG
VAR
    // 结构体数组

    {attribute 'ElemComment':='([var3(成员3[(成员31),(成员32),3()]),
    newStruct([5()]),newStruct1([5()]),newStruct2([5()])]),4()'}
    newVar: ARRAY[1..5] OF struct1;

END_VAR
```

- When internal member is FB type, only input, output, and input-output variables will be displayed, and variables of other types will not be displayed.
- When changing the data type from non array type to array type in the table, the instance comment will be cleared.
- The array type members of variables can display up to 1000 elements, and the editing display range can be adjusted.

3. Comment display

In the initial value editing interface, monitoring variable table interface, ladder diagram, mouse hover display comment, and other functions related to variable comment display, comment display takes priority over instance comment. If the variable does not have instance comment, the type comment of the variable is displayed.

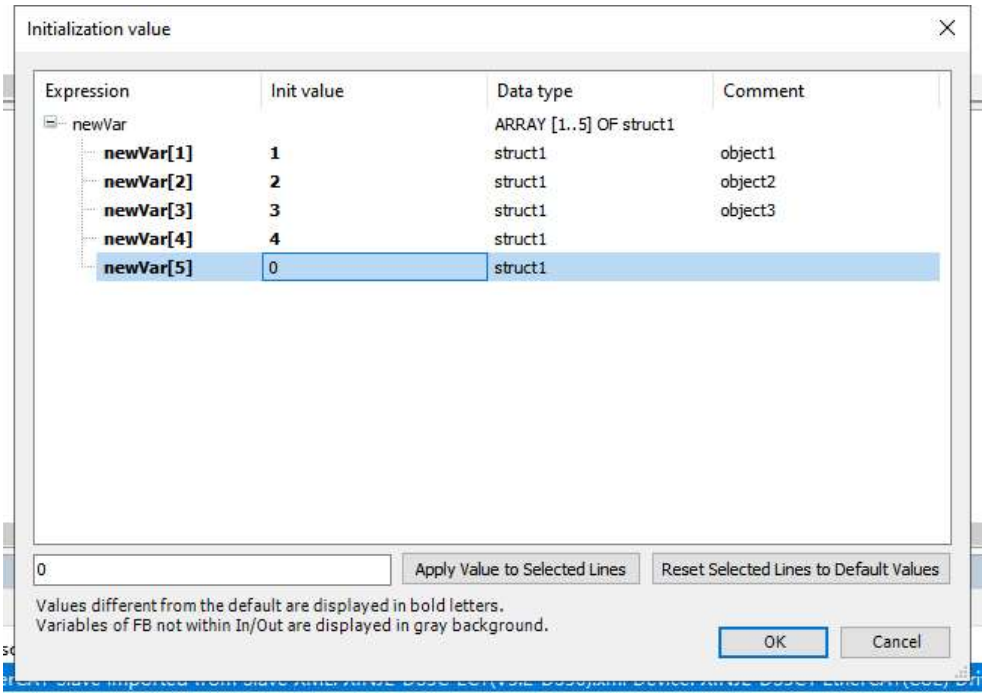
If the comment of array elements is involved in the ladder diagram, the comment of array elements should be merged and displayed; But the priority order rules above are also used for display. As shown in the following figure.


Instance comment:

■ Initial value setting of variables

1. Initial value setting

Click ... in the initial value column of the variable declaration.




2. It can also be declared directly in text form


■ Display address information for sub elements defined by variables

Click ... in the address column of the variable declaration, open the variable address setting window. The address display interface is as follows:

**Note:**
① The address column text box is read-only;
② Maximum number of display elements, with a maximum display of 1000 elements in the array, adjustable display range.

## 5-2-3. Variable type

Variable types include local variables (VAR), input variables (VAR_INPUT), output variables (VAR_OUTPUT), input-output variables (VAR_IN_OUT), global variables (VAR_GLOBAL), temporary variables (VAR_TEMP), static variables (VAR_STAT), and configuration variables (VAR_CONFIG).

Variable type declaration syntax: <TYPE> | Atribute
variable1;
variable2;
...
END_VAR

TYPE: type keyword, including VAR (local variable), VAR_INPUT (input variable), VAR_OUTPUT (output variable), VAR_IN_OUT (input output variable), VAR_GLOBAL (global variable), VAR_TEMP (temporary variable), VAR_STAT (static variable), VAR_CONFIG (configuration variable).
Attribute: Attribute keywords, including RETAIN, PERSISTENT, CONSTANT, used to specify the range of variables.

- Variable types

| Variable type keywords | Variable Properties | External read and write | Internal Read and Write |
|---|---|---|---|
| VAR | Local variable | - | R/W |
| VAR_INPUT | Input variables, provided externally | R/W | R |
| VAR_OUTPUT | Output variables, with internal variables provided to external sources | W | R/W |
| VAR_IN_OUT | Input output variables | R/W | R/W |
| VAR_GLOBAL | Global variables that can be used within all configurations and resources | R/W | R/W |
| VAR_TEMP | Temporary variables, variables stored and used within programs and functional blocks | - | R |
| VAR_STAT | Static variable | | |
| VAR_EXTERNAL | External variables that can be modified within the program, but must be provided by global variables | R/W | R/W |

VAR, VAR_INPUT, VAR_OUTPUT and VAR_IN_OUT is the most commonly used type of variable in program organizational units (POUs).
VAR_GLOBAL global variables also need to be widely used in practical engineering projects.

- Variable Properties

| Variable Additional Attribute Keywords | Variable Additional Attribute |
|---|---|
| RETAIN | Holding type variable, used for power-off holding |
| PERSISTENT | Maintaining variables |
| VAR RETAIN PERSISTENT<br>VAR PERSISTENT RETAIN | Both have the same function, both are maintenance variables used for power-off maintenance |
| CONSTANT | Constant |

- **RETAIN**

Declare type variables with the keyword RETAIN. RETAIN type variables can maintain their original values even after the controller is normally closed, opened (or receives an online command "hot reset"), or even unexpectedly closed. As the program starts running again, the stored values can continue to function.

RETAIN type variable declaration format is as follows:

VAR RETAIN

< Identifier >:<Data type>;

END_VAR

But the RETAIN variable will be reinitialized after the "initial value bit", "cold reset", and program download.

Memory storage location: RETAIN type variables are only stored in a separate memory area.

In practical engineering applications, such as the piece counter on the production line, it is a typical example: after the power is cut off, it can still continue counting when restarted. And all other variables will be reinitialized at this time, becoming the specified initial value or standard initialized value.

- **PERSISTENT**

At present, only a few PLCs still retain independent memory areas for storing PERSISTENT type data. In XS Studio, its original function of power-off retention has been cancelled, and instead it is implemented through VAR RETAIN PERSISTENT or VAR PERSISTENT RETAIN, which are completely identical in function.

PERSISTENT type variable declaration format is as follows:

VAR GLOBAL PERSISTENT RETAIN

< Identifier >:< Data type >;

END_VAR

Memory storage location: Like the RETAIN variable, the RETAIN PERSISTENT and PERSISTENT RETAIN variables are also stored in a separate memory area.

- **CONSTANT**

A constant is a quantity that can only be read and cannot be modified during program execution, with the keyword CONSTANT. Constants can be declared as local or global constants.

CONSTANT declaration format is as follows:

VAR CONSTANT

< Identifier >:< Data type > := < Initialize value >;

END_VAR

In practical applications, important parameters or coefficients can usually be set as constants, which can effectively avoid other variables from modifying them and ultimately affect the overall stability and safety of the system. Here are some examples.

VAR CONSTANT

pi:REAL:= 3.1415926;

END_VAR

Once the program starts running, variables declared through CONSTANT are not allowed to be modified during the program's execution. Forcing system modifications can result in system errors.

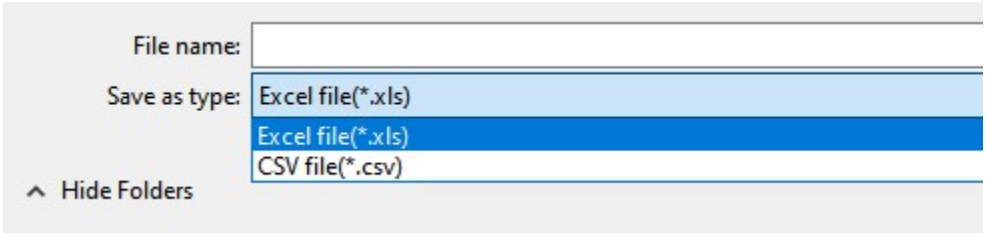
## 5-2-4. Variable import and export

Support variable import and export, export file type is XLS worksheet (. xls), presented in Excel spreadsheet form, can be added, deleted, or other variables editing externally before being imported into XS Studio programming software.

As shown in the following figure:

| | 类别 | 名称 | 地址 | 数据类型 | 初值 | 注释 | 特性 |
|---|---|---|---|---|---|---|---|
| 1 | VAR_GLOBAL | **newVarGlobal** | | BOOL | | | |
| 2 | VAR_GLOBAL CONSTANT | **newVarGlobal1** | | INT | | | |
| 3 | VAR_GLOBAL CONSTANT RETAIN | **newVarGlobal2** | | byte | | | |
| 4 | VAR_GLOBAL PERSISTENT | **newVarGlobal3** | | WORD | 100 | | |
| 5 | VAR_GLOBAL RETAIN PERSISTENT | **newVarGlobal4** | | DWORD | | TEST | |
| 6 | VAR_GLOBAL | **newVarGlobal5** | | UINT | | | |

Add some variables to the variable table and right-click to select the export variable table type Excel/CSV (CSV is a pure text file, Excel contains formatting information. CSV files are small, easy to create, distribute, and read, and are suitable for storing structured information. CSV files are opened in Excel by default on the Windows platform, which is essentially a text file), and there is no difference in the editing of variables between the two formats.




Open the exported file and edit (add new variables VAR1, VAR2, VAR3, VAR4) before importing. The effect is shown in the following figure:

| Type | Name | Address | DataType | InitValue | Comment | Attribute |
|---|---|---|---|---|---|---|
| VAR_GLOBAL | newVarGlobal | | BOOL | | | |
| VAR_GLOBAL CONSTANT | newVarGlobal1 | | INT | | | |
| VAR_GLOBAL CONSTANT RETAIN | newVarGlobal2 | | BYTE | | | |
| VAR_GLOBAL PERSISTENT | newVarGlobal3 | | WORD | 100 | | |
| VAR_GLOBAL RETAIN PERSISTENT | newVarGlobal4 | | DWORD | | TEST | |
| VAR_GLOBAL | newVarGlobal5 | | UINT | | | |
| | | | | | | |
| VAR_GLOBAL | VAR1 | | INT | 10 | | |
| VAR_GLOBAL | VAR2 | | BYTE | | 导入 | |
| VAR_GLOBAL | VAR3 | | WORD | | | |
| VAR_GLOBAL | VAR4 | | REAL | | | |

| | 类别 | 名称 | 地址 | 数据类型 | 初值 | 注释 | 特性 |
|---|---|---|---|---|---|---|---|
| 1 | VAR_GLOBAL | **newVarGlobal** | | BOOL | | | |
| 2 | VAR_GLOBAL CONSTANT | **newVarGlobal1** | | INT | | | |
| 3 | VAR_GLOBAL CONSTANT RETAIN | **newVarGlobal2** | | BYTE | | | |
| 4 | VAR_GLOBAL PERSISTENT | **newVarGlobal3** | | WORD | 100 | | |
| 5 | VAR_GLOBAL RETAIN PERSISTENT | **newVarGlobal4** | | DWORD | | TEST | |
| 6 | VAR_GLOBAL | **newVarGlobal5** | | UINT | | | |
| 7 | VAR_GLOBAL | **VAR1** | | INT | 10 | | |
| 8 | VAR_GLOBAL | **VAR2** | | BYTE | | 导入 | |
| 9 | VAR_GLOBAL | **VAR3** | | WORD | | | |
| 10 | VAR_GLOBAL | **VAR4** | | REAL | | | |

## 5-3. Power outage holding variable

### 5-3-1. PERSISTENT

The power-off retention variable retains its original value after PLC power-off or program download, and is commonly used to define important parameters in engineering to prevent the loss of important parameters caused by sudden PLC power-off or program download.

Power failure retention can be declared through the attribute keyword PERSISTENT RETAIN, or it can be implemented by mapping to the M power failure retention area.

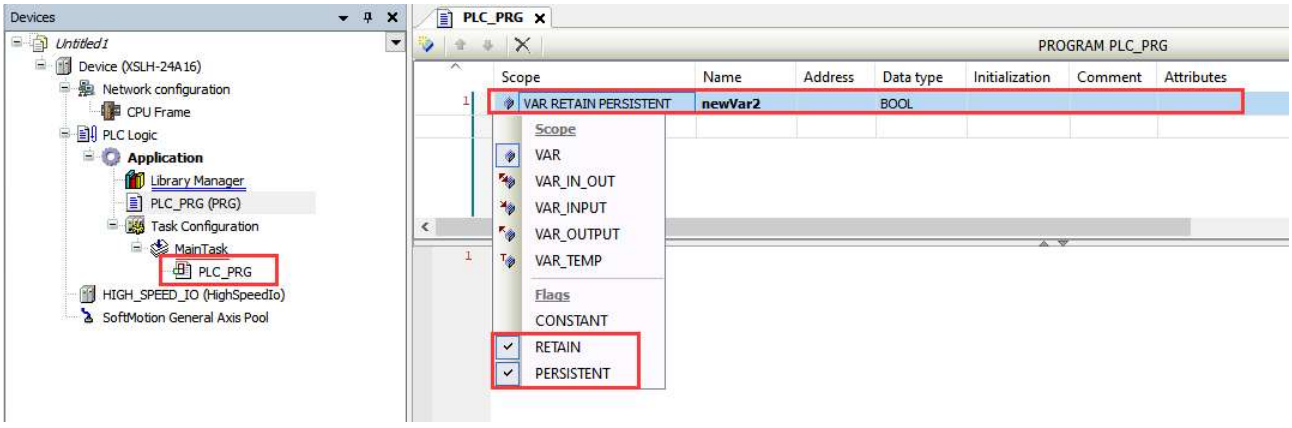
The command behavior table for retained variables online is as follows:

| Online command | VAR | VAR RETAIN | VAR PERSISTENT RETAIN | M retained area |
|---|---|---|---|---|
| Hot reset | Initial value | Retention value | Retention value | Retention value |
| Cold reset | Initial value | Initial value | Retention value | Retention value |
| Origin reset | Initial value | Initial value | Initial value | Initial value |
| Build | Initial value | Initial value | Retention value | Retention value |
| Online change | Retention value | Retention value | Retention value | Retention value |
| Rebuild | Initial value | Retention value | Retention value | Retention value |

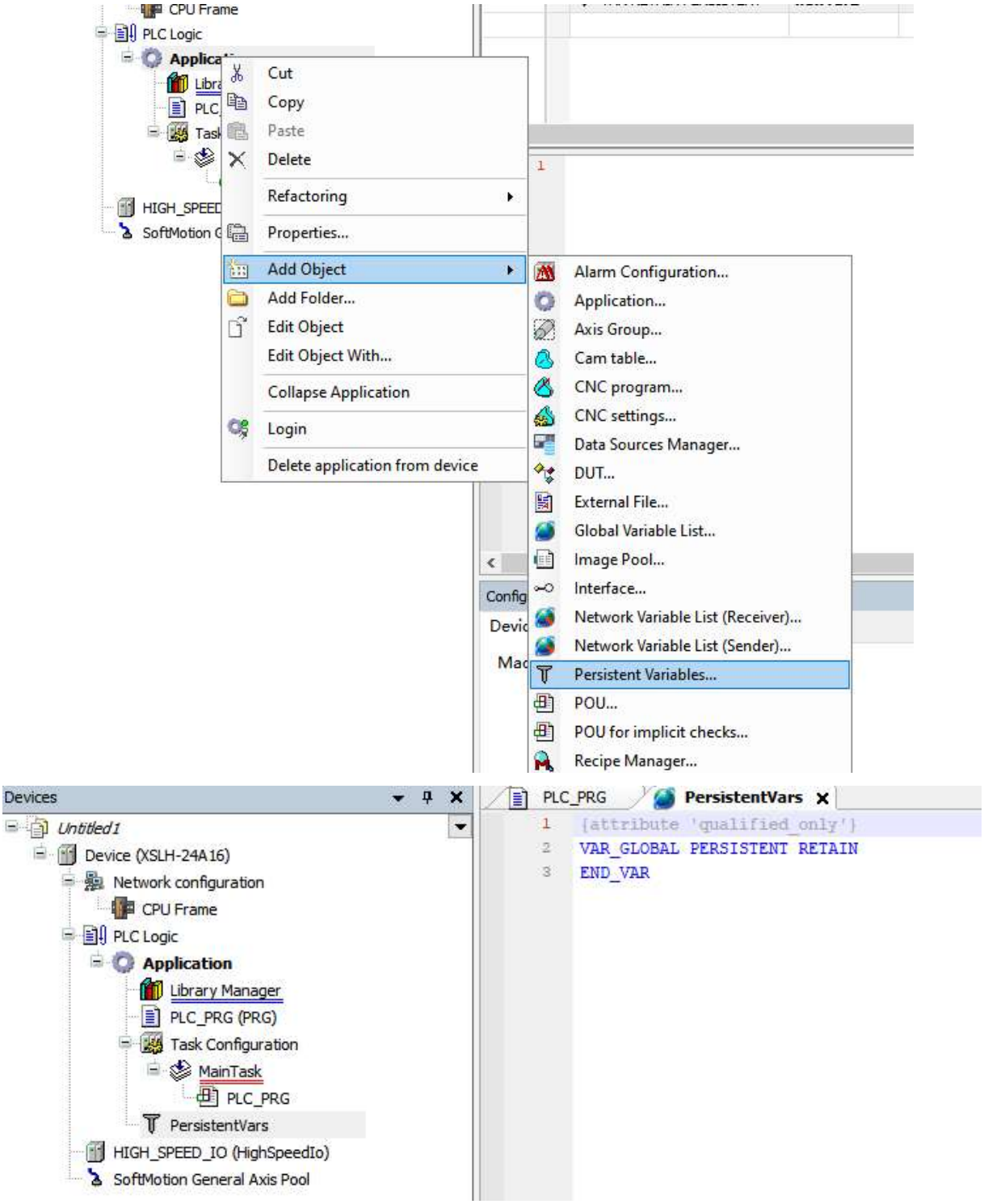
Note: Using PERSISTENT RETAIN for variable declaration has the same effect as using RETAIN PERSISTENT or PERSISTENT.

- Set Persistent properties

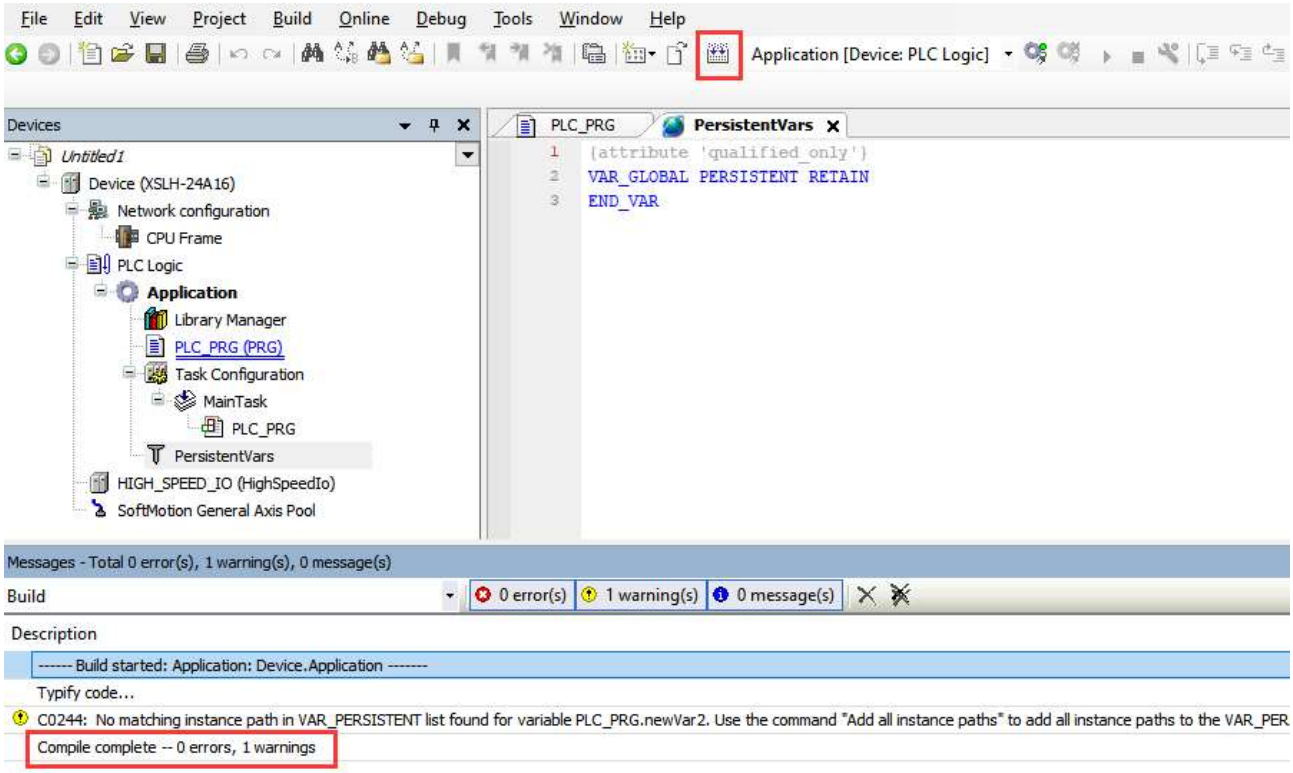
1. In PLC_PRG editor variable scope column select "RETAIN PERSISTENT", as shown in the figure;




2. Right click on "Application" in the left device tree, click "Add Object", and select "Persistent Variable" from the menu item. After adding, it will generate a persistent variable node in the device tree. As shown in the figure below

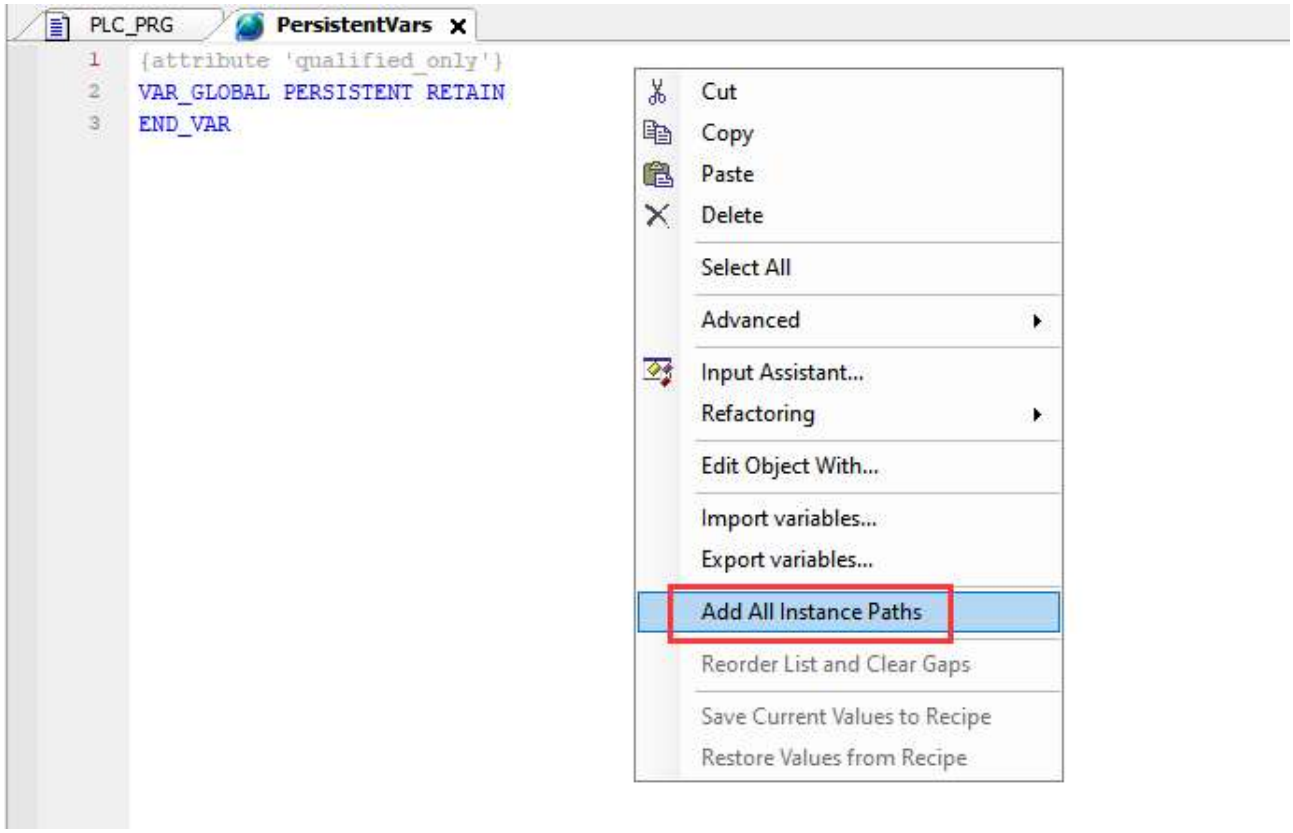



3. Click compile to check for errors in the project, and only after compiling can you proceed to the next step of adding instance paths to the object. As shown in the following figure:




As shown in the above figure, after compilation, the user will be prompted in the information output column to add an instance path.

4. Open the PersistentVars editing interface, right-click and select the "Add All Instance Paths" menu item, which will generate instance paths for all variables with the category "PERSISTENT RETAIN" in the "PLC_PRG" interface. As shown in the following figure:

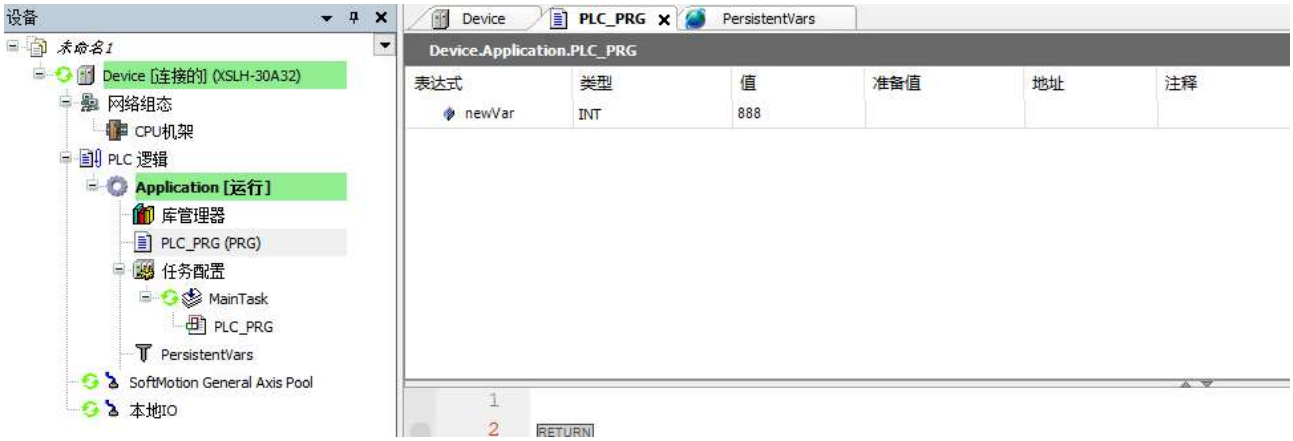


```
PLC_PRG    PersistentVars
1  {attribute 'qualified_only'}
2  VAR_GLOBAL PERSISTENT RETAIN
3      // Generated instance path of persistent variable
4      PLC_PRG.newVar2: BOOL;
5  END_VAR
```

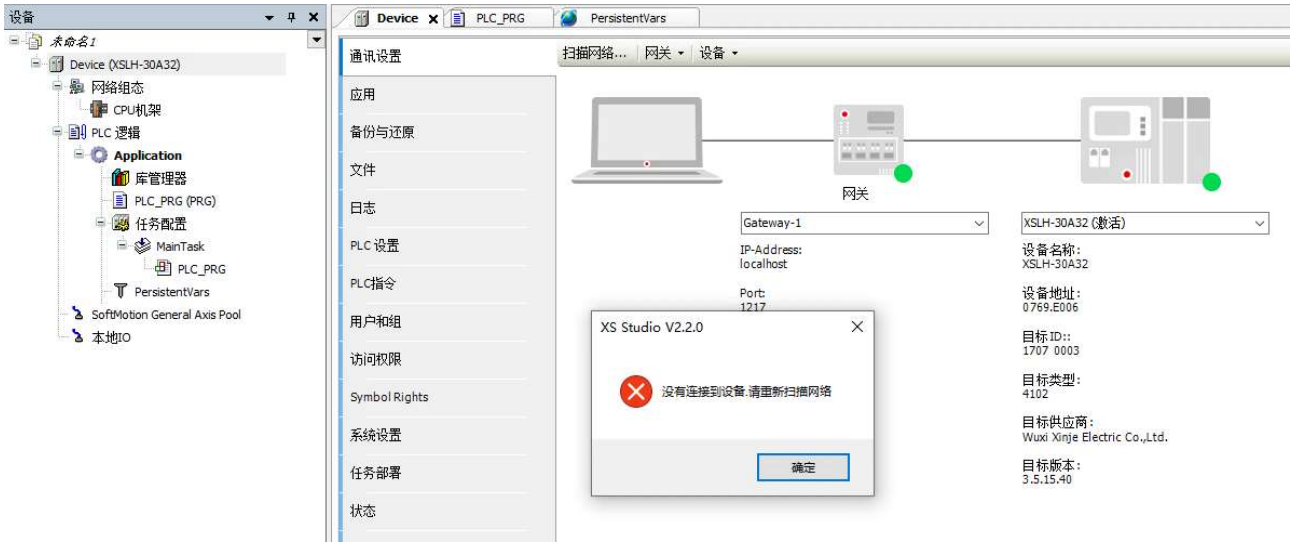
Only after adding the instance path in the PersistentVars object can the variable's retention property take effect.

5. Connect the PLC device, assign a value to the variable newVar after power on, and power on after abnormal power off. The newVar variable still maintains its value before power off. As shown in the following figure:

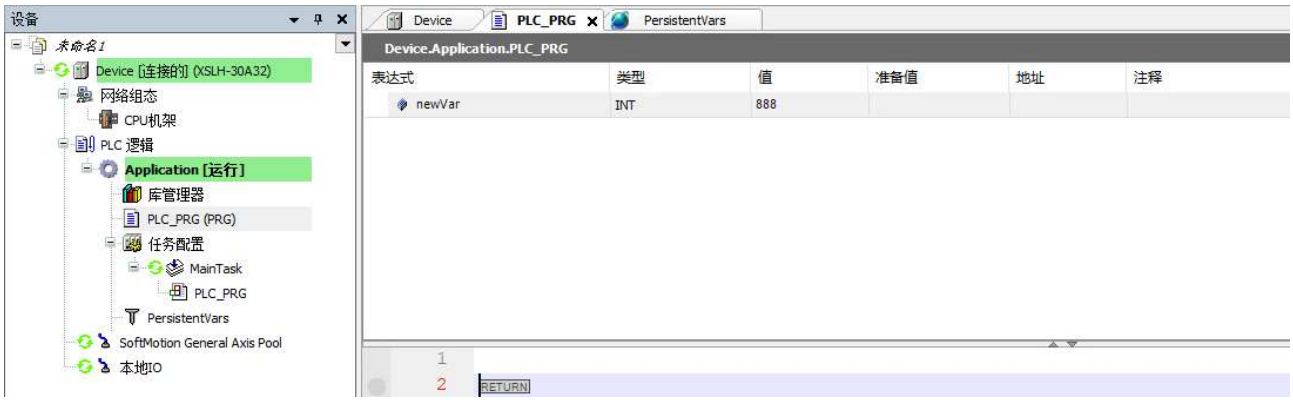
First power on and assignment.




Power on after abnormal power off:




The data did not change when powered on again, and the retained variable ran successfully.



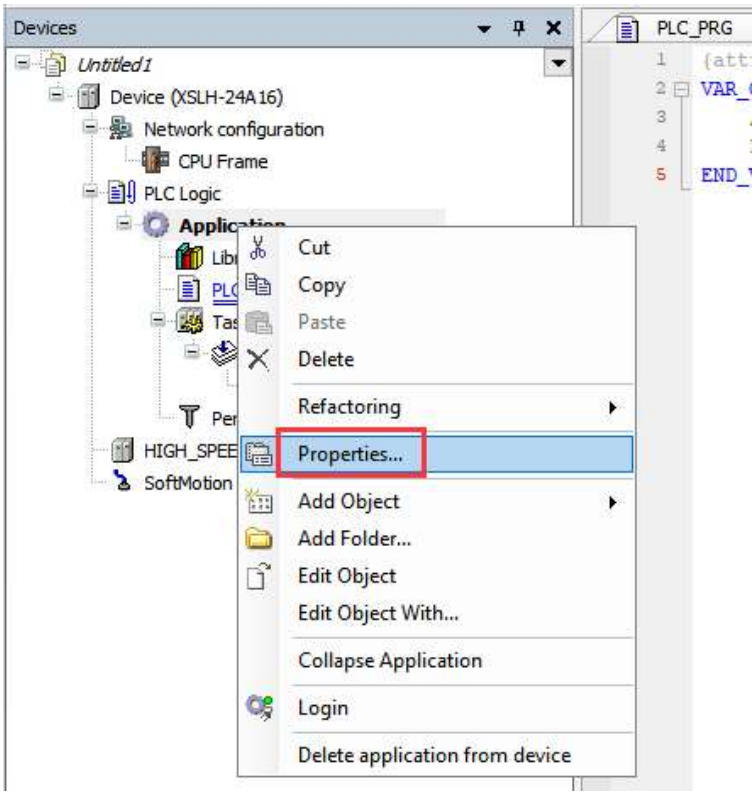
## 5-3-2. M retained area

Mapping the address to the variable in the M power-off storage area can prevent the impact of power failures. The variable value can still be retained after restarting.

Attention: When creating a new project, the default allocation of the M-zone power outage storage area is not allowed. Users can customize the power outage retention range for the M-zone according to their actual needs.

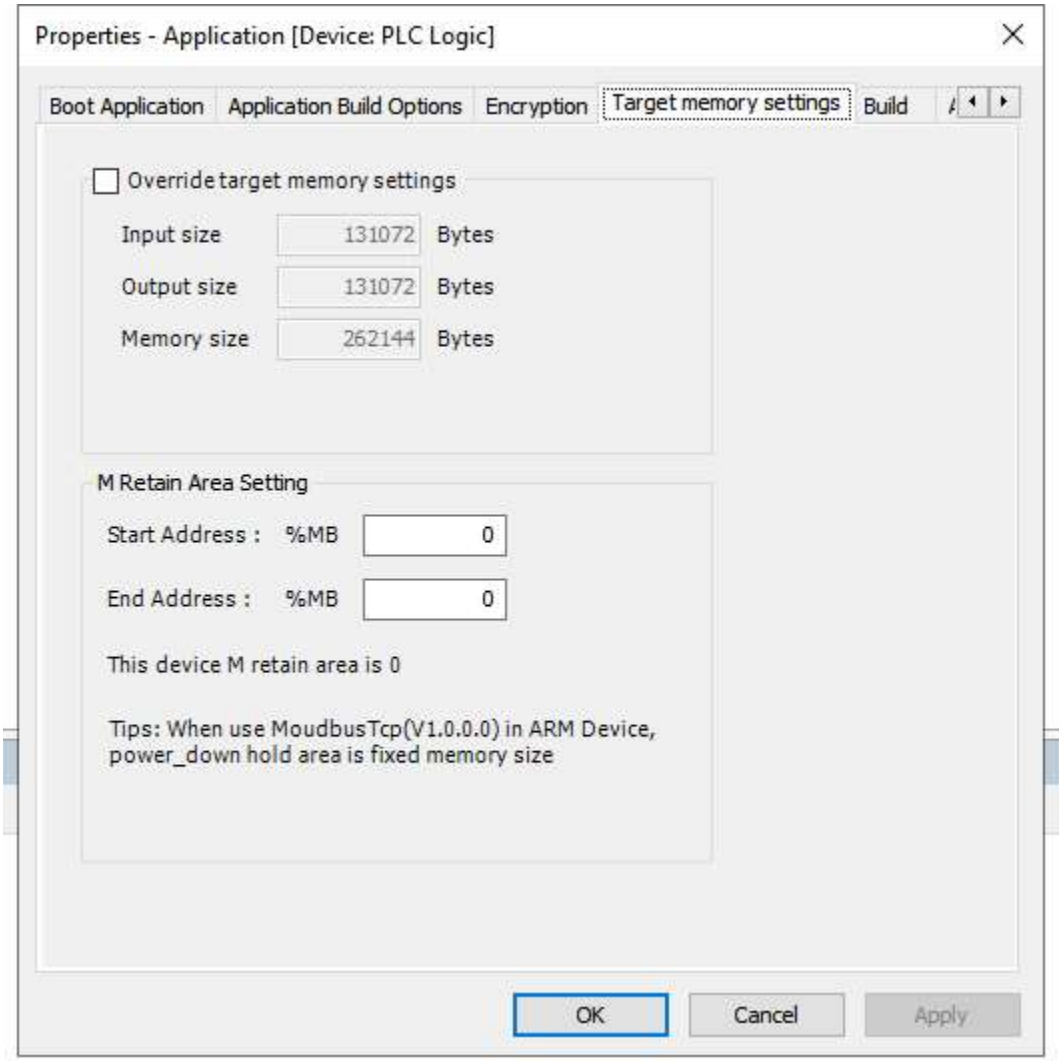
- Set M retained variable

1. Right-click the "Application" menu item in the left device tree, select the Properties menu, and open the "Properties Application [Device: PLC Logic]" window;




2. Select the "Target Memory Settings" menu to set the start or end address of the power outage in the power outage storage area.

3. Within the memory size range of [0-262143], the power-off storage area can be set as needed. As shown in the following figure:




**Note:**

① If the starting address of M retained area is "%MB50" and the ending address is "%MB100"; The non power-off holding area is "%MB0-%MB49" and "%MB101-%MB262143".

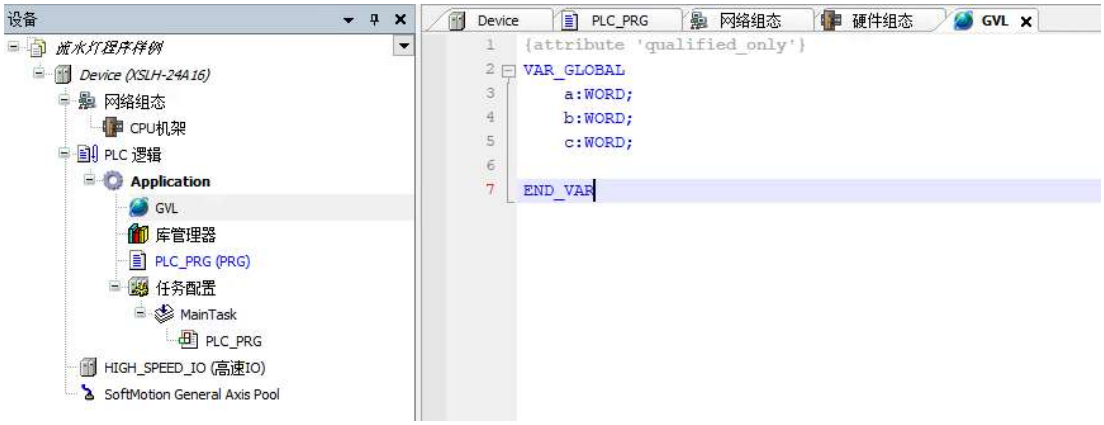
② If the mapping address of a variable occupies non power-off hold area and power-off hold area across regions, for example, if the variable address is "%MB49-%MB58", an error will be reported in the information output column during compilation: the variable address exceeds the limit, the variable address is "%MB49-%MB58", and the actual available range is "%MB0-% MB49" or "% MB101-% MB262143".

## 5-4. Recipe operation

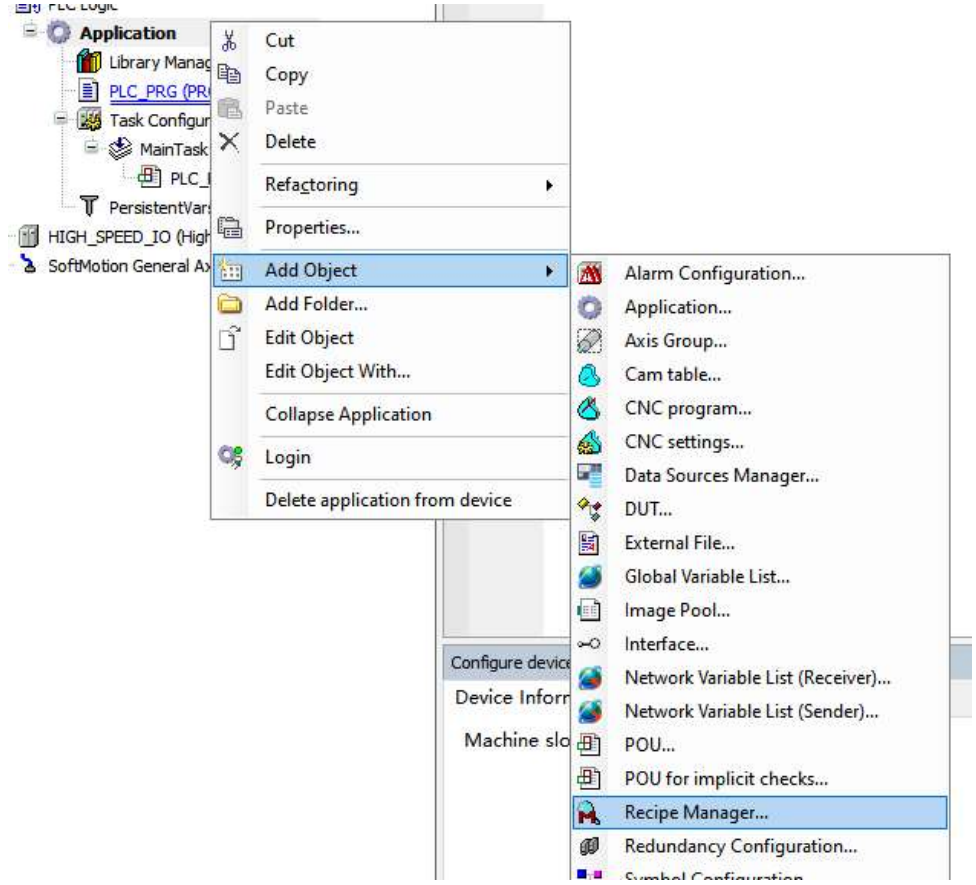
The function of the Recipe Manager is to provide a list of user-defined variables (recipe definitions) for maintenance. Users can configure the storage location, storage method, and storage category through the Recipe Manager, as shown in the following figure. After the recipe manager is successfully configured, users can upload and download recipe definitions.

### 5-4-1. Application example

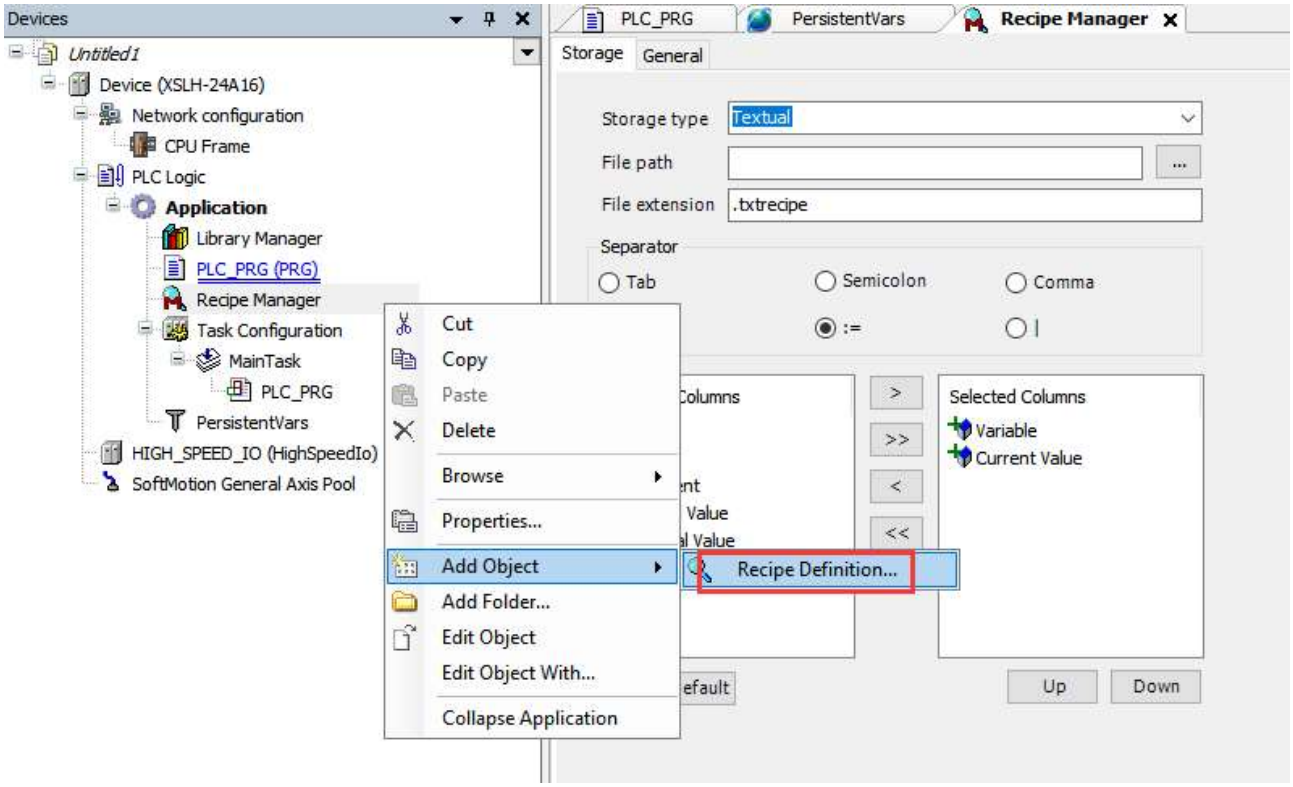
1. Create variable



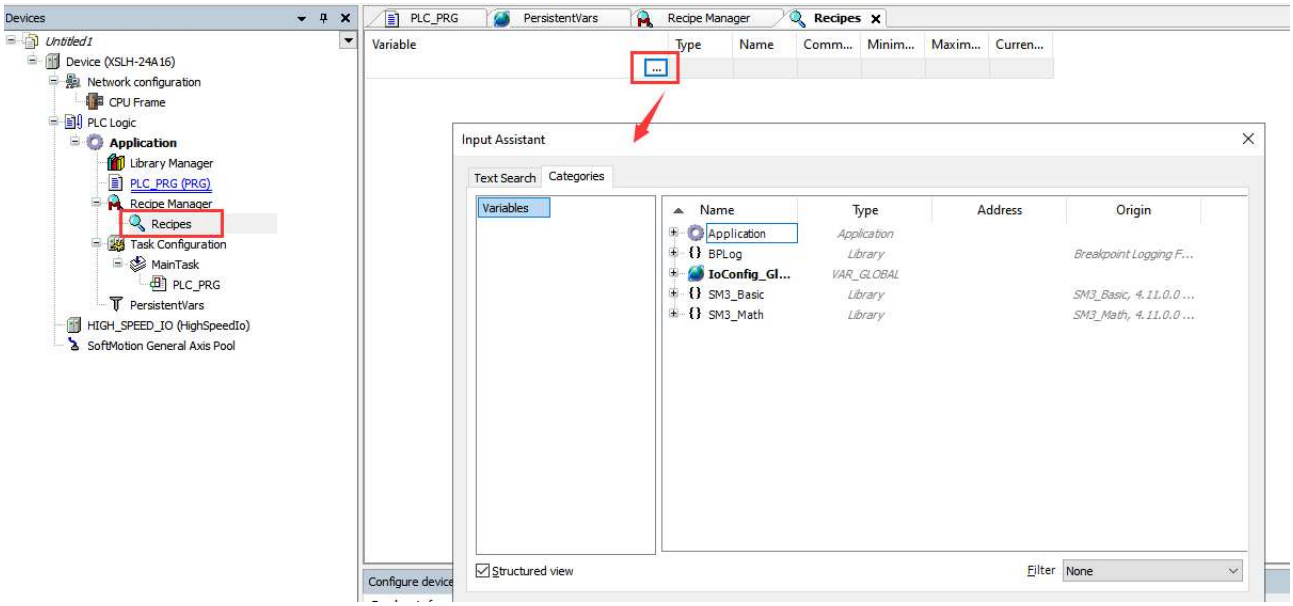

2. Create recipe manager



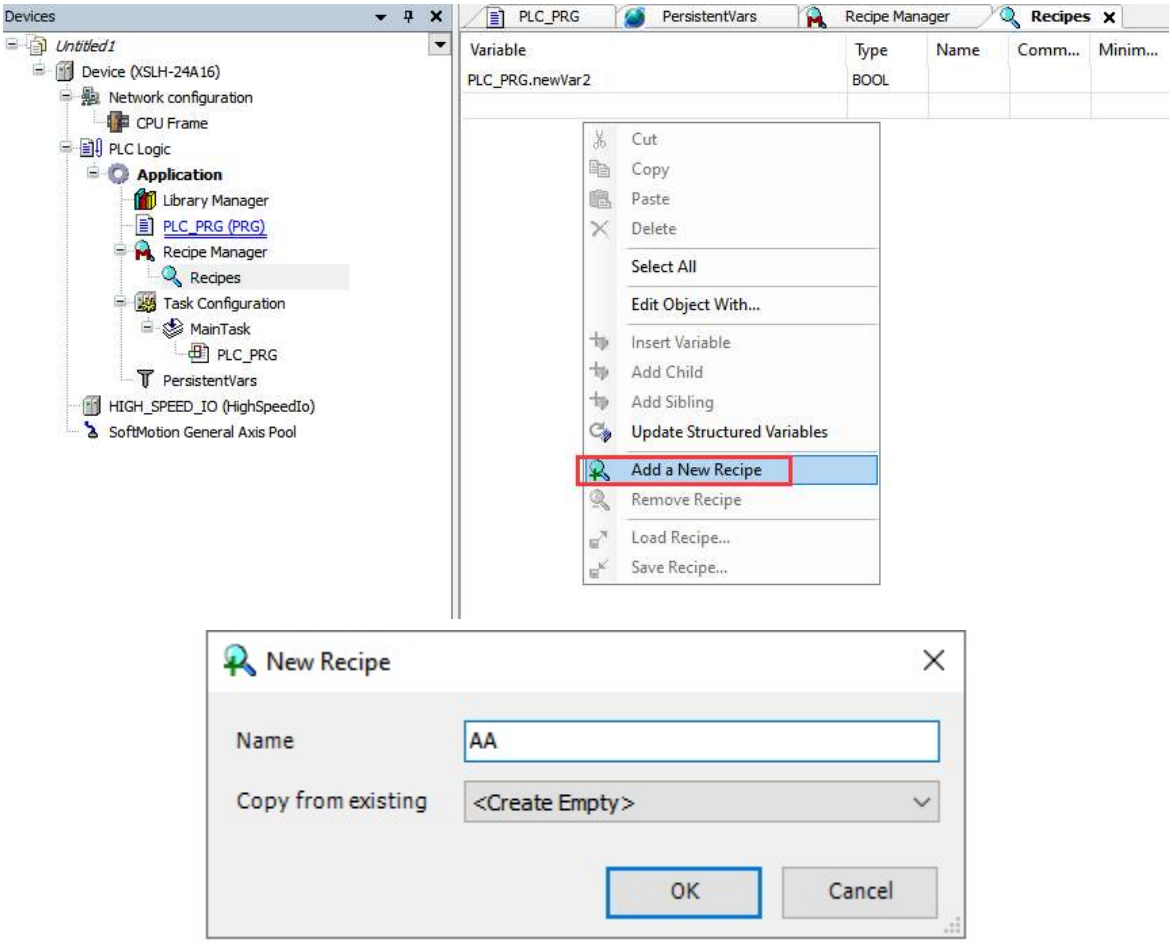
3. Create recipe definition



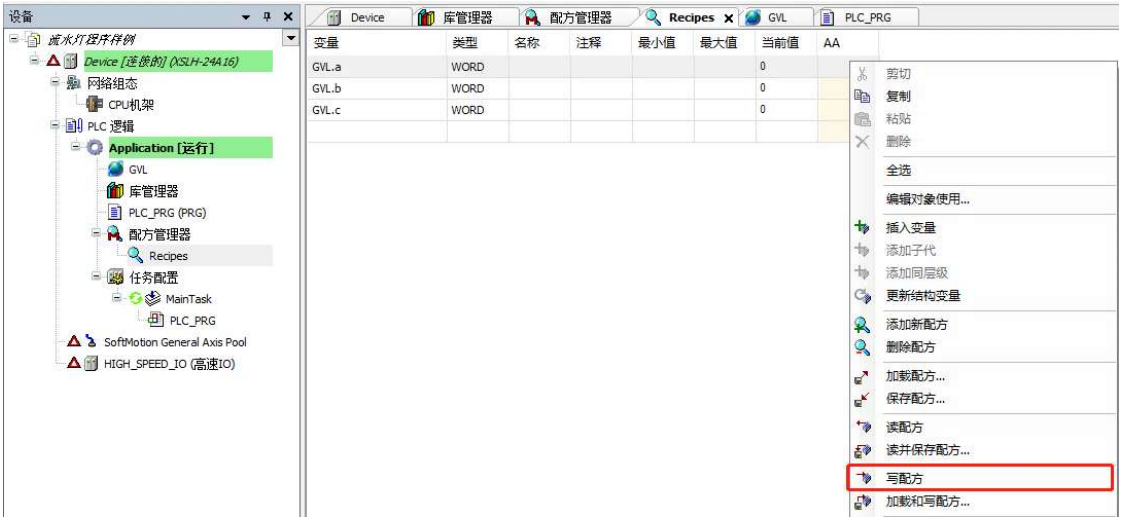

4. Select variables that require the use of recipe functionality




5. Add recipe




6. Login to the device and right-click on the table at the bottom of the corresponding recipe to perform operations such as writing and reading the recipe.




7. Instructions can also be used for recipe creation, reading, writing, and other operations.

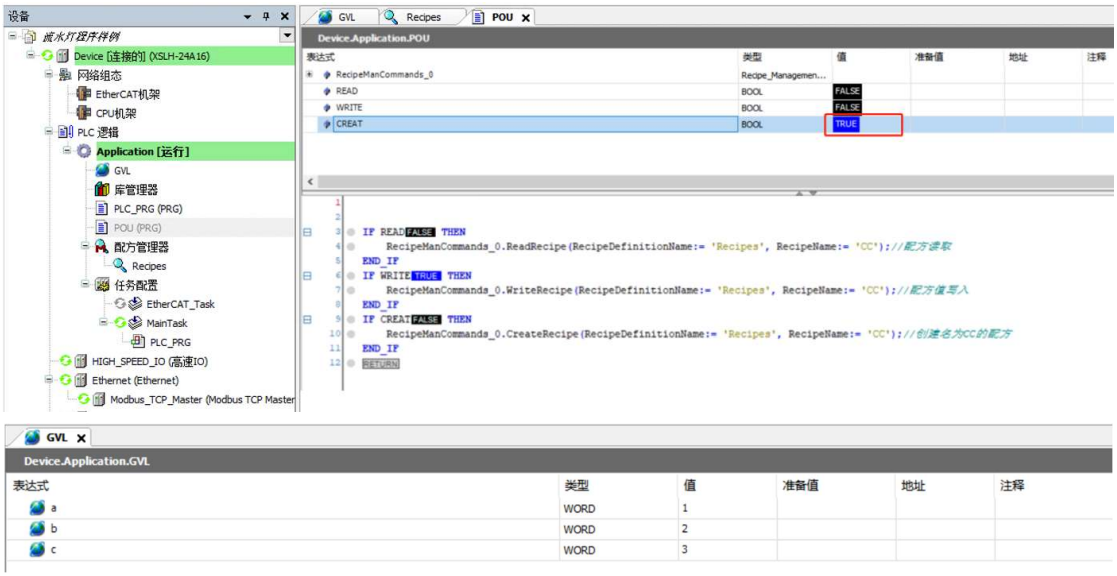
①Write a program for creating, reading, and writing recipes

Example:

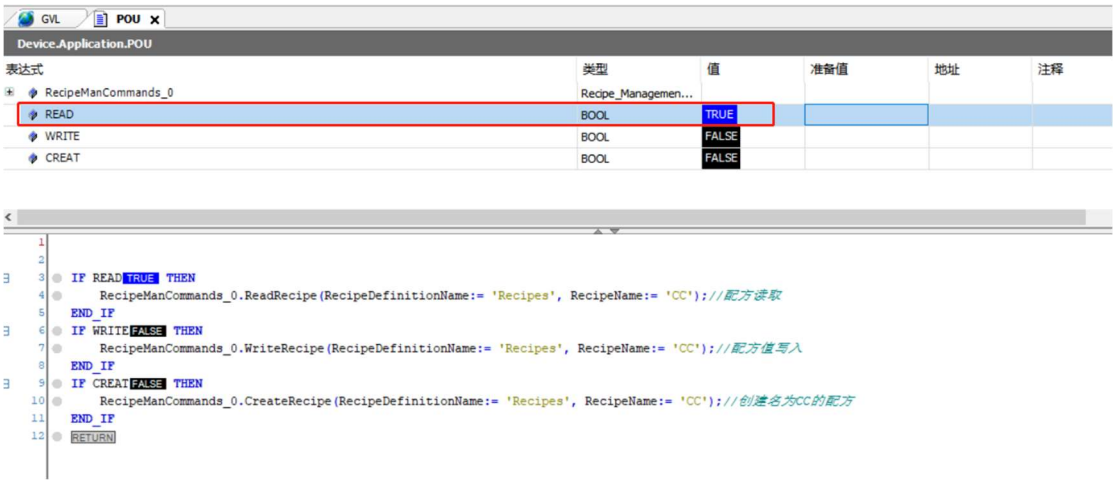
```
VAR
    RecipeManCommands_0:Recipe_Management.RecipeManCommands;
    READ:BOOL;
    WRITE:BOOL;
    CREAT:BOOL;
```

```
END_VAR

IF READ THEN
    RecipeManCommands_0.ReadRecipe(RecipeDefinitionName:= 'Recipes', RecipeName:= 'CC');//read recipe
END_IF
IF WRITE THEN
    RecipeManCommands_0.WriteRecipe(RecipeDefinitionName:= 'Recipes', RecipeName:= 'CC');//write recipe
END_IF
IF CREAT THEN
    RecipeManCommands_0.CreateRecipe(RecipeDefinitionName:= 'Recipes', RecipeName:= 'CC');//create a recipe named CC
END_IF
```

Login the created recipe.

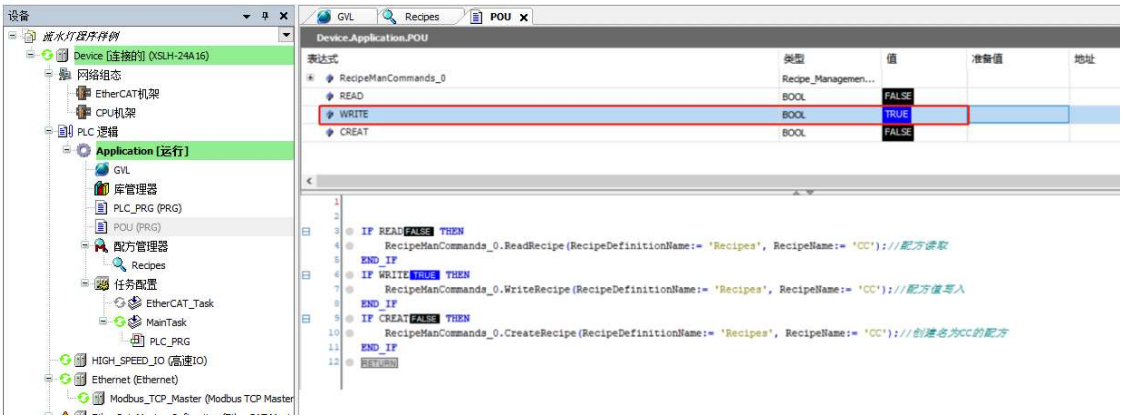



② Read the value to recipe

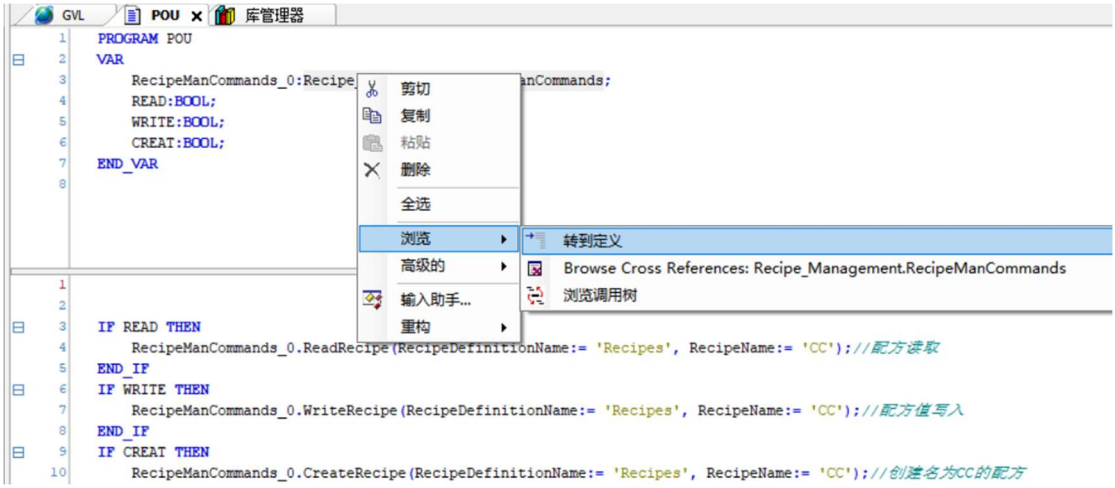


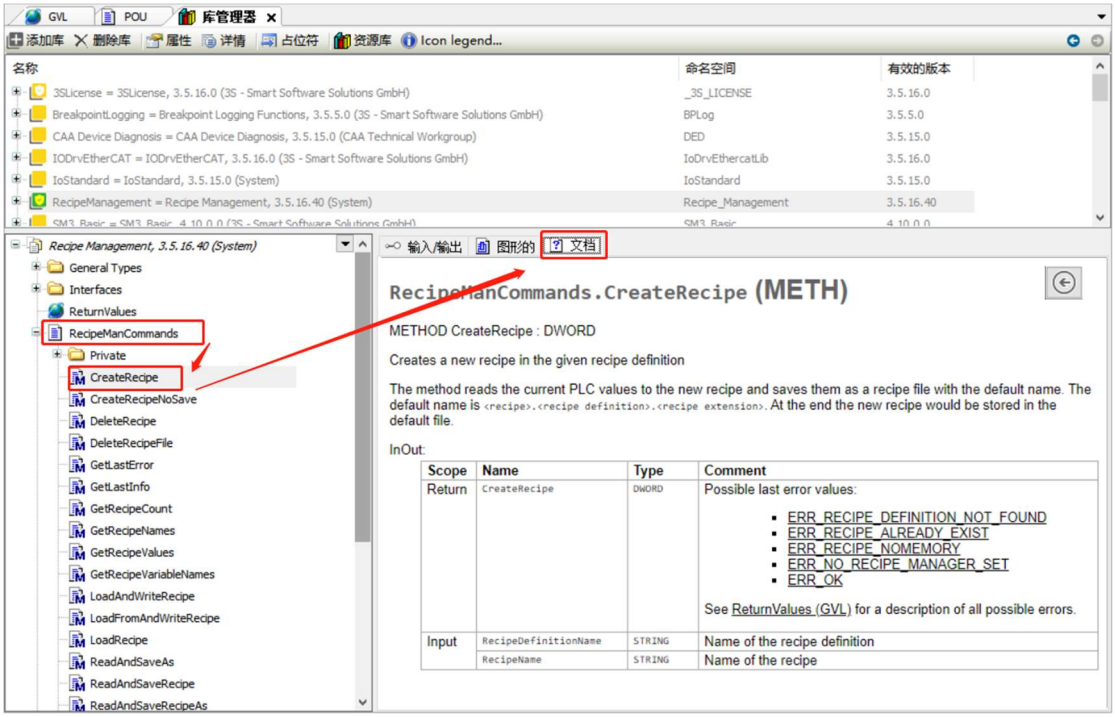

③ Change the current value to other value

④ Write recipe value




Note: Other functions of the recipe can be found and used in the definition



# 6. Programming language

## 6-1. XS Studio supported language

PLC programming languages supported by XS Studio programming software:

- Ladder diagram(LD)
- Function block diagram(FBD)
- Structured text (ST)
- Sequential function chart (SFC)
- Continuous function chart (CFC)

All the above languages support standard Ctrl and Shift editor shortcut keys in the editor interface. Shortcuts such as copy (Ctrl+C), paste (Ctrl+V), and undo (Ctrl+Z); Simultaneously supporting shortcut keys<F2>to start the input assistant, the system will provide corresponding input prompts or choices based on the current programming environment.

## 6-2. Structured text (ST)

### 6-2-1. Overview

Structured Text (ST) is an advanced text language that can be used to describe the behavior of functions, blocks, and programs. It can also describe the behavior of steps, actions, and transitions in sequential functional flowcharts. Structured text programming language is a high-level language, similar to Pascal, developed specifically for industrial control applications. It is also the most commonly used language in XS Studio. For those familiar with computer high-level language development, structured text language is easy to learn and use, as it can achieve functions such as selection, iteration, and jump statements. In addition, structured text languages are also easy to read and understand, especially when annotated with meaningful identifiers and annotations. In complex control systems, structured text can greatly reduce its code volume, making complex system problems simpler. The disadvantage is that debugging is not intuitive and compilation speed is relatively slow.
For example

```
FOR a:=0 TO 0 BY 1 DO
    D_temperature display value[a] :=TO_REAL(D_ temperature actual value [a]) / 10;
    D_temperature final value[a] := D_ temperature display value [a] + D_ temperature compensation value [a];
END_FOR

IF M_auto-tune switch THEN
    M_temperature control mode[0]:= 1;
END_IF
```

## 6-2-2. ST program execution sequence

1. Program execution sequence

The execution order of the program using structured text starts from top to bottom according to the "line number", as shown in the following figure:


2. Expression execution order

The expression includes operators and operands, which operate according to the rules specified by the operator to obtain the result and return it. Operands can be variables, constants, register addresses, functions, etc.

a+b+c;

3.14*R*R;

ABS(-10)+var1;

If there are several operators in the expression, they will be executed in the agreed priority order: the operator with higher priority will be executed first, and the operator with lower priority will be executed in order. If there are operators with the same priority in the expression, these operators are executed from left to right in writing order. The priority of operators is shown in the table below:

| Operator | Symbol | Priority |
|---|---|---|
| Parentheses | () | Highest |
| Function call | Function name (Parameter list) | |
| Exponentiation | EXPT | |
| Inversion | NOT | |
| Multiplication<br>Division<br>Mold taking | *<br>/<br>MOD | |
| Addition<br>Subtraction | +<br>- | |
| Compare | <,>,<=,>= | |
| Equal<br>Not equal | =<br><> | |
| Logical and | AND | |
| Exclusive-OR | XOR | |
| Logical or | OR | Lowest |

## 6-2-3. Statement

The structured text statements are shown in the following table:

| Instruction type | Instruction statement | Example |
|---|---|---|
| Assignment statement | := | bFan:= TRUE; |
| Function block/Function Call | Function block/Function name(); | |
| Selection statement | IF | IF < Booleans > THEN<br><statement contents>;<br>END_IF |
| | CASE | |
| Iteration statement | FOR | |
| | WHILE | |
| | REPEAT | |
| Jump statement | EXIT | |
| | CONTINUE | |
| | JMP | |
| Return statement | RETURN | |
| NULL statement | ; | |

1. Assignment statement

It is one of the most commonly used statements in structured text, which assigns the value generated by the expression on the right to the operand (variable or address) on the left, represented by ":=".

< variable>:=< expression>;

Example: Assign values to two Boolean variables separately, set bFan to True and bHeater to FALSE

```
VAR
bFan: BOOL;
bHeater:BOOL;
END_VAR

bFan:= TRUE;
bHeater:= TRUE;
```

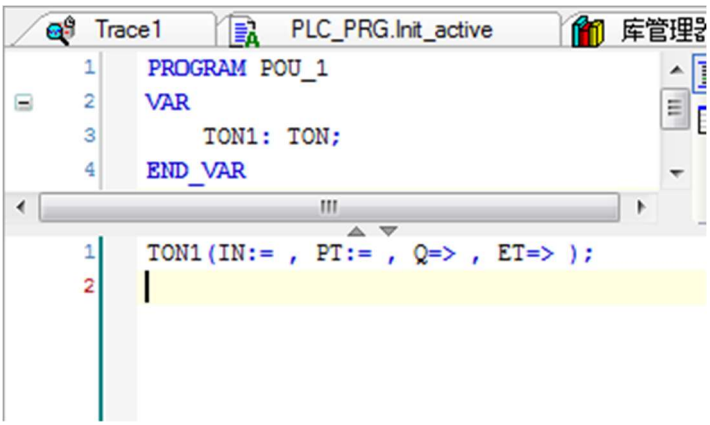
**2.** Function and function block calling

The function block call is implemented by instantiating the function block name, for example, Timer is the instance name of the TON function block, and the specific format is as follows

Function block instance name: (Function block parameter);

If you need to call the function block in ST, you can directly enter the instance name of the function block, and then assign values or variables to the parameters of the function block in parentheses. The parameters are separated by commas; Function block calls end with a semicolon.

For example, call the function block TON timer in structured text, assuming its instance name is TON1, and the specific implementation is as shown in the figure:




3. Selection statement

(1) IF

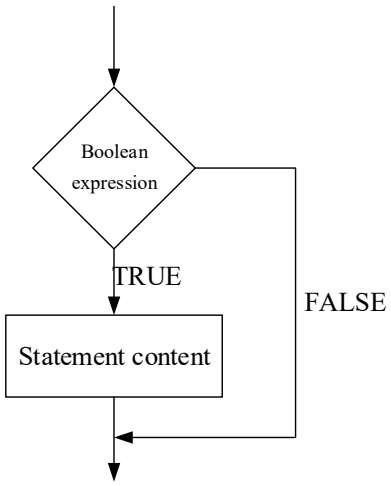
Implement a single branch selection structure using IF statements, with the basic format as follows:

IF < Boolean expression > THEN

< Statement content >;

END_IF

If the above format is used, the statement content is only executed when the<Boolean expression>is true, otherwise the<statement content>of the IF statement is not executed. The statement content can be a single statement, an empty statement, or multiple statements in parallel. The execution flowchart of this statement expression is shown in the figure:




(2) IF...ELSE

Implement a dual branch selection mechanism using IF statements, with the basic format as follows:

IF < Boolean expression > THEN

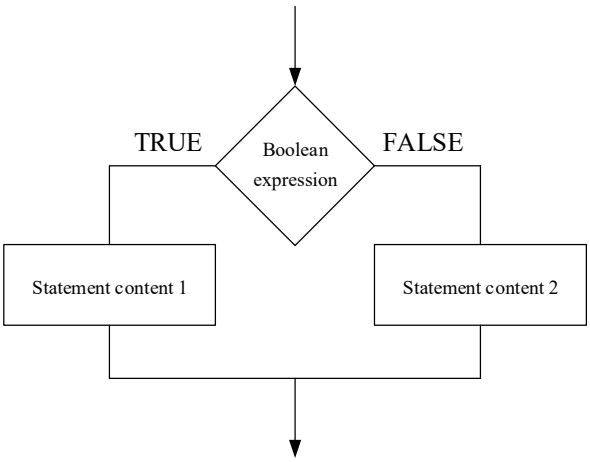
< Statement content 1>;

ELSE

< Statement content 2>;

END_IF

The above expression first determines the value within the<Boolean expression>. If it is true,<statement content 1>is executed. If it is false, <statement content 2> is executed. The program execution flowchart is shown in the figure:




When there is more than one conditional determinant in the program, another nested IF... ELSE statement, namely the multi branch selection structure, is required. The basic format is as follows.

IF < Boolean expression 1> THEN
  IF < Boolean expression 2> THEN
    < Statement content 1>;
ELSE
    < Statement content 2>;
  END_IF
ELSE
    < Statement content 3>;
END_IF

As mentioned above, an IF... ELSE statement has been placed in IF... ELSE to achieve nesting. Below, an example is provided to illustrate the use of nesting. The above expression first checks the value within <Boolean expression 1>. If it is true, continue to check the value of <Boolean expression 2>. If the value of <Boolean expression 1> is false, execute <statement content 3>, and return to <Boolean expression 2> to check. If <Boolean expression 2> is true, execute <statement content 1>. Otherwise, execute <statement content 2>.

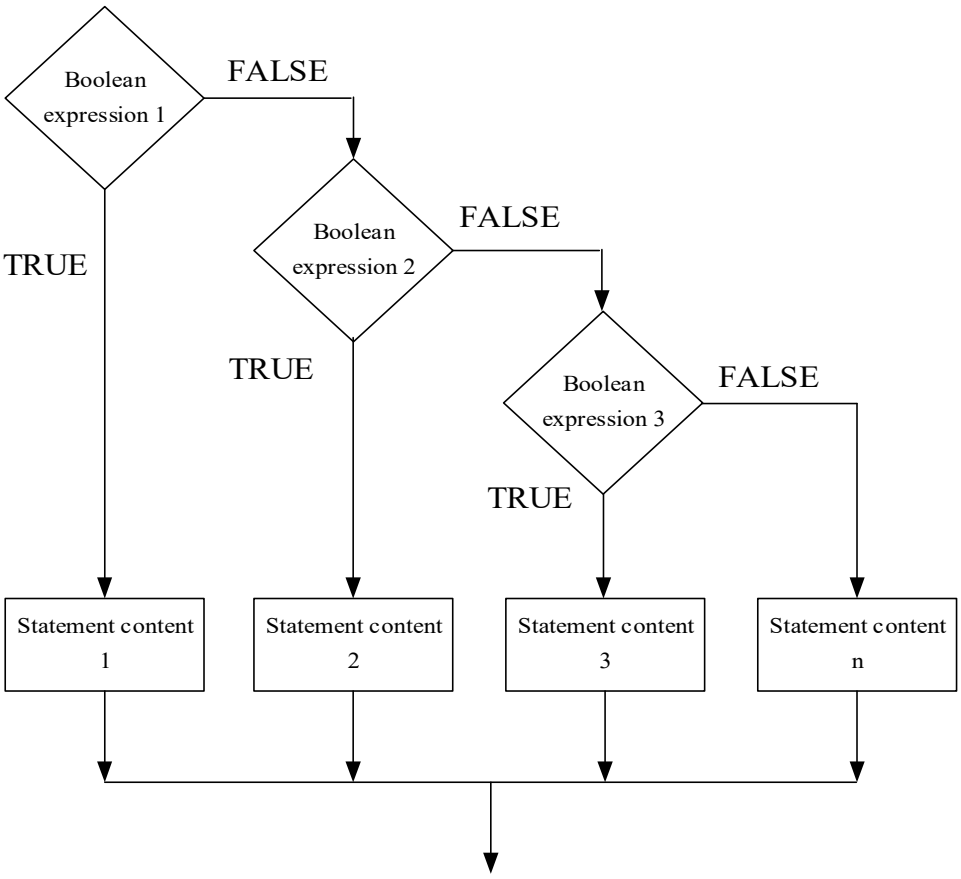
(3) IF..ELSIF..ELSE

In addition, the multi branch selection structure can also be presented in the following ways. The specific format is as follows

IF < Boolean expression 1> THEN
< Statement content 1>;
ELSIF < Boolean expression 2> THEN
< Statement content 2>;
ELSIF < Boolean expression 3> THEN
< Statement content 3>;
. . .
. . .
ELSE
< Statement content n>;
END_IF

If the expression<Boolean expression 1> is true, only the instruction <statement content 1> is executed, and no other instructions are executed. Otherwise, the judgment starts from the expression <Boolean expression 2> until one of the Boolean expressions is true, and then the statement content corresponding to this Boolean expression is executed. If the values of the Boolean expression are not true, only the instruction <statement content n> is

executed, and the program execution flowchart is shown in the figure.




(4) CASE statement

The CASE statement is a multi branch selection statement that selects a branch from multiple branches for execution based on the value of an expression. The basic format is as follows:

CASE < Conditional variables > OF
< Value 1>: < Statement content 1>;
< Value 2>: < Statement content 2>;
< Value 3, Value 4, Value 5>: < Statement content 3>;
< Value 6 .. Value 10>: < Statement content 4>;
...
< Value n>: < Statement content n>;
ELSE
<ELSE Statement content >;
END_CASE;

The CASE statement is executed in the following pattern:

- If the value of <conditional variable> is <value i>, execute the instruction <statement content i>.
- If the <conditional variable> does not have any specified value, execute the instruction <ELSE statement content>.
- If several values of a conditional variable require the same instruction to be executed, the values can be written together one after another and separated by commas. In this way, the common instructions are executed, as shown in the fourth line of the program.
- If the conditional variable needs to execute the same instruction within a certain range, it can be separated by writing the initial and final values as two points. In this way, the common instructions are executed, as shown in the fifth line of the program.

4. Iteration statement

Iterative statements are mainly used for repeatedly executing programs. In XS Studio, common iterative statements include FOR, REPEAT, and WHILE statements. The following is a detailed explanation of these statements:

(1) FOR

The FOR loop statement is used to compute an initialization sequence. When a condition is true, the nested statement is executed repeatedly and an iteration representation expression sequence is computed. If it is false, the loop is terminated. The specific format is as follows.

FOR<Variable>:=<Initial value>TO<Target value>{BY<Step size>} DO

< Statement content >

END_FOR;

The execution order of the FOR loop is as follows:

- Calculate whether the <variable> is within the range of <initial value> and <target value>.
- When the <variable> is less than the <target value>, execute the <statement content>.
- When the <variable> is greater than the <target value>, the <statement content> will not be executed.
- Every time the <statement content> is executed, the <variable> always increases its value by the specified step size. The step size can be any integer value.

If the step size is not specified, its default value is 1. When the <variable> is greater than the <target value>, exit the loop.

In a sense, the principle of FOR loop is similar to that of a copier. The copier first sets the number of copies to be copied, which is the condition of the loop. When the condition is met, that is, the number of copies is equal to the set number of copies, and copying stops.

FOR loop is the most commonly used type of loop statement. FOR loop embodies a function of specifying the number of times and repeating it step by step. However, due to different code writing methods, other loop functions can also be implemented. Below, an example is used to demonstrate how to use FOR loop.

Example: Using a FOR loop to calculate the quintic of 2.

```
VAR
Counter: BYTE; (*cycle counter *)
Var1:WORD; (*output result*)
END_VAR

FOR Counter:=1 TO 5 BY 1 DO
Var1:=Var1*2;
END_FOR;
```

Assuming the initial value of Var1 is 1, then after the loop ends, the value of Var1 is 32.

**Note:**

If the <target value> is equal to the limit value of the <variable>, it will enter a dead cycle. Assuming that the count variable Counter in the above example is of type SINT (-128 to 127), setting <target value> to 127 will cause the controller to enter a dead loop. Therefore, limit values cannot be set for <target value>.

(2) WHILE

The method of using WHILE loop is similar to that of FOR loop. The difference between the two is that the ending condition of the WHILE loop can be any logical expression. You can specify a condition, and when it is met, the loop is executed. The specific format is as follows.

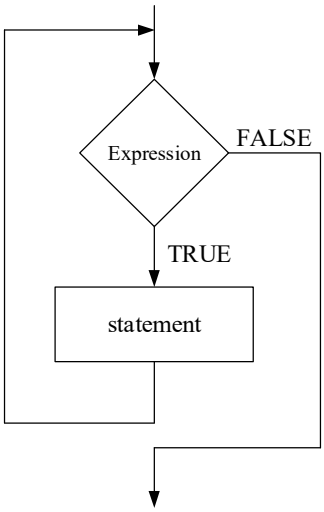
WHILE < Boolean expression >

< Statement content > ;

END_WHILE;

The execution order of the WHILE loop is as follows:

- Calculate the return value of <Boolean expression>.
- When the value of <Boolean expression> is true, execute the <statement content> repeatedly.
- When the initial value of <Boolean expression> is FALSE, the instruction <statement content> will not be executed and will jump to the end of the WHILE statement. The flowchart is shown in the following figure:




**Note:**
If the value of <Boolean expression> is always true, it will result in a death loop, which should be avoided. It is possible to avoid the occurrence of dead loops by changing the conditions of loop instructions. For example, using a counter that can be incremented or decremented to avoid the occurrence of dead loops.
The WHILE statement is used in engineering to control a motor. When the "start" button is pressed (Boolean expression is True), the motor rotates continuously. When the "stop" button is pressed (Boolean expression is FALSE), the motor also stops immediately. Here is an example to demonstrate how to use the WHILE loop.
Example: As long as the counter is not zero, the program inside the loop is always executed.

```
VAR
Counter: BYTE; (*Counter*)
Var1:WORD;
END_VAR

WHILE Counter<>0 DO
Var1 := Var1*2;
Counter := Counter-1;
END_WHILE
```

In a certain sense, the WHILE loop is more powerful than the FOR loop because the WHILE loop does not need to know the number of loops before executing the loop. Therefore, in some cases, only these two types of loops are sufficient. However, if the number of loops is clearly known, then FOR loops are better because FOR loops can avoid death loops.

(3) REPEAT

The REPEAT loop is different from the WHILE loop because it only checks the end condition after the instruction is executed. This means that regardless of the ending condition, the loop should be executed at least once.

```
REPEAT
< Statement content >
UNTIL
< Boolean expression >
END_REPEAT;
```

The execution order of the REPEAT loop is as follows:

- When the value of <Boolean expression> is FALSE, execute <statement content>.
- When the value of <Boolean expression> is true, stop executing <statement content>.
- After the first execution of <statement content>, if the value of <Boolean expression> is true, then

<statement content> is only executed once.

**Note:**

If the value of <Boolean expression> is always true, it will result in a death loop, which should be avoided. It is possible to avoid the occurrence of dead loops by changing the conditions of the loop instruction section. For example, using a counter that can be incremented or decremented to avoid the occurrence of dead loops.
Example: REPEAT loop. When the counter is 0, the loop stops.

```
VAR
Counter: BYTE;
END_VAR

REPEAT
Counter := Counter+1;
UNTIL
Counter=0
END_REPEAT;
```

The result of this example is that each program cycle enters the REPEAT cycle with a Counter of BYTE (0-255), which means 256 self addition calculations were performed within each cycle.
As mentioned earlier, "this means that regardless of the ending condition, the loop is executed at least once." Therefore, whenever the REPEAT statement is entered, the Counter is set to 1, and the Counter:=Counter+1 instruction is executed 256 times in each cycle until the Counter variable is accumulated to overflow to 0 and the loop is exited. And then added to the overflow, so it goes back and forth.

5. Jump statement

(1) EXIT

If the EXIT instruction is used in the FOR, WHILE, and REPEAT loops, the inner loop stops immediately regardless of the ending condition. The specific format is as follows:

```
EXIT;
```

Example: Use the EXIT command to avoid division by zero when using iterative statements.

```
FOR Counter:=1 TO 5 BY 1 DO
INT1:= INT1/2;
IF INT1=0 THEN
EXIT; (* Avoiding program division by zero *)
END_IF
Var1:=Var1/INT1;
END_FOR
```

When INT1 equals 0, the FOR loop ends.

(2) CONTINUE

This instruction is an extension of the IEC 61131-3 standard and can be used in three loops: FOR, WHILE, and REPEAT.
The CONTINUE statement interrupts the current loop, ignoring the code following it and starting a new loop directly. When multiple loops are nested, the CONTINUE statement can only start a new loop for the loop statement that directly contains it. The specific format is as follows:

```
CONTINUE;
```

Example: Use the CONTINUE instruction to avoid division by zero when using iterative statements.

```
VAR
Counter: BYTE; (*cycle counter *)
INT1,Var1: INT; (*intermediate variable *)
Erg: INT;    (*output result*)
END_VAR

FOR Counter:=1 TO 5 BY 1 DO
INT1:= INT1/2;
IF INT1=0 THEN
CONTINUE;    (* Avoid division by zero *)
END_IF
Var1:=Var1/INT1; (*Only execute when INT1 is not equal to 0 *)
END_FOR;
Erg:=Var1;
```

(3) JMP

Jump statements, jump instructions can be used to unconditionally jump to a line of code marked with a jump. The specific format is as follows:

```
< Identifier >:
.
JMP < Identifier >;
```

< Identifier > can be any identifier, which is placed at the beginning of the program line. The JMP instruction is followed by a jump destination, which is a predefined identifier. When the JMP instruction is executed, it will jump to the program line corresponding to the identifier.

Note: It is necessary to avoid creating a dead loop and can be used in conjunction with IF conditional control jump instructions.

Example: Using JMP statements to loop a counter within the range of 0 to 10.

```
VAR
nCounter: BYTE;
END_VAR

Label1:nCounter:=0;
Label2:nCounter:=nCounter+1;
IF nCounter<10 THEN
JMP Label2;
ELSE
JMP Label1;
END_IF
```

In the above example, Label1 and Label2 belong to labels and are not variables, so variable declarations are not necessary in the program.

Use the IF statement to determine if the counter is within the range of 0-10. If it is within the range, execute the JMP Label2 statement, and the program will jump to Label2 in the next cycle. Execute the program nCounter:=nCounter+1 to add 1 to the counter. Otherwise, it will jump to Label1, execute nCounter:=0, and reset the counter to zero.

The functionality in this example can also be achieved by using FOR, WHILE, or REPEAT loops. In general, the use of JMP jump instructions should be avoided as it reduces the readability and reliability of the code.

(4) RETURN

RETURN is return command, Used to exit the Program Organization Unit (POU), the specific format is as follows:

```
RETURN;
```

Example: Using an IF statement as a judgment, terminate the execution of this program immediately when the condition is met.

```
VAR
nCounter: BYTE;
bSwitch: BOOL; (*switch signal*)
END_VAR

IF bSwitch=TRUE THEN
RETURN;
END_IF;
nCounter:= nCounter +1;
```

When bSwitch is FALSE, nCounter always performs a self increment of 1. If bSwitch is True, nCounter maintains the previous cycle's value and immediately exits this POU.

6. Null statement

Not executing any content.

The specific format is as follows.

```
;
```

7. Comment

(1) Add the comment

Comments are a crucial part of a program, making it more readable while not affecting its execution. Comments can be added anywhere in the declaration or execution section of the ST editor.
In ST language, there are two comment methods:
Method 1: Multiple line comments start with (*, end with *). This comment method allows for multiple lines of comments, as shown in the following figure:

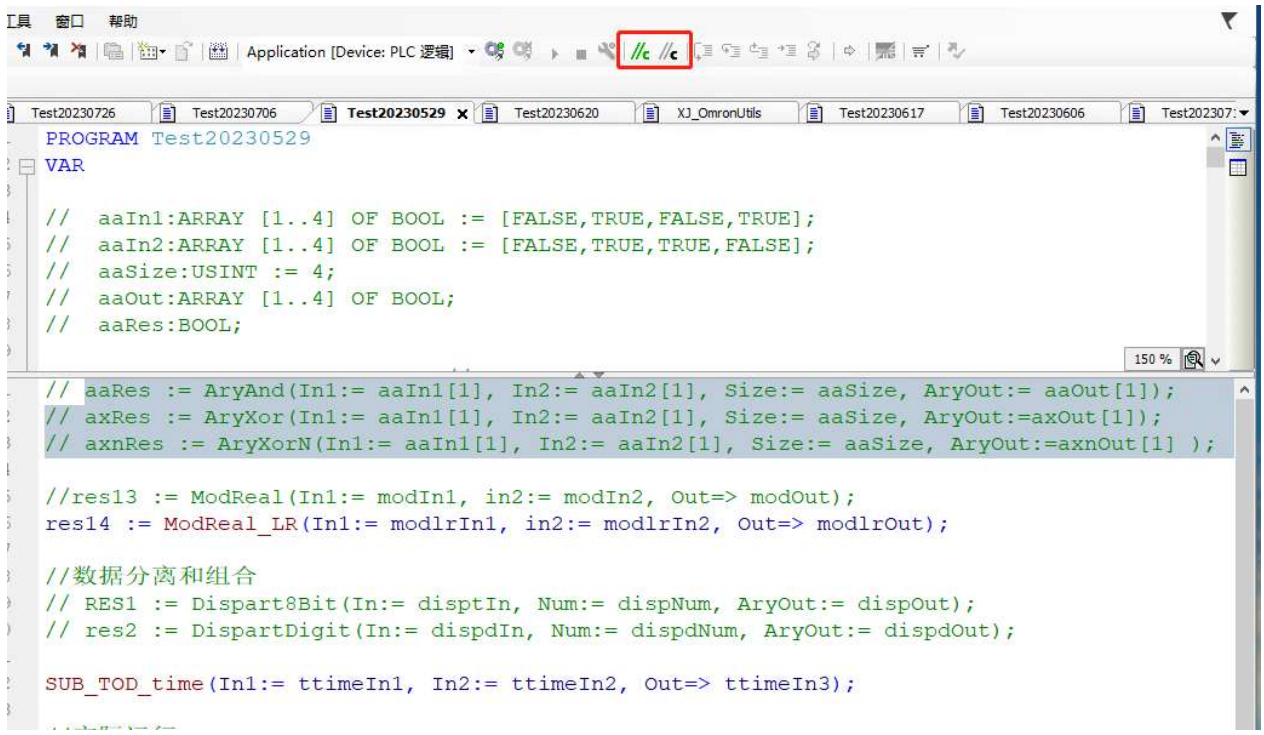
```
(*
    bOperationActive:=FALSE;
    bOrderActive:=FALSE;
    bRecipeActive:=FALSE;
    bInfoActive:=FALSE;
    bServiceActive:=FALSE;
    bSimulationActive:=FALSE;
*)
    IF iMainAreaIndex = 0 THEN
        bOperationActive:=TRUE;
    ELSIF iMainAreaIndex = 1 THEN
        bOrderActive:=TRUE;
```

Method 2: Single line comments start with "//" and continue until the end of the line. This is the method of single line comment, as shown in the following figure:

```
// gesture handling:
// only when mouseup was done
IF xRight AND bDragCanStart = FALSE THEN
    xRight := FALSE;
    IF iMainAreaIndex < MAX_MODULES-1 THEN
        iMainAreaIndex := iMainAreaIndex + 1;
        bIndexChanged := TRUE;
    END_IF
```

(2) Comments switching

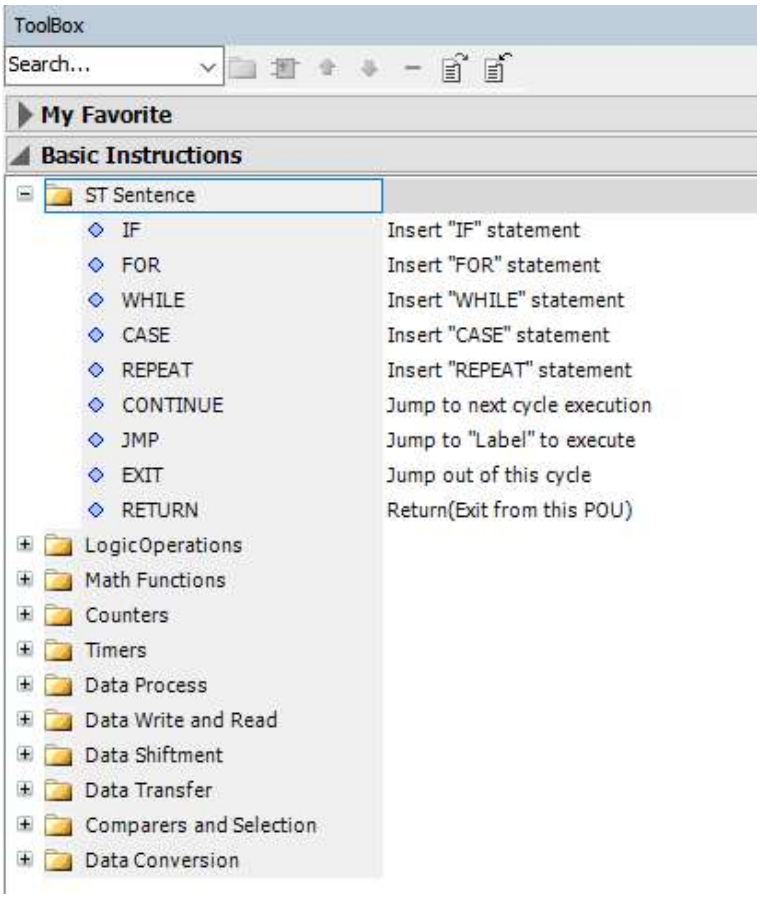
Use the shortcut keys Ctrl+U, Ctrl+M, or click from the menu bar to quickly comment or uncomment code selection.



## 6-2-4. ST editing

1. ST toolbox

The tool category interface is as follows:

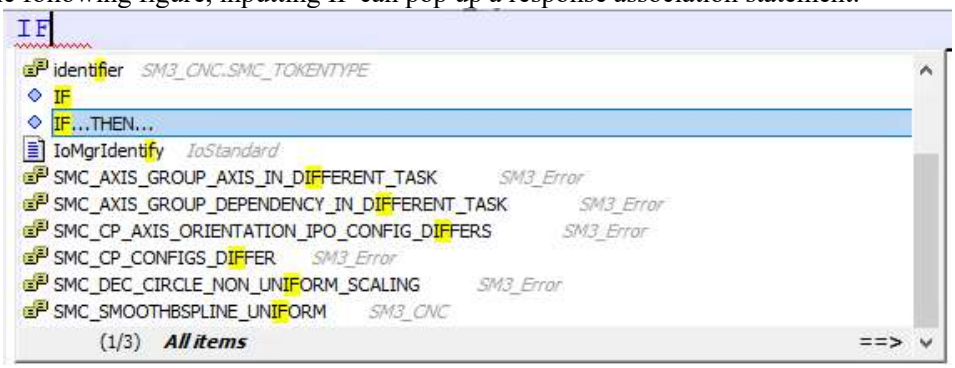



The toolbox contains ST statements, logical operations, mathematical functions, counters, timers, data processing, data writing and reading, data shifting, data transfer, comparison and selection, and data conversion. They can be dragged or double clicked into the programming area, such as ST statements, IF statements, WHILE statements, REPEAT, CASE statements, CONTINUE, JMP, EXIT, RETURN, and statement templates are automatically inserted upon insertion.

2. Smart alert

(1) keyword matching

Enter the ST statement type keyword, which can automatically match. The statement includes IF statement, WHILE, FOR, CASE, REPEAT, and the formatting template can be found in the attached statement template.
As shown in the following figure, inputting IF can pop up a response association statement:




(2) TAB key shortcut function

- Capable of automatically formatting input and output for functional blocks, functions, methods, actions, and programs;
- Capable of automatically formatting input and output for function block instances and their methods and

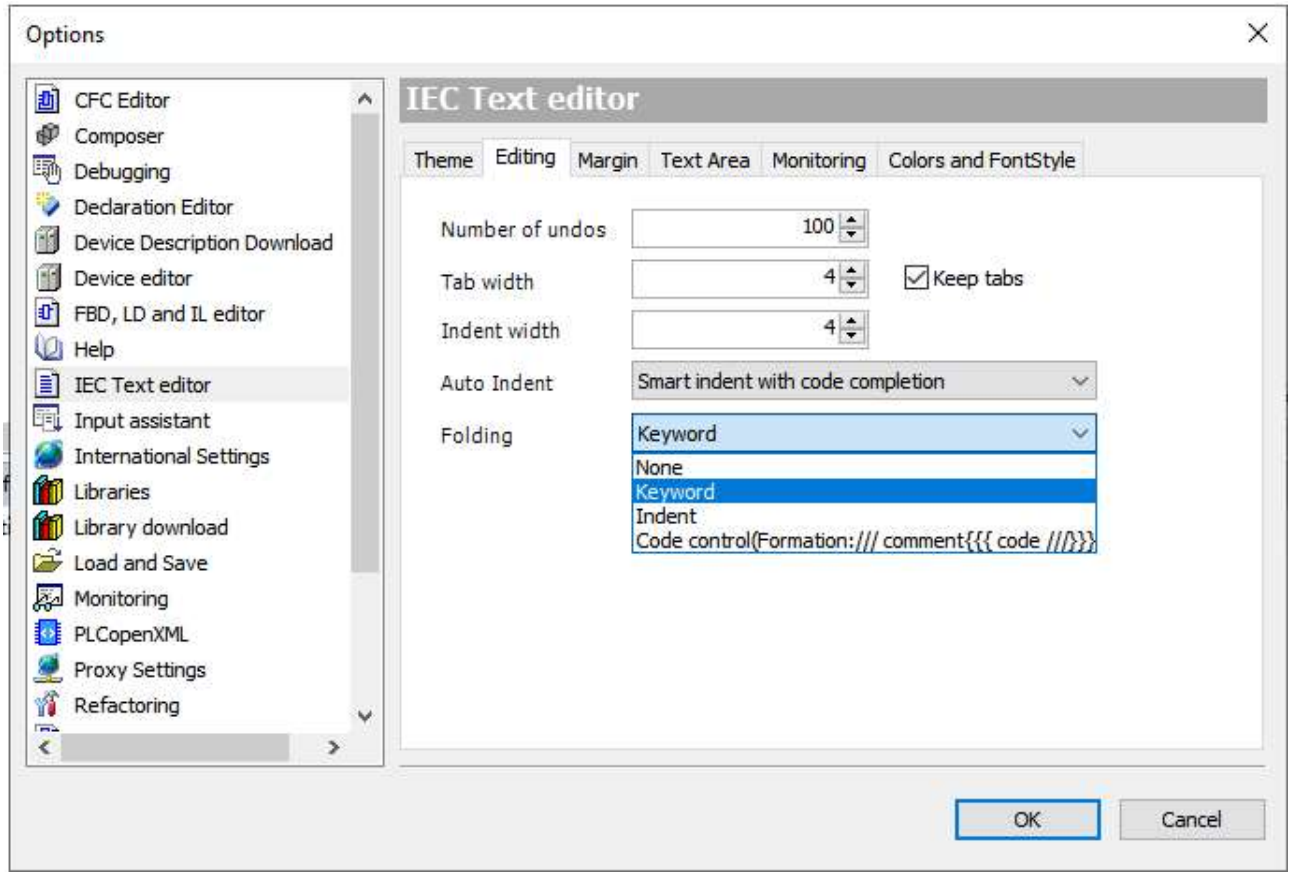
actions;

- Capable of automatically formatting IF, WHILE, FOR, CASE, and REPEAT statements. The formatting template can be found in the attached statement template, such as the function type name, function block instance, etc. After input, press the Tab key to automatically format.

3. Fold Zoom

- Folding method supports keywords: keywords includeVAR, VAR_INPUT, VAR_GLOBAL, VAR_OUTPUT, VAR_IN_OUT, VAR_TEMP, VAR_STAT, VAR_ESTERNAL, CASE, FOR, REPEATED, IF/ELSE/ELSIF, WHILE, STRUCT, UNION, TYPE, __TRY, __CATCH, __FINALLY.
- If intelligent indentation is selected in the automatic indentation function, the tab length will be automatically added based on the above keywords. If intelligent indentation is selected and automatically completed, the end of the keyword will be automatically completed, such as VAR, FOR, WHILE, and nesting is supported.
- When using intelligent indentation, if the line is a keyword, the tab character will be automatically added after the line breaks. If it is not a keyword, it will be indented the same as the line. Block highlighting. Display block highlighting information between brackets, WHILE, FOR, IF, ElSE, CASE, REPEAT, RUCT, UNION, TYPE, TRY, etc. There are highlighting markers at both text boundaries and text regions.

Here, the "IEC Text Editor" settings interface can be opened by clicking on the "Tools" ->"Options" menu in the menu bar. Users can set the folding method according to their actual needs. As shown in the following figure:



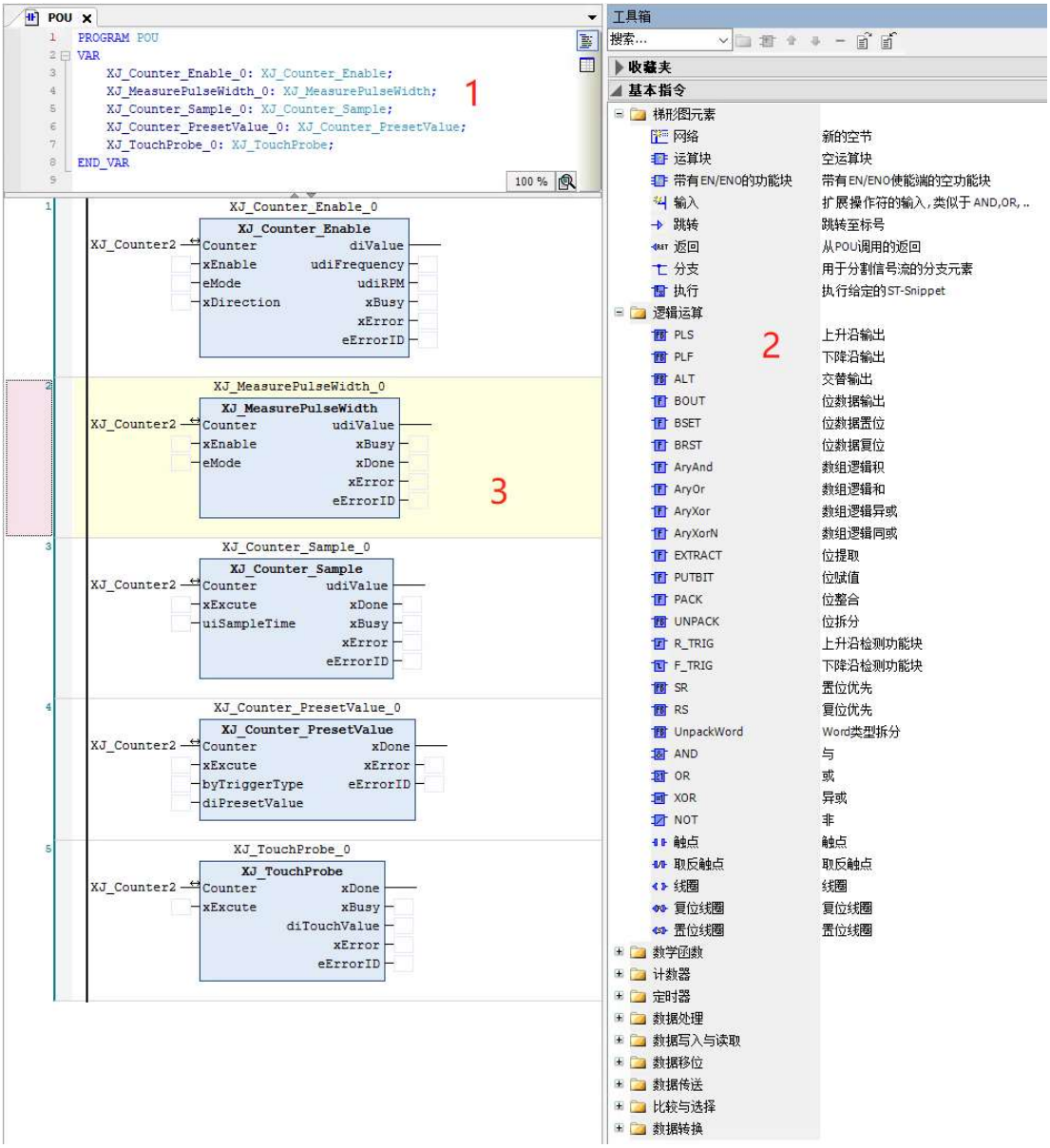
# 6-3. Ladder diagram

## 6-3-1. Overview

The ladder diagram originated from the United States and is based on graphical representation of relay logic. It is the most widely used graphical language in PLC programming. There are two vertical power trajectories on the left and right sides of the ladder program. The power trajectory on the left nominally provides energy for the power flow from left to right along the horizontal steps through various contacts, functions, blocks, coils, etc. The endpoint of the power flow is the power trajectory on the right. Each contact represents the state of a Boolean variable, and each coil represents the state of an actual device. The function or functional block corresponds to the standard library or user created function or functional block in IEC 1131-3.

Ladder diagram is the most widely used programming language in China, and it is also one of the three graphical programming languages in IEC 1131-3. Ladder diagram is the most commonly used graphical programming language in traditional PLCs and is also known as the first programming language of PLCs. Based on the status and logical relationship of each contact point in the ladder diagram, calculate the status of the programming components corresponding to each coil in the diagram, which is called the logical solution of the ladder diagram.

Some programming components in the ladder diagram use the name relay, such as coils, contacts, etc., but they are not real physical relays, but rather some storage units (soft relays), each corresponding to a storage unit in the image register of the PLC memory. If the storage unit is in a "True" state, it indicates that the coil of the corresponding soft relay in the ladder diagram is "energized", with its normally open contact connected and normally closed contact disconnected. This state is called the "True" or "ON" state of the soft relay. If the storage unit is in the "FALSE" state, the coil and contact states of the corresponding soft relay are opposite to the above, and the soft relay is called in the "FALSE" or "OFF" state. These "soft relays" are often referred to as programming components during use. The ladder diagram editing interface is shown in the following figure:




Explanation: In the above figure, 1 is the variable definition area, 2 is the toolbox, and 3 is the ladder diagram programming area.

## 6-3-2. LD program execution sequence

The execution process of the ladder diagram is carried out in order from left to right and from top to bottom, as shown in the figure:

1. Execution process

(1) Generatrix

The ladder diagram adopts a network structure, and the network of a ladder diagram is bounded by the left busbar. When analyzing the logical relationship of ladder diagrams, in order to borrow the analysis method of relay circuit diagrams, it can be imagined that there is a left positive and right negative DC power supply voltage between the left and right busbars (left and right busbars), and there is "energy flow" between the busbars flowing from left to right. The right busbar is not displayed.

(2) Section

A section is the smallest unit in a ladder network structure, and the logical network from the input condition to a coil is called a section. In the editor, sections are arranged vertically. In XS Studio, each section is indicated by a series of section numbers on the left, containing input and output instructions, composed of logical expressions, arithmetic expressions, programs, function or function block call instructions, jump or return instructions.

To insert a section, you can use the command to insert the section or drag it from the toolbox. The elements contained in a section can be copied or moved by dragging and dropping them in the editor.

When executing a ladder diagram, it starts from the section with the smallest label, determines the state of each element from left to right, and determines the state of the connecting elements on the right side. It is executed one by one to the right, and the result of the operation is output by the execution control element. Then proceed to the execution process of the next section. The above figure shows the execution process of the ladder diagram.

(3) Energy flow

As shown in the above figure, the bold blue line represents the energy flow, which can be understood as an imaginary "conceptual current" or "energy flow"

(PowerFlow) flows from left to right, which is consistent with the order of logical operations when executing user programs. Flow can only flow from left to right. The concept of energy flow can help us better understand and analyze ladder diagrams.

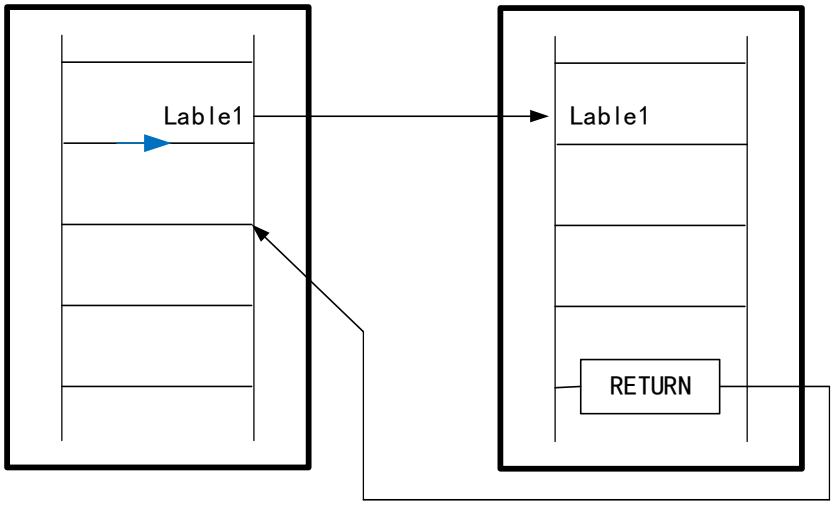
(4) Branch

When there are branches in the ladder diagram, the state of each graphic element is analyzed based on the execution order from top to bottom and from left to right. The state of the right connecting element is determined according to the relevant regulations for vertical connecting elements, and the calculation process is executed one by one from left to right and from top to bottom. In the ladder diagram, the evaluation without feedback paths is not very clear. All external input values related to these contacts must be evaluated before each step.

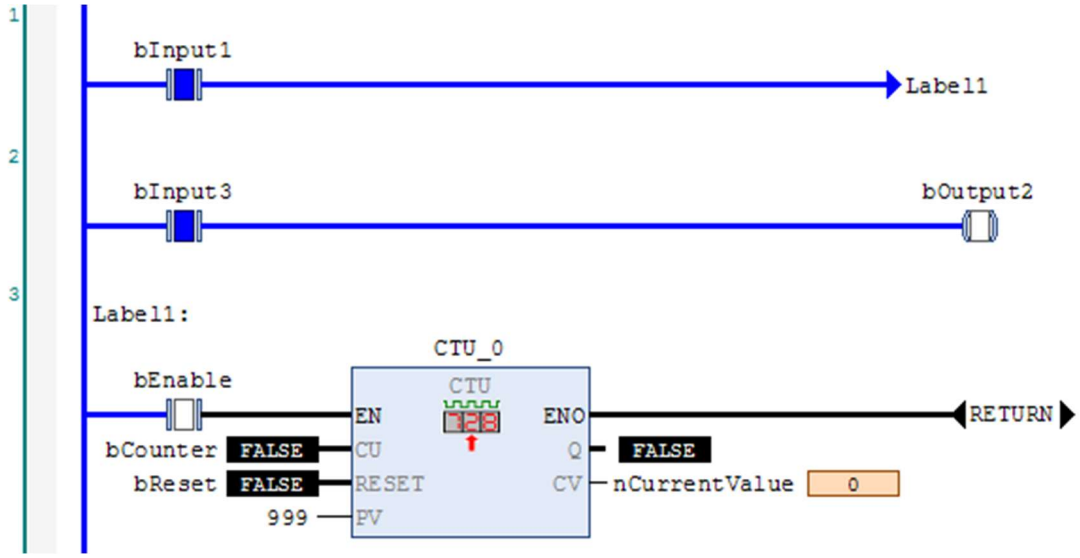
2. Executing control

(1) Jump and return

When the jump condition is met, the program jumps to the section labeled in the Label and starts executing until that part of the program reaches RETURN, returning to the original section and continuing execution. Its structural diagram is shown in the following figure:



The jump and return instructions for using ladder diagrams in XS Studio are as follows.
Example: using jump instructions to execute a program:

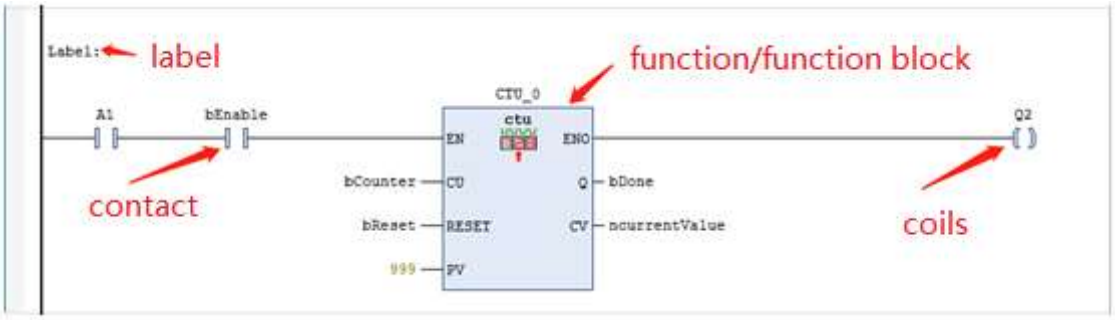



As shown in the figure, bInput1 is set to True, so a jump statement is executed. Based on label Label1, the program jumps to Label in section 3. Therefore, although bInput3 in section 2 is set to ON, bOutput2 is never set to True because the program directly skips the statement. Only when B1 is False and bInput3 is True, bOutput2 will be True.

## 6-3-3. Constituent elements

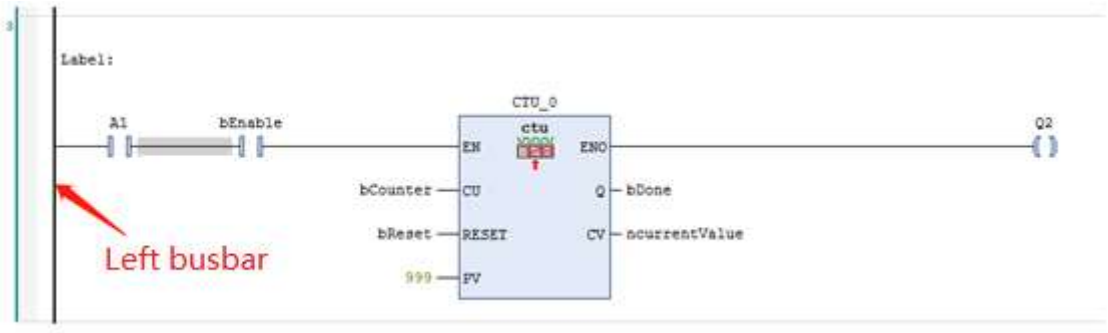
The ladder diagram language in IEC 1131-3 is a reasonable absorption and reference for the ladder diagram language of various PLC manufacturers. The graphic symbols in the language are basically consistent with those of each PLC manufacturer. The following diagram shows the ladder diagram editor view. The main graphical symbols of IEC 61131-3 include:

- Basic connection categories: power rail, connection elements.
- Contact type: normally open contact, normally closed contact, positive conversion readout contact, negative conversion contact.
- Coils: general coils, reverse coils, set (latch) coils, reset/unlock coils, hold coils, set hold coils, reset hold coils, positive conversion readout coils, negative conversion readout coils.
- Function and functional blocks: including standard functions and functional blocks, as well as user-defined functional blocks.




1. Power supply rail

The graphic elements of the power rail in a ladder diagram are also known as busbars. Its graphical representation is located on the left side of the ladder diagram, also known as the left power bus.




2. Connection elements

In a ladder diagram, each graphic symbol is connected by connecting elements, which are represented by horizontal and vertical lines. They are the most basic elements that make up the ladder diagram. The following figure is a graphical representation of the horizontal and vertical connecting elements:

(a) Horizontal connecting elements

(b) Vertical connection elements

3. Label

A label is an optional identifier and its address can be determined when defining a jump. It can contain any character.

4. Contact

Contact points are graphical elements of a ladder diagram. The contact of the ladder diagram follows the contact term of the electrical logic diagram, used to represent the state changes of Boolean variables. A contact is a ladder element that transmits a state to the horizontal connecting element to its right.

Contact points can be divided into Normally Open Contact (NO) and Normally Closed Contact (NC). Normally open contact refers to the state of FALSE when the contact is open under normal operating conditions. Normally closed contact refers to the state of true when the contact is closed under normal operating conditions. Table 2-6-2-1 lists the commonly used contact graphic symbols and explanations in ladder diagrams.

| Type | Symbol | Explanation |
|---|---|---|
| Normally open contact |  | If the contact corresponds to a Boolean variable value of True, then the contact pull-in. If the state of the connecting element on the left side of the contact is True, then the state True is passed to the right side of the contact, causing the state of the connecting element on the right side to be True. On the contrary, when the Boolean variable value is False, the state of the right connected element is False. |
| Normally closed contact |  | If the contact corresponds to a Boolean variable value of False, then the normally closed contact is in a pull-in state,<br>If the state of the connecting element on the left side of the contact is True, then the state of True is passed to the right side of the contact, making the state of the connecting element on the right side True. On the contrary, when the Boolean variable value is True, the contact opens, and the status of the right connected element is False. |
| Insert right contact |  | Multiple contacts can be connected in series and inserted on the right side. Multiple series connected contacts are in a closed state<br>Only the last contact can transmit True. |
| Insert normally |  | Multiple contacts can be connected in parallel, with normally open contacts inserted in parallel below the contacts. |

| Type | Symbol | Explanation |
|---|---|---|
| open contact under parallel connection | | If only one contact is True between two parallel contacts, then parallel lines transmit True. |
| Insert normally closed contact under parallel connection |  | Multiple contacts can be connected in parallel, with normally closed contacts inserted in parallel below the contacts. Normally closed contacts is in closed state by default, if the contact corresponds to a Boolean variable value of False and the state of the left connected element is True, then the parallel contact transmits True to the right. |
| Insert normally open contacts in parallel |  | Multiple contacts can be connected in parallel, with normally open contacts inserted in parallel on the upper side of the contacts. If only one contact is True between two parallel contacts, then parallel lines transmit True. |

5. Coil

A coil is a graphical element of a ladder diagram. The coil in the ladder diagram follows the coil term of the electrical logic diagram, used to represent the state changes of Boolean variables. According to the different characteristics of the coil, it can be divided into instantaneous coil and latch coil, with latch coil divided into set coil and reset coil. The following table lists the commonly used coil graphic symbols and explanations in ladder diagrams.

| Type | Symbol | Explanation |
|---|---|---|
| Coil |  | The state of the left connecting element is passed to the relevant Boolean variables and the right connecting element, if the state of the left side connected element is true, then the Boolean variable of the coil is true, otherwise the coil is false. |
| Set coil |  | There is an S in the coil. When the state of the left connected element is true, the Boolean variable of the coil is set and held until it is reset by the Reset coil. |
| Reset coil |  | There is an R in the coil. When the state of the left connected element is true, the Boolean variable of the coil is reset and held until it is set by the Set coil. |

6. Auxiliary

It can perform edge detection, inversion, and set/reset operations on coils and contacts.

| Type | Symbol | Explanation |
|---|---|---|
| Inversion |  | Invert the signal. |
| Edge detection |  | There are two modes, P and N, which can be switched by clicking the tool. P is triggered at the rising edge of the collected signal, N is triggered at the falling edge of the collected signal. |
| Set/reset |  | There are two modes, R and S, which can be switched by clicking on the tool. S is set, R is reset. |

For example, on the right side of the network in LD, there can be any number of coils, represented by parentheses "( )". They can only be connected in parallel. A coil transfers the connected value from left to right and copies it to a corresponding Boolean variable. At the entry line, a value of ON (equivalent to Boolean variable TRUE) or a value of OFF (equivalent to Boolean variable FALSE) can appear. It is also possible to reverse the contact and coil (in the example, contact SWITCH1 and coil %QX3.0 are reversed). If a coil is negated (identified by the slash "(/)" in the coil symbol), it will copy the negated value into the corresponding Boolean variable. If a junction is negated, it is only connected when the corresponding Boolean value is FALSE.

7. Function and function block calls

Along with the contacts and coils, you can also insert functional blocks and programs. In the network, they must have an input and an output with Boolean values, and can be used at the same position like a junction, that is, on the left side of the LD network.

| Type | Symbol | Explanation |
|---|---|---|
| Insert operation block |  | Insert a function or function block, and use the mouse to select the desired function and function block based on the pop-up dialog box. Suitable for those who are not familiar with functions and functional blocks. |
| Insert empty calculation block |  | Directly insert a rectangular block and enter the function or function block name at the "?" field, suitable for users who are familiar with functions and function blocks. |
| Insert calculation block with EN/ENO |  | Only when EN is true, the function or function block is executed and the state is allowed to be passed downstream. Suitable for those who are not familiar with functions and functional blocks. |
| Insert empty calculation block with EN/ENO |  | Insert a rectangular block with EN/ENO, enter the function or function block name directly at "?", and only execute the function or function block when EN is true, allowing the state to be passed downstream. Suitable for users who are familiar with functions and functional blocks. |

The ladder diagram programming language supports calling functions and function blocks. When calling functions and function blocks, the following precautions should be taken:

(1) In a ladder diagram, functions and function blocks are represented by a rectangular box. A function can have multiple input parameters but only one return parameter. Function blocks can have multiple input parameters and multiple output parameters.

(2) The input is listed on the left side of the rectangular box, and the output is listed on the right side of the rectangular box.

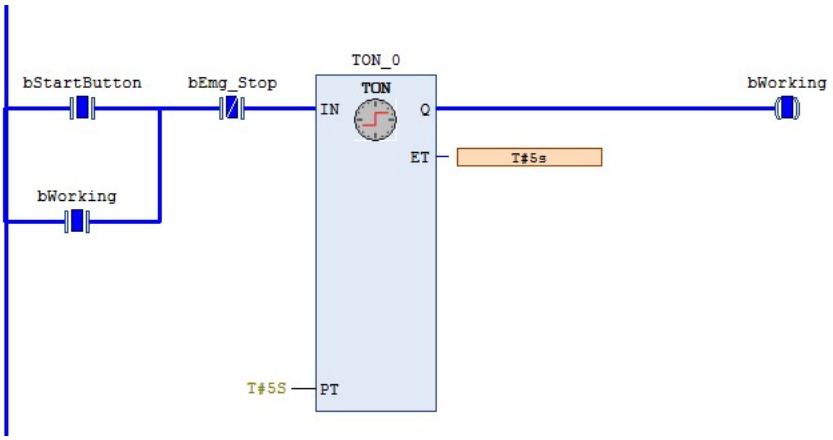
(3) The names of functions and function blocks are displayed in the upper and middle parts of the box. Function blocks need to be instantiated, and instance names are listed in the upper and middle parts outside the box. Use the instance name of the function block as its unique identifier in the project.

(4) To ensure that the energy flow can pass through functions or function blocks, each called function or function block should have at least one input and output parameter. In order for the connected functional blocks to execute, at least one Boolean input should be connected horizontally to the vertical left power rail.

(5) When calling a function block, the actual parameter value can be directly filled in at the external connection line of the function block for the internal parameter variable name.

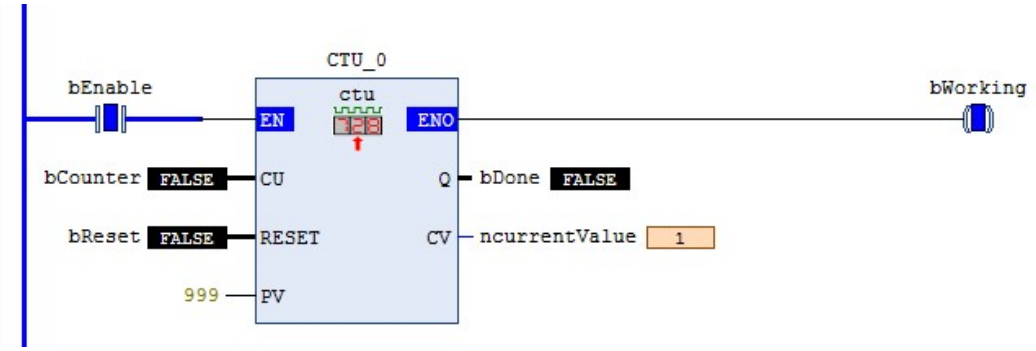
Example: Setting of function block call arguments.

Calling the TON delay ON function block, TON_ 1 is the instance name after instantiating the function block TON. The input parameter PT of the function block is set to t # 5s. Output parameters Q and ET, and variables can be left unconnected when there is no need to output parameters such as ET in the example.




It can be seen that the output Q of the function block TON is connected to the coil bWorking. When the contact bStartButton is True and bEmg_Stop is False and lasts for more than 5 seconds, bWorking is True. When bEmg_Stop is true when disconnected, bWorking is false.

(6) If there are no dedicated input and output parameters for EN and ENO, functions and function blocks will be automatically executed and the state will be passed downstream.




It can be seen that when the bCounter has a rising edge trigger signal, the parameter output variable CV is calculated by adding 1.

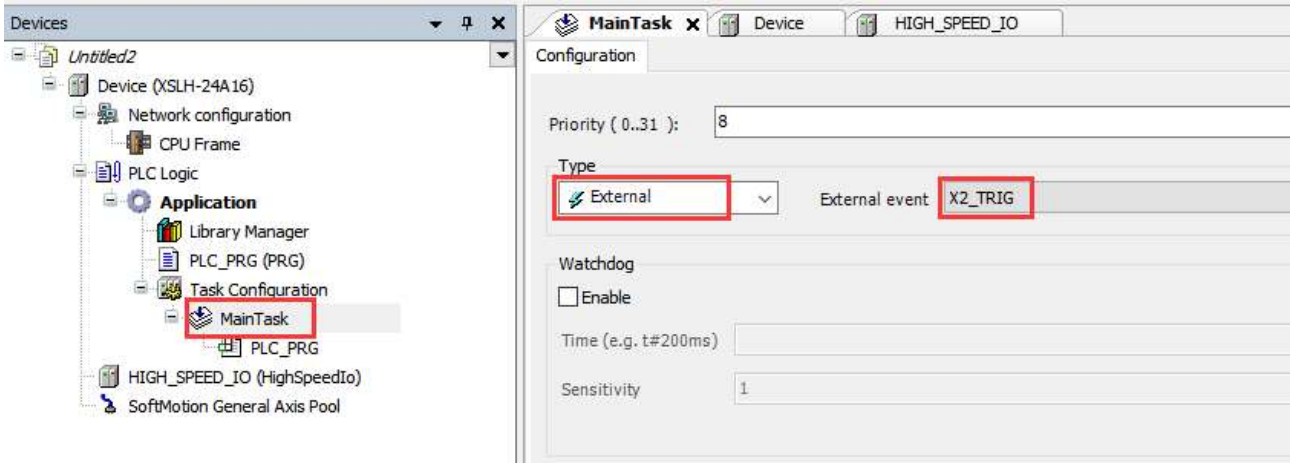
- When EN is False, the operations defined by the function block ontology are not executed, and the value of ENO is also correspondingly False.
- When the value of ENO is True, it indicates that the function block is being executed.

# 7. Special function

## 7-1. External interrupt

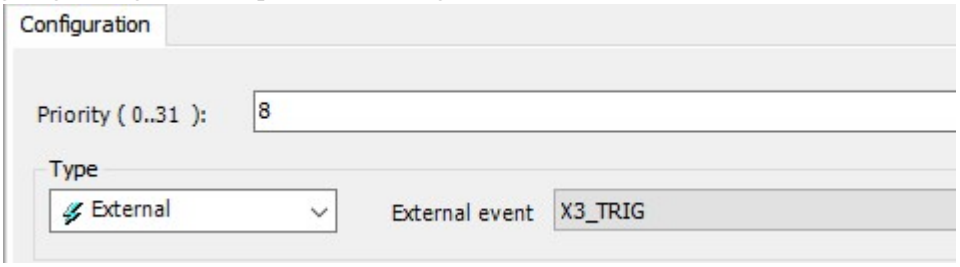
### 7-1-1. Application for firmware below 1.1.0

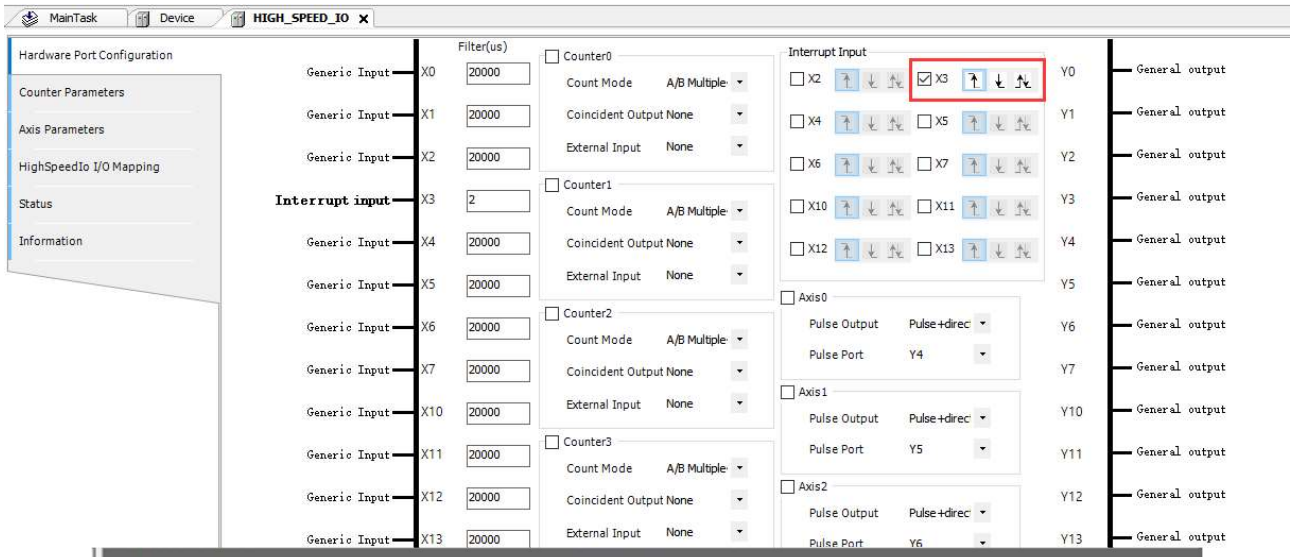
The XS series PLC supports X terminal interrupts, and the same terminal supports rising and falling edge interrupts. Interrupts are used in XS Studio through external event forms in the task type. Like X2R_TRIG represents X2 rising edge interrupt, X2F_TRIG represents the falling edge interrupt, and the number and type of interrupts supported by each model can be found in the external event "External" option.
Double click on "MainTask" and set it as an external event in the pop-up interface. External interrupts use terminal X, and the priority of external interrupt events can also be set.




### 7-1-2. Application for firmware 1.1.0

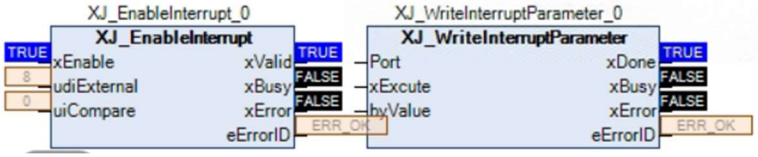
Need to use the 【XJ_Interrupt】 and 【XJ_WriteInterruptParameter】 commands and interfaces (see the instruction manual for details). Set X3 as an external interrupt input, take its dual edge signal, which can be configured on the hardware parameter configuration interface or using XJ_WriteInterruptParameter instruction. The self add 1 instruction in the POU program under another task (configured as external, X3_TRIG) is executed once an X3 edge signal is given. The parameter configuration and instructions are shown in the following figure:






Device.Application.PLC_PRG

| 表达式 | 类型 | 值 | 准备值 | 地址 | 注释 |
|---|---|---|---|---|---|
| XJ_EnableInterrupt_0 | XJ_EnableInterrupt | | | | |
| xEnable | BOOL | TRUE | | | 使能 |
| udiExternal | UINT | 8 | | | 打开外部输入中断，例如 |
| uiCompare | UINT | 0 | | | 打开比较一致中断，例如 |
| xValid | BOOL | TRUE | | | 中断生效 |
| xBusy | BOOL | FALSE | | | 正在运行 |
| xError | BOOL | FALSE | | | 错误标志 |
| eErrorID | HSIO_ERROR | ERR_OK | | | 错误代码 |
| XJ_WriteInterruptParameter_0 | XJ_WriteInterruptParameter | | | | |
| Port | UINT | 8 | | | 端口号，例如（外部输入中 |
| xExcute | BOOL | TRUE | | | 触发 |
| byValue | BYTE | 2 | | | 值(0: 上升沿，1 为下降沿， |
| xDone | BOOL | TRUE | | | 完成标志 |
| xBusy | BOOL | FALSE | | | 正在运行 |
| xError | BOOL | FALSE | | | 错误代码 |
| eErrorID | HSIO_ERROR | ERR_OK | | | 错误标志 |

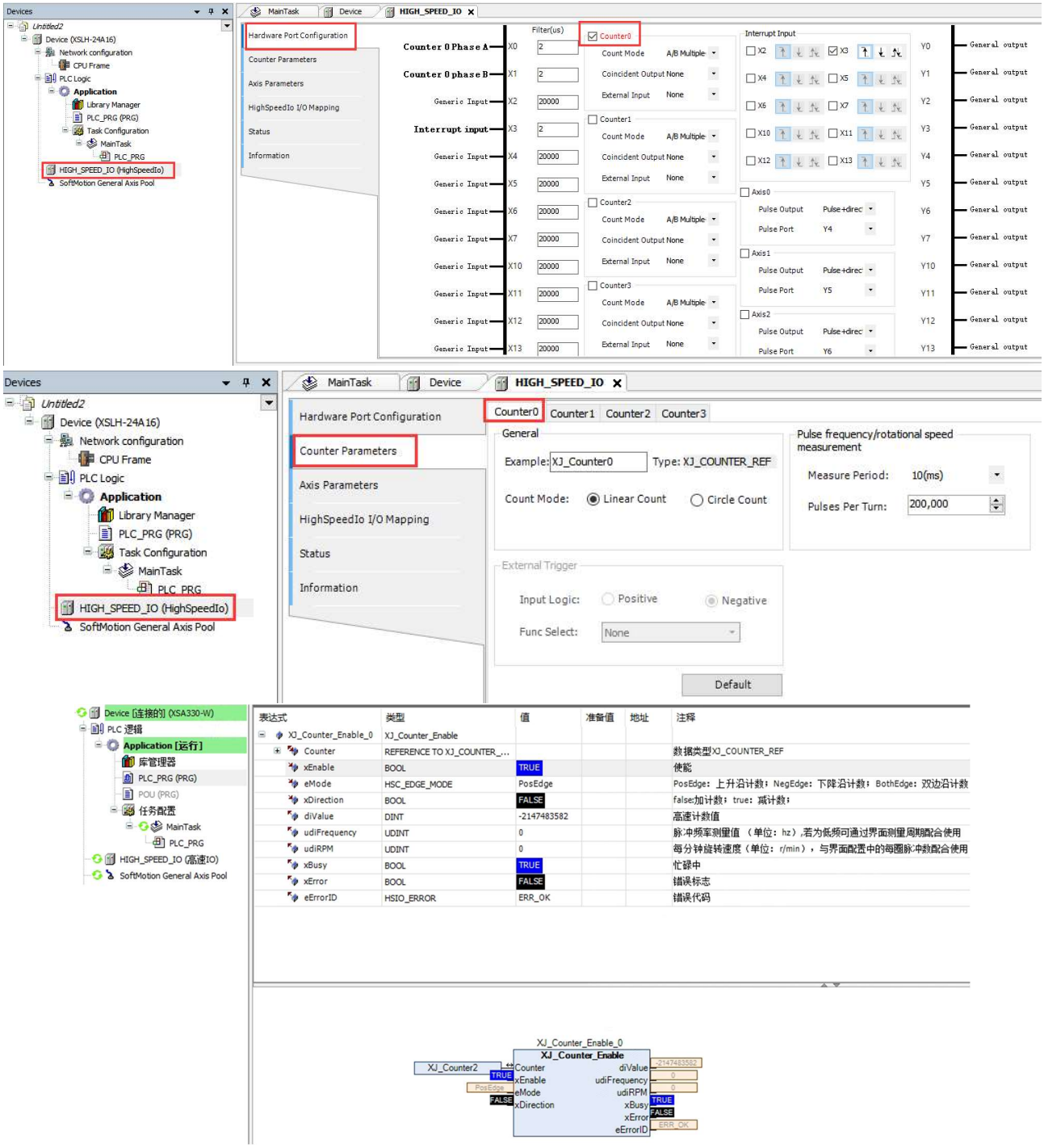


# 7-2. High speed counting

■ Application for firmware 1.1.0

Note: Firmware versions below 1.1.0 do not support high-speed IO interfaces. For high-speed counting instructions, please refer to the XS series PLCopen instructions manual.

Example 1: Using the [XJ_CounterEnable] command, measure the external high-speed signal input and configure it as shown in the following figure.



## 7-3. High speed IO configuration

Double click on the HIGH-SPEED-IO option in the device tree to open the hardware parameter configuration interface for high-speed IO. In this interface, the high-speed pulse output function and PWM output function can be configured (XS Studio 1.1.0 and above versions only support XSLH-24A16 and XSLH-24A8). The default parameter configuration interface is shown in the following figure:


1. High speed pulse output function

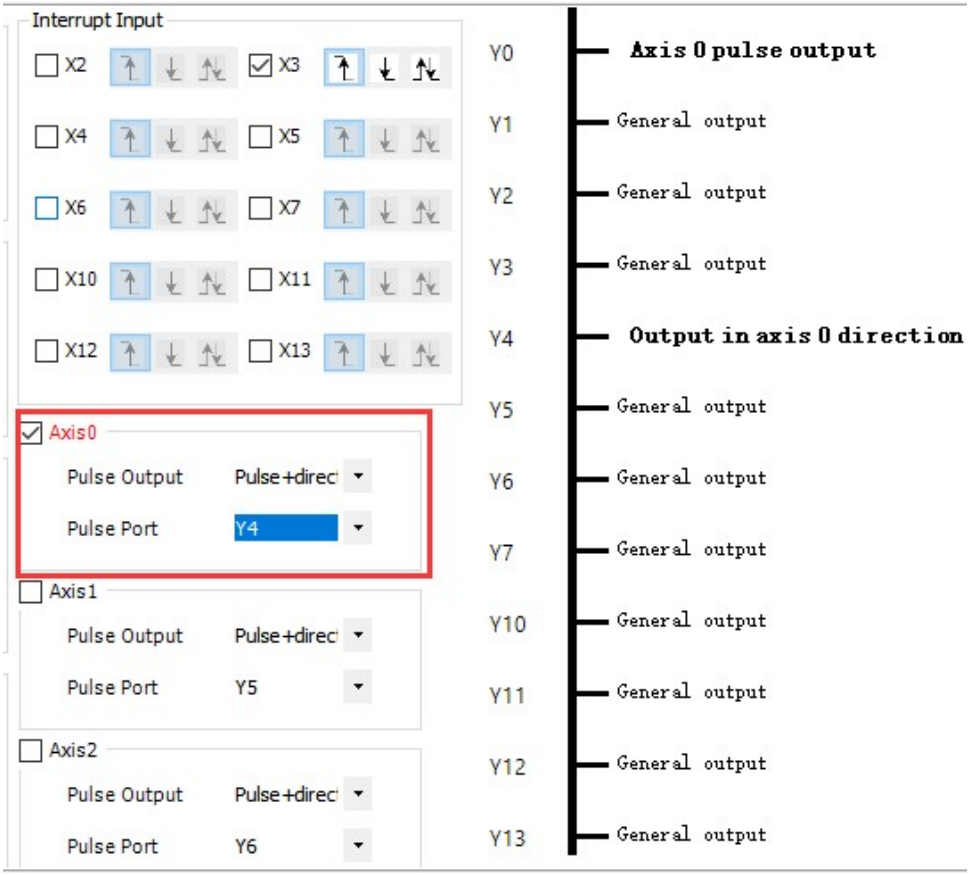
■ Pulse instructions

| Instruction | Function |
|---|---|
| MC_Power | Enable the axis |
| MC_Reset | Reset related errors inside the axis |
| MC_Jog | Jog |
| MC_Stop | Stop the motion |
| MC_MoveAbsolute | Move the axis to the absolute position |
| MC_MoveRelative | Move the axis to the relative position |
| MC_MoveVelovity | The axis keep moving at the specified speed |
| MC_SetPosition | Set the axis current position |
| MC_ReadStatus | Read the axis status |
| MC_ReadSetPosition | Read the current axis set position |
| MC_ReadActualPosition | Read the current axis current position |
| XMC_ZRN | Pulse homing |

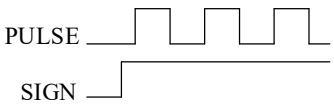
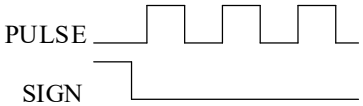
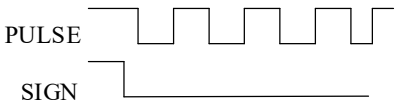
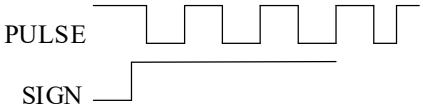
In the above instructions, XMC_ZRN is the instruction in the XJ_HSIO library (supported by PLC firmware above V2.2.0), while the rest are instructions in SM3_Basic library. For specific instructions, please refer to the XS series PLCopen standard controller user manual [Instruction Section].

■ Configure the high speed pulse output function

- Mainly including pulse output mode and pulse direction port (taking axis 0 configuration as an example, axis number supports 0-3).
- Check the axis 0; The configuration after checking is shown in the figure:



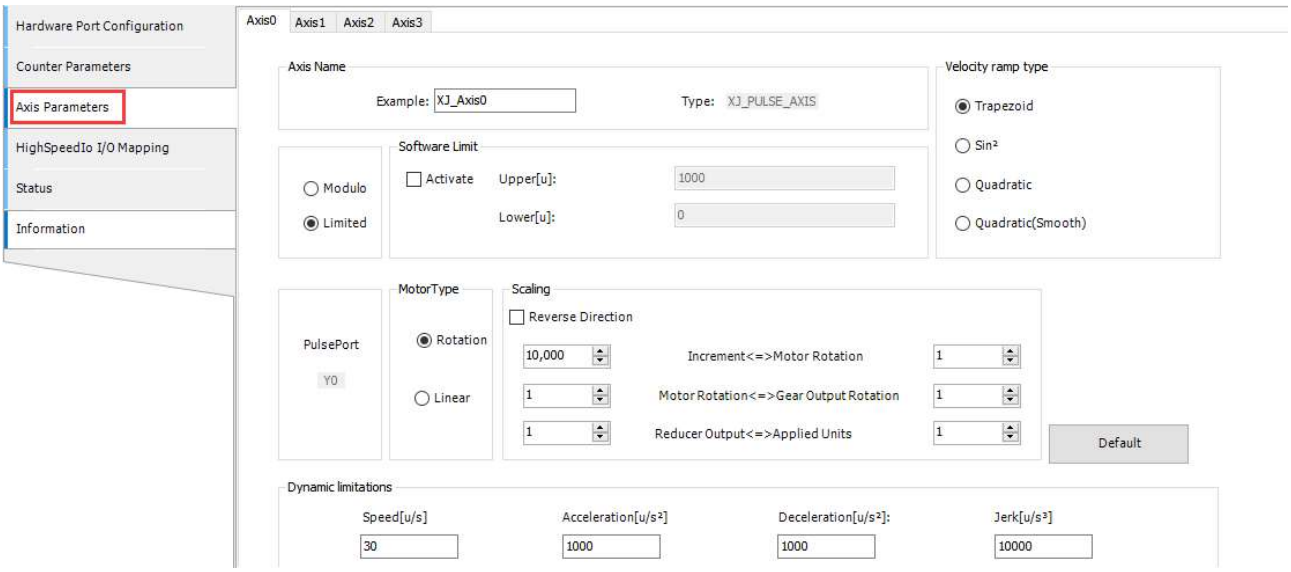

- Configure the high speed pulse output working mode:

| Pulse command format | pulse+direction | |
|---|---|---|
| | Forward run | Reverse run |
| Forward direction |   |   |
| Reverse direction |   |   |

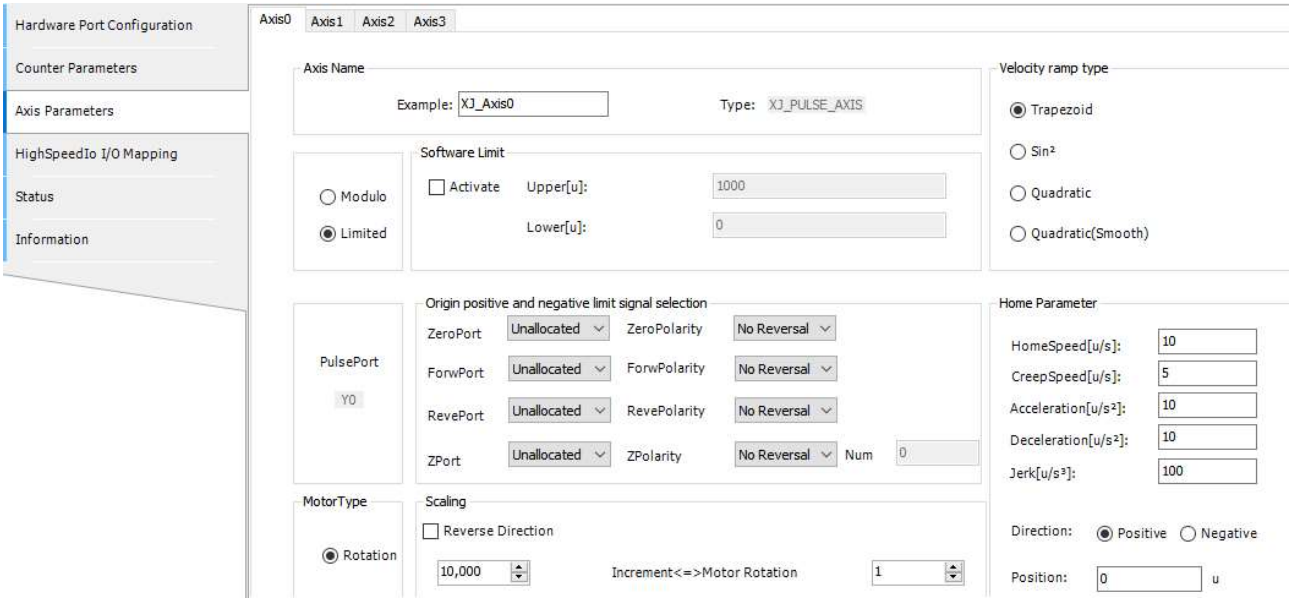
- Pulse axis parameter configuration

In the axis parameter setting interface, operations such as instantiation can be performed on configured axes. Taking axis 0 as an example, the configuration interface is shown in the following figure:

- High speed IO version: 1.0.1.0 corresponds to the following interface.



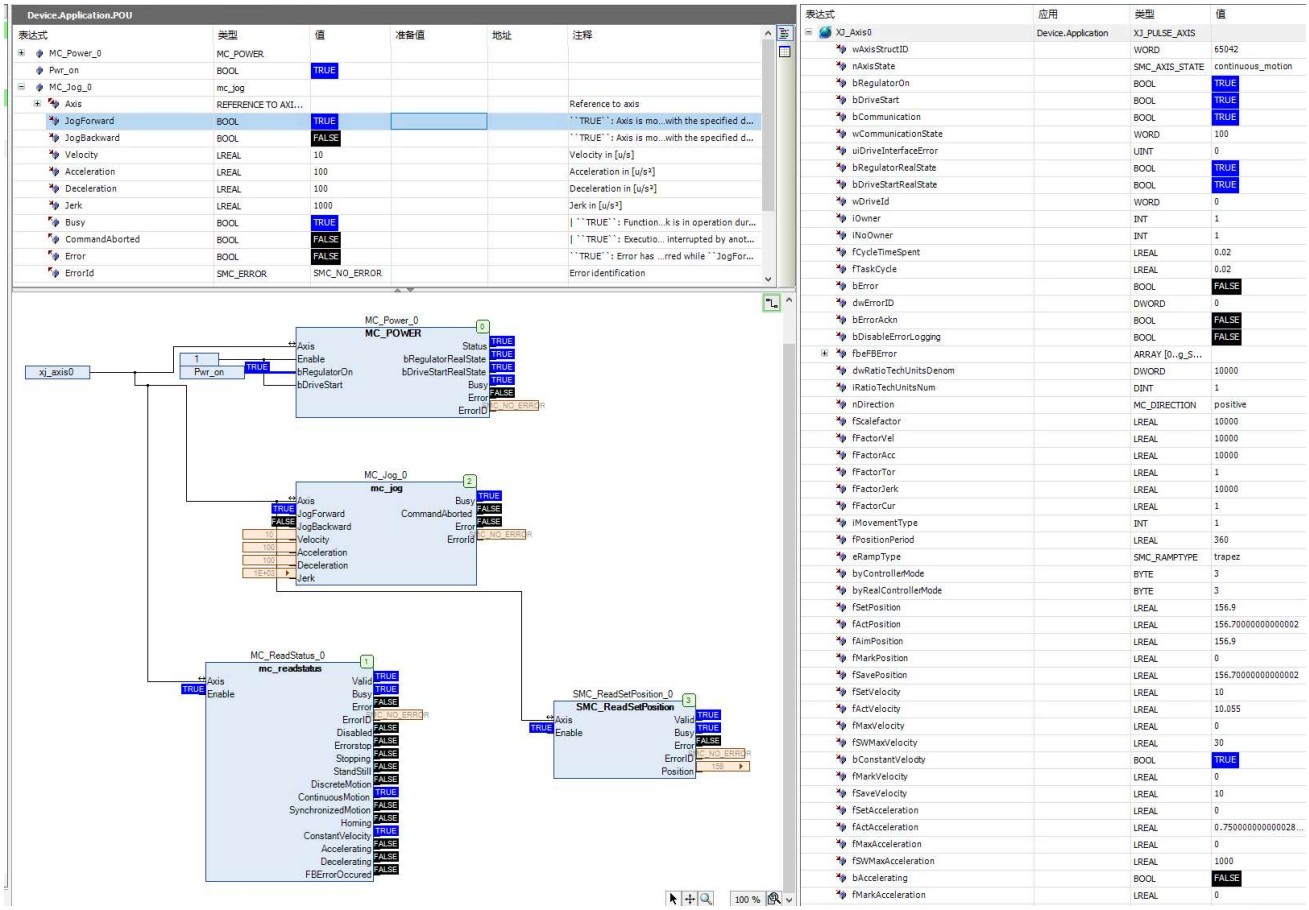

- High speed IO version: 1.1.0.0 corresponds to the following interface (new positive and negative limits and pulse homing configuration parameters).




- The default instantiation name for axis 0 is XJ_ Axis0; Support users to manually modify it.

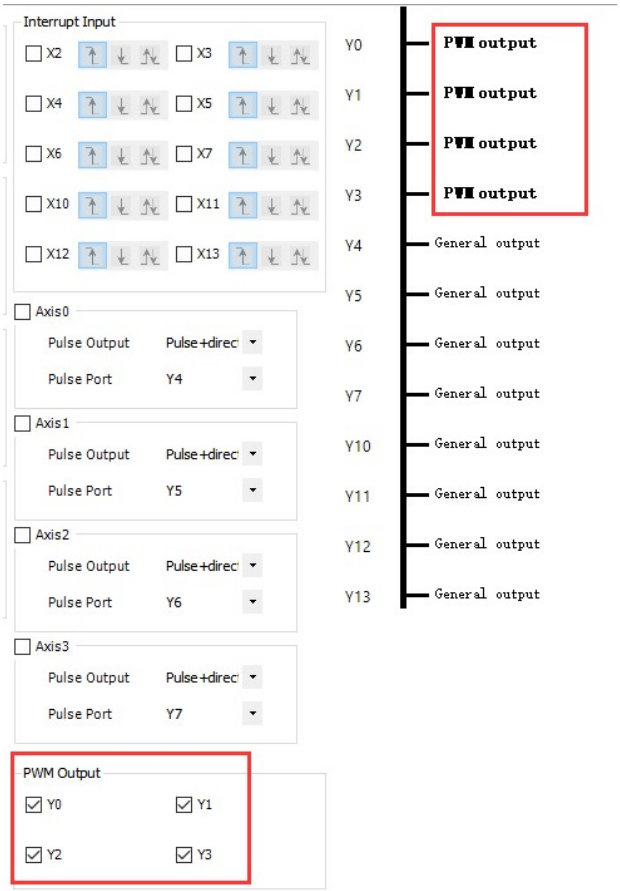
- Example to use the commands

Use commands such as [MC_POWER] and [MC_JOG] to achieve pulse axis jog, position and axis status acquisition. The configuration is shown in the following figure.

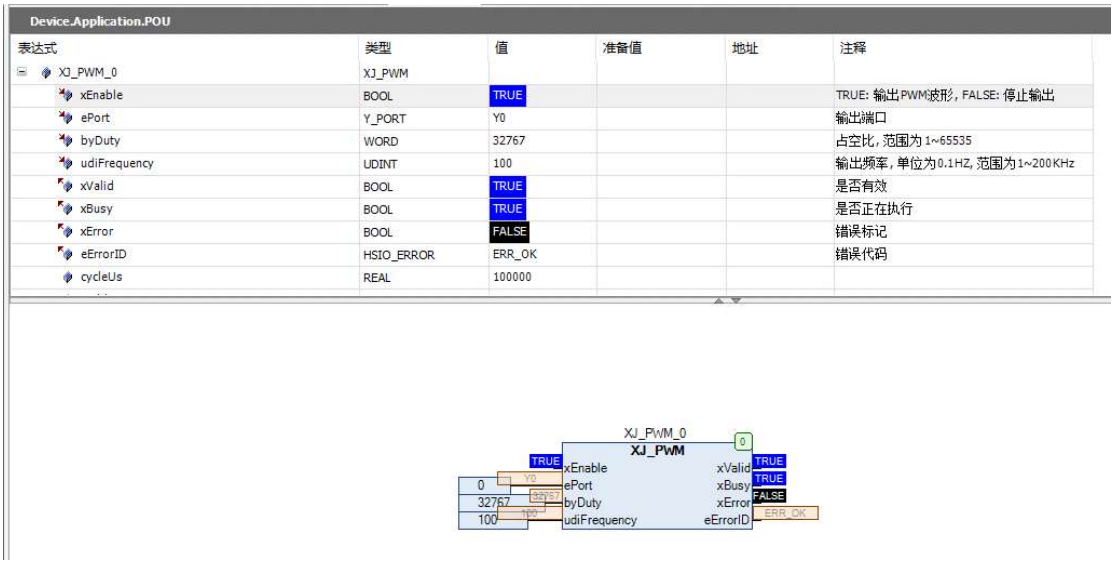



2. PWM output function

PWM output can be configured on the hardware parameter configuration interface. When using the XJ_PWM instruction, it is necessary to first check the corresponding port on the hardware parameter interface. The configuration is shown in the figure:



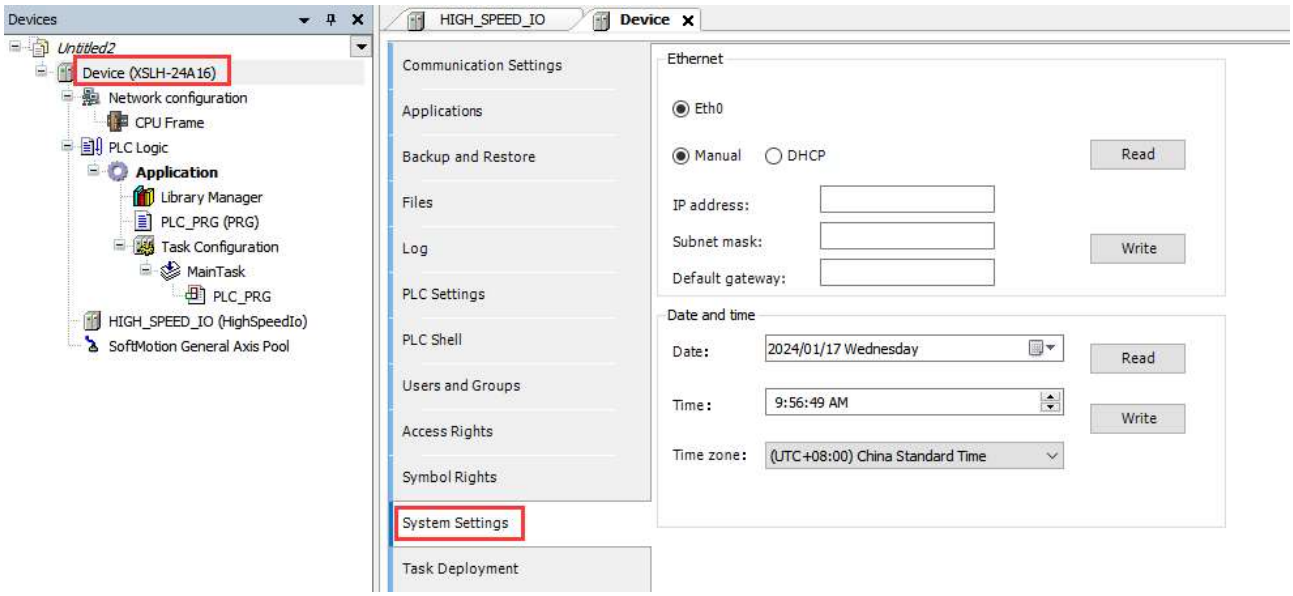
It can be used in conjunction with the pulse width modulation [XJ_PWM] command, using the Y0 terminal as an example for PWM output. The configuration is shown in the following figure:



## 7-4. System settings

■ Application for firmware 1.1.0

In the application project, double-click on the "Device" and find "System Settings". In the system settings interface, you can read/set the network port IP and system time.




**Note:**

① Read IP - If there is no Ethernet cable inserted into the network port, it is not possible to obtain all IP information of the network port.

② Write/Read Date and Time -- Simultaneously read/write date, time, and time zone information.

# 7-5. PLC commands

Note: This feature only supports XSDH, XSLH, and XS3 series

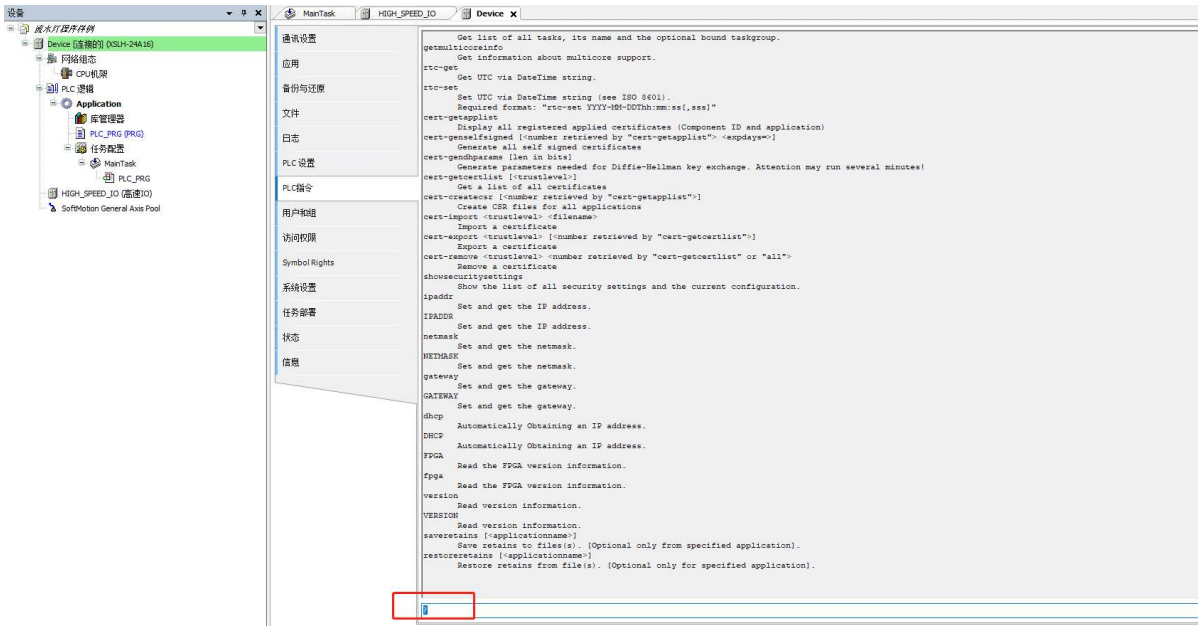
The PLC instruction function is a text-based control monitor that can be used to query specific information of the controller, input specified commands in the input window, and receive responses from the controller in the result window.

■ Command list

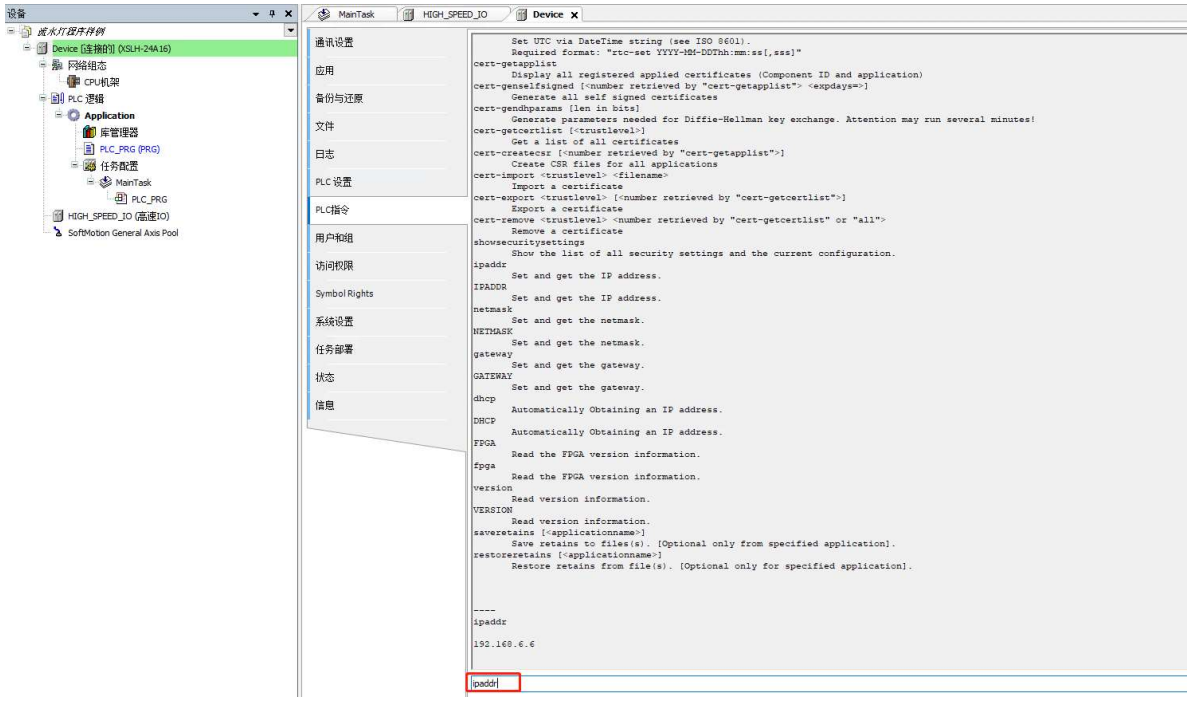
| Command name | Function |
|---|---|
| ipaddr / IPADDR | Obtain/set the IP address of the PLC |
| netmask / NETMASK | Get/Set Subnet Mask for PLC |
| gateway / GATEWAY | Get/Set PLC Gateway |
| dhcp / DHCP | Set IP to automatically obtain |
| fpga / FPGA | Obtain the FPGA version of the PLC |
| version / VERSION | Obtain the firmware version of the PLC |
| rtc-get / RTC-GET | Get the current UTC time |
| rtc-set / RTC-SET | Set UTC time |

## 7-5-1. Application example

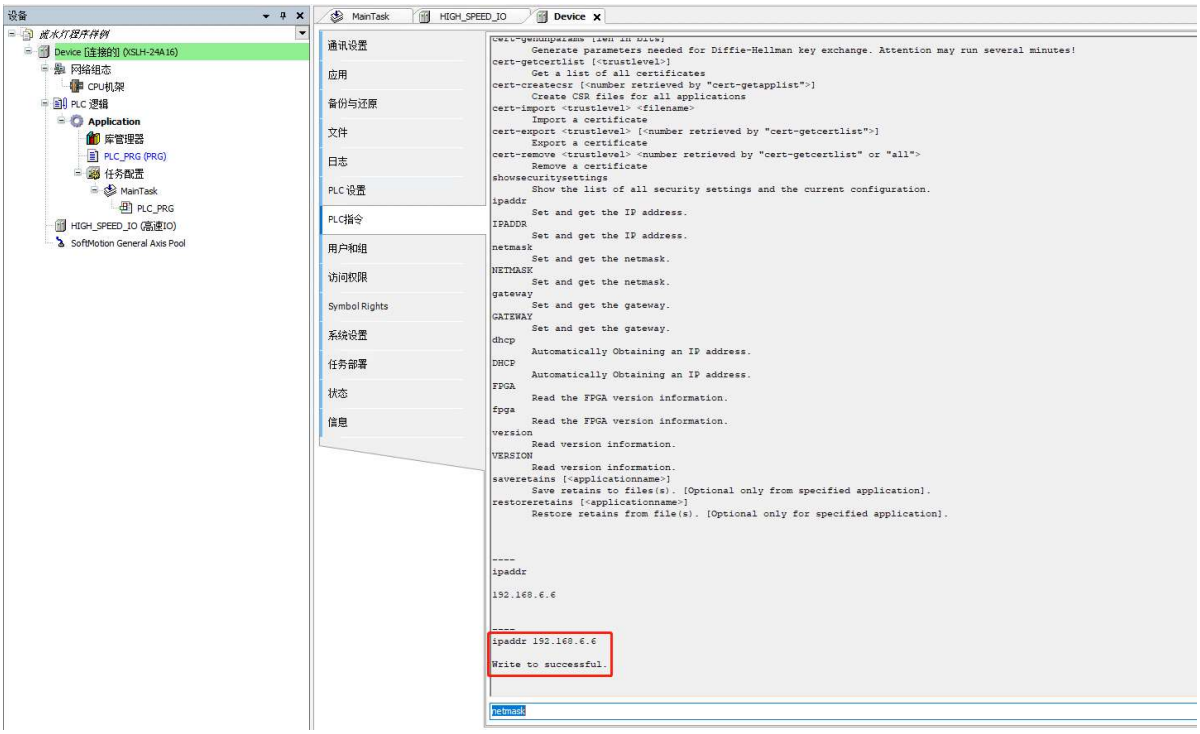
Double click on "Device" and enter "?" in "PLC shell" to display all functions. You can modify the IP here, obtain the firmware version, set/read clock information, and so on.



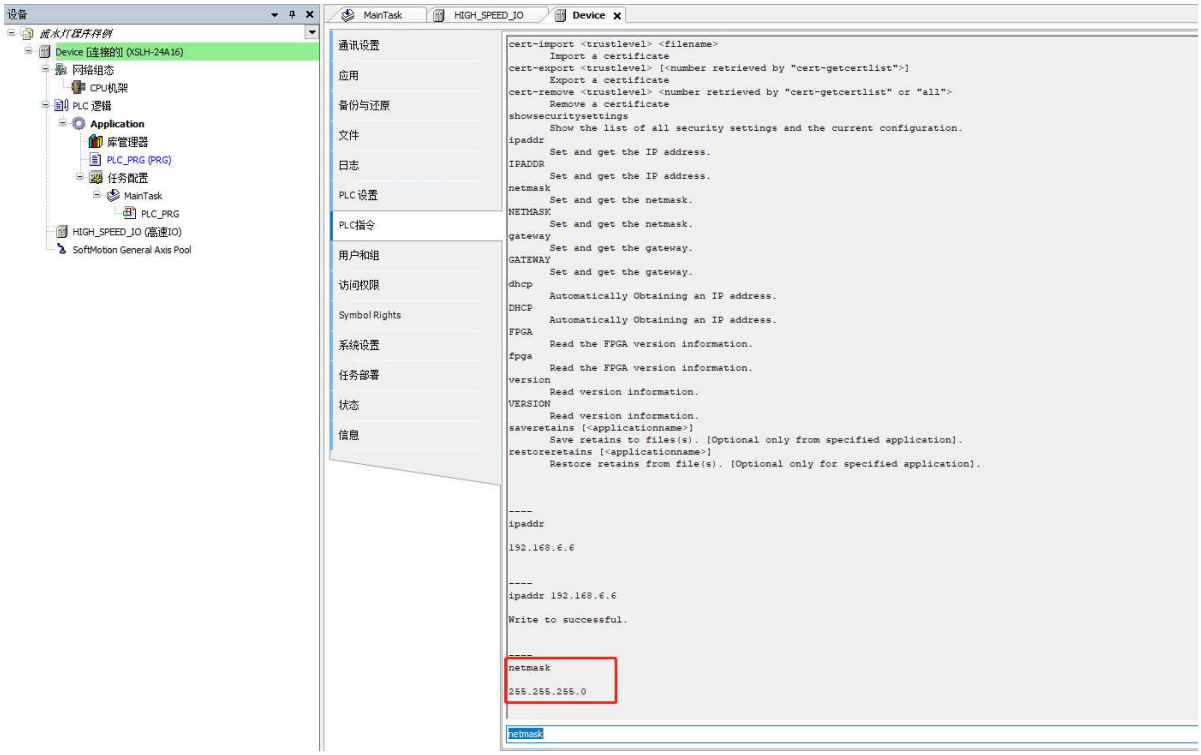
For example, entering "ipaddr" can obtain the current IP address of the PLC.



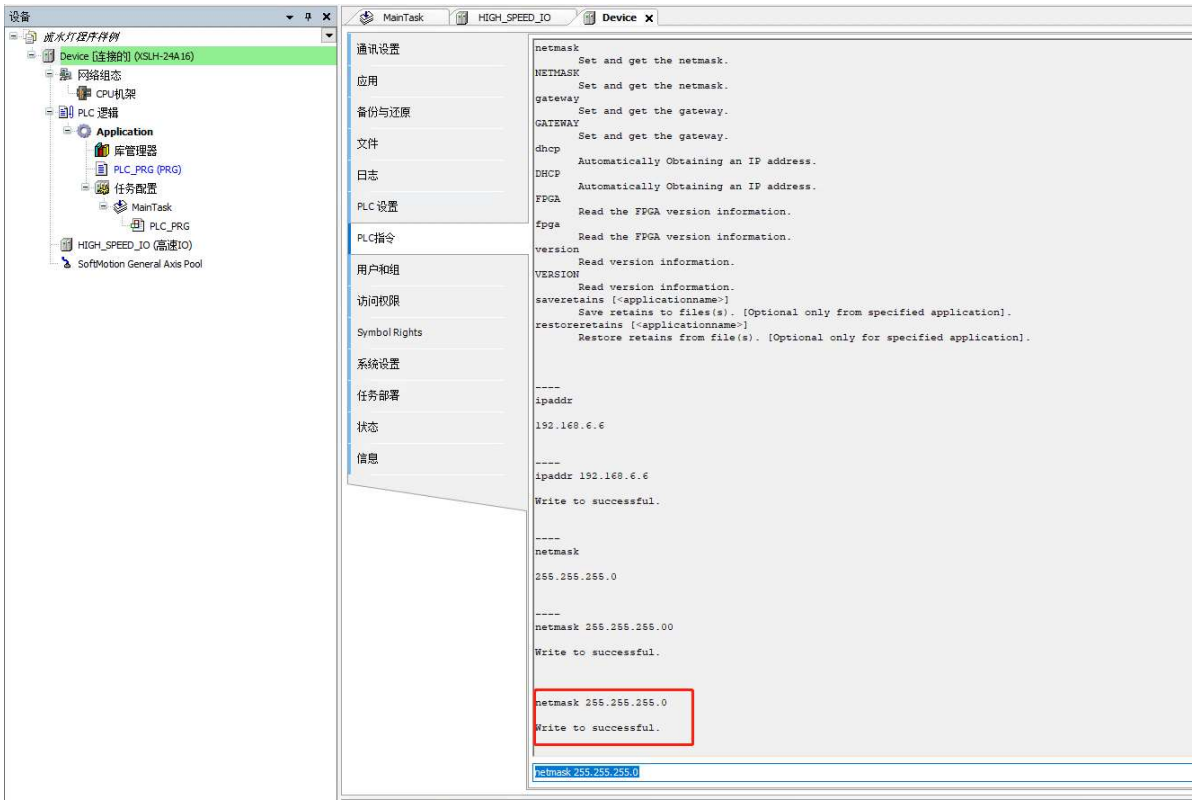

Enter "ipaddr 192.168.6.10" and set the IP address of the PLC. If "Write to successful" is displayed, the write will be successful and it will take effect when powered on again.



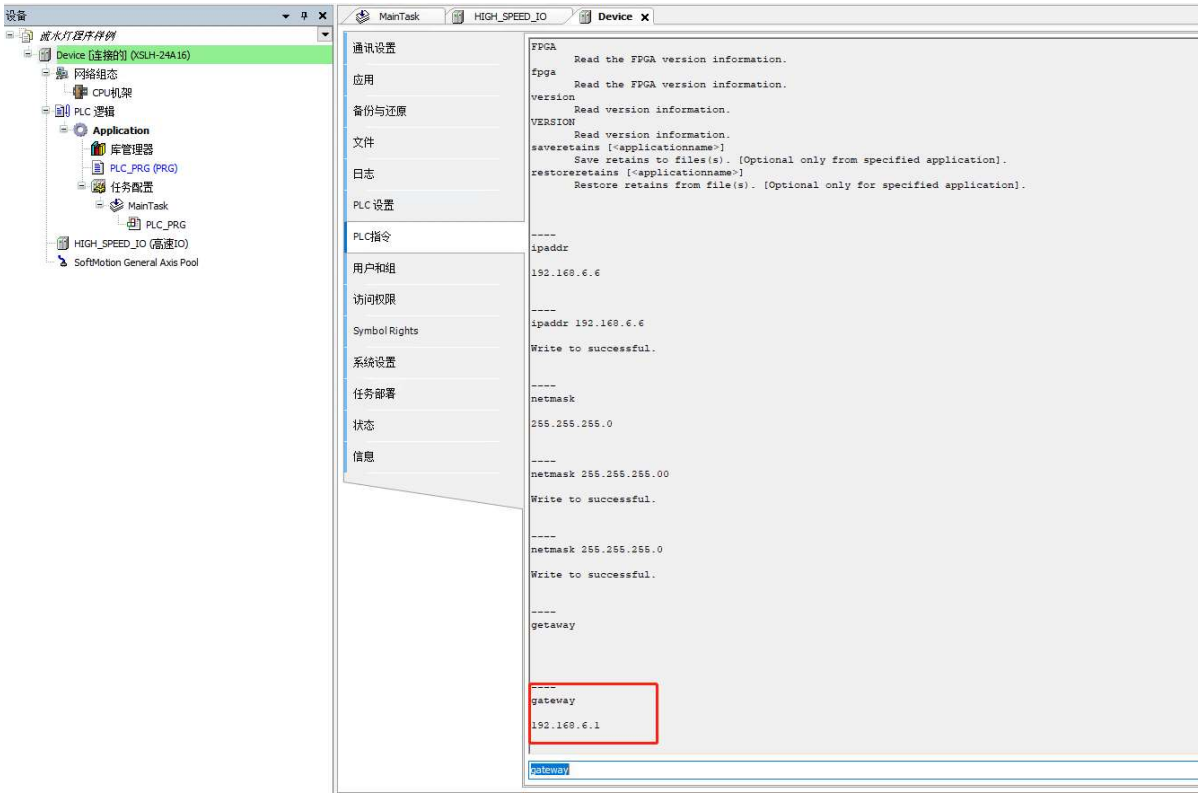
Enter "netmask" to obtain the current subnet mask of the PLC.



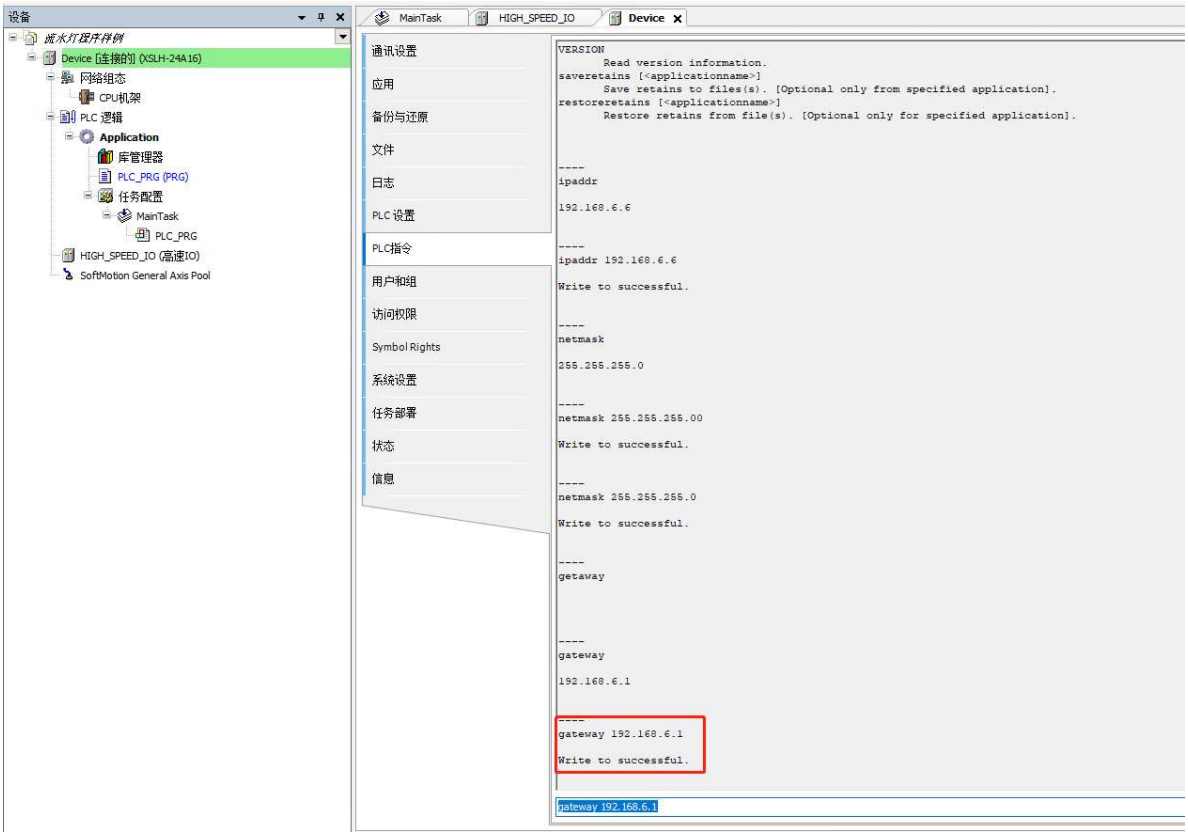

Enter "netmask 255.255.254.0", set the subnet mask of the PLC, and display "Write to successful" to indicate successful writing.



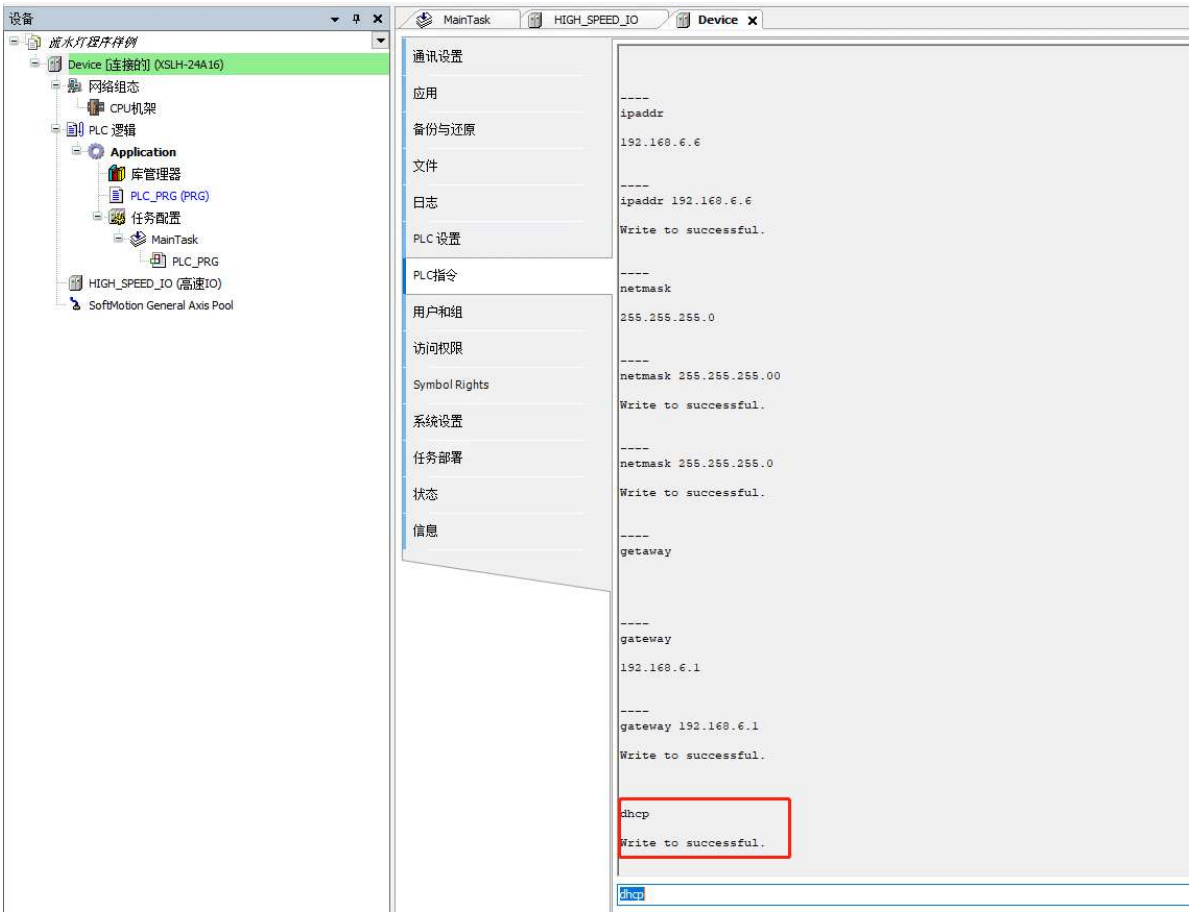
Enter "gateway" to obtain the current default gateway of the PLC.



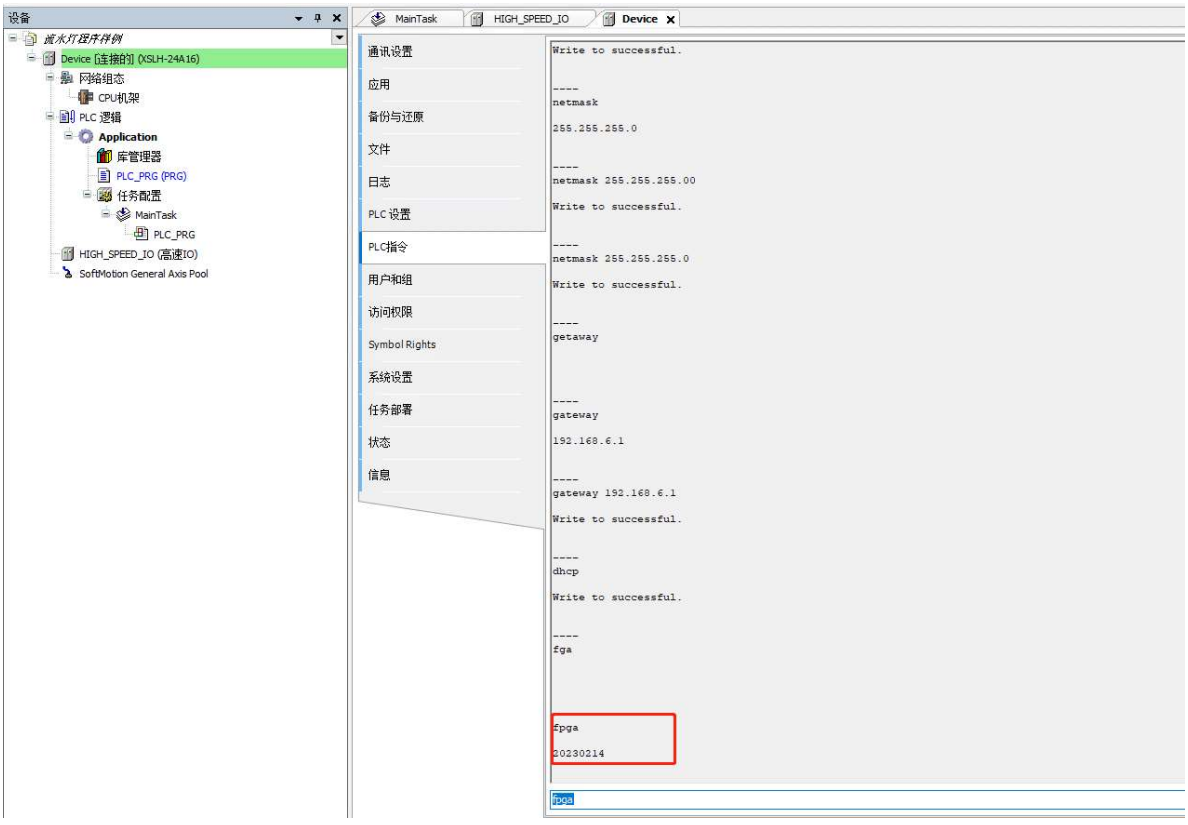

Enter "gateway 192.168.6.1" and set the PLC gateway. If it displays "Write to successful", the write will be successful.



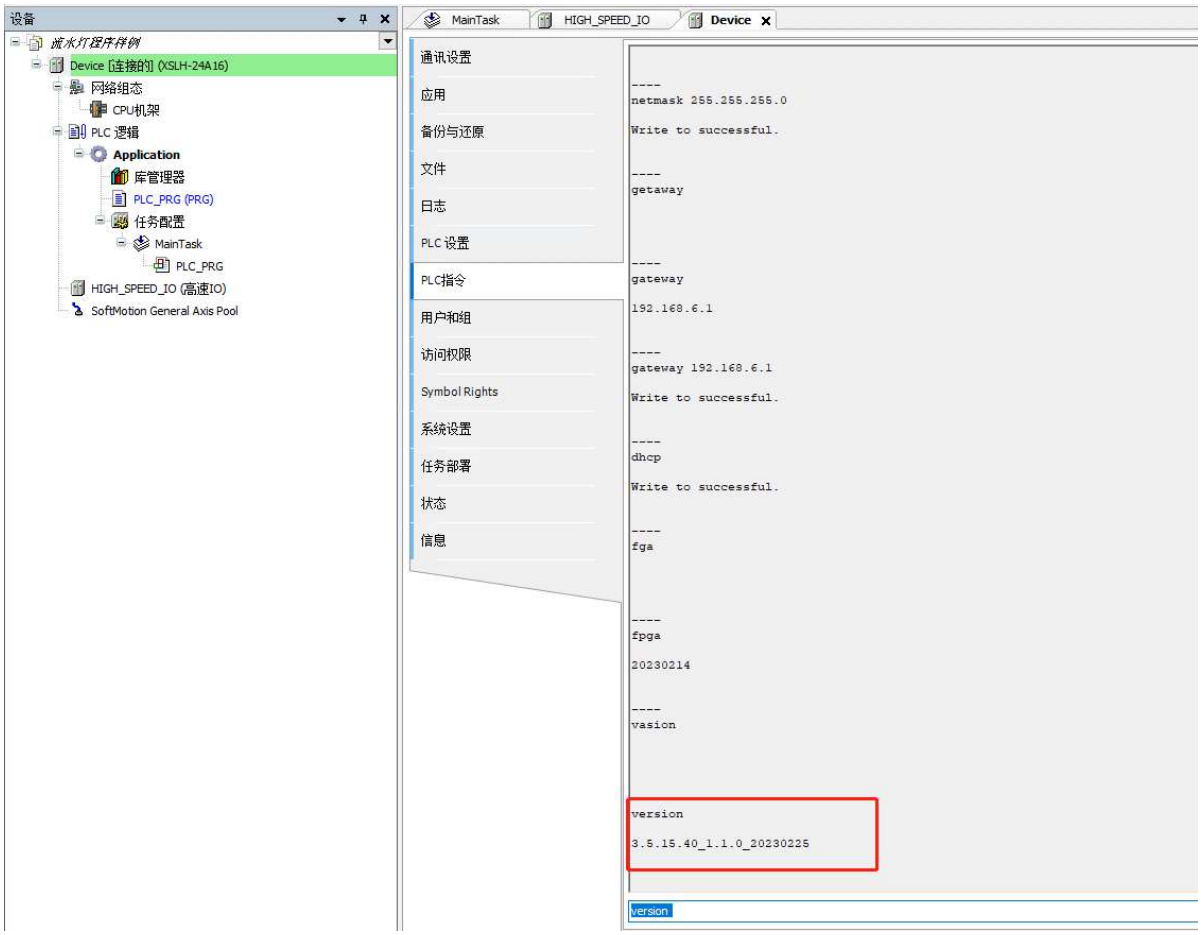
Enter "dhcp" and set the PLC's IP acquisition method to automatic obtain. If "Write to successful" is displayed, the write will be successful. When the IP acquisition method is automatic, it is necessary to ensure a good network environment.



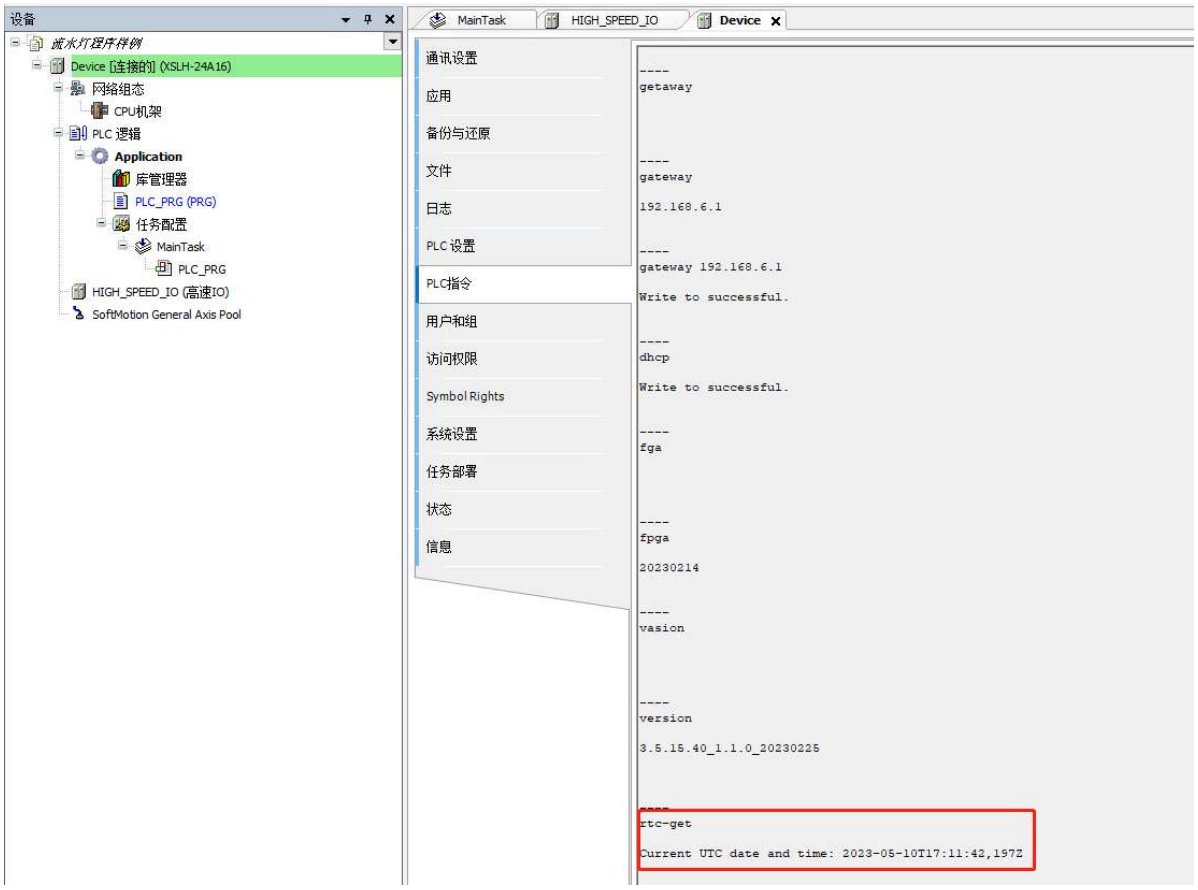

Input "fpga" to obtain the current FPGA version of the PLC.



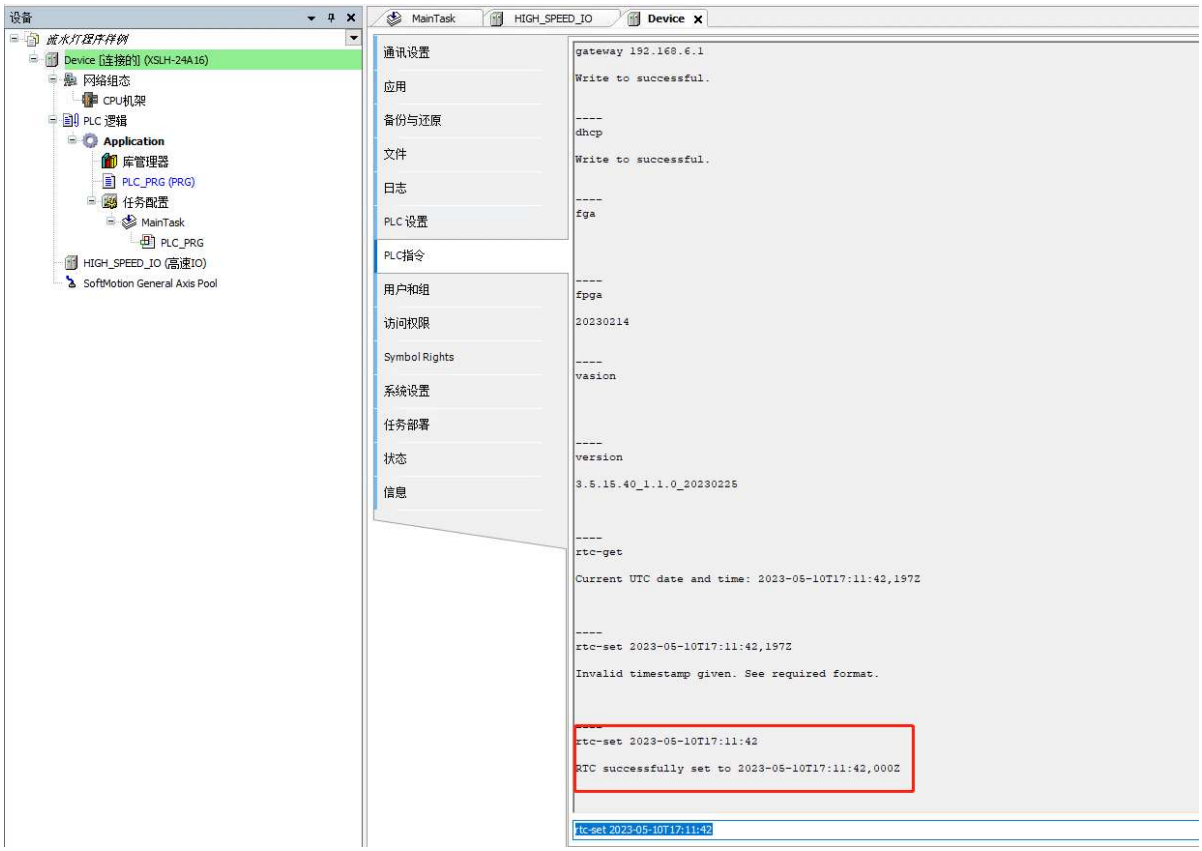
Enter "version" to obtain the current firmware version of the PLC.




Enter "rtc get" to obtain the current UTC time.



Enter "rtc-set 2021-10-25T18:24:30" to set the UTC time. If "RTC successfully set to 2021-10-25T18:24:30000Z" is displayed, the write is successful. The display of "000Z" is uncertain.



# 7-6. Clock

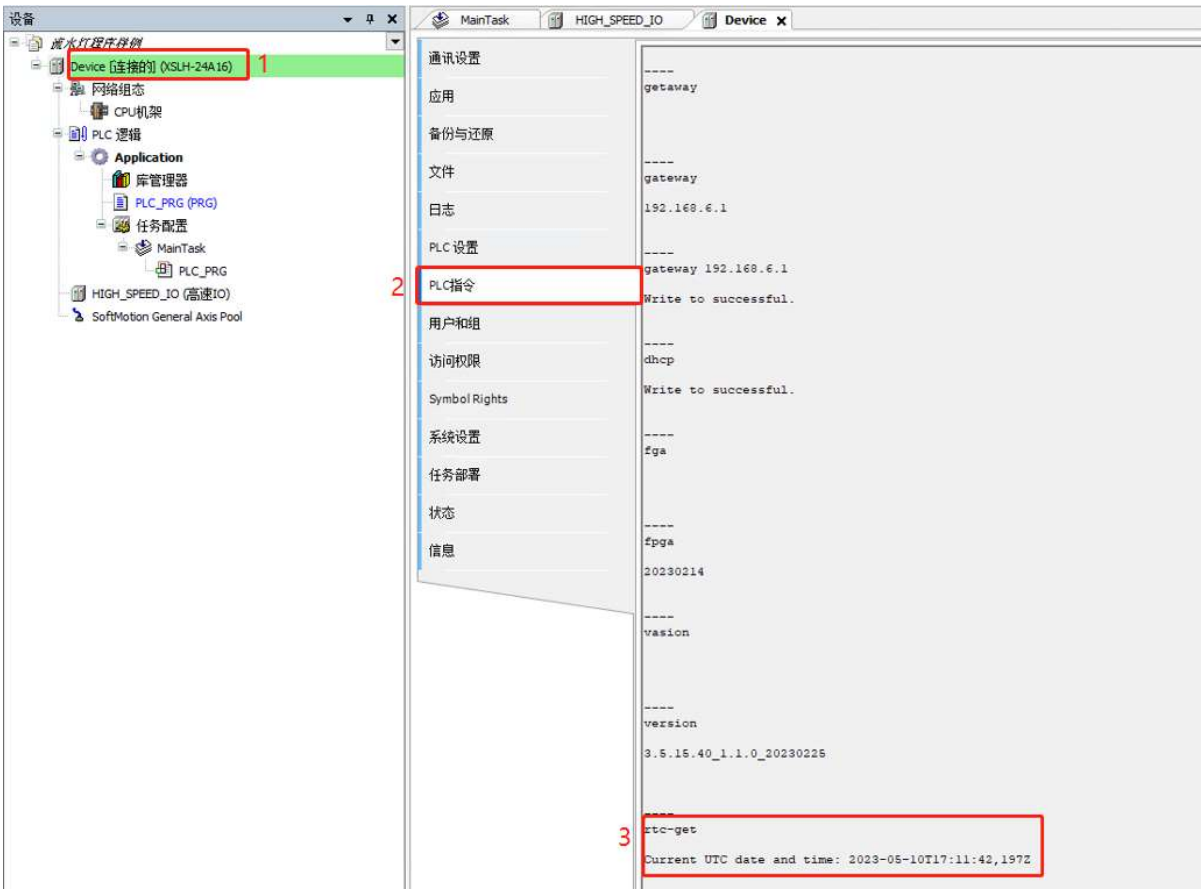
## 7-6-1. Function overview

The XS series PLC integrates RTC, which is used to record the current system time. The clock is powered by batteries, ensuring the accuracy of time and also supporting users to manually modify RTC time.

## 7-6-2. Application example

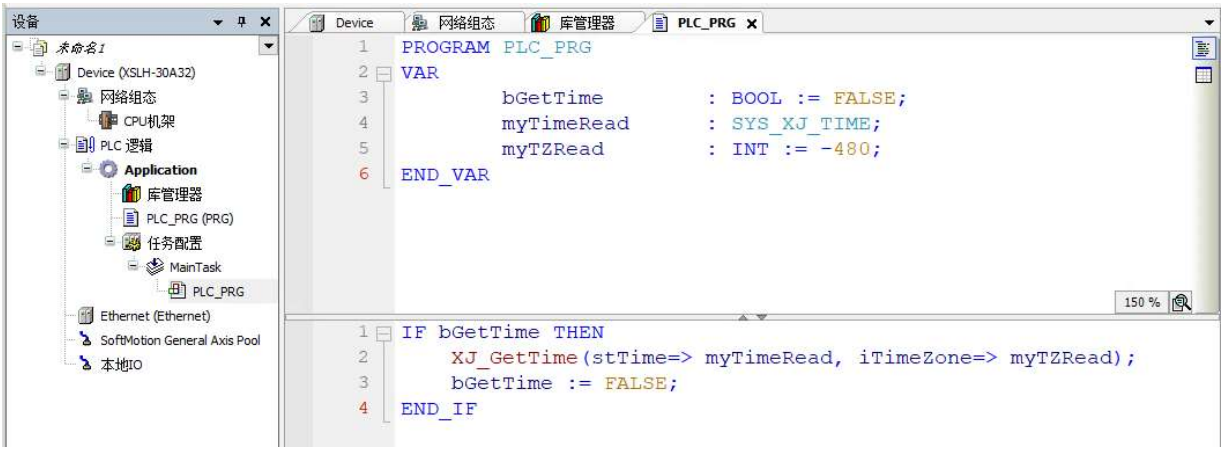
How to obtain time:

1. Double click on "Device" and enter "rtc-get" in the "PLC shell" to obtain the current time.

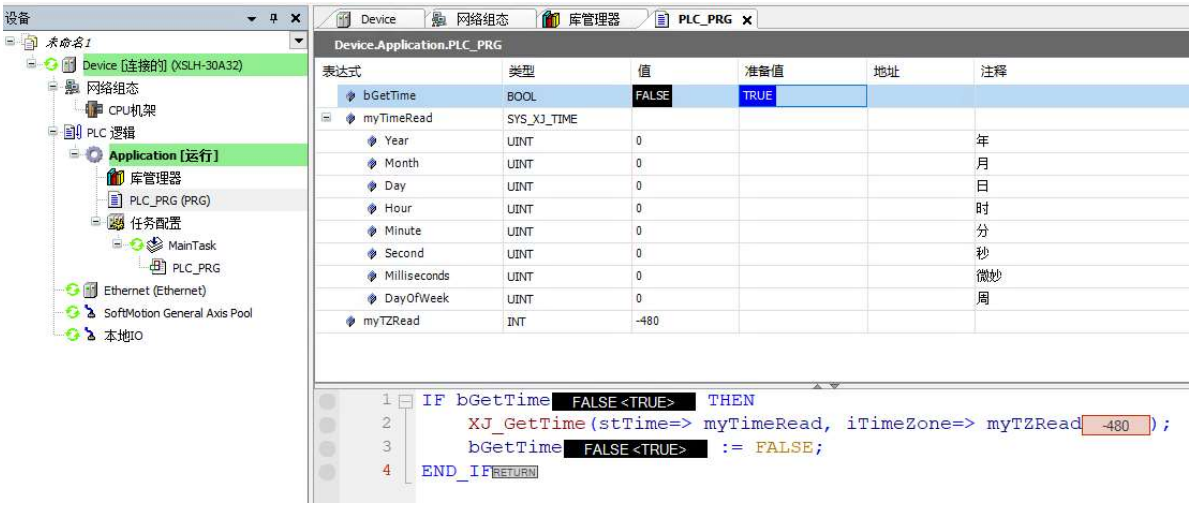


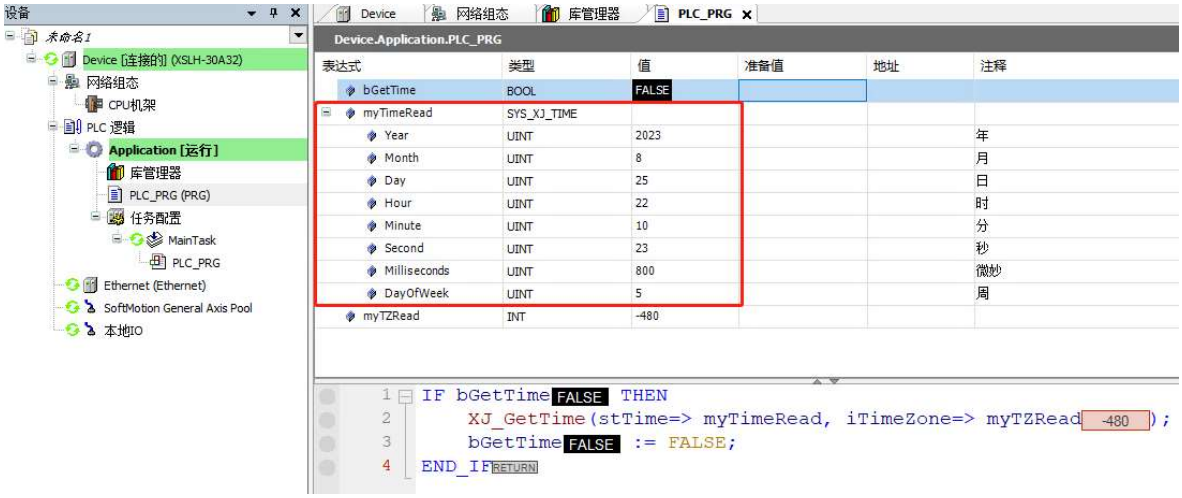

2. Clock instruction

(1) Open the software and write XJ_GetTime instruction in the PLC-PRG editor. As shown in the following figure.




(2) Establish a connection with the PLC device, log in and run it. As shown in the following figure.




After running, the time has been correctly read and displayed.

# 8. Appendix: Q&A

## 8-1. Package

### 8-1-1. Package naming rule

Naming rule: XSDH-60A32_3.5.15.40_1.0.0_P1_20211027
①　②　③　④　⑤

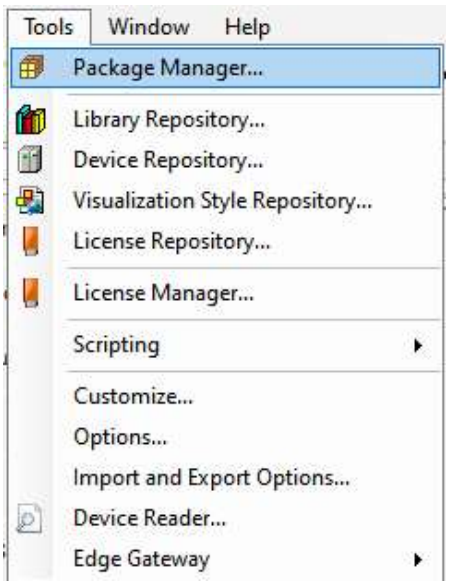
| Number | Name | Note |
|---|---|---|
| ① | XSDH-60A32 | PLC model |
| ② | 3.5.15.40 | Runtime version |
| ③ | 1.0.0 | Package version |
| ④ | P1 | The first online upgrade package after production |
| ⑤ | 20211027 | Package update date |

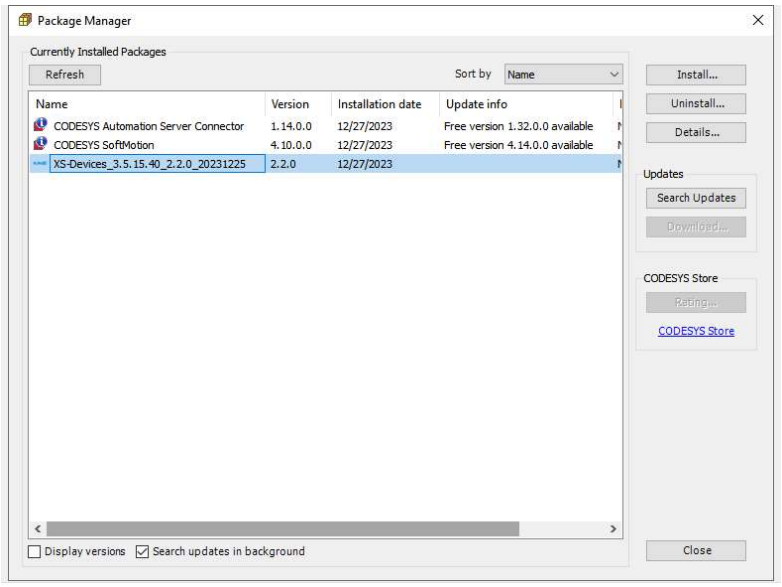
### 8-1-2. Package

Please obtain the package on our webiste or contact technical support, website address: www.xinje.com; Technical service hotline: 400-885-0136.

### 8-1-3. Package installation

Select "Tools" - "Package Manager", install the Package in the pop-up interface, select "Install", find the location of the Package, and install it. For example, if you want to install the XSLH-24A16 package, it is best to uninstall the previous package before installing the new one.



# 8-2. XS series PLC firmware update

## 8-2-1. Firmware naming rule

Naming rule: XSDH-60A32_3.5.15.40_1.0.0_P1_20211027

① ② ③ ④ ⑤

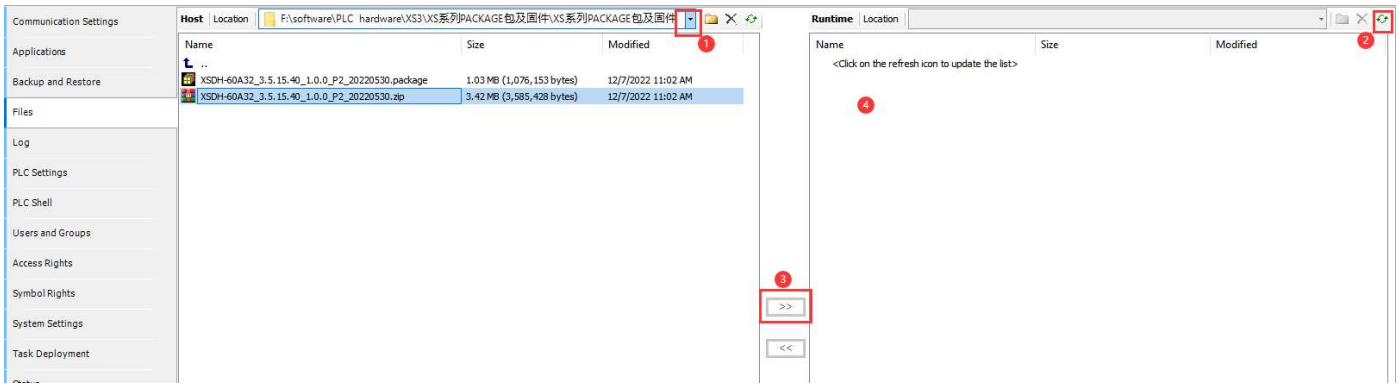
| Number | Name | Note |
|---|---|---|
| ① | XSDH-60A32 | PLC model |
| ② | 3.5.15.40 | Runtime version |
| ③ | 1.0.0 | Firmware production version |
| ④ | P1 | The first online firmware upgrade after production |
| ⑤ | 20211027 | Firmware upgrade date |

## 8-2-2. Firmware obtain

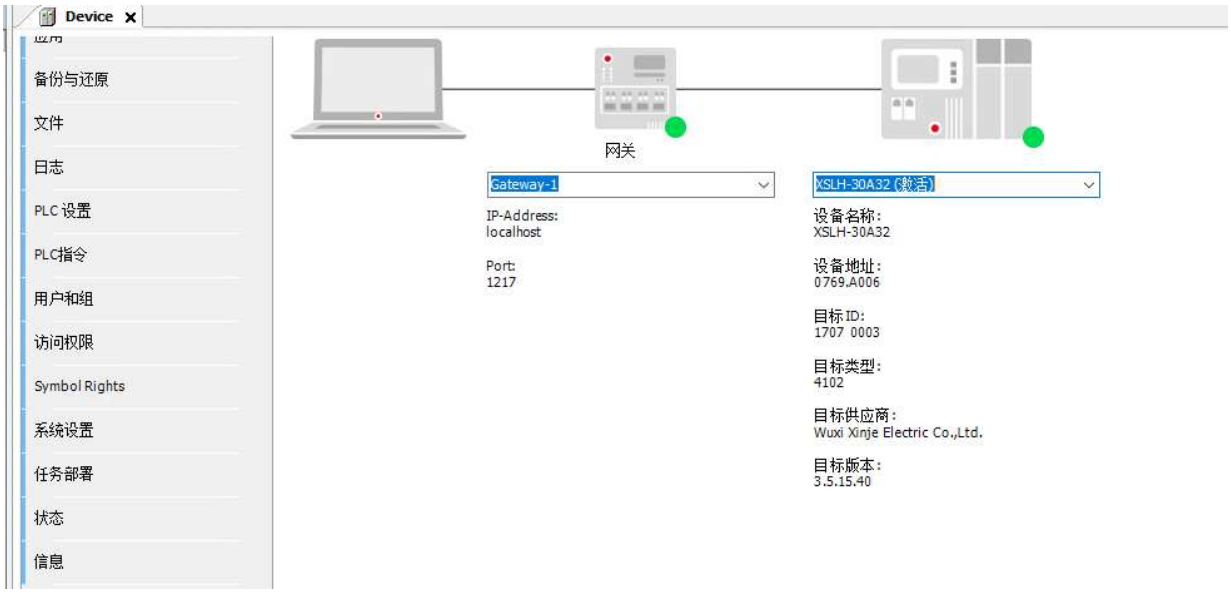
Please contact us, email address is sales@xinje.com.

## 8-2-3. Firmware installation and precautions

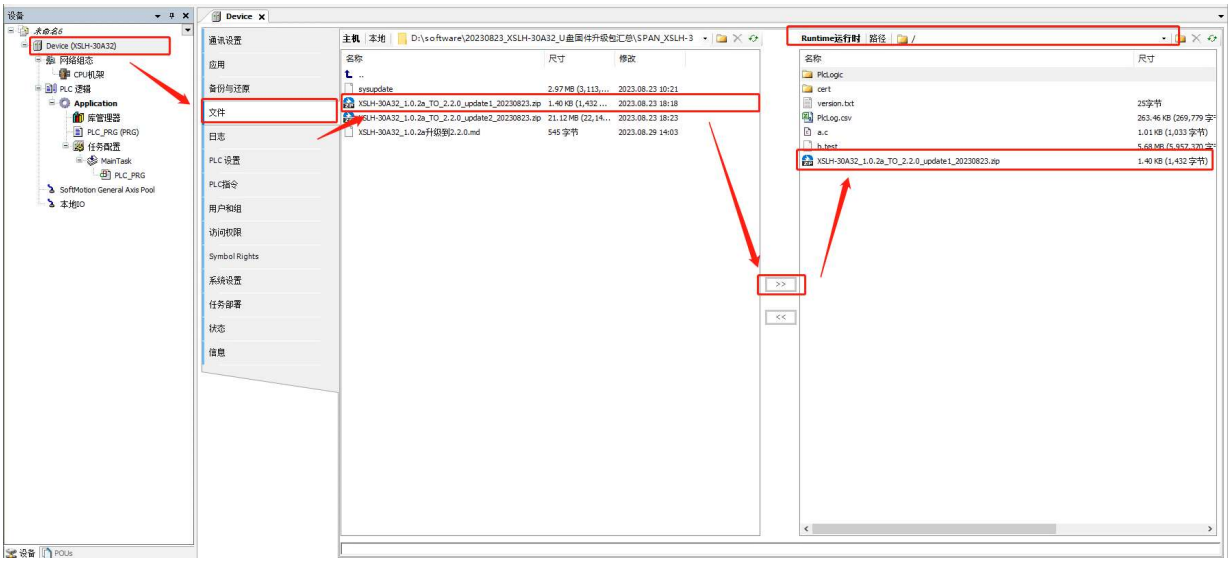
Method 1: Upgrade firmware through newpack package:
Create a device standard project, connect the device, select the "File" option in the main device directory, click "Refresh" in the upper right corner, transfer the newpack upgrade package to runtime, wait for the transfer to complete, restart the device, and the ERR light will remain on during the upgrade. After the update is completed, the ERR will turn off, and the device can be scanned.




Method 2: Upgrade firmware version V1.0.2a or V1.1.0 to V2.2.0
Note: This method is only applicable to ARM series models(XSLH, XSDH, XS3).
Here, taking upgrading XSLH-30A32 model equipment as an example, the operation steps for other types of equipment are the same.
(1) Establish a connection between the upper computer and PLC equipment, as shown in the following figure:

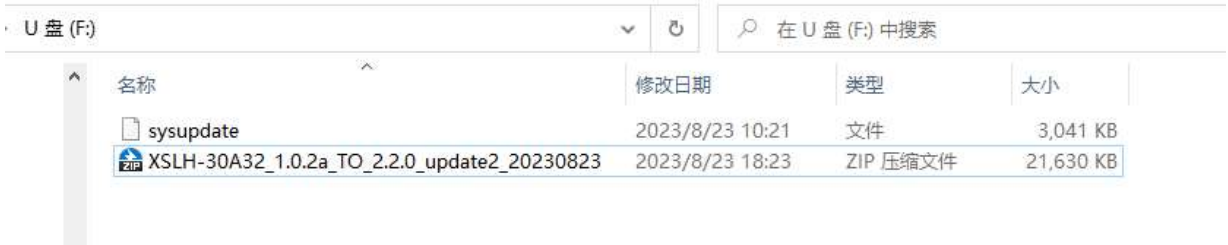



(2) In the "Files" window of the "Device", select the local file "XSLH-30A32_1.0.2a_TO_2.2.0_update1_20230823.zip" and send to the PLC runtime root directory (/), as shown in the following figure:



(3) Restart the PLC after power failure. During the upgrade process, the ERR light flashes for approximately 1-2 seconds; After the upgrade is completed, the RUN light will light up.

(4) Copy the file “XSLH-30A32_1.0.2a_TO_2.2.0_update2_20230823.zip” and “sysupdate” to the root directory of the SD card; As shown in the following figure:




(5) Power off PLC, insert SD card, and power on; After power on, the PWR light remains on, but the ERR light flashes and goes off. At this time, only the PWR light is on.

(6) Power off, remove SD card, and power on; You can scan the connection.

**Note:**

- Users are not allowed to modify the name of the PLC firmware upgrade package without authorization;
- The USB drive or SD card is in FAT32 or NTFS format;

- The PLC firmware upgrade package can only be placed in the root directory of the USB drive/SD card, and cannot be placed in other subdirectories. Only one upgrade file can be placed, and multiple copies are not allowed. Otherwise, it will not be executed;
- Before the firmware upgrade of the USB flash drive/SD card is completed, it is recommended not to unplug the USB flash drive/SD card. The ERR light flashes for at least two seconds, indicating that the upgrade is in progress. At this time, the ERR light goes off, indicating that the upgrade is complete and only the PWR light is on. At this time, the USB drive/SD card can be unplugged. If the ERR light remains on at this time, it indicates that the update has failed;
- After the firmware upgrade is completed, the original program will be initialized. If the user wants to run the program, they need to download it again;
- Do not power off during firmware upgrade process;
- If the upgrade fails, unplug the USB drive/SD card, power on again, and run the original program;

Method 3: Upgrade method for firmware version V2.2.0 and above (this method will be used for subsequent firmware upgrades)

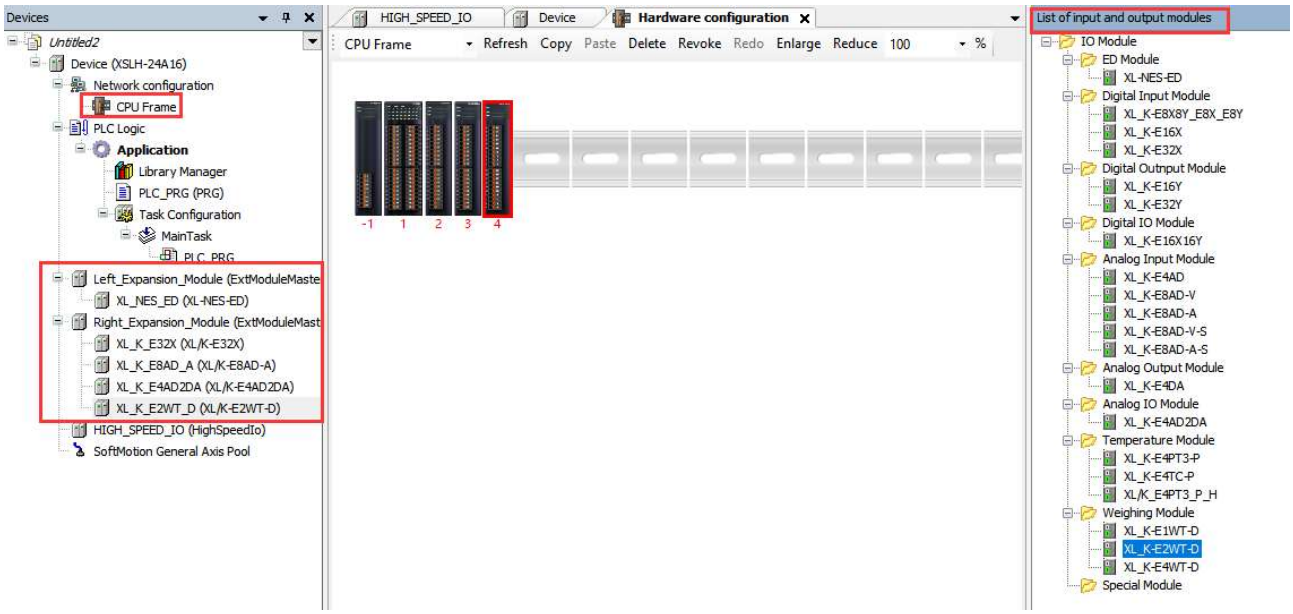
**Note:**

① PLC firmware V2.2.0 and above support firmware upgrade through USB drive or SD card.

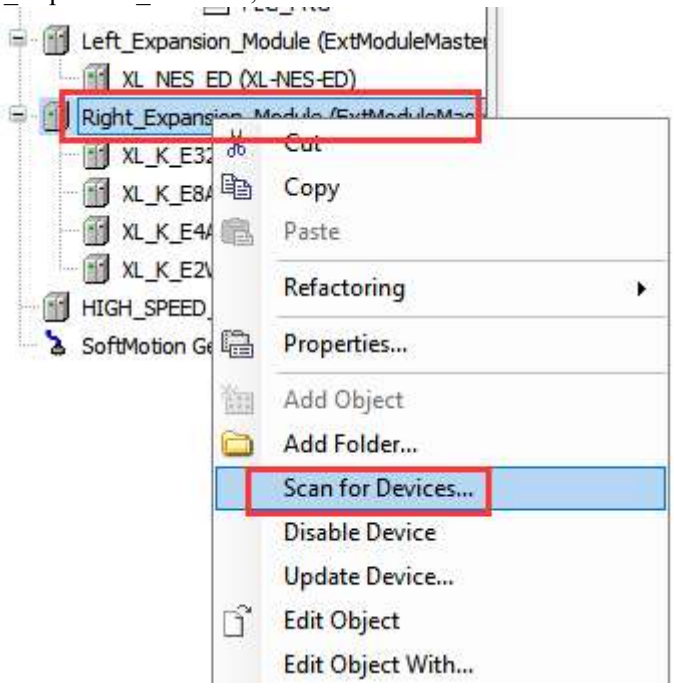
② Currently, USB drives are used to upgrade X86 industrial control equipment (XSA series), and SD cards are used to upgrade ARM equipment (XSLH, XSDH, XS3 series).

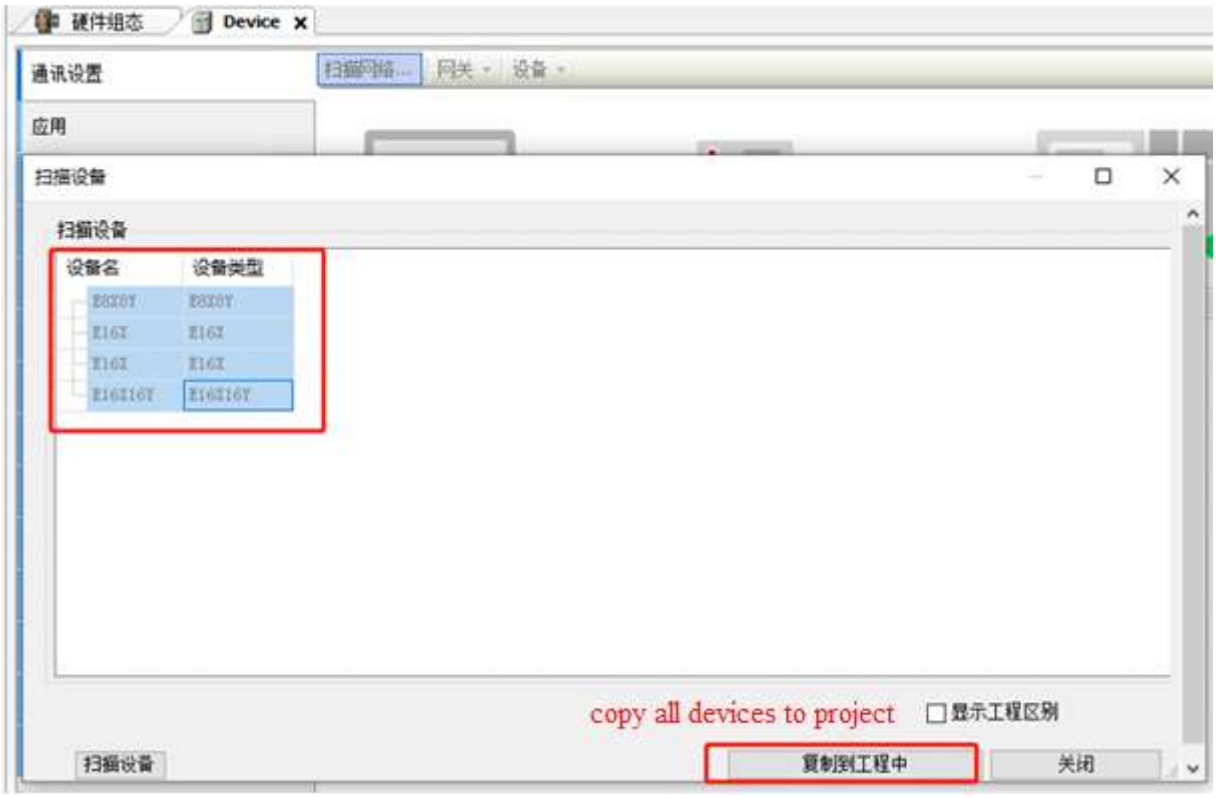
## 8-3. XS series local expansion modules

① Double click on the CPU frame bus node under the network configuration node to open the local hardware configuration interface and the "I/O module list" interface on the right. Local IO modules can be added through the "Input/Output Module List". As shown in the figure.

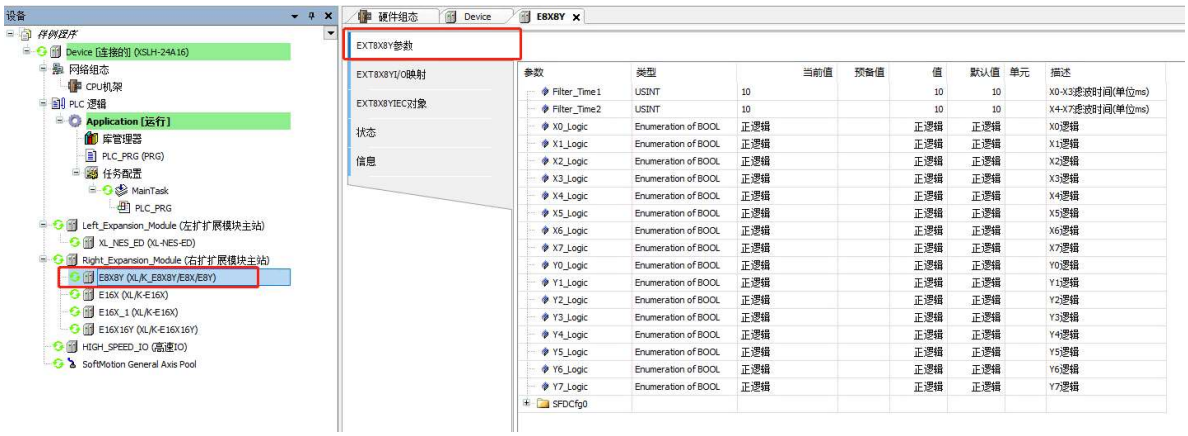



② Right click “Right_Expansion_Module”, select “scan for devices” to add the right expansion modules.



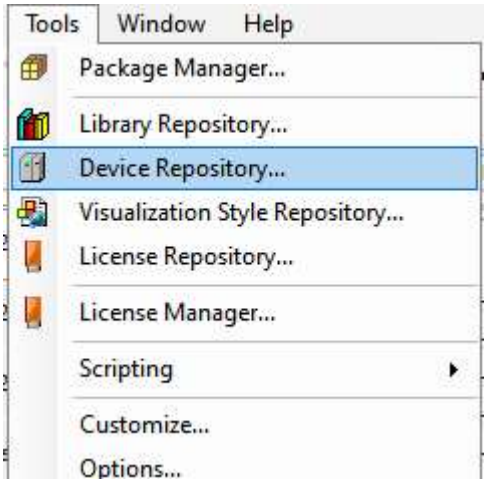


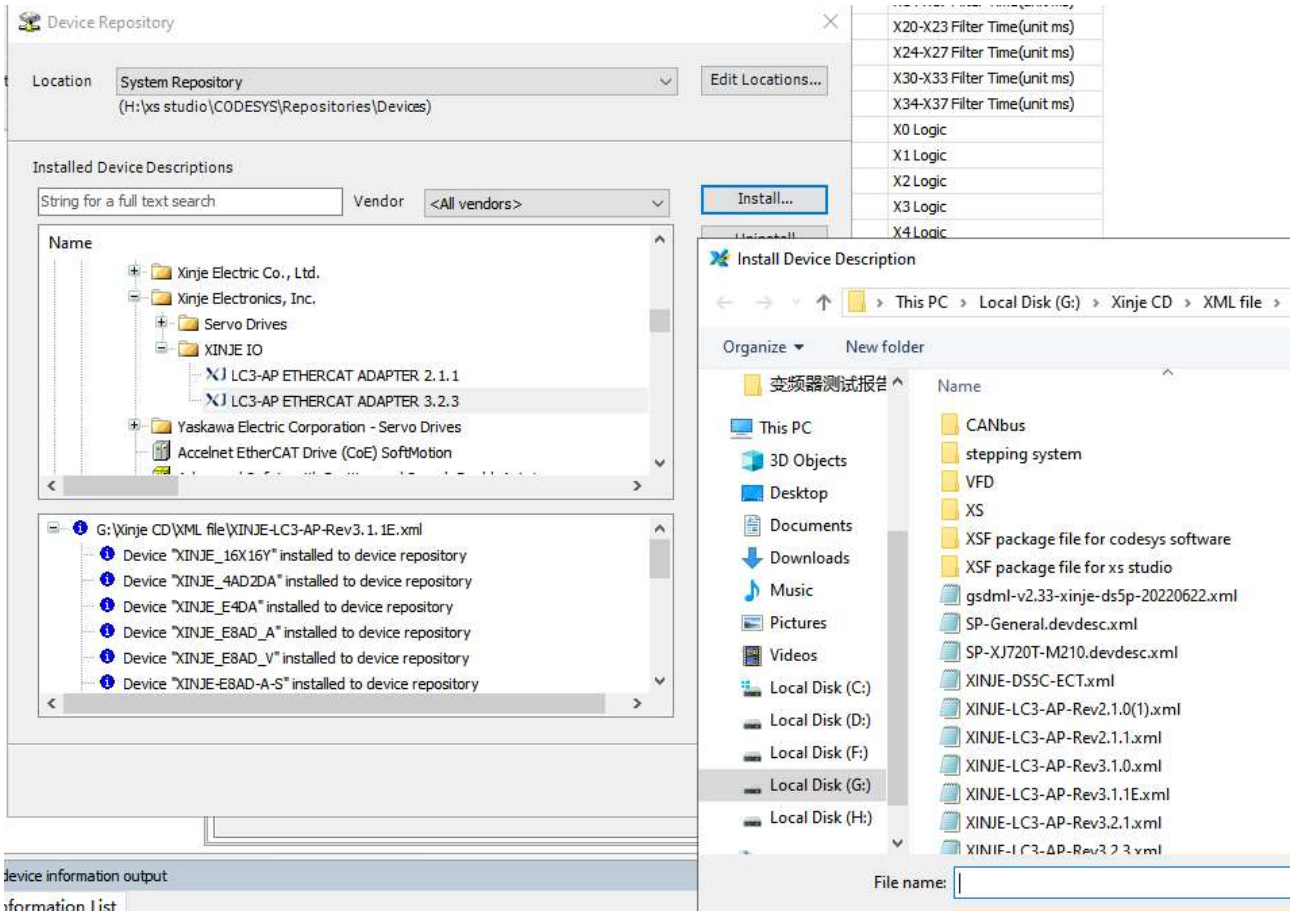

③After scanning and adding, connect the PLC device and log in to run it. As shown in the following figure.



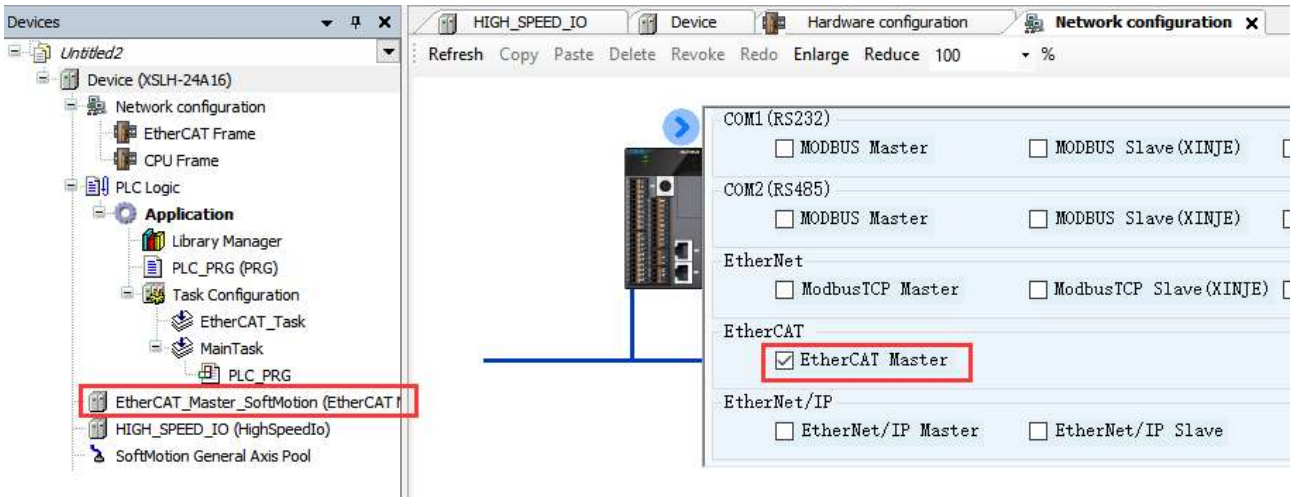
## 8-4. XS series remote expansion modules

① Connect the LC3-AP remote module to a 24V power supply.
② Add LC3-AP xml file.

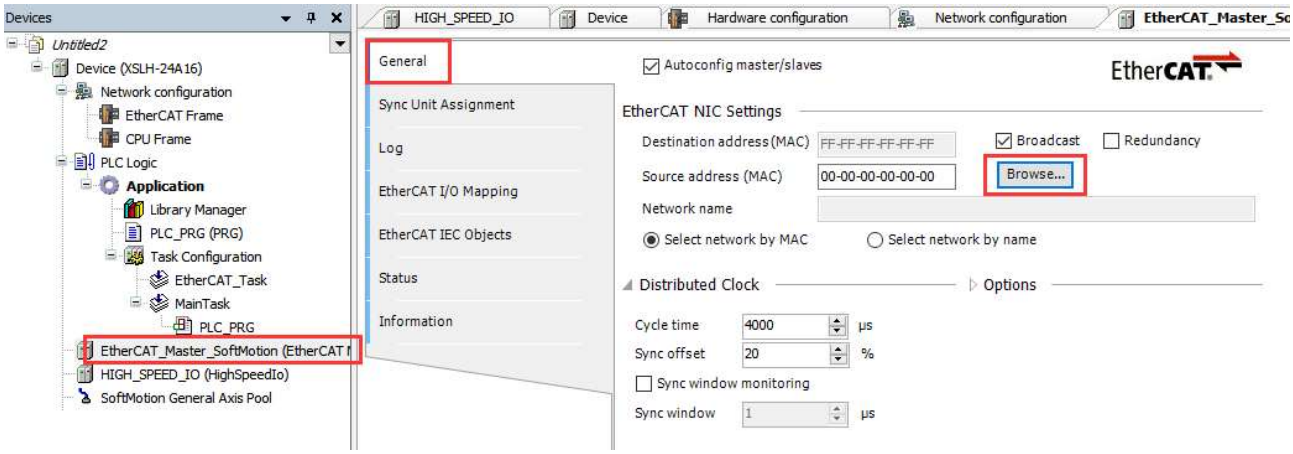


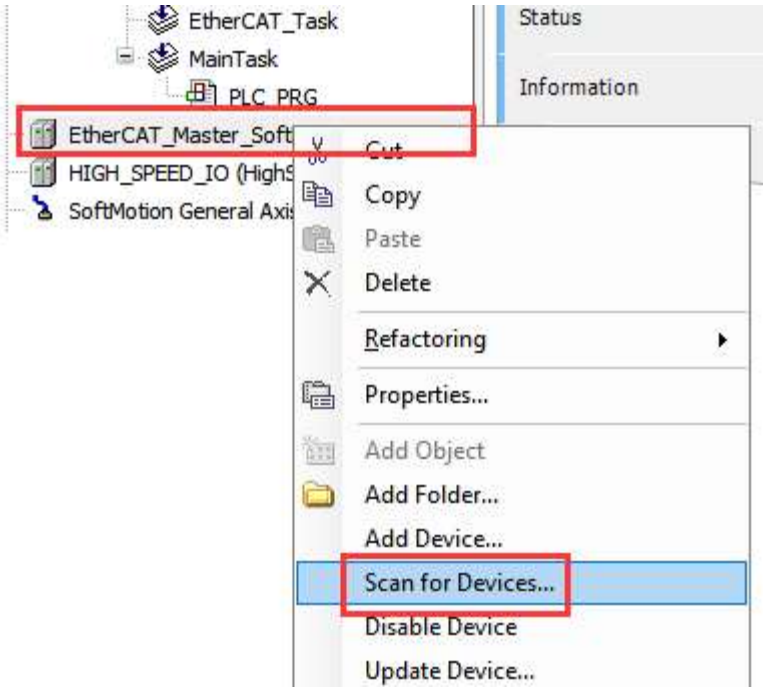

③ Add EtherCAT master station.




④ Select the network port for communication.




⑤ Scan to add the LC3-AP module.



⑥ Copy all devices to the project.


## 8-5. Dial switch

XSDH-60A32-E supports dialing function, and its specific functions are as follows:
00: Normal startup, no special handling, loading user program;
10: Initialize IP;
01: Power on without loading user program.

## 8-6. After install XS Studio and compile, there are many errors

Generally speaking, it is caused by missing libraries. In the project bar, double-click to open the library manager, click to download the missing library, and wait for the missing library to be downloaded.

## 8-7. The gateway displayed red point

It is possible that the gateway service has been shut down. You can open the service "Codesys Gateway V3" in the Task Manager or restart your computer.

## 8-8. There are warnings after adding multiple EtherCAT slave stations

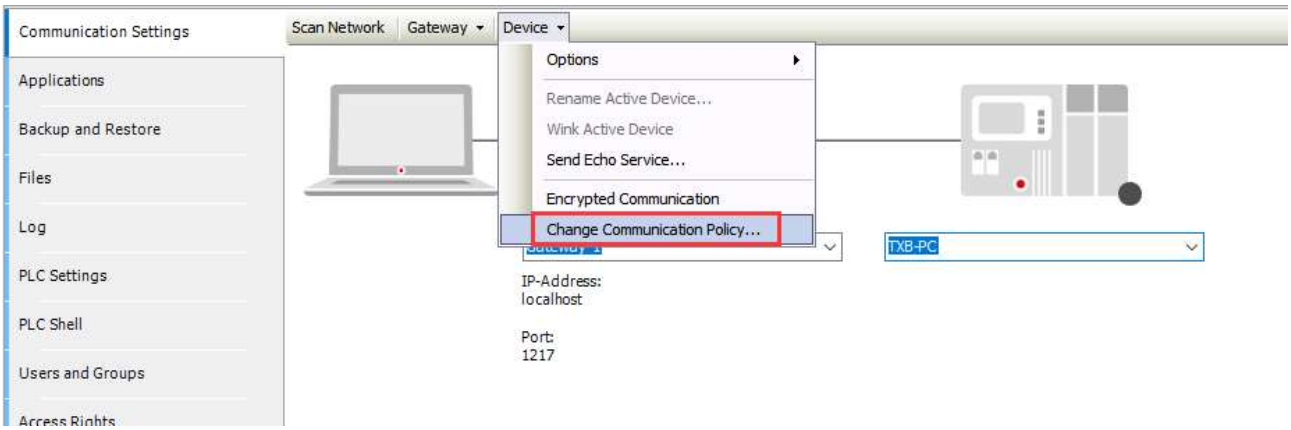
It is because the servo station number is duplicate, which will not affect use. If you want to clear the warning and double blue underline, scan the servo again, and then modify the duplicate station number.

## 8-9. Once the EtherCAT axis running, the communication will disconnect

EtherCAT related POUs must be placed under EtherCAT tasks as they have a position synchronization cycle.

## 8-10. How to cancel the password login

(1) In the "Device" section of the scanning device interface, click on "change communication policy". In the pop-up interface, select "New Policy" in Device User Management and change it to "Optional User Management".




(2) Select "Device" in the Devices interface - right-click and select "Initial Reset Device [Device]". After this operation, there is no need to require a password every time you log in.

If the customer wants to enter their password when logging in, they will click on "Change Communication Policy" in the "Device" section of the scanning device interface. In the pop-up interface, they will select "New Policy" in Device User Management and change it to "Forced User Management".

Note:

XS3 factory default

User name: Administrator

Default password: xinje

## 8-11. Why cannot connect to the PLC

The reasons of cannot connect to a PLC is generally summarized as follows:

1. Confirmed as XS series products (there have been many cases where XD and XG series products are treated as XS series products).

2. Without unchecking the "Filter network scan by target and ID" menu item, confirm that the engineering equipment on the upper computer is consistent with the target device, otherwise the device may not be scanned.

3. Confirm whether the IP addresses of both parties are in the same network segment by unchecking the "Filter Network Scan by Target and ID" menu item. If the scanned device does not display a green label, it is a cross network segment device. The IP address of the device can be viewed in the right information bar; You can also confirm whether it can be pinged through the ping command; If the IP address cannot be confirmed, you can try setting dial 1 to ON and then restarting the device (initializing the IP to 192.168.6.6 when powered on), and then scanning and connecting again; If the network segments are the same but the subnet masks are different, the device cannot be scanned, but the IP address can be directly entered to connect to the device.


4. If the IP is confirmed to be correct and the device cannot be connected, it may be due to the PLC program crashing (there is a dead cycle in the program or exceeding the load capacity of the PLC). At this time, dial 2 can be set to ON (power on without loading the user program), and the connected device can be scanned again; If the connection can be scanned, download an empty program at this time, erase the abnormal program, and then restore the dialing status. At the same time, check for abnormal programs (whether there are excessively long loops or task cycle times are too small).
5. If the above steps still fail to connect the device, please contact us.

## 8-12. IP address modification unsuccessful

If the network segment is different after modifying the IP, the gateway needs to be modified at the same time. After successful modification, power on again to take effect.

## 8-13. Prompt: “No source code available for this object. Do you want to browse the original library to display the source code?”

① Pointer illegal access: null pointer, pointer pointing to illegal area (the address pointed to by the pointer conflicts with the internal address of the operating system)
② Array out of bounds
③ Dividing by 0
④ Assignment operation between signed and unsigned variables
⑤ Improper use of for, while, and repeat loop conditions

## 8-14. Repower on after setposition cleared the position, absolute encoder position changed

① Store the current position in the power-off hold area when a power outage occurs.
② Xinje servo firmware version 3792 uses MC_Home, mode 35.

## 8-15. PLC crashes

① ARM series (XS3, XSDH, XSLH): turn on dial switch 1, cut power and power on again, not load the program. Then download a new program, turn off the dial switch 1.
② X86 series (XSA): Disk D—CODESYS folder--Plclogic—delete the Application.

## 8-16. Program lost when online downloading

Check the box for online download as shown in the following figure:


## 8-17. Different computers may sometimes connect to other devices on the same LAN

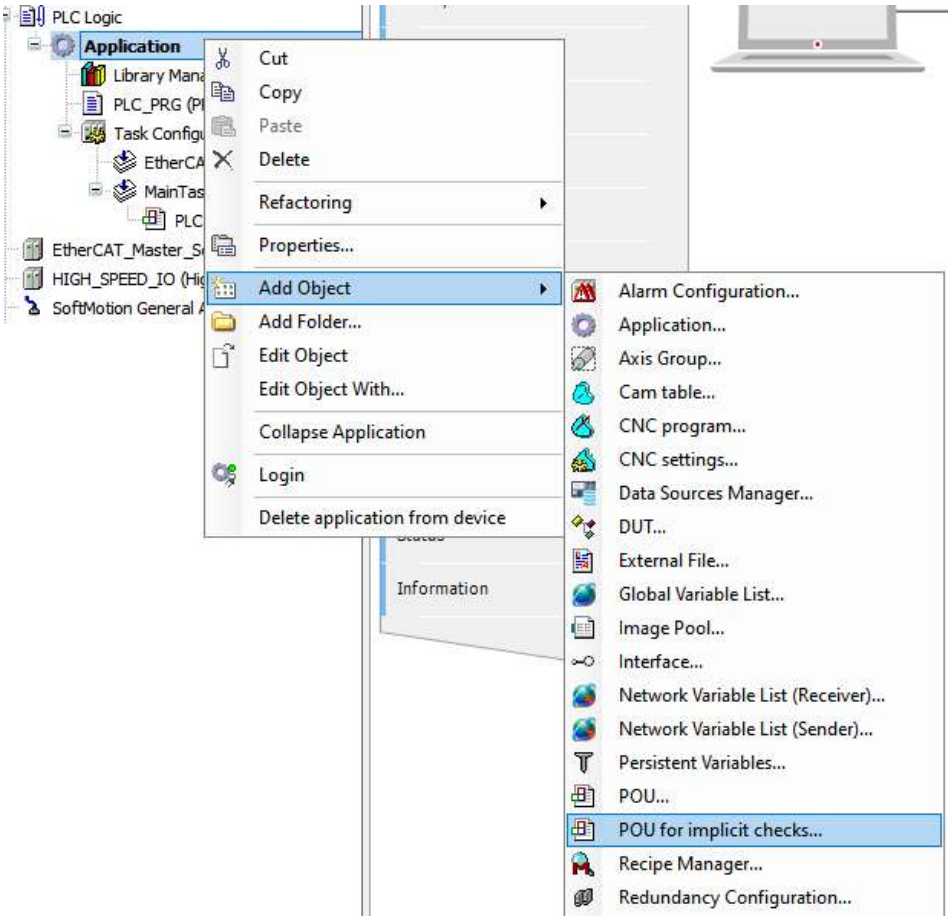
Solution: Turn off the network and reconnect to the PLC, or use flashing to determine if the scanned device is actually connected.

## 8-18. Add implicit check function

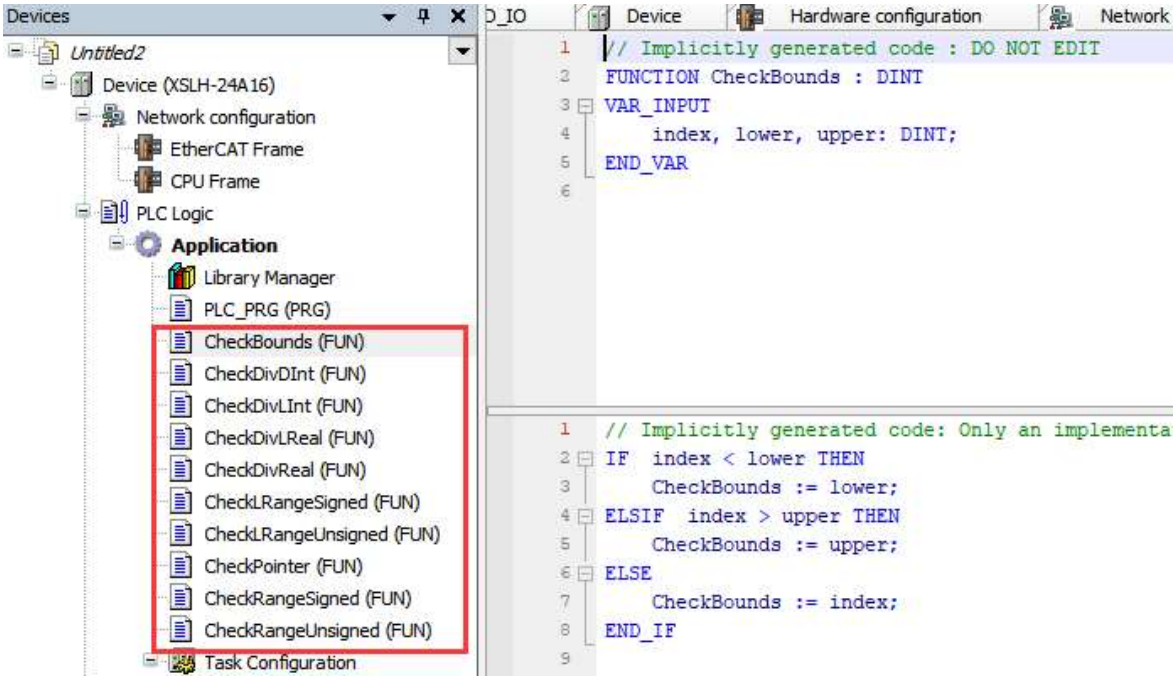
During the programming process, the following situations may occur:

- The dividend of a division operation may be zero in some cases;
- The pointer may accidentally point to an empty address during the assignment process;
- When calling an array, the array boundary overflowed.

XS Studio has a dedicated solution for the above situation, which allows for the addition of special POUs in an application. However, this POU program must exist in the application, and implicit checking function can check the array and bounds of functions, as well as divisors to zero and pointers in the running system.
Note: If the verification function of the device is provided by a special library, then this function can be disabled.
After adding a check in the POU, it will open according to the selected programming language. The default programming environment is the ST language editor. Users can right-click on the application and select "Add Object", select "POU for Implicit Checks", and then the system will pop up a dialog box, as shown in the figure:




We will introduce these commonly used functions.




(1) CheckBounds

This function checks if there is any violation of the boundaries of the array (for example, by setting or changing the index through detected error flags). A variable array type is assigned to this function, which is called a hidden function.

When calling this function, refer to the following input parameters:

Index: The index of field elements;

Lower limit: The lower limit of the field section;

Upper limit: The lower limit of the field section.

As long as the index is within the range, the return value is the index itself. Otherwise, the corresponding fields

that violate the upper or lower limit range will be returned.
For example, if "a" exceeds the upper limit in the array of the program, the program is as follows:,

```
PROGRAM PLC_PRG
VAR a: ARRAY[0..7] OF BOOL;
b: INT:=10;
END_VAR

a[b]:=TRUE;
```

At the beginning of the program, array a only had eight members ranging from 0 to 7. However, in actual programs, the b-th member of array a is true, while b is defined as 10 in the program, which actually exceeds the definition range of array a.
After using the CheckBound function, the index value will be changed from "10" to the upper limit of "7". Therefore, the value TRUE will be assigned to the array element a [7].

(2) Check+data type

To check the value of the divisor and avoid divisors being zero, the check functions CheckDivInt, CheckDivLint, CheckDivReal, and CheckDivLReal can be used. After including them in the application, each division process that occurs in the relevant code will generate a preprocessing of this function call.
For example, using the division command, the specific program is as follows:

```
PROGRAM PLC_PRG
VAR
erg:REAL;
v1:REAL:=799;
d:REAL;
END_VAR

erg:= v1 / d;
```

In the above example, erg is equal to v1 divided by d, and d is not given an initial value at the beginning of the variable definition, so its initial value is 0. If the number is directly divided by 0 in the program, the system will make an error. However, if the value of the divisor "d" becomes "1" during initialization after being checked by the CheckDivReal function pointing to division in the instruction. Therefore, the final result of division is 799, which can effectively avoid controller errors.

(3) CheckRange(Un)Signed

To check domain restrictions during runtime, the functions CheckRangeSigned or CheckRangeUnsigned can be used. The purpose of this check function is to handle subset violations appropriately, such as setting a detected error flag or changing values. When the subset type of a variable is confirmed, this feature will be hidden for access.
When accessing this function, the following input parameters are obtained:

- Value: The value assigned to the domain type
- Low: The lower limit of the domain
- High: The upper limit of the domain

If the assigned value is within a valid domain, it will be used as a return value in the function. Otherwise, values that exceed the range will either have their upper or lower limits returned.
For example, assigning i:=10*y will be implicitly replaced by

```
i:=CheckRangeSigned(10*y, -4095, 4095);
```

If the value of y is 1000, variable i will not be assigned to the 10*1000=10000 provided by the original execution, but will be replaced by 4095, as the maximum upper limit value set by the function is 4095.
For example, an example of a dead loop:

```
VAR
ui : UINT (0..10000);
END_VARFOR ui:=0 TO 10000 DO
...
END_FOR
```

The FOR loop will never leave because the check function has stopped the UI from exceeding 10000.
Note that using the CheckRangeSigned instruction and the functionality of CheckRangeUnsigned may result in an infinite loop, for example, if a subbound type is used as an increment for loop mismatch subranges.

(4) CheckPointer

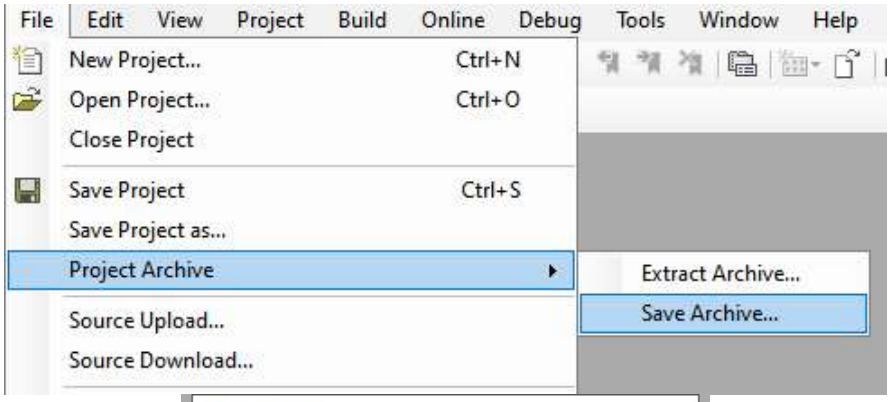
The CheckPointer function checks whether all pointer references to an address are within a valid memory range. During runtime, users may be able to use CheckPointer to check pointer access for each pointer operation.

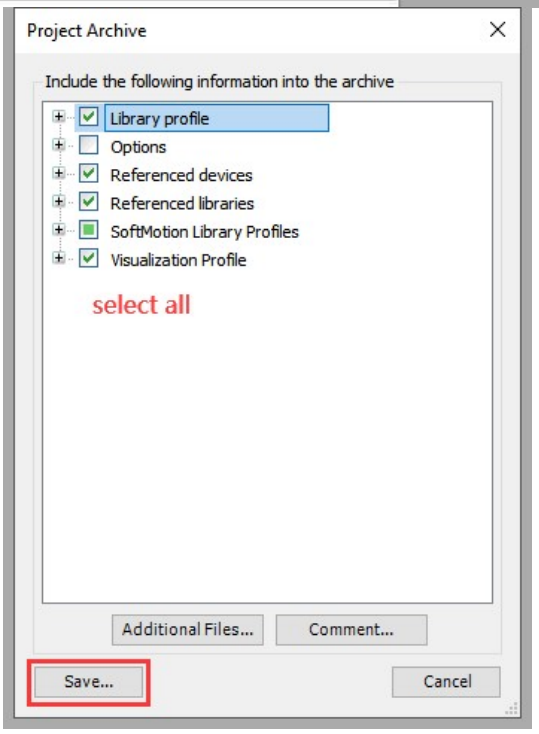
## 8-19. Points for retain function

1. For adding or deleting the retain function, it is necessary to log in and download the program, or check the update auto-start program when making online modifications.
2. After Modifying the retain area, the memory allocation is rearranged. The data will be cleared to 0.
3. ARM models (XSLH, XSDH, XS3) and X86 models (XSA) have an internal UPS that can perform retain function, while other X86 models (M210) need to determine whether they are equipped with UPS.
4. Determine if there are any other places in the program for assignment.

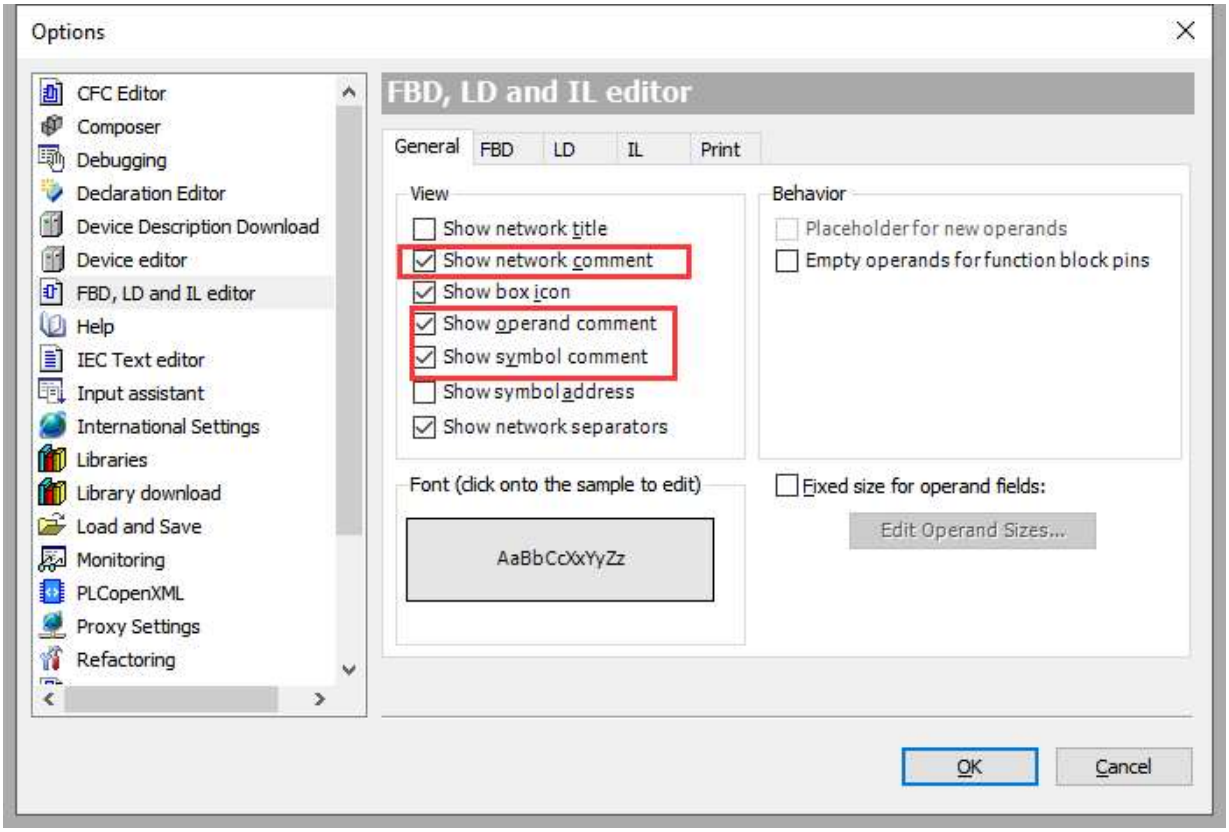
## 8-20. Report error when open the project, save project as archive

The project format project does not contain all information. When opening someone else's project or opening it in a different version, information will be lost. It should be stored in a packaged format to avoid losing information.



## 8-21. How to enable adding line and section comment



WUXI XINJE ELECTRIC CO., LTD.

Address: No. 816 Jianzhu West Road, Binhu District, Wuxi City, Jiangsu Province, China

Tel: 0510-85134136 Fax: 0510-85111290

Website: www.xinje.com Email: sales@xinje.com, fiona.xinje@vip.163.com