INVT Medium- and Large-Scale PLC Series

Software Manual

SHENZHEN INVT ELECTRIC CO., LTD.

Preface

Overview

Thank you for choosing our medium- and large-scale series PLC.

This manual contains the information necessary for using the medium- and large-scale series PLC. Please read this manual carefully before use to fully understand the functions and performance of the product, and complete system construction, which helps to give full play to the product's superior performance.

Target Audience

This manual is intended for personnel with professional knowledge of electrical engineering (e.g., qualified electricians or personnel with equivalent knowledge).

Scope of Application

This manual is applicable to the TM and TP series PLCs.

Online Support

This manual is not delivered with the product. To obtain an electronic version of the PDF file, you can:

Visit our website (www.invt.com), choose **Support > Download**, enter a keyword, and click **Search**.

Scan the QR code on the product housing > Enter a keyword and download the manual.

Revision History

Due to product version upgrades or other reasons, this manual is subject to changes from time to time without prior notice.

Number	Revision Description	Version	Release Date
1	First release.	V1.0	September 2024
2	● Deleted section 1.3.2 PC and TP6000 Communication Configuration; updated chapter 1 Introduction to the PLC. ● Added TM700 content to sections from 1.1.1 Product Overview to 1.1.3 Product Configuration and Module Description. ● Added an introduction to TM700 hardware connection to section 1.2.2 Hardware Connection. ● Updated chapter 2 Getting Started. ● Updated chapter 3 Basic Functions. ● Renamed section 4.1.4 Normal Output to 4.1.4 Normal Input and Output; updated chapter 4 Hardware Configuration.	V1.1	May 2026

Number	Revision Description	Version	Release Date
	● Renamed chapter 5 Network Configuration to chapter 5 Communication Network Configuration**;** updated the chapter accordingly. ● Deleted section 6.2.1 I/O Diagnosis; updated chapter 6 Diagnosis. ● Updated chapter 7 PLC Upgrades and Settings. ● Updated Appendix A Project Examples. ● Added support for TP2000 controllers. ● Added sections 1.3.1 PC and TM700 Communication Configuration, 2.3.7 Fast Configuration, 3.9.3 Power-Failure Retention Mode: INVT, 3.13 Toolbox, 5.3.1.4 Function Codes, 5.3.2.2 Function Codes, 5.5 EtherNet/IP, 6.7 Mismatch between EtherCAT Upper Computer Configuration and Physical Connection, 6.9 Version Compatibility between the Upper Computer and Lower Computer, 7.2 Device Connection, 7.8 Log–7.12 Developer Mode, and A.4 Example of Atomic Operation. ● Updated sections 4.1.5 High-speed Pulse Output, 5.1.2 Modbus TCP Master Communication Configuration, 5.2.2 Modbus RTU Master Broadcast Configuration, 5.2.3 Modbus RTU Master Communication Configuration, 5.3.2.3 Process Data Object (PDO), 5.3.4 Dual EtherCAT Masters, 7.3 Firmware Upgrade–7.7 File Transfer.		

Contents

1 Introduction to the PLC

1.1 Overview

1.1.1 Product Overview

INVT's medium- and large-scale PLCs currently include two series: TM700 and TP2000, which follow the IEC 61131-3 programming language system and support six standard programming languages: IL, LD, FBD, ST, SFC, and CFC. Through the EtherCAT bus, they can realize high-level motion control functions such as electronic cam, electronic gear, synchronous control, and positioning. Owing to rich communication interfaces and versatile I/O modules, they can provide users with flexible and intelligent automation solutions to meet their diverse application needs.

The TM700 series is a high-performance PLC with a modular structure design. It is mainly used in scenarios with high motion control requirements and complex control networks. It has greatly improved control performance, communication capabilities, programming efficiency, etc., allowing you to build a control network more flexibly and realize data interaction with the information layer through OPC UA more conveniently, further improving device productivity, shortening development cycles, and bringing a more excellent experience.

The TP2000 series intelligent high-performance controller is a 128-axis, high-performance production-line controller based on the x86 hardware platform and fully compliant with the PLCopen specification. It features flexible networking and powerful performance, enabling complex multi-axis motion control with ease. A multifunction display further improves usability, while the compact, low-power, fanless design saves cabinet space. The built-in UPS provides 5MB large-capacity data retention during power failure, and the built-in high-speed I/O supports functions such as probe, high-speed compare output, and pulse control. It is widely used in industries such as printing, packaging, food, lithium battery, 3C electronics, photovoltaics, woodworking, small and medium-sized production lines, and process control.

1.1.2 Product Specifications

Model	TM750	TM751	TM752	TM753
Rated working voltage	DC24V (-15%–+20%)			
Memory				
Program capacity	20MB			
Data capacity	64MB			
Data capacity in power failure retention mode	1MB			
Max. capacity of expanded SD card	32G			
I/O				
Number of local IO modules	16			
High-speed input	4 groups of high-speed counters, supporting single phase, phase A/B, CW/CCW, and pulse+direction, of which phase A/B supports 1x, 2x, and 4x frequencies.			

Model	TM750	TM751	TM752	TM753
High-speed output	8-channel 200kHz sink-type high-speed output, supporting 4-axis pulse motion control			
Support for I/O interruption	8-channel high-speed interruption			
PWM output	4-channel PWM output			
Communication Network and Interface				
Ethernet	×2, RJ45, 100Base-TX Support for PLC software download, Modbus TCP, TCP// IP, OPC UA protocol, and EtherNet IP			
EtherCAT	×1, RJ45, 100Base-TX, distance between two slaves ＜ 100 m			
Serial communication (RS485)	×2, Modbus RTU master/slave, plug-in terminal			
USB	×1, Type-C, PC communication, program download and debugging			
Memory card	×1, Micro SD, for upgrading applications			
Communication expansion	CANopen, 4G, Wi-Fi			
Motion Control				
Maximum number of controlled axes	4	8	16	32
Motion control functions	Point-to-point motion, interpolation motion (straight line, circular arc, etc.), electronic gear, electronic cam (flying shear, chasing shear, etc.), axis group, CNC, etc.			
Configuration Programming				
Programming language	IL, ST, FBD, LD, CFC, and SFC			

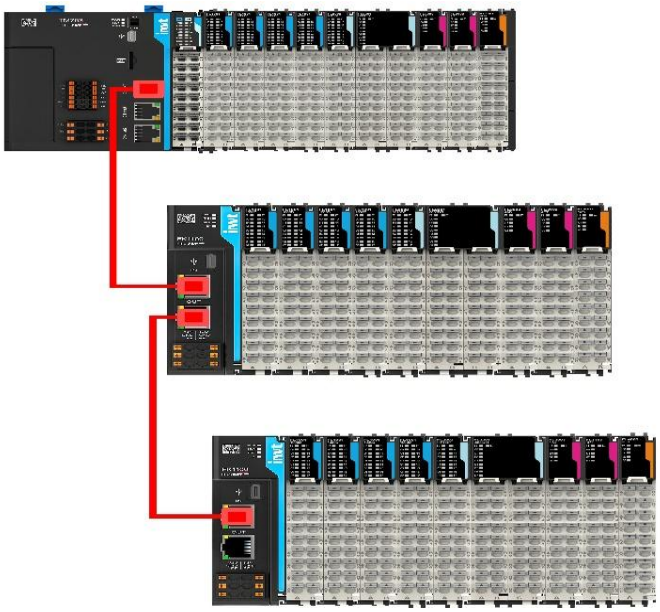
Model	TP2321-1016	TP2321-1032	TP2422-1048	TP2422-1064	TP2422-1096	TP2422-1128
Memory	8GB					
Storage	64GB		128GB			
Number of EtherCAT masters	×1		×2			
Number of EtherCAT axes	16	32	48	64	96	128
Ethernet	2×RJ45, 10/100/1000Base-TX adaptive Configurable as a soft router					
RS485	2×RS485, push-in terminal					
Local I/O count	1 channel of ABZ differential encoder input, supporting up to 2MHz 8 channels of HSDI single-end input, supporting up to 200kHz 8 channels of HSDO single-ended output, supporting up to 200kHz 1 channel of machine start/stop control input 1 channel of app start/stop control input					
Remote I/O count	Up to 128000 points (EtherCAT bus)					
USB	2×USB3.0, Type-A					
Display	1×DisplayPort 1x1.8 inch TFT color screen					
Key	6x touch key					
Operating system	Linux					

Model	TP2321 -1016	TP2321 -1032	TP2422 -1048	TP2422 -1064	TP2422 -1096	TP2422 -1128
Programming language	IEC 61131-3 programming languages (IL, ST, FBD, SFC, CFC, and LD)					
Program execution mode	Compiled execution					
Programming platform	Invtmatic Studio 3.0.0.6 or later					
User program storage space	256M Byte					
User data storage space	256M Byte					
Power-failure retention space	5M Byte					
Real-time clock	Supported (CR2032 button battery is user provided)					
Power supply	24V DC (-15%–+20%)/1.5A Protection against reverse connection and surges					
Entire machine power dissipation	<36W					
Cooling method	Natural cooling					
IP rating	IP20					
Operating temperature	-10°C–+55°C					
Operating humidity	10%–95% (no condensation)					
Storage temperature	-25°C–+70°C					
Storage humidity	5%–100% (no condensation)					
Atmosphere	Free of corrosive gases					
Altitude	2000m					
Pollution degree	Degree 2 or lower, compliant with IEC 61131-2					
ESD immunity level	4kV CD or 6kV AD					
Vibration resistance	5–8.5Hz, vibration amplitude of 3.5mm; 8.5–150Hz, acceleration of 10m/s^2; X/Y/Z axis, 10 cycles					
Product weight	Net weight: approx. 1.41kg Gross weight: approx. 1.59kg					

1.1.3 Product Configuration and Module Description

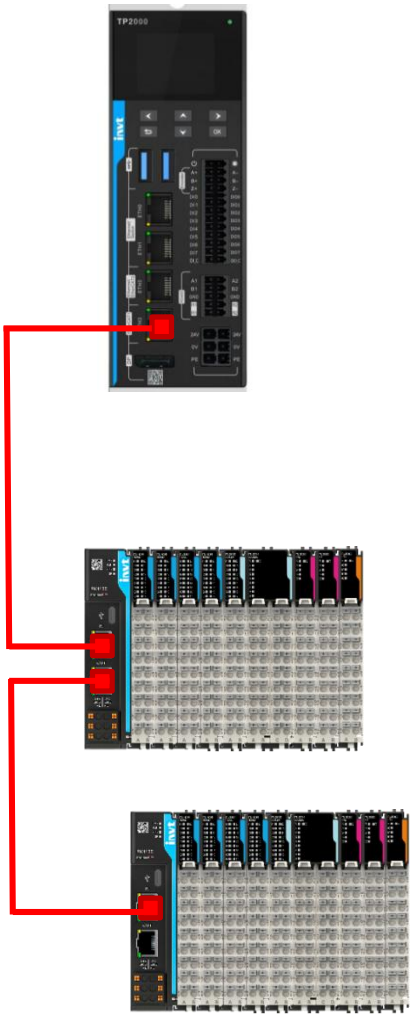
The TM700 series's CPU can be directly connected to the Flex series I/O modules, or expanded to the Flex series I/O modules through a communication coupler, as shown in Figure 1-1.

Figure 1-1 System Integration Diagram 1



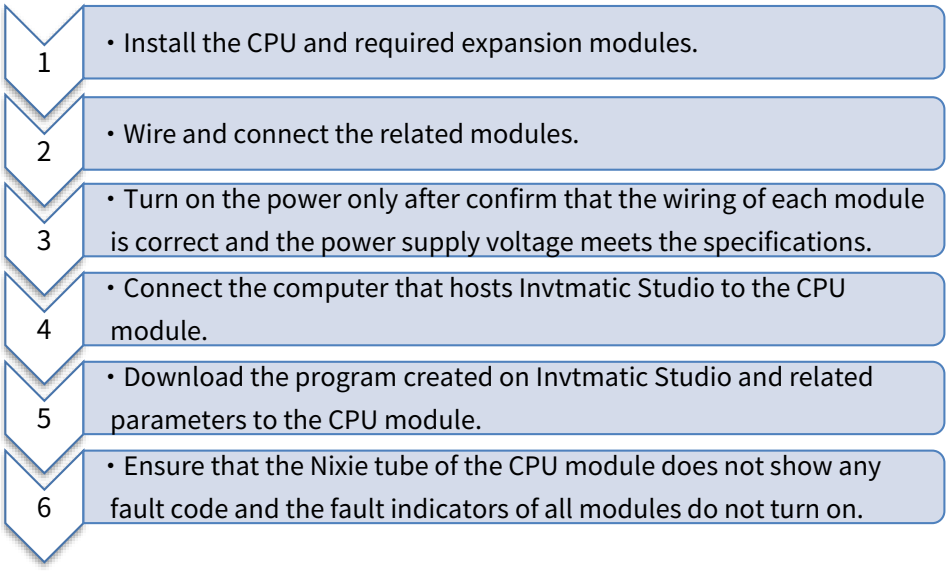
The TP2000 series's CPU cannot be directly connected to the Flex series I/O modules. It can be expanded to the Flex series I/O modules through a communication coupler, as shown in Figure 1-2.

Figure 1-2 System Integration Diagram 2



1.1.4 System Application Process

Figure 1-3 System Application Process



1.2 Invtmatic Studio Overview

Invtmatic Studio is a programming platform developed by Shenzhen INVT Electric Co., Ltd. It supports the IEC 61131-3 programming language system and six standard programming languages: IL, LAD, FBD, SFC, ST, and CFC.

1.2.1 Software Installation and Uninstallation

1.2.1.1 Software Obtaining

INVT's PLC user programming software contains the Invtmatic Studio platform, installation files, and related reference materials. You can obtain them in the following way:

Visit our website www.invt.com and go to Support > Download > Software to download the software installation package for free.

1.2.1.2 Software Installation Requirements

You can install the software on a desktop PC or a laptop:

- Operating system: Windows 7/Windows 8/Windows 10
- CPU clock speed: 2 GHz or higher
- Memory: 2 GB or higher
- Available hardware space: 5 GB or higher
- Installation requirements:
  - ✧ Use an administrator account
  - ✧ Turn off the antivirus software
  - ✧ Disable the encryption system

✧ Do not use Chinese paths

1.2.1.3 Preparation

- If it is the first time to install Invtmatic Studio, check whether your computer meets the software installation requirements. If yes, you can install it directly.
- If you want to install the latest version of Invtmatic Studio, check the version information about the installed software by choosing **Help** > **About**. If it is not the latest version, you can upgrade the software using the online upgrade method.

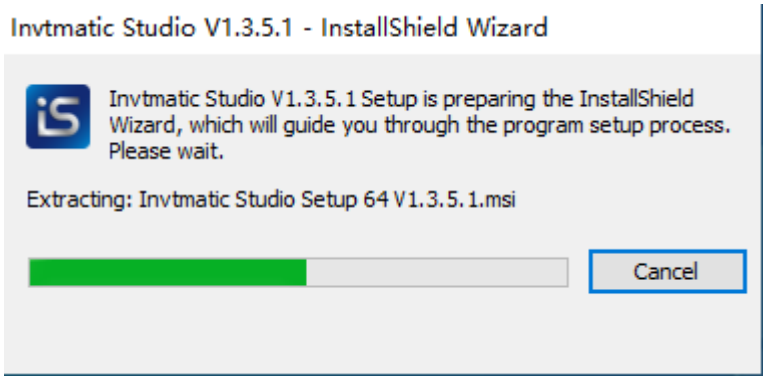
Figure 1-4 Version Information Display Interface

1.2.1.4 Installing the Software

Step 1 Locate the installation file storage path, and double-click Invtmatic Studio Setup 64 Vx.x.x.exe (The installation procedure is the same for all versions of Invtmatic Studio. Version V1.3.5.1 is used here as an example).

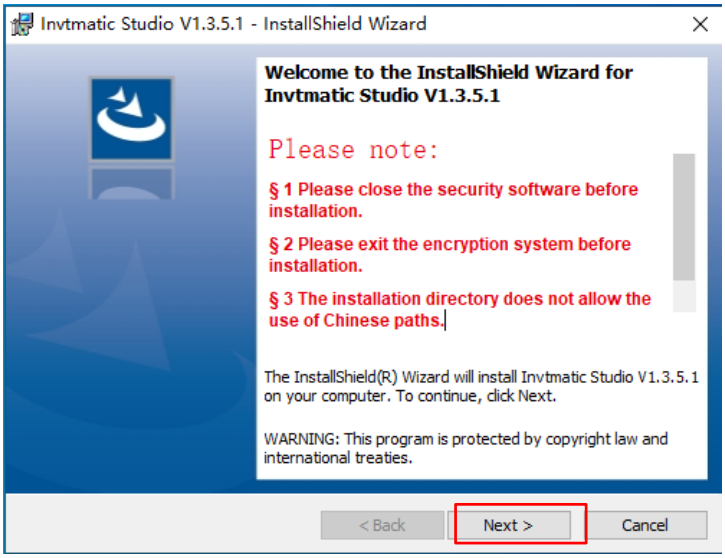
Step 2 The installation starts. See the following figure.

Figure 1-5 Installation Preparation interface



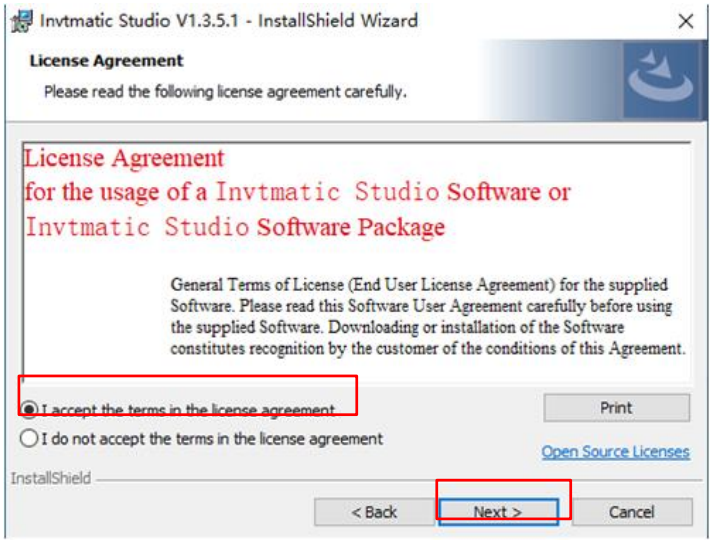
Step 3 When the dialog box shown in the following figure appears, click **Next**.

Figure 1-6 Installation Wizard Interface



Step 4 Then the License Agreement dialog box appears. Check **I accept the terms of the license agreement**, and then click **Next**.

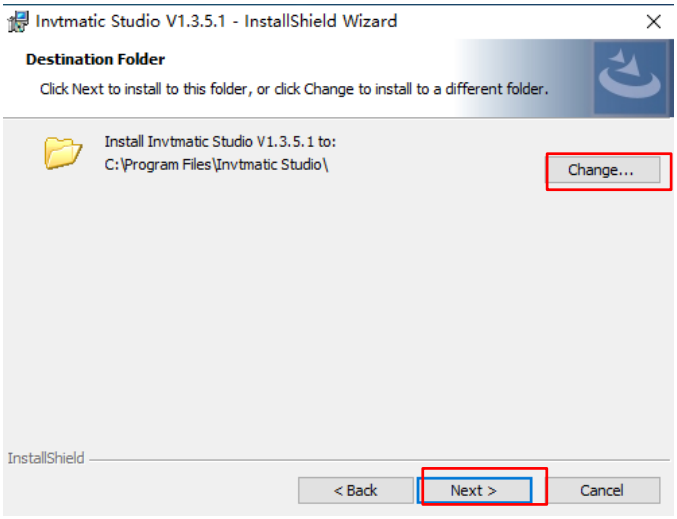
Figure 1-7 License Agreement Interface



Step 5 Set the software installation path, and click **Next**.

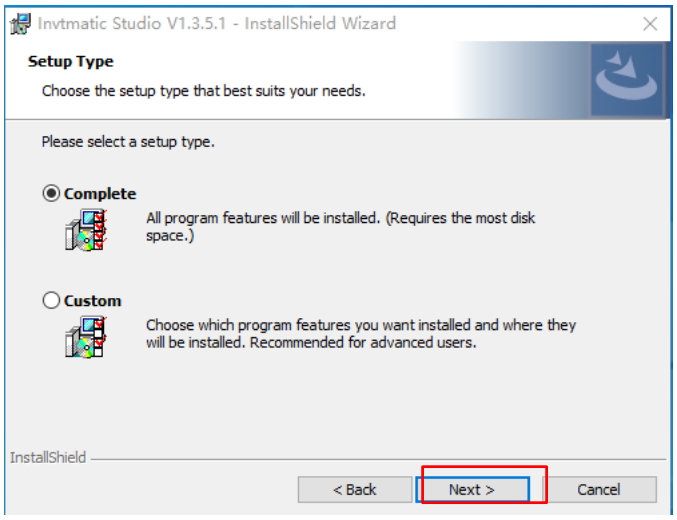
Note: The software installation path cannot be in Chinese.

Figure 1-8 Installation Path Confirmation



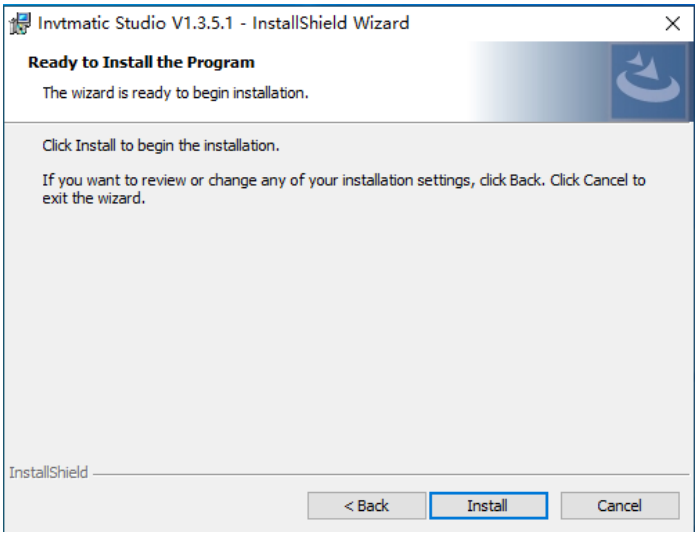
Step 6 The installation component selection interface appears. You can select the Complete or Custom installation mode. If you have no special requirement, keep the default selection, and click **Next**.

Figure 1-9 Installation Mode Selection Interface



Step 7 When the following interface appears, click **Install**.

Figure 1-10 Installation Preparation Interface



Step 8 An installation progress bar appears. Click Finish when the installation is completed.

Figure 1-11 Installation Progress Interface

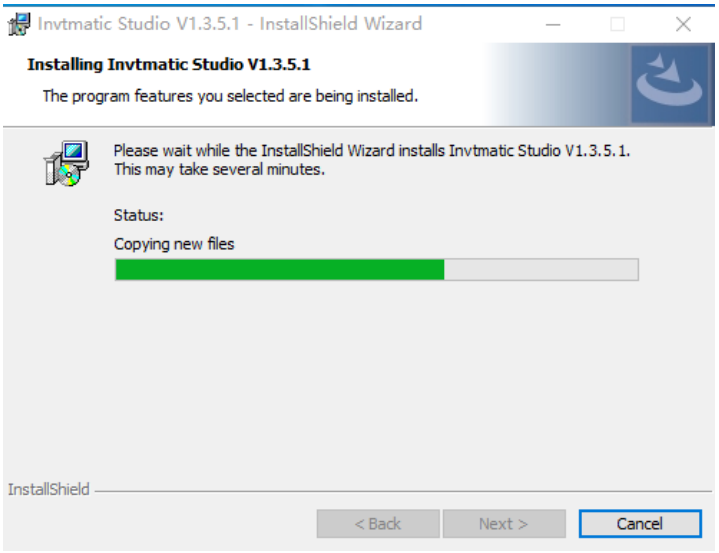
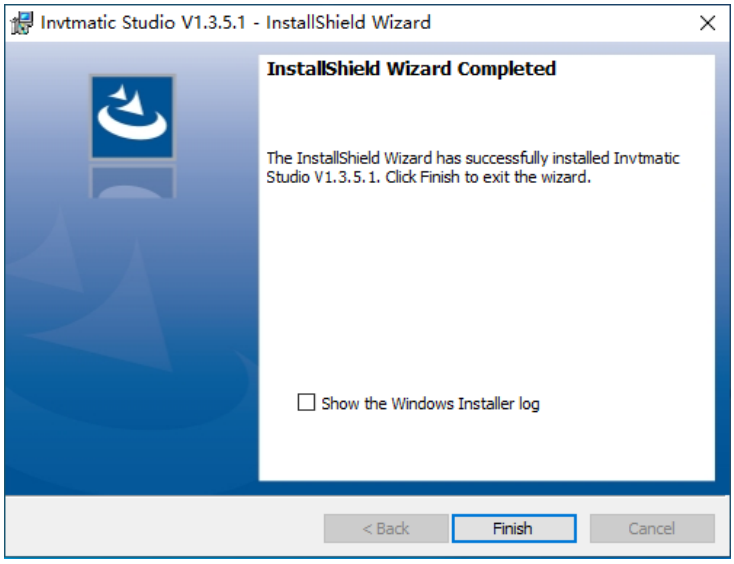


Figure 1-12 Installation Complete Interface



1.2.1.5 Uninstalling the Software

Uninstall Invtmatic Studio by using the standard software uninstallation method of a Windows system. The procedure is as follows:

Step 1 Shut down Invtmatic Studio running programs, including the backend running program.

Step 2 Enter the control panel, find and right-click Invtmatic Studio, and click **Uninstall**.

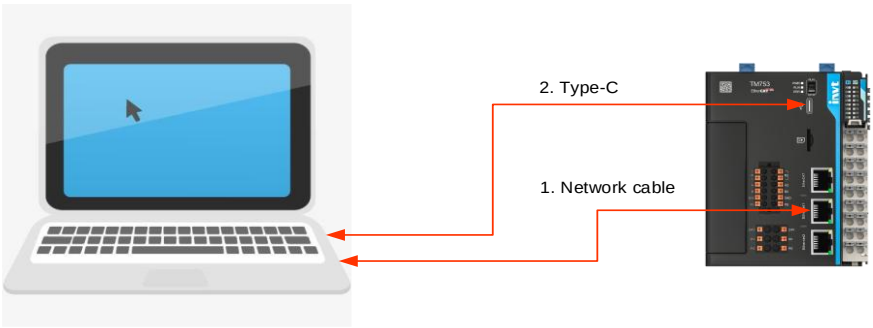
Step 3 Wait until the software is uninstalled.

1.2.2 Hardware Connection

The hardware connection between an upper computer and the TM700 series PLC can be realized by two methods:

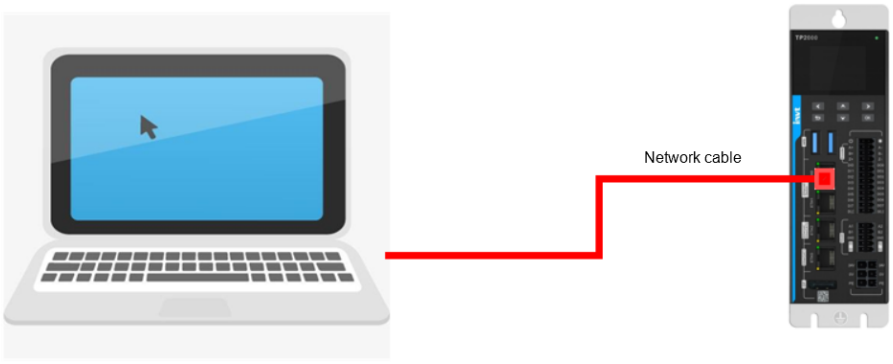
1. Using a LAN network cable.
2. Using a Type-C cable.

Figure 1-13 Hardware Connection between an Upper Computer and the TM700 Series PLC



The hardware connection between an upper computer and the TP2000 series PLC can only be realized through a LAN network cable.

Figure 1-14 Hardware Connection between an Upper Computer and the TP2000 Series PLC



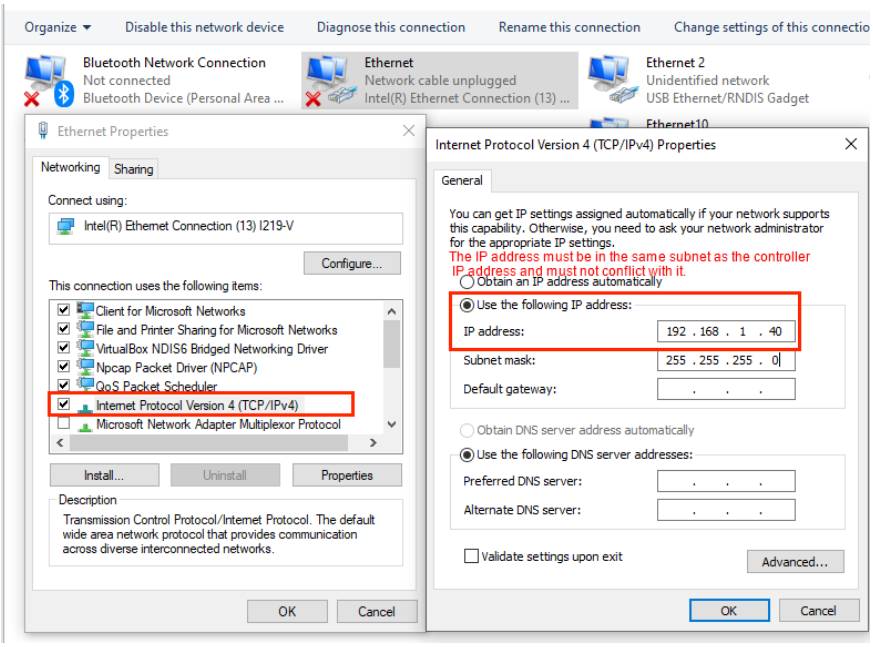
1.3 PC Communication Configuration

1.3.1 PC and TM700 Communication Configuration

- **Using a LAN network cable**

  If the hardware is connected with a LAN network cable, ensure that the IP address of the PC and the IP address of the controller are in the same network segment. The factory default IP address of the TM700 series is as follows: Ethernet1: 192.168.1.10; Ethernet2: 192.168.2.10, both having program debugging and downloading functions. Connect the network cable to Ethernet1/Ethernet2. The IP address of the PC should be set to 192.168.1.xxx/192.168.2.xxx (xxx means any integer value in the range of 1 – 254 except 10).

  Figure 1-15 PC and TM700 Communication Configuration Using a LAN Network Cable



■ **Using a Type-C cable**

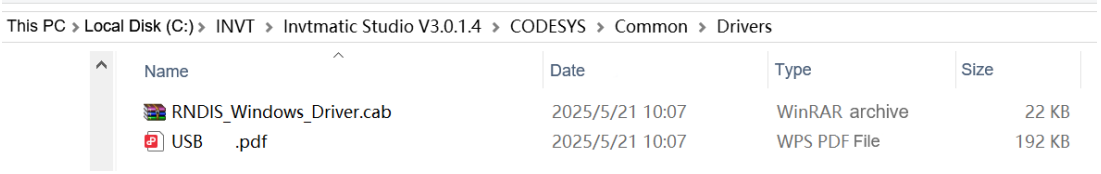
- Install the USB driver

When the PC runs Windows 10, install the USB driver as follows:

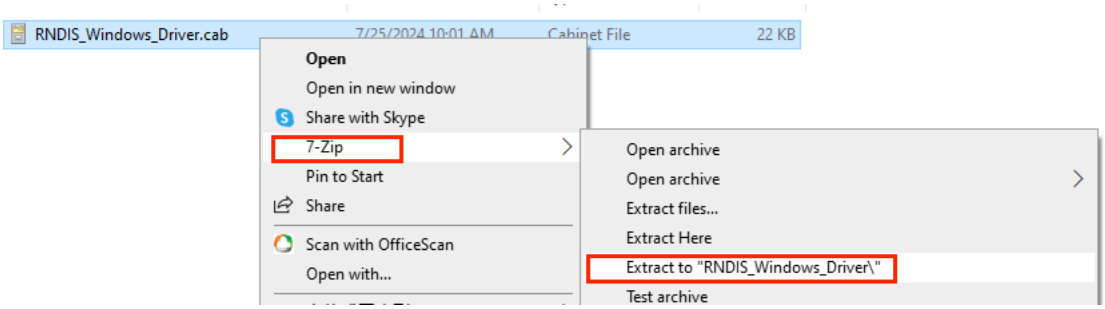
Step 1 Right-click the upper computer software.

Step 2 Select **Open file location**.

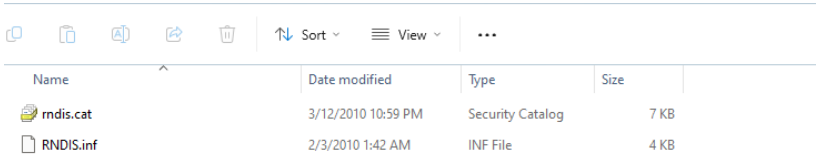
Step 3 Locate **Drivers** > **RNDIS_Windows_Driver.cab**.



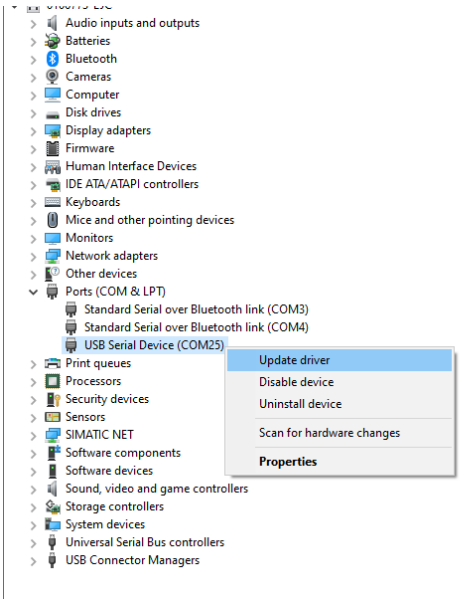
Step 4 Right-click the file "RNDIS_Windows_Driver.cab" and unzip it.



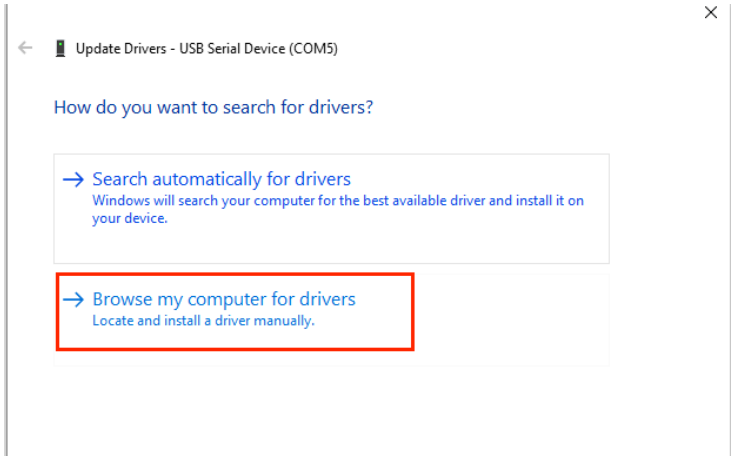
Step 5 When the following interface appears, press any key.

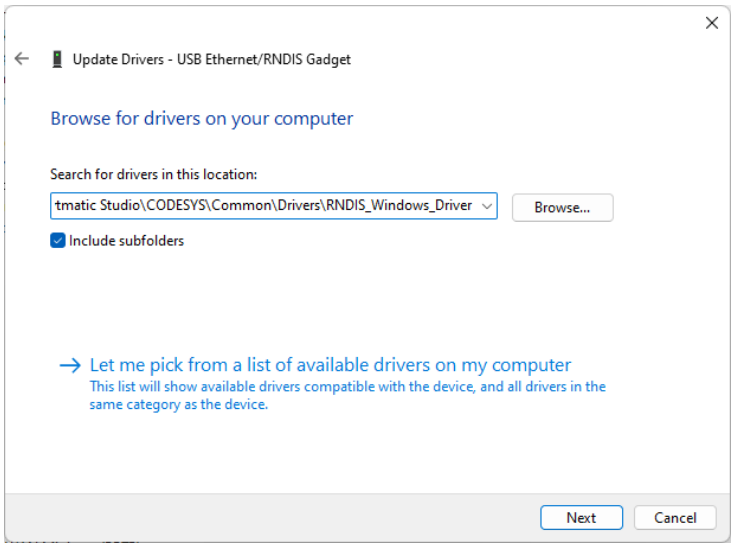


Step 6 Connect the computer and the PLC with a USB cable: Open Device Manager, select Ports > USB serial device, and right-click Update driver.

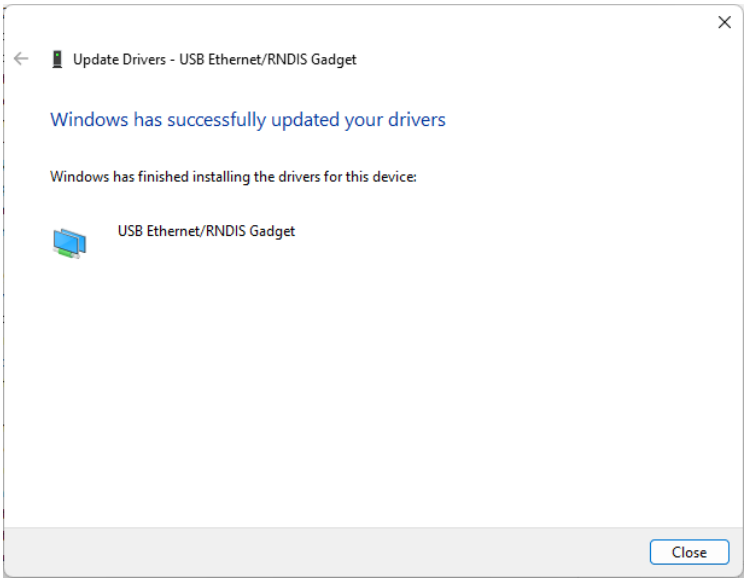


Step 7 Click Browse my computer for drivers and select the driver folder.



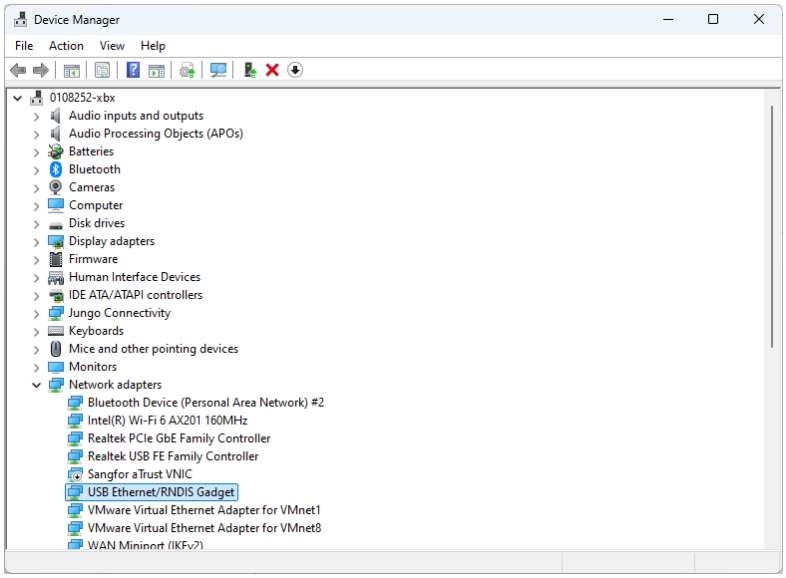


Step 8 Wait for the installation process to complete, and then click Close.

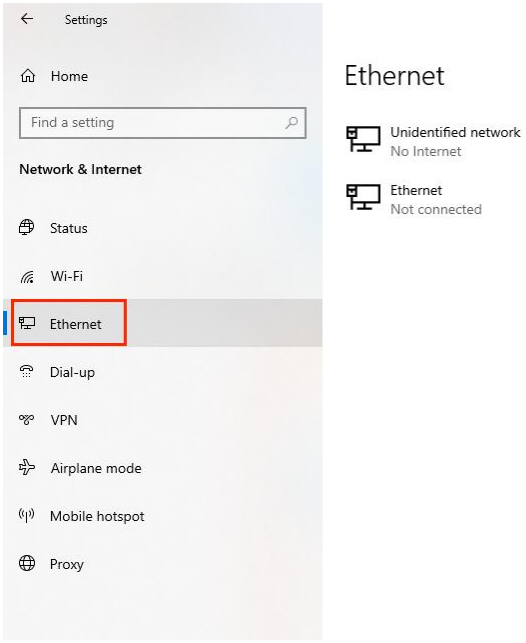


■ **Configure the USB network port**

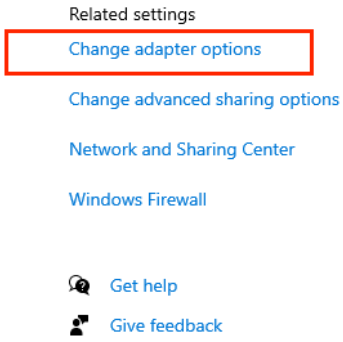
The USB RNDIS item has been added to the **Network adapters** under **Device Manager**.



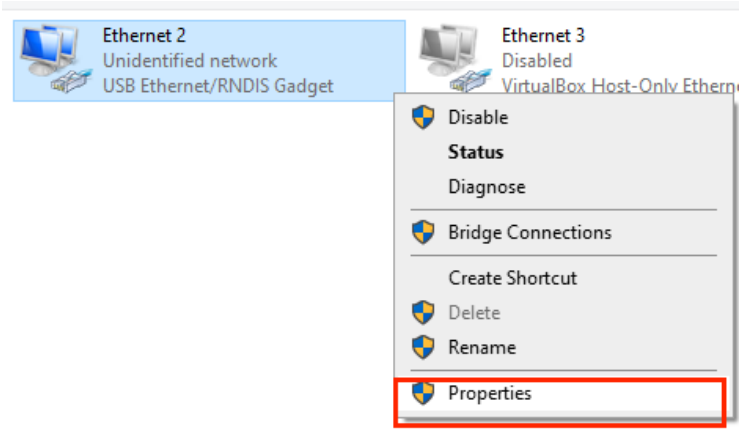
Step 1 Right-click the **Ethernet**.



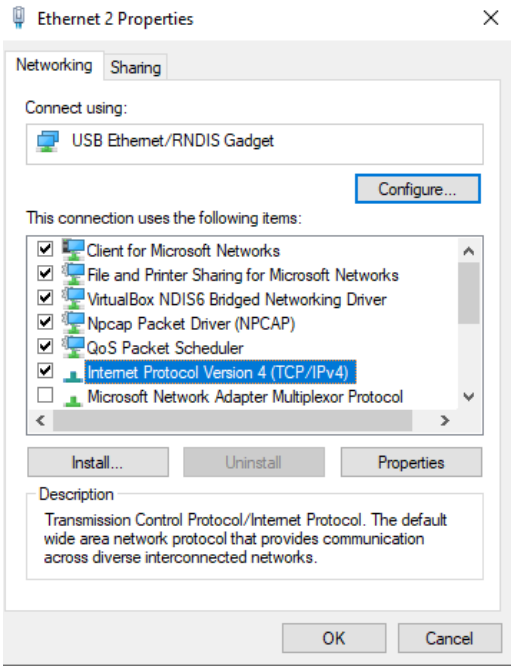
Step 2 Click **Change adapter options**.



Step 3 Right-click the **Unidentified network** (with USB RNDIS in its name), and select **Properties**.

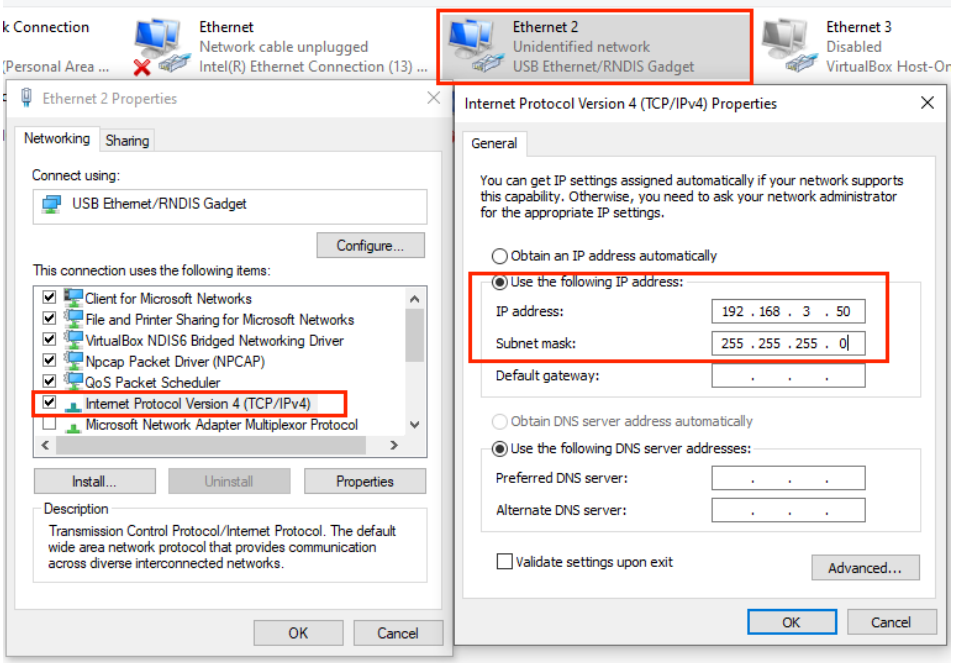


Step 4 Check **Internet Protocol Version 4** and click **Properties** to set the IP address.



Step 5 Set the IP address manually.

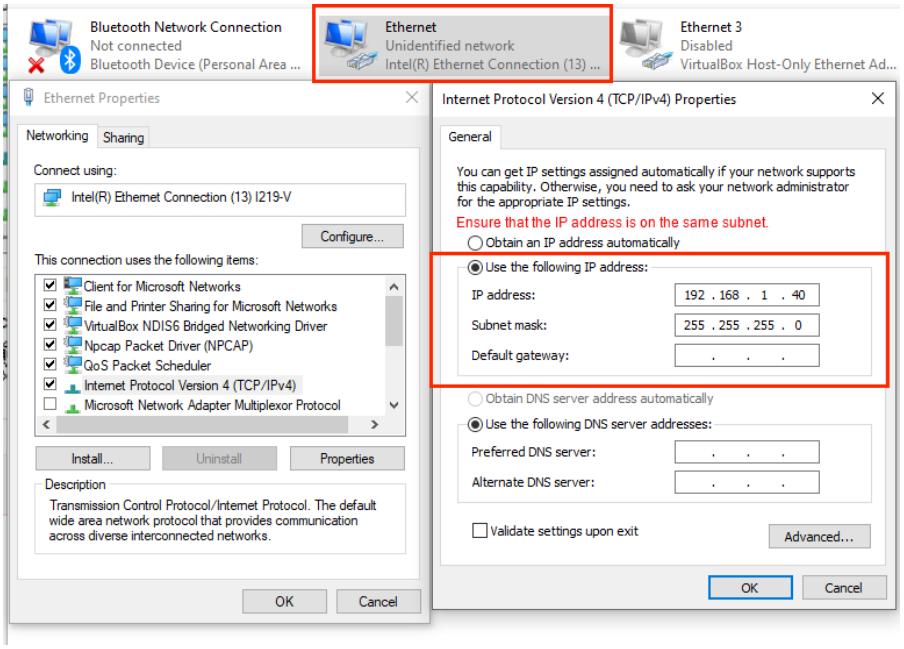
Note: The IP address must be in the network segment 192.168.3.xxx (xxx means any integer value in the range of 1 – 254 except 10).



1.3.2 PC and TP2000 Communication Configuration

If the hardware is connected with a LAN network cable, ensure that the IP address of the PC and the IP address of the controller are in the same network segment. The factory default IP address of the TP2000 series is as follows: Ethernet1: 192.168.1.10; Ethernet2: 192.168.2.10, both having program debugging and downloading functions. Connect the network cable to Ethernet1/Ethernet2. The IP address of the PC should be set to 192.168.1.xxx/192.168.2.xxx (xxx means any integer value in the range of 1 – 254 except 10).

Figure 1-16 PC and TP2000 Communication Configuration Using a LAN Network Cable



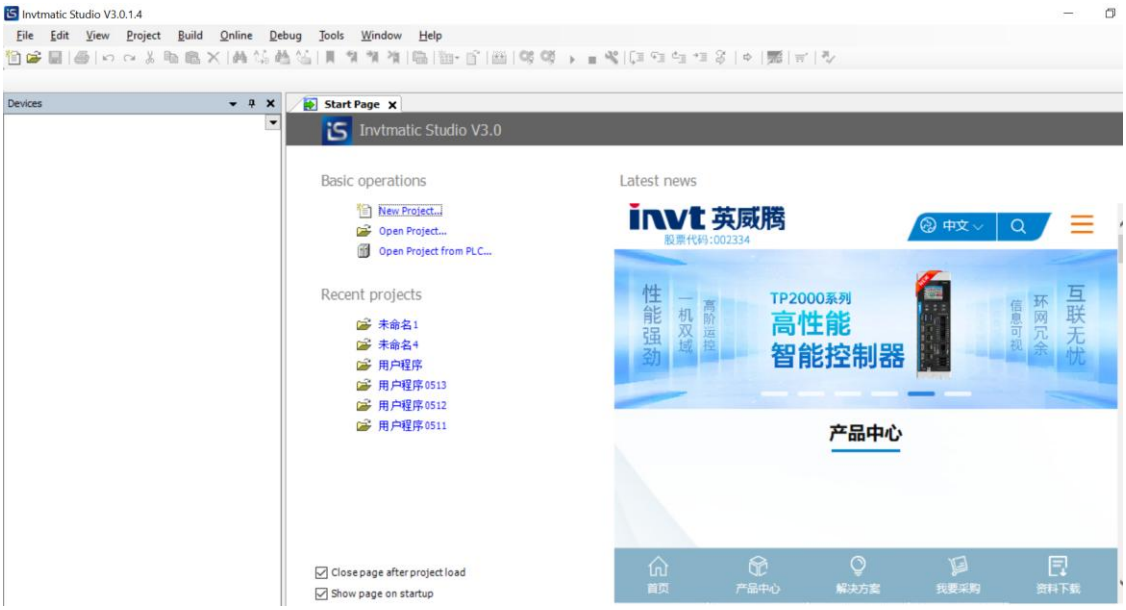
2 Getting Started

2.1 Project Creation

2.1.1 Starting the Programming Environment

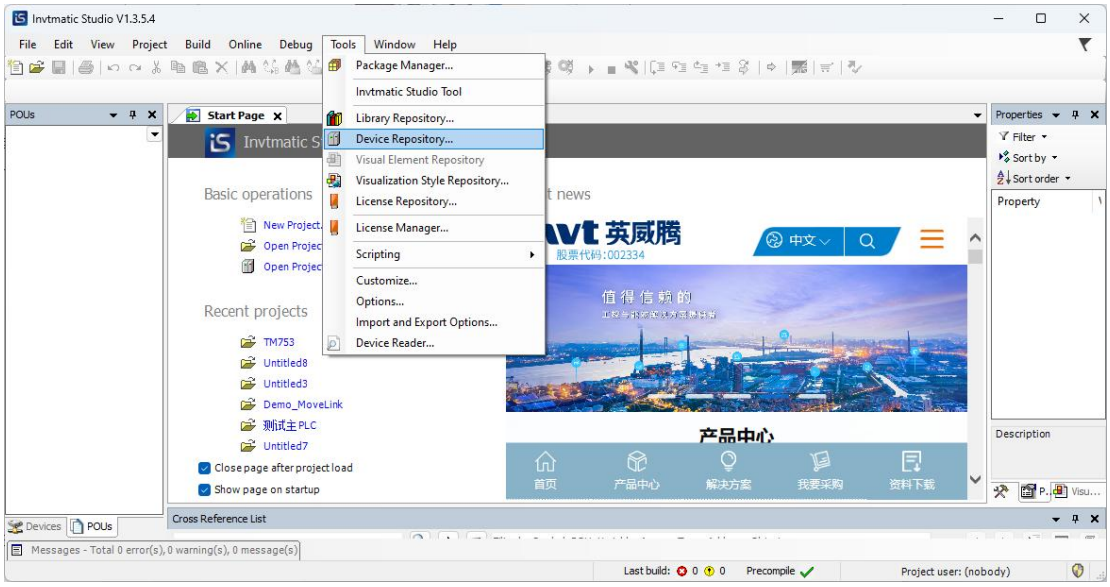
Taking Invtmatic Studio V3.0.1.4 as an example, start the programming environment in the following steps:

Step 1 Double-click the software icon of Invtmatic Studio . The programming environment is as follows:

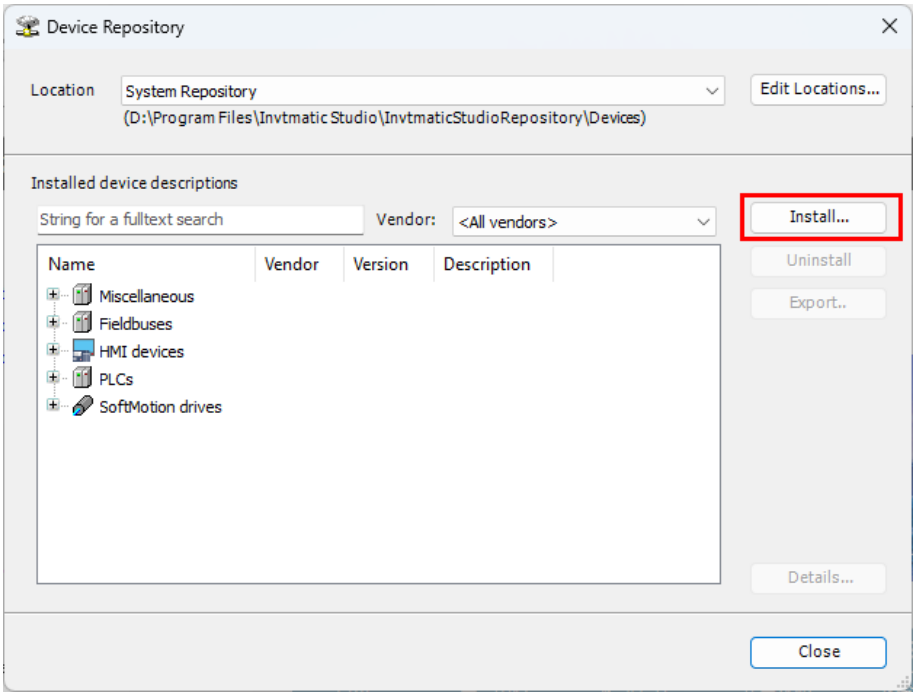


Step 2 In the tool bar, select **Tools > Device Repository** to add a device profile.

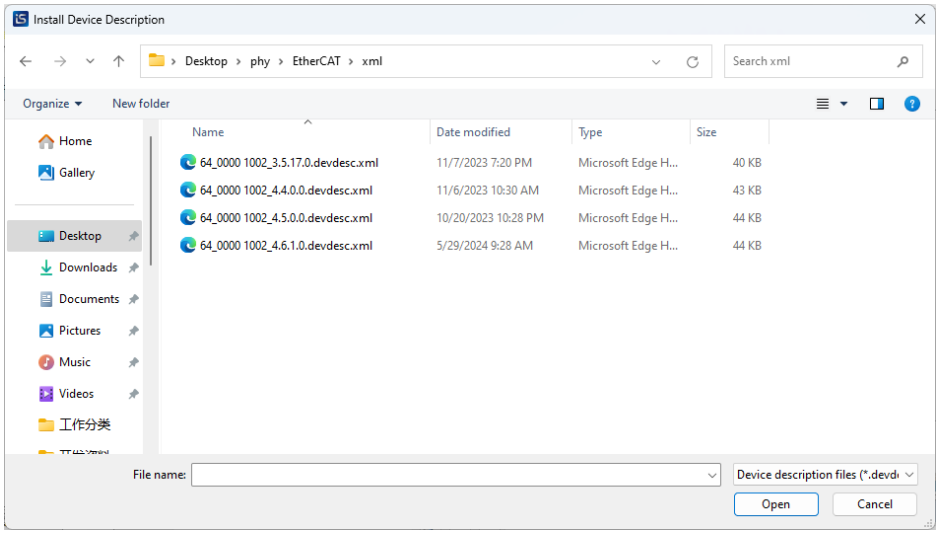
Note: Invtmatic Studio V3.0.1.4 is installed with the INVT device profile by default. For third-party devices, it needs to be added manually.



Step 3 In the **Device Repository** pop-up window, click **Install**.



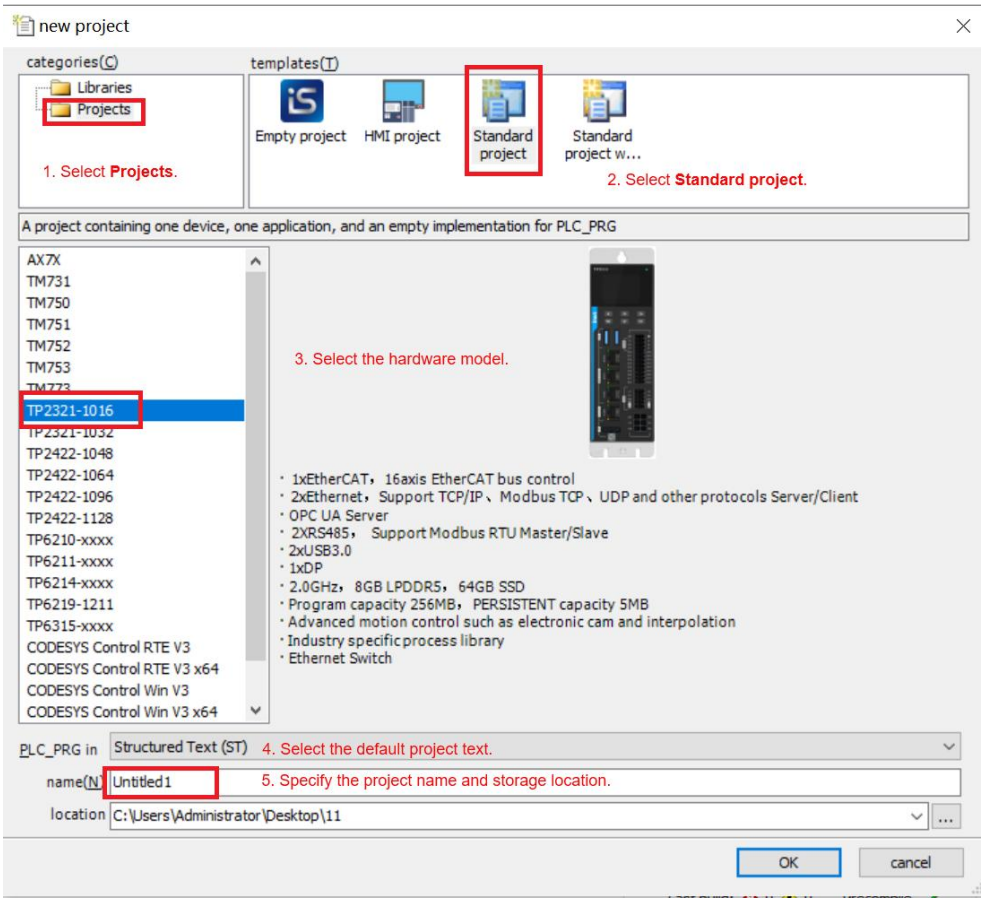
Step 4 From the pop-up window, select the device profile to be installed from a local folder and then click **Open**.



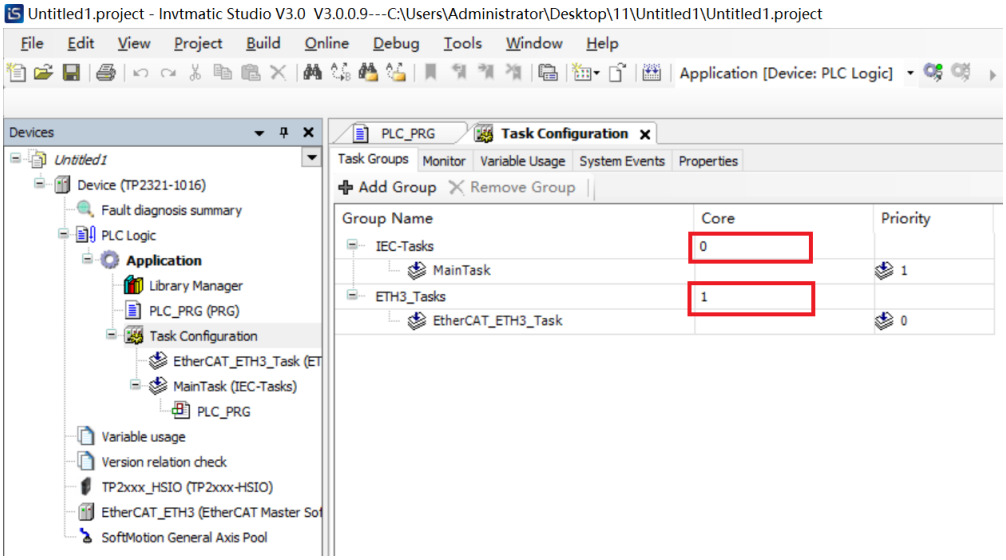
Note: Here you can follow the above steps to add all device profiles provided by INVT, or add third-party device profiles.

2.1.2 Creating a New Project

Click the project creation icon  at the upper left corner or select **File** > **New Project**, or directly click **New Project** in the window to quickly create a project. Select the project category, project template, device type, programming language, project name, and project storage directory, as shown in the following figure.



After completing the above operations, go to the configuration and programming interface of Invtmatic Studio, configure the task, and double-click **PLC_PRG(PRG)** to write programs.



Note: The TP2321 is a dual-core model where EtherCAT is bound to Core 1, while all other tasks run on Core 0. The TP2422 is a quad-core model where EtherCAT is bound to Cores 2 and 3, with other tasks running on Cores 0 and 1.

2.2 Typical Steps of Project Writing

From the above example given in section 2.1.1 Starting the Programming Environment, writing a user program with MC motion control functions generally requires the following steps:

Step 1 Application system hardware configuration

Configure the network according to the main controller, expansion modules, network type, servo slave, and other hardware used.

Step 2 User program writing

According to the control function to be implemented, write motion control with one POU (such as POU1), and write common logic control with another POU (such as POU2).

Step 3 Servo drive parameter configuration

Configure the objects of SDO and PDO according to the servo name and running mode in the hardware configuration, and ensure that the communication objects required between the MC function block of the user program and the servo are filled in the configuration table.

Step 4 Servo motor parameter configuration

Correctly fill in the resolution of the servo motor encoder, the transmission ratio of the mechanical structure, the characteristics of the axis movement range, and other parameters so that the displacement instruction imposed on the control object corresponds accurately to the actual displacement.

Step 5 Core allocation

TP2321 (Dual-core model): EtherCAT is bound to Core 1; all other tasks run on Core 0.

TP2422 (Quad-core model): EtherCAT is bound to Cores 2 and 3; all other tasks run on Cores 0 and 1.

Step 6 Task arrangement

Based on the real-time requirements of control, execute the motion control function POU1 in the EtherCAT task; set the cycle to 250 μs, 500 μs, 1 ms, 2 ms, or 4 ms and make it the same as the EtherCAT master synchronization cycle, with the priority of 0; execute the common logic control POU2 in MainTask or other common tasks and set the cycle to 1 ms, 4 ms, or 20 ms, with the priority of 1.

Step 7 Online debugging

Connect the TP2000 Series PLC to the PC via a LAN network cable correctly and then power on the PLC. Download and debug the user program to eliminate user program bugs (if possible, you can connect the servo drive system to the TP series PLC, and then perform debugging. If the servo system is not available, you can set the servo drive system as a virtual axis; if the TP series PLC is not available, you can simulate and debug the user program on the PC to eliminate possible errors in the user program until it is compiled without errors).

2.3 Examples of Program Writing and Debugging

Here is an example of a basic servo control program to give you a first glimpse of the programming process before you go through the principle of the programming system and the method of compiling the motion control program. Write a simple program that allows the TP series PLC to implement the following functions:

the servo motor repeats rotating forward 50 revolutions, and then reversing 50 revolutions.

The programming method and steps are as follows:

Step 1 Add the corresponding devices: EtherCAT master, servo drive, and motor shaft.

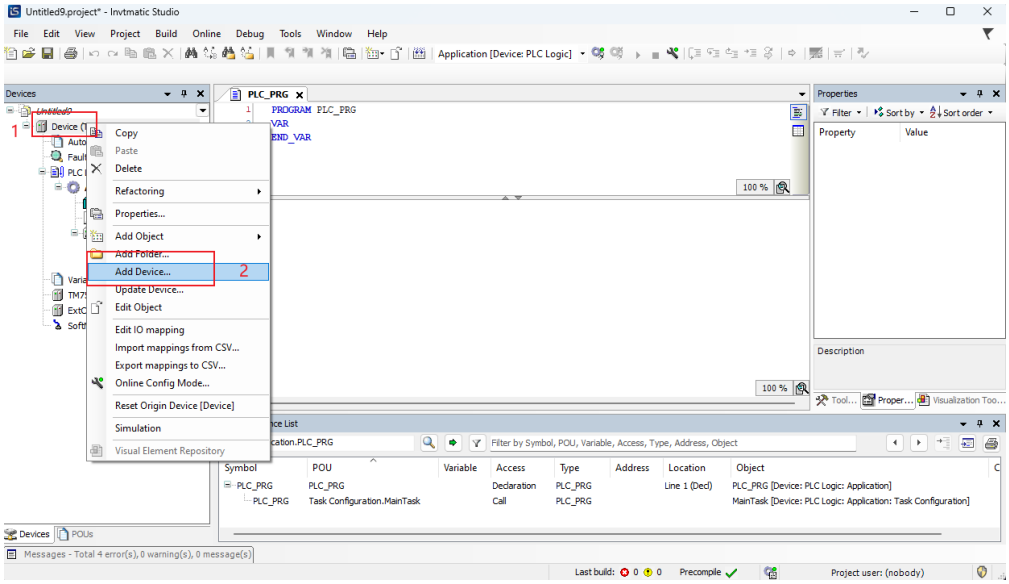
Step 2 Handle the motion control of the servo in the high real-time EtherCAT task cycle.

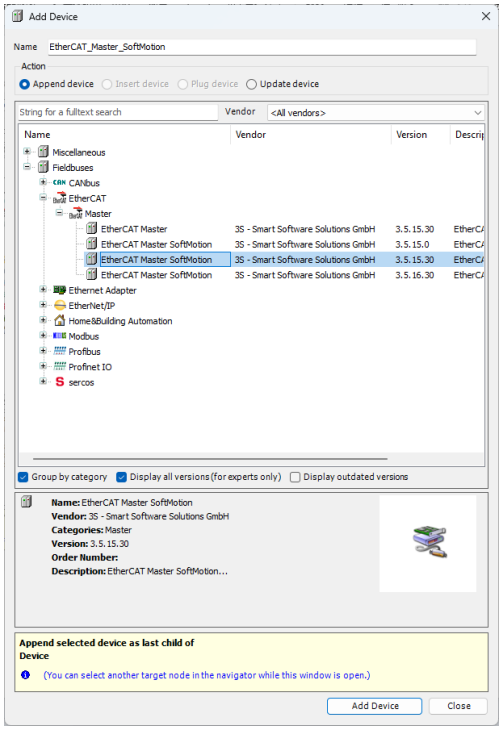
Step 3 Set relevant parameters.

Step 4 Write the program.

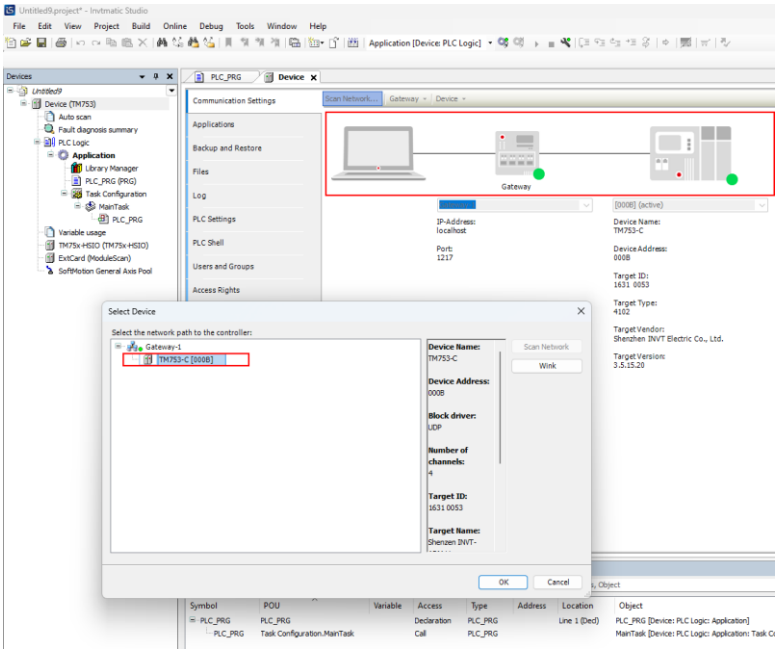
2.3.1 Adding a Device

Step 1 Right-click Device in the device tree of the software interface, select Add Device and then Fieldbuses > EtherCAT > Master > EtherCAT Master SoftMotion (corresponding master), and click Add Device. This will add the EtherCAT master bound to the ETH3 port by default.





Step 2 Double-click **Device** on the left side of the software interface, and click **Scan for Network** on the pop-up window. When the PLC light on the right is green, it indicates the PLC device is connected successfully and you can download the PLC program.



Step 3 Add a slave.

There are two methods to add a slave: automatic scanning for device configuration, and offline device configuration, which are introduced below respectively.

■ Automatic scanning for device configuration

Usually, after the PLC and EtherCAT devices are powered on and the EtherCAT bus has connected EtherCAT devices such as the servo motor and IO modules, the bus devices are scanned by software to automatically complete the configuration. The operation steps are as follows:

Step 1 Right-click the required master device in the device tree and select **Scan For Devices**, as shown in Figure 2-1. Then, the slave devices can be scanned and the scanning result window will pop up, as shown in Figure 2-2.

Figure 2-1 EtherCAT Master Scanning Interface

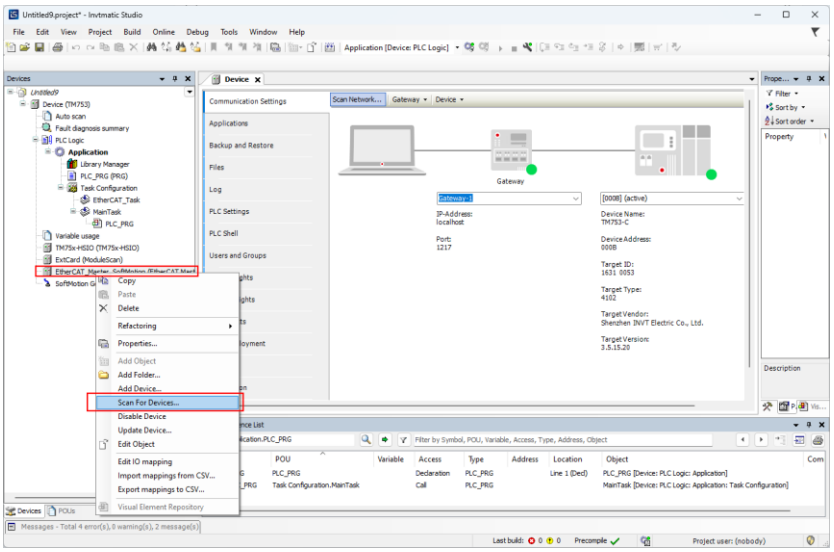
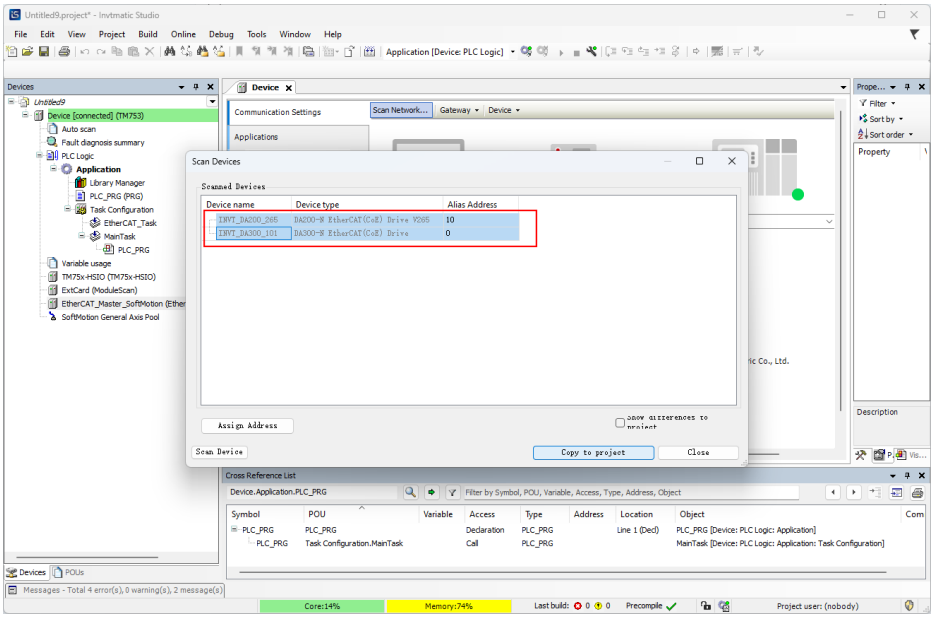
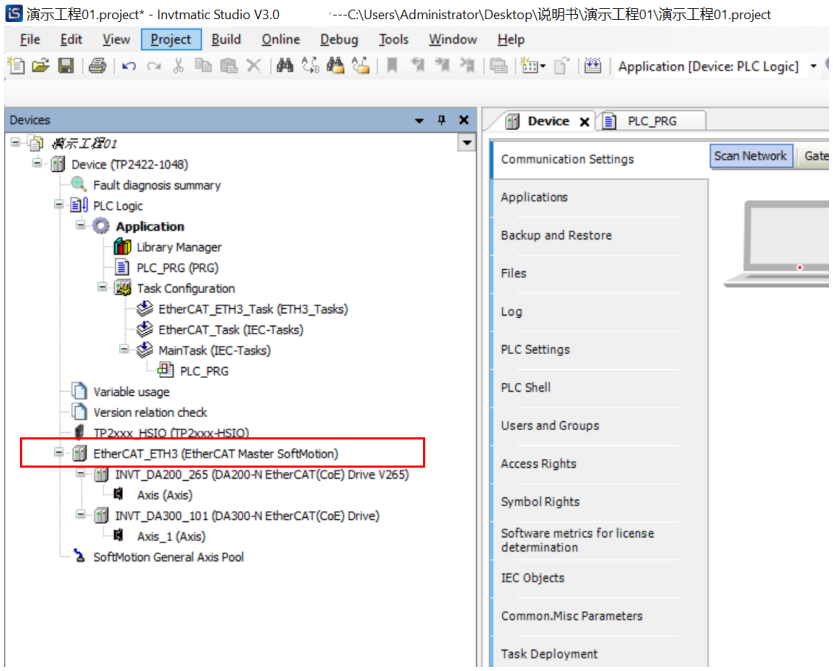


Figure 2-2 Adding a Slave to the EtherCAT Master



Step 2 Click **Copy to project** on the Scan Devices interface. The servo drive and the corresponding 402 axis will be automatically added.



Note: INVT servo motors do not require manual axis addition later, but third-party servo motors require it. The double underlines under the EtherCAT master indicate a prompt: if the required task "Ethercat_Task" does not exist, the EtherCAT master may not function correctly.

- **Offline Device Configuration**

  If the network connection for the PLC, the servo motor, and other devices is not available, you can perform device configuration directly on the Invtmatic Studio software.

Right-click **EtherCAT_Master_SoftMotion** in the device tree, and select **Add Device**.

Figure 2-3 Adding an EtherCAT Slave Offline

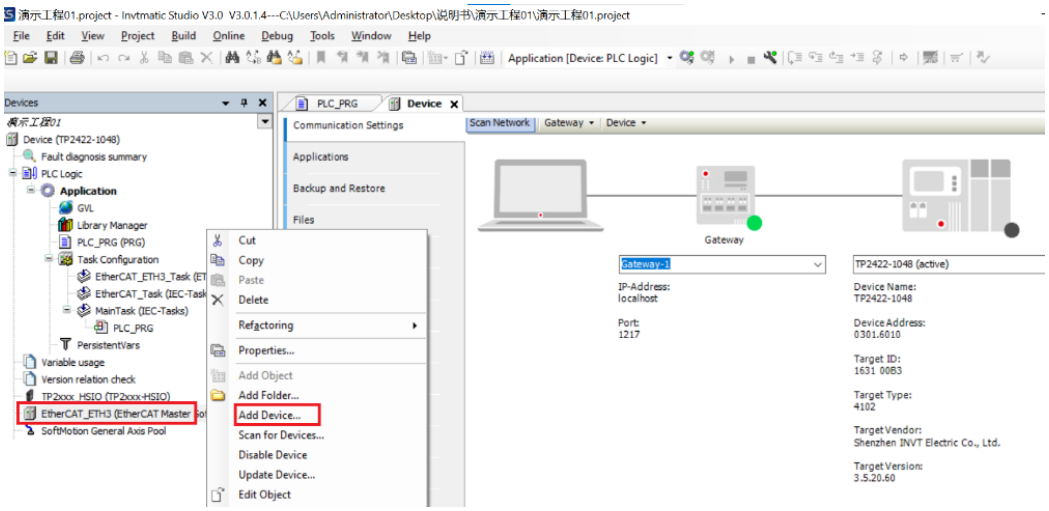
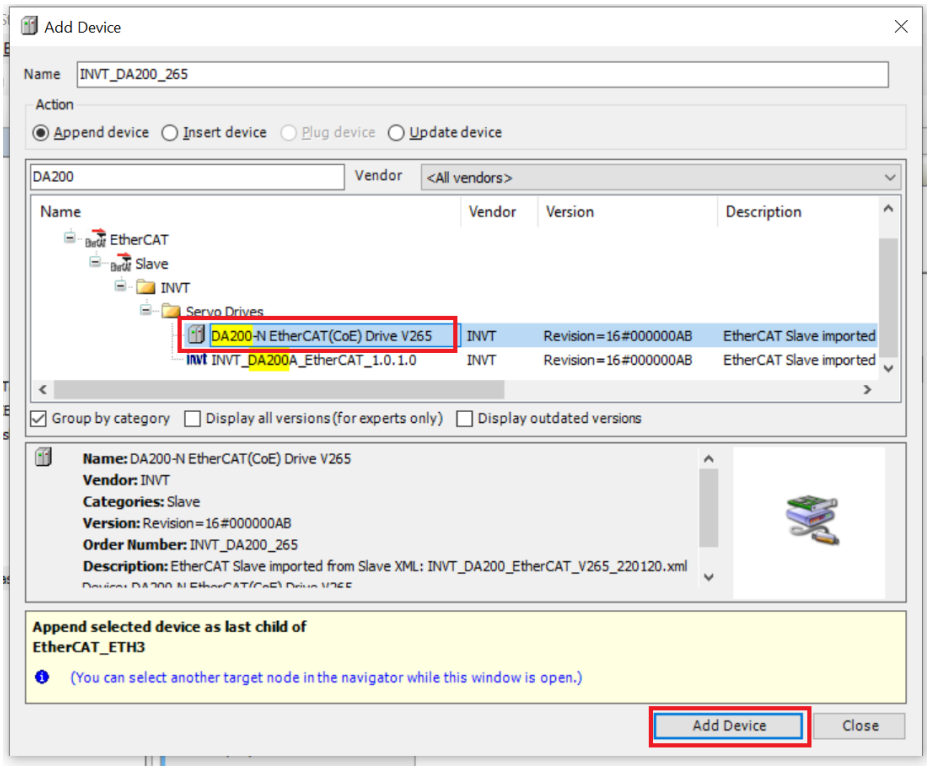
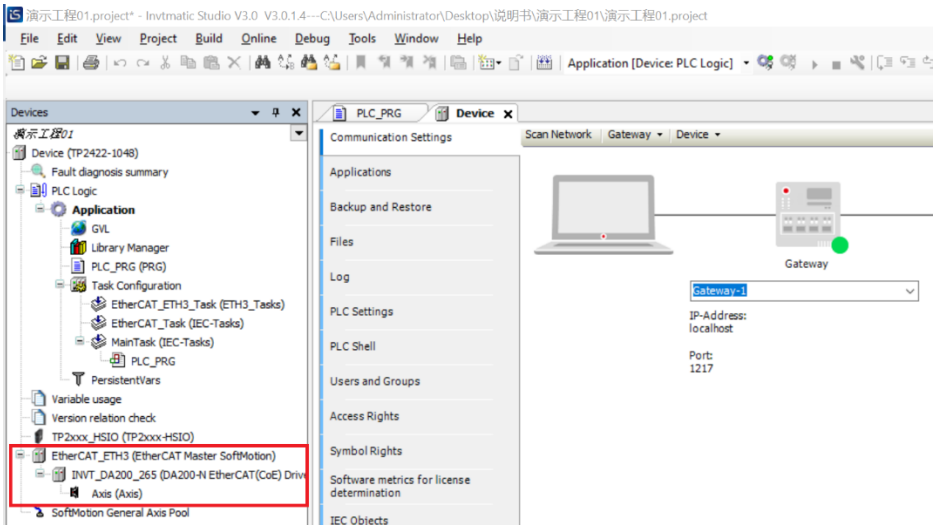


Figure 2-4 Selecting an EtherCAT Slave Device



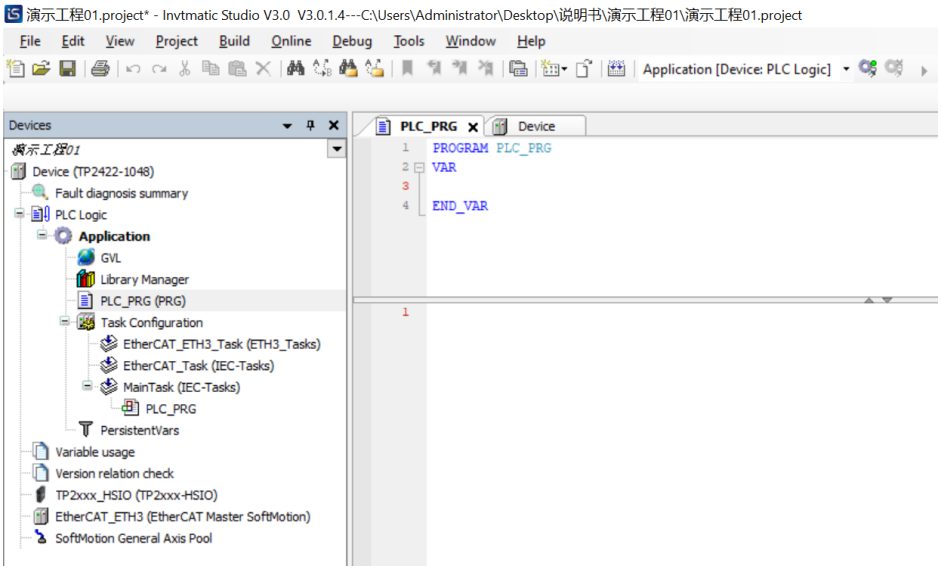
After the servo slave and 402 axis are added to EtherCAT, it is shown as below.



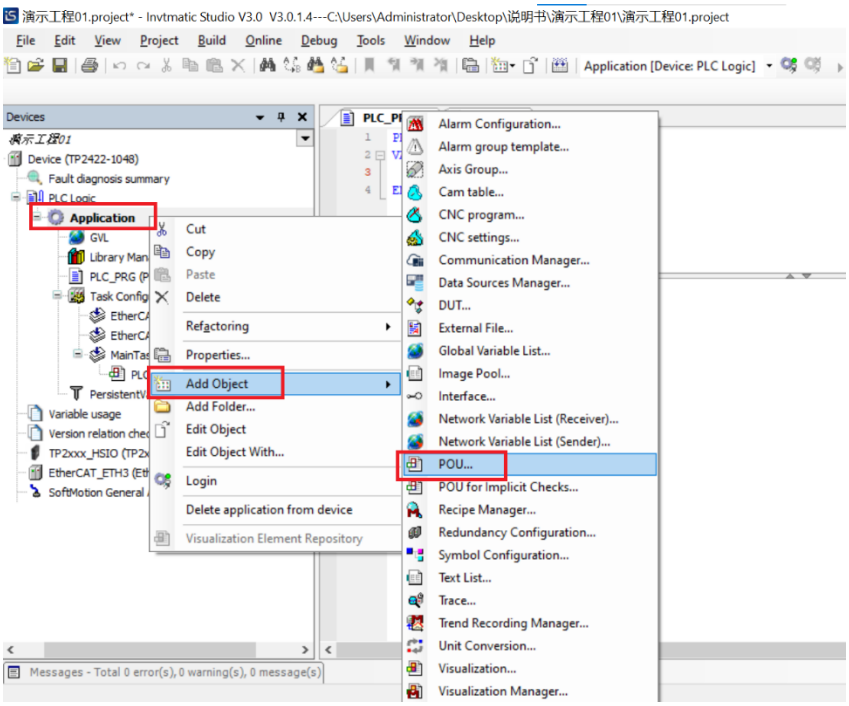
2.3.2 Writing a Function to Handle POUs

In Invtmatic Studio programming environment, there is an EtherCAT_ETH3_Task task, a HSIO_Task task, and a MainTask task for the default task configuration. The MainTask task contains a POU named PLC_PRG which is created when the new project is created, as shown in Figure 2-5. We need to create a POU dedicated to servo control and put it under the EtherCAT_Task task. The creation steps are as follows:

Figure 2-5 PLC_PRG Programming Interface

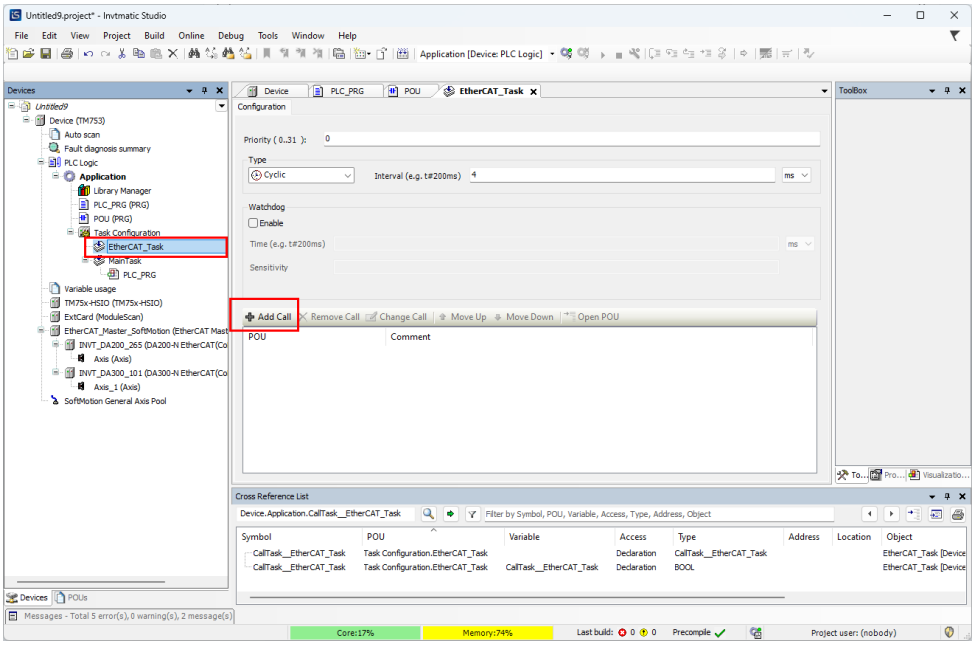


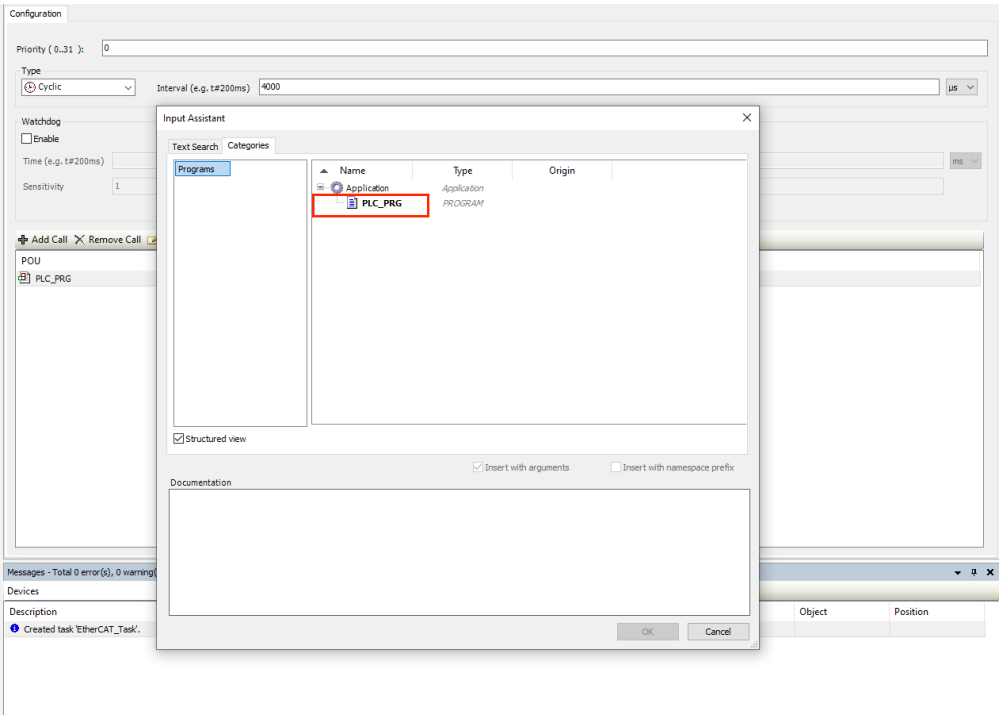
Step 1 Right-click **Application** in the device tree, select **Add Object** > **POU**, and add a POU dedicated to EtherCAT servo control.



Step 2 Double-click **EtherCAT_ETH3_Task** in the device tree, click **Add Call** on the configuration interface, and select a **POU**.

Figure 2-6 Calling a POU for the EtherCAT task

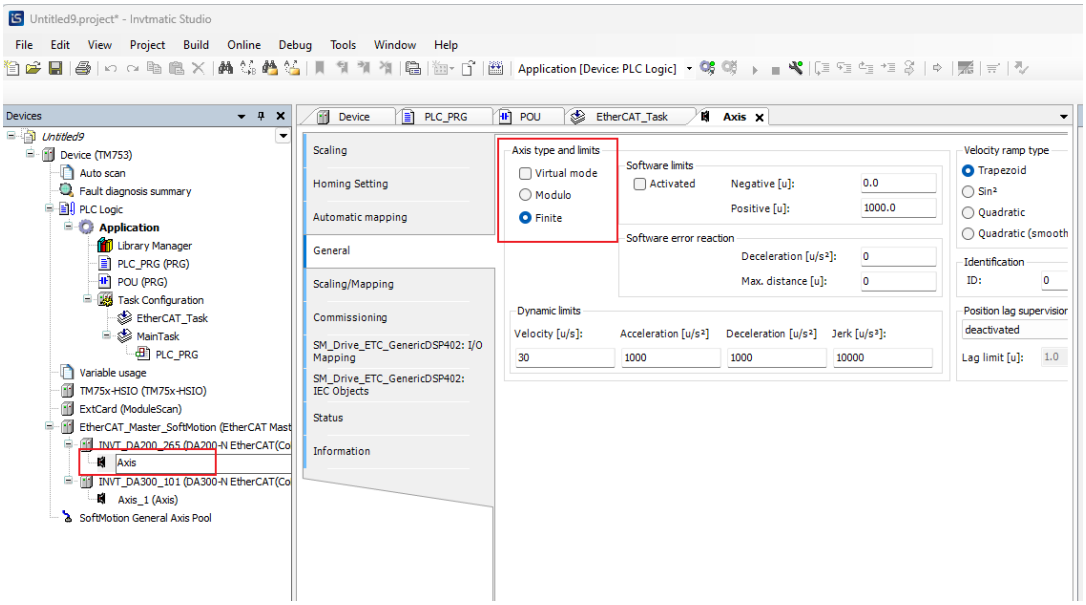




2.3.3 Setting Motor Parameters

For precise control of the movement position, the PLC must accurately calculate the position of the servo motor. Based on the operating characteristics and stroke characteristics of the application system, as shown in the figure below, select **Axis type and limits** so that the PLC can calculate the feedback information from the motor encoder to obtain the accurate position and avoid errors caused by the accumulation of encoder pulses.

Figure 2-7 Motor Parameter Settings



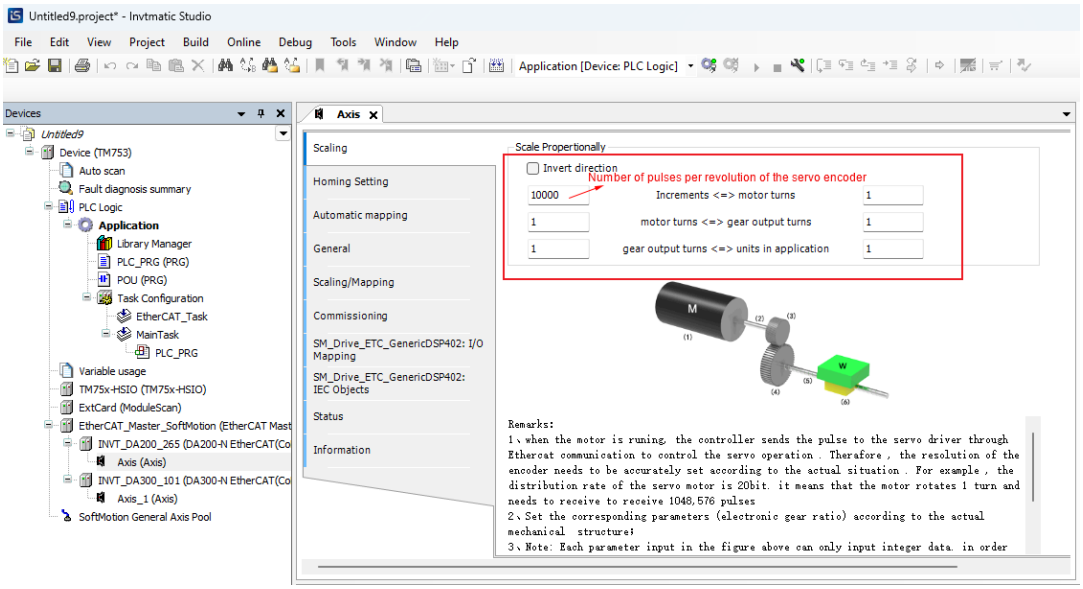
Note:

- For the reciprocating mechanism of the lead screw type, **Finite** is preferred as the lead screw stroke is limited and we should know its absolute position within the stroke range.

- For a single-direction axis, **Modulo** is preferred as the linear mode may cause position counting overflow, resulting in position calculation errors.

The encoder parameters of the motor (such as resolution) and the mechanical reduction ratio of the application system may be different. They need to be set based on the actual situation during programming, as shown in the following figure.

Figure 2-8 Motor Encoder Parameter Settings

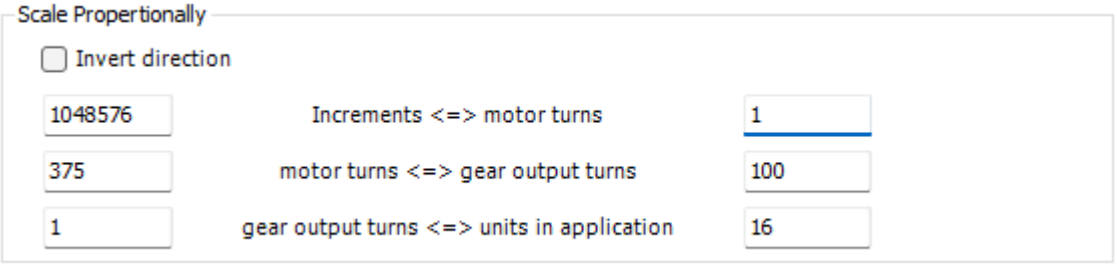


When the motor is running, the PLC sends the required number of pulses to the servo drive through EtherCAT communication to control the servo operation. Therefore, the encoder resolution needs to be accurately set according to the actual situation. For example, if the resolution of the servo motor is 20 bits, it means that for the motor to make one revolution, 1,048,576 pulses need to be received.

You need to set the corresponding parameters (electronic gear ratio) according to the actual mechanical structure.

Note: Please note that only integer values can be entered for the parameters shown in the above figure. In order to ensure that the ratio of the parameters in the corresponding rows on the left and right sides is effective, you can adjust the integer values on the left and right sides appropriately. For example, a 20-bit servo motor drives a lead screw of 16 mm (the screw slider moves 16 mm when the screw rod rotates 1 circle) after passing through a mechanical reduction mechanism with a ratio of 3.75:1. The parameter settings are shown in the figure below.

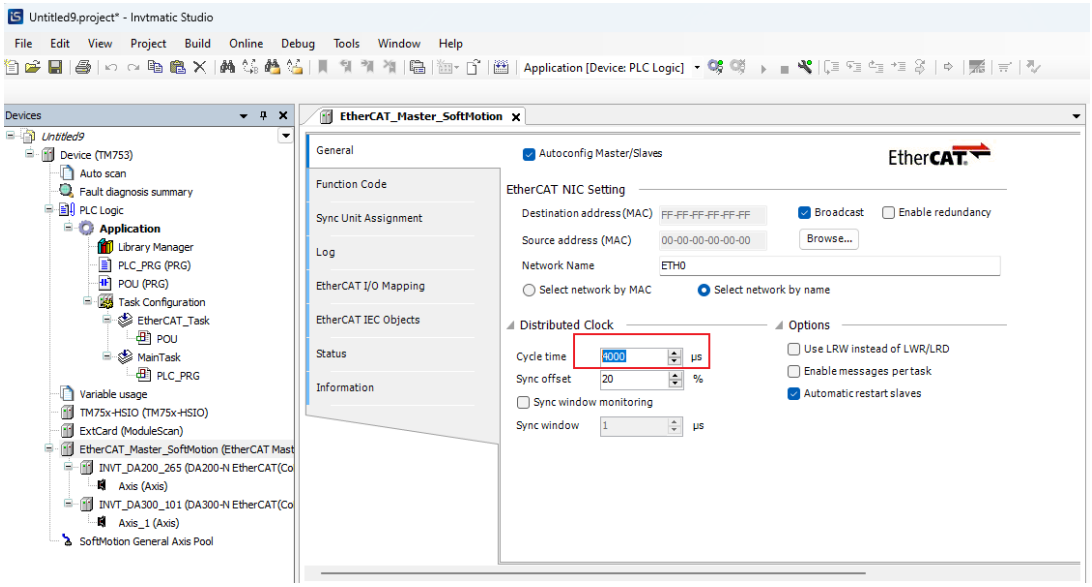
Figure 2-9 Parameter Setting Example



2.3.4 Writing a Function for Forward/Reverse Motor Direction Control

For the motion control of the servo axis, the default synchronization period is 4 ms, and you can select it according to actual needs, as shown in the figure below.

Figure 2-10 Servo Axis Motion Control Cycle Setting



The program in the figure below is written in the ST language, and the relevant code is as follows.

Figure2-11 ST Code

```
VAR
    MC_Power : MC_Power;
    MC_MoveAbsolute: MC_MoveAbsolute;
    iStatus: INT:=0;
    i:UINT:=1000;  //力矩限制
END_VAR
```

```
CASE iStatus OF
0:
MC_Power(Axis:= SM_Drive_GenericDSP402, Enable:= TRUE, bRegulatorOn:= TRUE, bDriveStart:=TRUE , );
IF MC_Power.Status
THEN
    iStatus:=iStatus+1;
END_IF
1:
MC_MoveAbsolute(Axis:=SM_Drive_GenericDSP402 , Execute:= TRUE, Position:=200 , Velocity:=5 , Acceleration:= 5, Deceleration:= 5,);
IF MC_MoveAbsolute.Done
THEN
MC_MoveAbsolute(Axis:=SM_Drive_GenericDSP402 , Execute:= FALSE,);
iStatus:=iStatus+1;
END_IF
2:
MC_MoveAbsolute(Axis:=SM_Drive_GenericDSP402 , Execute:= TRUE, Position:=0 , Velocity:=4, Acceleration:= 5, Deceleration:= 5,);
IF MC_MoveAbsolute.Done
THEN
 MC_MoveAbsolute(Axis:=SM_Drive_GenericDSP402 , Execute:= FALSE,);
   iStatus:=1;
END_IF
END_CASE
```

2.3.5 Compiling the User Program

If there is a writing error, the error type and reason will be listed in Figure2-11. Once you double-click the error description, the cursor will jump to the corresponding program editing window to facilitate revision. After the revision, compile it again until all compilation problems are eliminated.

The operation steps are as follows:

Step 1 Double-click **Device** in the device tree, and select **Communication Settings** > **Scan Network**.

Step 2 After selecting the corresponding device, click **Flash**. At this time, the PLC display on the connected device's LED screen will flash three times.

Step 3 After confirming the device, download the user program to the CPU module.

Figure 2-12 Program Compilation Interface

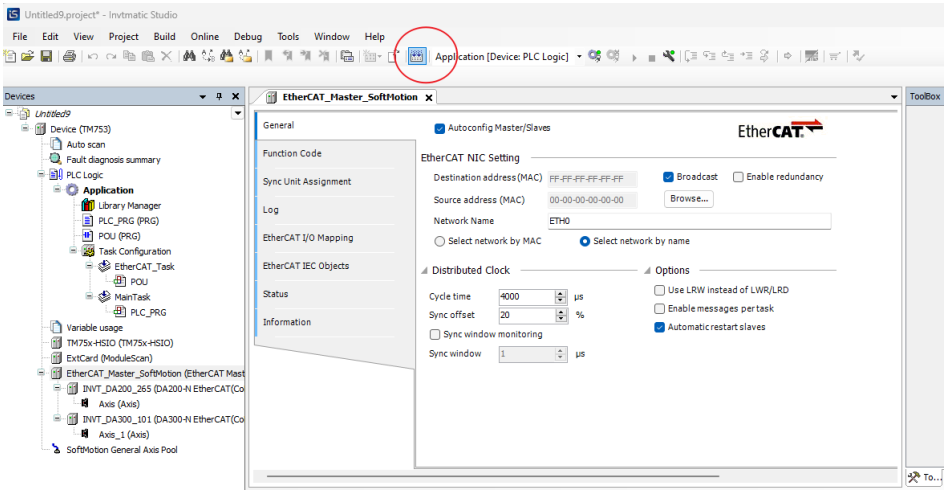


Figure 2-13 Connecting to the PLC

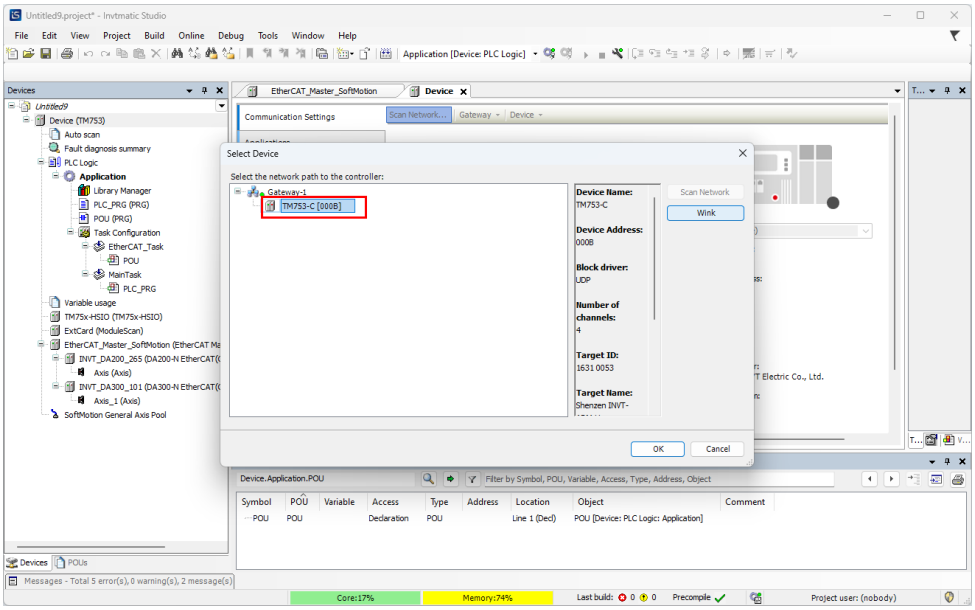
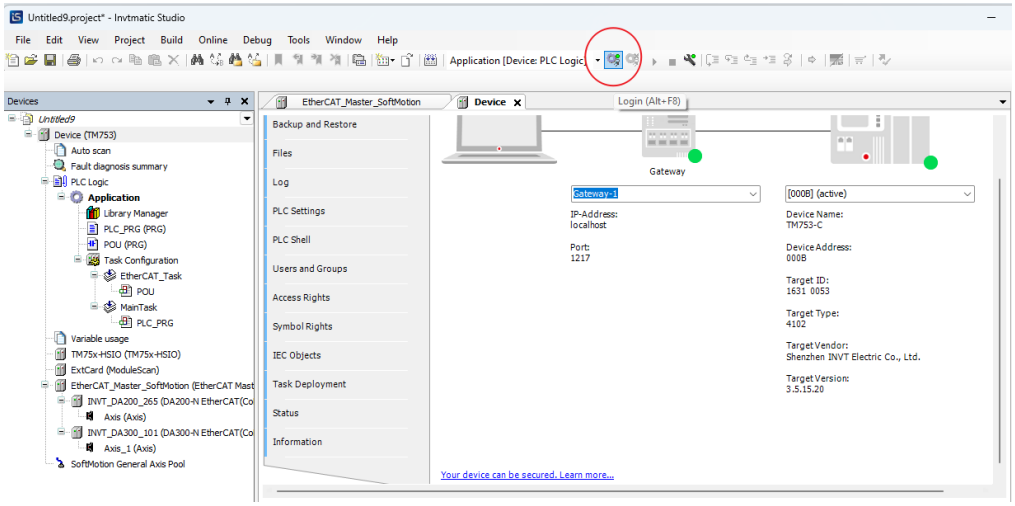
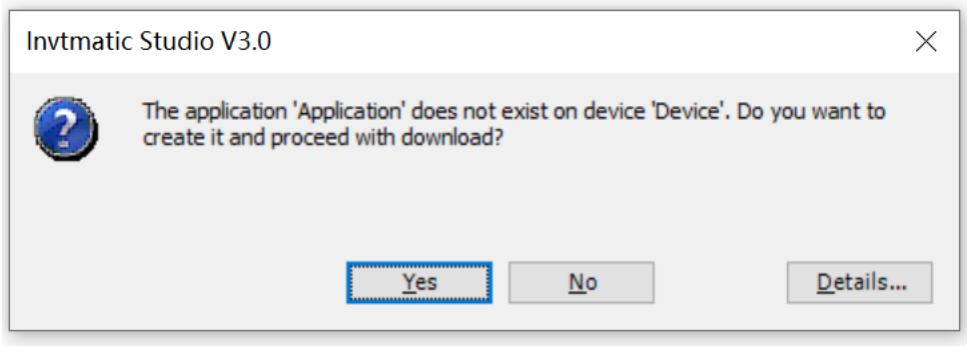


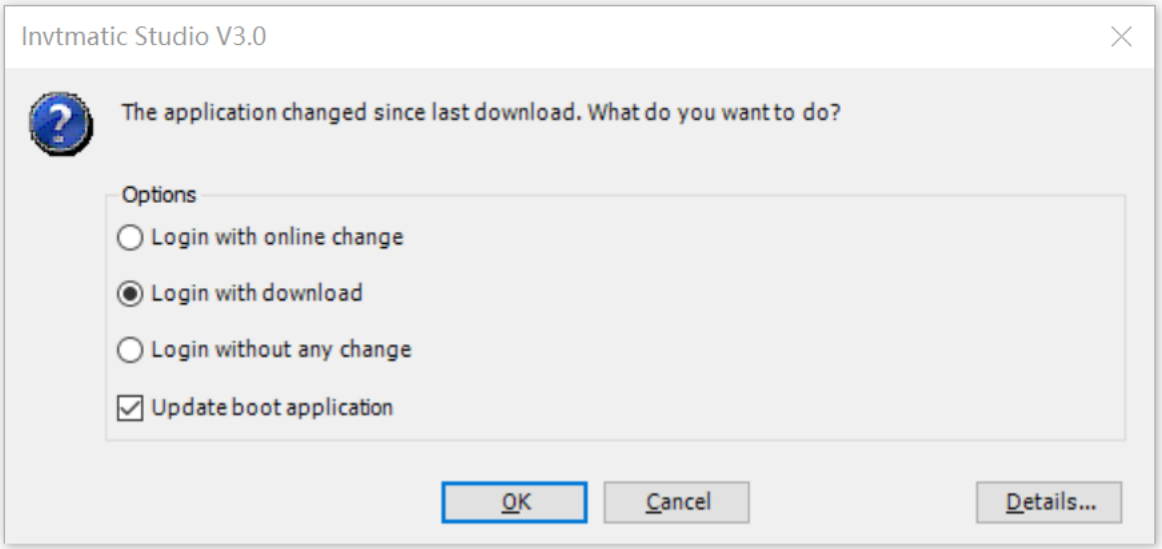
Figure 2-14 User Program Download



When downloading for the first time, a prompt as shown in the figure below will appear. Select **Yes**.



If an application already exists in the PLC, a different prompt will appear when downloading the changed application again, as shown below. Generally, select **Login with download**. The option **Update boot application** means that the PLC will automatically run this application the next time it is powered on.



Note: For large applications, execute **Clean** and perform a full download after 5 to 10 consecutive online changes, or when the code changes are substantial.

2.3.6 Running the Monitor Program

As shown in Figure 2-14, after logging in to the device, you can know the program is running by observing the actual operation of the servo or checking the position value of the servo axis of the upper computer. At this point, the required servo jogging and 2-cycle running triggering functions have been implemented, which means the programming process is complete. You can also view the CPU and memory usage in the lower left corner of the interface, as shown in Figure 2-15. In the task configuration interface, you can view the execution cycle time of each task (after the program is powered on for the first time, you need to right-click and select Reset to view the accurate time), as shown in Figure 2-16.

Once communication with the EtherCAT slave is established, you can use InvtMatic Studio to commission the servo through function codes. You can both view and configure the servo function code parameters.

Figure 2-15 Device Monitoring

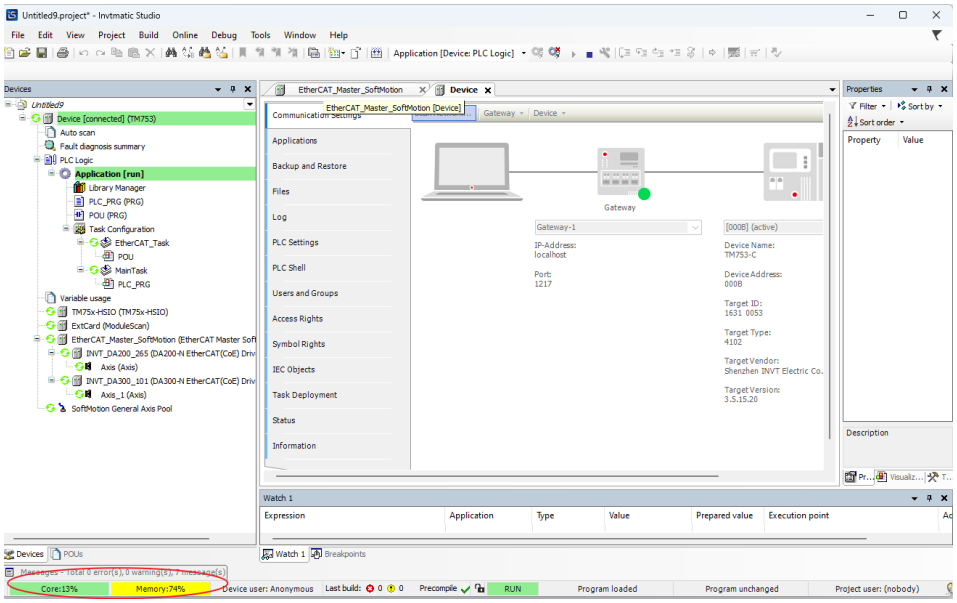
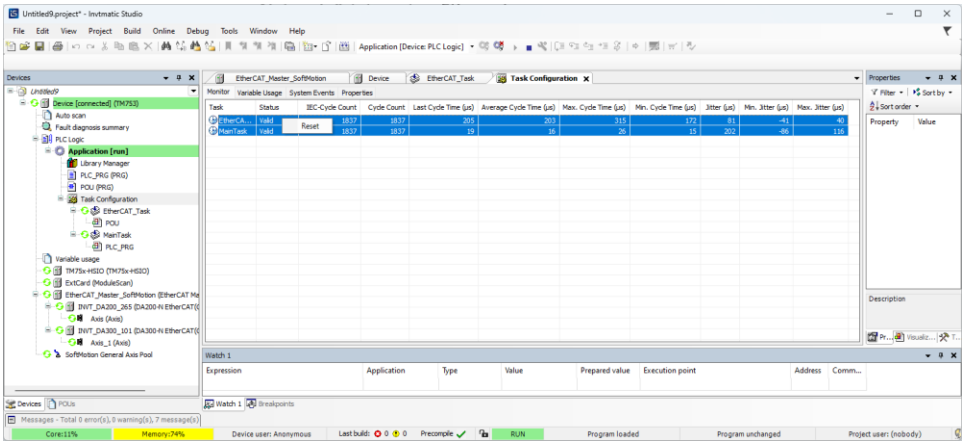
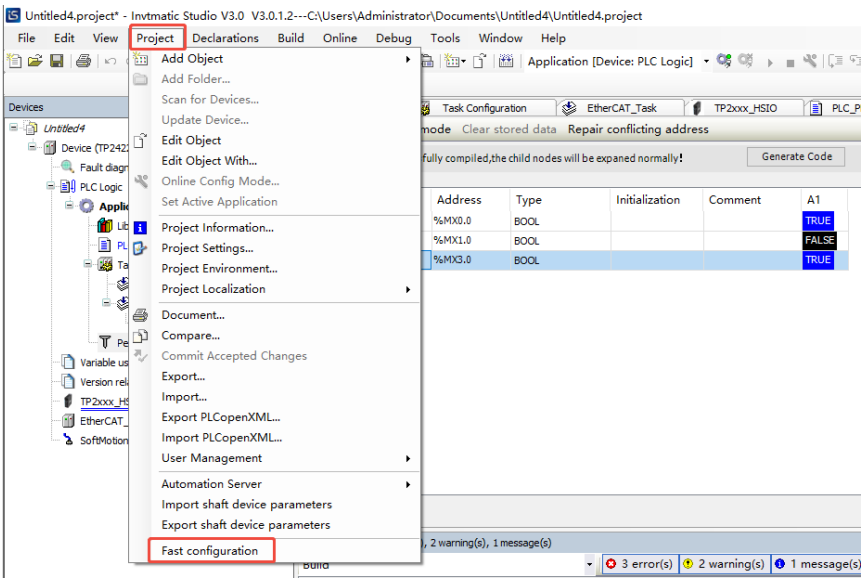


Figure 2-16 Task Monitoring



2.3.7 Fast Configuration

Select **Project** > **Quick Configuration** to open the **Fast Configuration** interface, as shown below.



Quick Configuration Function Description

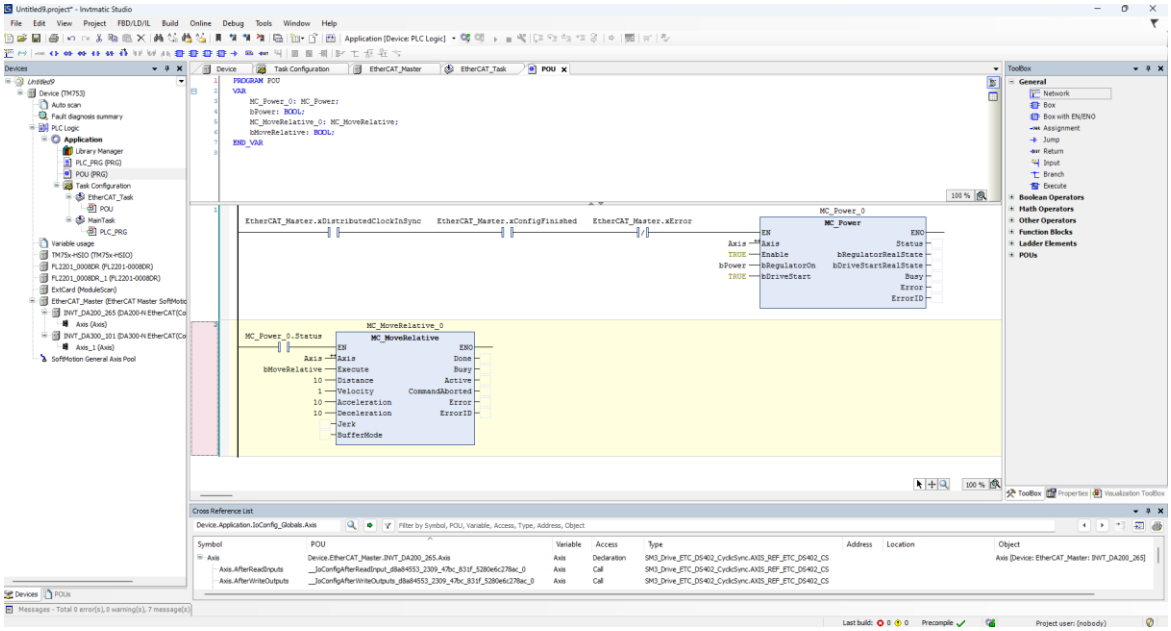
- Selected device preview window: Allows you to view the list of selected devices by category, quickly delete devices, and adjust the order of devices within the same type. Depending on the PLC model, it displays single or dual EtherCAT masters. The common types displayed include: Backplane Bus (Localbus), Modbus RTU 1, Modbus RTU 2, Modbus TCP 1, Modbus TCP 2, EtherCAT 1, and EtherCAT 2 (other types are currently not supported). This window does not display devices that have already been added to the project.
- Move up: Moves the selected device up in the preview window.
- Move down: Moves the selected device down in the preview window.
- List of devices to be added: Displays all subdevices that can be added based on the selected device type. Devices that cannot be added are hidden.
- All-version display: Switches between showing only the latest-version devices and showing devices of all versions. By default, only the latest-version devices are shown.
- Device details: Displays detailed device information, including vendor, type, ID, version, and description.
- Device addition: Allows you to customize the device name and quantity for batch addition. If the total number of added devices exceeds the limit, a warning prompt will appear and block the operation. When adding multiple identical devices at once, add sequential suffixes to their names to differentiate them. The default quantity is 1; if you manually change this number, it will automatically reset to 1 after the addition is complete.
- Reset: Clears all devices that have been added to the device preview.
- Add to project: Adds the selected devices into the project. The results of this operation will be displayed in the operation log.
- Close: Exits the **Quick Configuration** window.

3 Basic Functions

3.1 Interface Layout

Search in the **Start** menu of the Windows system or click the shortcut on the desktop to open the Invtmatic Studio software.

Figure 3-1 PLC Project Interface of Invtmatic Studio



The interface mainly includes the following content:

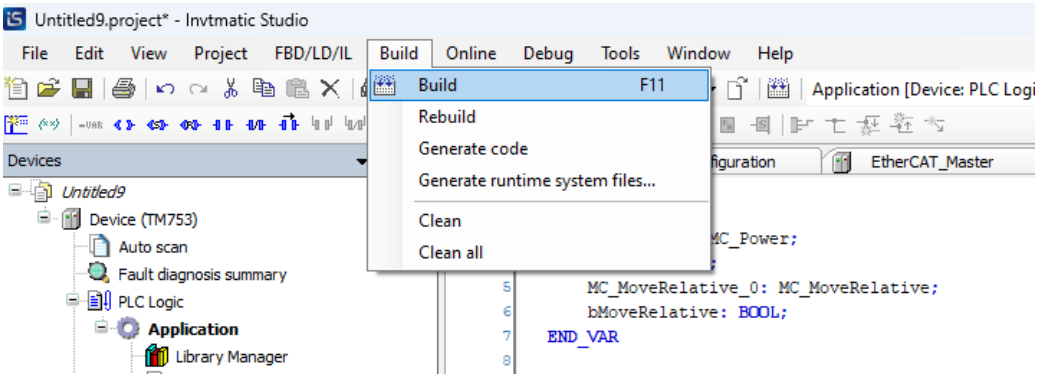
- PLC device information, you can right-click **Device** to change the PLC model;
- Application, i.e. the user program management unit;
- EtherCAT_Task, i.e. the EtherCAT bus task;
- HSIO pulse output task, i.e. the HSIO_Task;
- MainTask, i.e. the main program task;
- Local module configuration TP2xxx_HSIO;
- EtherCAT_ETH3, i.e. the EtherCAT bus configuration;
- EtherCAT bus slave INVT_DA200_265 and 402 axes, or remote IO modules.
- Shortcut keys for compilation, login, and debugging.

3.2 Build Menu

The **Build** menu integrates diverse functions such as compilation and clearing, which are described as follows:

- **Build**: Compile and link all codes into those that can be executed by the PLC.
- **Generate code**: For the Generate File function required for symbol configuration in label communication, you can click the Compile button to implement it, or click the shortcut key .
- **Generate runtime system files…**: Packages the project into a controller executable file, which can then be updated using the update tool.
- **Clean**: Clear the last compiled and downloaded information.
- **Clean all**: Clear compiled information, downloaded information, and reference information, and refresh all library and project data compilation information.

Figure 3-2 Compile Menu



3.3 Variable Usage Table

3.3.1 Overview

The variable usage table has the following functions:

- Display the address usage of I/Q/M areas, including the classified display of used addresses, conflicting addresses, and free addresses.
- Display the used size, available size, and usage rate of the program area, data area, power failure retention area, and I/Q/M areas.

3.3.2 Function Introduction

Select **Device** > **Variable Usage** in the device tree and double click **Variable Usage** to open the interface as shown in the figure below.

Note: The maximum storage capacities of the TP2000 series program area, the data area, the power failure retention area, the M area, the I area, and the Q area are 256M, 256M, 5M, 5M, 128K, and 128K, respectively.

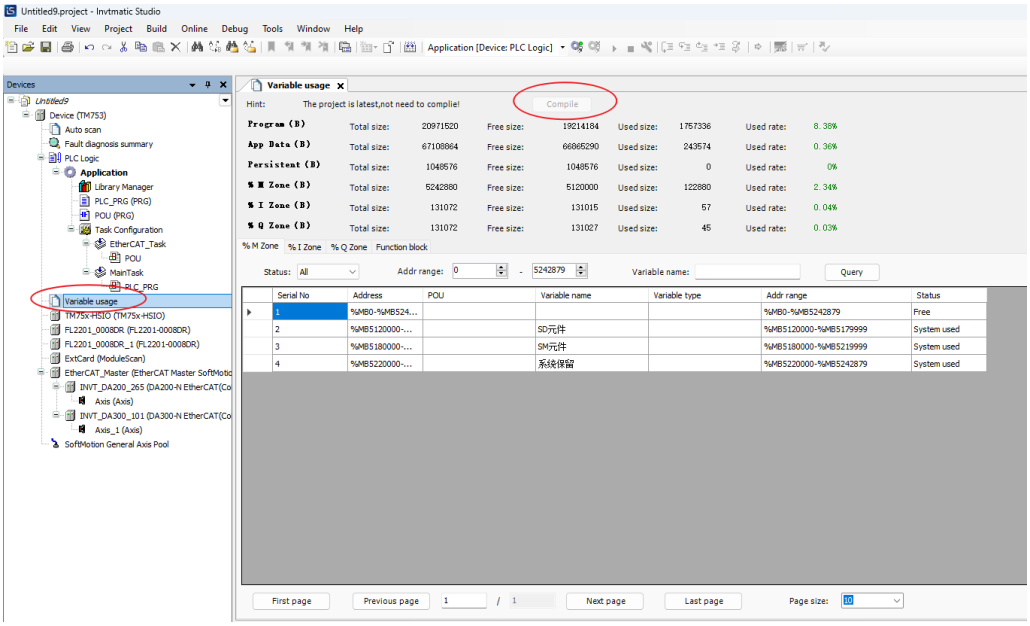
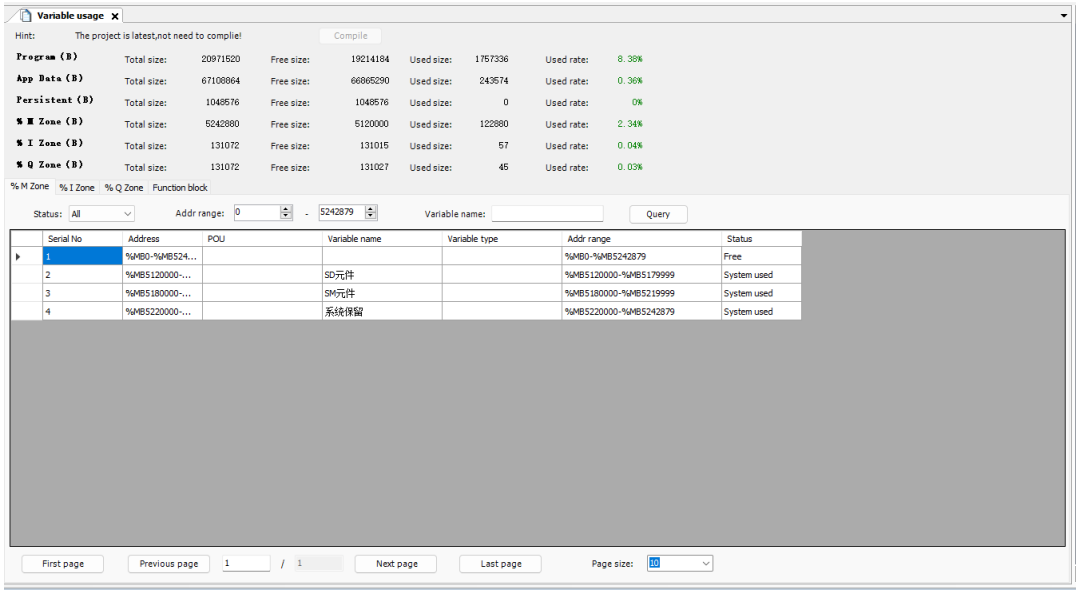


Table 3-1 Description of Variable Usage

Number	Item	Description
1	Data refresh area	If you click the **Compile** button, the interface will extract the information it needs based on the generated data and conduct a refresh once the compilation is succeeded. Alternatively, you can click the menu **Compile** > **Generate Code** in Figure 3-2 to trigger an interface refresh. **Note:** This function will refresh the interface only after successful compilation.
2	Area usage information area	It mainly displays the basic information of the program area, data area, power failure retention area, and I/Q/M areas, including the total capacity of the area, the size of the used area, the size of the available area, and the area utilization rate.
3	Detailed I/M/Q area usage information and function block information display area	It displays the address usage of I/M/Q areas, including address-associated variable information, address conflict status, information of unused address areas, etc. The function block information mainly includes the type and size of the function block called in the program. It also supports functions such as search, address range setting, and page number positioning.

The information display interface is shown in the figure below. It is divided into 4 display options, namely **%M area**, **%I area**, **%Q area**, and **Function block size**, which display the usage information of the corresponding areas respectively.



3.3.3 Menu Options

Menu options include display options, query, address range settings, and page turning options.

Table 3-2 Description of Variable Usage Statistics Menu

Number	Item	Description
1	Display options	For the **%M area**, **%I area**, and **%Q area**, the status options include **All**, **In use**, **System use**, **Conflict**, and **Idle**; For the **Function block size**, the status options include **All**, **Function block**, **Union**, **Structure**, and **Alias**.
2	Query options	You can perform character matching on the names in the four columns of **Address**, **POU**, **Variable name**, **Variable type**, and **Address range**. If any one of them matches, the condition is met. **Note:** Any changes in the query options will trigger a table refresh.
3	Address range settings	You can check the address range set by you according to actual needs, and the range cannot exceed the limit of the corresponding area. You need to click the **Query** option to enable the range display. **Note:** This item is not available in the function block size interface.
4	Page turning options	To ensure the table refresh performance, you can add a maximum of 1,000 address segment units by default. When the data exceeds 1,000 address segment units, the system will use paging display. You can click **Previous Page**/**Next Page** or enter a specified page number, and the system will jump to the corresponding page.

Table options: They mainly display the usage information of address segments that meet the filter conditions.

Table 3-3 Description of Variable Usage Statistics Table

Number	Item	Description
1	Address	The address area is displayed in ascending order, with the smallest unit being Byte. When the address occupied by a variable does not conflict, it will be displayed in the form of an address area (e.g.: %MB0–%MB3). One address may be associated with one or multiple variables. When there are multiple variables, the address bar will display multiple duplicate addresses.
2	POU	All POU names of variables area displayed.
3	Variable Name	The variable name associated with the area is displayed.
4	Variable type	The variable type is displayed.
5	Address Range	The entire address range of the variable is displayed.
6	State	The usage of the address area is displayed. When it is used, the status is **In Use**; when there is a conflict, the status is **Conflict**; when it is free, the status is **Idle**. **Note:** ● Address conflict detection is performed based on the byte position. When different variables use different bits of the same address, the system will mark it as an address conflict during detection. The specific conflict situation is based on the actual bit usage. ● When a user-defined variable conflicts with the IO address assigned to the device, it only means that the same address is used in multiple places. You can determine whether there is a problem based on whether the actual function is used.

3.3.4 Direct Address Storage Area

For PLC data, the addresses in the %I and %Q areas cannot be saved after a power failure, but the addresses in the %M area can still be saved. The TP2000 series programming system provides a 128kB (Byte) input area (%I area), a 128kB (Byte) output area (%Q area), and a 5MB (Byte) storage area (%M area). The first 480kB in the storage area can be used directly by users, and the last 32kB is the system area (mainly used as soft components) and should not be used directly by users. During programming, you can directly access addresses or define variables and map them to addresses for indirect access. The definition of storage areas and the address range used by them are shown in the following table.

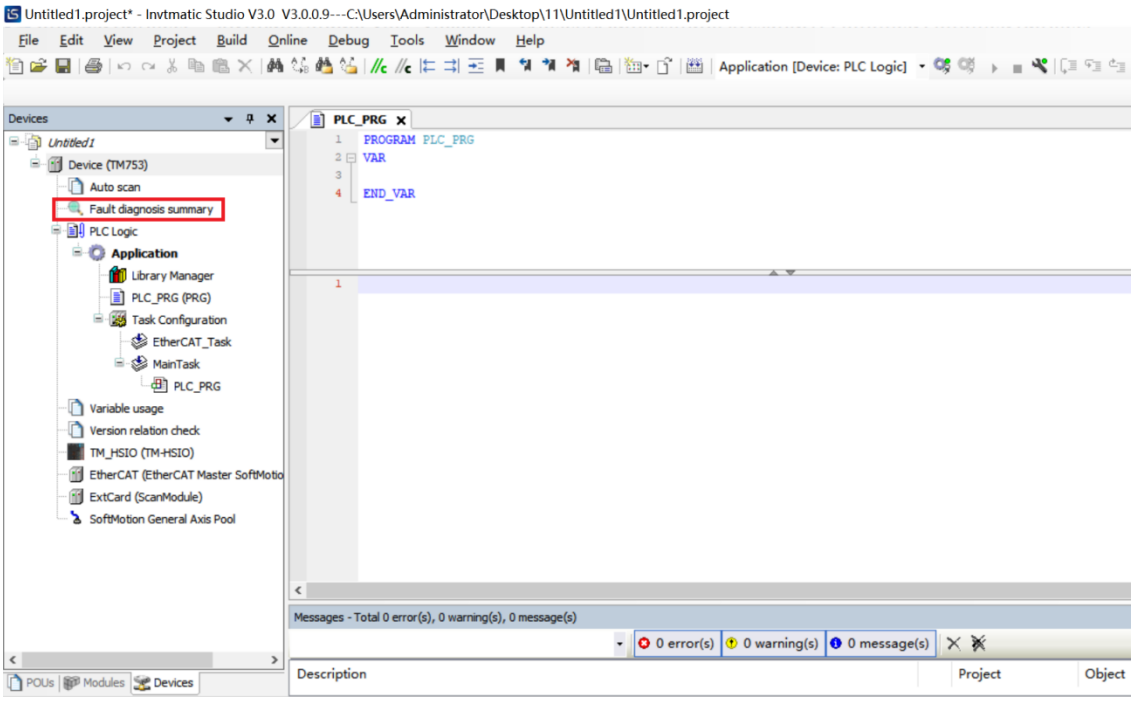
Table 3-4 Description of Storage Areas and Address Range Used by Them

Area	Purpose	Size	Address Range
I area (%I) 128kB	User area	64k Words	%IW0–%IW65535
Q area (%Q) 128kB	User area	64k Words	%QW0–%QW65535
M area (%M) 5MB	User area	2530k Words	%MW0–% MW2,589,999
	SM components	20000 Words	%MW25,90,000-%MW2,609,999
	Extended user area	11440 Words	%MW26,10,000-%MW2,621439

3.4 Fault Diagnosis

Diagnosis is intended to quickly locate errors that occur during PLC operation and find solutions according to the error information and status. The diagnostic interface of Invtmatic Studio can only be obtained and displayed after logging in to the PLC. The Invtmatic Studio programming system supports the diagnosis of various communication devices and can generate fault information, offline information, and other information according to the actual running status of each communication device. The module types involved in fault diagnosis mainly include CPU module, Modbus, ModbusTCP, etc. The Invtmatic Studio programming system mainly provides four diagnosis routes: configuration diagnosis, diagnostic information list, device self-diagnostic information list, and diagnostic programming interface. All diagnoses are obtained through fault code analysis, and the fault codes correspond to the diagnostic programming interface.

Fault diagnosis can be used to display fault information of all devices, and provide detailed description of relevant fault information and methods for troubleshooting the causes. It can also provide more detailed diagnostic information for special situations. After the device is connected, double-click **Fault diagnosis summary** in the device tree to open the device fault diagnosis interface. The fault diagnosis interface is shown in the figure below.



The functions of the fault diagnosis interface are described as follows:

- **Device Type Window**

It displays the current fault type and provides the fault display filter function, which can display fault information by device type. Device types include CPU module, Modbus RTU module, and Modbus TCP module. You can select a different device type, and the diagnostic display list will show the corresponding type of diagnosis. All device diagnoses are displayed by default.

- **Description of Interface Options**

  ✧ Device Type: Filter a certain type of faulty bus device.

  ✧ Module Name: Filter faulty devices with a specific name.

  ✧ Search: Search for matching fault information based on the device type or module name.

- ✧ Refresh: Used to refresh device fault information.
- ✧ Clear: Clear the fault information in the table.
- ✧ Export to Excel: Export fault information in the table.
- ✧ Fault Information Count: Display the number of faults.
- ✧ Fault Information List: mainly used to display specific module fault information, including device type, module name, error code, and error name.
- ✧ Detailed Information window: When a certain piece of fault information is selected in the Fault Information List, the detailed information of the fault will be displayed in the detailed information window, which includes three options: Error Details, Troubleshooting, and In-depth Diagnosis. Error Details window: Display possible causes of the fault. Troubleshooting window: used to provide the specific operation method for troubleshooting. In-depth Diagnosis window: For some complex errors, more detailed information is needed to locate them.

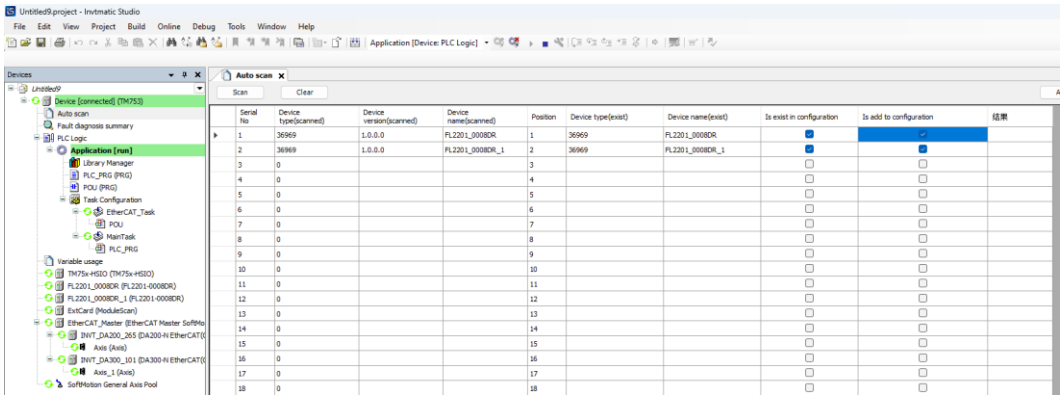
3.5 Automatic Scanning

If you need to increase the number of IO ports, you can expand the IO module. The steps to add an expansion module to the PLC body in the software are as follows:

Step 1 Open the Invtmatic Studio programming software, create a new project, and select the programming language.

Step 2 Connect to the PLC and scan local expansion modules. You need to log in to the PLC, but you may not start it.

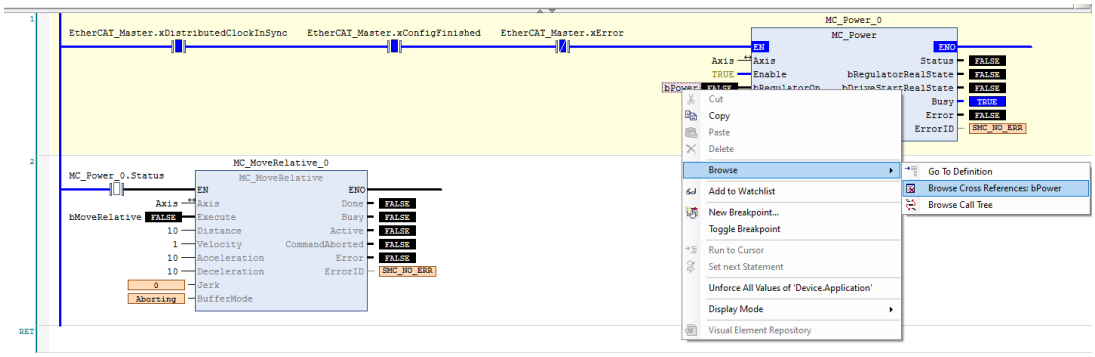
Step 3 Double-click **Automatic Scanning** in the device tree, and click **Scan** in the automatic scanning interface. Then, the installed extension modules will be automatically scanned in the list, as shown in the figure below. Click **Add to Configuration** in the upper right corner to complete the scanning and configuration addition of the expansion module.



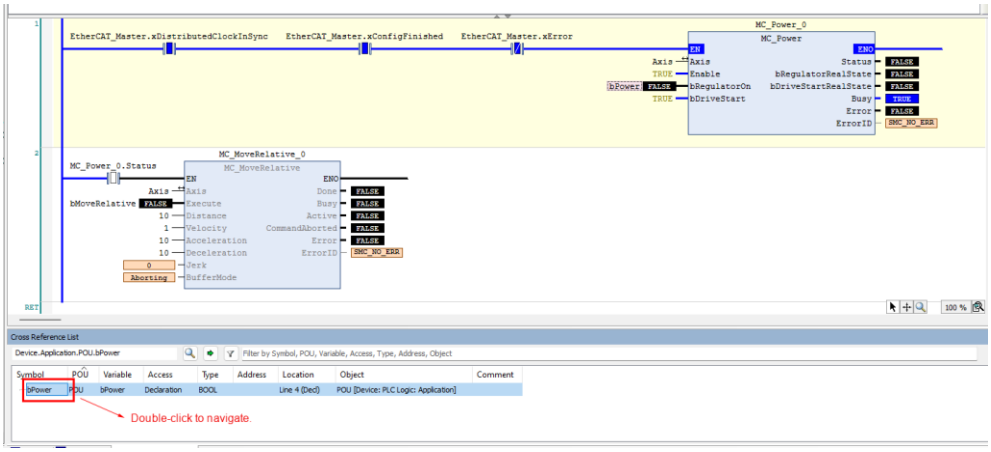
3.6 Cross-reference

The cross-reference function can be used to quickly find the calling location of the target object in the entire project. The operation steps are as follows:

Step 1 Find the object you want to cross-reference, right-click and select **Browse** > **Browse cross References:xEnCntCout**.



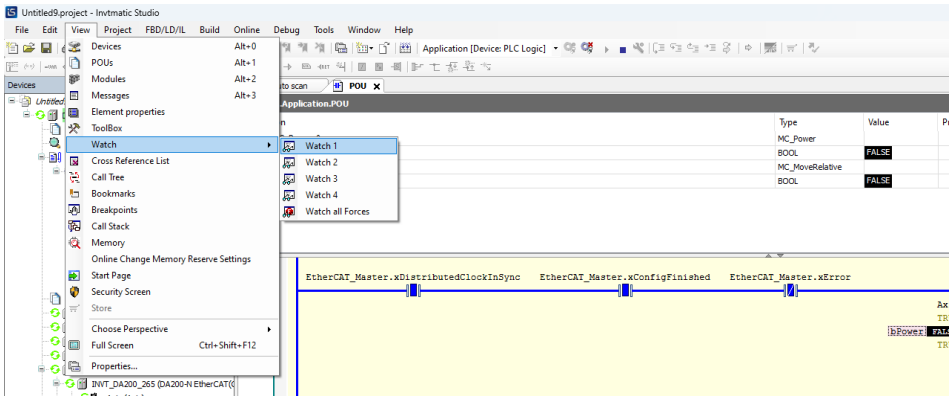
Step 2 View the usage of the “target object” in the entire project from the **Cross-reference List** under the project. Double-click the information in the cross-reference list to jump to the specific usage location in the project.



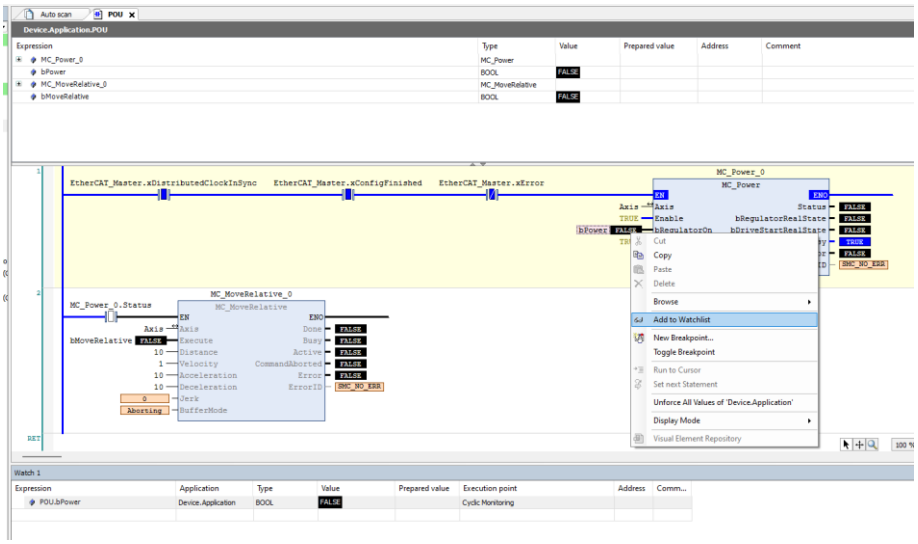
3.7 Monitor List

The Monitor List function can be used to monitor variables and addresses. During program running, the Monitor List can be used to view the data type and current value of the monitored variable, and the variable can be assigned a value by writing a value. The operation steps are as follows:

Step 1 Click **View** > **Watch** in the Toolbar, and add **Watch 1** to the monitor options.



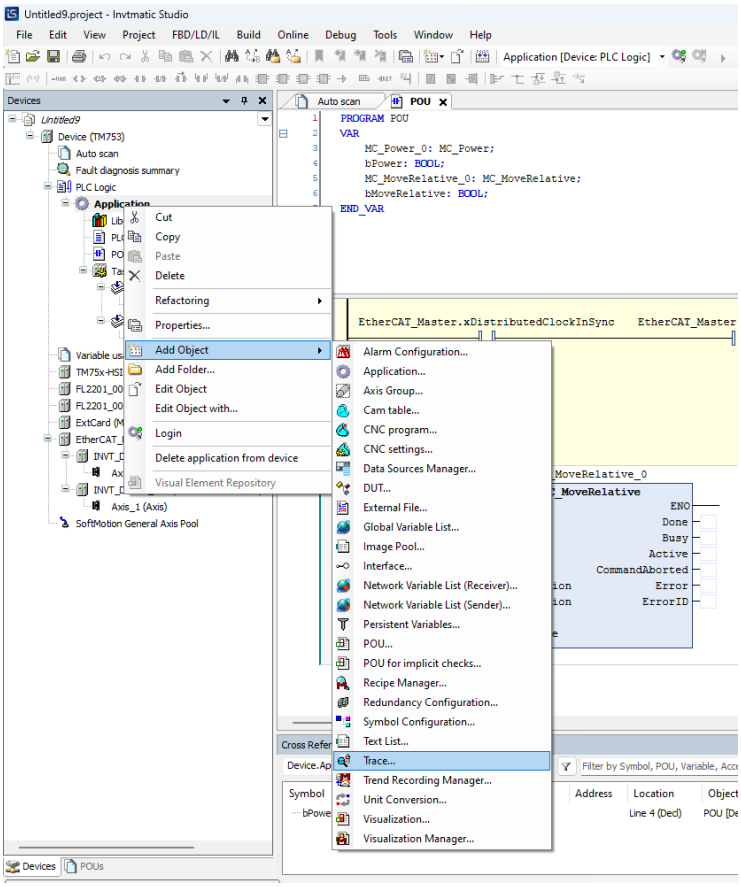
Step 2 Add the variables you want to monitor under the project bar, or right-click **Variable** and select **Add to Watchlist**.



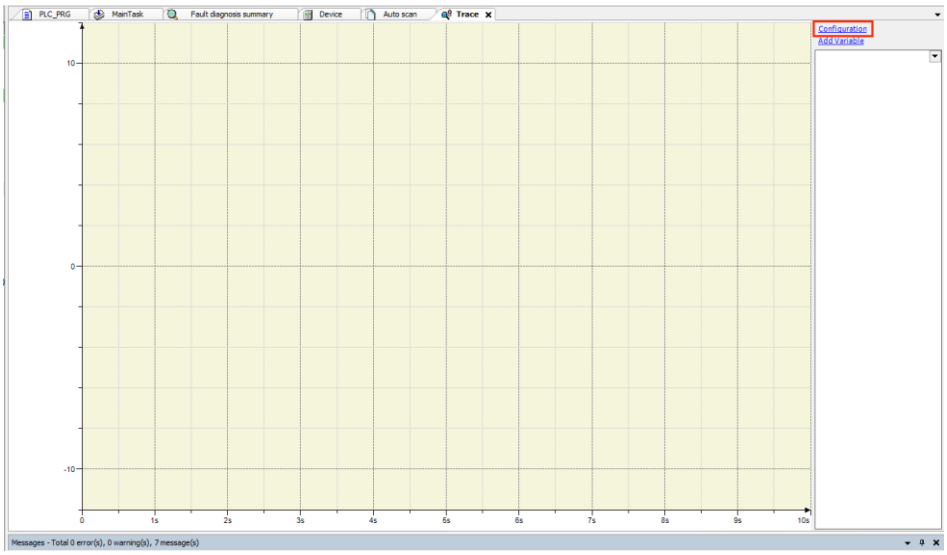
3.8 Sampling Tracking

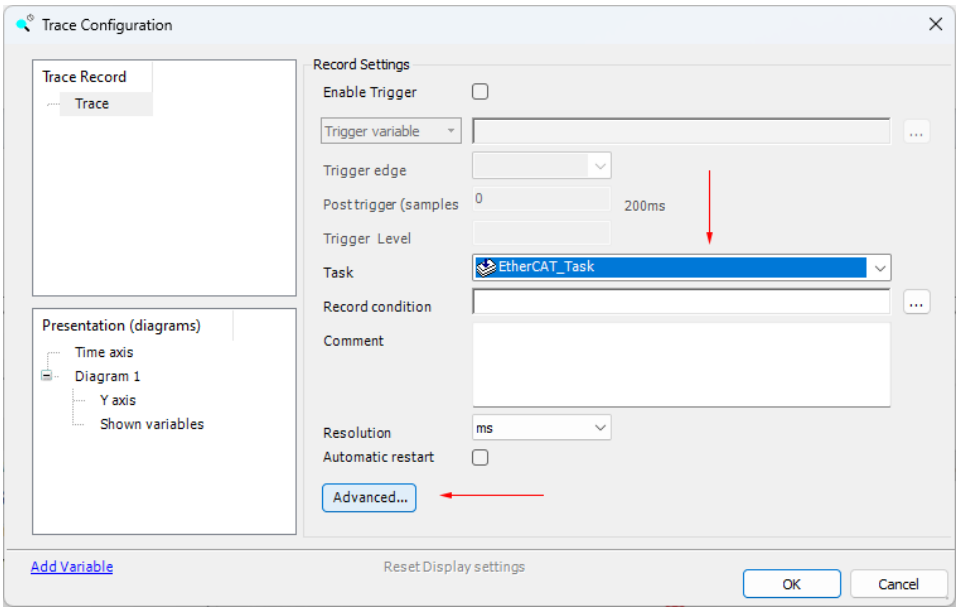
Invtmatic Studio can display the trace curve of variable changes during program debugging. The operation steps are as follows:

Step 1 Right-click **Application > Add Object > Trace** in the project tree, as shown in the figure below.

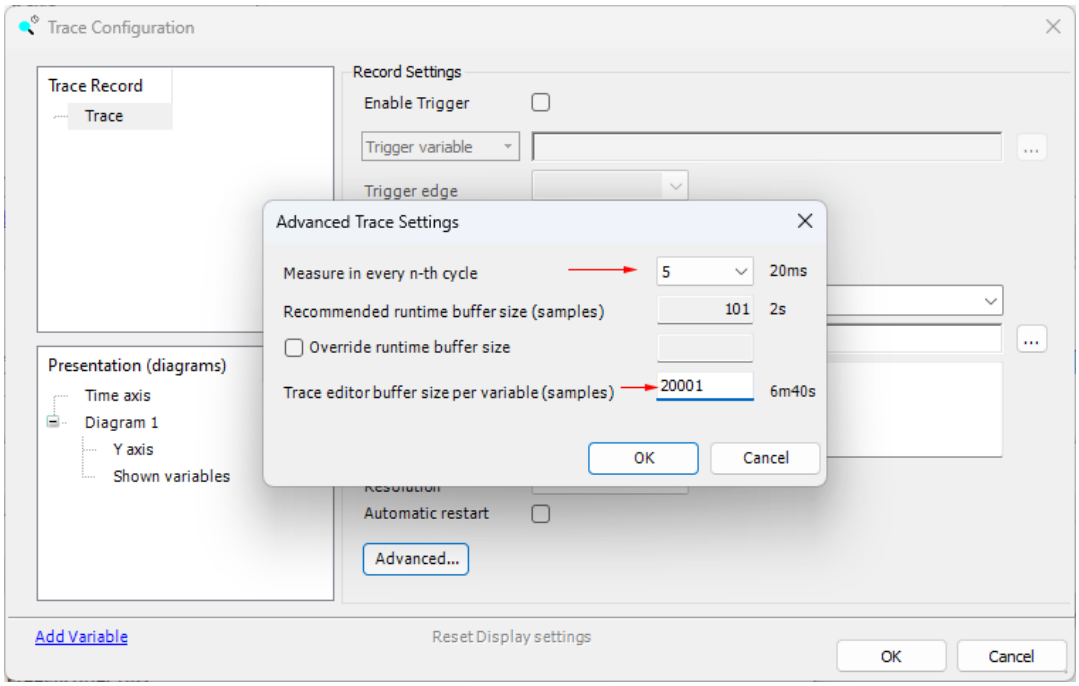


Step 2 In the pop-up dialog box, fill in a name for the trace curve, such as Trace in the figure below. Double-click the trace curve name in the project tree to open the trace curve interface. In the upper right corner of the trace curve interface, click **Configuration**. Then, a dialog box pops up as shown below. In the Task Options box, select **EtherCAT_Task**.

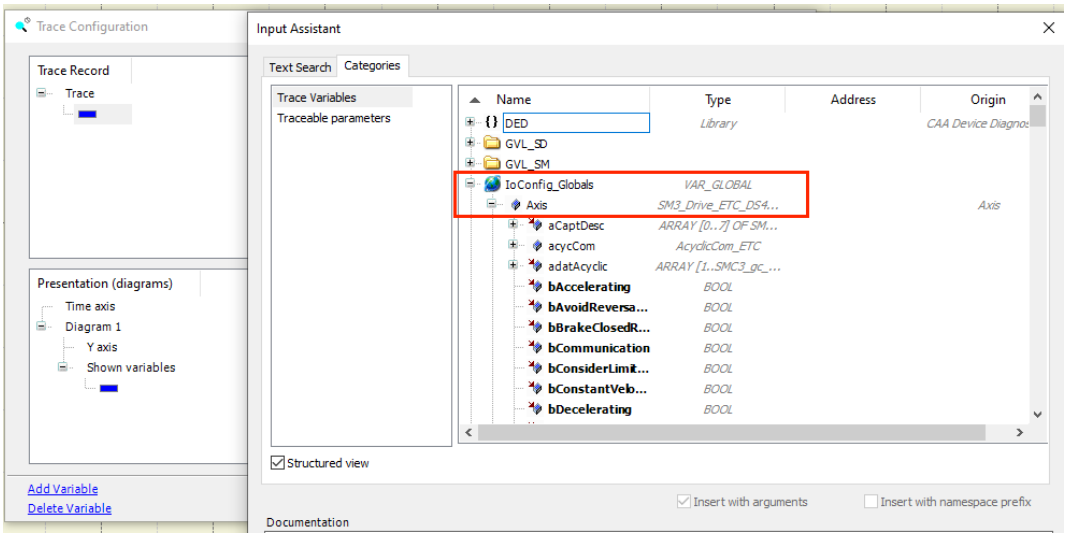


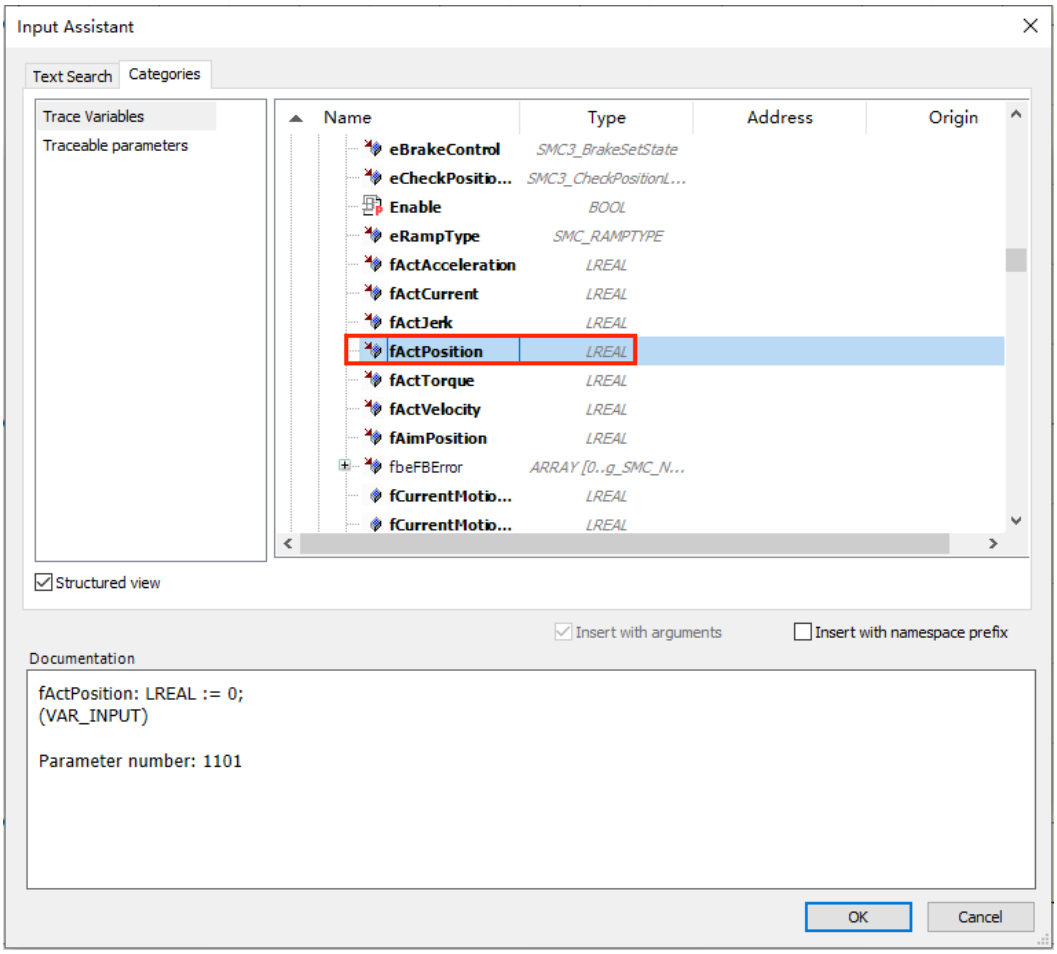


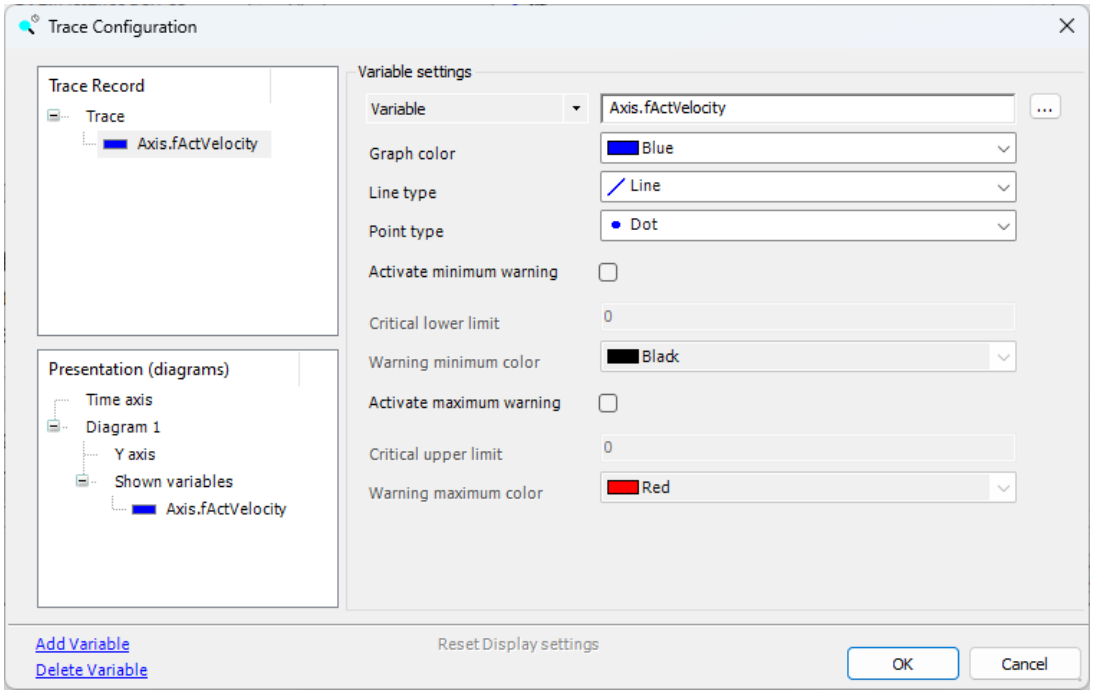
Step 3 Click **Advanced** in the Trace Configuration interface to show the **Advanced Trace Settings** Interface, in which you can set **Measure in every n-th cycles** as the sampling period and add **Trace editor buffer size per variable (samples)** to extend the tracking time.



Step 4 Select **IOConfig_Globals** > **Axis** from Variables in the Trace Configuration interface, find the axis position variable **fActPosition**, click **OK**, and then add the axis speed variable **fActVelocity**.




Input Assistant
Text Search
Categories
Trace Variables
Traceable parameters
Name
Type
Address
Origin
eBrakeControl SMC3_BrakeSetState
eCheckPositio... SMC3_CheckPositionL...
Enable BOOL
eRampType SMC_RAMPTYPE
fActAcceleration LREAL
fActCurrent LREAL
fActJerk LREAL
fActPosition LREAL
fActTorque LREAL
fActVelocity LREAL
fAimPosition LREAL
fbeFBError ARRAY [0..g_SMC_N...
fCurrentMotio... LREAL
fCurrentMotio... LREAL
Structured view
Insert with arguments
Insert with namespace prefix
Documentation
fActPosition: LREAL := 0;
(VAR_INPUT)
Parameter number: 1101
OK
Cancel


Trace Configuration
Trace Record
Trace
Axis.fActVelocity
Presentation (diagrams)
Time axis
Diagram 1
Y axis
Shown variables
Axis.fActVelocity
Variable settings
Variable
Axis.fActVelocity
Graph color
Blue
Line type
Line
Point type
Dot
Activate minimum warning
Critical lower limit
0
Warning minimum color
Black
Activate maximum warning
Critical upper limit
0
Warning maximum color
Red
Add Variable
Delete Variable
Reset Display settings
OK
Cancel

3.9 Persistent Variable

3.9.1 Characteristics

Power failure retention variables can retain their original values after the PLC loses power or the program is downloaded. They are often used to define important parameters in the project to prevent the loss of important parameters due to sudden power failure of the PLC or program download. The power failure retention feature is mainly declared through the attribute keyword PERSISTENT RETAIN.

Code example for setting a power failure retention variable:

```
VAR_GLOBAL PERSISTENT RETAIN

iVarPers1 AT %MW100: DINT;

bVarPers AT %MX1.1 : BOOL;

END_VAR
```

The following table lists the response actions of different power failure retention variables after a reset, power failure, or other events.

Table 3-5 Variable Response Actions

Action	VAR	VAR PERSISTENT RETAIN/ VAR RETAIN PERSISTENT	VAR RETAIN
Power failure	Initialization	Retain the original value	Retain the original value
Warm reset	Initialization	Retain the original value	Retain the original value
Cold reset	Initialization	Retain the original value	Initialization
Initial value reset	Initialization	Initialization	Initialization
Program download	Initialization	Retain the original value	Initialization
Online modification	Retain the original value	Retain the original value	Retain the original value

Note:

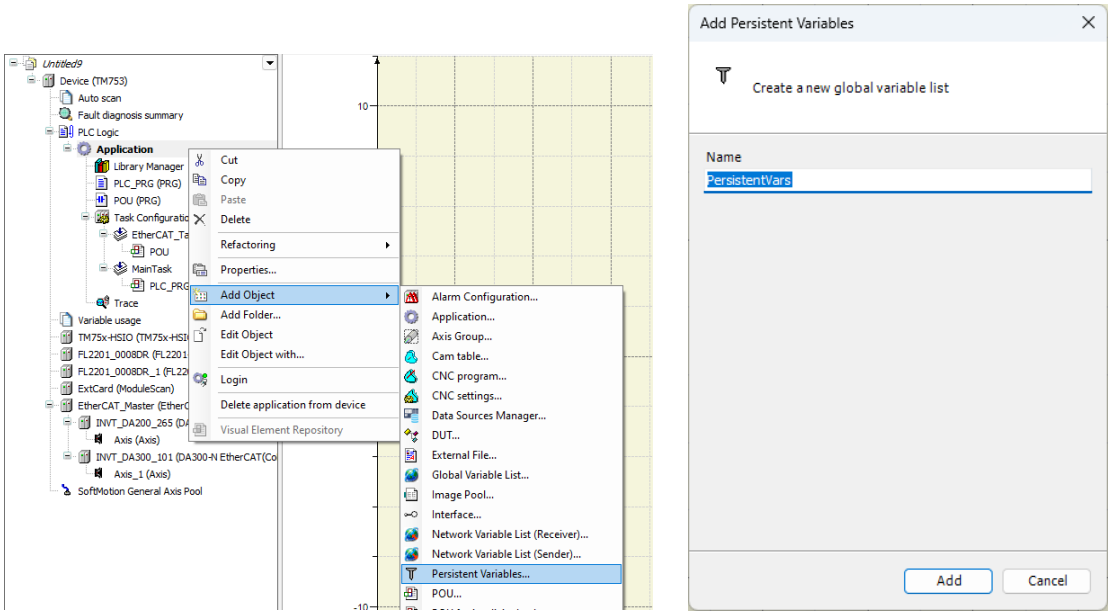
- RETAIN and PERSISTENT RETAIN variables are both persistent variables, but they have different characteristics.
- Direct variables mapped to %M addresses can be declared as persistent variables, while direct variables mapped to %I and %Q addresses cannot be declared as persistent variables.

3.9.2 Power Failure Retention Variable List

If power failure retention variables are defined in the project, a power failure retention variable list must be generated; otherwise, the defined variables will not have the power failure retention function. You can use the power failure retention variable list to add power failure retention variables. The operation steps are as follows:

Step 1 Right-click **Application** in the device tree, select **Add Object > Persistent Variables…**.

Step 2 In the pop-up dialog box, enter the name of the power failure retention variable list and click **Open** to open it.



You can also declare power failure retention variables directly in the power failure retention variable list, as shown in Figure 3-3. However, this method has a disadvantage that it does not support variables defined by direct addresses (such as MX0.0). It is recommended to declare the PERSISTENT RETAIN keyword in the global variable list, and then right-click a blank area on the PersistentVars interface to select **Add all Instance Paths** to generate an instance path in the power failure retention variable list, as shown in Figure 3-4.

Note: Among the PERSISTENT RETAIN variables declared in the global variable list, at least one variable is used in the program before it can be added to the power failure retention variable list.

Figure 3-3 Direct declaration of retention variables in the retention variable list

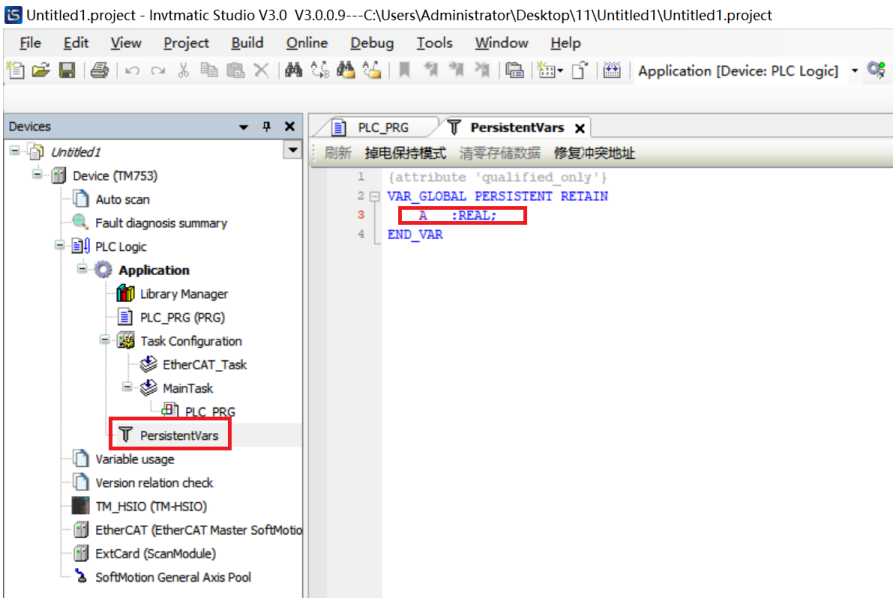
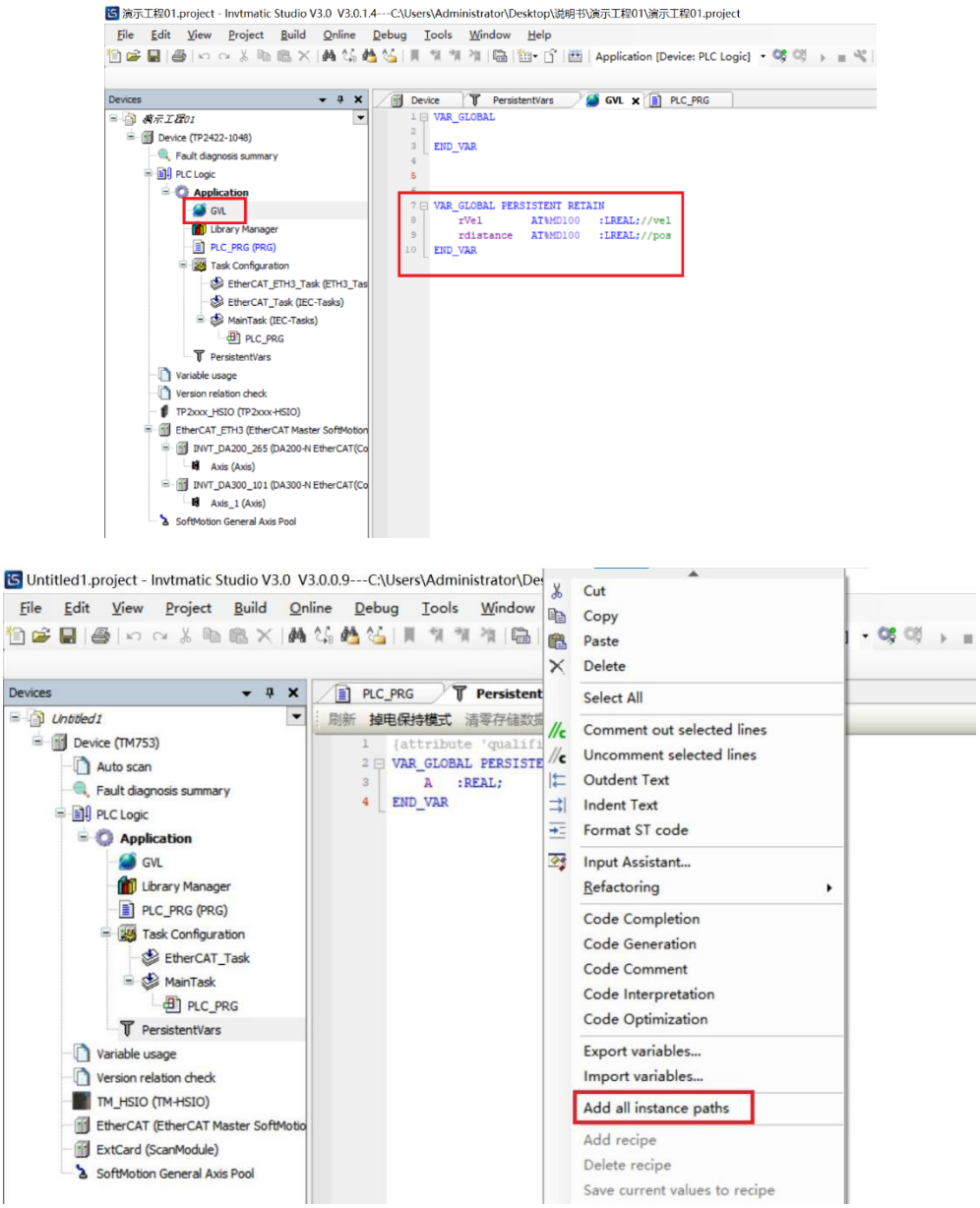


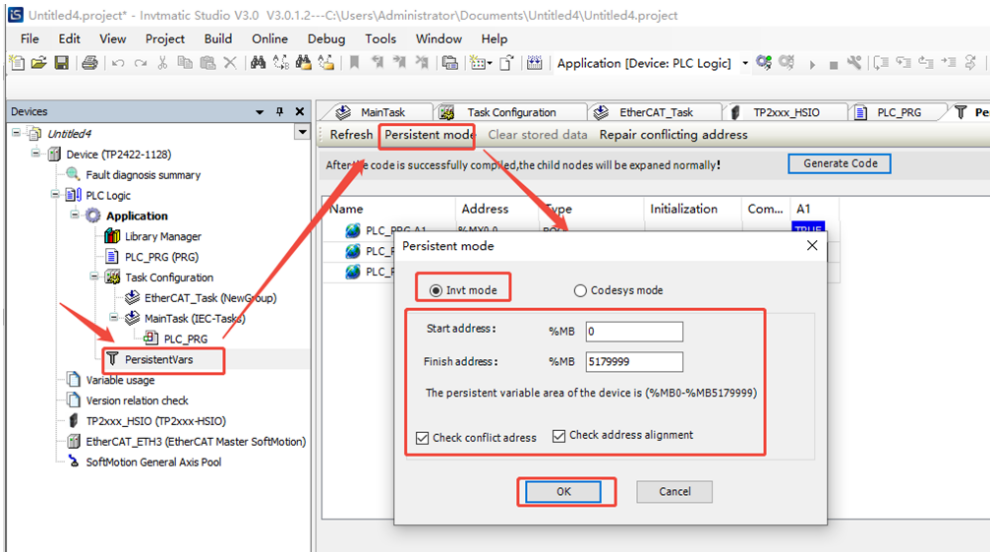
Figure 3-4 Declaration of PERSISTENT RETAIN in the global variable list



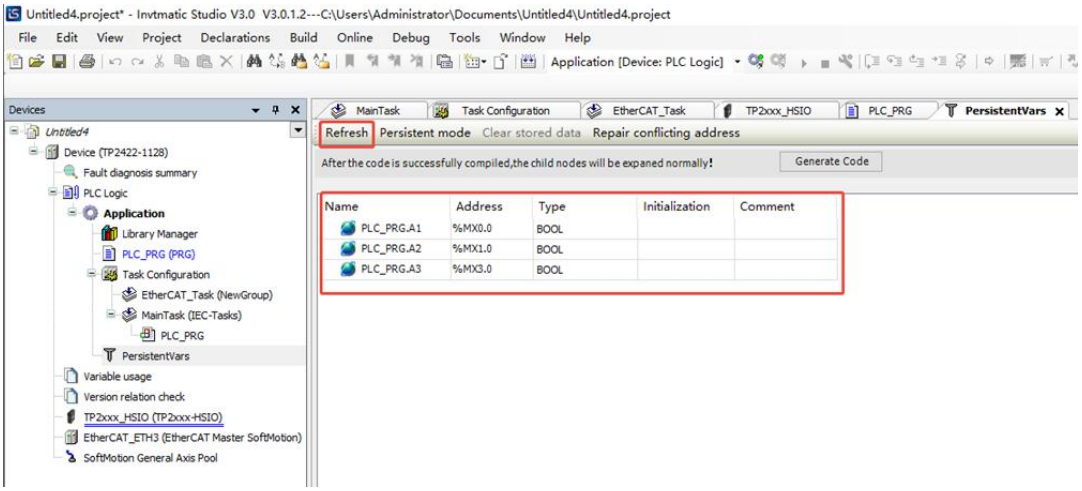
3.9.3 Power-Failure Retention Mode: INVT

Power-Failure Retention: INVT Mode Settings

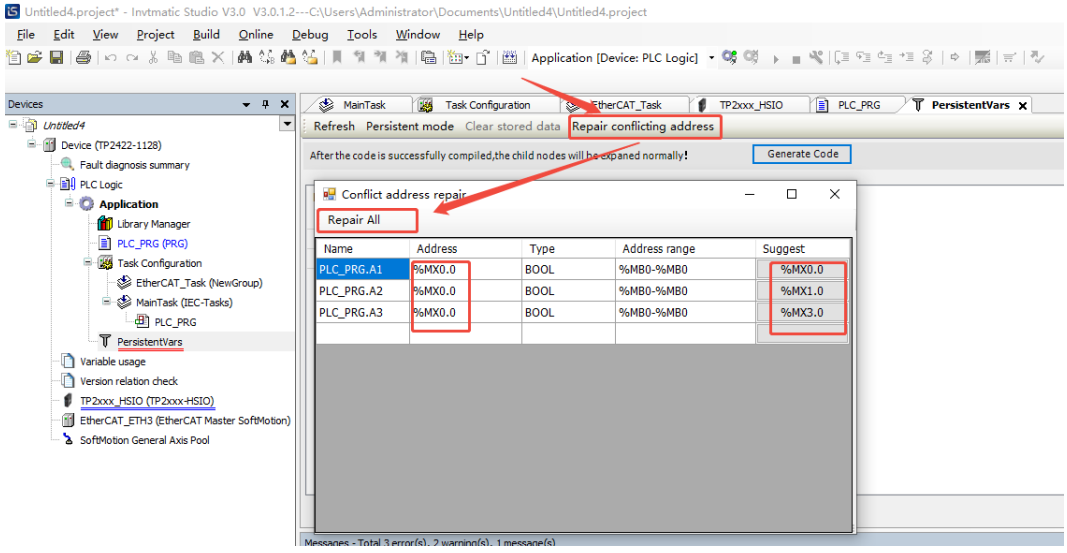
When you select INVT mode, you can dynamically adjust the range of the power-failure retention area. Any variable assigned to an address within this range must be a power-failure retention variable.



After clicking **OK**, any power-failure retention variables declared in other POUs will be automatically added when you refresh or recompile the project.

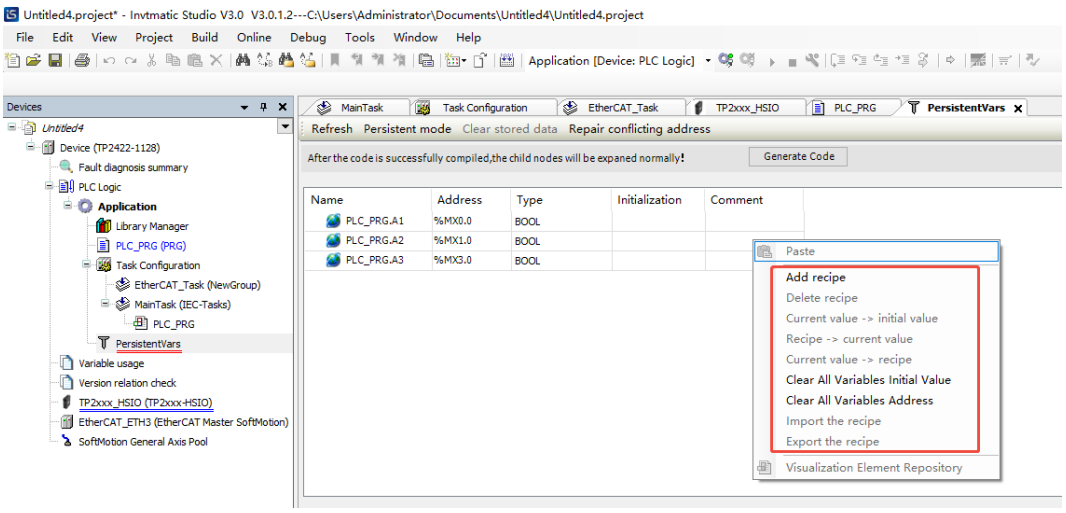


Address Conflict Repair: If retention variable addresses are duplicated, a compilation error will occur. You can use this function to adjust the addresses.

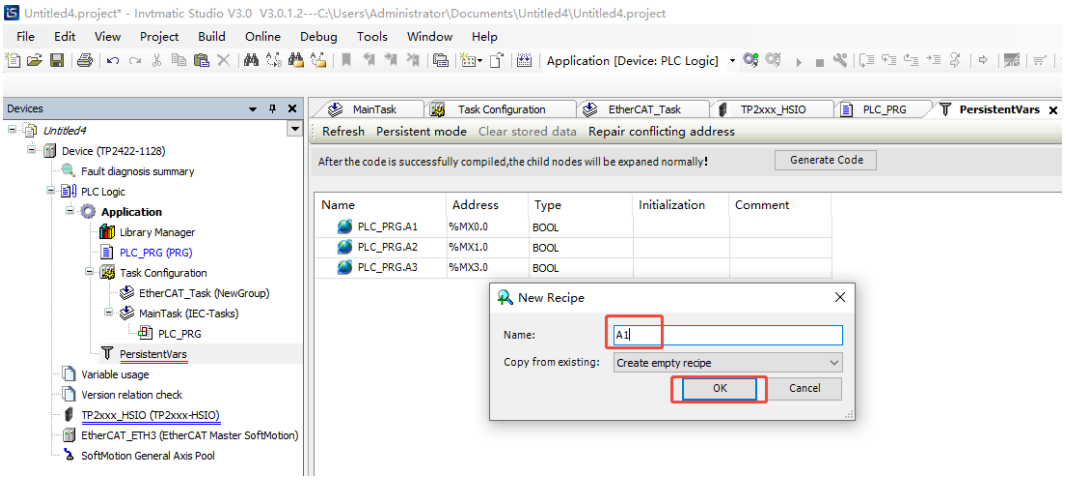


Recipe Integration in Power-Failure Retention INVT Mode

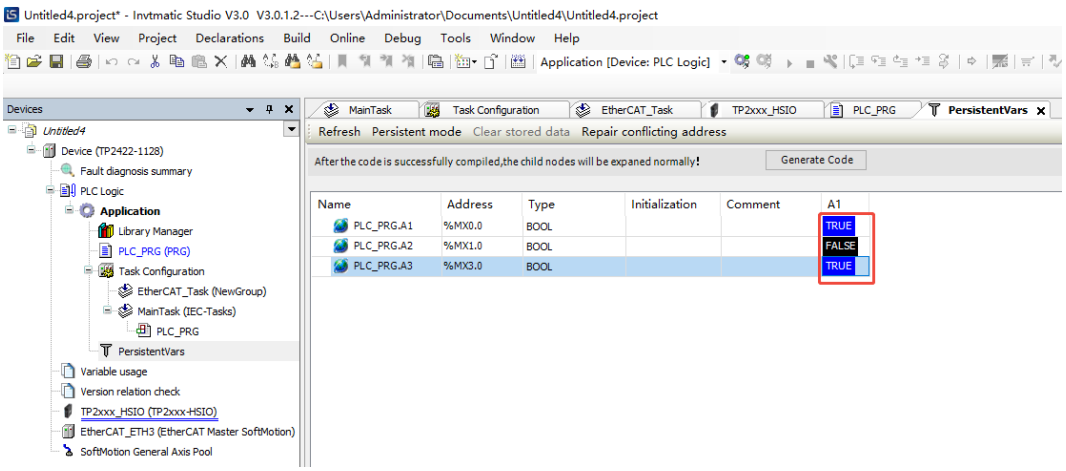
In INVT mode, power-failure retention variables can be directly associated with recipes, eliminating the need to configure them separately in the recipe manager. This feature allows you to quickly create and delete recipes, and perform the following operations: current value -> initial value, recipe -> current value, current value -> recipe, clear initial values, clear all addresses, import the recipe, and export the recipe.



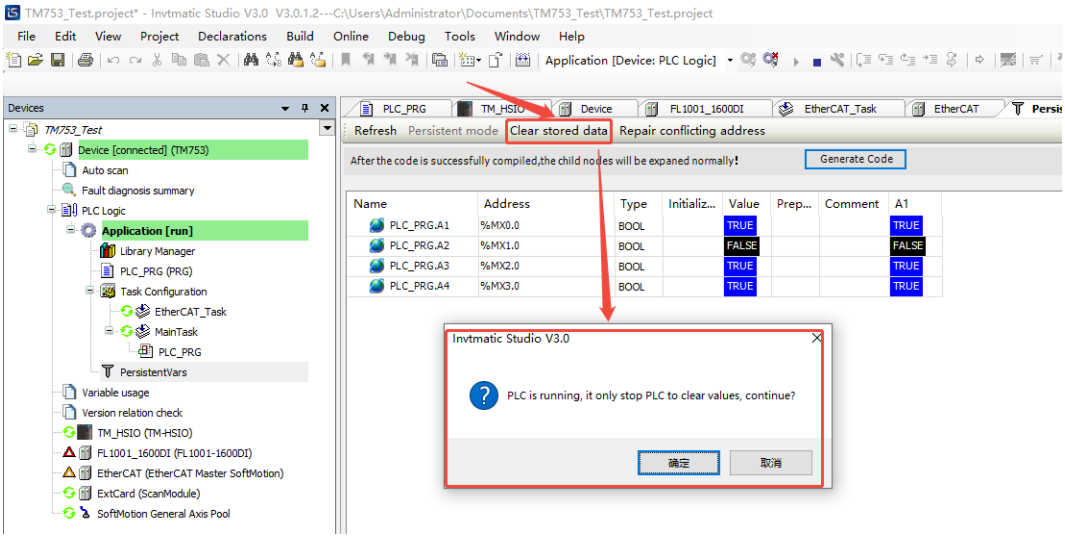
Add a recipe.



Edit the recipe.



Clear the stored power-failure retention data.

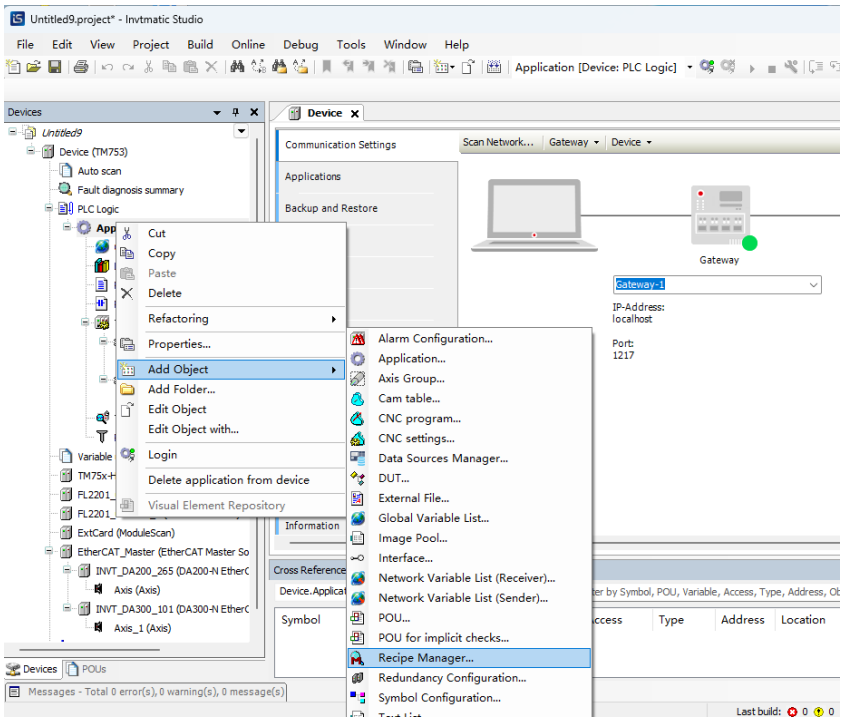


3.10 Recipe Manager

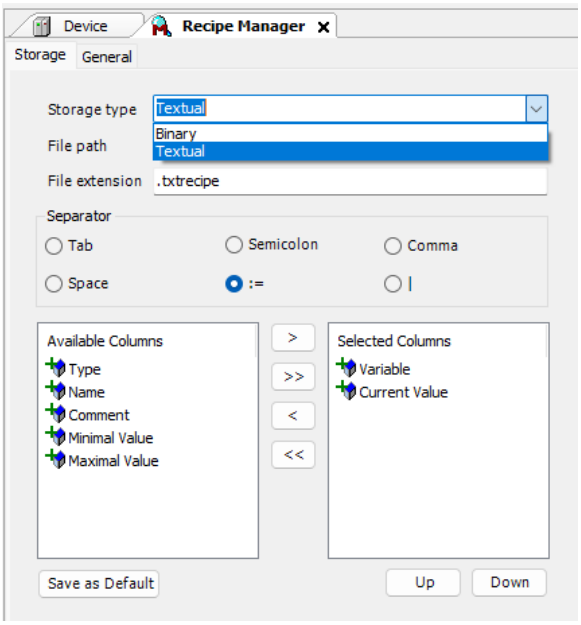
A recipe is a set of parameter values that provides the information required to produce a product and control the production process. For example, the raw materials of biscuits such as sugar, eggs, butter, and flour, baking time, and other parameters. Recipes can also be used to set and monitor PLC control parameters.

In order to save the information required in the production process, you can read recipes from the PLC, write data to it, load recipes from files, or save recipes as files. These interactions can be achieved through the view elements that have been set up.

Step 1 Right-click **Application** in the device tree, and select **Add Object** > **Recipe Manager** to add a recipe manager, as shown in the figure below.



Step 2 After adding a recipe manager, you need to set the manager name. Click **Open** to show the configuration interface of the recipe manager in the figure below.

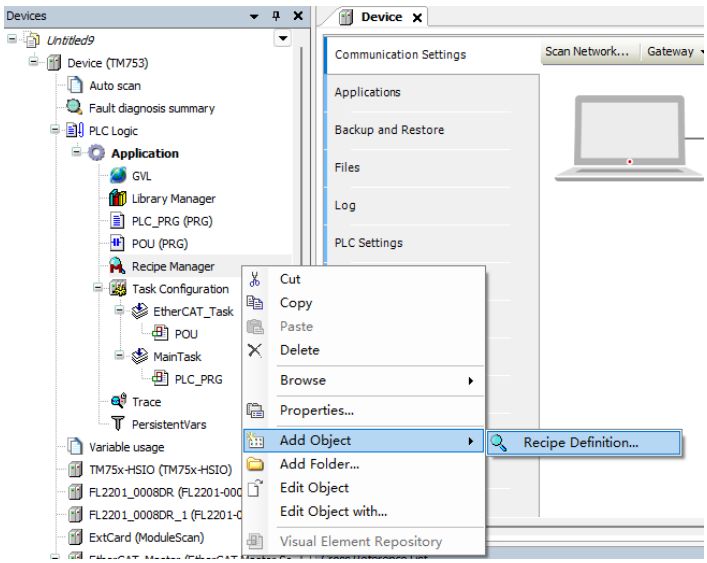


Storage Type: You can select Textual or Binary storage type.

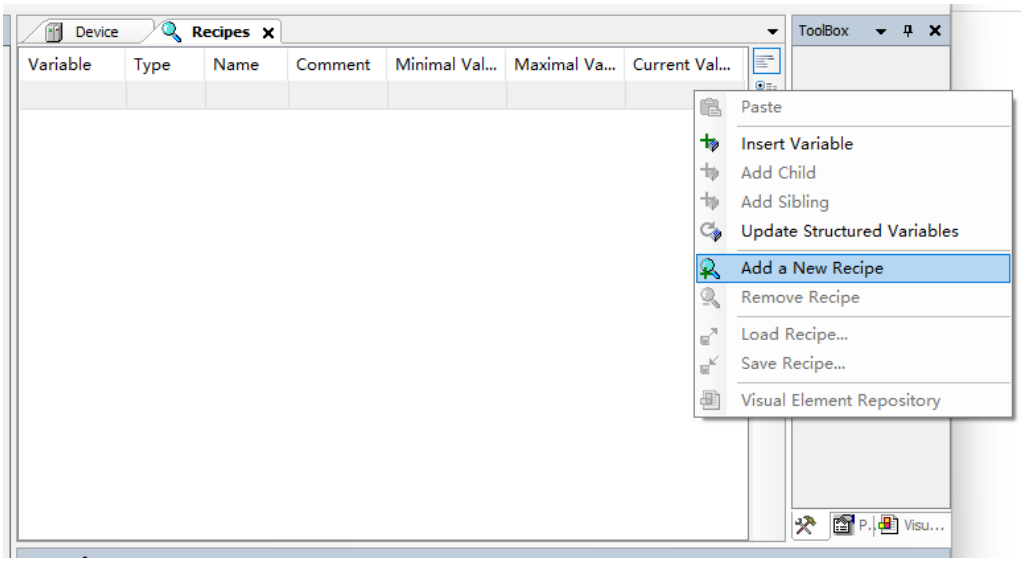
File Path: You can specify the storage location through the file path, or define the file extension for storing the recipe; the storage of text will be separated according to the selected separator, and the separator is only effective when the Textual storage type is enabled.

Available Columns: The right side contains the columns defined for the current recipe, i.e. **Selected Columns**, and the content of the selected columns will be stored. You can click the icon button > or < to shift between **Available Columns** and **Selected Columns**. You can also shift all entries from one side to the other at once by clicking the icon button >> or <<. The icon buttons Up and Down can be used to adjust the order of the selected columns, which represents the order of the columns in the storage file.

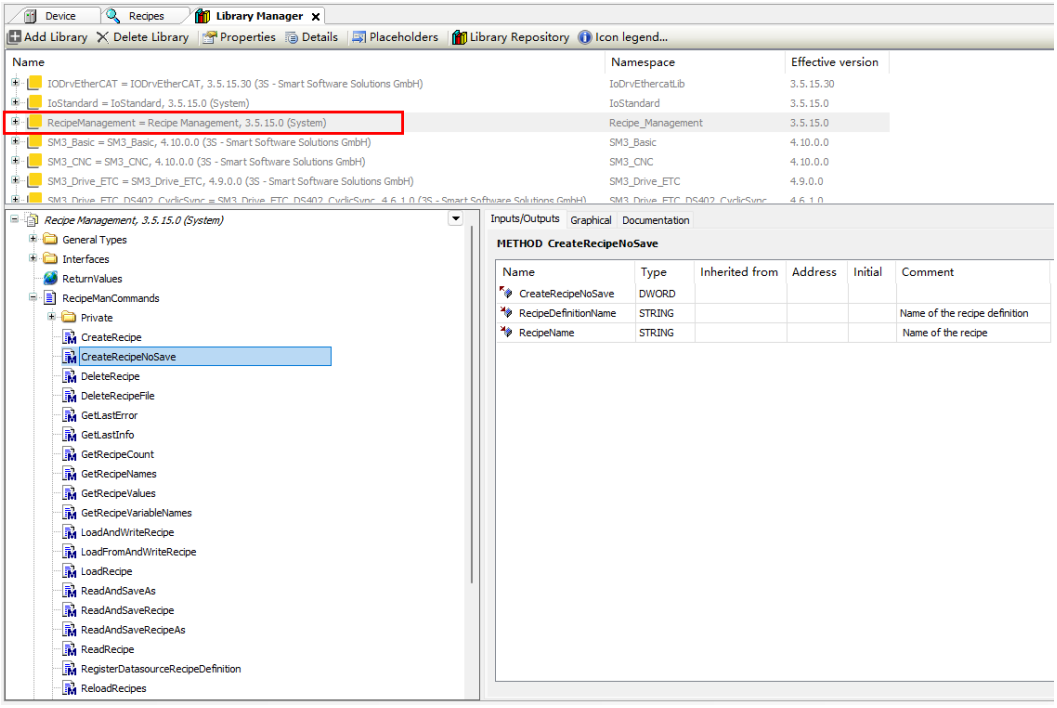
Step 3 After the Configuration page is set up, right-click **Recipe Manager** in the device tree and select **Add Object** > **Recipe Definition…** to add a new recipe definition, as shown in the following figure.



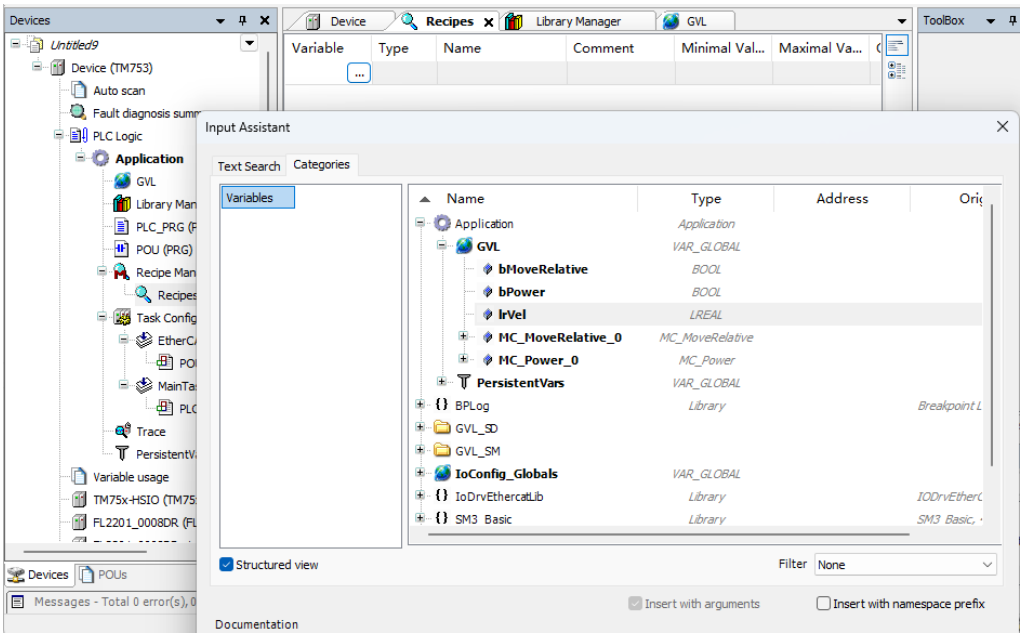
Step 4 Set the name of the new recipe and click Open to show the added recipe in the figure below. You can right-click a blank area and select **Add a New Recipe** to configure different recipe values. You can also select a recipe value (such as recipe 1) and right-click it to delete, load, or save the recipe, as shown in the following figure.



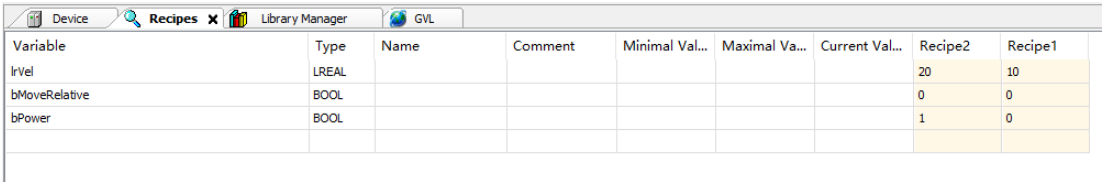
In addition to editing recipe files directly on the interface, you can also add “Recipe Management” library files and use code to create, delete, load, and save recipes.



Step 5 After adding a recipe, you need to define the variables that the recipe needs to use in the program, and then insert the variables into the recipe and click ... .

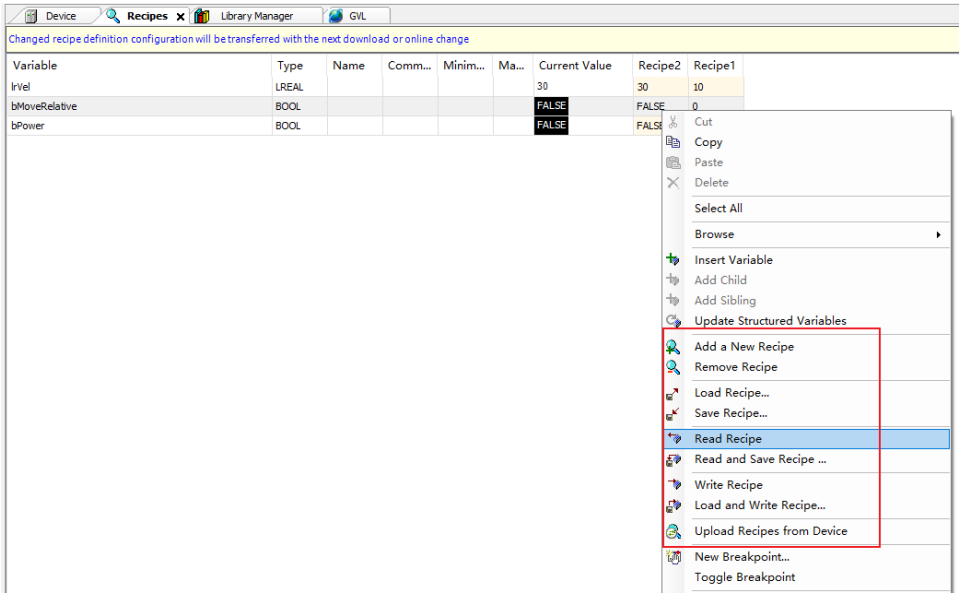


After the variables are inserted, you can fill in the parameters in Recipe 1 and Recipe 2, as shown in the following figure.



Variable	Type	Name	Comment	Minimal Val...	Maximal Va...	Current Val...	Recipe2	Recipe1
lrVel	LREAL						20	10
bMoveRelative	BOOL						0	0
bPower	BOOL						1	0

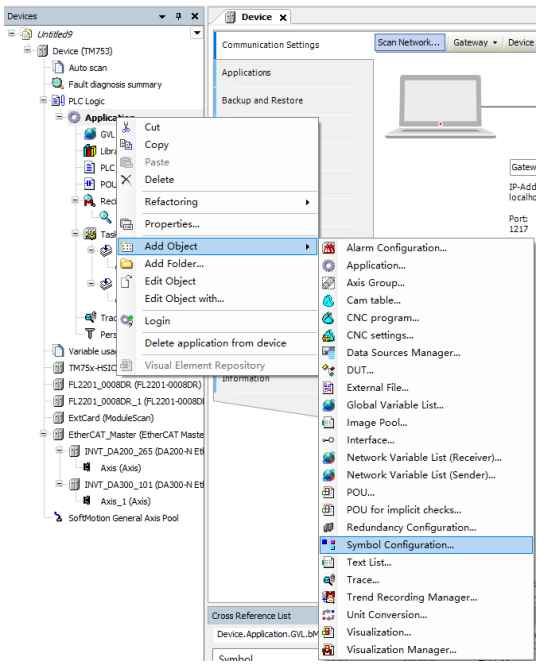
After login online, you can click **Write Recipe** to write the currently selected recipe value to the controller, or click **Read Recipe** to read the current value from the controller into the recipe, as shown in the following figure.



3.11 Symbol Configuration

The symbol configuration feature is used to configure project variables as symbols that require special access. With these symbols, variables can be accessed by external applications, such as an OPC server. If you select **Generate Code** in the **Compile** menu bar, a symbol configuration file (with the suffix *.xml in the project directory) is generated. The naming method is as follows: device name + application name.xml, which contains the description of the symbol. For example, import the XML into the HMI for label communication and access to variables.

Right-click **Application** in the device tree, and select **Add Object** > **Symbol Configuration**.



After selecting symbol configuration, a pop-up window appears as shown below.

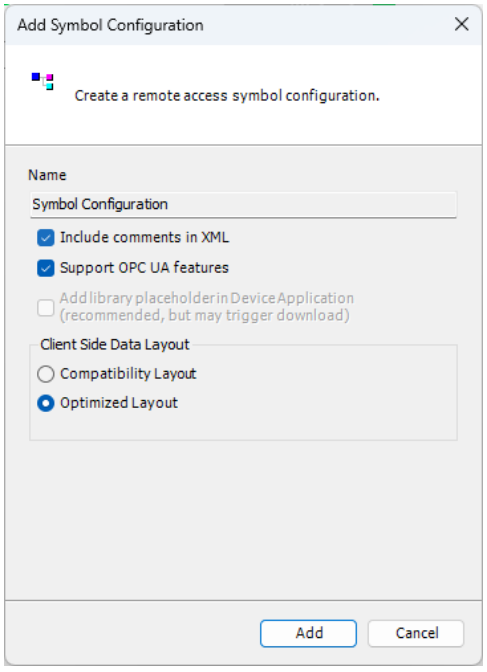
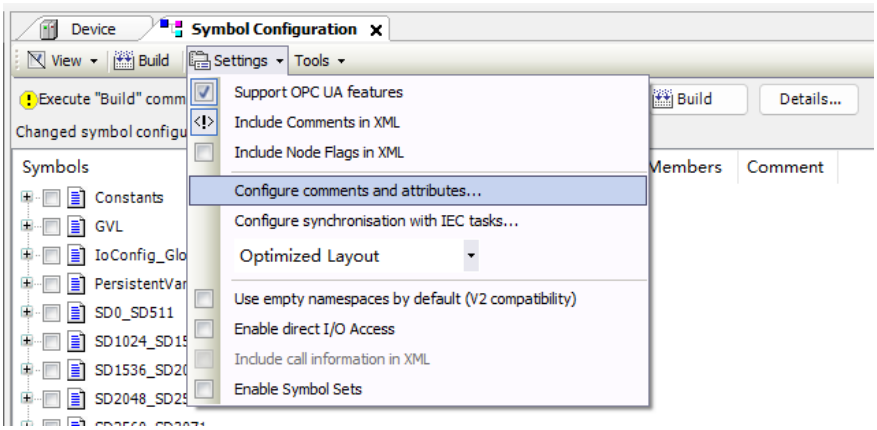


Table 3-6 Description of Symbol Configuration

Option	Description
Include comments in XML	The exported XML contains comments assigned to variables
Support OPC UA features	The symbol variables can be accessed by OPC UA
Compatibility layout	The offset size is consistent with the type member definition. If the type member does not fully support symbol access, the offset size is the actual compilation offset, and there will be a blank in the middle
Optimized layout	The offset is calculated based on the selected type member, and no offset calculation will be performed if the type member is not selected.

■ **Symbol configuration-Settings**

After generation, the Symbol Configuration Settings interface is as shown in the figure below.



The configuration comments and attributes are as shown in the figure below. The upper left side shows the data downloaded to the PLC, and the upper right side shows the exported XML data format.

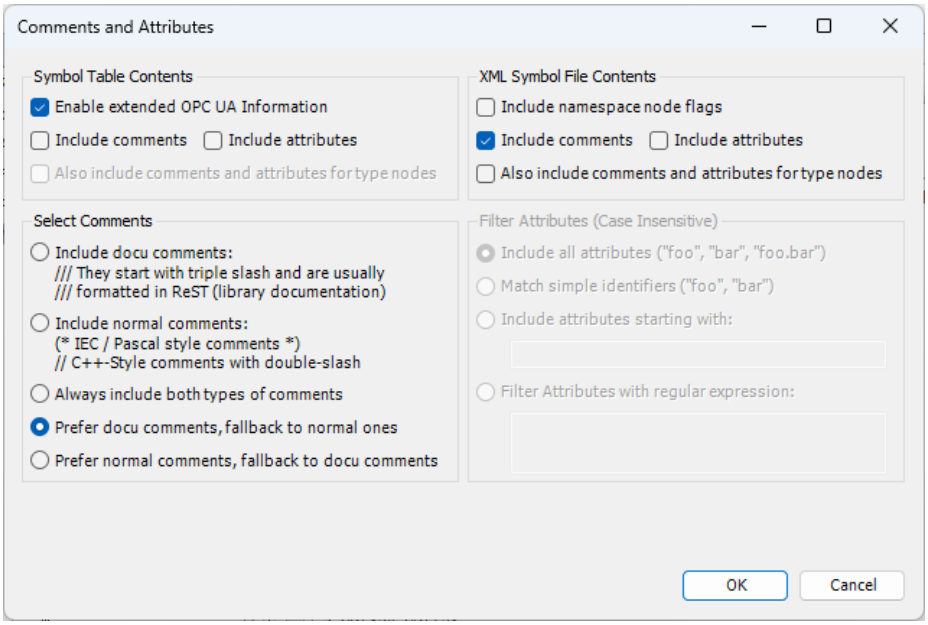


Table 3-7 Description of Symbol Configuration Comments and Attributes

Option	Description
Symbol	Generally, symbols represent variables, and symbol attributes represent variable characteristics (Attribute information).
Comment format	It indicates the comment download or display format.
Attribute matching	It indicates which attribute information is included when the XML file is exported or downloaded to the PLC.

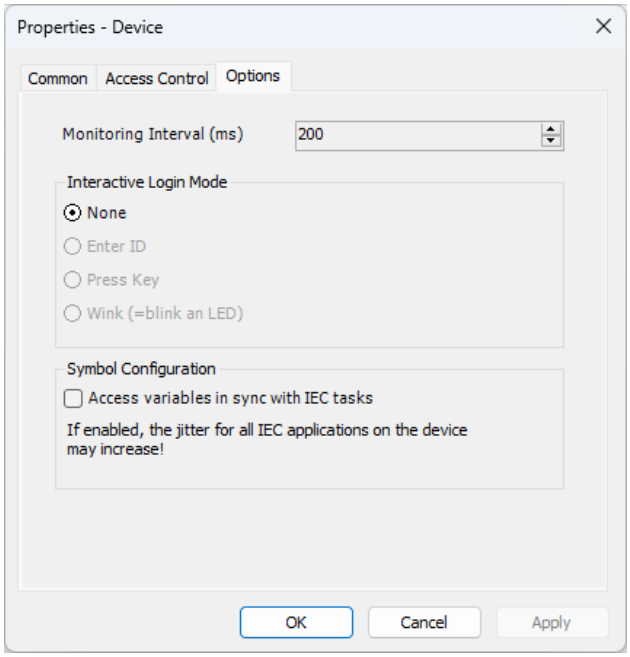
Symbol: Generally, symbols represent variables, and symbol attributes represent variable characteristics (Attribute information).

Comment format: It indicates the comment download or display format.

Attribute matching: It indicates which attribute information is included when the XML file is exported or downloaded to PLC.

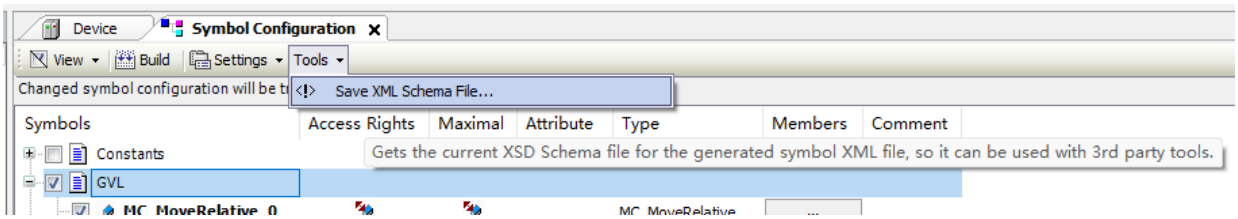
Four matching rules: Include all attributes, Match simple identifiers, Include attributes starting with:, and Filter Attributes with regular expression.

Configure synchronization with IEC tasks: In the **Device Properties** options, select whether to synchronize with IEC tasks when other interfaces access symbol variables. Access cannot be made during IEC execution to prevent variables from being out of sync.



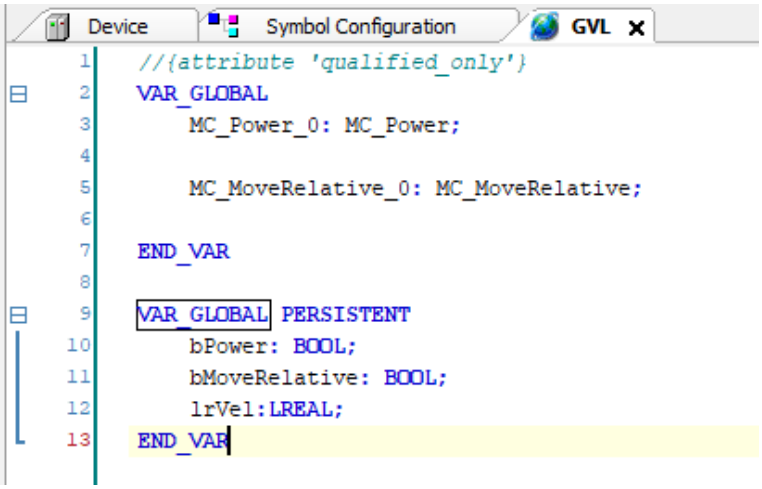
■ **Symbol Configuration-Tools**

You can use it to save the file in XML format to export an XML data model, which can be used as a reference if a third party parses symbols offline



■ **Symbol Configuration Process Example**

Create new global variables bLatch and iMasterYout, and apply at least one of them in the user program, as shown in the figure below.

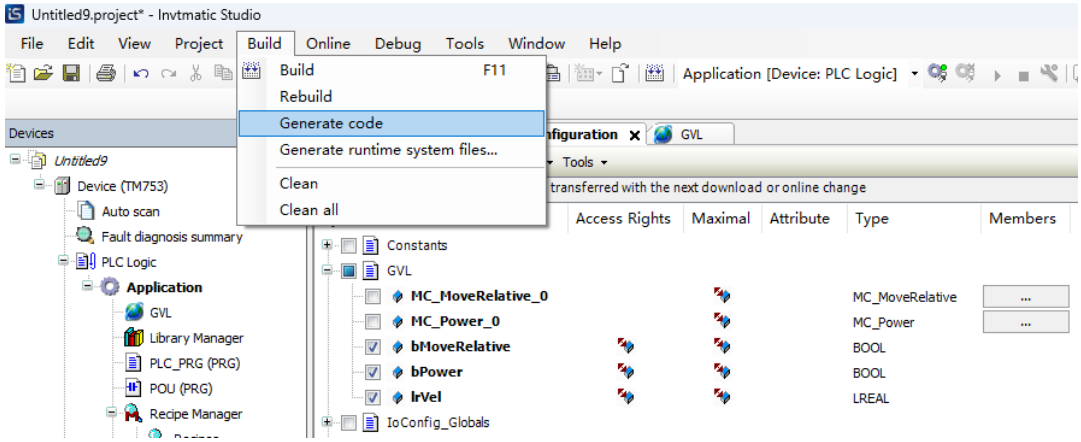


Note:

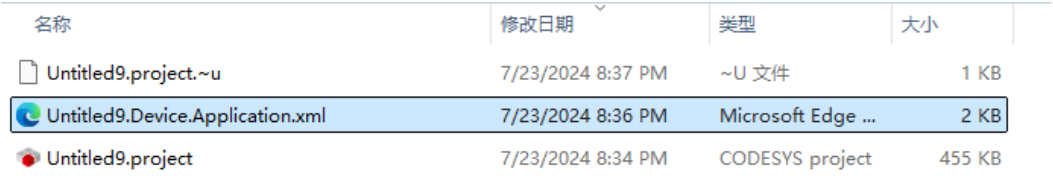
- If any variable in a single global variable list is not used in the user program, the corresponding variable list will not appear in symbol configuration unless one of the global variables is called within a PRG task.
- If any declared global variables contain Chinese variables, you need to go to **Project Settings**→ **Compile options**, and enable **UTF8 Encoding for STRING**.

The steps to configure the sample are as follows:

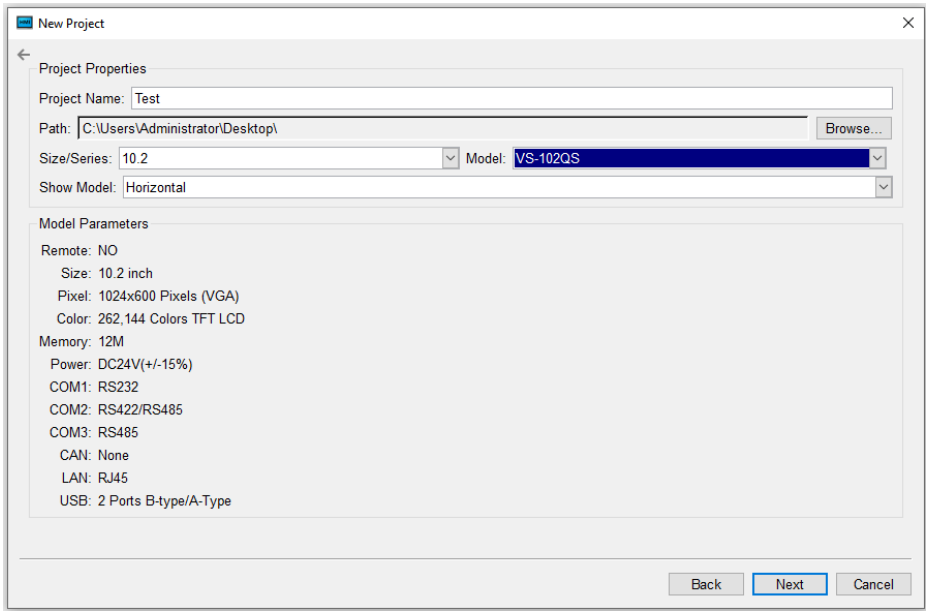
Step 1 After adding **Symbol Configuration** in **Application**, check **Include comments in XML**, and click **Build** > **Build** in the upper Toolbar. At this time, the corresponding variable list and variables appear in the right symbol configuration. Check the variable list to be configured, configure the corresponding access rights (read-only, write-only, read/write), and select **Build** > **Generate code** in the upper Toolbar.

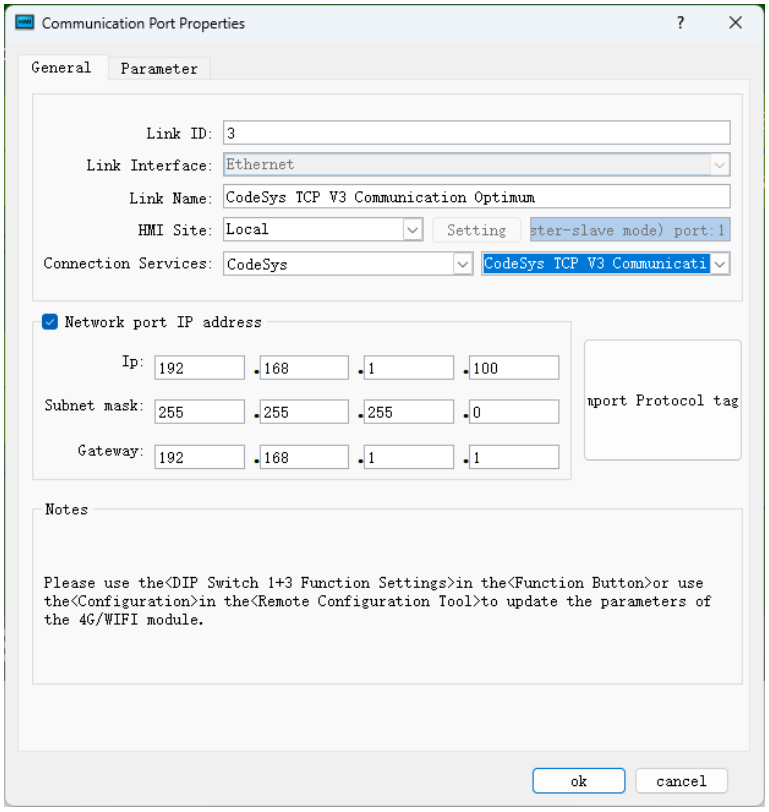


Step 2 You can find the generated file with the suffix .xml in the directory folder where the project is located, which is imported into the touch screen for label communication, as shown in the figure below.

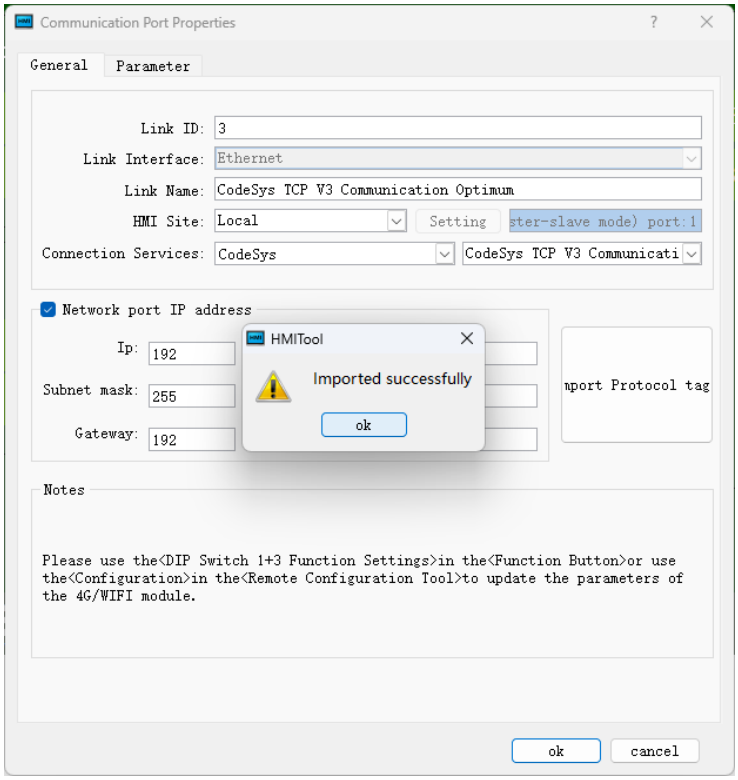


Step 3 After completing the above steps on the PLC, taking VS-102QS touch screen as an example, open the HMITOOL software, click **New Project**, Select "VS-102QS" as the **Model,** "Ethernet Port" as the **Connection Port**, and "CodeSys" and "CodeSys TCP V3 Communication Optimum" as the **Connectivity Service**, as shown in the figure below.

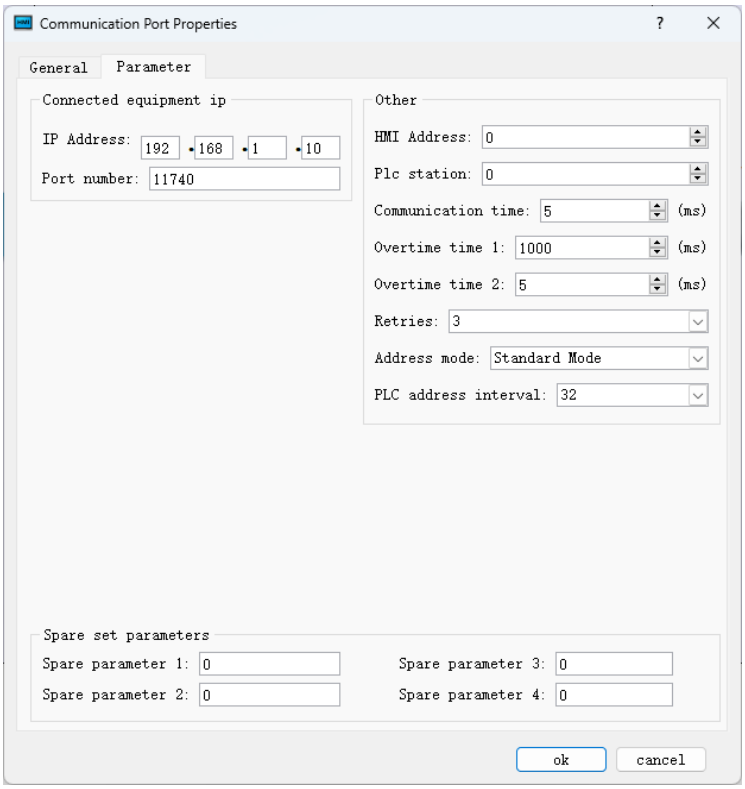




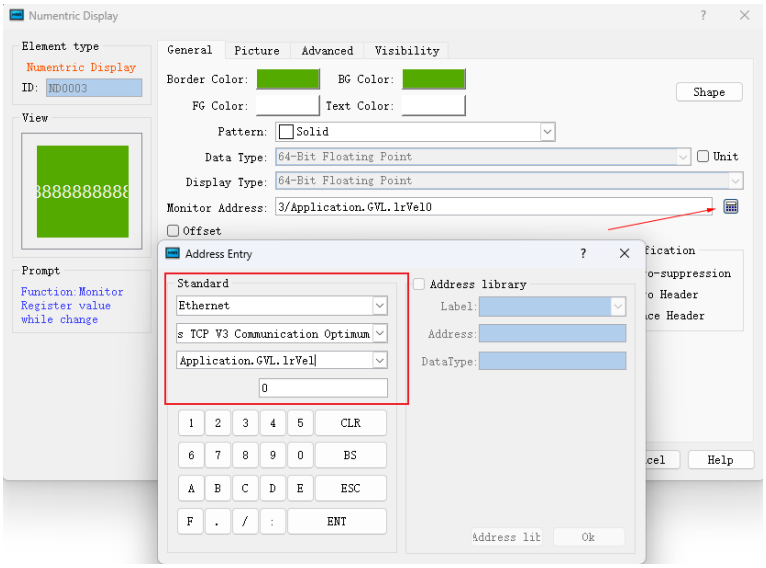
Step 4 Click **Import Communication Labels**, select the XML label just generated, and click **Open**. If **Imported successfully** is displayed, click **OK**, as shown in the figure below.



Step 5 Set the **IP address** of communication parameters to “192.168.1.10” and **Port number** to “11740”, and then click **OK**, as shown in the figure below.



Step 6 Select **Input Box** in the figure above, and bind the corresponding PLC address, as shown in the figure below. After completion, label communication can be implemented.

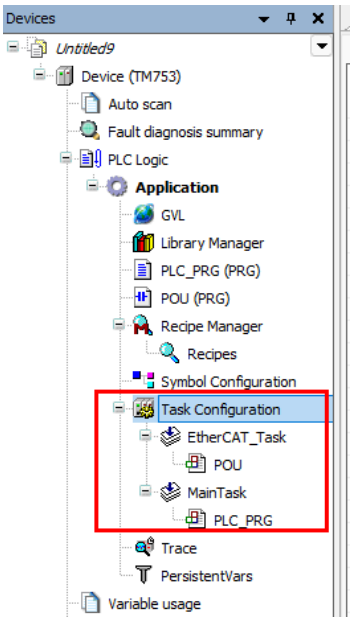


3.12 Task Configuration

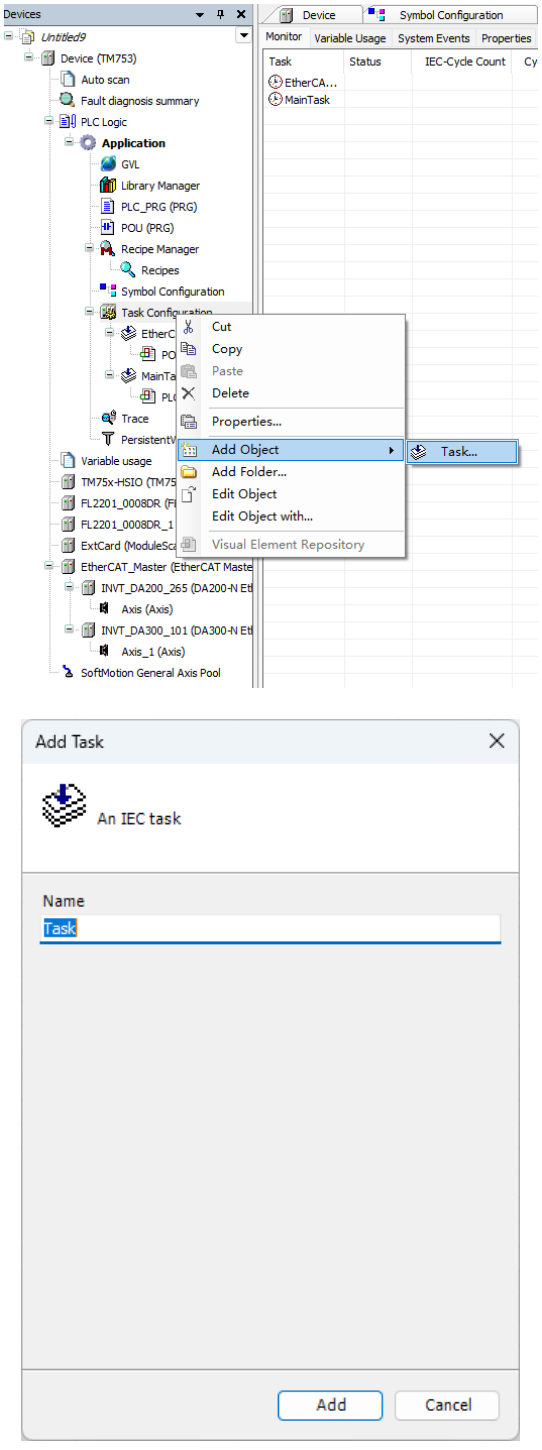
Each PLC application (project) contains “Task Configuration” in the project directory tree. In Task Configuration, one or more tasks can be defined to control and execute the application in the controller. After a “task” is configured, a series of programs or function blocks can be executed cyclically or triggered by a specific event to start executing the program. A task can call one or more program blocks (POUs). By setting priorities and conditions for tasks, you can define the order in which the tasks are processed. You can also configure a watchdog for each task, and the controller will prompt an exception when the execution cycle of the task is too long.

3.12.1 Adding a Task

After creating a new project, Invtmatic Studio will automatically add **Task Configuration**, the automatically generated tasks are shown in the figure below.

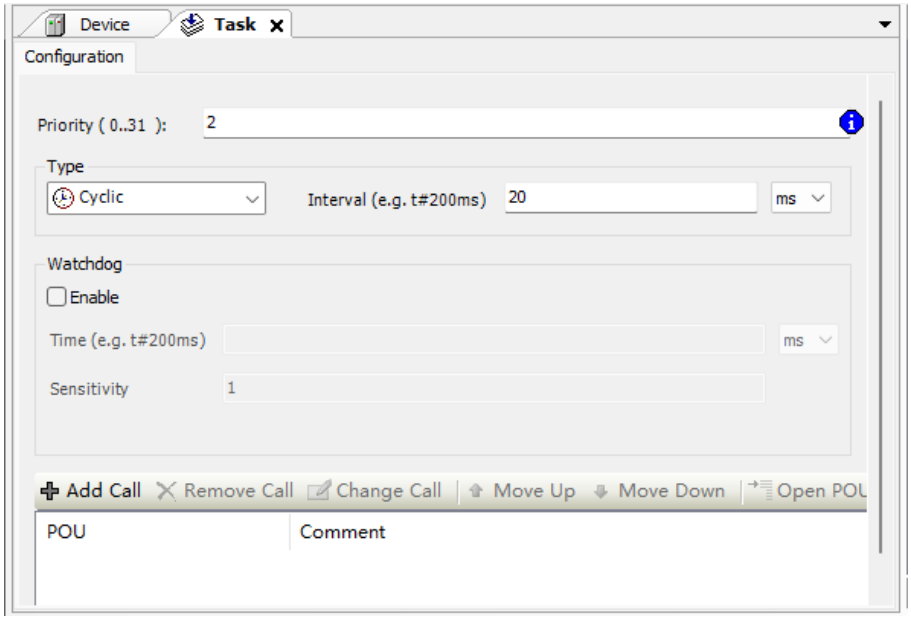


If you need to add a new task, you can right-click **Task Configuration** and select **Add Object** > **Task**. After entering a custom task name in the pop-up window, click **Open** to add the task, as shown in the figure below.



3.12.2 Task Settings

After opening the newly created task, the Settings interface shown in the figure below appears.



- Priority: 0–31. The larger the value, the lower the priority, and 0 indicates the highest priority.
- Type: Task type and its execution logic. See the table below for details.

Table 3-8 Task Type

Type	Execution Logic	Variable
Cyclic	The task is executed cyclically according to the set interval	Time interval
Event	The task starts execution at the rising edge of the set global variable associated to the trigger event	Trigger variable
Freewheeling	The task automatically starts execution at the beginning of the program and at the end of the complete process in a continuous cycle	-
State	When the set global variable is TRUE, the task starts execution	Trigger variable

- Watchdog: If a task exceeds the currently set time of the watchdog, the task will be suspended in an error state (exception). The application in which the task with the error occurs and its child applications will also be suspended. All tasks of the affected application will also be stopped. The watchdog-related settings are detailed in the table below.

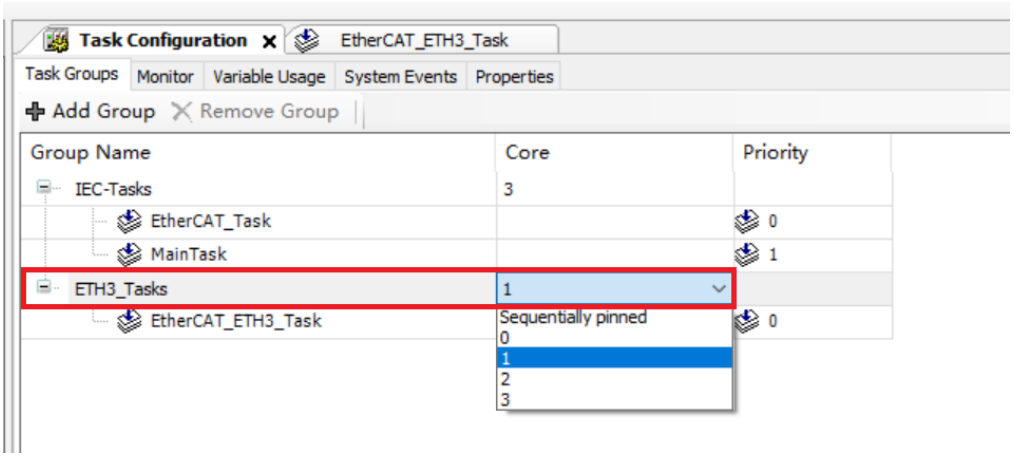
Table 3-9 Watchdog settings

Option	Function
Enable	It is used to enable the watchdog function
Time	It is used to set the watchdog timeout period
Sensitivity	It is used to set the value, that is, trigger the watchdog after exceeding the set time by several times

- POU call: You can set the POU called by the task, and the order of POU calls will be affected.

3.12.3 TP Series Multi-core Task Configuration

The TP2000 series PLC supports dual-core configuration. In the **Task Configuration** interface, there are 2 task groups by default. You can add task groups as needed, which can be assigned to use Core 0 or Core 1. You can assign a priority to each task. Generally, it is required that the ETH3_Task containing EtherCAT_Task be configured as Core 1 for better real-time performance.

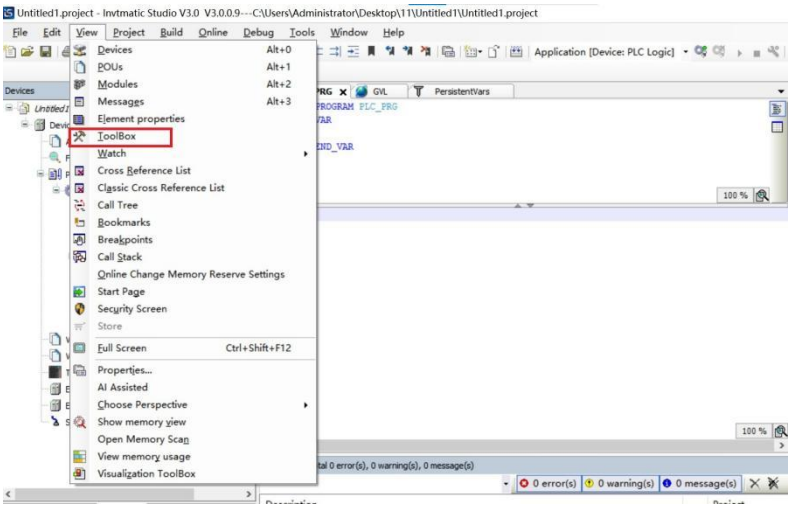


Note: The TP2321 is a dual-core model where EtherCAT is bound to Core 1, while all other tasks run on Core 0. The TP2422 is a quad-core model where EtherCAT is bound to Cores 2 and 3, with other tasks running on Cores 0 and 1.

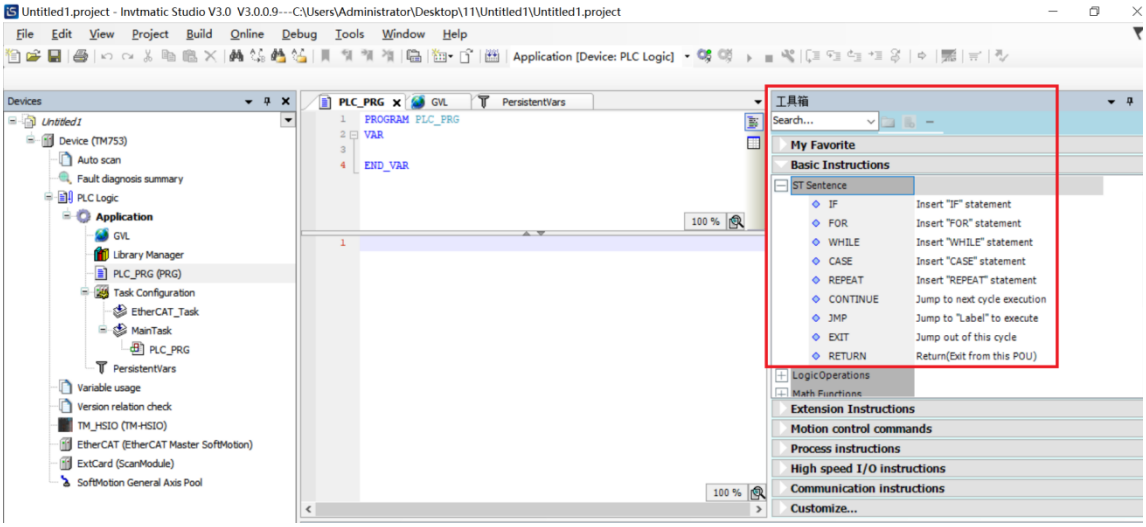
3.13 Toolbox

The toolbox contains basic instructions, extended instructions, motion control instructions, and more. Use the toolbox as follows:

Select **View** > **ToolBox**.



The toolbox contents are displayed on the right side of the interface.


Untitled1.project - Invtmatic Studio V3.0 V3.0.0.9---C:\Users\Administrator\Desktop\11\Untitled1\Untitled1.project
File Edit View Project Build Online Debug Tools Window Help
Application [Device: PLC Logic]
Devices
Untitled1
Device (TM753)
Auto scan
Fault diagnosis summary
PLC Logic
Application
GVL
Library Manager
PLC_PRG (PRG)
Task Configuration
EtherCAT_Task
MainTask
PLC_PRG
PersistentVars
Variable usage
Version relation check
TM_HSIO (TM-HSIO)
EtherCAT (EtherCAT Master SoftMotion)
ExtCard (ScanModule)
SoftMotion General Axis Pool
PLC_PRG
GVL
PersistentVars
1 PROGRAM PLC_PRG
2 VAR
3
4 END_VAR
100 %
1
100 %
工具箱
Search...
My Favorite
Basic Instructions
ST Sentence
IF Insert "IF" statement
FOR Insert "FOR" statement
WHILE Insert "WHILE" statement
CASE Insert "CASE" statement
REPEAT Insert "REPEAT" statement
CONTINUE Jump to next cycle execution
JMP Jump to "Label" to execute
EXIT Jump out of this cycle
RETURN Return(Exit from this POU)
LogicOperations
Math Functions
Extension Instructions
Motion control commands
Process instructions
High speed I/O instructions
Communication instructions
Customize...

4 Hardware Configuration

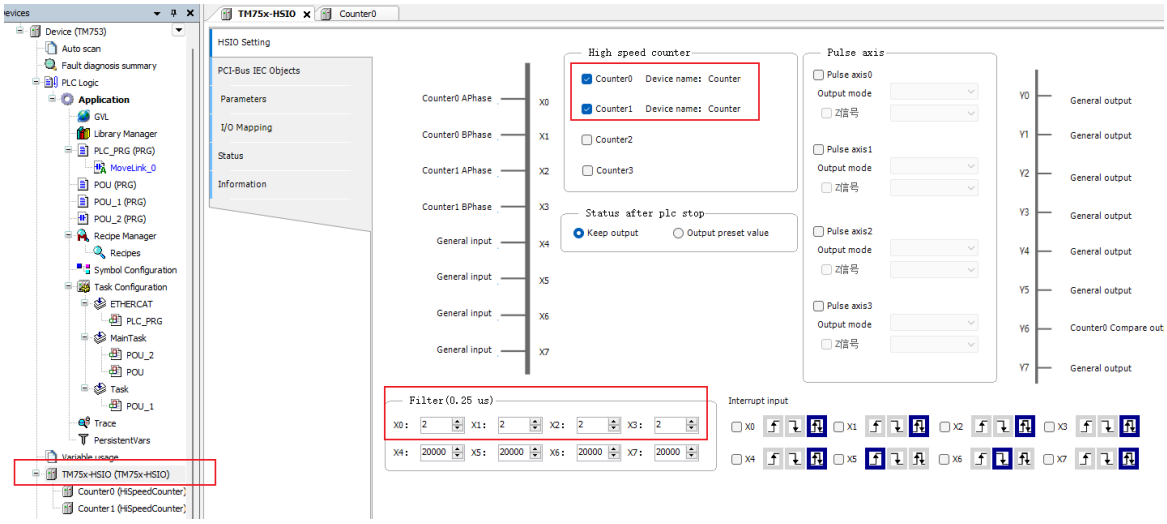
4.1 High-speed I/O Configuration

For the TP2000 (TM700) series PLC, double-click **TP2xxx_HSIO (TM75x-HSIO)** in the device window. Then, the HSIO parameter configuration interface will pop up. You can configure high-speed I/O functions and associated parameters on this interface, including high-speed counter, high-speed output, and high-speed input edge interrupt.

Note:

1. If **Counter** is not checked, it can be used as a normal input port.
2. To use the counter function, check **Counter 0 (1, 2, 3)**. Then, the corresponding input port is used as the input signal source of the counter. X0 and X1 are the input signal sources of counter 0 by default. The corresponding relationship between subsequent counters and high-speed input ports is similar. Once the counter function is enabled, a Counter device is generated and you can make detailed settings.

Figure 4-1 High-speed I/O Port Configuration Interface

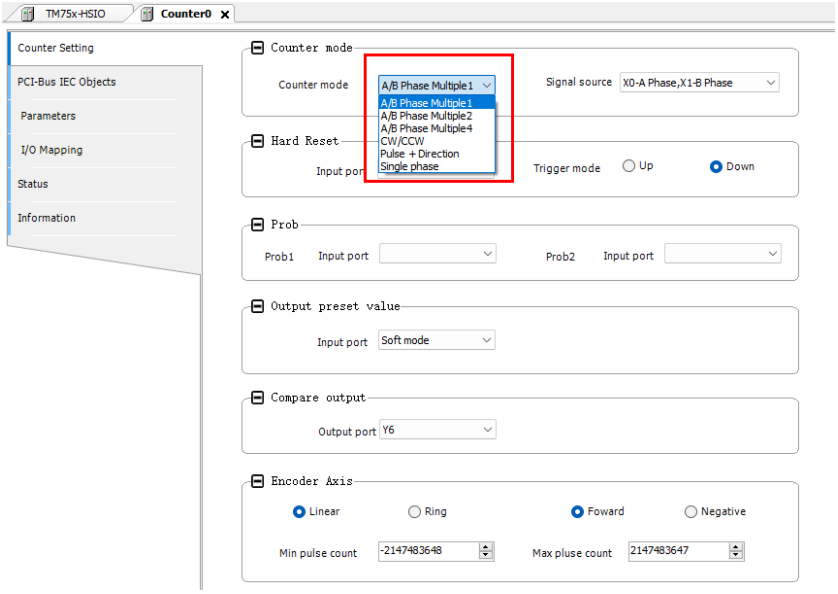


4.1.1 Counter Interface Configuration

You can set 5 functions for the input port: normal input, counter, trigger latch, Z signal, and pulse width measurement. The counter function module can count and calculate the input pulses, detect position, speed, frequency, etc. The maximum frequency of the input pulse is 200 kHz.

You can double-click **Counter** under **TP2xxx_HSIO** to make detailed settings for the high-speed counter.

Figure 4-2 High-speed Counter Parameter Configuration



- Counter mode: It indicates the counter mode for high-speed counters, which can be selected from the following 6 options: phase A/B 1x, phase A/B 2x, phase A/B 4x, pulse+direction, single phase, and CW/CCW. If you choose to use phase A/B 1x/2x/4x, pulse+direction, or CW/CCW, X0 and X1 are the default input signal sources of counter 0, and the corresponding relationship between subsequent counters and high-speed input ports is similar. The signal source can also be set as needed; if single phase is selected, the input signal source of counter 0 can be selected from X0 to X7. The configuration of differential counters is identical to that of high-speed counters.
- Differential counter: Supports four counting modes: phase A/B 1x/2x/4x and CW/CCW. Filter parameters can be used to filter noise from input signals.
- Input counting direction: Positive or negative.
- Counting mode: Rotary or linear. Note that the modes use different library versions in **Library Manager**.
- Max. number of pulses: 2147483647 by default, ranging from -2147483648 to 2147483647.

Note: The upper limit value cannot be less than the lower limit value. If the set value is less than the lower limit value, the system will automatically change the lower limit value to be consistent with the upper limit value.

- Min. number of pulses: -2147483648 by default, ranging from -2147483648 to 2147483647.
- Preset input terminal: When the preset function is configured as external DI preset mode, this input port needs to be configured. You can select any input port from IN0 to IN7 to implement the preset value function, or you can set it as a software-triggered preset input.
- Probe: Each counter can be configured with 2 probe input ports to realize the function of latching the counter value. Probe 1 is #1 probe input port and can be configured as any input port from IN0 to IN7. Probe 2 is #2 probe input port and can be configured as any input port from IN0 to IN7.
- Output comparison terminal: When using the high-speed output comparison (single point/linear/queue) function, configure this output port in the range of OUT0–OUT7.
- Hardware reset: When the count value reset function is configured as external DI mode, you need to configure this input port. You can select any input port from IN0 to IN7 to realize the count value clearing function.

4.1.2 Counting Functions

The INVT HSIO counting control is primarily implemented through the HSIO counting library. You can achieve specific functions by calling the relevant function block instructions.

The **Library Manager** will automatically call the corresponding pulse library instructions based on the HSIO parameter configuration interface.

Pulse count values are primarily represented in the form of an encoder axis. You can directly refer to the CiA402 axis application and use the relevant properties of the encoder axis structure.

For detailed usage of each instruction, refer to the *INVT Medium- and Large-Scale PLC Programming Manual*.

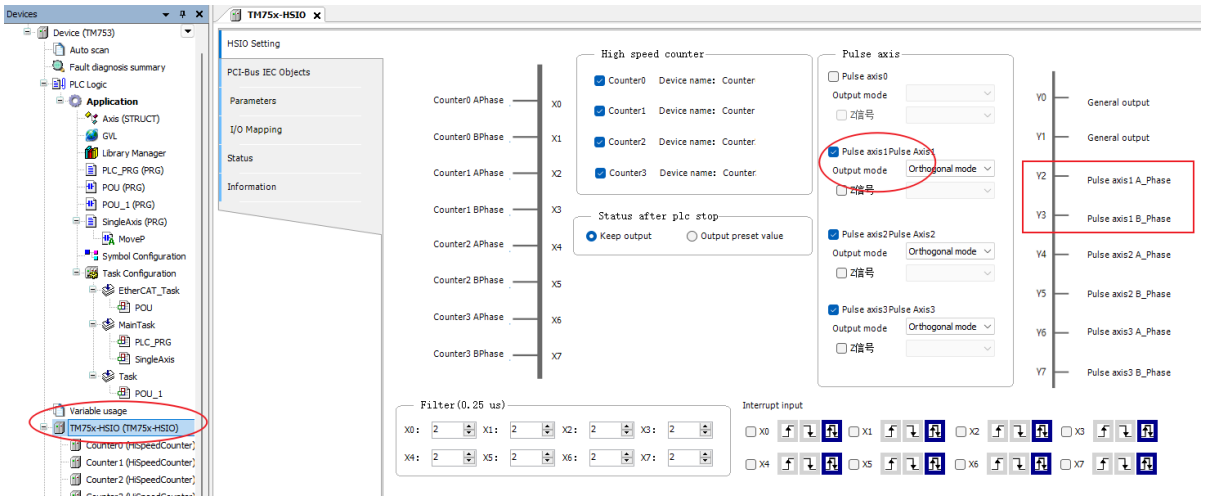
Table 4-1 HSIO Pulse Library Counting Instructions Description

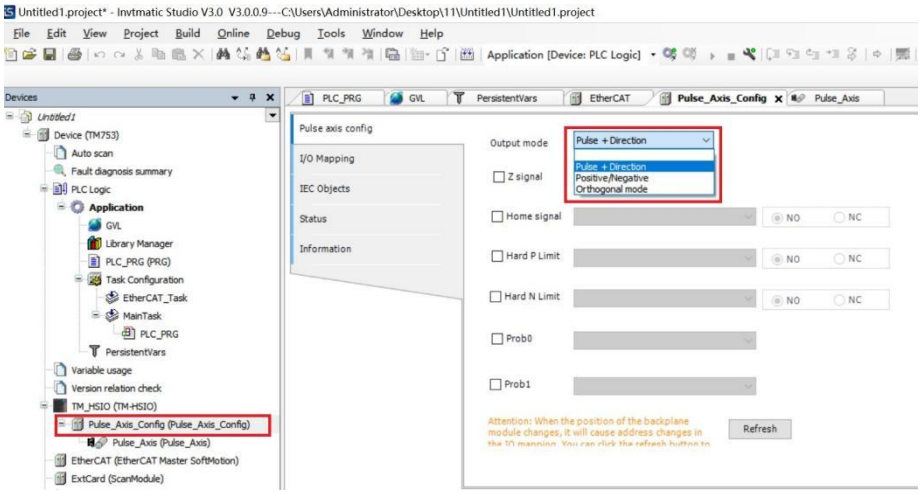
Instruction Category	Name	Function
Local encoder axis	ENC_Counter	Encoder enable (high-speed counter)
	ENC_CounterReset	Encoder reset
	ENC_CounterPreset	Encoder preset
	ENC_CounterProbe	Encoder probe
	ENC_SetUnit	Set axis gear ratio
	ENC_SetLineRotationMode	Set axis operation mode

4.1.3 Description of Output Port Functions

You can set 3 functions for the output port: normal output, high-speed pulse output, and output comparison. As shown in the figure below, the normal output port ranging from Y0 to Y7 is used by default. When the pulse output function is required, you need to check the pulse axis of the corresponding channel.

Figure 4-3 High-speed Output Port Configuration



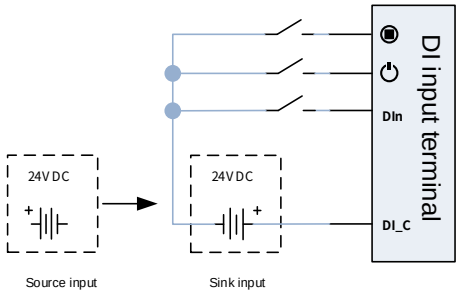


4.1.4 Normal Input and Output

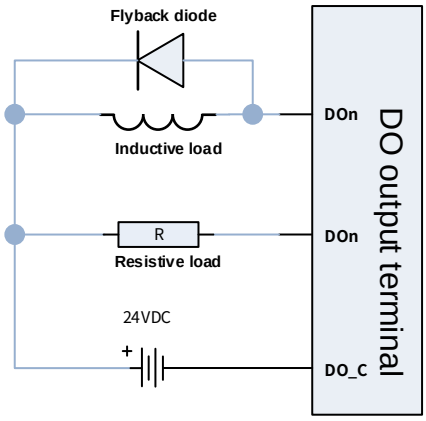
The normal input and output ports ranging from X0 to X7 and from Y0 to Y7 are used by default.

The output port of the TP2000 series PLC contains 8 input signals, 8 output signals. The output signal type is sink output. Y0–Y7 share the common terminal COM.

DI ouput specifications

Item	Specification
Wiring terminal	Removable push-in terminal
Input type	Digital input
Input mode	Source/sink
Input voltage	24V DC±10% (21.6V DC–26.4V DC)
Input current (typical)	14mA (actual value prevails)
ON voltage	>15V
OFF voltage	<5V
Hardware response time (ON/OFF)	2.5μs/2.5μs
Software filter time	Supported
Input resistance	Reference value: approx. 2.3kΩ
Isolation	Yes (opto-isolated)
Wiring specification	Cable length less than 3 m
Wiring diagram	

DO output specifications

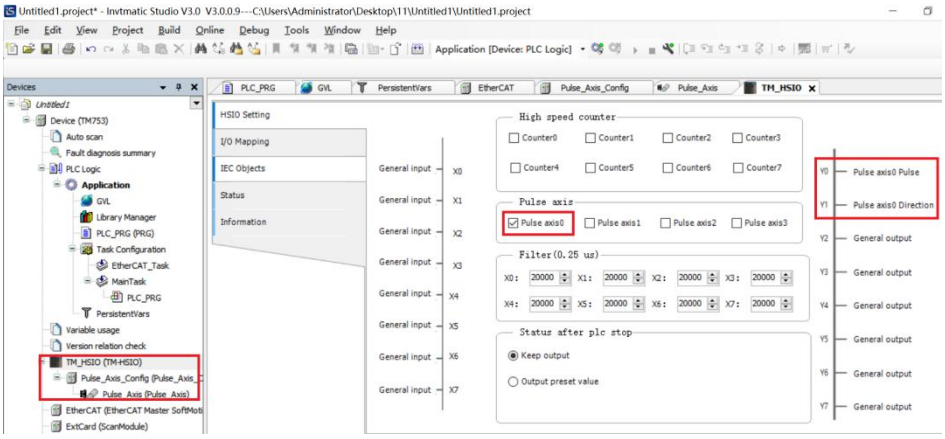
Item	Specification
Wiring terminal	Removable push-in terminal
Output type	Digital output, low-side output
Output mode	Sink
Output load (resistive load)	0.5A/point, 4A/module
Output load (inductive load)	7.2W/point, 24W/module
Output load (lamp load)	5W/ point, 18W/module
Hardware response time (ON/OFF)	Resistive load: 2.5μs/2.5μs Inductive load: 2.5ms/2.5ms Lamp load: 2.5ms/2.5ms
Leakage current at OFF	10μA
Isolation	Supported
Output action display	When the output is in the driving state, the corresponding point on the LCD screen lights up.
Protection functions	Overcurrent protection (short-circuit protection), reverse connection protection
Wiring specification	Cable length less than 3m
Wiring diagram	

4.1.5 High-speed Pulse Output

Once the pulse axis is checked, the signal port is configured as high-speed pulse output, and all 8 output ports can be configured as high-speed pulse outputs.

High-speed pulse output supports four modes: pulse+direction, FWD/REV pulse, quadrature-encoded pulse, and PWM. When configured as pulse+direction, FWD/REV pulse, or quadrature-encoded pulse, the output axis can be used as a standard CiA402 output axis, allowing the use of motion control instructions from the SM3_Basic library in the CodeSys platform, such as MC_Power and MC_MoveVelocity. When configured as PWM, the output can be implemented using INVT-developed function blocks. For detailed usage of each instruction, refer to the *INVT Medium- and Large-Scale PLC Programming Manual*.

Figure 4-4 High-Speed Output Mode

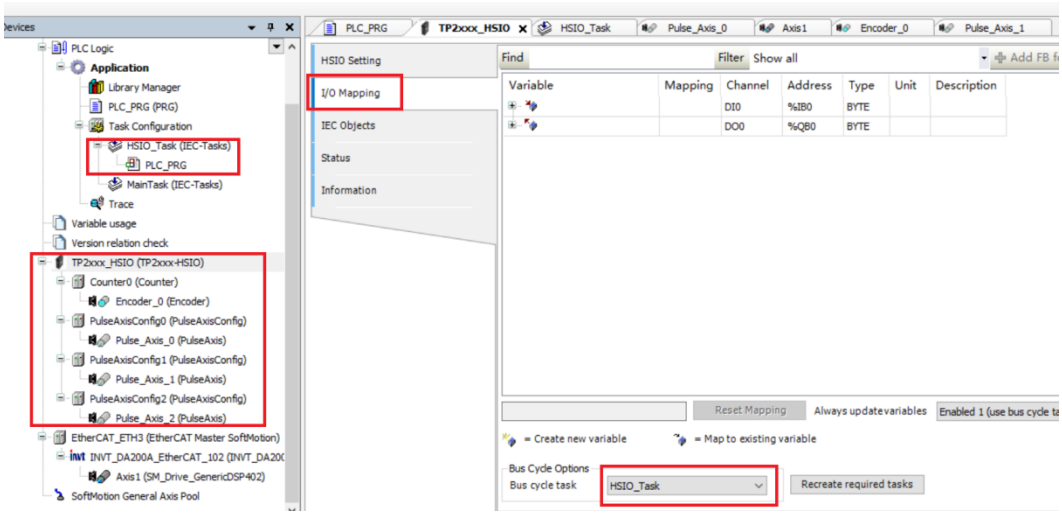


Note: The HSIO pulse output task cycle time can only be set to 1ms, 2ms, or 4ms. Setting any other cycle time will prevent the relevant function blocks from operating correctly.

HSIO Task Configuration

To ensure the proper operation of high-speed I/O functions, the program (PRG) utilizing these functions and the associated high-speed I/O devices must be assigned to the same task.

As shown in the figure below, the program PLC_PRG is assigned to run under HSIO_Task. The I/O mapping of the high-speed I/O device TP2XXX_HSIO (as well as other subdevices) is also assigned to HSIO_Task. Otherwise, the high-speed I/O function cannot operate properly.



To utilize high-speed I/O functions, the associated programs and devices must be assigned to the same task (either HSIO_Task or EtherCAT_Task is acceptable. However, the cycle time of the selected task must match the HSIO_Task cycle time).

Note: The content in this chapter applies to the pulse counting and pulse output functions of the TM series and TP2000 series PLCs. The HSIO pulse output task cycle time can only be set to 1ms, 2ms, or 4ms. Configuring any other cycle time will prevent the function blocks from operating correctly.

For the InvtMatic Studio V1.3.7.0 series upper computer, the namespace of the TM high-speed I/O library is as follows:

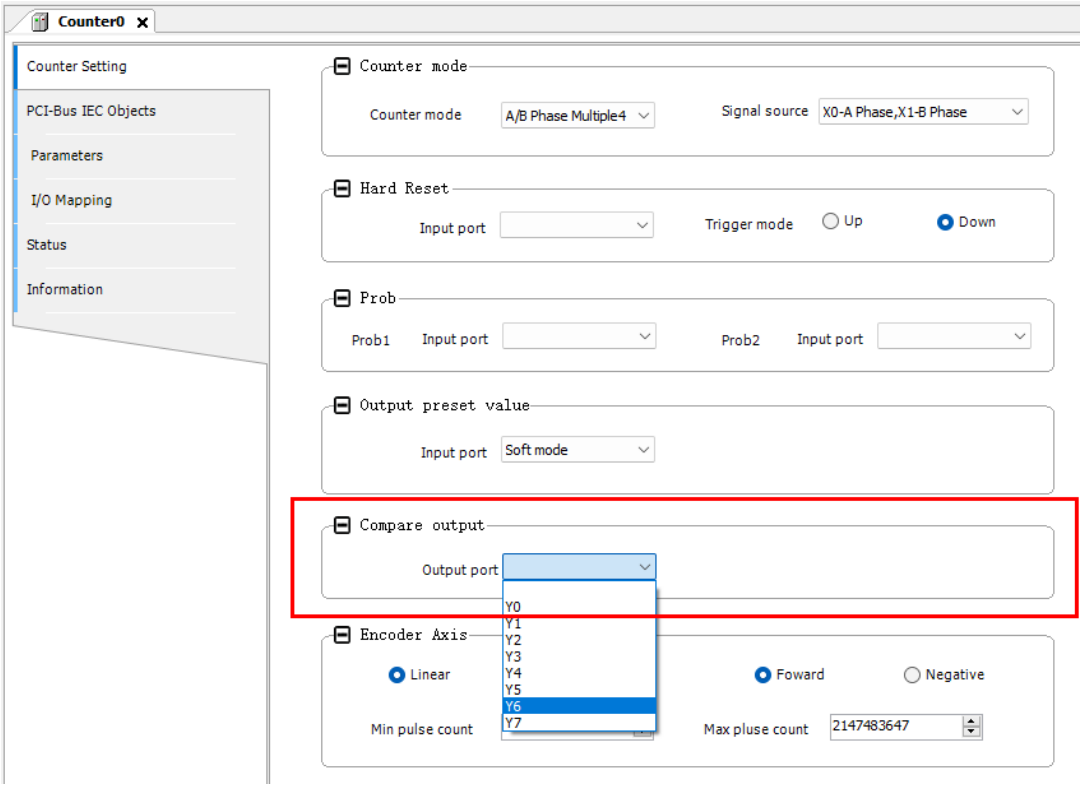
In the current V3.0.1.4 series upper computer, the namespaces for both TM and TP high-speed I/O libraries have been unified, as shown in the figure below. The default prefix used in programming is unified as INVT_HSIO.

4.1.6 Comparison Output

The comparison output function outputs the result of the counter single value comparison, and each counter channel has an comparison output function. If the counter value is equal to the set comparison value, a high output will be given; and if it is not equal, a low output will be given.

- **The comparison output port is configured in the corresponding counter channel**

Figure 4-5 Comparison Output Configuration



- **Comparison output control**

Table 4-2 HSIO Pulse Library Comparison Output Instructions

Instruction Category	Name	Function
Comparison output instruction	ENC_CompareArray	Array comparison output
	ENC_CompareSingle	Single-point comparison output
	ENC_CompareStep	Equidistant comparison output

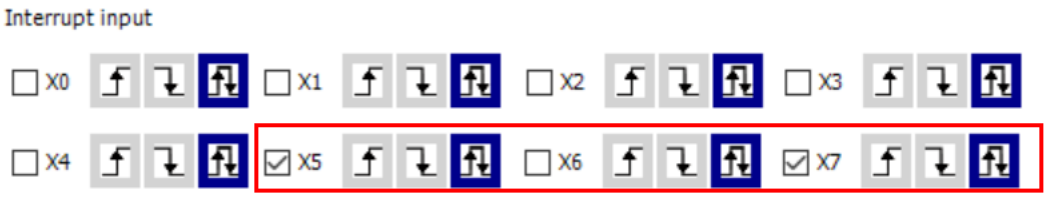
4.1.7 Description of External Interrupt

An interruption refers to the process in which the CPU suddenly stops the execution of task A to execute task B, and then comes back to continue executing task A.

The steps to implement the external interrupt function are as follows:

Step 1 Set the input port for the interrupt function, select rising edge, falling edge, or rising/falling edge trigger, and set the filter parameter which is 0.5μs (2*0.25μs) by default.

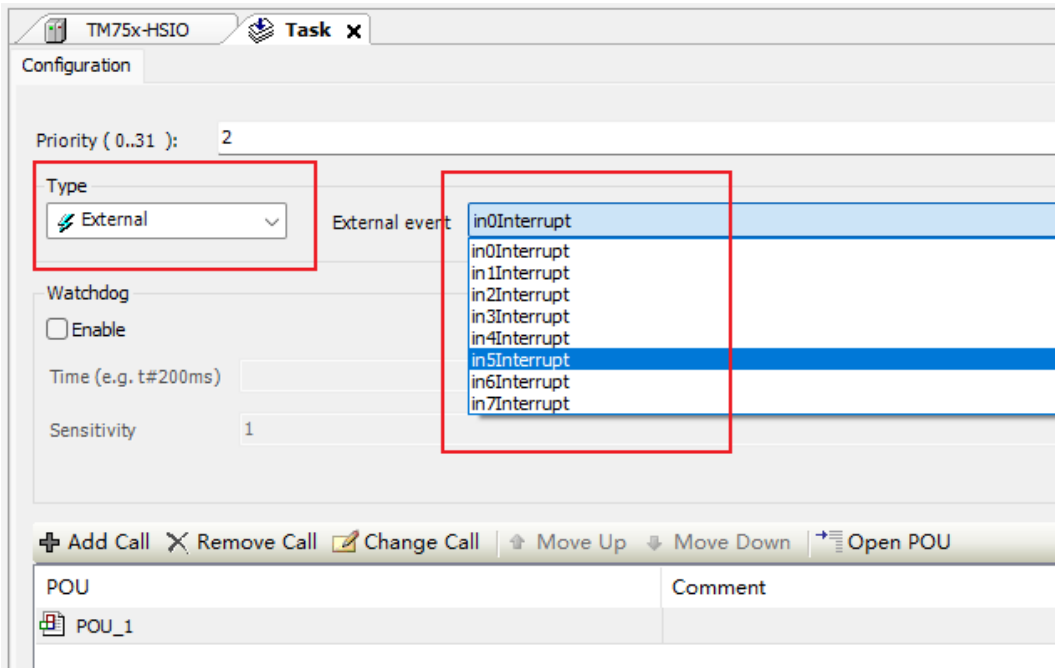
Figure 4-6 External Interrupt Input Configuration



Step 2 Select an interrupt task

In the Invtmatic Studio task, select External as the task type, select the event inxInterrupt of the input port X0–X7, where x ranges from 0 to 7, and trigger the corresponding POU_1 program by the interrupt task.

Figure 4-7 Task Triggering by an External Interrupt



4.2 Local I/O Expansion Module Configuration

The PLC body can be directly expanded with Flex series modules, or connected to the FK1100 coupler via the EtherCAT bus to use expansion modules. A single TM700 series PLC or FK1100 coupler can be expanded with up to 32 Flex series modules (including the power relay). The TP2000 series PLC body cannot be directly connected to the Flex series modules, but it can be connected to the FK1100 coupler via the EtherCAT bus and the expansion modules can be assembled to the coupler. For the models and functions of Flex series modules, see Table 4-3 for details.

Table 4-3 Types of Expansion Modules

Module Name	Flex Series Module	Description
Coupler	FK1100	EtherCAT bus adapter, expandable to 32 bus nodes
Digital Input Module	FL1001	16-channel digital input module, source/sink input
	FL1002	32-channel digital input module, source/sink input
Digital Output Module	FL2002	16-channel digital output module, source output
	FL2003	32-channel digital output module, source output
	FL2102	16-channel digital output module, sink output
	FL2103	32-channel digital output module, sink output
	FL2201	8-channel relay output
Analog Input	FL3003	4-channel analog input module, current/voltage
Temperature Module	FL3103	4-way temperature module, thermal resistance type
	FL3203	4-way temperature module, thermocouple type
Analog Output	FL4003	4-channel analog output module, current/voltage
Hybrid input module	FL5005	16-channel digital input + 16-channel digital output
	FL5105	16-channel digital input + 16-channel digital output

4.2.1 Expansion Module Configuration

For TM700 series series PLCs, you can configure expansion modules through two methods: manual addition and automatic scanning. Automatic scanning takes precedence.

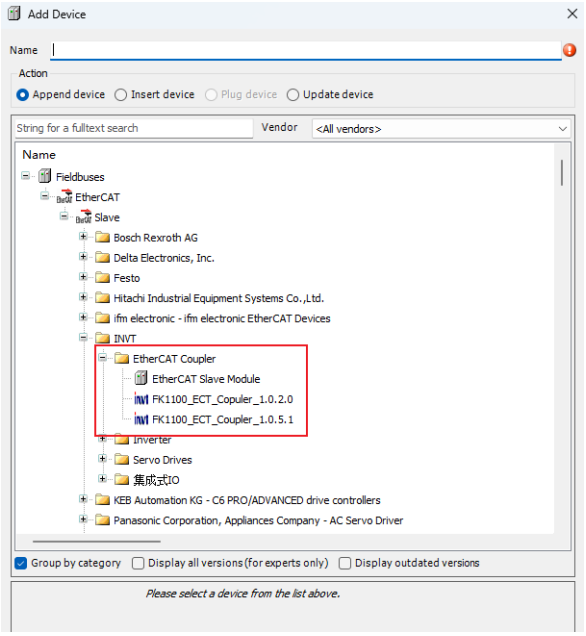
Method 1: manual addition

Open the Invtmatic Studio programming software, create a new project, select the programming language (for details, see chapter 2 Getting Started), and add the required extension modules.

1. Add the FK1100 coupler module

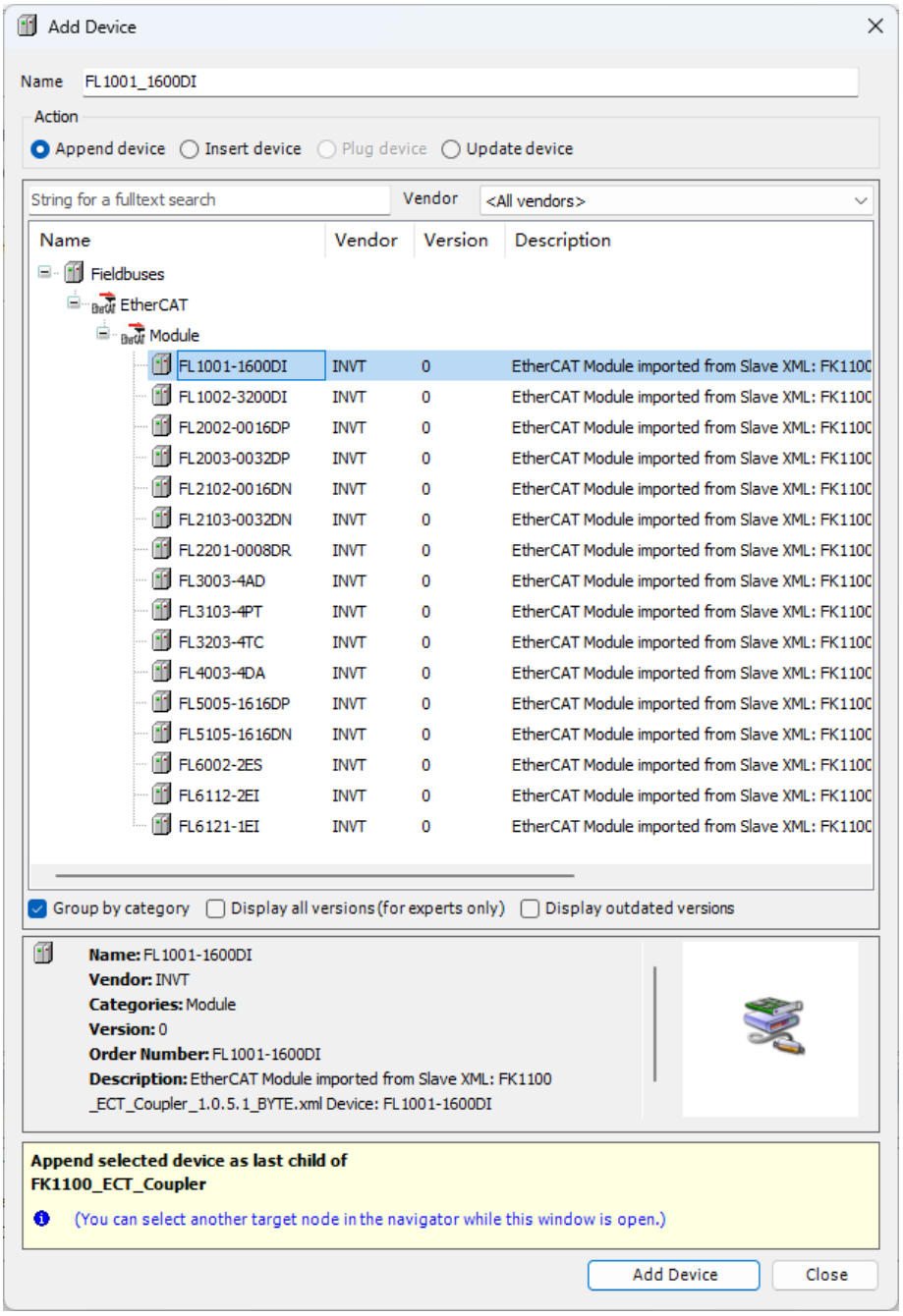
Step 1 Right-click **Device** in the left device tree, click **EtherCAT_Master_SoftMotion** to add it, right-click **EtherCAT_Master_SoftMotion**, select **Add Device**, and add the coupler to the device tree.

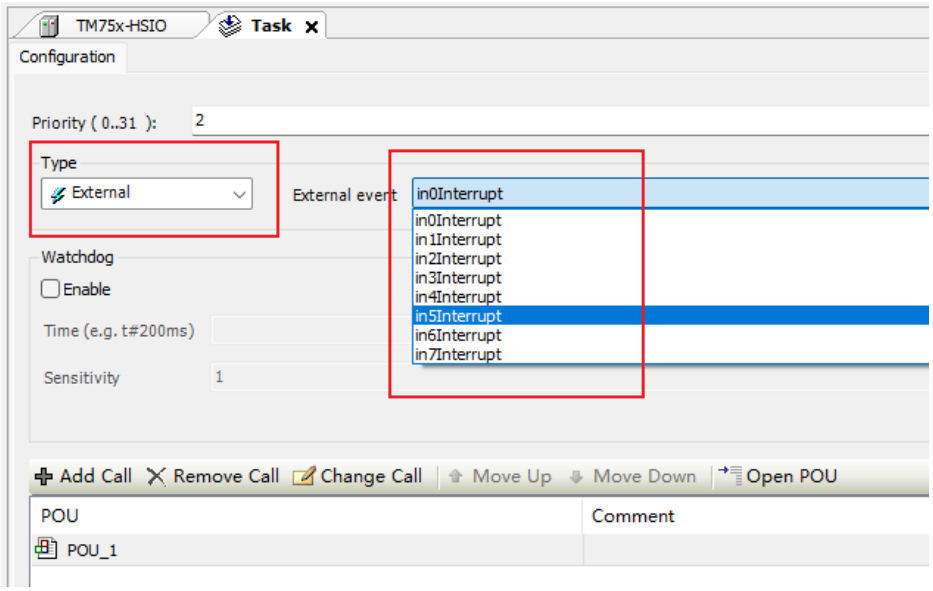
Figure 4-8 Adding a Coupler Manually



Step 2 Select the **FK100_ECT_Coupler** in the device tree, click **Add Device**, select the required module in the **Module** category, and click **Add Device** to add it into the device tree, as shown in Figure 4-9.

Figure 4-9 Adding an Expansion Module Manually

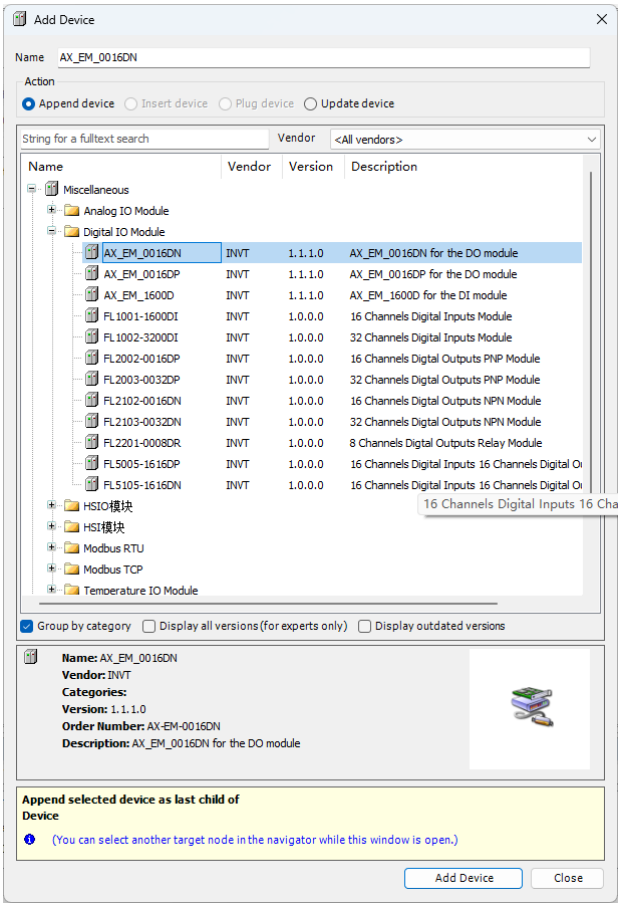


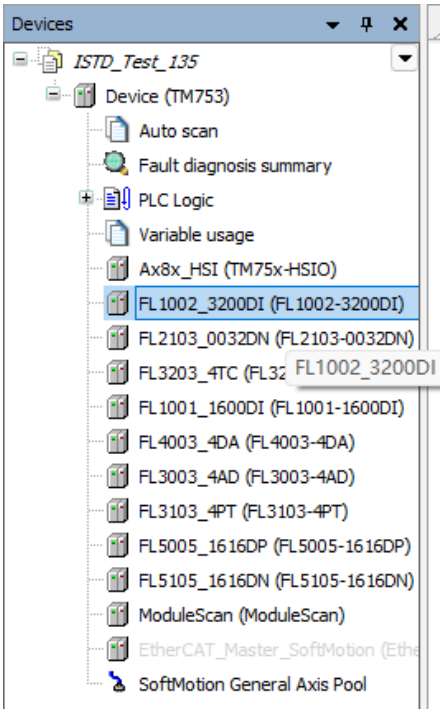


2. Add modules to the controller (TM700 series PLC)

Right-click **Device** in the device tree, choose **Add Device**, select the required module under the **Miscellaneous** category, and click **Add Device** to add it into the device tree, as shown in Figure 4-10Figure 4-9.

Figure 4-10 Adding an Expansion Module to the PLC





Method 2: automatic scanning (recommended)

1. Scan EtherCAT bus configuration

Step 1 Open the Invtmatic Studio programming software, create a new project, and select the programming language.

Step 2 Connect to the PLC and add EtherCAT master. For details, see chapter 2 Getting Started.

Step 3 Right-click **EtherCAT_Master_SoftMotion** and select **Scanned Devices**. The connected device will be automatically displayed. Click **Copy All devices to Project**.

Figure 4-11 Scanning Couplers and Expansion Modules

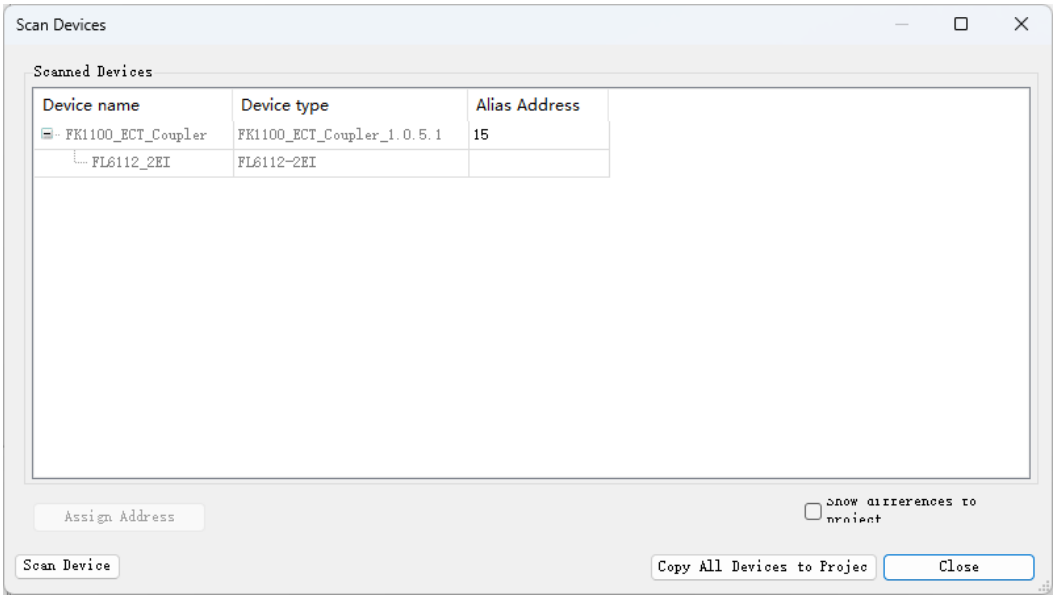
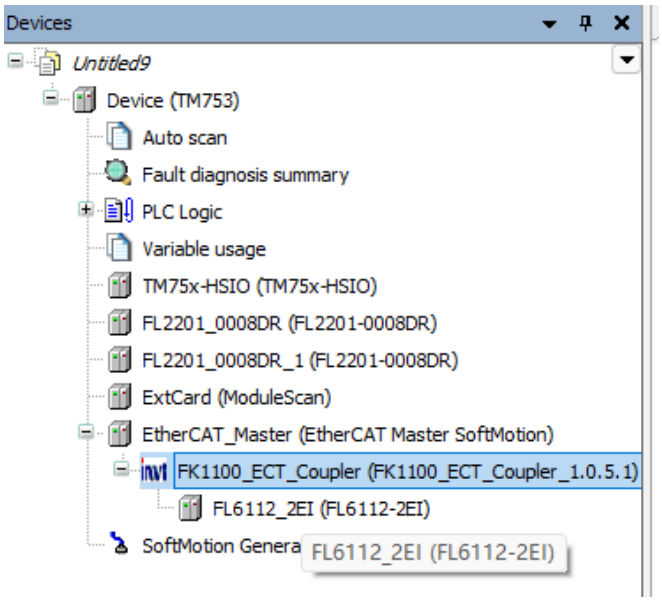


Figure 4-12 Couplers and Expansion Modules Added after Scanning



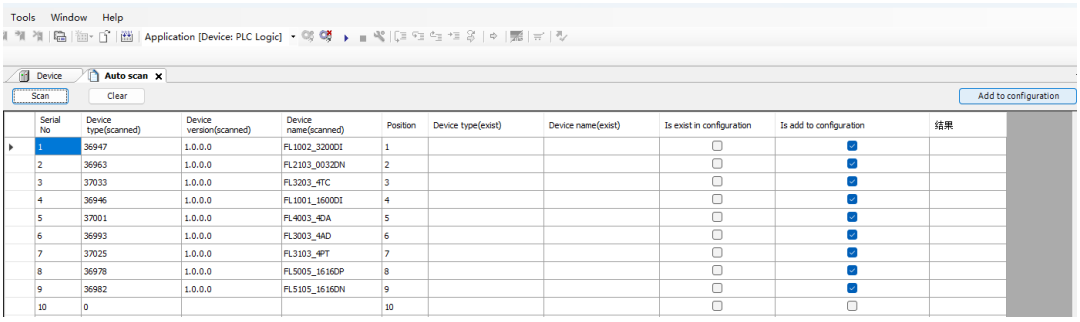
2. Scan local expansion modules

Step 1 Open the Invtmatic Studio programming software, create a new project, and select the programming language.

Step 2 Connect to the PLC. To scan local expansion modules, you need to log in to the PLC (PLC running is not required). For details, see chapter 2 Getting Started.

Step 3 Double-click **Automatic Scanning** and click **Scan**. The list shows the expansion modules connected to the device. Click **Add to configuration**. At this time, the configuration is completed.

Figure 4-13 PLC Backplane Scanning

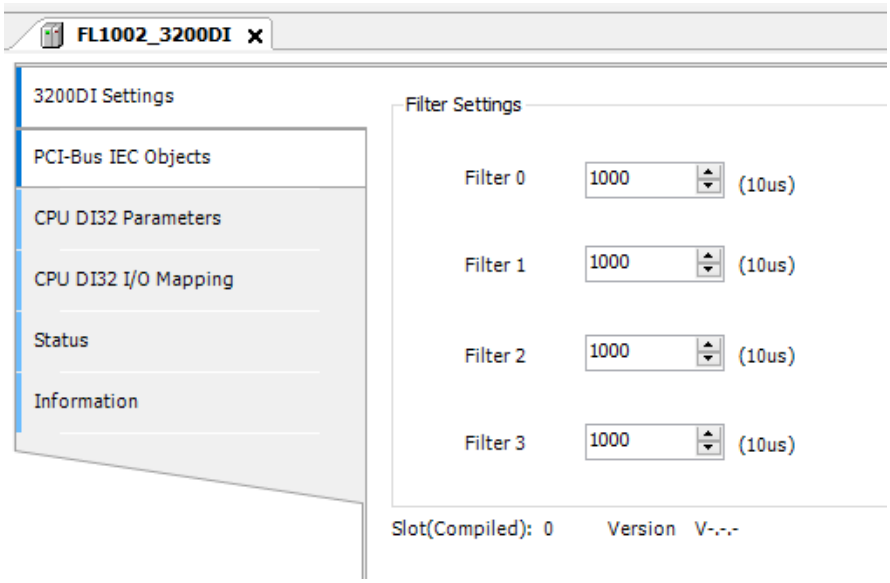


4.2.2 Digital Input Module

The digital input module is mainly used to configure filter parameters and has I/O Mapping, Status, and Information pages. Generally, you only need to map the I/O variables on the I/O Mapping page to obtain the digital input values. The 32-point digital input module is taken as an example in the figure below.

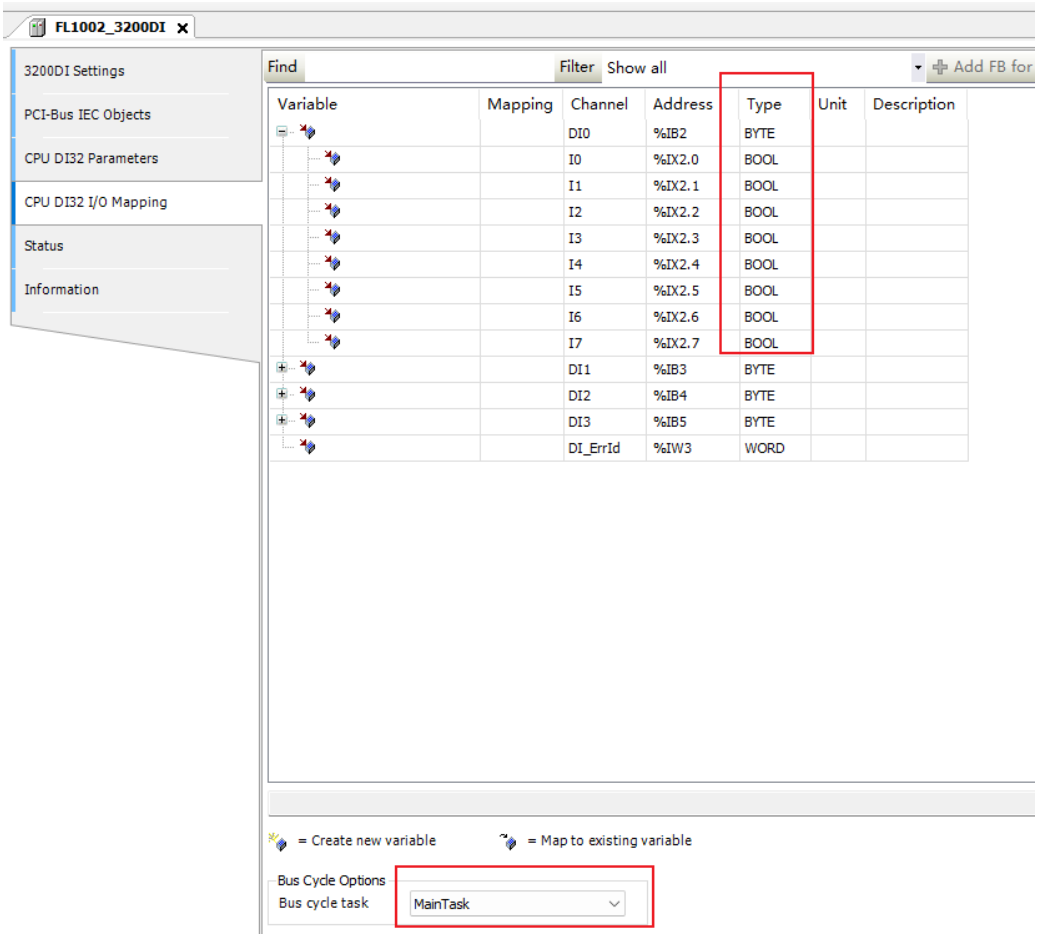
- Filter parameters: 10 ms (1000*10 μs) by default.

Figure 4-14 Input Module Configuration



- DI32 I/O mapping: You can get the input state through the BYTE or BOOL type.

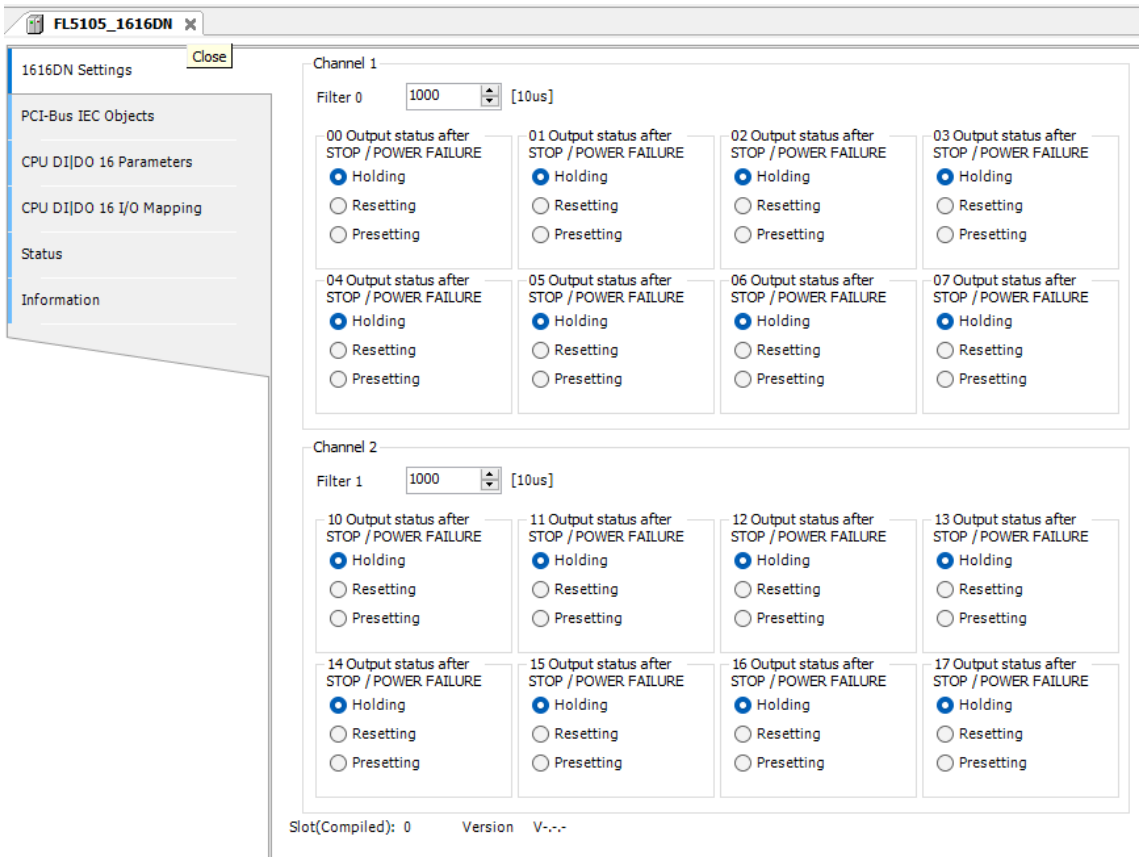
Figure 4-15 Variable Mapping of the Input Module



4.2.3 Digital Output Module

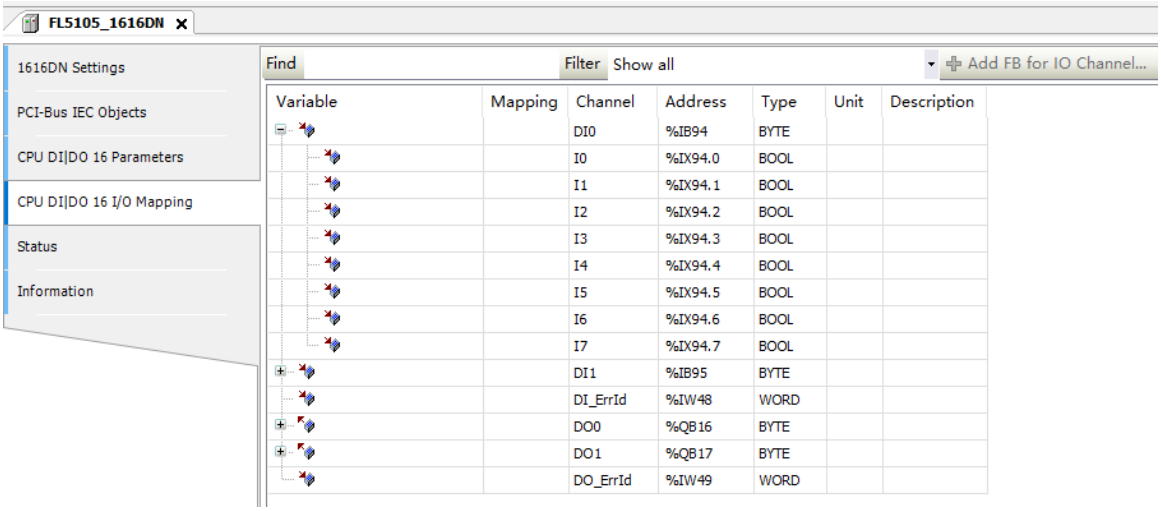
The digital output module is mainly used to set the output state after stopping and has I/O Mapping, Status, and Information pages. Generally, you only need to map the I/O variables on the I/O Mapping page to obtain the digital output values. The 16-point digital output module is taken as an example in the figure below.

Figure 4-16 Output Module Configuration



DN16 I/O mapping: You can control the output state through the BYTE or BOOL type.

Figure 4-17 Variable Mapping of the Output Module

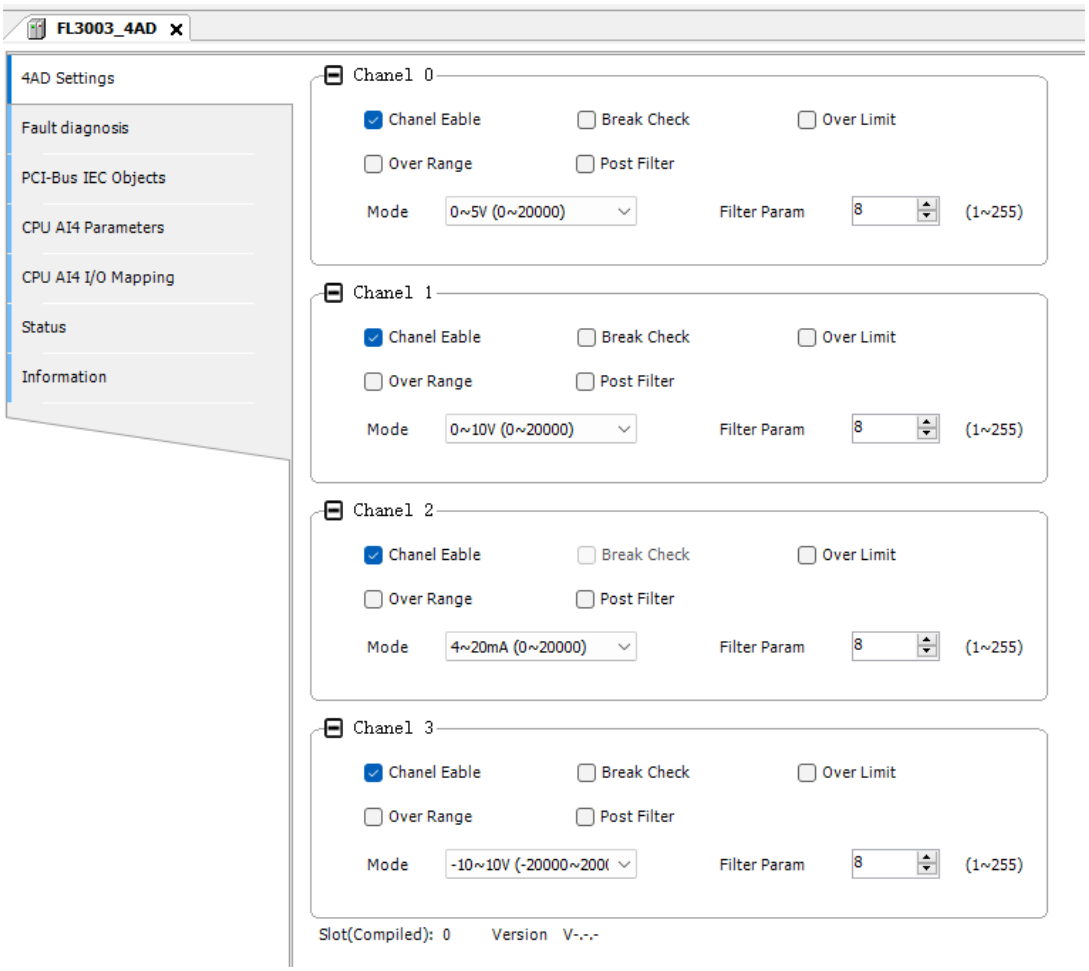


4.2.4 Analog Input Module

■ General Configuration

The analog input module includes 4 channels, each requiring parameter configuration and I/O mapping register (16-bit) settings. Only one channel of each module is described below.

Figure 4-18 Analog Input Module Configuration



- Channel enable: The channel can only be used after it is enabled.
- Filter parameter: Analog input channel filter, ranging from 1 to 255.
- Conversion mode: Set the analog input conversion type, which determines the channel input conversion type and the range of the conversion value. For details on the conversion type, see Table 4-4.
- Disconnection detection: Set whether disconnection detection is enabled for the analog input channel. Since it is impossible to distinguish between analog input 0 and disconnection, the disconnection flag cannot be activated in all conversion modes that include 0 value input.
- Over-limit detection: Set whether over-limit detection is enabled for the analog input channel.
- Over-range detection: Set whether over-range detection is enabled for the analog input channel.
- Enhanced filtering: Set whether the analog input channel uses enhanced filtering.

■ 4AD I/O Mapping

4AD means 4-channel analog input. Each channel of analog input corresponds to a 16-bit integer value. For the relationship between analog value and digital value, see the general analog input configuration. You can map a variable to each 16-bit integer value on this interface to obtain the digital value corresponding to the analog value of an input channel. For details, see Table 4-4.

Figure 4-19 Variable Mapping of the Analog Input Module

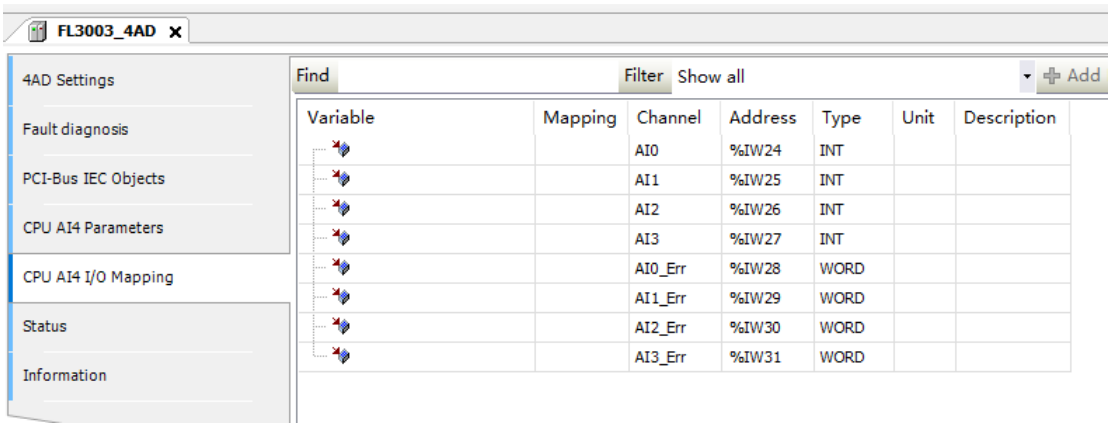


Figure 4-20 4AD Information Interface of the Analog Input Module

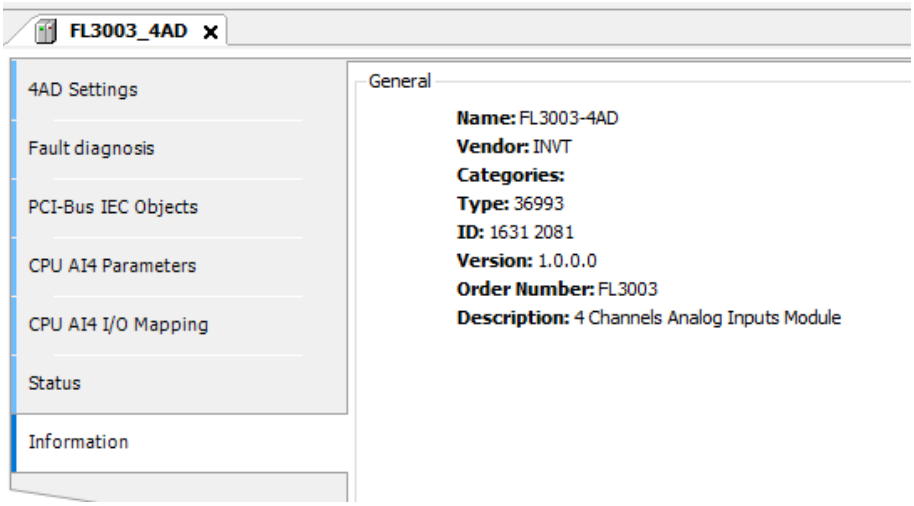


Table 4-4 Correspondence between Mapped and Actually Input Analog Values

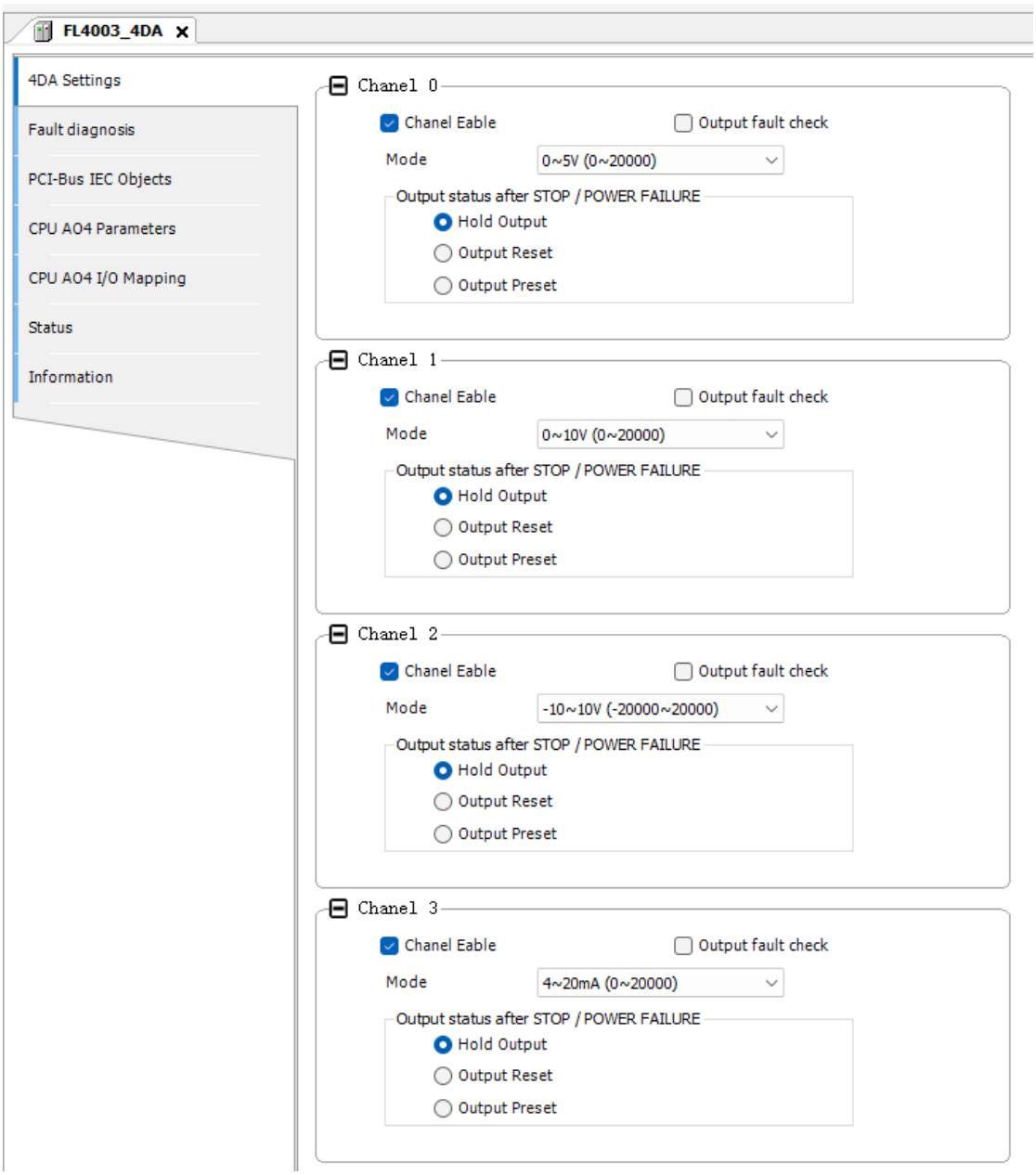
Type	Enter the Rated Range	Rated Corresponding Digital Quantity
Analog voltage input	-10V–10V	-20000–+20000
	0V–10V	0–20000
	-5V–+5V	-20000–+20000
	0V–5V	0–20000
Analog current input	-20mA–20mA	-20000–20000
	0mA–20mA	0–20000
	4mA–20mA	0–20000

4.2.5 Analog Output Module

■ General Configuration

The analog output module includes 4 channels, each requiring parameter configuration and I/O mapping register (16-bits) settings. Only one channel of each module is described below.

Figure 4-21 Variable Mapping of the Analog Output Module



- Conversion mode: Set the analog output conversion type, which determines the channel output conversion type and the range of the conversion value. For details on the conversion type, see Table 4-5.
- Keep output: After the module stops running, the output will keep the last output value.
- Clear output: After the module stops running, the output is always 0.
- Output preset value: After the module stops running, the output is always the preset value. The

preset value range is related to the current conversion mode. See Table 4-5 for details.

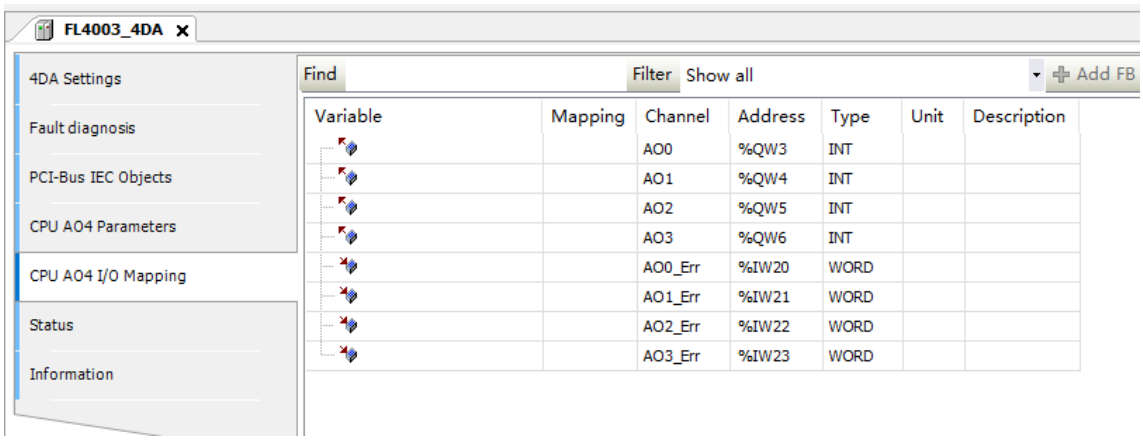
Table 4-5 Correspondence between Mapped and Actually Input Analog Values

Type	Enter the Rated Range	Rated Corresponding Digital Quantity
Analog voltage output	-10–10 V	-20000–+20000
	0–10 V	0–20000
	-5–+5 V	-20000–+20000
	0–5 V	0–20000
Analog current output	4–20 mA	0–20000
	0–20 mA	0–20000
	-20–20 mA	-20000–+20000

■ 4DA I/O Mapping

4DA means a 4-channel analog output. Each channel of analog output corresponds to a 16-bit integer value. For the relationship between analog value and digital value, see Table 4-5. You can map a variable to each 16-bit integer value on this interface to output the variable value to the current channel, and then convert it into an analog value output through the analog output module.

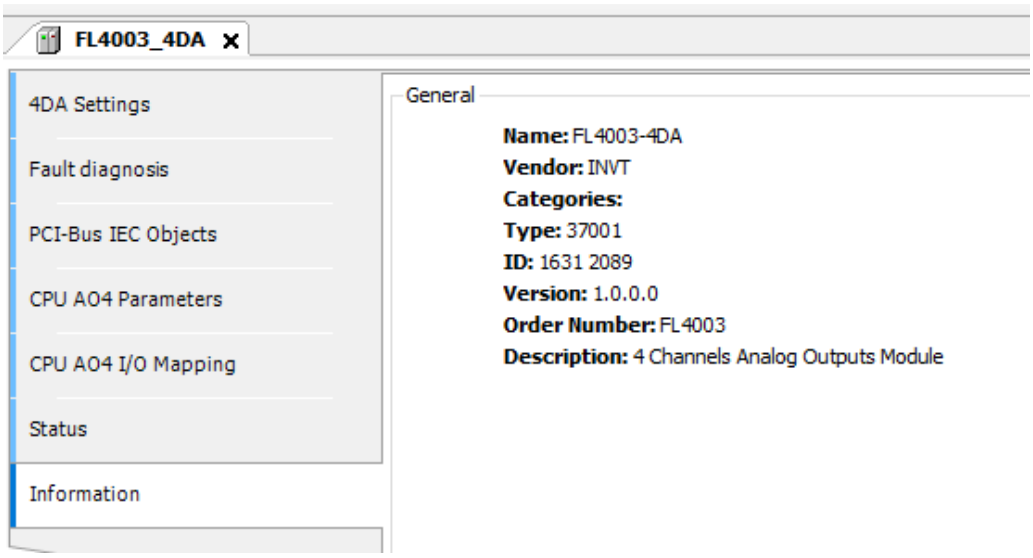
Figure 4-22 Variable Mapping of the Analog Output Module



■ Information

It displays the basic information of 4DA module devices: name, manufacturer, group, type, ID, version, model number, and description. In addition, after logging in to the Information interface, the system will read and display the single-board software version (4DA embedded software version) and logic software version (FPGA software version in the 4DA module) of the 4DA module.

Figure 4-23 4DA Information Interface of the Analog Output Module



4.2.6 Temperature Module

The temperature module includes two types: 4TC (4-channel temperature detection module, supporting thermocouples) and 4PT (4-channel temperature detection module, supporting thermal resistors).

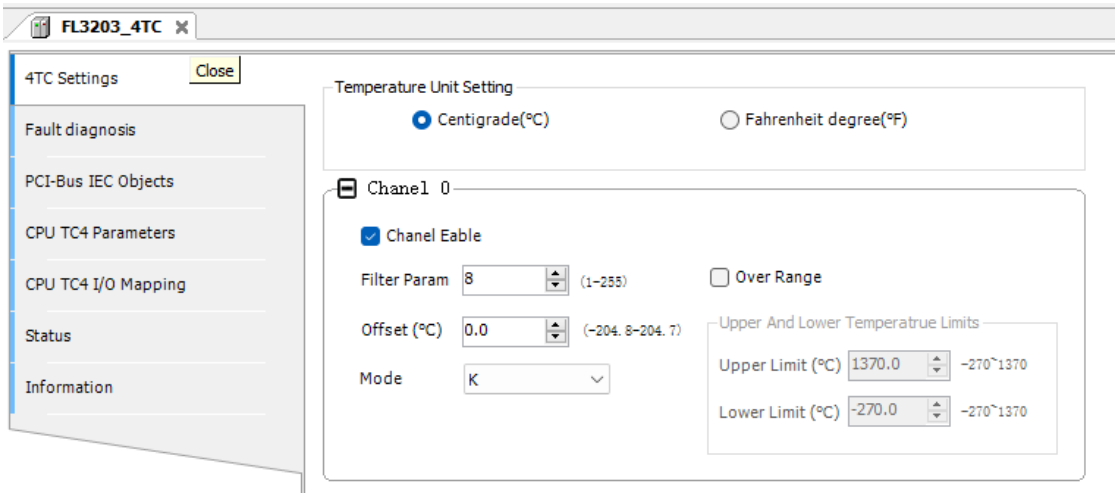
4.2.6.1 4TC Temperature Module

The 4TC temperature module (4-channel temperature detection module, supporting thermocouples) has corresponding general configuration and channel configuration. The general configuration is used to configure the unit type and sampling period of the temperature module, while the channel configuration is used to configure the sensor type, filter time, over-limit settings, temperature offset, and other parameters of each channel.

■ **Channel Configuration**

Different types of modules support different channels, and 4TC supports 4 channels. Since the configuration parameters of each channel are basically the same, only one channel is described here. The configuration of a 4TC channel is shown in Figure 4-24.

Figure 4-24 4TC Temperature Module Configuration



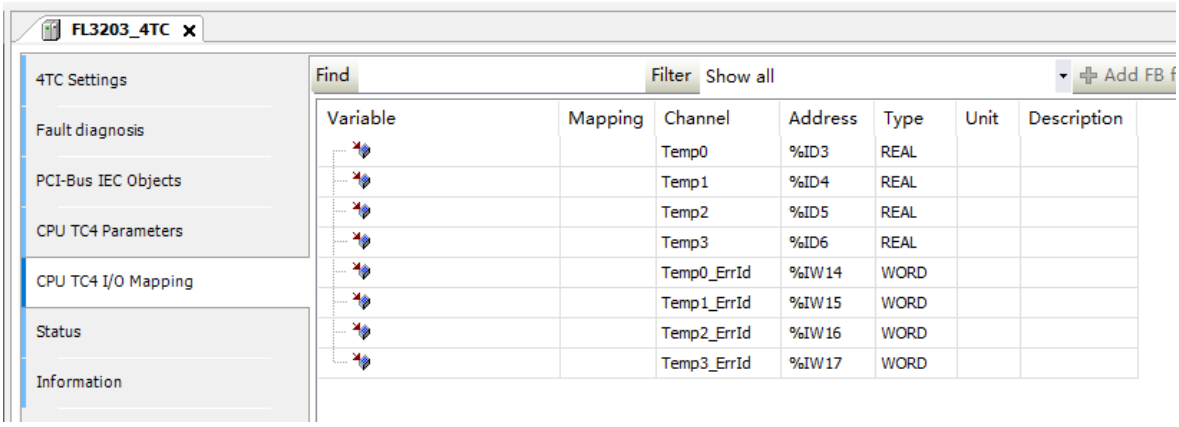
- Temperature unit: The unit used for the temperature module input, which supports Celsius or Fahrenheit.
- Channel enable: The channel can only be used after it is enabled.
- Sensor type: 4TC sensor type. See Table 4-6 for the sensor specifications. The K sensor is used by default.
- Filter parameter: The filter used by this channel of the temperature module, ranging from 1 to 255, 8 by default.
- Over-range detection: When it is enabled, if the temperature is not between the upper and lower limits, an over-limit fault will be reported. For details, see Table 4-6.

- Offset value (°C): It is used to set the temperature module offset compensation value, ranging from -204.8 to 204.7.

■ **I/O mapping**

Different types of temperature modules have different numbers of channels and IO mappings. The figure below shows the I/O mapping interface of 4TC. The parameter value of each channel is the temperature value.

Figure 4-25 Variable Mapping of the 4TC Temperature Module



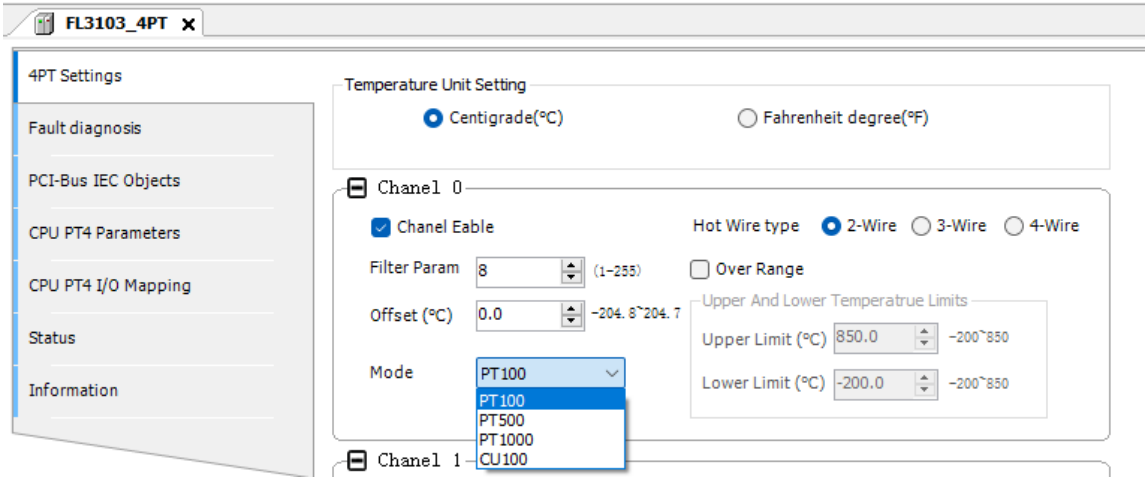
4.2.6.2 4PT Temperature Module

The 4PT temperature module (4-channel temperature detection module, supporting thermal resistors) has corresponding general configuration and channel configuration. The general configuration is used to configure the unit type and sampling period of the temperature module, while the channel configuration is used to configure the sensor type, filter time, over-limit settings, temperature offset, and other parameters of each channel.

■ **Channel configuration**

4PT supports 4 channels. Since the configuration parameters of each channel are basically the same, only one channel is described here. The configuration of one channel of 4PT is shown in the figure in Figure 4-26.

Figure 4-26 4PT Temperature Module Configuration

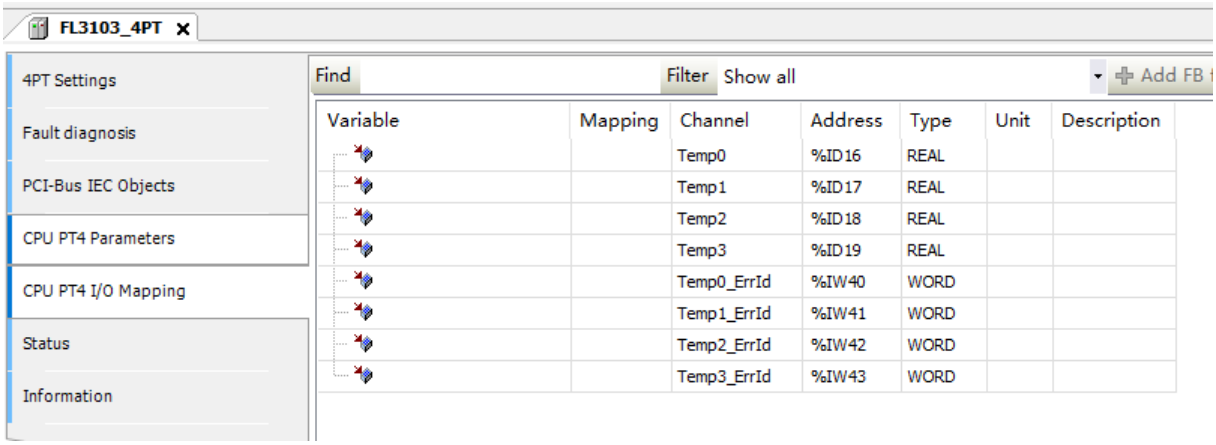


- Temperature unit: The unit used for the temperature module input, which supports Celsius or Fahrenheit.
- Channel enable: The channel can only be used after it is enabled.
- Thermal resistor wiring mode: 2-wire, 3-wire, or 4-wire.
- Sensor type: 4PT sensor type. See Table 4-6 for the sensor specifications. The PT100 sensor is used by default.
- Filter parameter: The filter used by this channel of the temperature module, ranging from 1 to 255, 8 by default.
- Over-range detection: When it is enabled, if the temperature is not between the upper and lower limits, an over-limit fault will be reported. For details, see Table 4-6.
- Offset value ((℃): It is used to set the temperature module offset compensation value, ranging from -204.8 to 204.7.

■ I/O mapping

Different types of temperature modules have different numbers of channels and IO mappings. The figure below shows the I/O mapping interface of 4PT. The parameter value of each channel is the temperature value.

Figure 4-27 Variable Mapping of the 4PT Temperature Module



■ **Information**

Figure 4-28 Information Interface of the 4PT Temperature Module

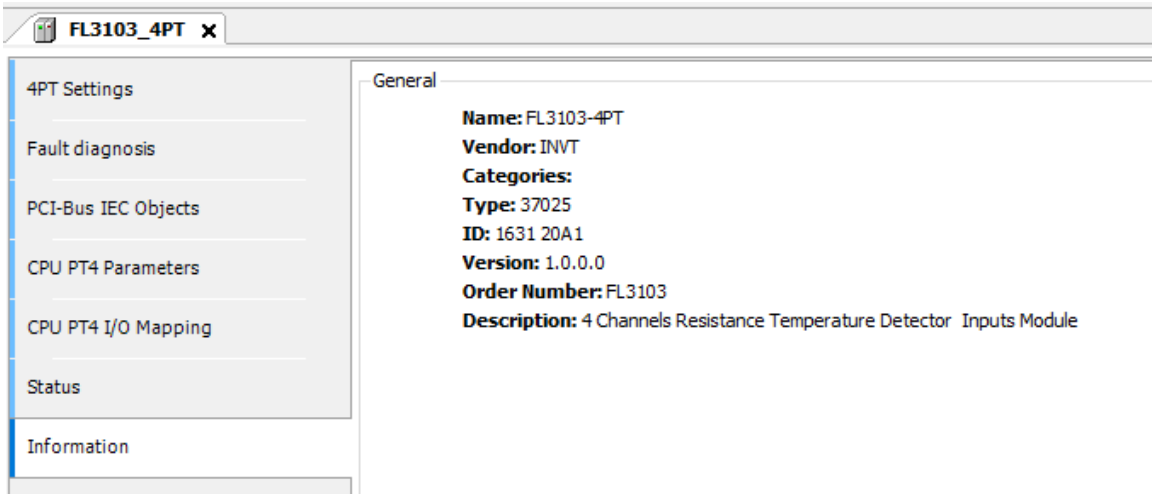


Table 4-6 Supported Sensor Types and Measurement Ranges

Item	Sensor Type	Temperature Range in Celsius	Temperature Range in Fahrenheit
Thermal resistor type	PT100	-200.0–850°C	-328.0–1562.0°F
	PT500	-200.0–850°C	-328.0–1562.0°F
	PT1000	-200.0–850°C	-328.0–1562.0°F
	CU100	-50.0–150°C	-58.0–302.0°F
Thermocouple type	B	200.0–1800°C	392.0–3272.0°F
	E	-270.0–1000°C	-454.0–1832.0°F
	N	-200.0–1300°C	-328.0–2372.0°F
	J	-210.0–1200°C	-346.0–2192.0°F
	K	-270.0–1370°C	-454.0–2498.0°F
	R	-50.0–1765°C	-58.0–3209.0°F
	S	-50.0–1765°C	-58.0–3209.0°F
	T	-270.0–400°C	-454.0–752.0°F

4.3 Priority Setting for Each Module Task (Recommended Value)

If the created project contains multiple functional modules, create multiple tasks and set the task priority as shown in Figure 4-29. For the recommended value of task priority, see Table 4-7.

Figure 4-29 Example of Project Task Priority Setting

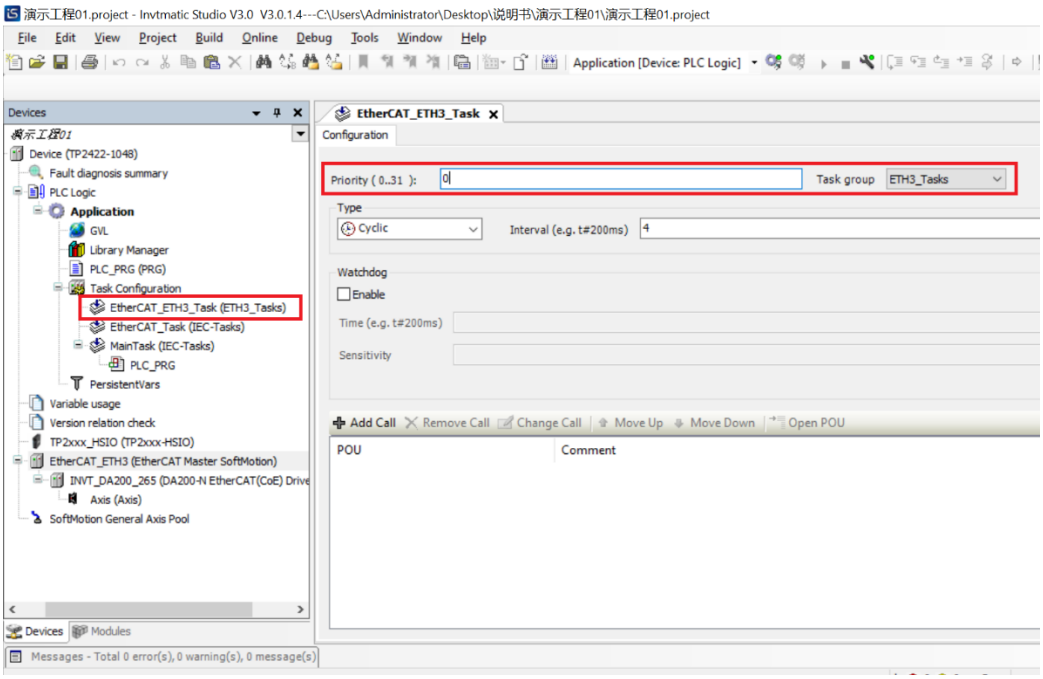


Table 4-7 Priority Setting

Function Module	Recommended Priority
EtherCAT	0
High-speed I/O (default configuration to EthercatTask)	0
MainTask	1–10
Modbus RTU, Modbus TCP	5–15
Controller configuration module (such as time reading and IP setting)	31

5 Communication Network Configuration

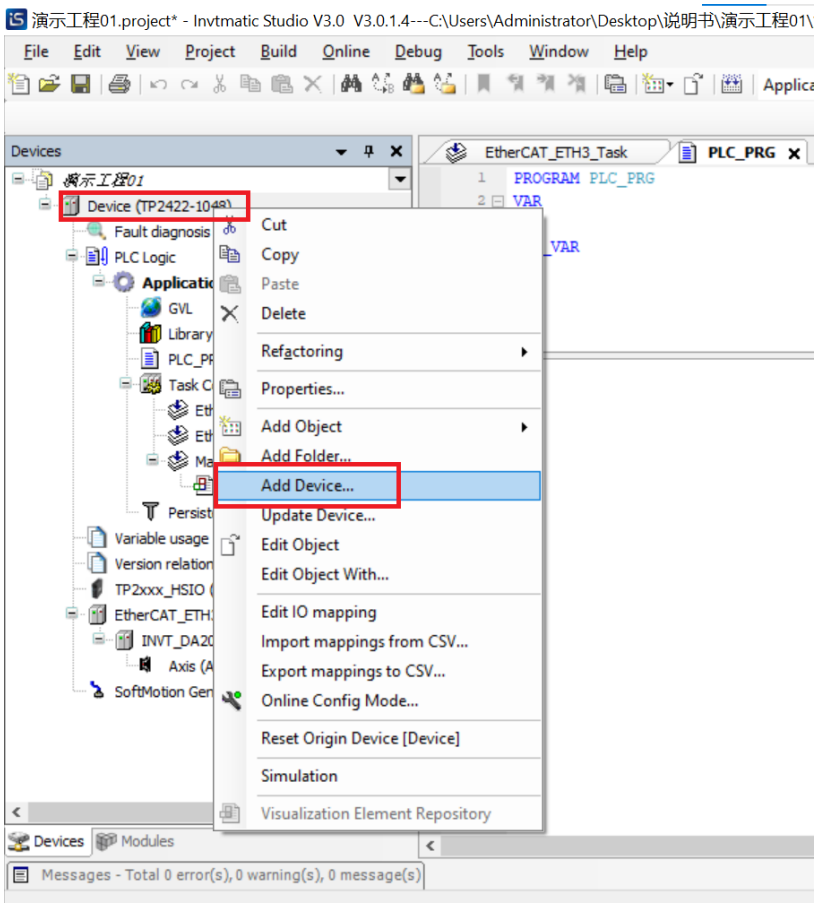
The PLC network configuration mainly involves the following networks: Modbus TCP, Modbus RTU, EtherCAT, CANopen, and EtherNet/IP.

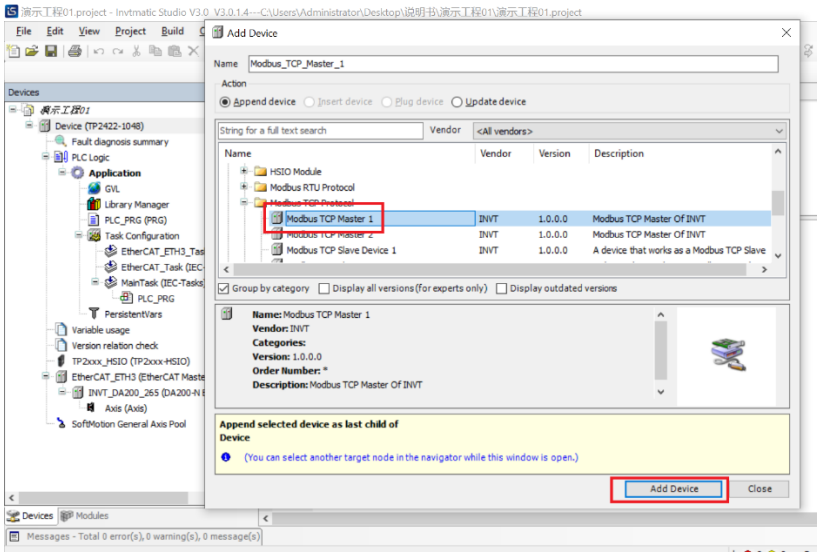
5.1 Modbus TCP

5.1.1 Modbus TCP Master Configuration

When the PLC is used as a Modbus TCP master, right-click **Device** in the left device tree, select **Add Device**, and then **Dedicated Device** > **Modbus TCP Protocol** > **Modbus TCP Master 1** in the pop-up window, and click **Add Device** in the lower right corner.

Figure 5-1 Adding a Modbus TCP Master





Double-click the master device **Modbus_TCP_Master 1** in the device tree to open the Modbus master configuration window, as shown in Figure 5-2. The frame interval (ms) refers to the time interval between the master receiving the last response data frame and the next request data frame. This parameter can be used to adjust the data exchange rate.

Figure 5-2 Modbus TCP Master Settings

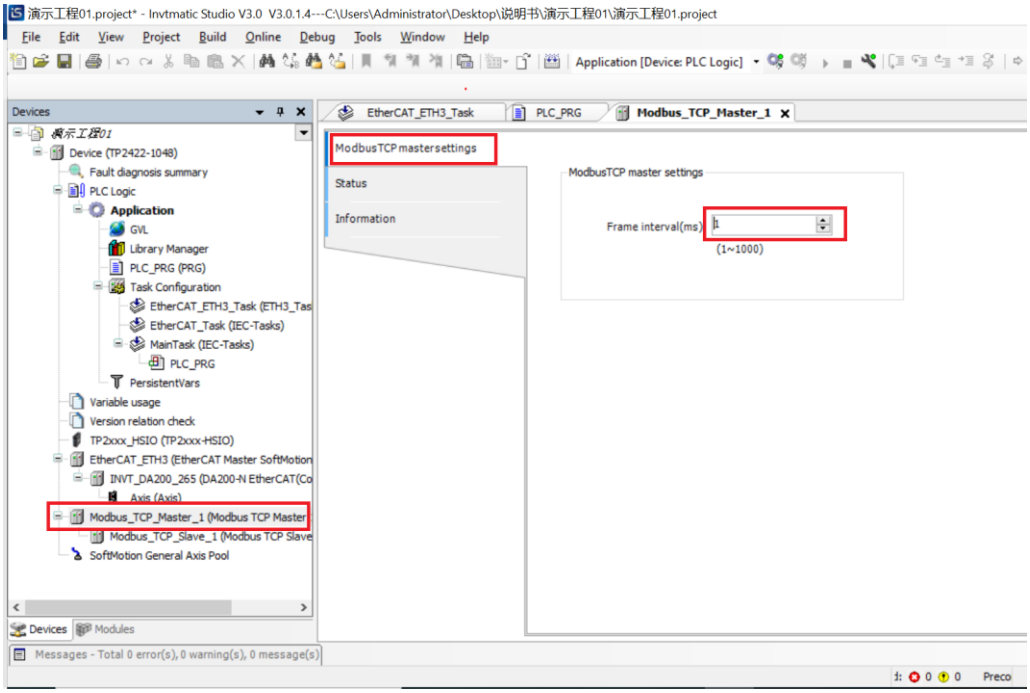


Table 5-1 Definition of the Number of Variables Accessible by Modbus TCP

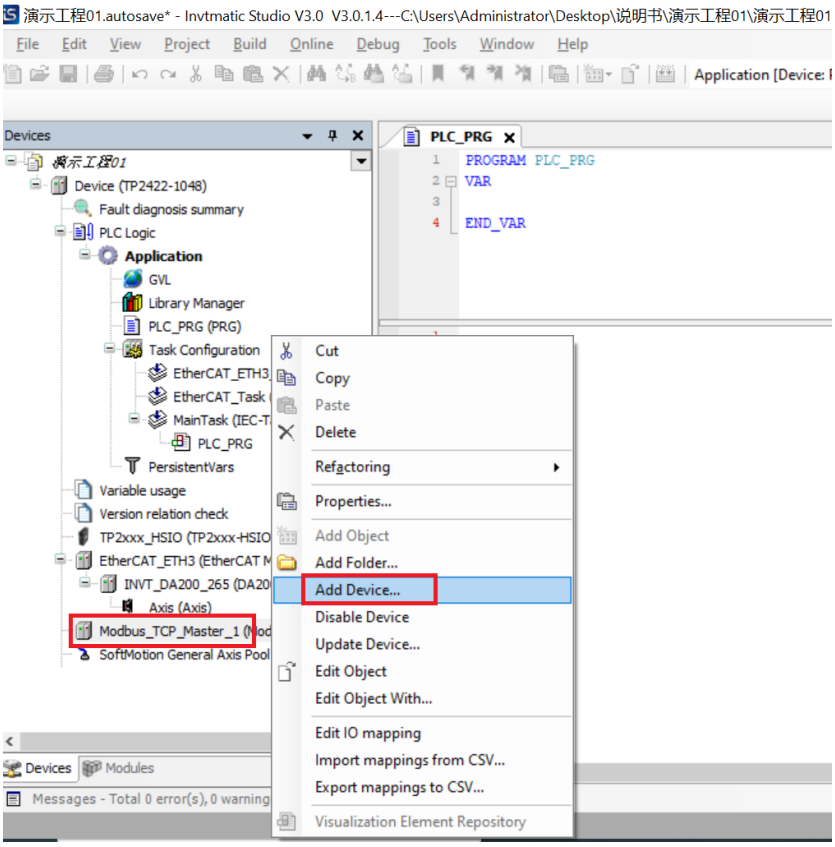
Variable	Qty.
Read coils (0x01)	1–2000 (0x7D0)
Read discrete coils (0x02)	1–2000 (0x7D0)
Read holding registers (0x03)	1–125 (0x7D)
Read input registers (0x04)	1–125 (0x7D)
Write single coil (0x05)	-
Write single register (0x06)	-
Write multiple coils (0x0F)	1–1968 (0x7B0)
Write multiple registers (0x10)	1–123 (0x7B)

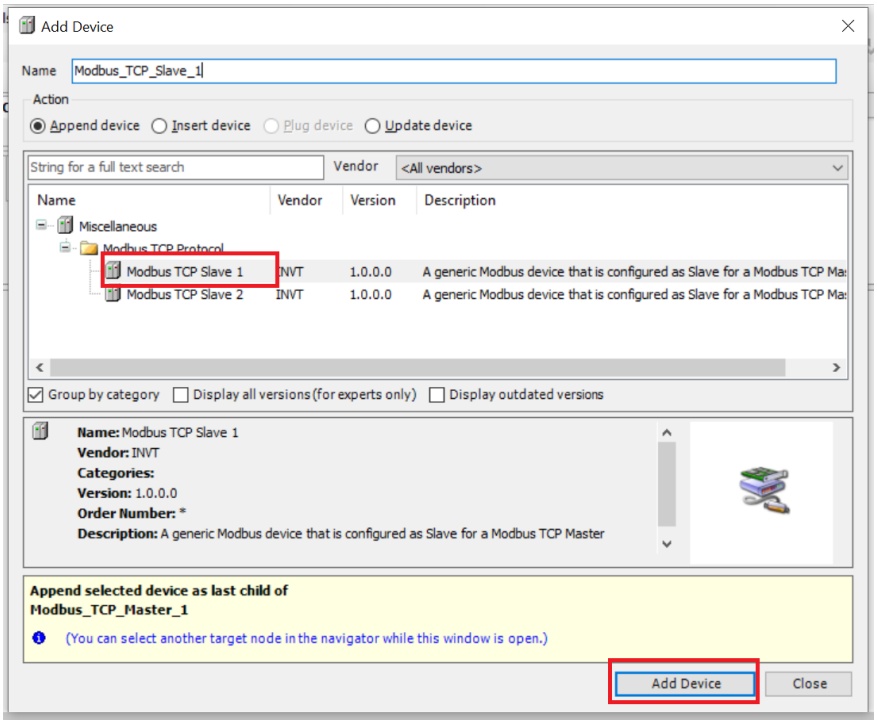
5.1.2 Modbus TCP Master Communication Configuration

■ Adding a Modbus TCP Slave

Right-click **Modbus_TCP_Master 1** in the left device tree, select **Add Device**, and then **Modbus TCP Slave 1** in the pop-up window, and click **Add Device** in the lower right corner.

Figure 5-3 Adding a Modbus TCP Slave





■ **Setting the Modbus TCP Slave**

Figure 5-4 Setting the Modbus TCP Slave

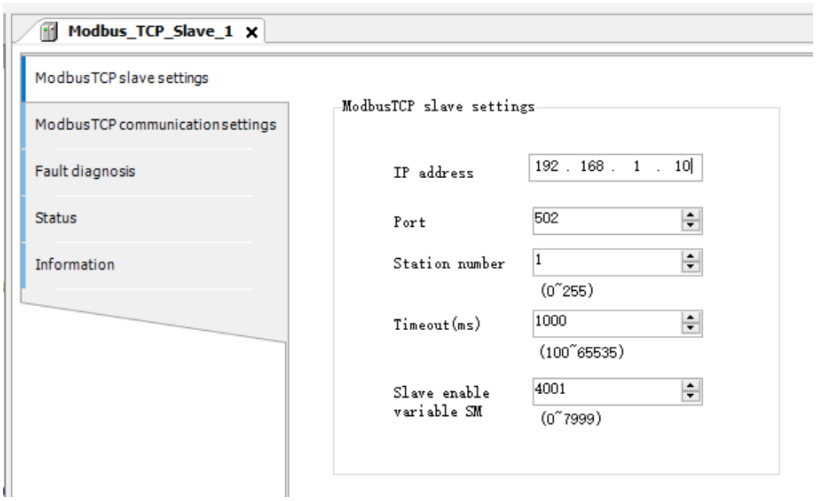


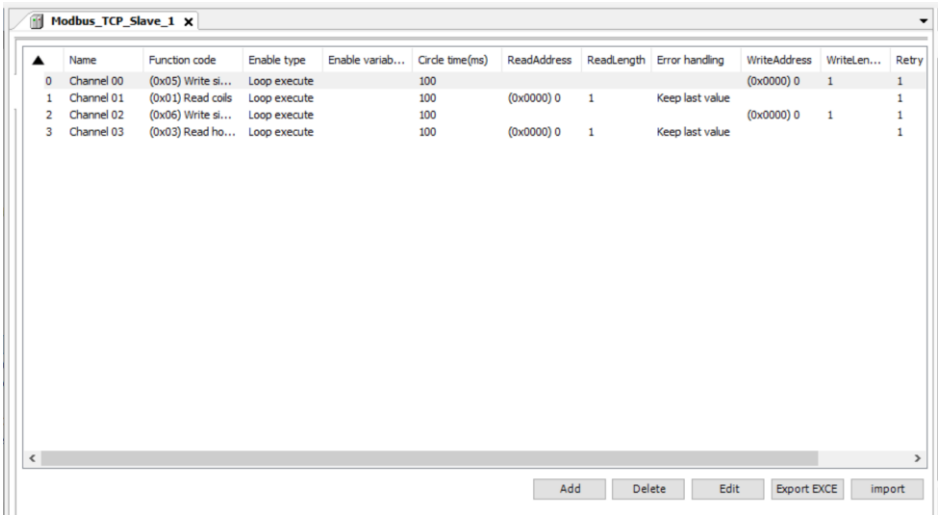
Table 5-2 Description of Modbus TCP Slave Parameter Settings

Parameter	Function
IP address	IP address for connecting the master to the Modbus TCP slave
Port	TCP port number for connecting the master to the Modbus TCP slave
Node number	Protocol node number for connecting the master to the Modbus TCP slave
Timeout period (ms)	After the master sends a frame, if the slave does not respond within the time period, the master reports a receive timeout.
Slave enable	After programming and enabling this variable, the master starts to send communication frames to the slave.

■ **Modbus TCP master to Modbus TCP slave communication settings**

Each channel represents an independent Modbus TCP request, as shown in Figure 5-5. The third line defines an operation of writing a single register (function code 0x06) at a cycle of 10 ms, i.e. writing 2 bytes of data to the register with an offset address of (0x0014)20 (1 register occupies 2 bytes).

Figure 5-5 Modbus TCP Master to Modbus TCP Slave Communication Settings



- Add: Click **Add** to open the dialog box for adding a new channel for the Modbus TCP slave, and then click **OK** to create a new channel.
- Edit: Select a channel in the Modbus TCP slave channel list and click **Edit** to open the dialog box for changing the channel configuration by modifying the parameters in it, and then click **OK** to update channel settings.
- Export to EXCEL: Click **Export EXCEL** to export channel parameters to an EXCEL table in batches, and then click **Import** to import channel parameters into the settings in batches.
- Copy: Select a channel in the Modbus TCP slave channel list. Right-click to open the context menu and select **Copy**, or press **Ctrl + C**, to copy the selected channel configuration to the clipboard.
- Paste: Select a channel in the Modbus TCP slave channel list. Right-click to open the context menu and select **Paste**, or press **Ctrl + V**, to paste the copied channel configuration into the data list.
- Delete: Select a channel in the Modbus TCP slave channel list. You can click **Delete**, right-click and select **Delete**, or press **Ctrl + D**. In the confirmation dialog box that appears, click **Yes** to delete the selected channel configuration.
- Import: Click **Import** to open the file selection window. Select the Excel file containing the channel configurations, then click **OK** to import the data into the list.

When adding or editing a channel, you will see the following pop-up dialog box:

Figure 5-6 Dialog Box for Modbus TCP Master to Modbus TCP Slave Communication Settings

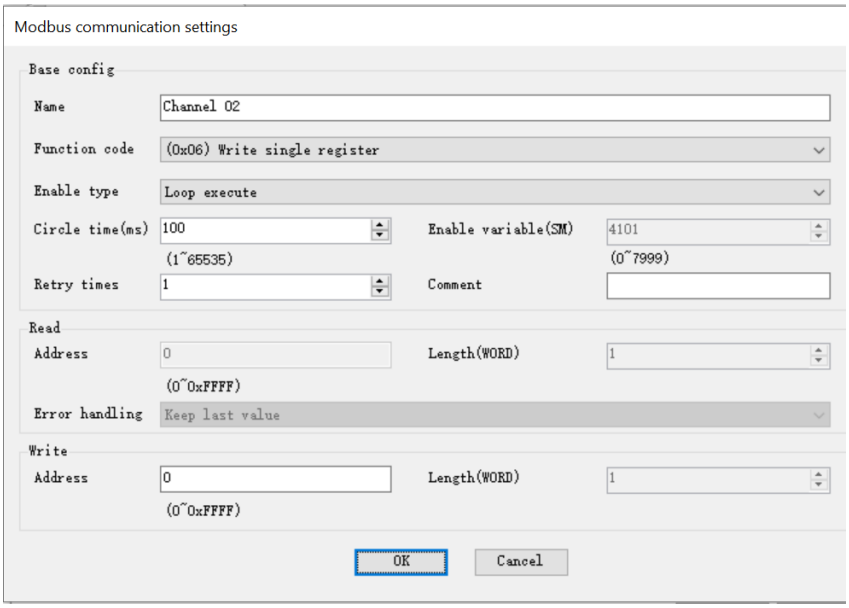


Table 5-3 Description of Modbus TCP Communication Parameter Settings

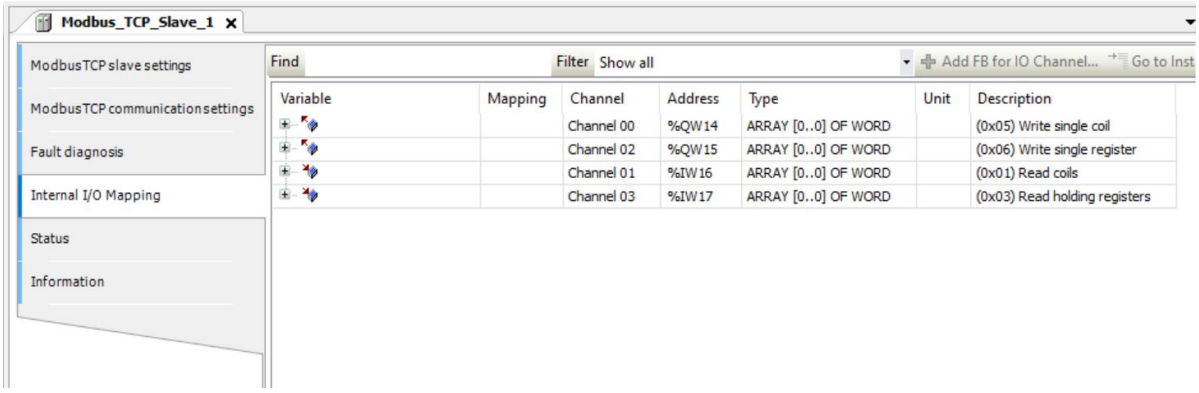
Parameter	Description
Name	A string that names the channel
Function code	Read coils (function code 0x01) Read input coils (function code 0x02) Read holding registers (function code 0x03) Read input registers (function code 0x04) Write single coil (function code 0x05) Write single register (function code 0x06) Write multiple coils (function code 0x0F) Write multiple registers (function code 0x10)

Parameter		Description
Enable type		Cyclic: requests triggered cyclically Cycle time: time for execution again Level trigger: triggered when programming changes Trigger variable (SM): SM element triggered (after enabling, the trigger variable is set after the communication is completed)
Circle time		The time required for the master device to complete one polling cycle for reading from and writing to all slave devices.
Enable variable		Once this variable is enabled in the program, the master begins sending communication frames to the slave.
Retry times		If a communication failure occurs and no slave return frame is obtained, resending is performed according to the retry times.
Comment		A short text area that describes the data
Read register	Starting address	The starting position of the register read
	Length	Number of registers read
	Error processing	Keep the last value: Keep the data at the last valid value Set to 0: Set all values to zero
Write register	Starting address	The starting position of the register written
	Length	The length of the register written

- **Internal I/O Mapping of Modbus TCP Slave**

After adding the master/slave communication configuration to the Modbus TCP slave communication settings, the mapping address of each configuration will be automatically assigned in the internal I/O mapping. For example, %IW98 in the third line of the figure below means that the read coil value is mapped to the address %IW98. In addition, you can also map custom variables in the program to I/O addresses by using the Input Assistant or by directly entering the example variable path.

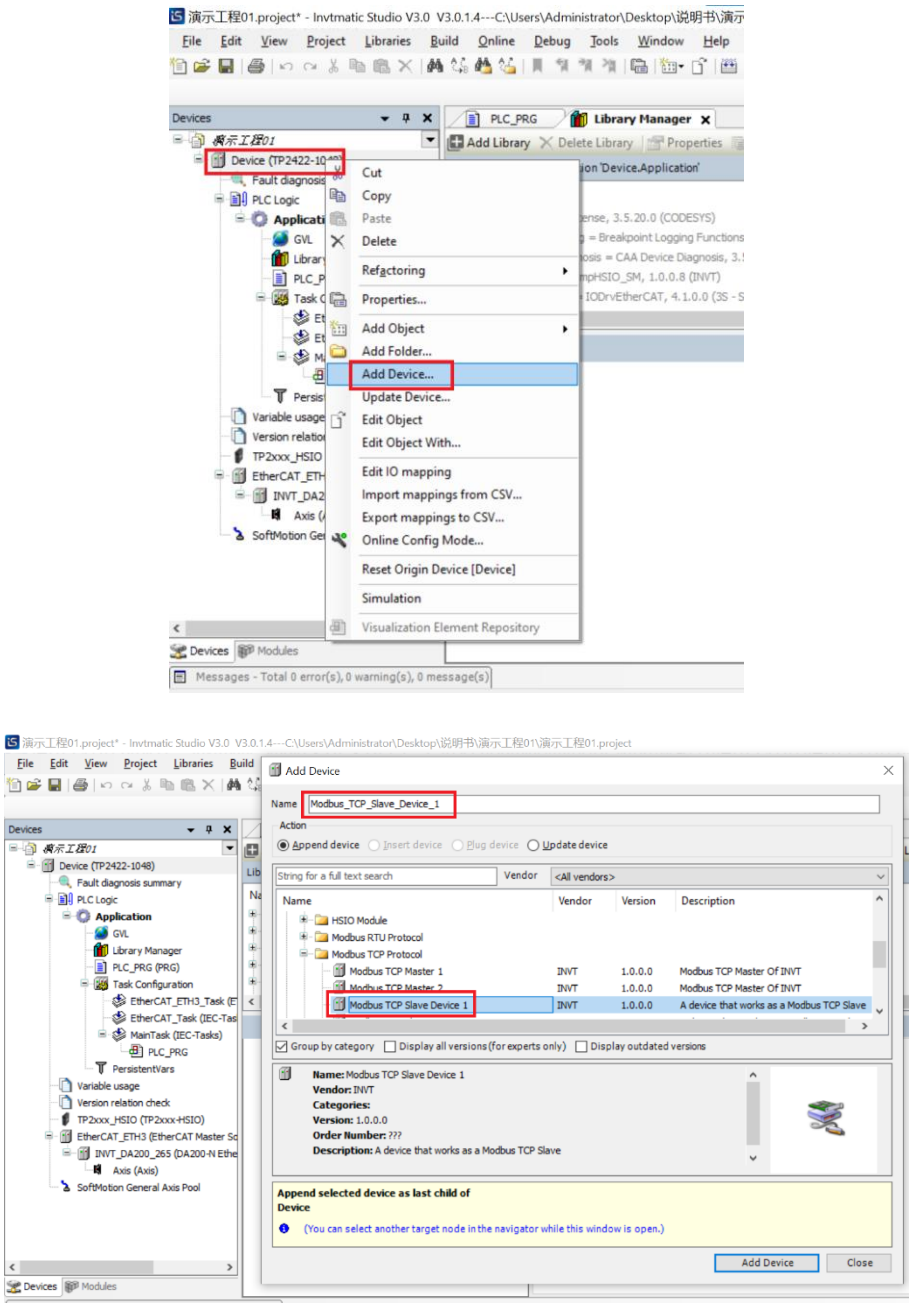
Figure 5-7 Internal I/O Mapping for Modbus TCP Master Connection to Slave



5.1.3 Modbus TCP Slave Configuration

When the PLC is used as a Modbus TCP slave, right-click **Device** in the left device tree, select **Add Device**, and then **Dedicated Device** > **Modbus TCP Slave Device1** in the pop-up window, and then click **Add Device** in the lower right corner.

Figure 5-8 Adding a Modbus TCP Slave



Double-click the slave device in the device tree to open the Modbus TCP slave configuration window, as shown in Figure 5-9.

Figure 5-9 Modbus TCP Slave Settings

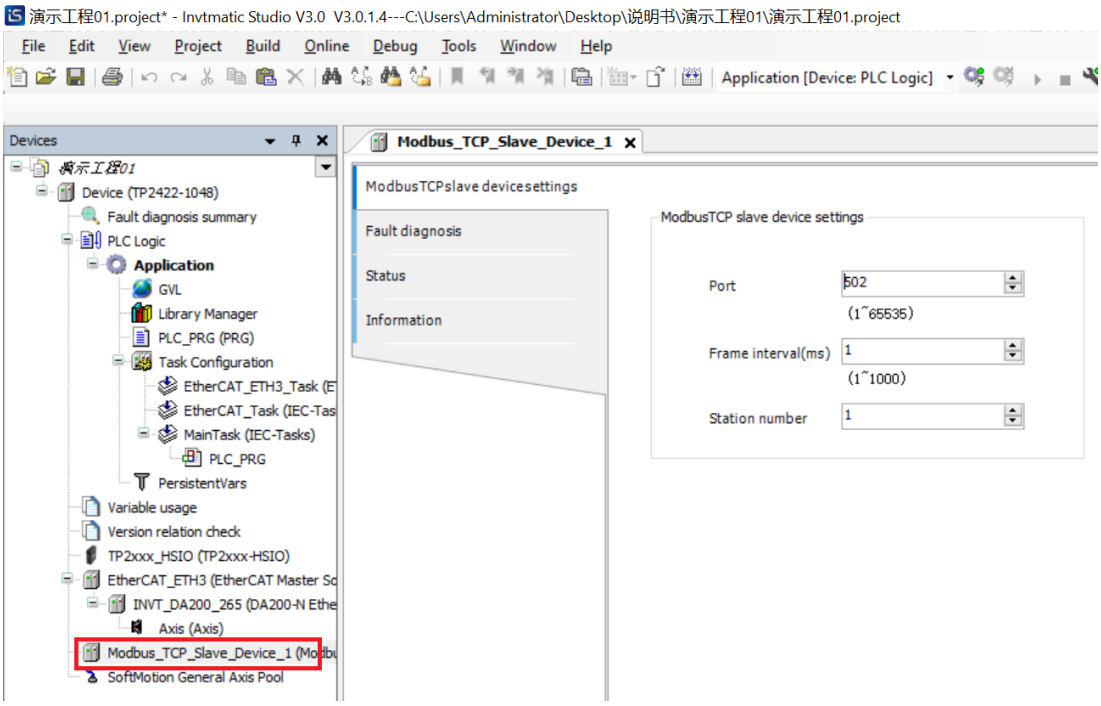


Table 5-4 Modbus TCP Slave Parameter Settings

Parameter	Function
Port	TCP port number of the Modbus TCP slave
Frame interval (ms)	The delay time for the Modbus TCP slave to send a response frame after receiving a communication frame
Node number	The number to identify the node

The Modbus_TCP_Slave defines storage areas accessible by external devices, which are shown in the following table.

Table 5-5 Modbus_TCP_Slave Register Address Ranges Accessible to the Master

TCP Master Function Code	Address name	Range	Offset from Standard Modbus Address
0x01	%QX	0–65535	None
0x05/0x15	%QX	0–65535	None
0x02	%IX	0–65535	None
0x04	%IW	0–65535	None
0x03	%MW	0–65535	None
0x06/0x16	%MW	0–65535	None

Table 5-6 Example of Correspondence between Controller Bit, Byte, Word, and Double Word

%MX	192.0–192.7	193.0–193.7	194.0–194.7	195.0–195.7
%MB	192	193	194	195
%MW	96		97	
%MD	48			

5.1.4 Modbus TCP Device Diagnosis

The Modbus TCP slave device diagnosis interface displays slaves and communication parameters with errors.

Figure 5-10 Modbus TCP Slave Device Diagnosis Interface

5.2 Modbus RTU

The PLC supports two Modbus serial port communications, namely COM1_RS485 and COM2_RS485, both of which support the standard Modbus RTU protocol and can be independently configured as a master or slave station, supporting seven baud rates: 2400, 4800, 9600, 19200, 38400, 57600, and 115200.

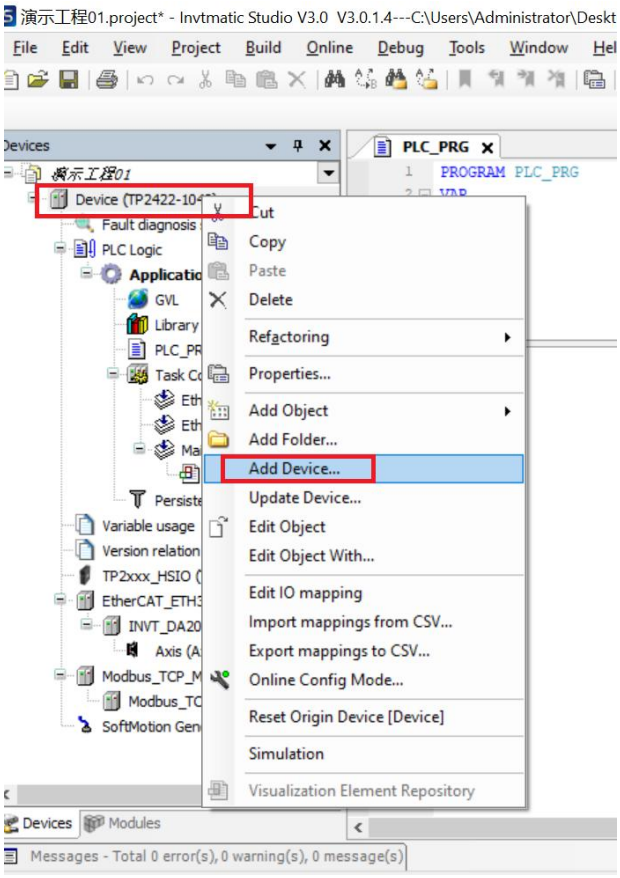
Table 5-7 Definition of the Number of Variables Accessible by Modbus RTU

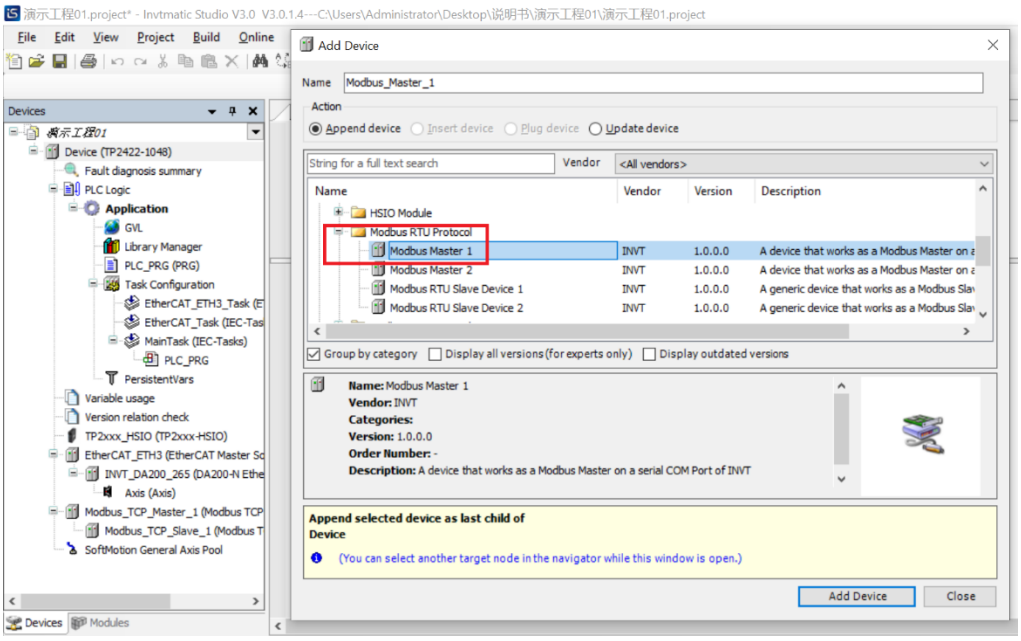
Variable	Qty.
Read coils (0x01)	1–2000 (0x7D0)
Read discrete coils (0x02)	1–2000 (0x7D0)
Read holding registers (0x03)	1–125 (0x7D)
Read input registers (0x04)	1–125 (0x7D)
Write single coil (0x05)	-
Write single register (0x06)	-
Write multiple coils (0x0F)	1–1968 (0x7B0)
Write multiple registers (0x10)	1–123 (0x7B)

5.2.1 Modbus RTU Master Configuration

When the PLC is used as a Modbus RTU master, right-click **Device** in the left device tree, select **Add Device**, and then **Dedicated Device** > **Modbus RTU Protocol** > **Modbus Master 1** in the pop-up window, and click **Add Device** in the lower right corner.

Figure 5-11 Adding a Modbus RTU Master





Open the Modbus master configuration window, as shown in the figure below.

Note: Normal communication is possible only when the Modbus master and slave communication parameters are consistent.

Figure 5-12 Modbus Master Settings

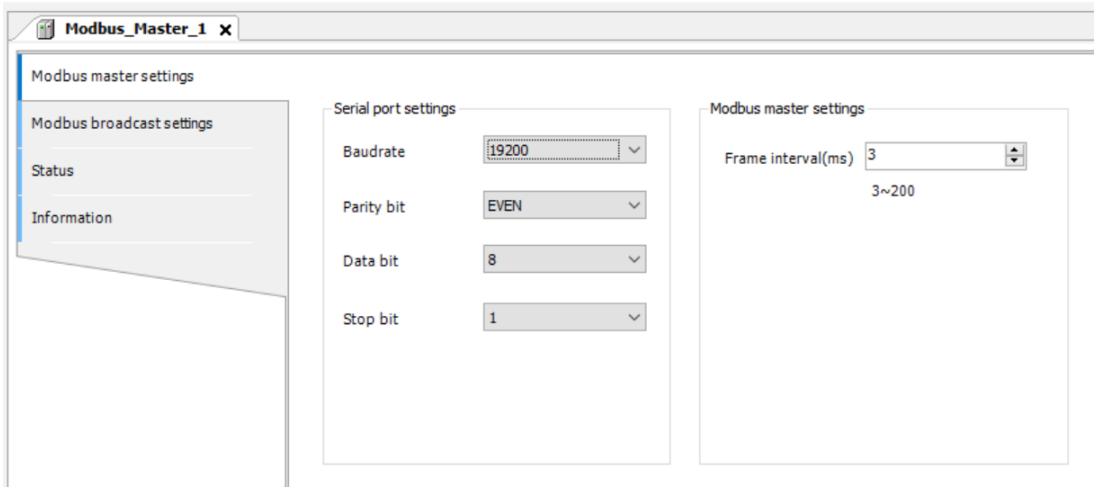


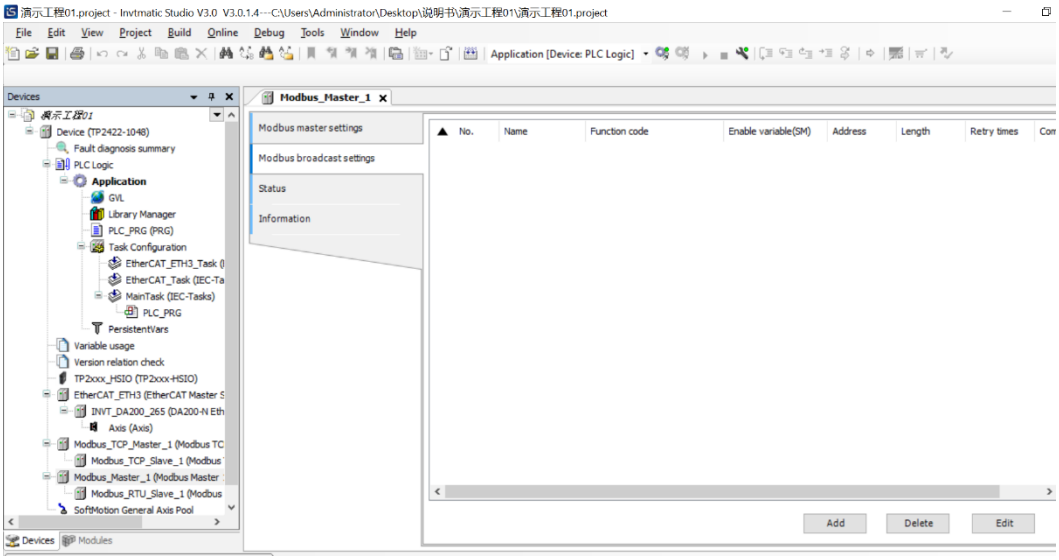
Table 5-8 Description of Modbus Master Parameter Settings

Parameter	Description
Baud rate	Communication rate: 2400, 4800, 9600, 19200, 38400, 57600, 115200
Check bit	Communication frame check mode: None Parity, Odd Parity, Even Parity
Data bit	The actual data bits contained in the communication frame
Stop bit	Identify the last bit of a single packet during communication
Frame interval (ms)	The time interval between the master receiving the last response data frame and the next request data frame

5.2.2 Modbus RTU Master Broadcast Configuration

After adding a Modbus RTU Master, double-click it to open the configuration interface, and click the **Modbus broadcast settings** interface to access the broadcast configurations. A maximum of 10 Modbus RTU broadcasts can be configured.

Figure 5-13 Modbus RTU Master Broadcast Settings



- Add: You can add an item by clicking the **Add** button, right-clicking within the broadcast settings list and selecting **Add** from the context menu, or pressing **Ctrl + I**. In the dialog box that appears, configure the Modbus broadcast settings, and click **OK** to add a new Modbus RTU broadcast.

- Edit: You can edit a broadcast by selecting it from the list and clicking the **Edit** button, right-clicking the broadcast and selecting **Edit** from the context menu, or pressing **Ctrl + E**. In the dialog box that appears, edit the Modbus broadcast settings as needed and click **OK** to save the changes.
- Delete: You can delete a broadcast by selecting a broadcast from the list and clicking the **Delete** button, or right-clicking the broadcast and selecting **Delete**. In the confirmation dialog box that appears, click **Yes** to delete the selected broadcast.
- Copy: Right-click a broadcast and select **Copy**, or press **Ctrl + C**, to copy the selected broadcast information to the clipboard.
- Paste: Right-click in the list and select **Paste**, or press **Ctrl + V**, to paste the copied broadcast information into the list.

Table 5-9 Modbus Broadcast Settings

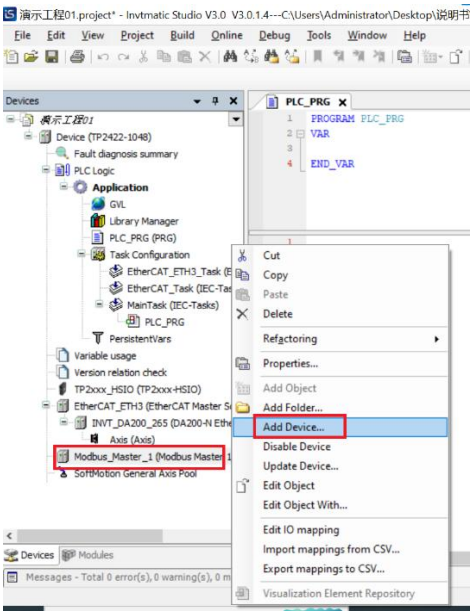
Parameter	Description
Name	Modbus broadcast name
Function code	Write single coil (function code 0x05) Write single register (function code 0x06) Write multiple coils (function code 0x0F) Write multiple registers (function code 0x10)
Enable Variable	Once this variable is enabled in the program, the master begins sending communication frames to the slave.
Address	The starting register address for the write operation.
Length	The number of registers to write.
Retry times	If a communication fault occurs and no response frame is received from the slave, the current frame will be resent according to the configured retry count.
Comment	A short text area for describing the data.

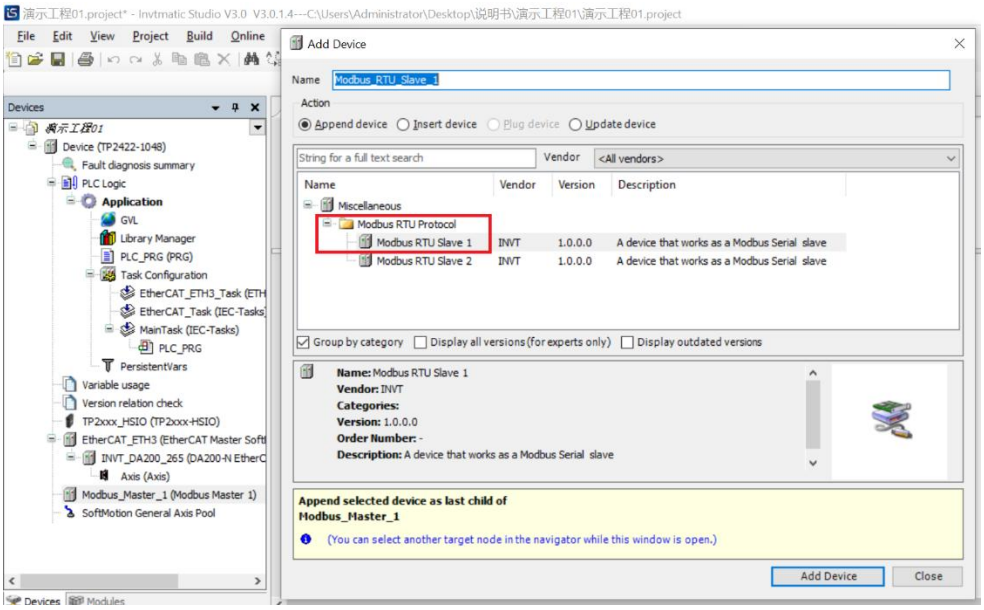
5.2.3 Modbus RTU Master Communication Configuration

■ Adding a Modbus Slave

When the PLC is used as a Modbus master, right-click **Modbus Master 1** in the left device tree, select **Add Device**, and then **Dedicated Device** > **Modbus RTU Protocol** > **Modbus RTU Slave1** in the pop-up window, and click **Add Device** in the lower right corner.

Figure 5-14 Adding a Modbus RTU Slave When the Port Is Used as a Master





■ Setting the Modbus Slave

Double-click the slave device in the device tree to open the Modbus slave configuration window as shown in Figure 5-15.

Figure 5-15 Modbus RTU Slave Settings When the Port Is Used as a Master

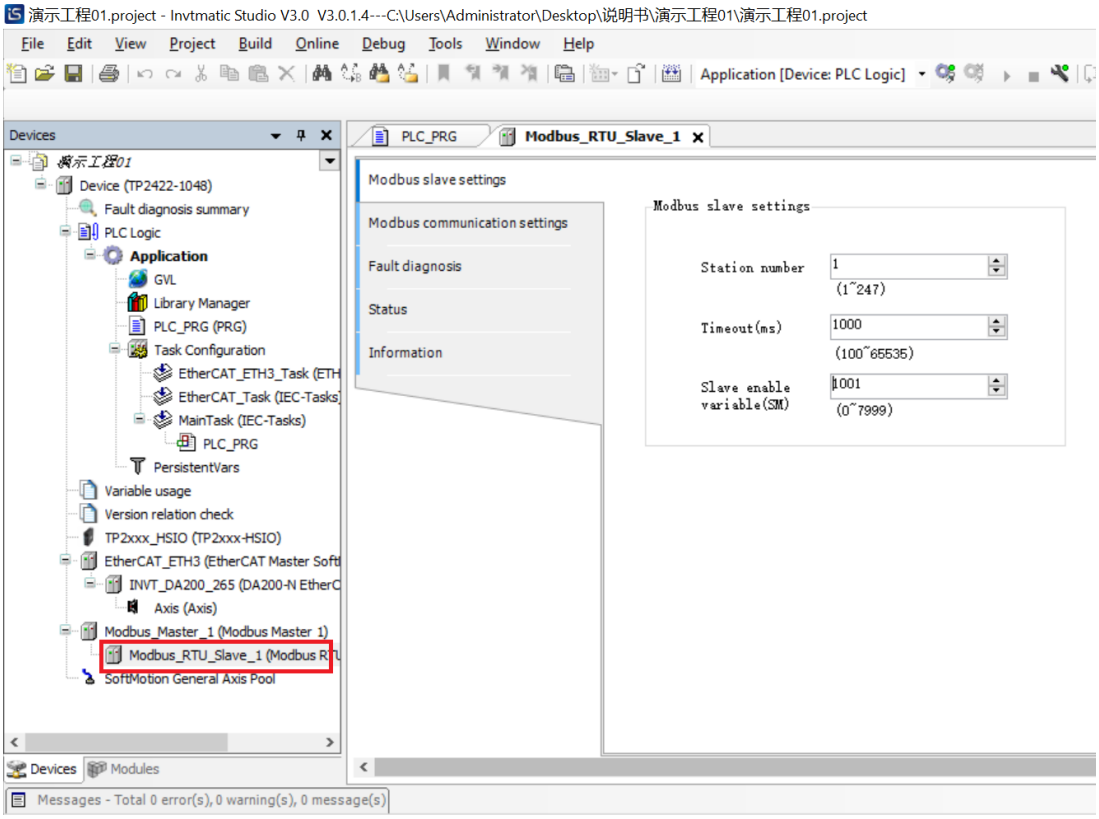


Table 5-10 Description of Modbus RTU Slave Parameter Settings When the Port Is Used as a Master

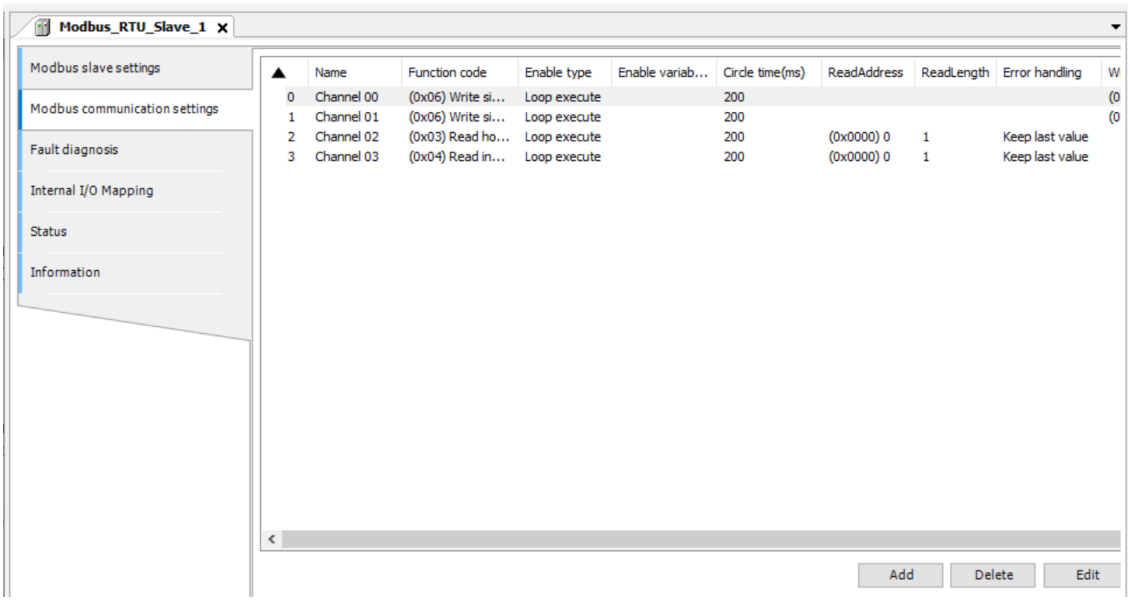
Variable	Function
Node number	The number to identify the salve, ranging from 1 to 247
Timeout period (ms)	After the master sends a frame, if the slave does not respond within the time period, the master reports a receive timeout.
Slave enable (SM)	After programming and enabling this variable, the master starts to send communication frames to the slave.

■ **Modbus master to Modbus slave communication settings**

Switch to the Modbus slave communication settings window and add a Modbus slave communication configuration. The configuration table supports up to 60 configurations.

Each channel represents an independent Modbus request, where the first line defines a request to execute the action of writing a single register cyclically (function code 0x06), i.e. writing one word of data to a register at an offset of 0x2000.

Figure 5-16 Modbus Slave Communication Settings When the Port Is Used as a Master



- Add: Click **Add** to open the dialog box for adding a new channel for the Modbus slave. Click **OK** to create a new channel.
- Edit: Select a channel in the Modbus slave channel list and click **Edit** to open the dialog box for changing the channel configuration by modifying the parameters in it, and then click **OK** to update channel settings.
- Export to EXCEL: Click **Export to EXCEL** to export channel parameters to an EXCEL table in batches, and then click **Import** to import channel parameters into the settings in batches.
- Copy: Select a channel in the Modbus TCP slave channel list. Right-click to open the context menu and select **Copy**, or press **Ctrl + C**, to copy the selected channel configuration to the clipboard.
- Paste: Select a channel in the Modbus TCP slave channel list. Right-click to open the context menu and select **Paste**, or press **Ctrl + V**, to paste the copied channel configuration into the data list.
- Delete: Select a channel in the Modbus TCP slave channel list. You can delete the channel by clicking

Delete, right-clicking to open the context menu and selecting **Delete**, or pressing **Ctrl + D**. In the confirmation dialog box that appears, click **Yes** to delete the selected channel configuration.

- Import: Click **Import** to open the file selection window. Select the Excel file containing the channel configurations, then click **OK** to import the data into the list.

When adding or editing a channel, you will see the following pop-up dialog box:

Figure 5-17 Dialog Box for Modbus Slave Communication Settings When the Port Is Used as a Master

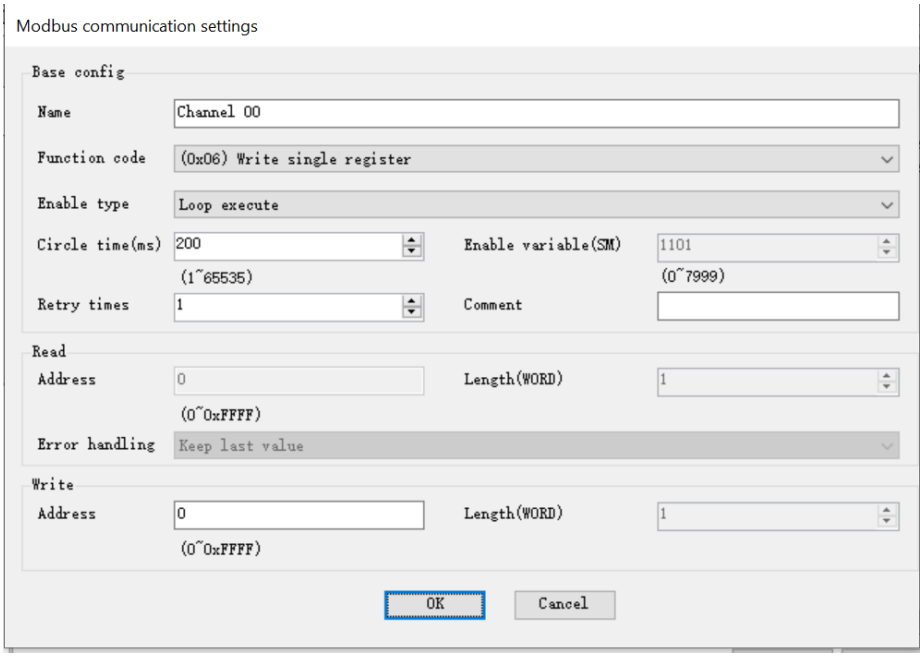


Table 5-11 Description of Modbus Communication Parameter Settings

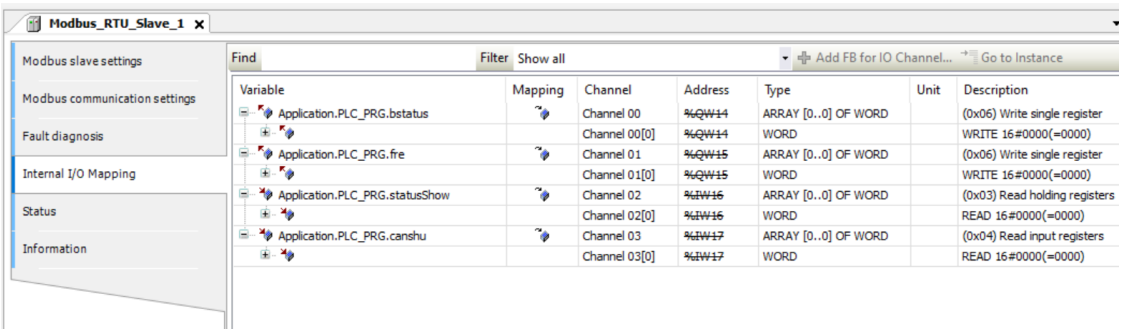
Parameter	Description
Name	A string that names the channel
Function code	Read coils (function code 0x01) Read input coils (function code 0x02) Read holding registers (function code 0x03) Read input registers (function code 0x04) Write single coil (function code 0x05) Write single register (function code 0x06) Write multiple coils (function code 0x0F) Write multiple registers (function code 0x10)
Enable type	Cyclic: requests triggered cyclically Cycle time: time for execution again Level trigger: triggered when programming changes Trigger variable (SM): SM element triggered. After triggered successfully, the element will be automatically reset.
Circle time	The time required for the master device to complete one polling cycle for reading from and writing to all slave devices.
Enable Variable	Once this variable is enabled in the program, the master begins sending communication frames to the slave.

Parameter		Description
Retry times		If a communication failure occurs and no slave return frame is obtained, resending is performed according to the retry times.
Comment		A short text area that describes the data
Read register	Starting address	The starting position of the register read
	Length	Number of registers read
	Error processing	Keep the last value: Keep the data at the last valid value Set to 0: Set all values to zero
Write register	Starting address	The starting position of the register written
	Length	The length of the register written

■ **Internal I/O Mapping of Modbus Slave**

After adding the master/slave communication configuration to the Modbus slave communication settings, the mapping address of each configuration will be automatically assigned in the internal I/O mapping. For example, %QW1 in the first line of Figure 5-18 means that the read coil value is mapped to the address %QW1. In addition, you can also map custom variables in the program to I/O addresses by using the Input Assistant or by directly entering the example variable path.

Figure 5-18 Internal I/O Mapping of Modbus Slave When the Port Is Used as a Master



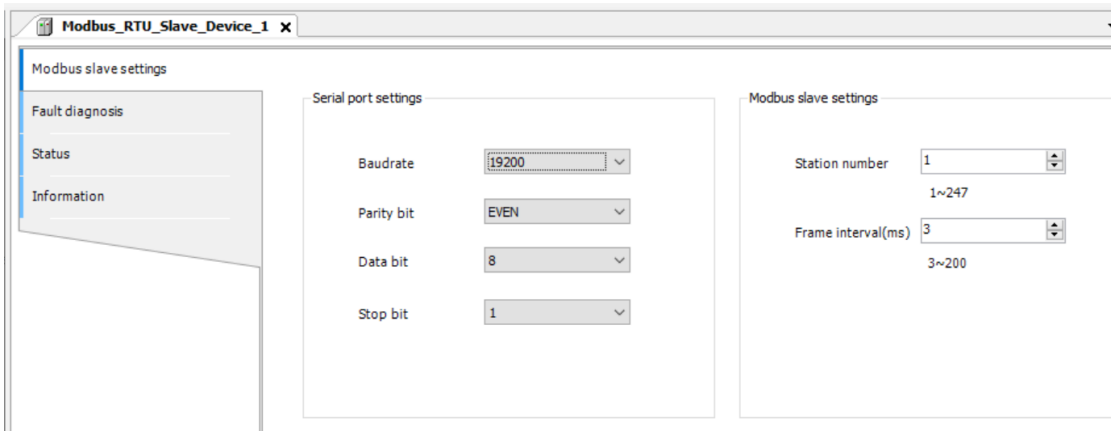
5.2.4 Modbus RTU Slave Configuration

When the PLC is used as a Modbus slave, right-click **Device** in the left device tree, select **Add Device**, and then **Dedicated Device** > **Modbus RTU Protocol** > **Modbus RTU Slave Device1** in the pop-up window, and click **Add Device** in the lower right corner.

Double-click the slave device in the device tree to open the Modbus slave configuration window as shown in Figure 5-19.

In the Modbus slave parameter settings, the serial port configuration has the same meaning as that of the Modbus master. The node number in the Modbus slave configuration refers to the node number of this device; the frame interval refers to the delay time for response to the master after receiving the communication frame sent by the master.

Figure 5-19 Configuration When Modbus RTU Is Used as a Slave



Note: Normal communication is possible only when the Modbus master and slave communication parameters are consistent.

The Modbus_RTU_Slave defines storage areas accessible by external devices, which are shown in Table 5-12.

Table 5-12 Modbus_RTU_Slave Register Address Ranges Accessible to the Master

RTU Master Function Code	Address name	Range	Offset from Standard Modbus Address
01	%QX	0–65535	None
05/15	%QX		
02	%IX		
04	%IW		
03	%MW		
06/16	%MW		

5.2.5 Modbus RTU Device Diagnosis

The Modbus master device diagnosis interface displays slaves and communication parameters with errors.

Figure 5-20 Modbus Master Device Diagnosis

5.2.6 Common Faults of Modbus RTU

The main faults that occur when the Modbus master is connected to the slave are as follows:

- The configurations of the Modbus master and slave are inconsistent, resulting in the inability to establish communication between the master and the slave.
- The Modbus master accesses an illegal address of the Modbus slave and an error response is returned.

- The Modbus master operates the Modbus slave to write a register, but the Modbus slave only supports read actions on this register. The Modbus master will receive an error response returned by the Modbus slave.

Error response frame format: slave address + (instruction code + 0x80) + error code + CRC check.

Note: This error frame applies to all operation instruction frames.

Table 5-13 Description of Modbus Error Response Frames

Number	Data (Byte) Meaning	Number of Bytes	Description
1	Slave address	1 byte	Value range: 1–247
2	Instruction code + 0x80	1 byte	Error instruction code
3	Error code	1 byte	1–4

5.3 EtherCAT Master

EtherCAT (Ethernet for Control Automation Technology) is an open-architecture, Ethernet-based fieldbus system. Compared with other fieldbuses, EtherCAT has the characteristics of good performance, high device synchronization accuracy, flexible topology, easy application, and low cost. Currently, more and more devices use the EtherCAT bus for communication.

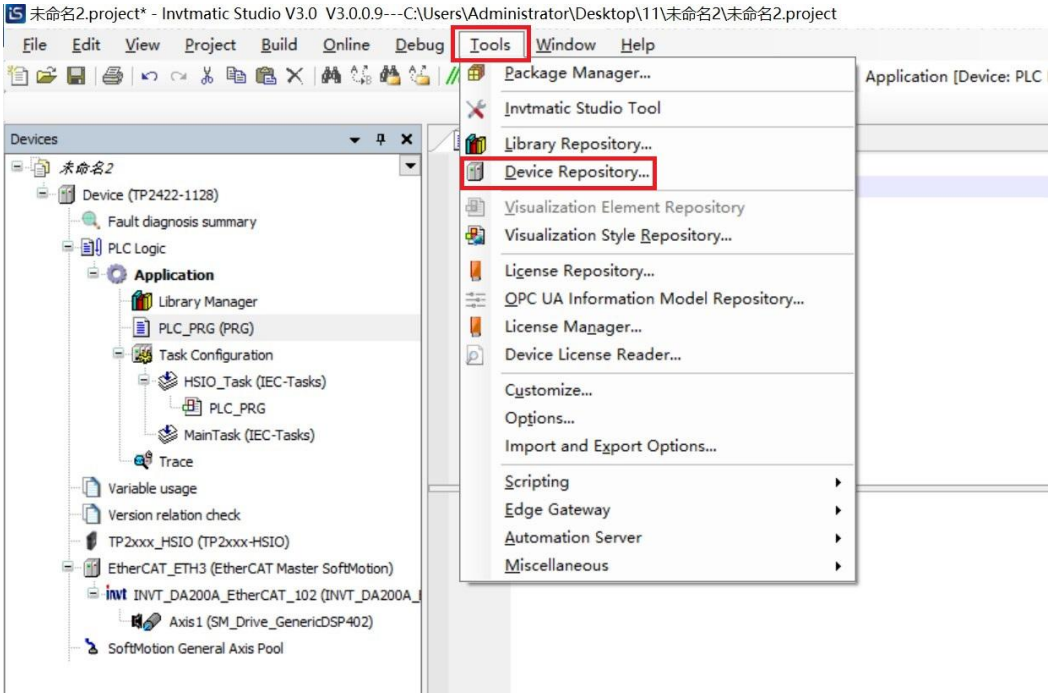
5.3.1 EtherCAT Master Configuration

5.3.1.1 Adding a Device Profile

INVT EtherCAT slave devices have been pre-added in Invtmatic Studio. If you use an INVT product, you can skip this step directly. When using an EtherCAT slave without a device profile added to the software, you need to add the corresponding profile to the software. The operation steps are as follows:

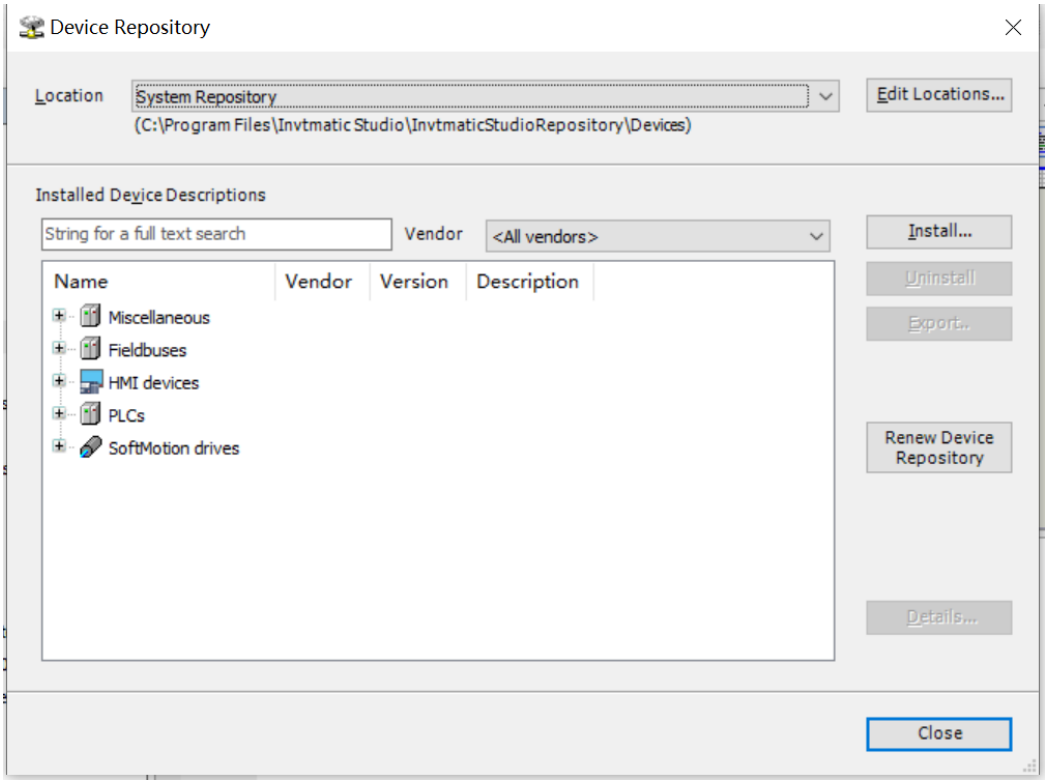
Step 1 Open Invtmatic Studio and select **Tools** > **Device Repository** in the Toolbar.

Figure 5-21 Selecting a Device Repository



Step 2 In the pop-up window, click **Install**.

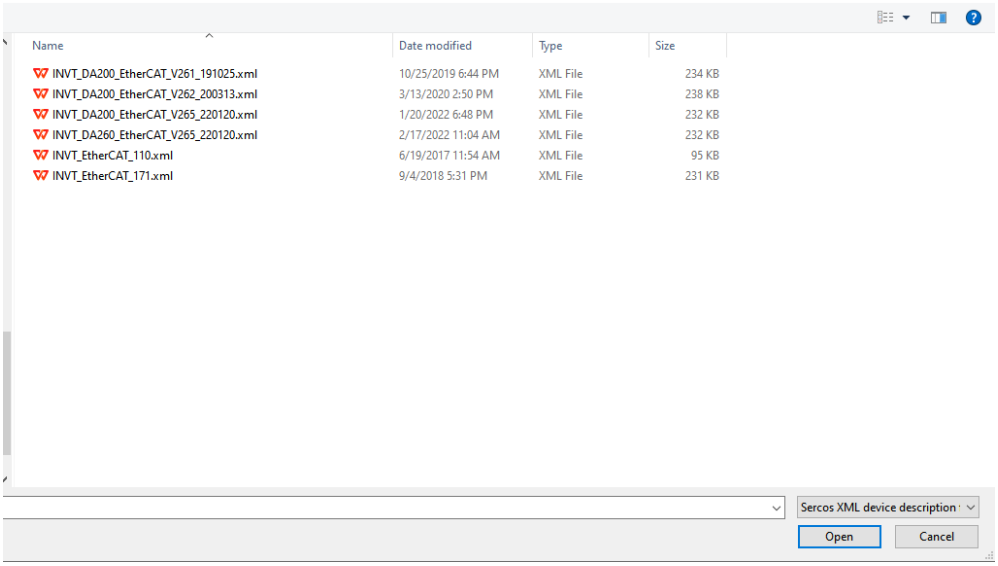
Figure 5-22 Installing the Device Repository

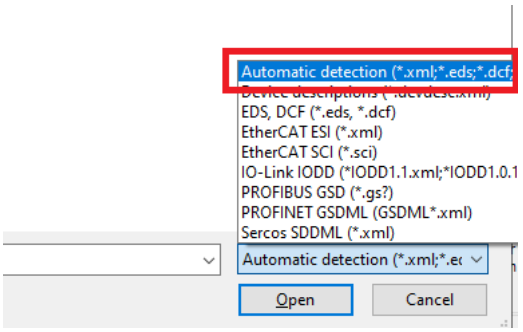


Step 3 In the pop-up window, find the path where the device profile is stored.

Note: Here, set the file type to **All supported description files**. If you select other file types, you will not be able to select the XML file normally. After finding the corresponding XML, click **Open**.

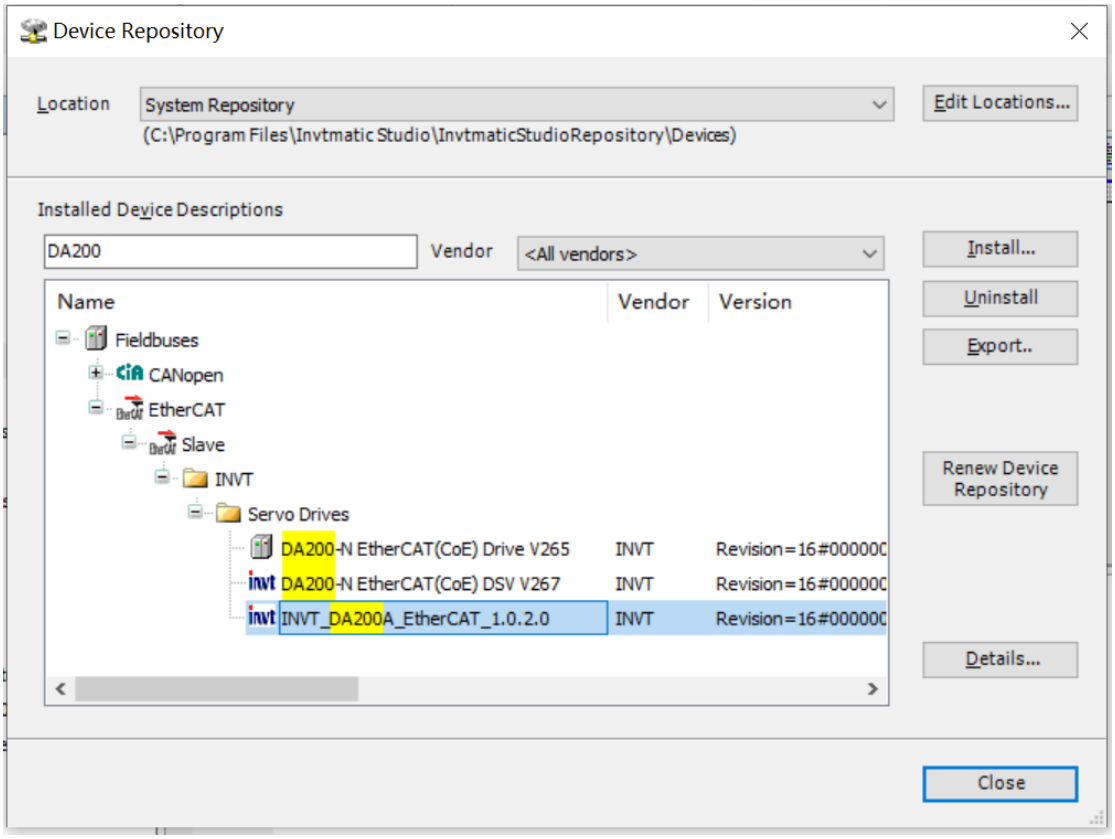
Figure 5-23 Selecting Device Profile Path and Type





Step 4 After opening the file, the software will automatically import the device. At this time, you can observe the information output box and confirm that the installation is complete. Then, Click **Close**.

Figure 5-24 Importing the Device Profile into the Device



5.3.1.2 Slave Configuration

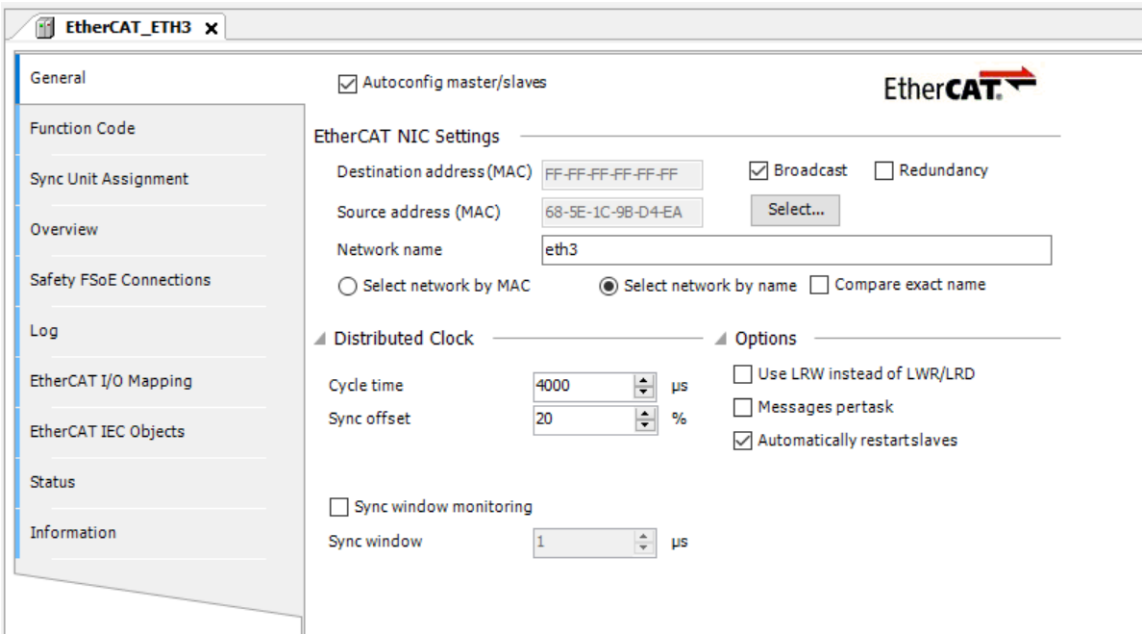
In the Invtmatic Studio software, you can configure the EtherCAT slave through two methods: automatic scanning and manual addition. For detailed operation steps, see section 2.3 Examples of Program Writing and Debugging. When adding an EtherCAT slave by automatic scanning, you must first ensure that the slave device hardware used in the project is correctly connected to the PLC. Only when normal communication is possible can the PLC scan the EtherCAT slave device.

5.3.1.3 Master Settings

You can set the EtherCAT master parameters on the EtherCAT Master Settings interface. As shown in Figure 5-25, you need to set the cycle and synchronization offset of the distributed clock.

- Cycle: The interval for sending EtherCAT data frames, which must be the same as the cycle time of the EtherCAT task.
- Synchronization offset: The percentage of the relative offset of the EtherCAT task relative to the Sync0 interrupt of the slave, which is 20% by default, and cannot be modified generally.

Figure 5-25 Setting Master Cycle and Synchronization Offset



In the DC mode of EtherCAT communication, after parsing the data transmitted by the master, the slave also needs to process the data in the DC interrupt. Since the time it takes for the master to transmit data to each slave varies, in order to ensure data synchronization, there must be enough time between the data frame and the synchronization interrupt for the slave to receive and process the data. This period of time is the offset time.

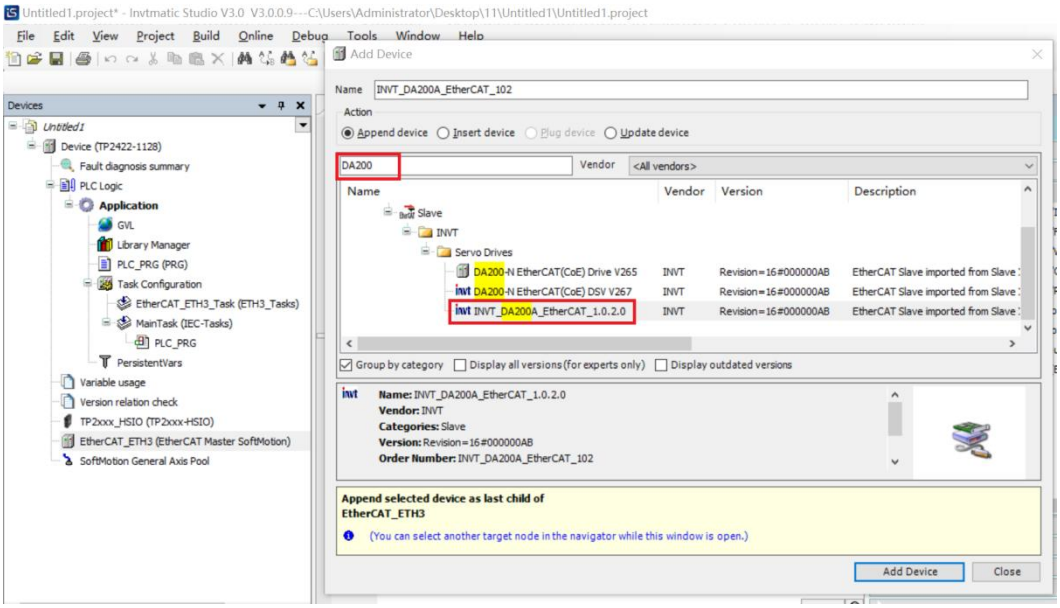
Note: When using it, you need to add the master name. For example, if the master name is EtherCAT_Master_SoftMotion, then when restarting the master, the variable that needs to be triggered is EtherCAT_Master_SoftMotion.xRestart. Common parameters of the EtherCAT master are listed in Table 5-14.

Table 5-14 Common Parameters of EtherCAT Master

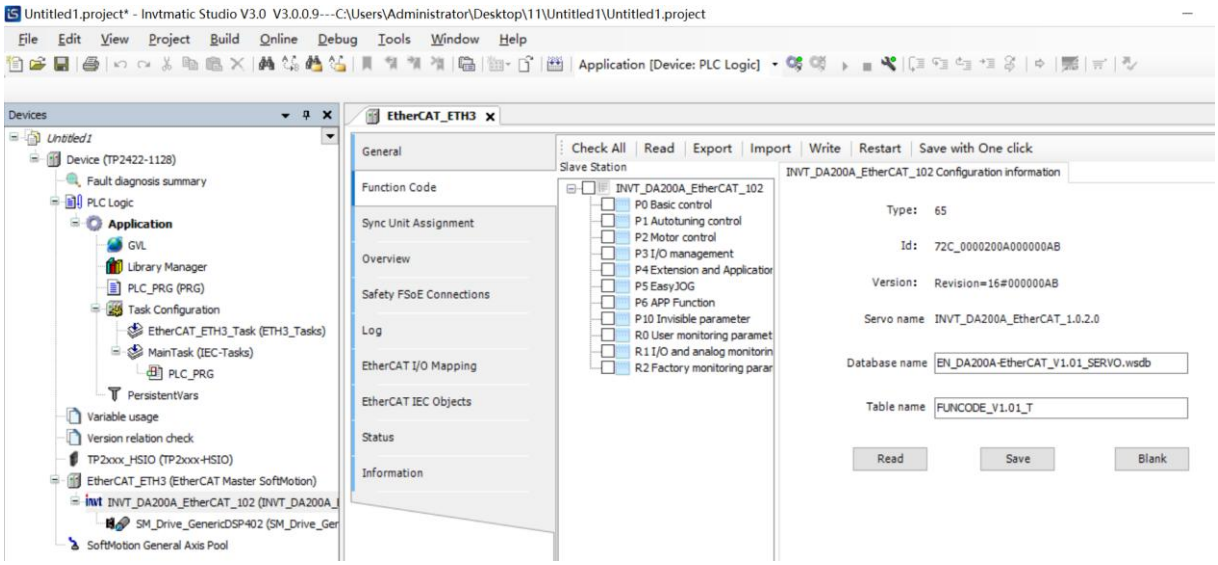
Variable Name	Function
xRestart	Rising edge trigger, restarted after being triggered
EtherCAT	Master
xConfigFinished	Master configuration completed
xDistributedClockInSync	EtherCAT distributed clock synchronization
xError	Error in the EtherCAT master

5.3.1.4 Function Codes

Add a servo drive on the EtherCAT master configuration interface.



On the **Function Code** interface, you can read and write servo function codes, and save basic configurations.



- Check All: Click **Check All** to select all checkboxes in the slave tree, specifying the function codes for read/write operations.
- Read: Click **Read** to read the current values of the selected function codes.
- Export: Click **Export** to export the information of the selected function codes to an Excel file.
- Import: Click **Import**. In the dialog box that appears, select the function code Excel file and click **OK** to import the values into the servo device and update the data list.
- Write: Modify the function codes and current values (which require a restart to take effect), then click **Write** to save the values to the servo device's EEPROM.
- Restart: Click **Restart** to restart the selected servo device.

- Save with one click: Click **Save with One Click** to save the read or modified values of the selected function codes as historical data and store the file in the PLC. This allows the HMI to call the function block to restore the function code configuration with one click.

5.3.2 EtherCAT Slave Configuration

5.3.2.1 General Settings

■ **EtherCAT address**

The configuration address of the EtherCAT slave refers to the sequential address of the slave device in the Invtmatic Studio device tree, starting from 1001 and increasing in sequence, as shown in Figure 5-26. You can find the EtherCAT address of the slave on the Slave Settings interface. This address will be used when the SDO read/write function block is called, as shown in Figure 5-27.

Figure 5-26 EtherCAT Slave Address

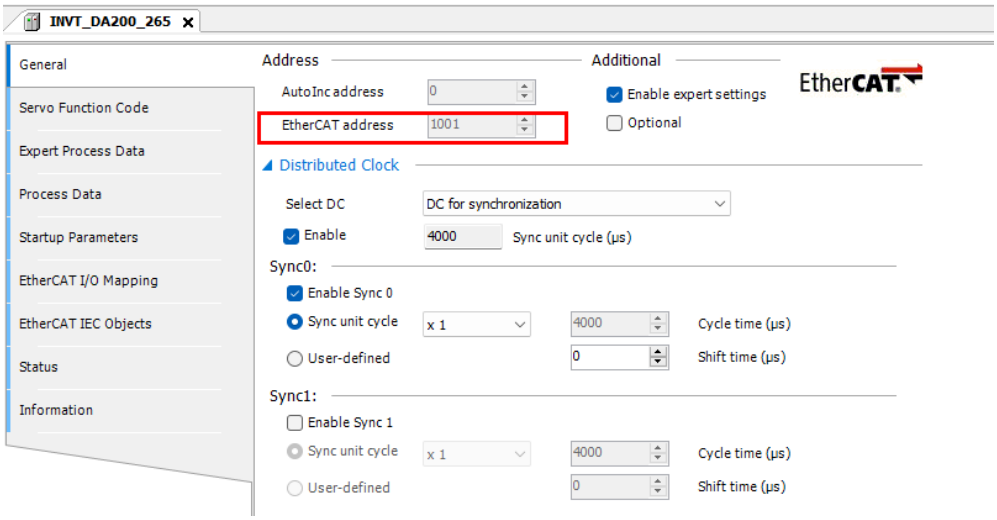
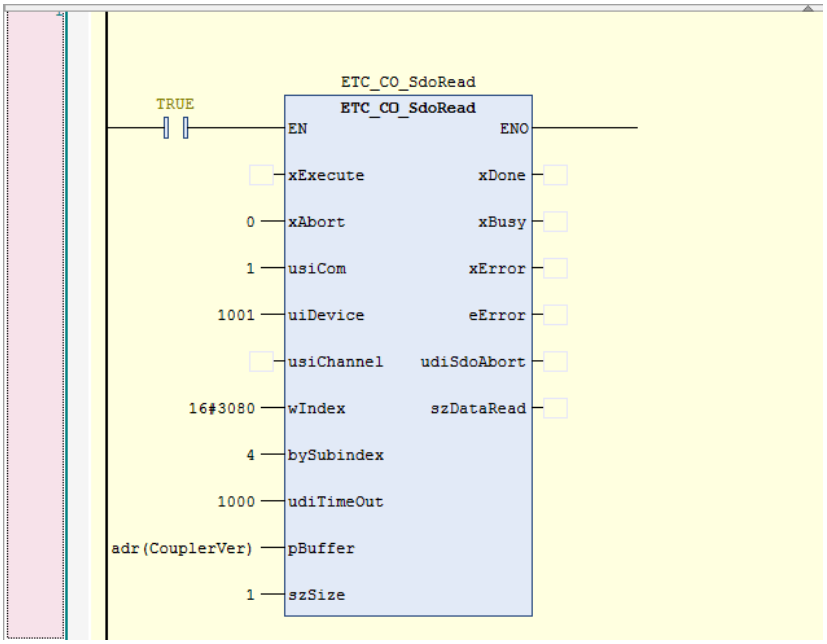


Figure 5-27 SDO Read Function Block

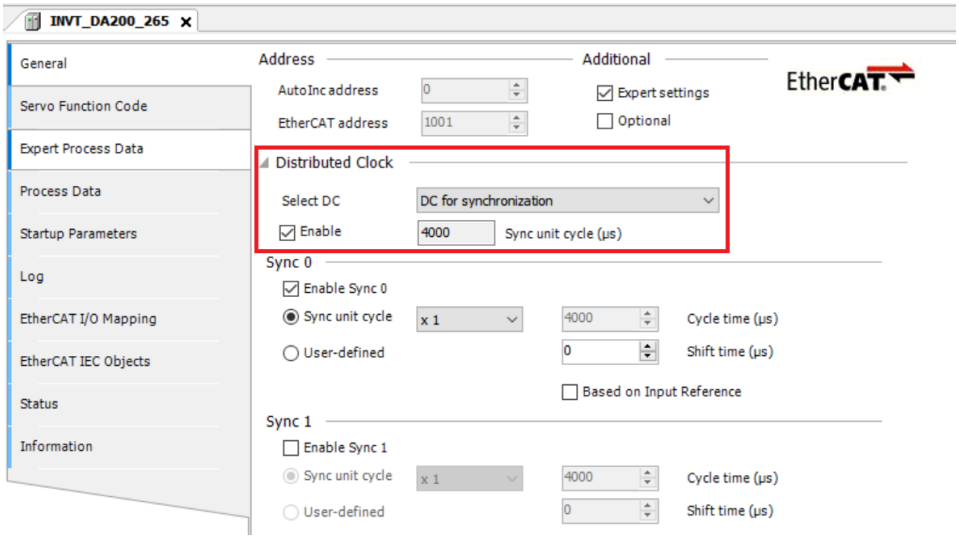


■ Distributed clock

This option is used to set the synchronous running mode of the slave. There are three synchronization modes: freewheeling (FreeRun), synchronization with an input/output event (SM-Synchron), and synchronization with a distributed clock (DC-Synchron).

The synchronization mode options supported vary depending on the selected slave. Generally, you do not need to modify this option, as long as the synchronization unit cycle in DC-Synchron mode is consistent with the EtherCAT task cycle, as shown in Figure 5-28.

Figure 5-28 Distributed Clock Parameters

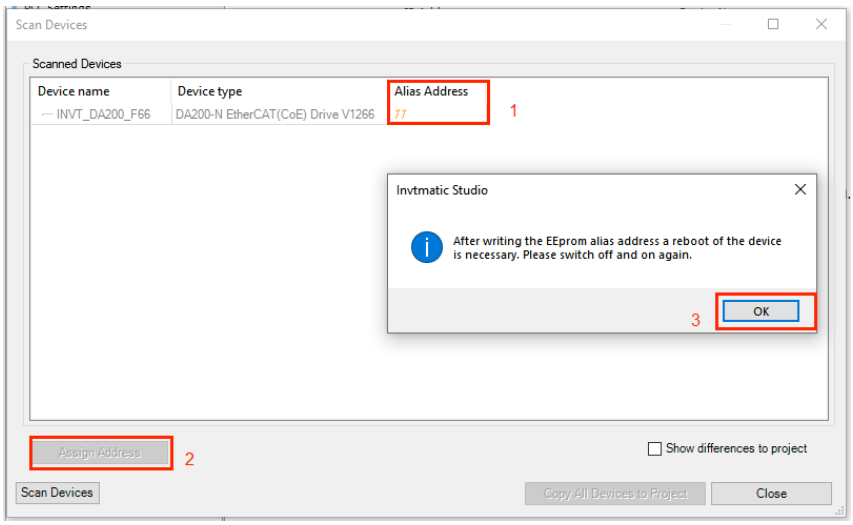


■ Node alias

When EtherCAT addresses are used, if the actual device connection order is inconsistent with the configuration order, the bus will not operate normally. If you hope that the actual slave connection order will not be affected by the configuration order, you can use the alias function to rename the servo. During the connection process, the servo is no longer identified by the automatically assigned node address, but by the name. To use the node alias function correctly, you need to set the node alias of the slave first. The operation steps are as follows:

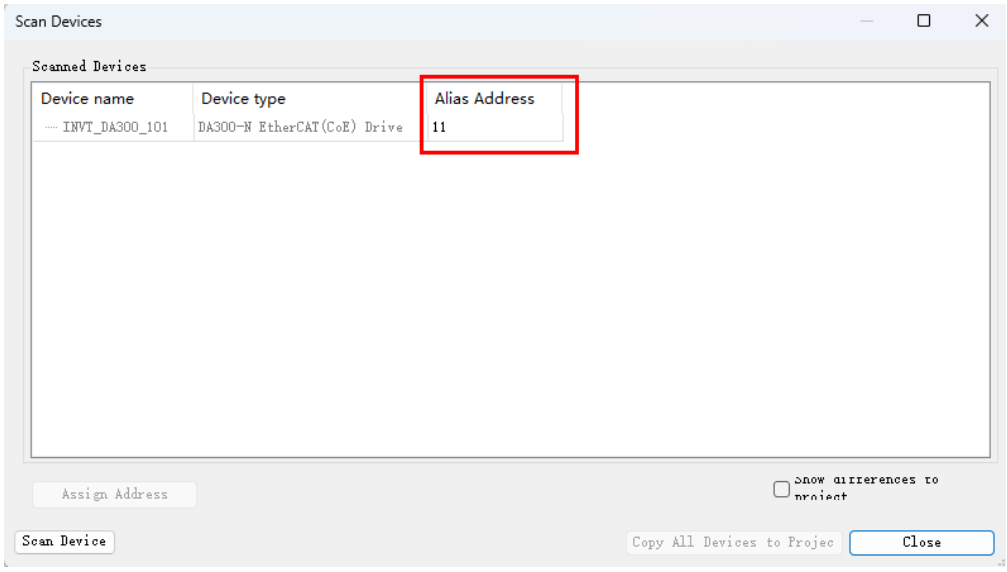
Step 1 Set the alias address after scanning the relevant device.

Figure 5-29 Setting a Node Alias



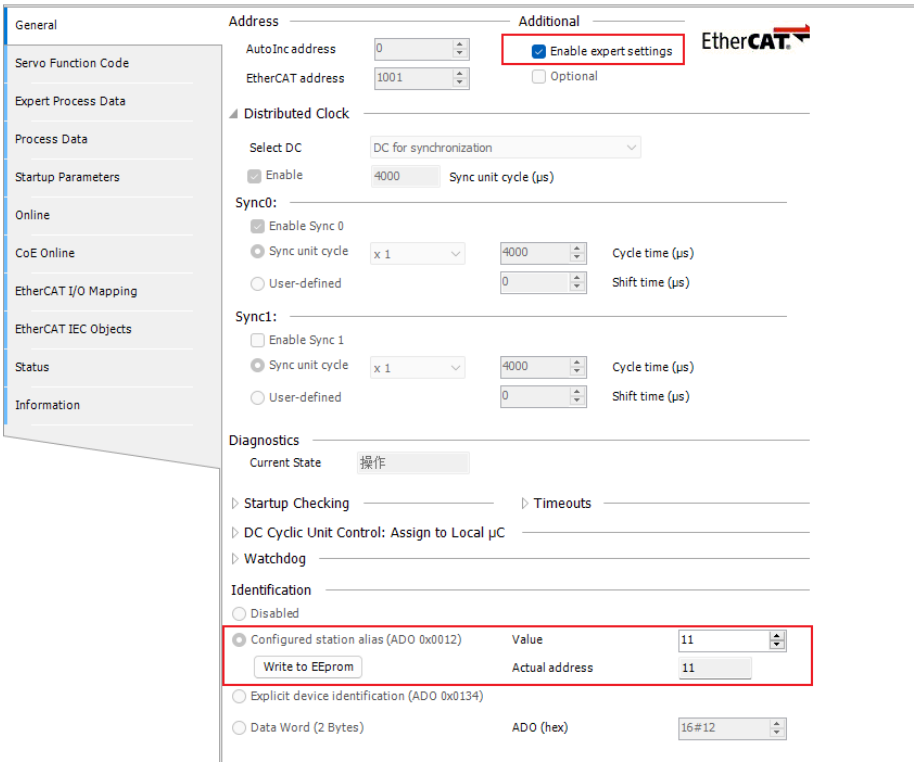
Step 2 After the assignment is completed, power off and restart the servo to activate it. Perform scanning again. At this time, the alias address has been written.

Figure 5-30 Restart to Activate the Node Alias



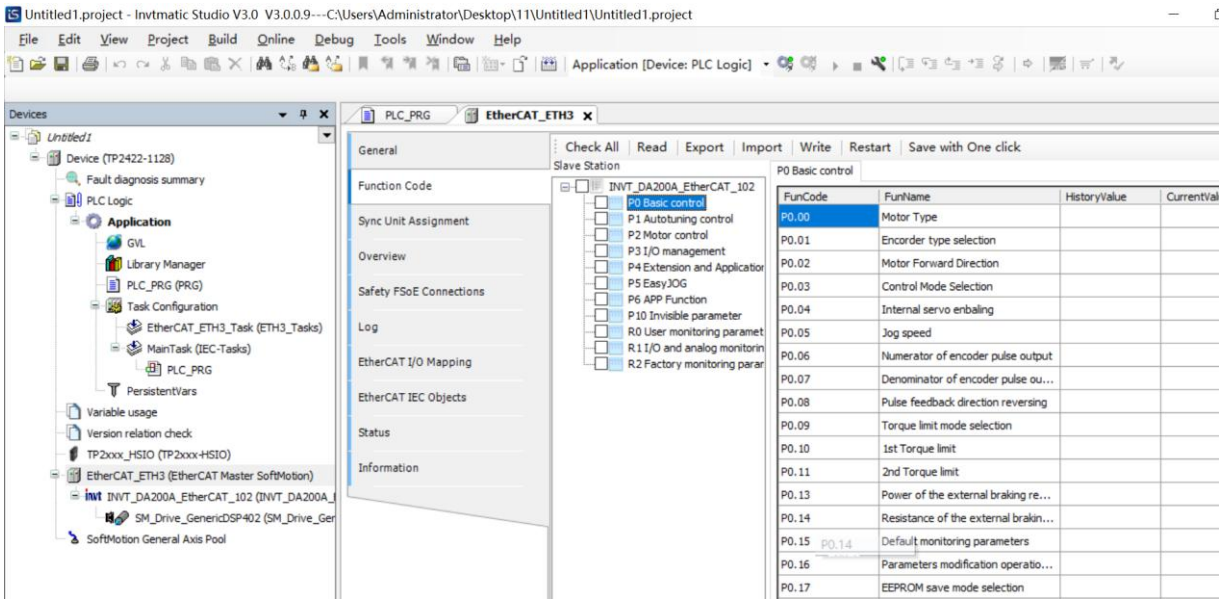
Step 3 Double-click the EtherCAT slave device **INVT_DA200_265** in the device tree, check **Enable Expert Settings** and select **Configured Node Alias** in the **Identification** options, and fill in the correct alias address, as shown in Figure 5-31Figure 5-31. Then, restart the controller.

Figure 5-31 Filling in the Node Alias



5.3.2.2 Function Codes

Double-click the servo device under the EtherCAT master, then click **Function Code** to open the function code interface. Click a specific function code group to view its data list.



- Check All: Click **Check All** to select all checkboxes in the list, specifying the function codes for read/write operations.
- Read: Click **Read** to read the current values of the selected function codes.
- Export: Click **Export** to export the information of the selected function codes to an Excel file.
- Import: Click **Import**. In the dialog box that appears, select the function code Excel file and click OK to import the values into the servo device and update the data list.
- Write: Modify the function codes and current values (which require a restart to take effect), then click **Write** to save the values to the servo device's EEPROM.
- Restart: Click **Restart** to restart the selected servo device.
- Save with one click: Click **Save with One Click** to save the read or modified values of the selected function codes as historical data and store the file in the PLC. This allows the HMI to call the function block to restore the function code configuration with one click.

5.3.2.3 Process Data Object (PDO)

In practical automation control systems, there are typically two types of data exchange between applications: time-critical and non-time-critical. Time-critical means that a specific action must be completed within a defined time window; any failure to communicate within this window may result in control failure. The process of cyclically transmitting time-critical data is known as Process Data Objects (PDO). Non-time-critical data, on the other hand, can be transmitted non-cyclically. In EtherCAT, this is implemented via Mailbox communication using Service Data Objects (SDO).

Output PDO (usually control word and given parameters) is shown in Figure 5-32. Input PDO (usually status word and feedback parameters) is shown in Figure 5-33. In the EtherCAT I/O mapping interface, you can map the corresponding variables and call them in the program, as shown in Figure 5-34.

Figure 5-32 Output PDO

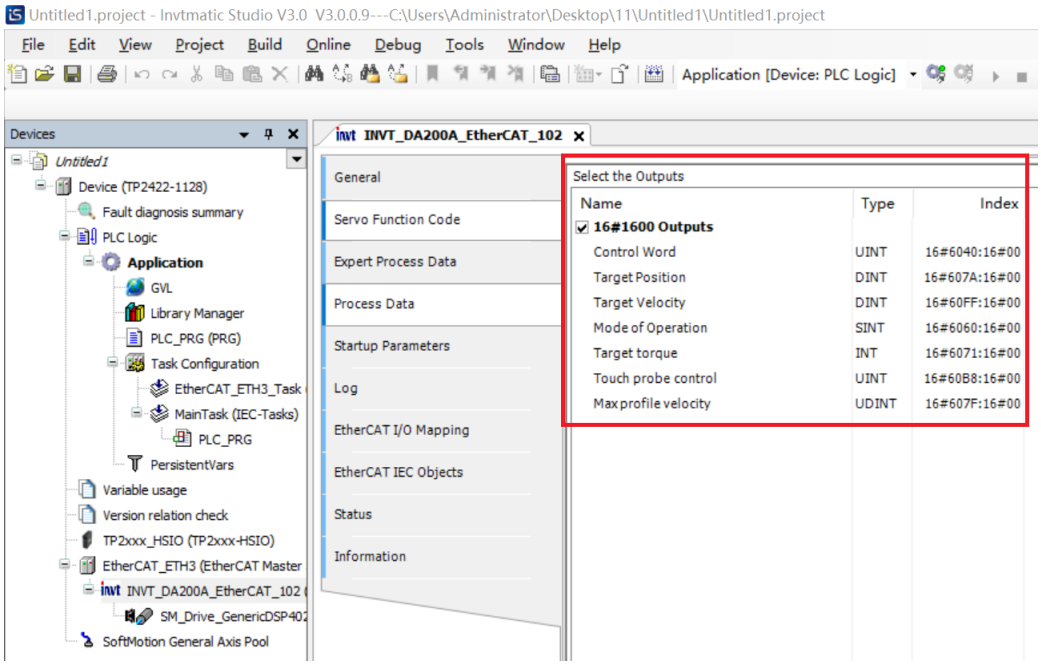
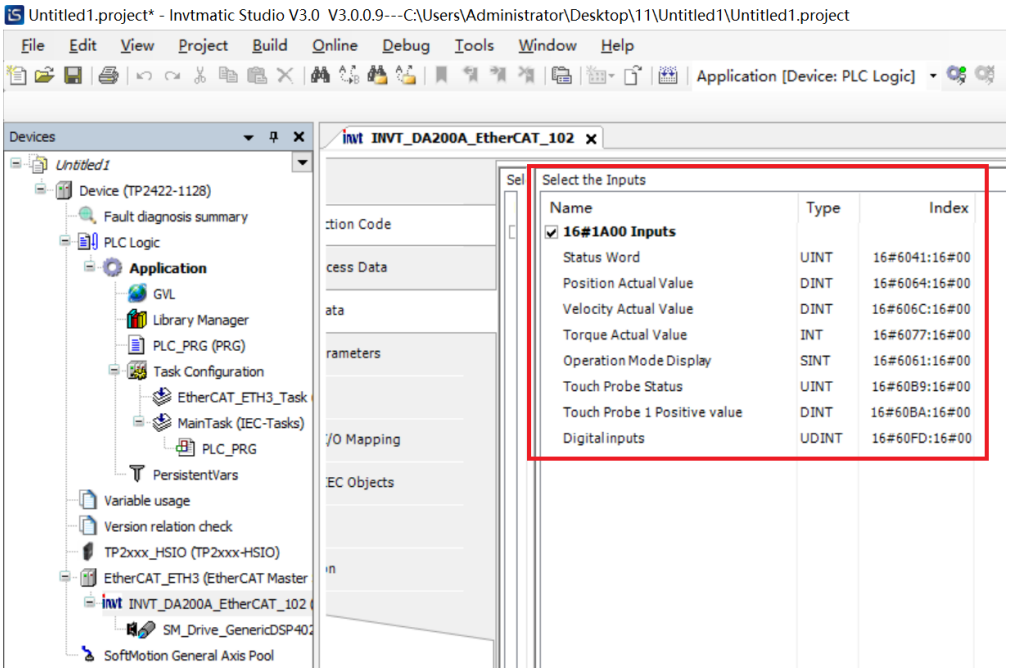


Figure 5-33 Input PDO



On the **Process Data** interface, the first row provides PDO editing buttons and displays PDO information, as shown in the following figure.

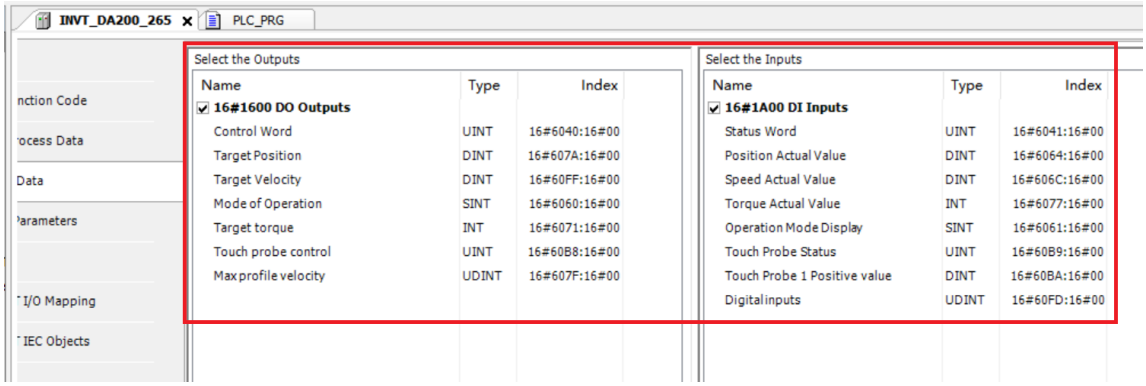
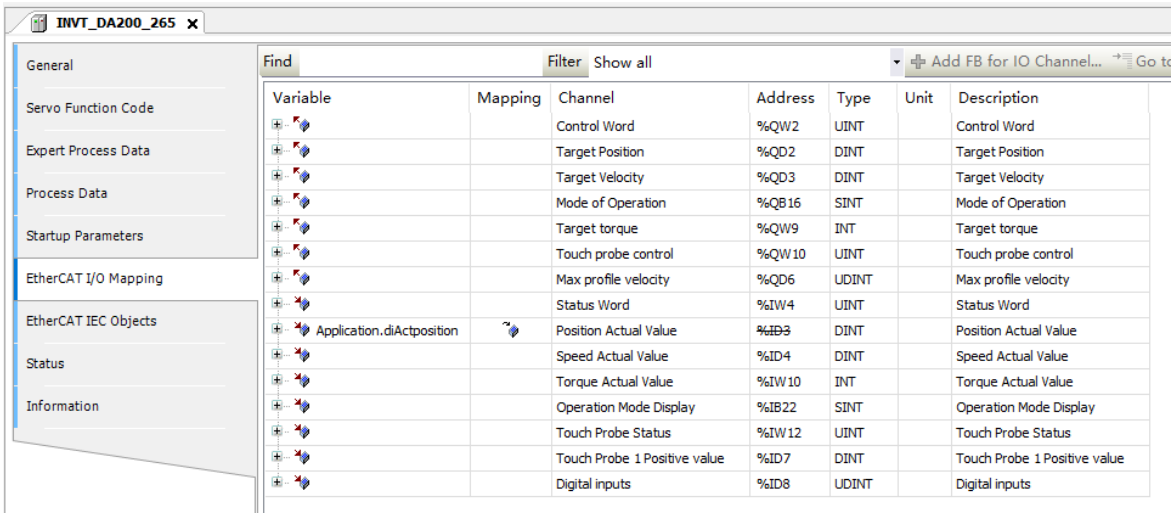


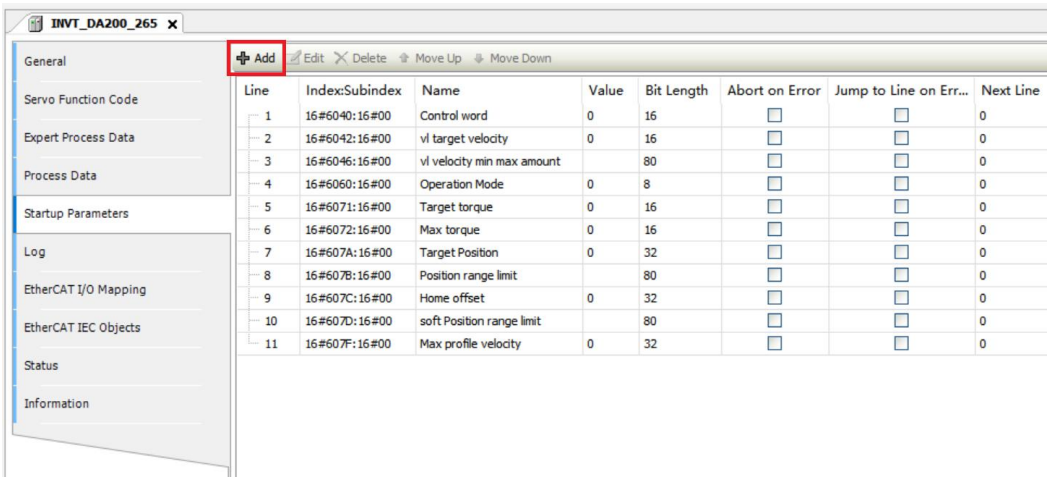
Figure 5-34 EtherCAT I/O Mapping



5.3.2.4 Startup Parameters

Some startup parameters can be written to the slave by the SDO write function when the slave is in the PreOP state, as shown in Figure 5-35.

Figure 5-35 Startup Parameters

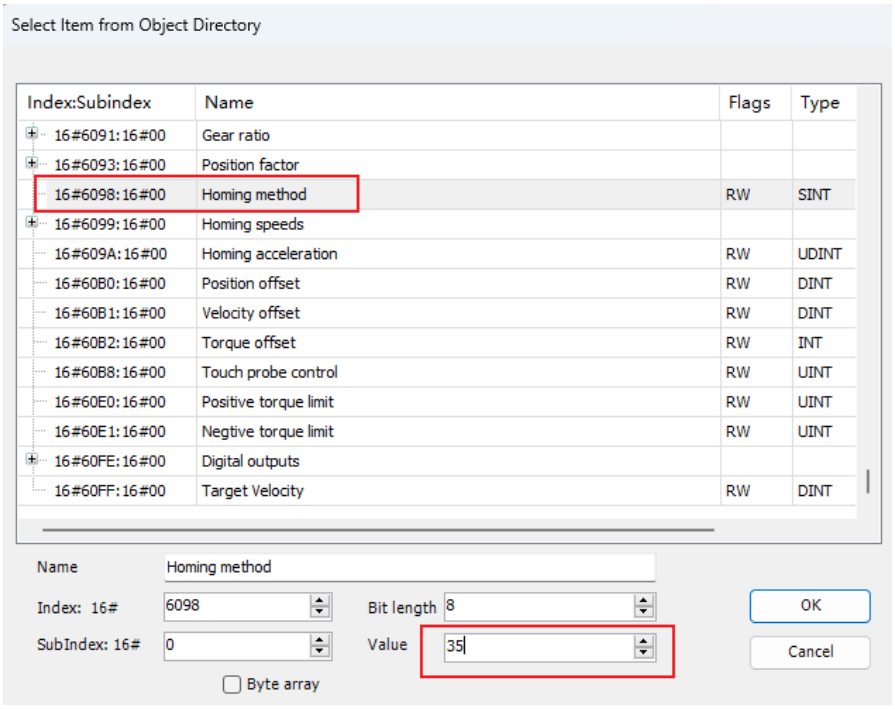


In actual use, you can add startup parameters as needed. Taking DA200 servo drive as an example, if you need to change the homing method to 35 at startup, follow the steps blow:

Step 1 Click **Add** on the Startup Parameters interface, as shown in Figure 5-35.

Step 2 In the pop-up window, find the object dictionary 16#6098, namely the homing method, as shown in Figure 5-36, set the value to 35, and then click **OK**.

Figure 5-36 Selection of the Homing Method



Step 3 Once added, the corresponding item will be displayed in the list, indicating that the startup parameter has been added successfully, as shown in Figure 5-35.

5.3.2.5 CiA 402 Axis

When using the motion control function in Invtmatic Studio, you need to first add a 402 axis under the EtherCAT slave, which will be used as the control object in the program. For details on the corresponding parameter configuration of the EtherCAT master, see section 5.3.1 EtherCAT Master Configuration. Here, the EtherCAT master connected to DA200 servo drive slave is taken as an example, which is for reference only.

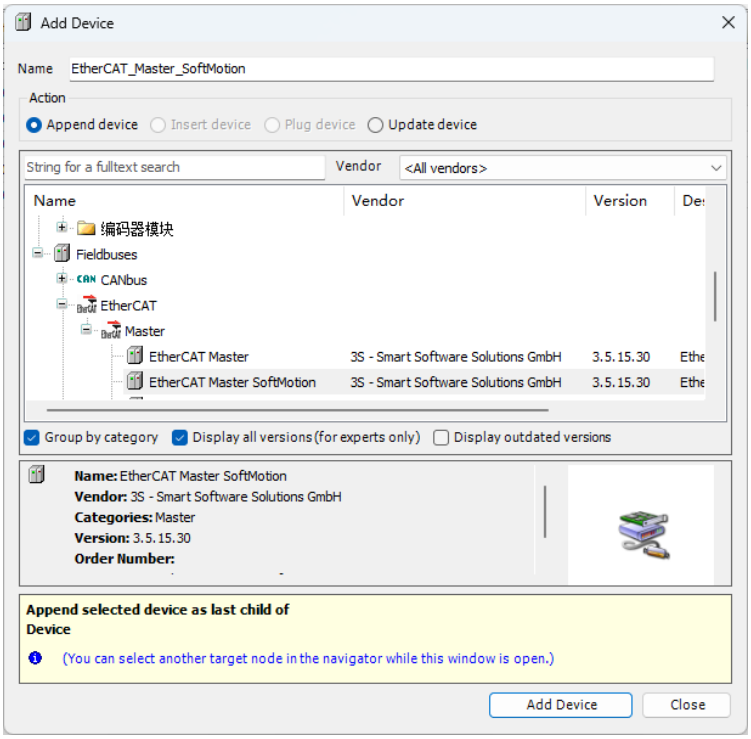
If you use a slave from INVT, Invtmatic Studio has already pre-processed it and will automatically add the corresponding axis when you add the slave. If you use a third-party slave, you need to manually add the 402 axis when using it, and add the library file “INVT_DA200_xxx.devdesc.xml” required by this module. Taking INVT_DA200_265 as an example, the operation steps are as follows:

Note:

- The highest task priority 0 is recommended for the creation of EtherCAT Master SoftMotion projects.
- It is recommended to keep the synchronization cycle and the task cycle consistent and set them ≥ 4 ms.
- To create EtherCAT Master SoftMotion, it is recommended to use separate tasks. For example, tasks such as I/O, analog input and output, and Modbus communication need to be separated from EtherCAT Master SoftMotion tasks.

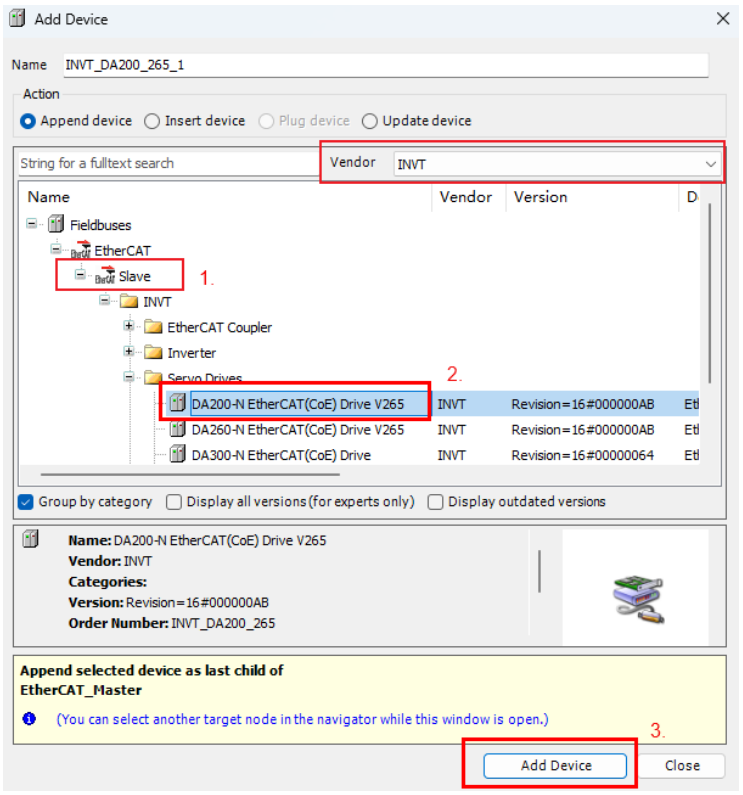
Step 1 Right-click **Device** in the device tree, select **Add Device** and then **Fieldbus** > **EtherCAT** > **Maser** > **EtherCAT Master SoftMotion**.

Figure 5-37 Process of Adding the EtherCAT Motion Control Master



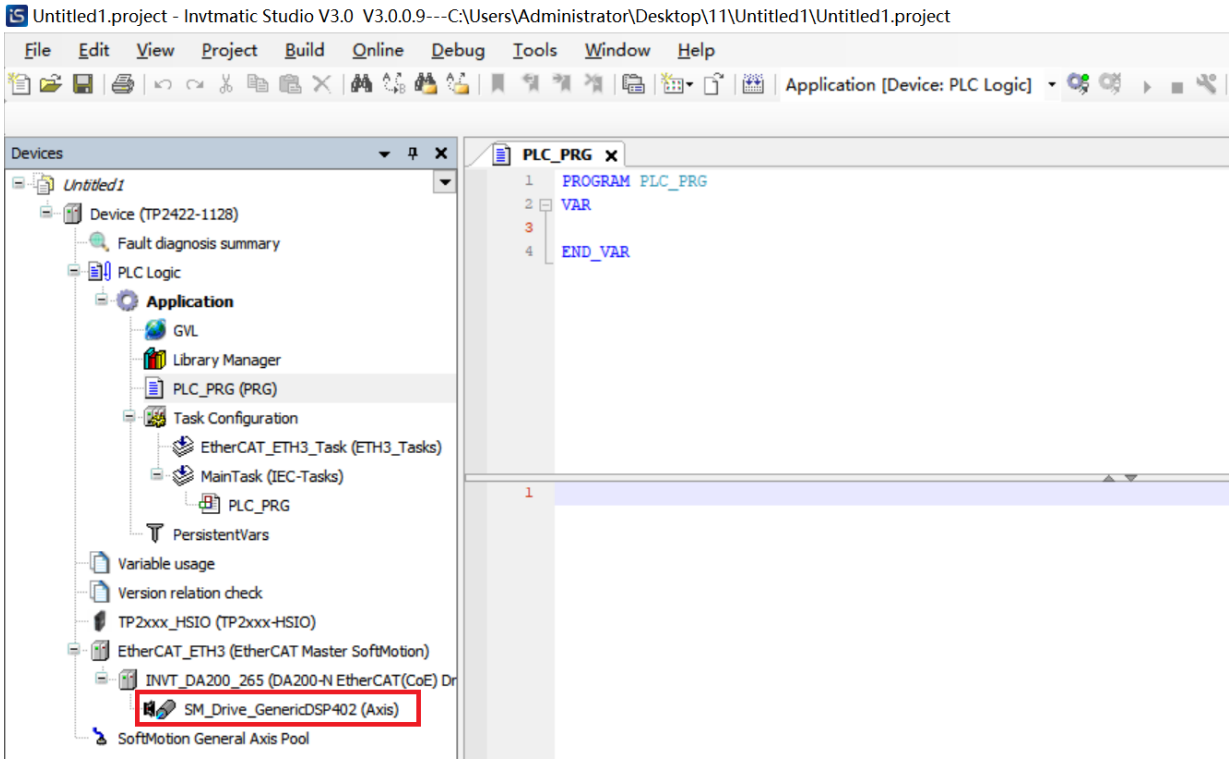
Step 2 Right-click **EtherCAT_Master_SoftMotion** and select **Add Device** to add the INVT DA200 servo drive. The operation steps are shown in Figure 5-38.

Figure 5-38 Adding DA200 Servo Drive



Step 3 Select INVT_DA200_265 in the device tree, right-click and add **SoftMotion CiA 402 Axis**, and add the calling program, as shown in Figure 5-39.

Figure 5-39 DA200 Servo Drive Application Example



5.3.2.6 Common Parameters of EtherCAT Slave

Common parameters of the EtherCAT slave are listed in Table 5-15. When using it, you need to add the slave name. For example, if the slave name is INVT_DA200_265, when the slave state is read, the corresponding variable is INVT_DA200_265.wState.

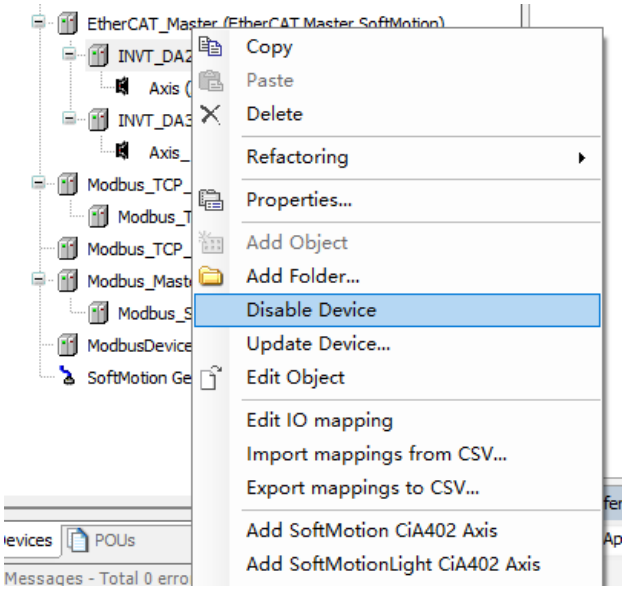
Table 5-15 Common Parameters of EtherCAT Slave

Variable Name	Function
wState	1: ETC_SLAVE_INIT 2: ETC_SLAVE_PREOPERATIONAL 3: ETC_SLAVE_BOOT 4: ETC_SLAVE_SAVEOPERATIONAL 8: ETC_SLAVE_OPERATIONAL
SlaveAddr	Slave address

5.3.2.7 Enabling/Disabling a Slave

In actual applications, you may encounter situations where the configuration is inconsistent with the actual hardware connection, resulting in the inability of the bus to operate normally. In view of this situation, Invtmatic Studio provides a solution that allows users to disable unconnected slaves to ensure that the bus can operate normally. When variables associated to a device are used in the program, disabling the device will prevent compilation errors, as shown in Figure 5-40.

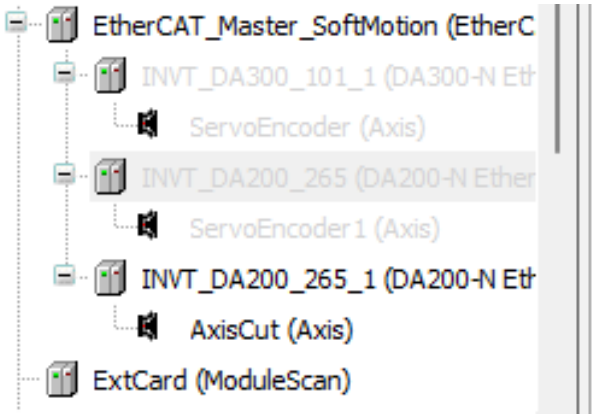
Figure 5-40 Disabling a Device



Note:

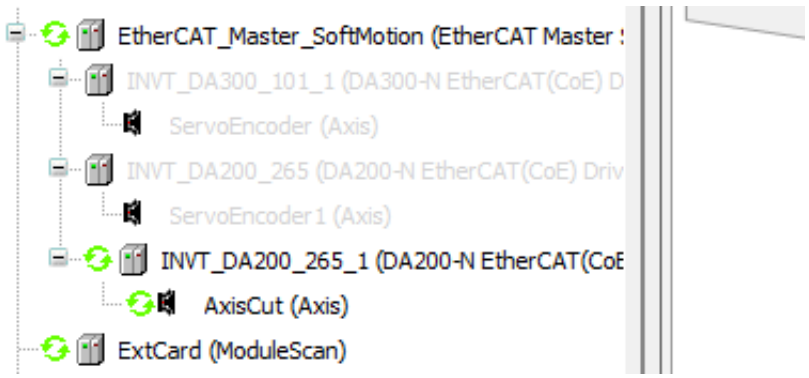
- Disabled devices in the device tree are displayed in light gray color, as shown in Figure 5-41.

Figure 5-41 Device Disabled State



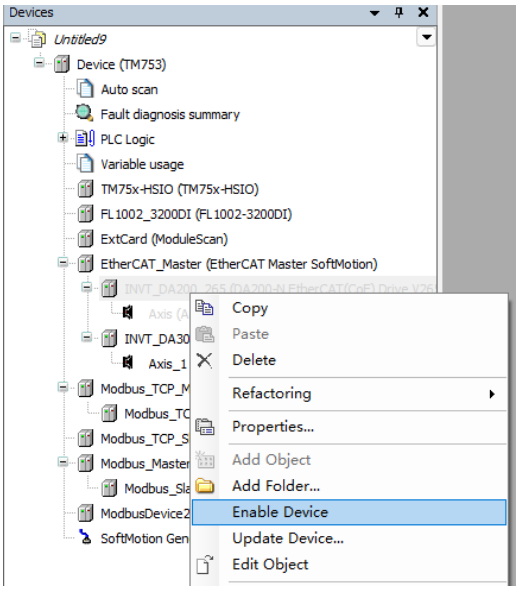
- When you log in to the system, the running states of disabled devices will not be displayed, as shown in Figure 5-42.

Figure 5-42 Program Running While Disabled Devices Inactive



- When you need to re-enable the device, click “Enable Device”, as shown in Figure 5-43.

Figure 5-43 Enabling a Device

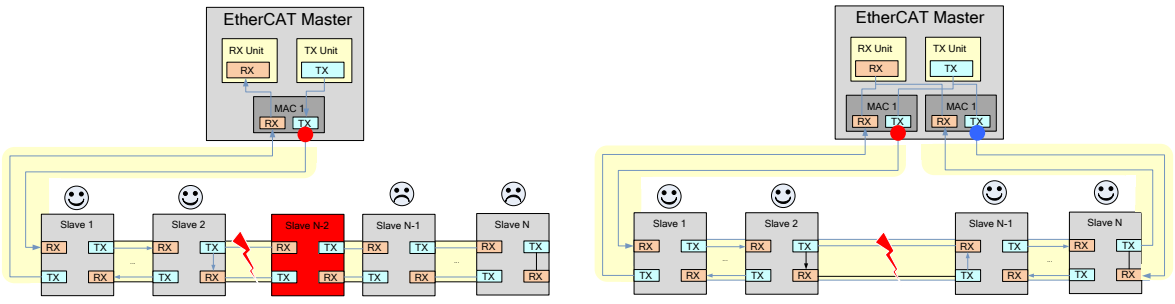


5.3.3 EtherCAT Cable Redundancy Function

EtherCAT can use the cable redundancy technology to meet the rapidly growing system reliability requirements. This technology can ensure that devices can be replaced without shutting down the network. Adding redundancy is not expensive: Just add a standard Ethernet port (no dedicated network card or interface required) and a network cable to the master device to transform the linear topology into a ring topology. When a device or cable fails, switching can be completed in just one cycle. Therefore, even for applications with motion control requirements, there will be no problems in the event of a network cable failure.

EtherCAT uses hot backup to support master cable redundancy. Once an outage, device failure, or other issues occur, the EtherCAT slave controller can automatically return Ethernet frames immediately, so the entire network will not be shut down. If there is a network interruption between Slave2 and SlaveN-2 in this topology, as indicated by the red part in Figure 5-44 a), the communication of all slaves after Slave N-2 will also be interrupted accordingly, which is also a disadvantage of the standard topology.

Figure 5-44 EtherCAT Cable Redundancy



In Figure 5-44 b), the master only needs two standard network ports to implement this topology. Using these two network ports, all slaves can form a loop. Even if the network is interrupted during use, as indicated by the red part in Figure 5-44 b), the master will immediately detect the error and automatically divide the communication into two paths, while the slaves can continue communication to ensure the stable operation

of the system.

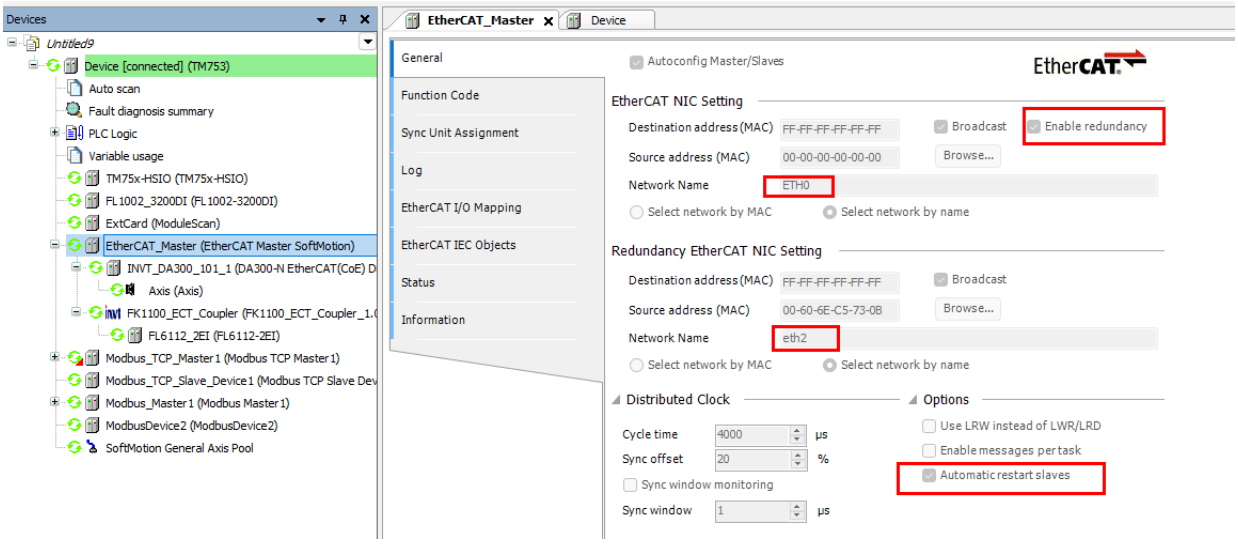
Note: According to the EtherCAT protocol standard, the first slave is the standard clock, so the failure of the first slave will affect the communication of the entire bus.

As shown in Figure 5-45, you can configure a cable redundancy ring network as follows:

Double-click **Slave EtherCAT_X3** to open it, and check **Enable redundancy** to show **Redundancy EtherCAT NIC Setting** parameters. For both **EtherCAT NIC Setting** and **Redundancy EtherCAT NIC Setting**, it is recommended to check **Select network by MAC** or **Select Network by name**. The latter one is used in Invtmatic Studio by default. Click **Browse** to confirm that the corresponding EtherCAT network interface is well configured, and then check **Automatic restart slaves**.

Note: When the network cable between the first and second slaves is unplugged, the three slaves on the bus can still operate normally.

Figure 5-45 EtherCAT Cable Redundancy Ring Network Configuration

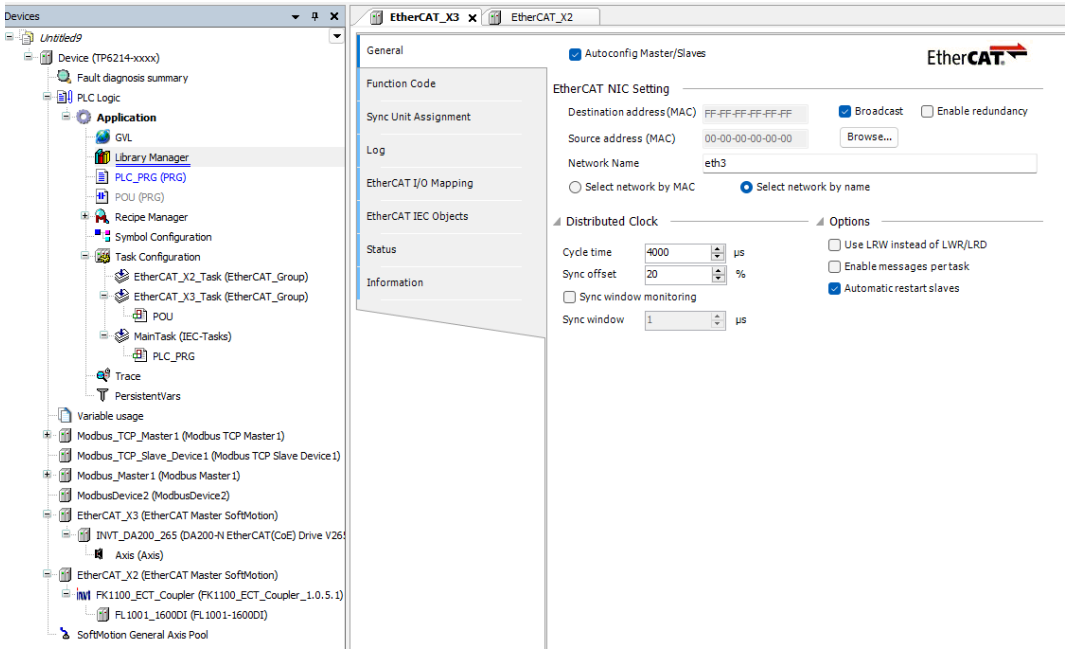


5.3.4 Dual EtherCAT Masters

5.3.4.1 Dual-Master Application

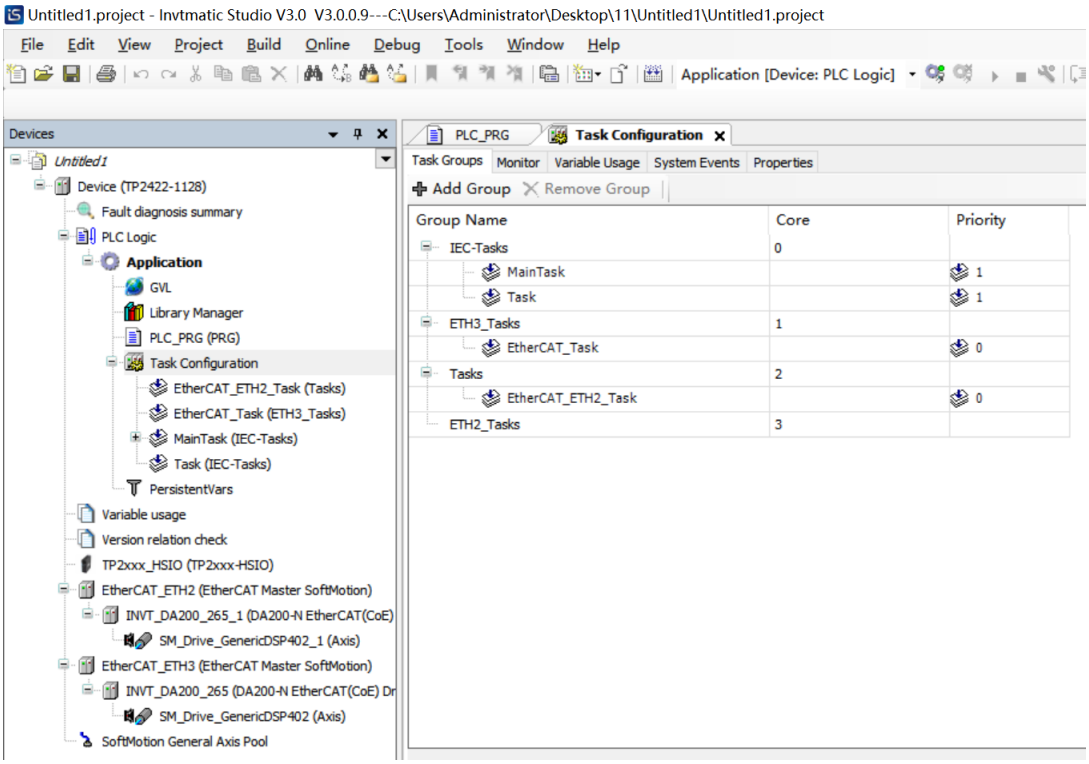
TP2422 series PLC supports dual EtherCAT masters. In Invtmatic Studio, when you select the device TP2422, you can add two EtherCAT masters and automatically assign EtherCAT network ports X3 and X2, as shown in Figure 5-46. It is generally recommended to connect the servo and 402 axis with higher real-time performance to EtherCAT_ETH3, and couplers to EtherCAT_ETH2.

Figure 5-46 Dual EtherCAT Master Connection Effect

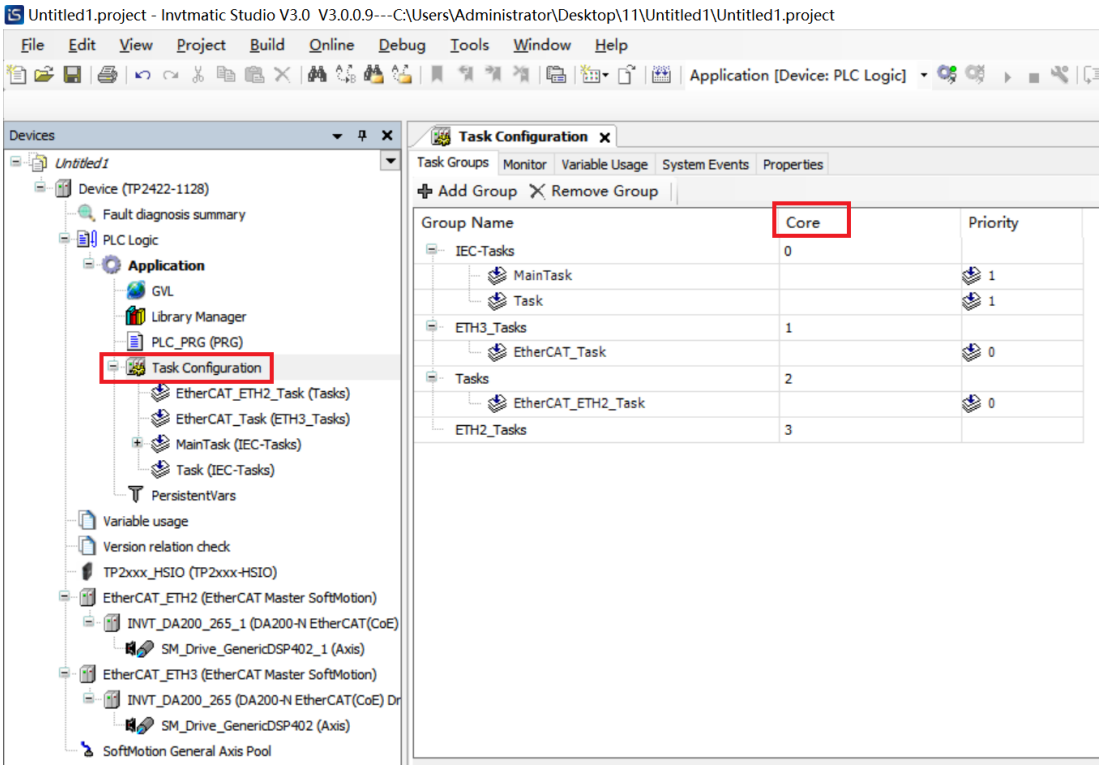


5.3.4.2 Task Configuration

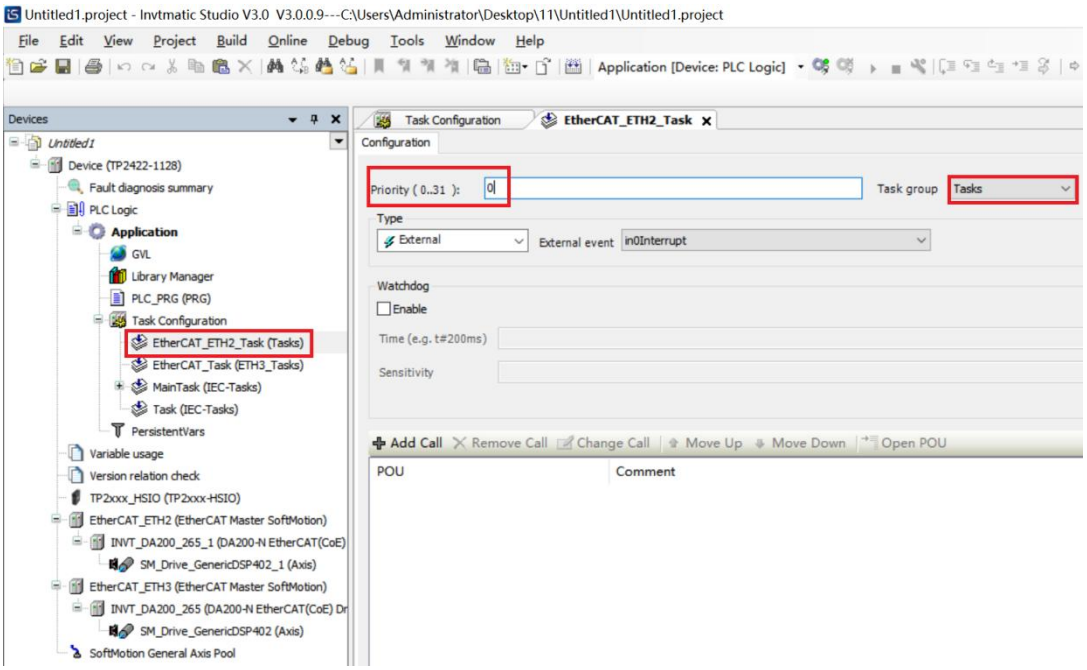
The TP2422 is a quad-core model. As shown in Figure 5-47, four tasks are configured: MainTask, EtherCAT_Task, EtherCAT_ETH3_Task, and EtherCAT_ETH2_Task. MainTask and EtherCAT_Task belong to the IEC-Tasks task group, EtherCAT_ETH2_Task belongs to the ETH2_Tasks task group, and EtherCAT_ETH3_Task belongs to the ETH3_Tasks task group.



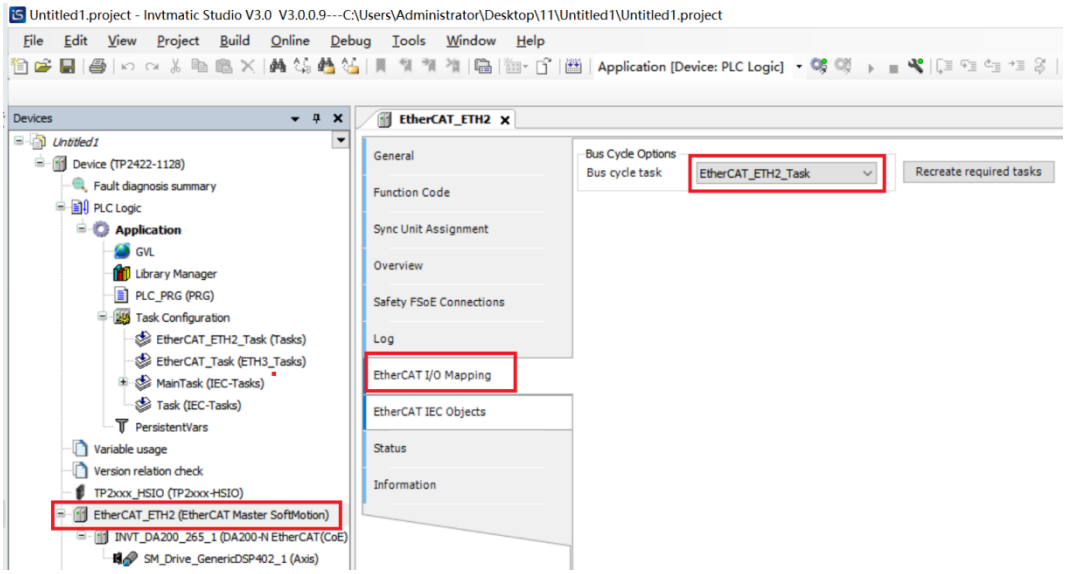
The core selection is shown in the figure below.



The selection of task priorities and task groups is shown in the figure below.



The selection of the bus cycle task is shown in the figure below.



5.4 CANopen

CANopen is a higher-layer communication protocol that is based on the Controller Area Network (CAN) protocol and extended from the CAL protocol, including communication profile and device profile.

The communication model defines four types of messages (communication objects)

1. Management message

It mainly involves layer management, network management, and ID assignment services: such as initialization, configuration, and network management (including node protection). The services conform to the LMT, NMT, and DBT services of the CAL, and uses the master/slave communication mode, which means there can only be one LMT, NMT, or DBT master node and one or more slave nodes in a CAN network.

2. Service Data Object (SDO)

SDO enables clients to use indexes and sub-indexes (in the first few bytes of a CAN message) to access items (objects) in the device (server) object dictionary.

SDO is implemented through a multi-domain CMS object in CAL that allows the transfer of data of any length (The data will be split into several messages when it exceeds 4 bytes).

SDO communication follows many protocol rules, generating a response for each message (two IDs are required for an SDO).

SDO request and response messages are fixed to 8 bytes (meaningless data lengths are indicated in the first byte which carries the protocol information).

3. Process Data Object (PDO)

PDO is used to transfer real-time data from a creator to one or more recipients. Data transfer is limited to 1 to 8 bytes (for example, one PDO can transfer up to 64 digital I/O values, or 4 16-bit AD values).

PDO communication has no protocol defined. PDO data content is defined only by CAN ID, assuming that the creator and recipients know the data content of the PDO.

Each PDO is described by two objects in the object dictionary:

A. PDO communication parameters determine which COB-ID will be used by the PDO, transmission type, prohibition time, and timer cycle.

B. PDO mapping parameter: a list of objects in the object dictionary that are mapped to the PDO, including their data lengths (in bits). The creator and recipients must know this mapping to interpret PDO content.

PDO message content is predefined (or configured at network startup). Mapping application objects to the PDO is described in the device object dictionary. If the device (creator and recipients) supports variable PDO mappings, the PDO mapping parameters can be configured using SDO messages.

PDO can be delivered in the following modes:

A. Synchronization (by receiving SYNC objects)

Acyclic: The transmission is pre-triggered by a remote frame or by an object-specific event defined in the device profile.

Cyclic: The transmission is triggered after every 1 to 240 SYNC messages.

B. Asynchronization

The transmission is triggered by a remote frame or by an object-specific event defined in the device profile.

4. Predefined messages or special function objects

A. SYNC

B. Time stamp

C. Emergency

D. Node guarding

5.4.1 CANopen Master Configuration

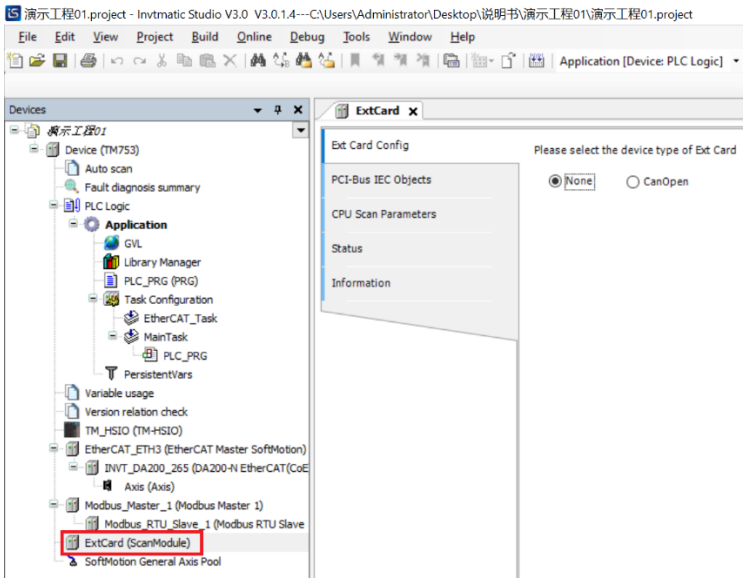
5.4.1.1 Using the Master

The associated CANopen slave device profile must first be installed into the system. The device profile can be a *.Devdesc.xml file or an EDS (Electronic Data Sheet) file for the manufacturer.

Install the corresponding CANopen slave device. For example, if the TM series PLC uses CANopen, you need to install the expansion module first. The operation steps are as follows:

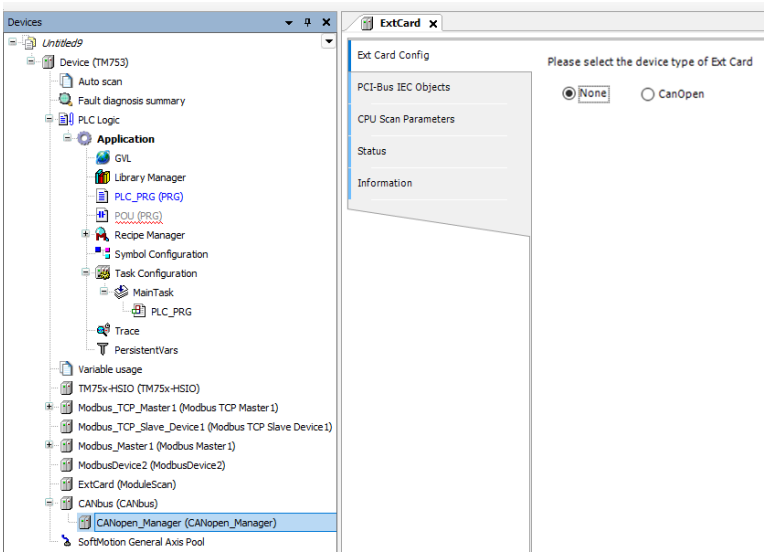
Step 1 Double-click **ExtCard** in the device tree and enter the Settings interface. The default option is **None**.

Figure 5-47 ExtCard Expansion Card Interface



Step 2 Select **CanOpen**. Then the left device tree will automatically add **CANbus** Bus and **CANopen_Manager**.

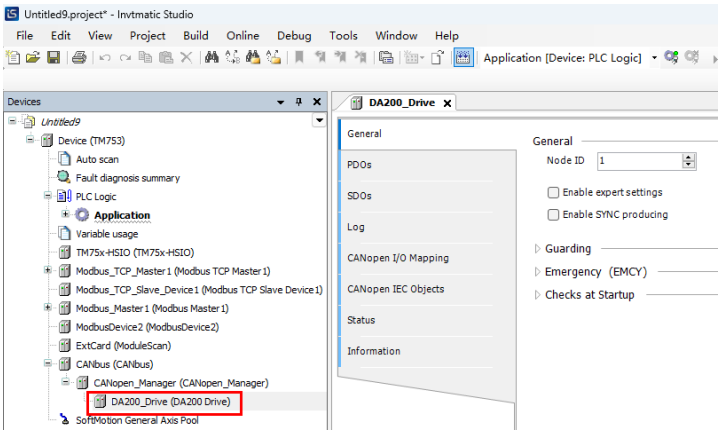
Figure 5-48 Selecting CANopen Extension



5.4.1.2 Adding a CANopen Slave Node

Take our DA200 CANopen slave node as an example. Add DA200 slave device under CANopen Manager after adding the EDS file of this slave node, as shown in Figure 5-49.

Figure 5-49 Device Tree Structure with a CANopen Slave

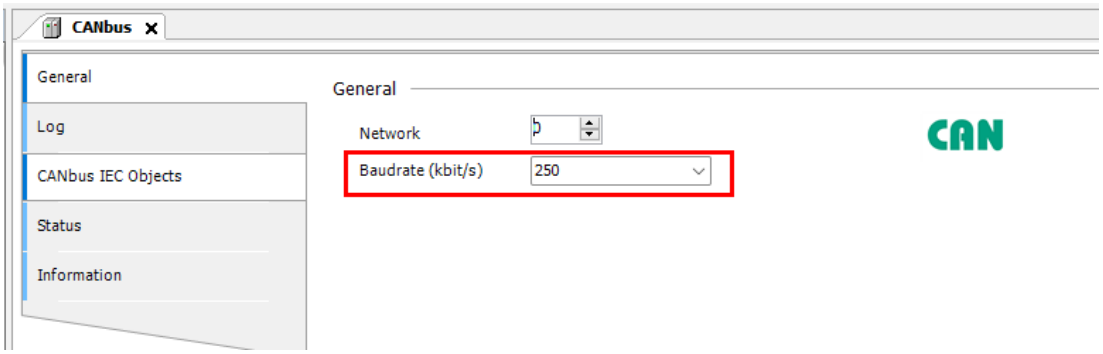


At this time, the software configuration of the CANopen master is completed.

5.4.2 CANopen Parameter Configuration

5.4.2.1 Network and Baud Rate Configuration Parameters

Figure 5-50 CANbus Bus Parameter Configuration



The baud rate must be consistent with the servo parameter P4.02 (CAN communication baud rate).

- Network: the number of CAN networks connected via the CANbus. Range: 0–100.
- Baud rate: the baud rate used for transmission on the bus. Different baud rates can be set (for example, 10kbps, 20kbps, 50kbps, 100kbps, 125kbps, 250kbps, 500kbps, 800kbps, 1000kbps).

  The maximum number of devices available at the physical layer depends on the drive power of the transceiver, mainly on whether the transceiver of the transmitting device can overcome the minimum total ohmic bus load. Each module connected to the bus reduces the ohmic bus load. Therefore, once the number of devices exceeds a certain limit, the drive will no longer be able to provide the required power.
- Termination resistor

  A typical CANopen bus requires a 120Ω termination resistor at each end to eliminate signal reflections within the communication cable. Signal reflections are caused by two conditions: impedance discontinuity and impedance mismatch.

- Impedance discontinuity

  Reflections occur when a signal suddenly encounters very low or zero cable impedance at the end of a transmission line, a principle similar to light reflection when light passes from one medium into another. To eliminate this, a termination resistor matching the cable's characteristic impedance should be connected at the end of the cable to ensure impedance continuity. Since signals are transmitted bidirectionally along the cable, a termination resistor with the same resistance should also be connected at the other end of the communication cable.

- Impedance mismatch

  Another cause of signal reflection is impedance mismatch between the data transceiver and the transmission cable. This type of reflection is mainly manifested as data disorder across the network when the communication line is idle. To reduce the impact of reflected signals on the communication line, methods such as noise suppression and bias resistors are commonly used. The 120Ω termination resistor parameters mentioned above applies to a 1Mbit/s transmission rate with a 40m maximum cable length. For different network lengths and baud rates, refer to Table 5-16.

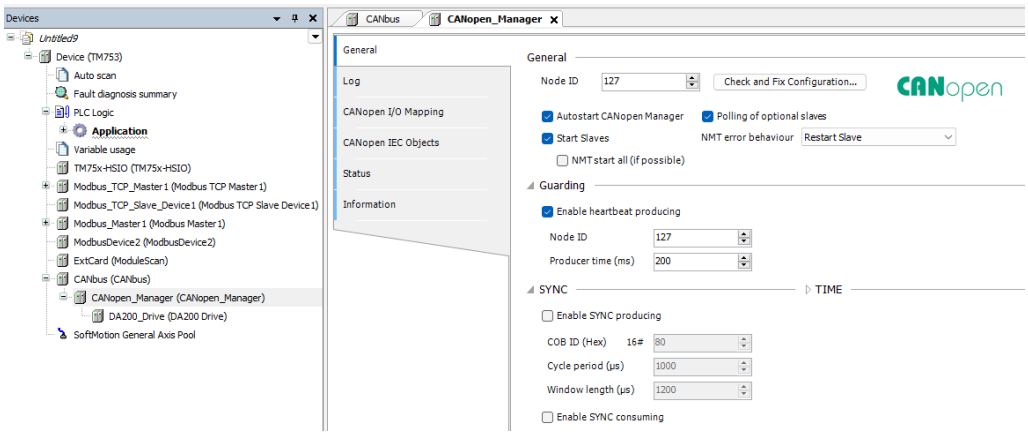
Table 5-16 Recommended Cable Resistance per Unit Length and Termination Resistance

Bus length (m)	Bus cable		Termination resistance (Ω)	Baud rate
	Resistance per unit length (mΩ/m)	Cross-sectional area (mm^2)		
0–40	70	0.25–0.34	124	1Mbit/s
40–300	<60	0.34–0.6	150–300	>500kbit/s
300–600	<40	0.5–0.6	150–300	>100kbit/s
600–1000	<26	0.75–0.8	150–300	>125kbit/s

5.4.2.2 Master Configuration Parameters

CANopen Management is a node under the CANbus node that supports CANbus configuration through internal functions. It is generally used as the CANbus master. The configuration page is shown in Figure 5-51.

Figure 5-51 CANopen Master Parameter Configuration



- **General**
  - Node ID: It provides an array pair module that CANopen Manager can correspond to one-to-one, with ID values of 1-127 (must be a decimal integer).
  - Autostart CANopen Manager: If this option is enabled and all slaves are ready, the CANopen Manager automatically starts and enters the Operation mode. Otherwise, it must be started manually, for example, by using the NMT function block in the CiA405 protocol through the CodeSys program.
  - Start Slaves: If this function is enabled, the CANopen Manager automatically starts all slaves. Otherwise, the slaves are started by the NMT function block in the CiA405 protocol.
  - Polling of optional slave: If a slave does not respond in time during boot-up, a request is sent once per second until the slave responds.
  - NMT error behavior: When an NMT fault occurs, you can select whether to restart or stop the slave.
- **Guarding**
  - Different from node protection, the heartbeat mode is a traditional protection mechanism that can be handled by the master and slave modules. Normally, the master is configured to send a heartbeat to the slave.
  - Enable heartbeat producing: If this option is enabled, the master will send heartbeats continuously according to an internally defined “heartbeat time”.
  - Node ID: The node ID sent by the Heartbeat Producer. The default value is 127, which indicates the CANopen master.
  - Producer time (ms): It defines the internal heartbeat time in milliseconds, that is, the interval at which heartbeat messages are sent.
- **SYNC**

  The synchronization function mainly provides the following features:

  Network-wide synchronization, especially in drive applications: Input values are stored simultaneously across the network and then transmitted. Output values are updated based on the messages received after the previous SYNC.

  Master/slave mode: The SYNC master node sends SYNC objects periodically, and the SYNC slave nodes execute their tasks synchronously upon receiving it.

  A synchronous PDO is transmitted within a specified time window after the SYNC message is transmitted.

  It is implemented using CMS objects of the basic variable types in CAL.

  CANopen recommends using the highest-priority COB-ID to ensure proper transmission of synchronization signals. A SYNC message may carry no process data to keep the message as short as possible.

  - Enable sync producing: Enables the function for sending SYNC messages.
  - COB-ID (Hex): Specifies the COB-ID (hexadecimal) of the transmitted SYNC frame.
  - Cycle period (μs): Specifies the cycle period for transmitting SYNC frames, in microseconds.
  - Window length (μs): Specifies the synchronous PDO time window, in microseconds.

- **Time**

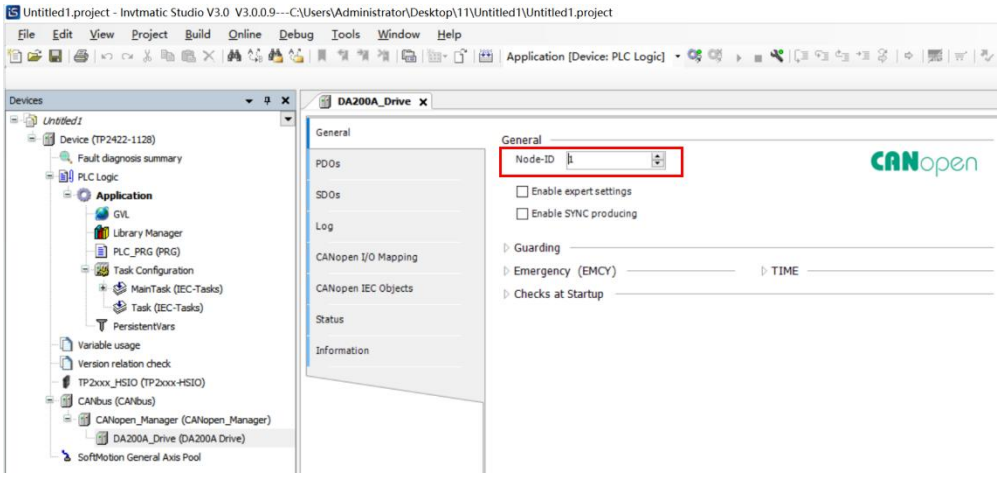
  Enable time producing: Enables the Time Stamp. After this function is enabled, the CANopen Manager sends time information according to the related settings.

  COB-ID (Hex): Specifies the COB-ID for transmitting the time stamp. The default value is 16#100H.

  Producer timer (ms): Specifies the interval for transmitting time stamps. The value must be a multiple of the task cycle time.

  Note: The master configuration parameters must be consistent with the servo configuration parameters.

5.4.2.3 Slave Configuration Parameters



- **General:**
  - **Node ID:** Specifies the slave node ID. The configurable range is 1–127. The node ID must be consistent with the servo parameter P4.05 CAN communication node.
  - **Enable sync producing:** If this option is enabled, the slave supports synchronous transmission. The specific synchronization time is set in the CANopen Manager.
  - **Optional device:** If this option is enabled, the device is not started in the CAN network.
  - **Not initialised:** If this option is enabled, the master does not send SDO configuration information or NMT start commands to the slave.
  - **Reset node:** If this feature is supported by the hardware, enabling it will reset all CANopen parameters.
- **Node guarding**
  - **Enable node guarding:** Enables the Node Guarding function.
  - **Guard time (ms):** Specifies the transmission interval.
  - **Life time factor:** Specifies the multiplier of the guard time.
  - **Enable heartbeat producing:** Enables the heartbeat producing function.
  - **Producer time (ms):** Specifies the transmission interval.

- **Emergency (EMCY)**
  - **Enable emergency:** Enables the EMCY function. If a bus fault occurs, an error message is transmitted using the configured COB-ID.
- **COB-ID:** Specifies the COB-ID for emergency transmission.
- **Time**
  - This function depends on whether it is supported by the corresponding slave device.
  - **Enable time producing:** Enables the Time Producing function.
  - **COB-ID**: The COB-ID for transmitting the time stamp.
  - **Enable time consuming:** Enables the Time Consuming function.
- **Checks at startup**

  If the related options are enabled, the CANopen slave firmware version information is compared with the information in the EDS file. If the information does not match, the slave cannot start.
  - **Check vendor ID:** Checks the vendor ID (index 1018, sub-index 1).
  - **Check product code:** Checks the product code (index 1018, sub-index 2).
  - **Check revision number:** Checks the revision number (index 1018, sub-index 3).

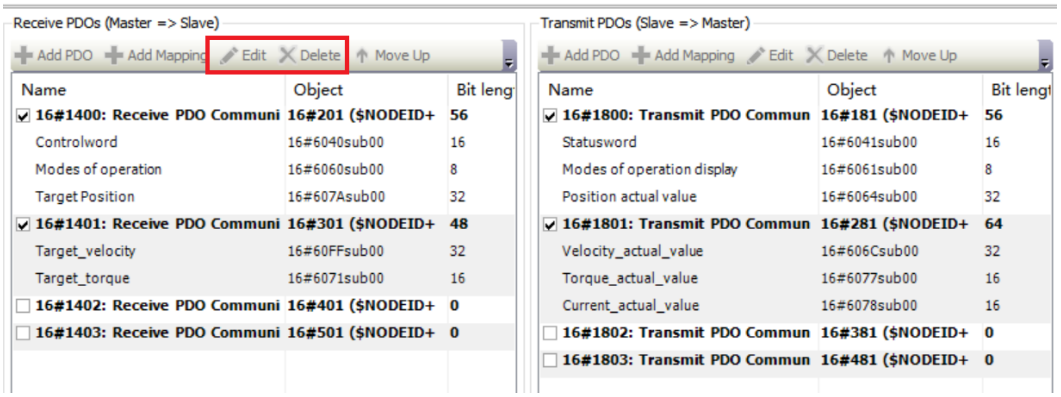
5.4.3 PDO Mapping Configuration

The number of PDO bytes directly affects the single-frame transmission time and bus load. Under a fixed update cycle, a larger data volume requires a higher baud rate; conversely, at a given baud rate, a larger PDO size results in a lower supported update frequency. Therefore, system design must strike a balance among real-time performance, data volume, communication distance, and network scale. Proper configuration of the PDO mapping and baud rate ensures the stable and efficient operation of the CANopen network.

5.4.3.1 PDO Mapping Modification

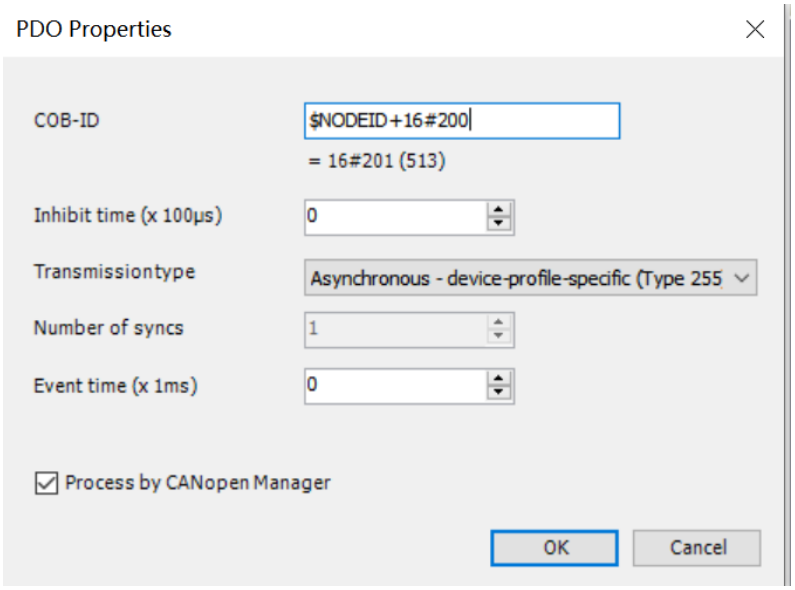
This option is used to display the specific parameters of the currently configured TPDOs and RPDOs, as shown in Figure 5-52. Mapping addresses can be added, deleted, or modified in **Receive PDOs** and **Transmit PDOs**.

Figure 5-52 PDO Mapping Modification



Double-click the bold black text in Figure 5-52 to configure the corresponding PDO mapping parameters. You can modify its COB-ID, transmission type, and other parameters. The configuration interface is shown in Figure 5-54.

Figure 5-53 PDO Property Settings



5.4.3.2 PDO Transmission Modes

Transmission Type	Synchronous Cyclic	Synchronous Acyclic	Asynchronous
0	-	√	-
1–240(*)	√	-	-
254	-	-	√

TPDO transmission type 1-240: The PDO is transmitted upon receiving every 1 to 240 SYNC signals.

Asynchronous TPDO transmission (254/255): The PDO is transmitted whenever the mapped data changes.

Asynchronous trigger (254): When the event timer is non-zero, the slave periodically transmits the Transmit PDO (PDO Tx).

When the event timer is zero, the slave will transmit the corresponding PDO Tx as long as its data changes. However, the transmission interval is restricted by the inhibit time; the same PDO Tx message can only be transmitted once within the inhibit time. This effectively reduces the bus load.

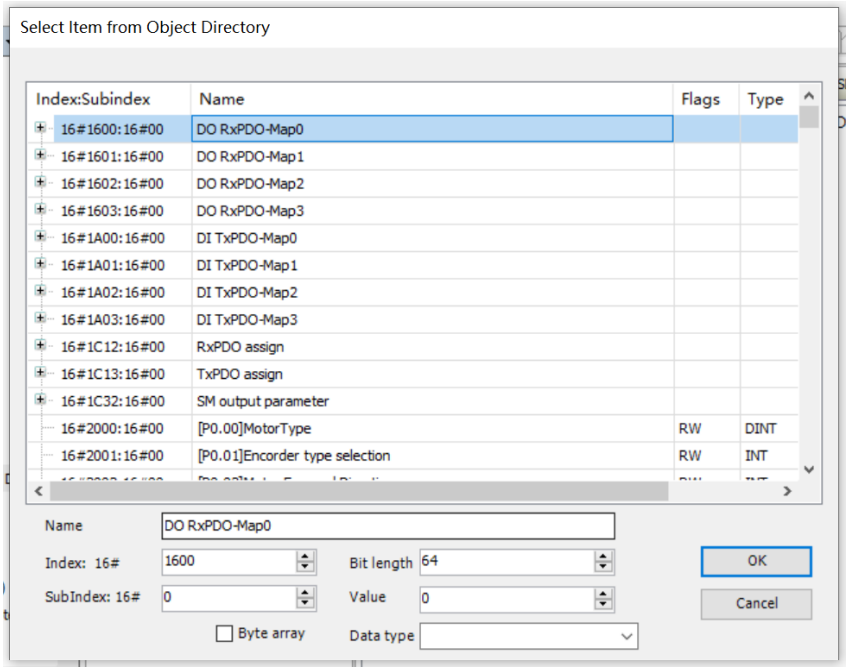
5.4.3.3 Addition of PDO Rx/PDO Tx

Figure 5-52 shows the PDO editing interface. By default, there are 4 Receive PDOs (PDO Rx) and 4 PDO Tx. If the total number of PDOs configured in the default EDS file is less than 8 and you need to add more PDOs, click **Add PDO**. The COB-ID and transmission type of the added PDO can be customized. After clicking **OK**, the system will generate a COB-ID for the PDO. However, the generated PDO does not yet possess communication capability; perform variable mapping for it. According to the CANopen protocol, each PDO supports a data transfer size of 8 bytes.

1. Adding Mappings

   Once the PDO is added, configure 8 bytes of communication variables for it. Select **Add Mapping**. Add the PDO and click **Add PDO**. The window shown in Figure 5-54 will pop up automatically. Select the index and sub-index, and then click **OK**.

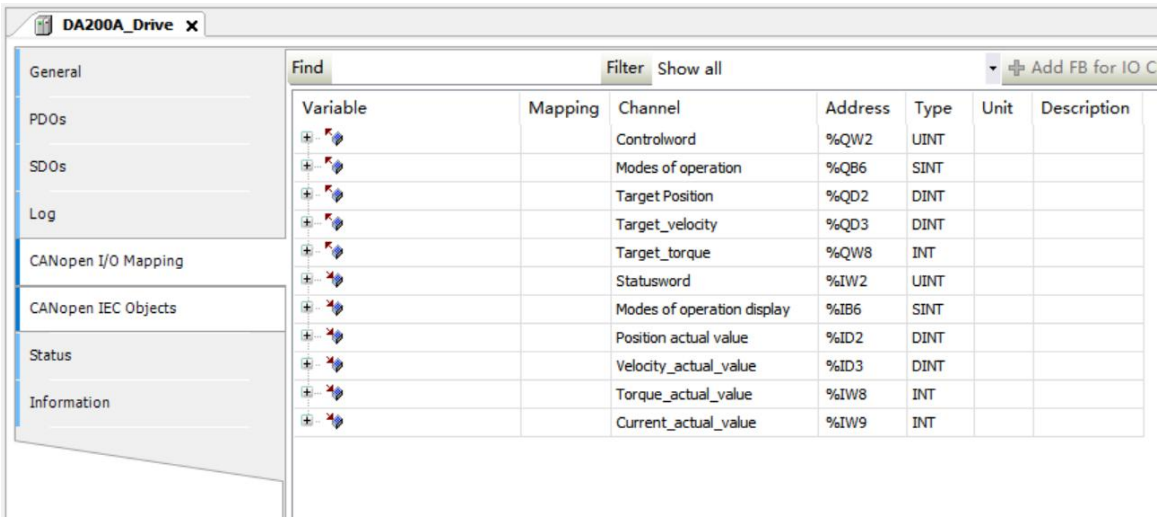
Figure 5-54 Mapping Addition



5.4.3.4 PDO Mapping View

After completing all CANopen communication configurations, you can view all configured communication variables in the **CANopen I/O Mapping** interface. Once online, you can monitor the status of these variables in real time, as shown in Figure 5-55.

Figure 5-55 CANopen I/O Mapping Status



5.4.4 SDO Communication Example

PDO-based data exchange is simple and straightforward. However, once configured as PDO data, the data occupies a certain amount of bus load. Therefore, avoid configuring too many slave devices on the same bus.

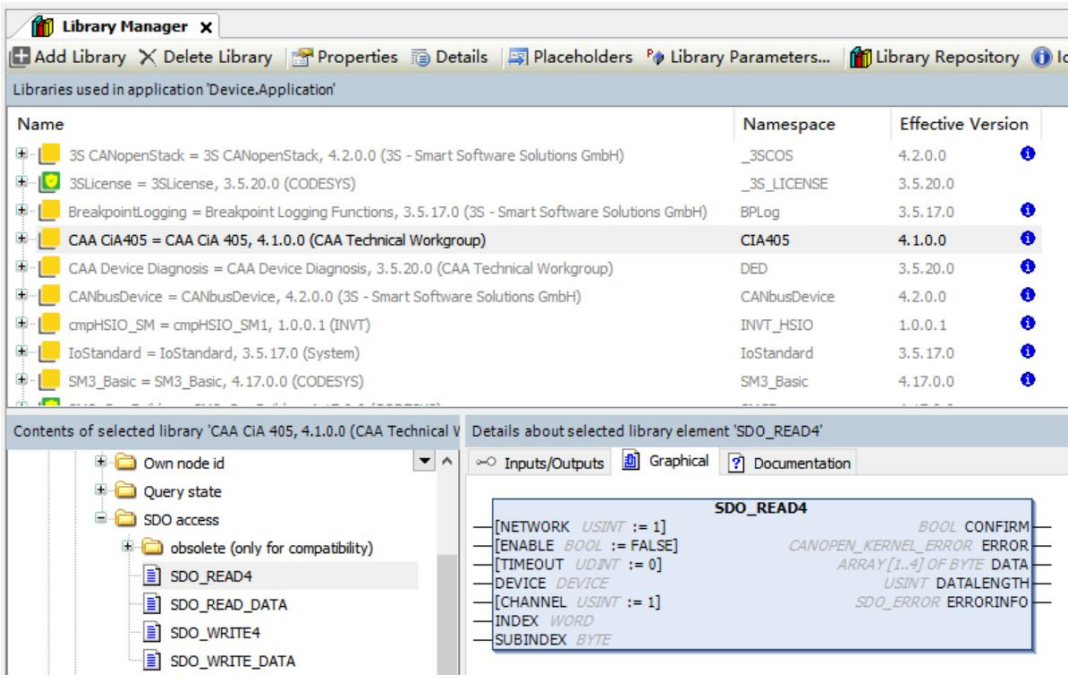
SDO communication is primarily used by the master node to configure parameters for slave nodes. It is designed to transmit large, low-priority data between devices, typically for configuring devices on a CANopen network. Before issuing NMT commands, the master uses SDO to read and write the PDO mapping data of each slave. After this is complete, the master sends the NMT start command, after which PDO operation is performed by default.

However, this does not mean that SDOs can only be used as a communication channel for parameters before the CANopen network officially starts. When too many PDOs are used on the bus resulting in a heavy bus load, SDO communication can be utilized through specific methods to transmit lower-priority data, such as reading the temperature value of the servo drive. The following sections describes how to implement this function by adding a specific function block in CoDeSys.

5.4.4.1 Library Configuration

In CODESYS, if the CANbus has already been added, the system will automatically include this library. Otherwise, you need to manually add the library CAA CiA405. Once added, you will see the **SDO access** folder within this library, as shown in the following figure.

In the **SDO access** folder, there are 4 CANopen SDO communication function blocks: The first group, SDO_READ_DATA and SDO_WRITE_DATA, is used to read/write parameter objects of any size. The second group is SDO_READ4 and SDO_WRITE4, which can read/write 4 bytes of an object.

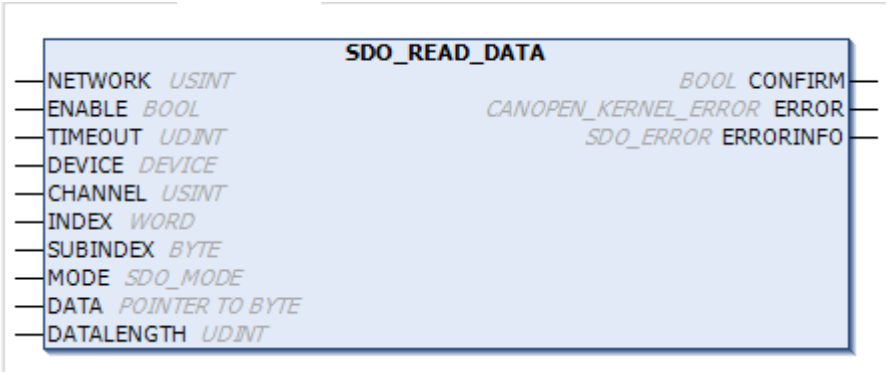


5.4.4.2 POU Addition

This section takes SDO_READ_DATA as an example to explain how to use this function block. Its graphical interface is shown in Figure 5-56. To use this function block, configure the required input and output

parameters. Table 5-2 provides detailed descriptions of each input and output parameter for this function block.

Figure 5-56 Library Addition



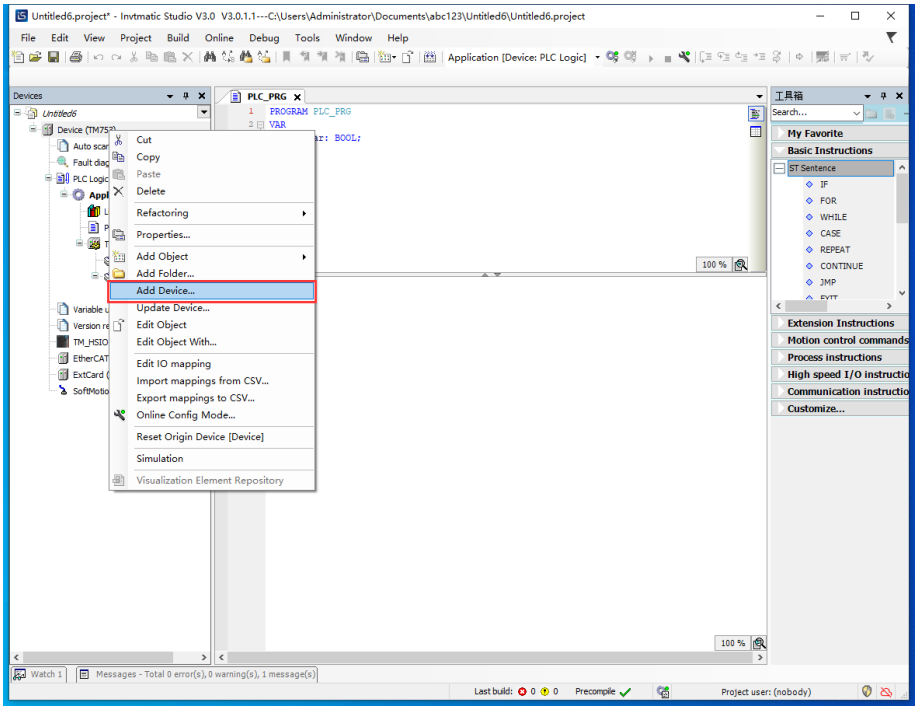
5.5 EtherNet/IP

5.5.1 EtherNet/IP Master Configuration

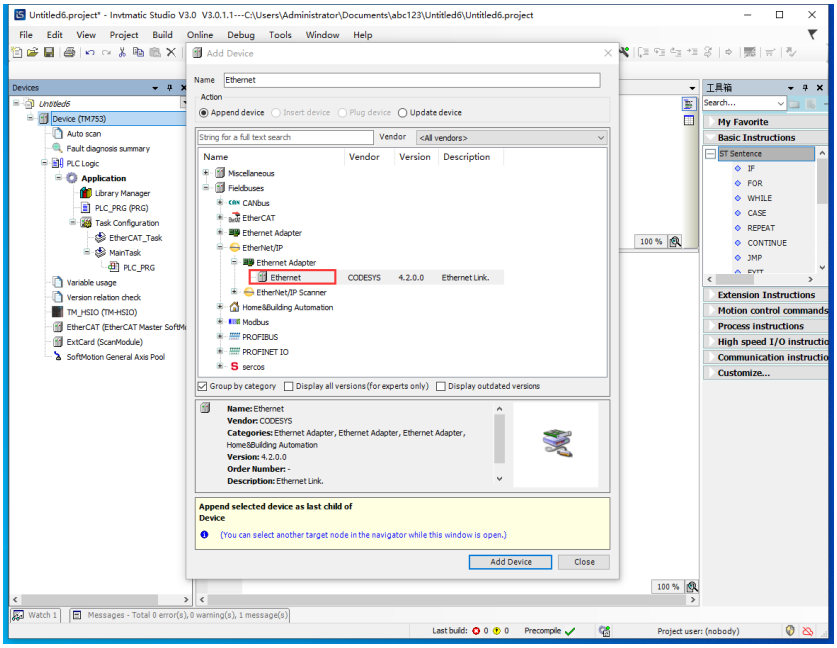
5.5.1.1 Configuration the Master

Assume that the TM753 is used as the master and the coupler is used as the slave. Configure the master as follows:

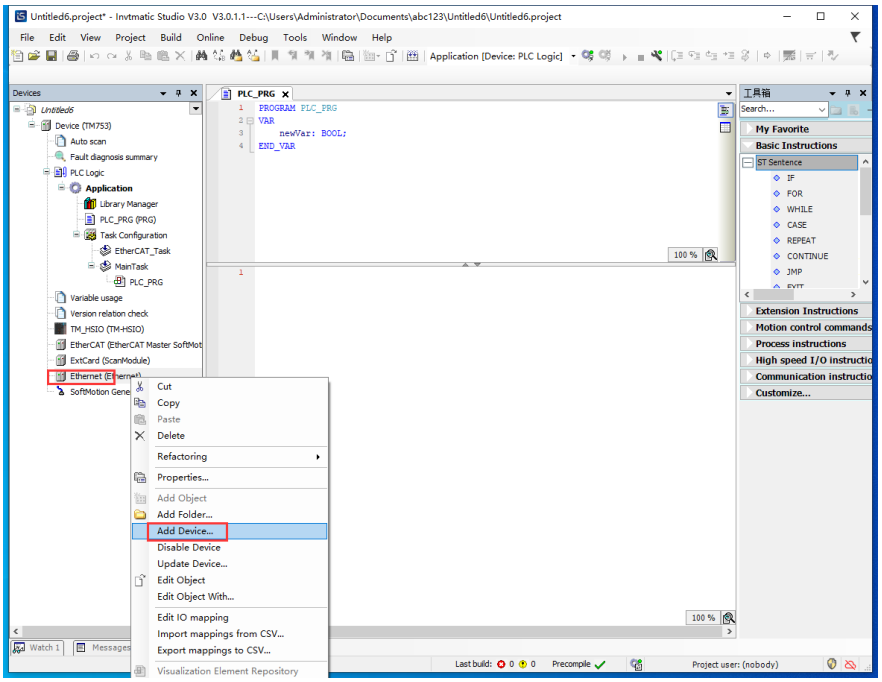
Step 1 Right-click **Device (TM753)** and select **Add Device**.



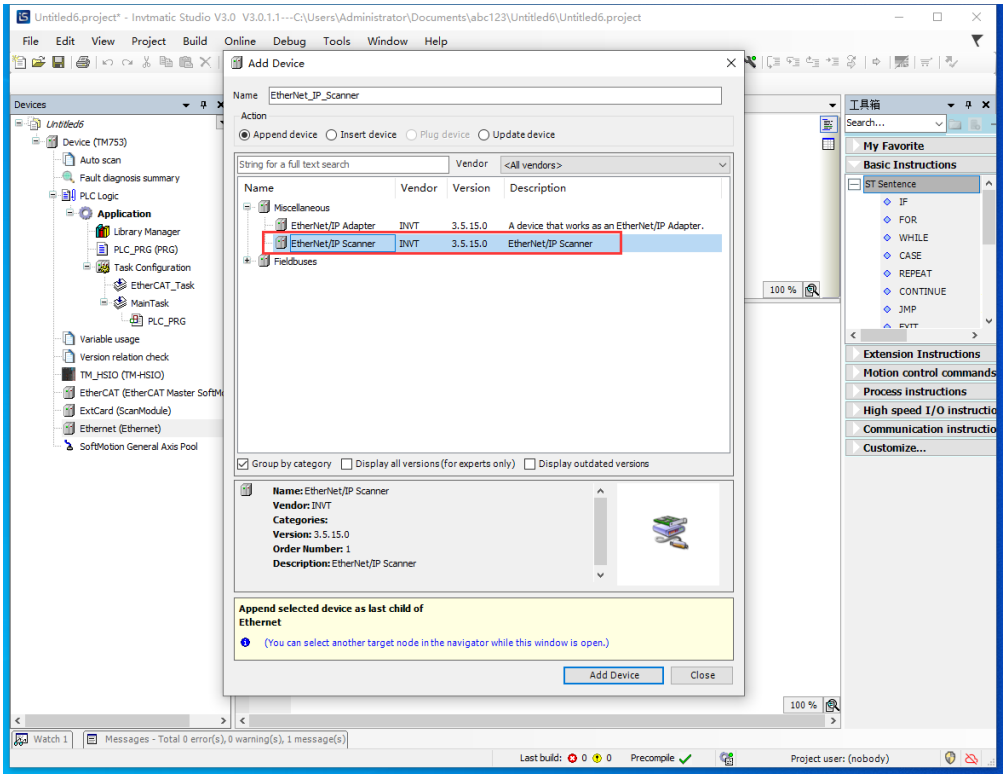
Step 2 In the **Add Device** dialog box, locate **Ethernet** and select **Add Device**.



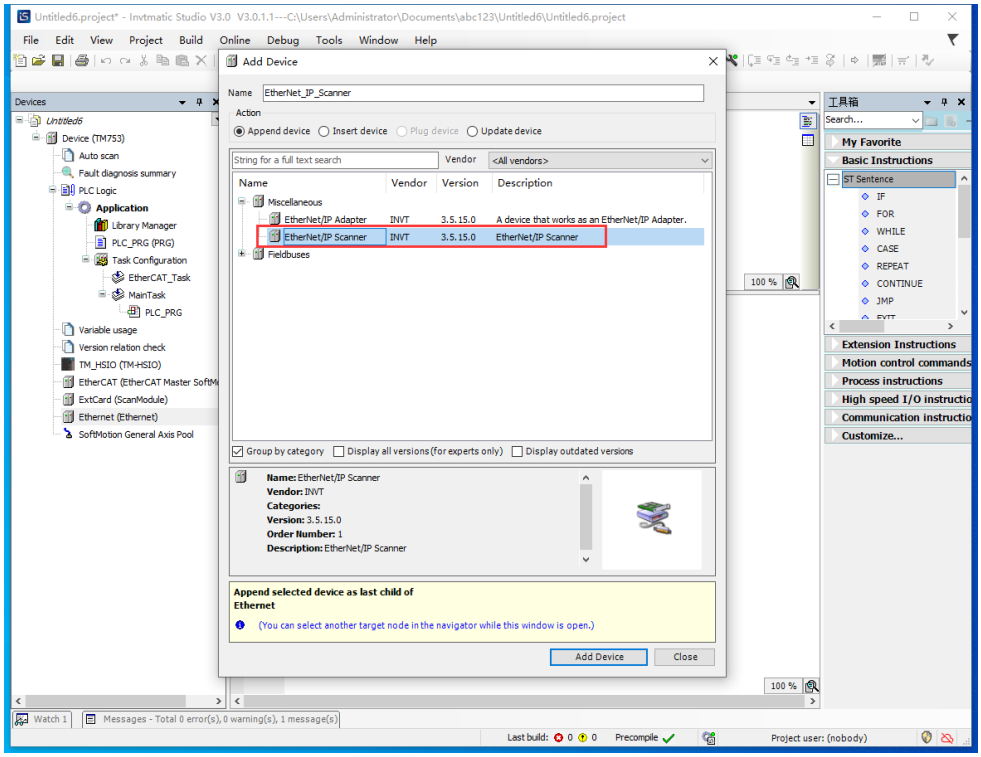
Step 3 Right-click **Ethernet** and select **Add Device**.



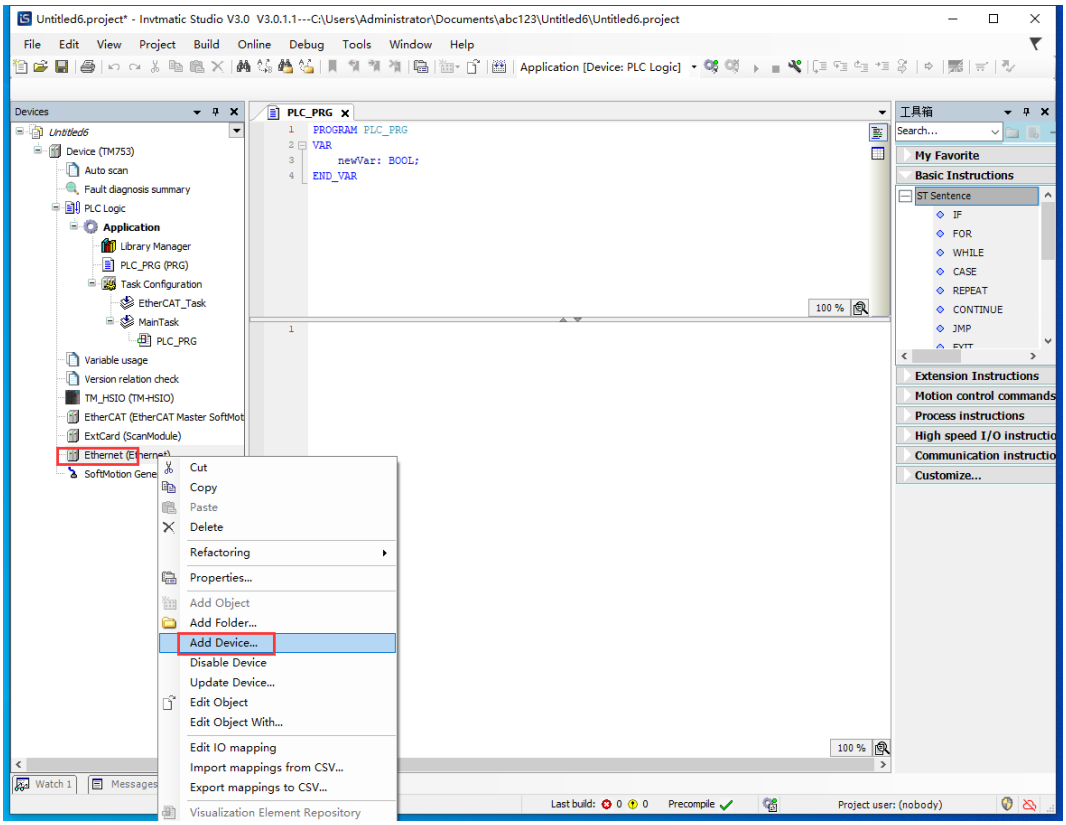
Step 4 In **Miscellaneous**, locate **EtherNet/IP Scanner**, and click **Add Device**.



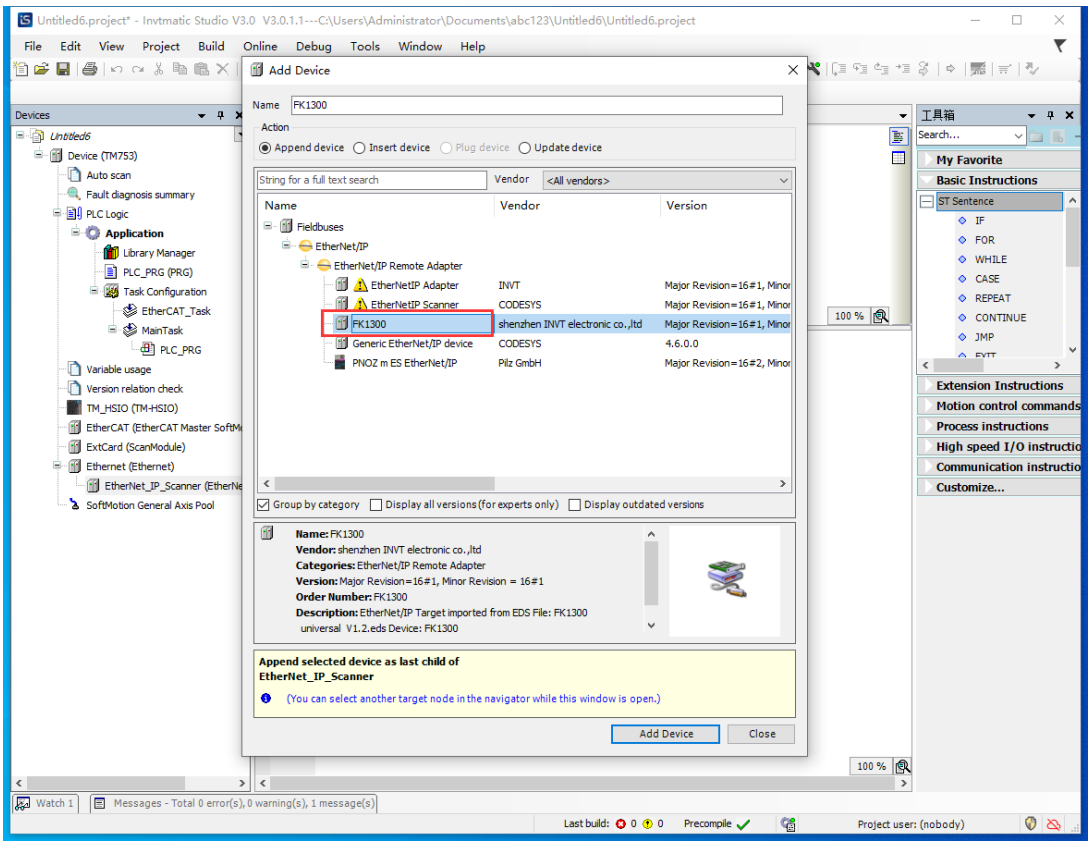
Step 5 The EtherNet/IP Scanner device is added successfully.



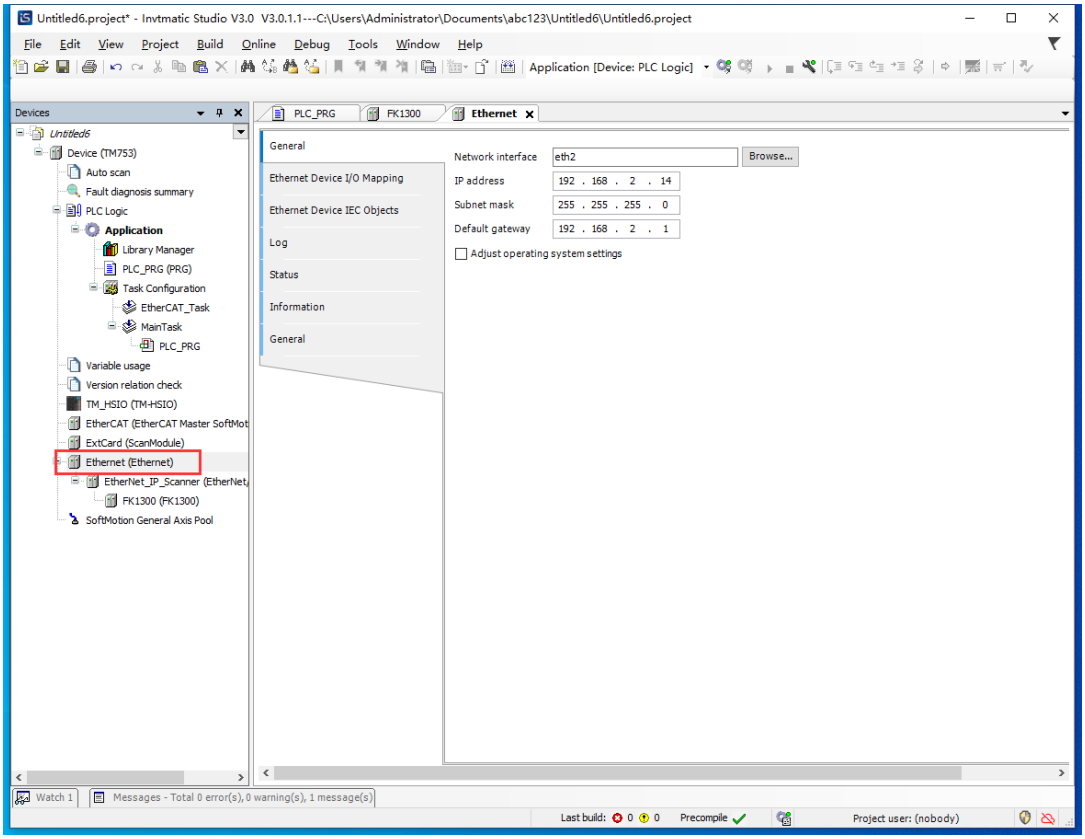
Step 6 Right-click **EtherNet/IP Scanner** and select **Add Device**.



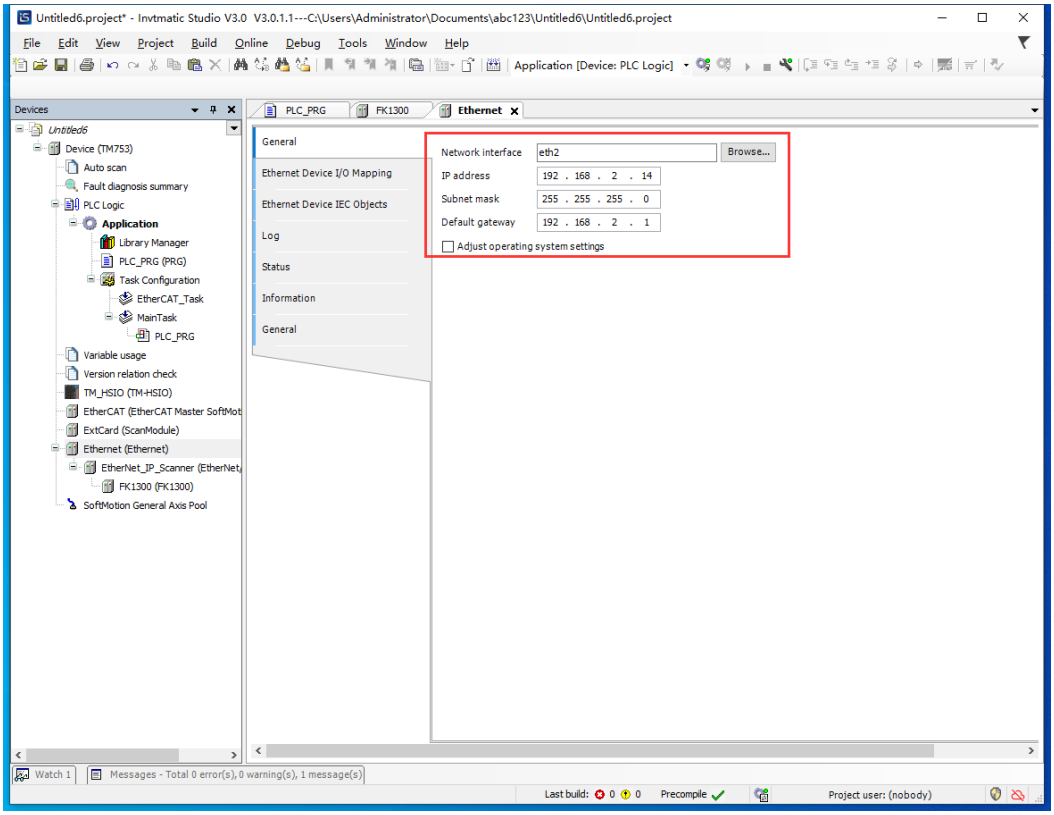
Step 7 Select **FK1300** (the generation procedure is explained later), and click **Add Device**.



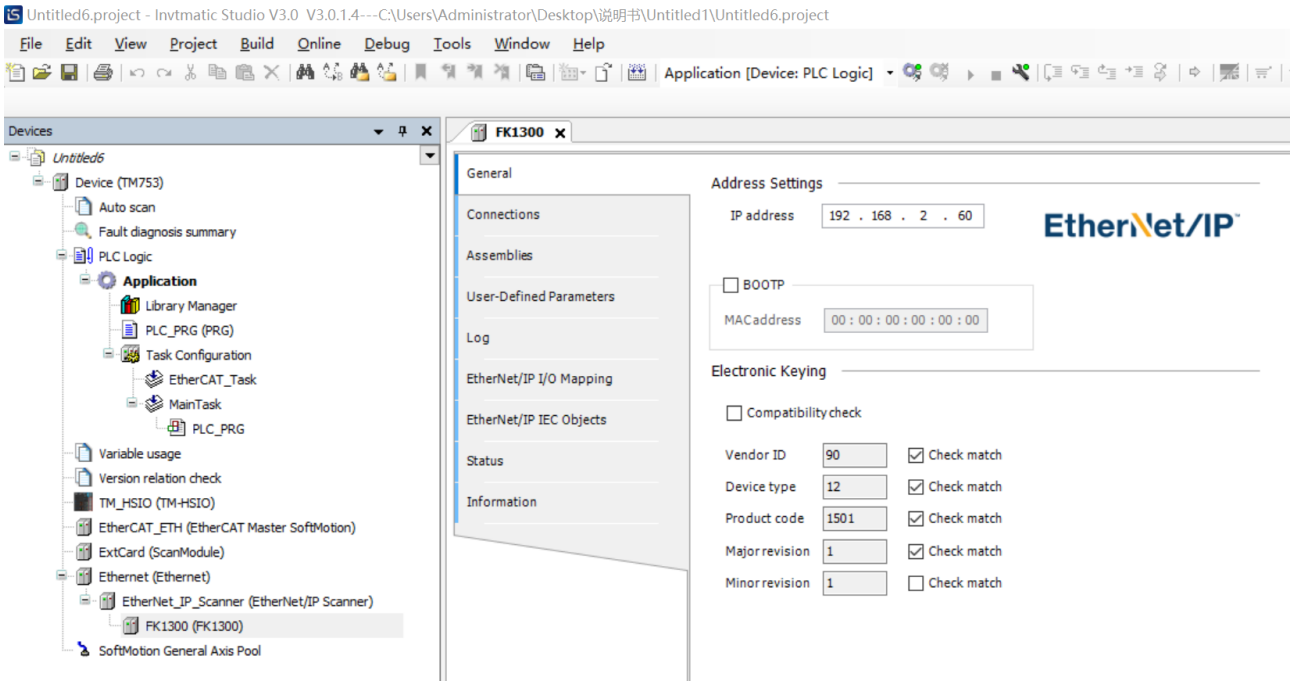
Step 8 As shown in the figure below, the configuration is complete.



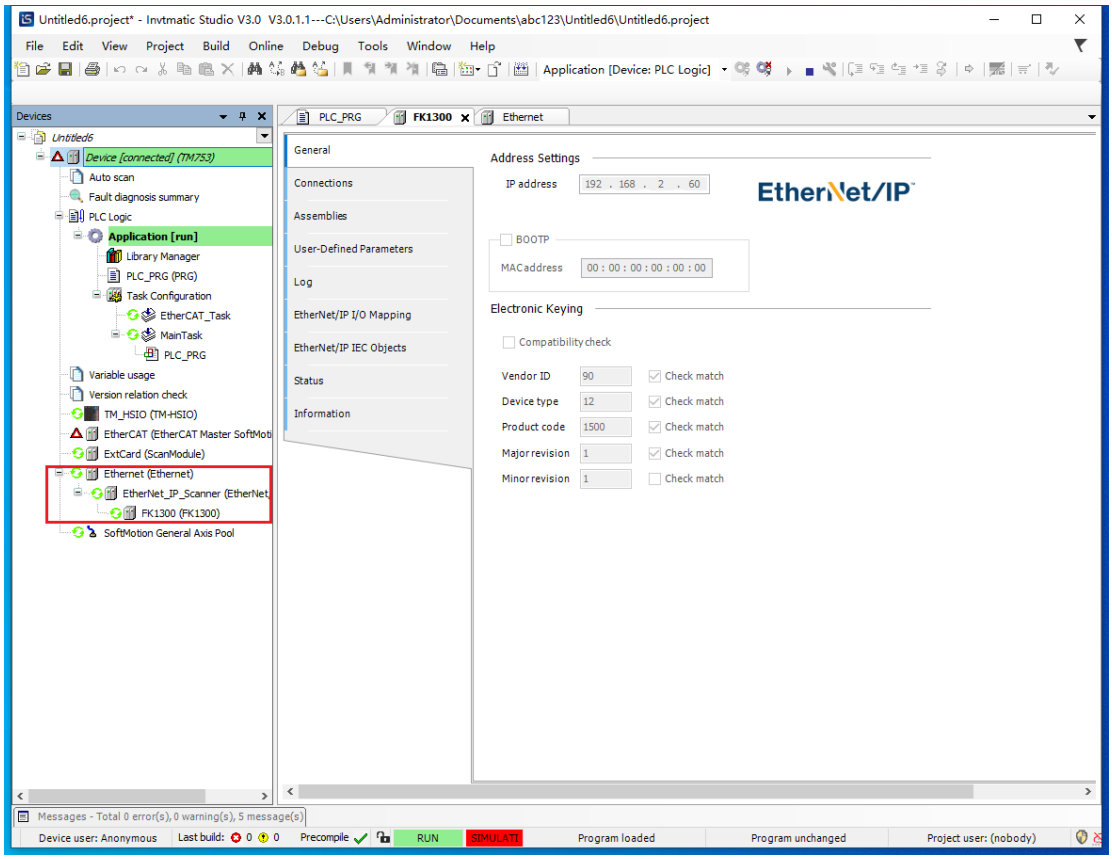
Step 9 Double-click **Ethernet** to set the network interface and IP address.



Step 10 Double-click **FK1300** to set its address (based on the actual address of the coupler).

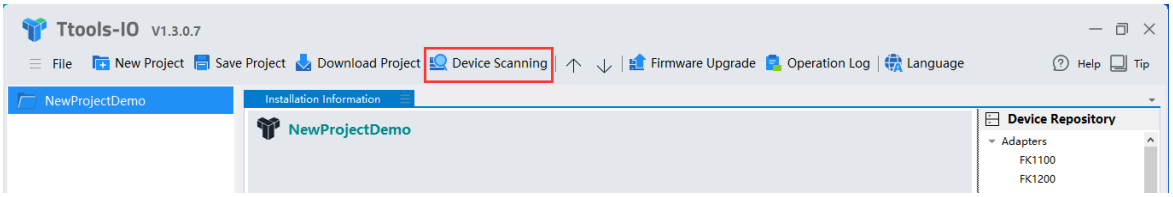


Step 11 With the TM753 as the master and the FK1300 coupler as the slave, EtherNet/IP communication is established successfully.

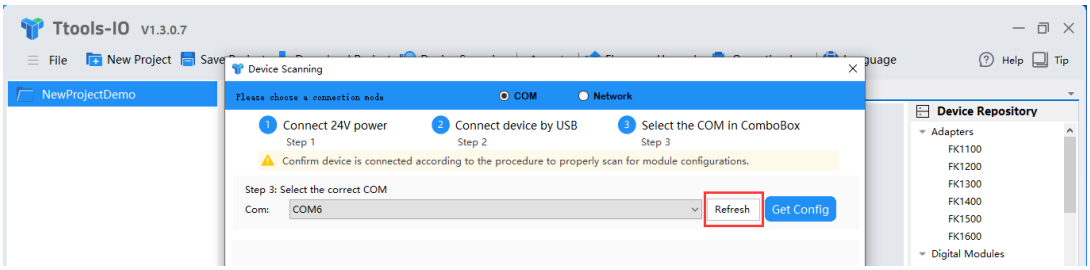


5.5.1.2 Generating a Coupler EDS File

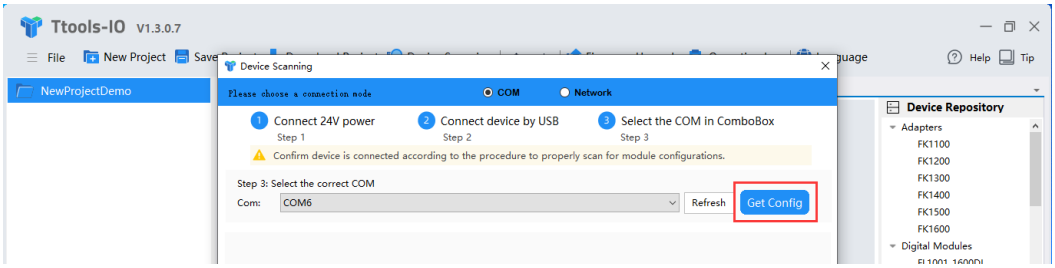
Step 1 Click  . When the coupler is used with other expansion modules, click **Device Scanning**.

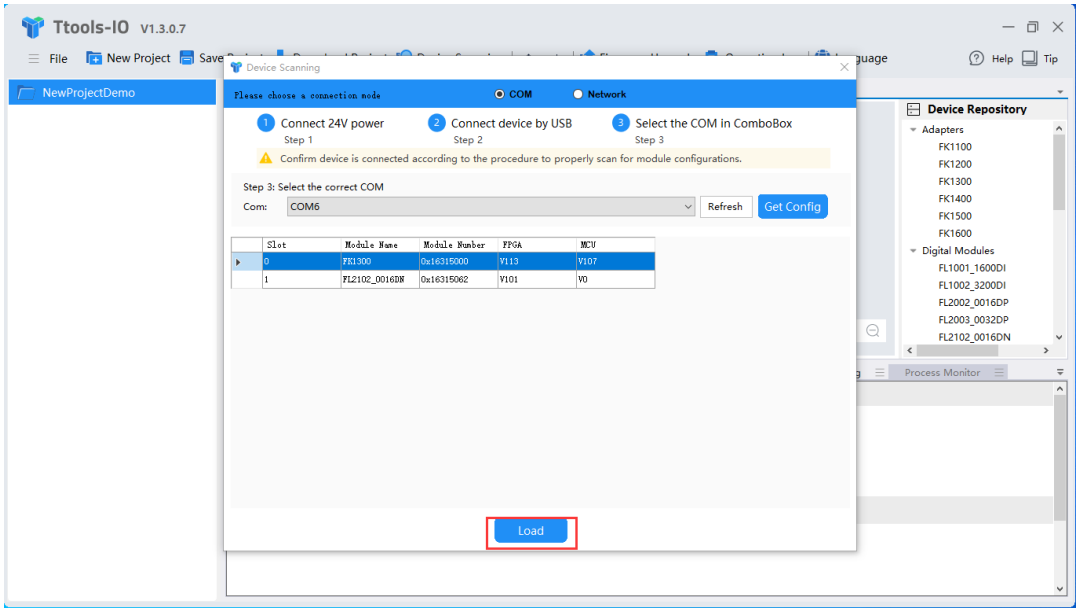


Step 2 After device scanning is complete, click **Refresh**.

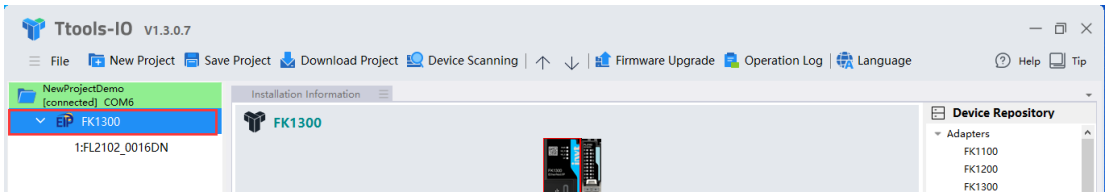


Step 3 Click **Get Config**.

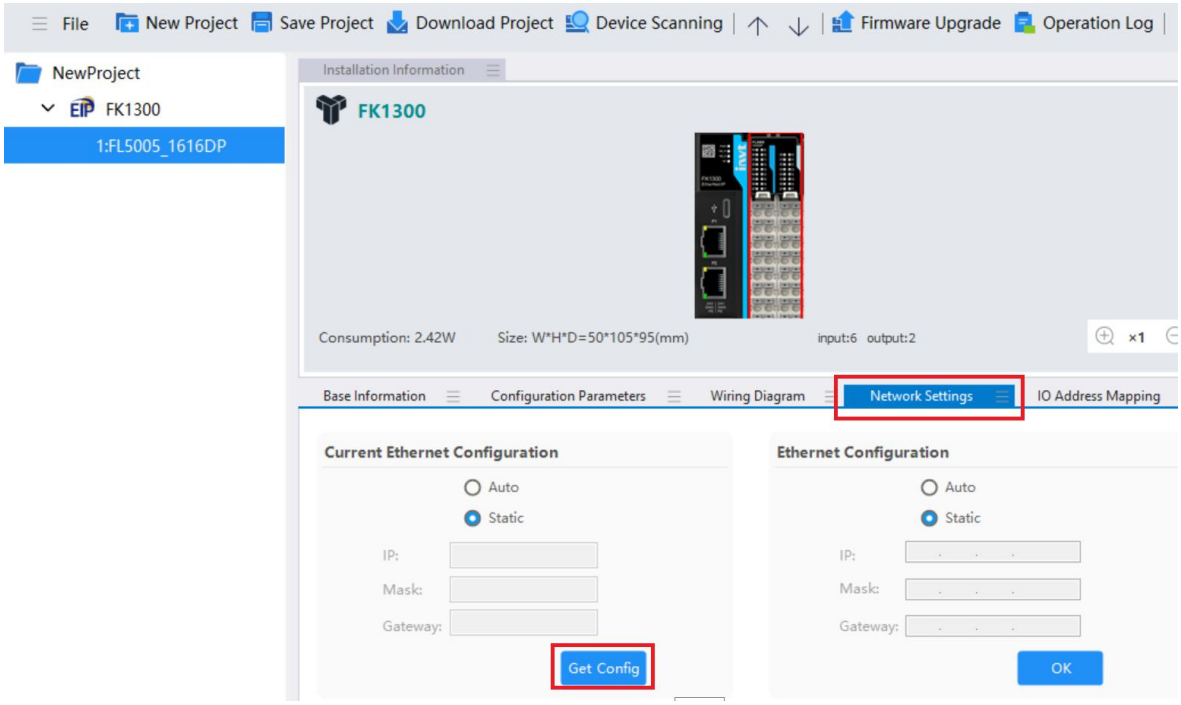




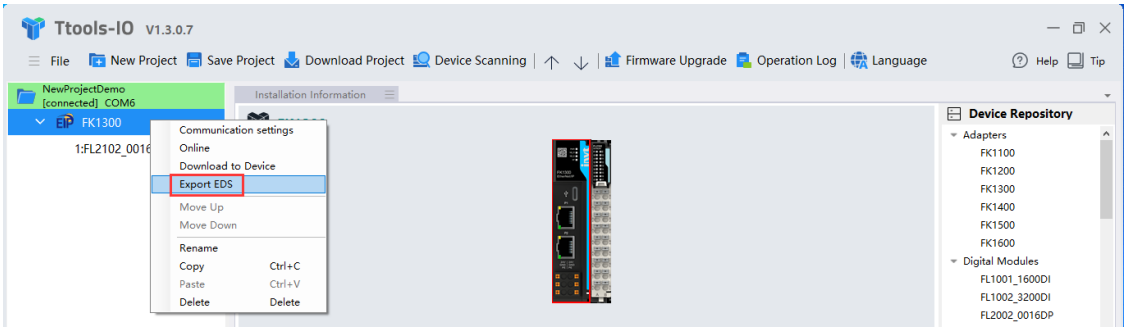
Step 4 The scanned device configuration is loaded successfully.



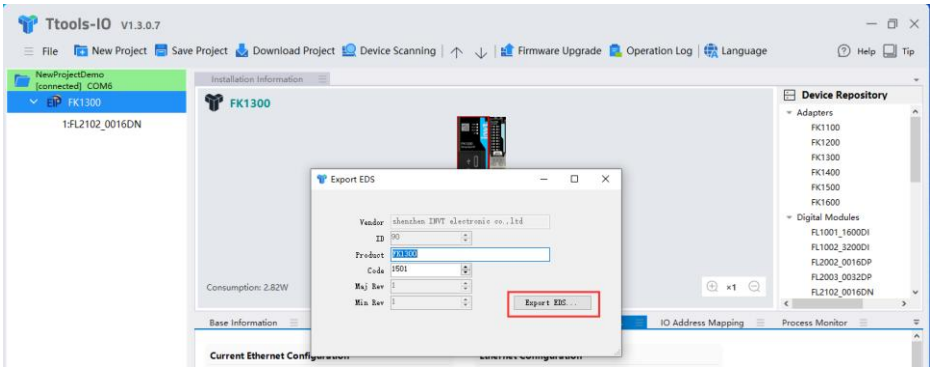
Step 5 Select **Network Settings** > **Get Config** to view the current network configuration of the coupler, which will be used later to set the coupler IP address.



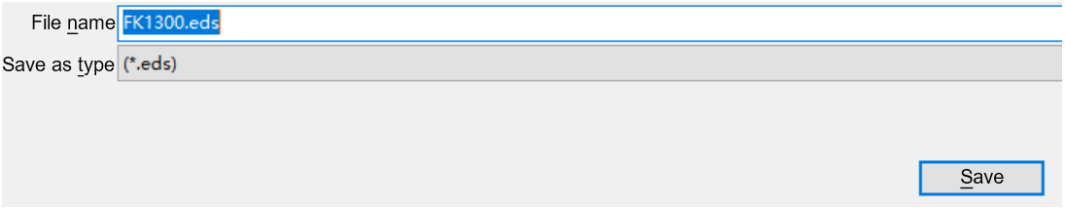
Step 6 Right-click **FK1300** and select **Export EDS**.



Step 7 Select **Export EDS**.

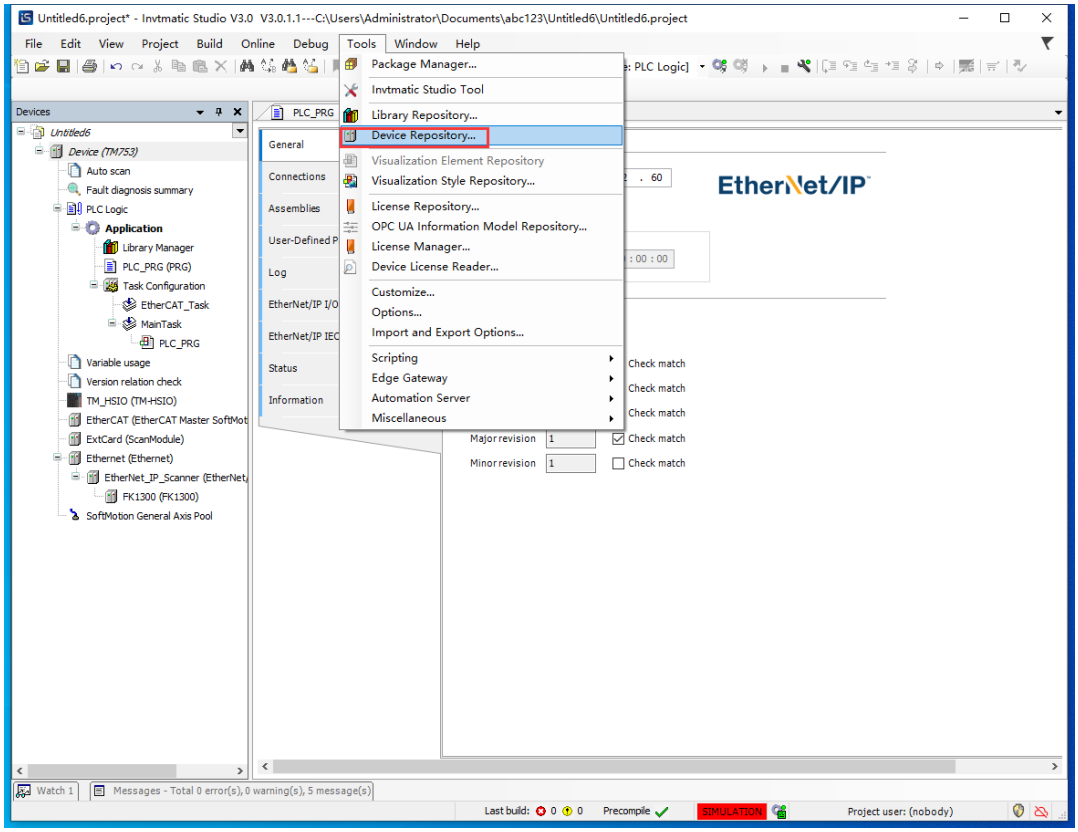


Step 8 After the FK1300.eds file is generated, save it to the specified path.

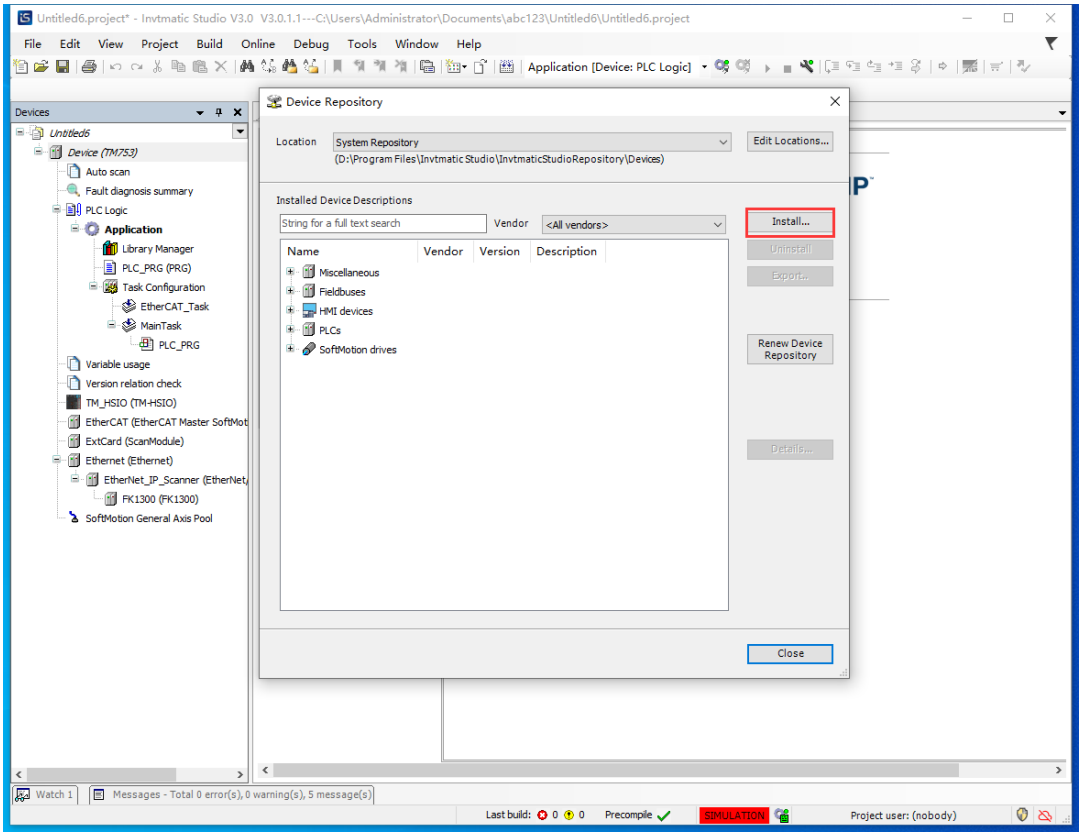


5.5.1.3 Installing a Coupler EDS File

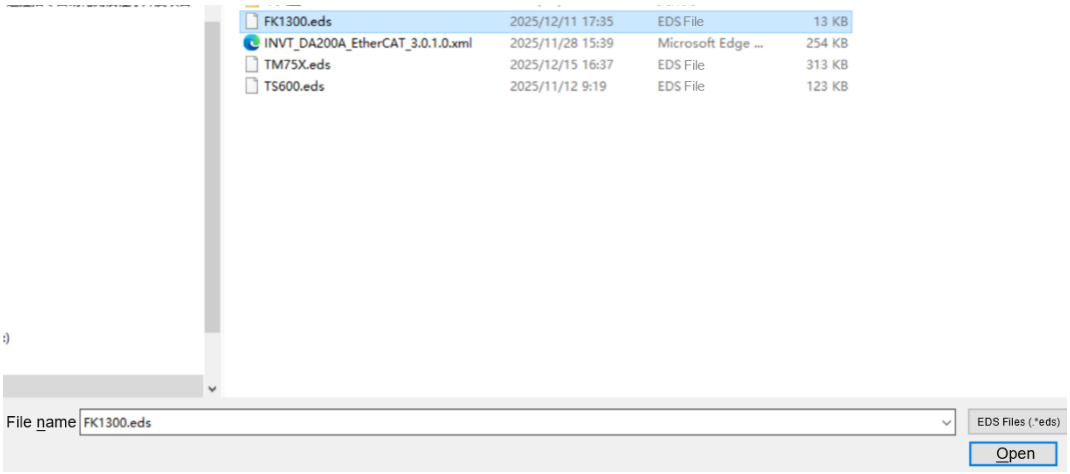
Step 1 Select **Tools** > **Device Repository**.



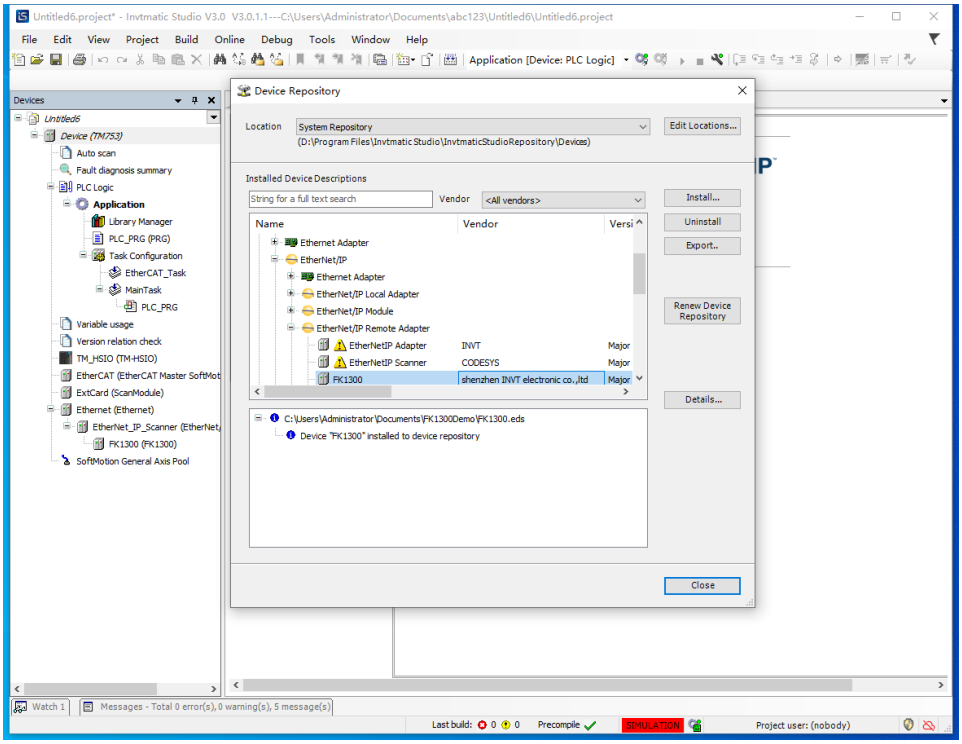
Step 2 Click **Install**.



Step 3 Navigate to the path where the FK1300.eds file is located and click **Open**.



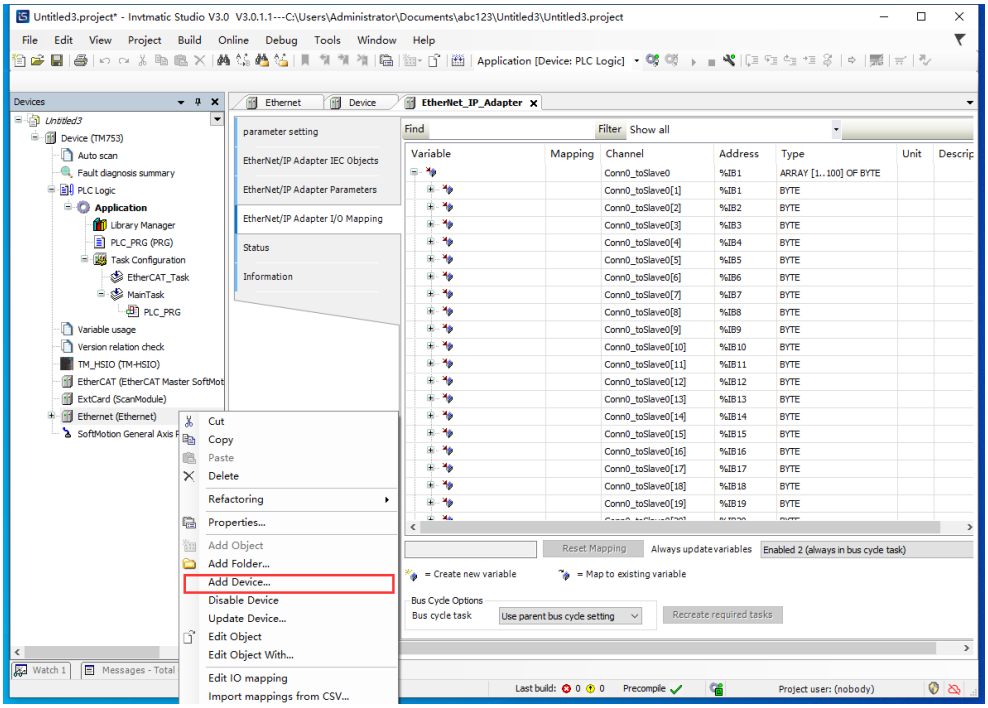
Step 4 The FK1300.eds file is installed successfully.



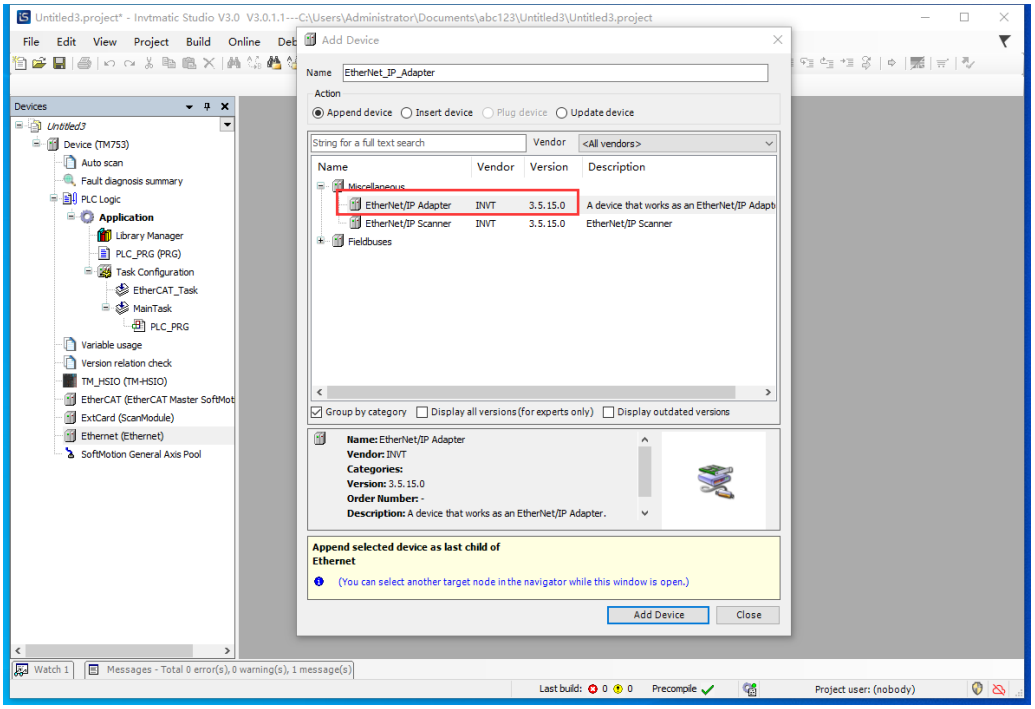
5.5.2 EtherNet/IP Slave Configuration

5.5.2.1 Configuring the Slave

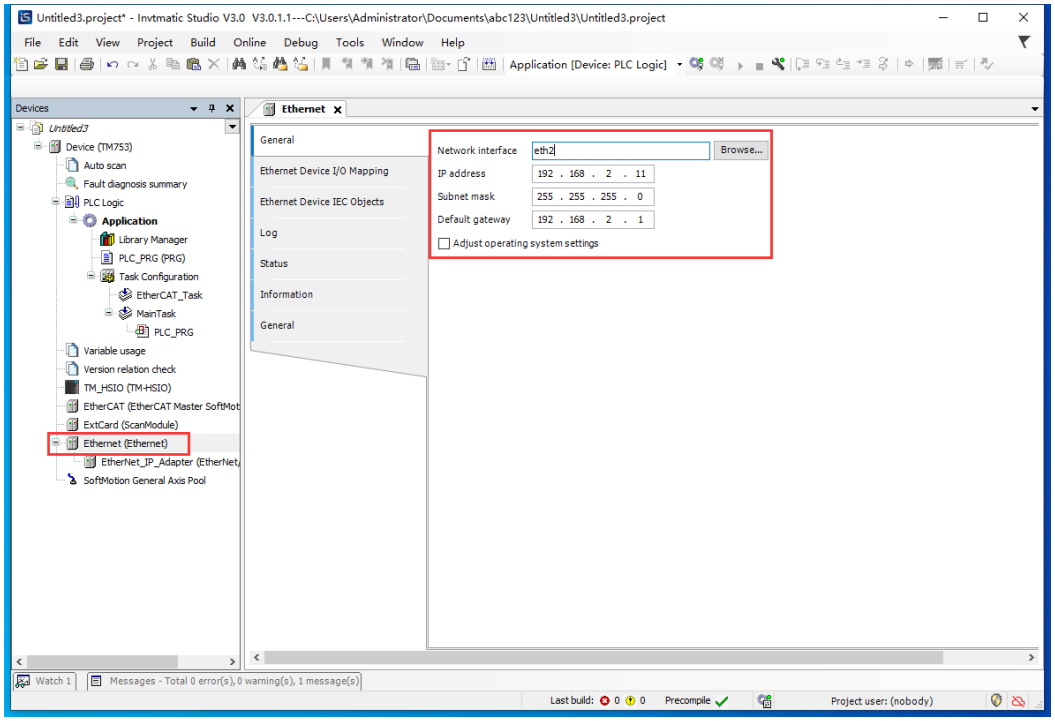
Assume that one TM753 is used as the master and another TM753 as the slave. The slave configuration procedure is as follows:



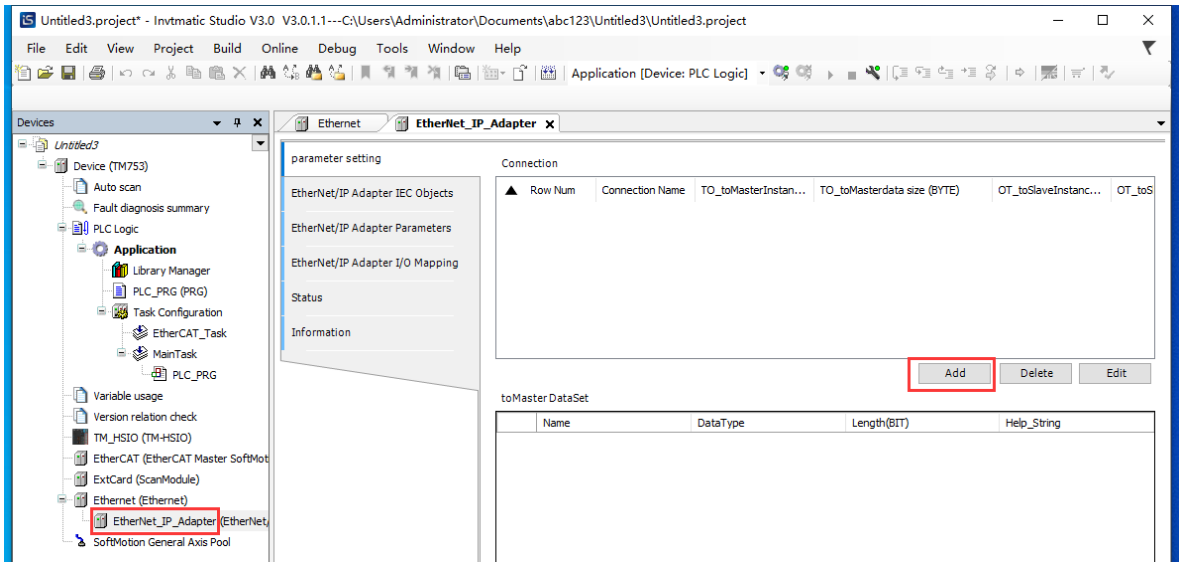
Step 1 Select **EtherNet/IP Adapter** and click **Add Device**.



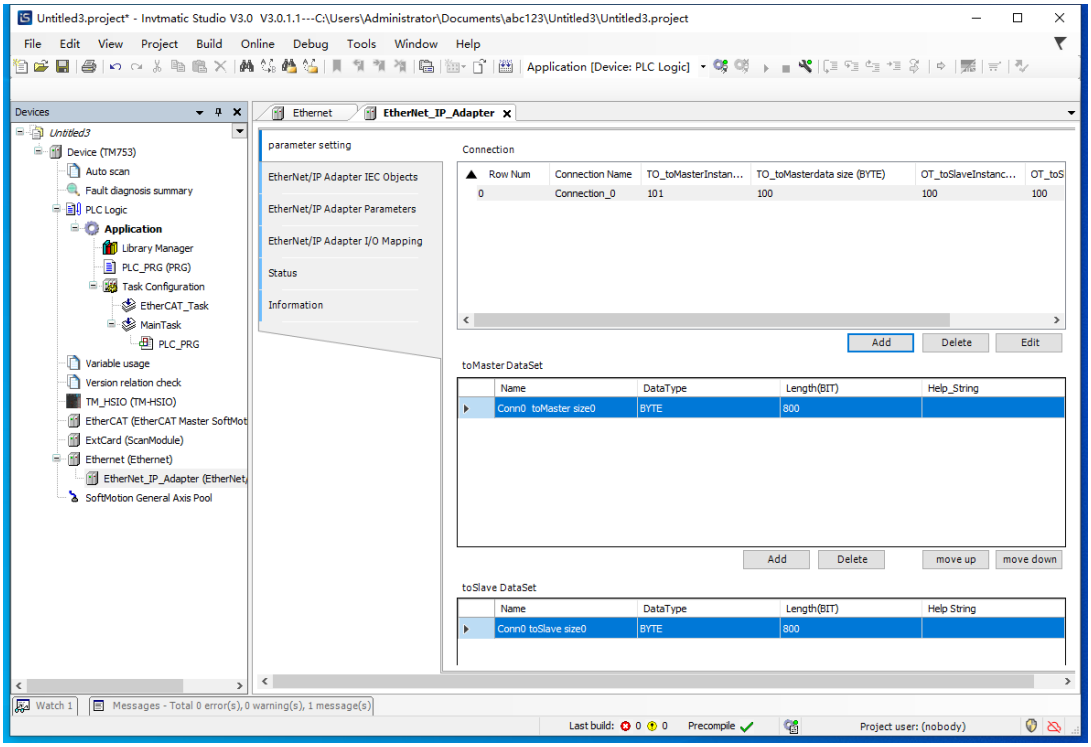
Step 2 After the slave configuration is added, double-click **Ethernet**, configure the network interface and IP address, and select **eth2** (select the IP address of the network interface that is connected to the master device).



Step 3 Click **Add**.

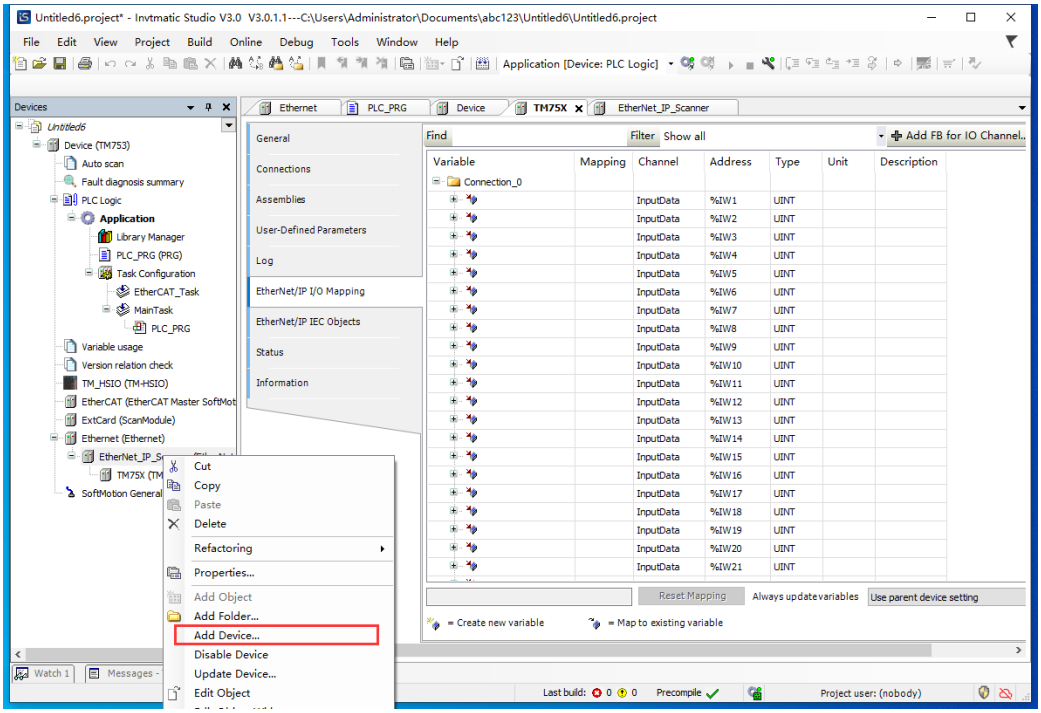


Step 4 The connection is added successfully.

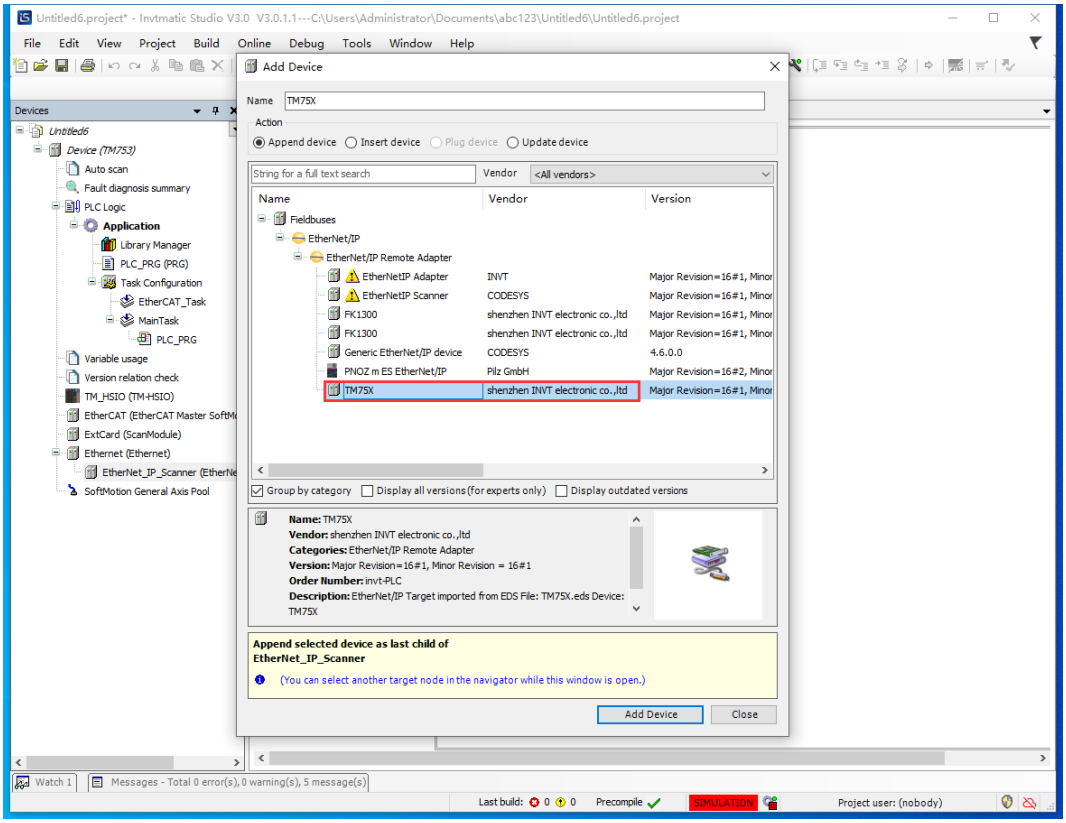


5.5.2.2 Configuring the Master

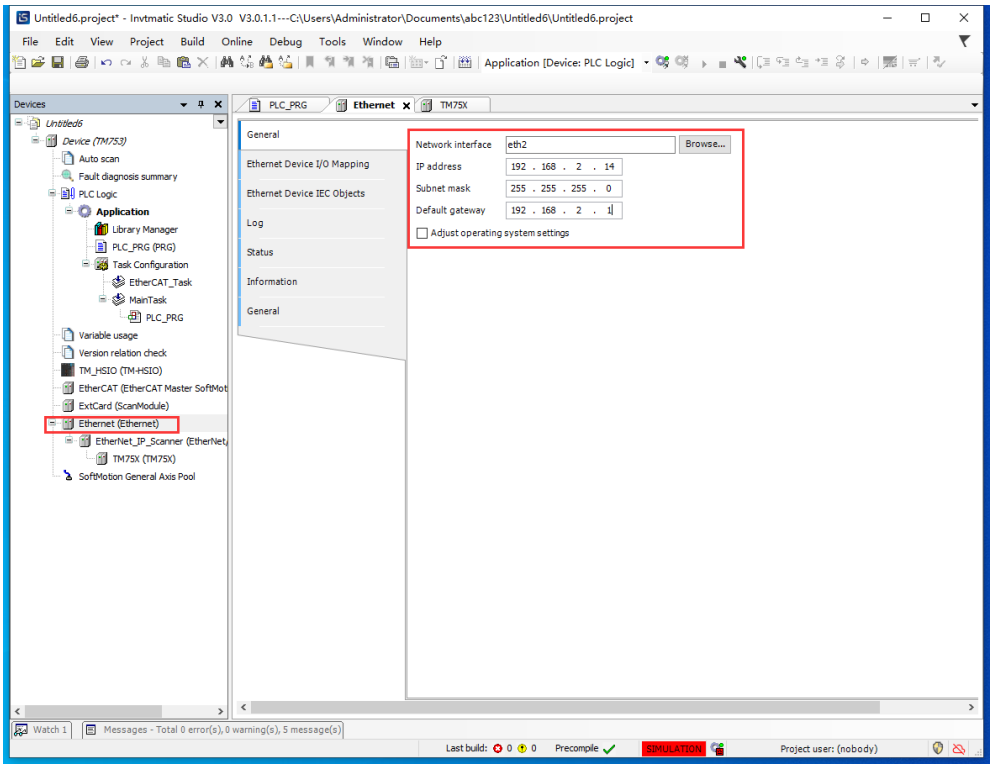
Step 1 When configuring TM753 as the master, the configuration procedure for EtherNet_IP_Scanner is the same as that described in section 5.5.1 EtherNet/IP Master Configuration. Right-click **EtherNet_IP_Scanner** and select **Add Device**.



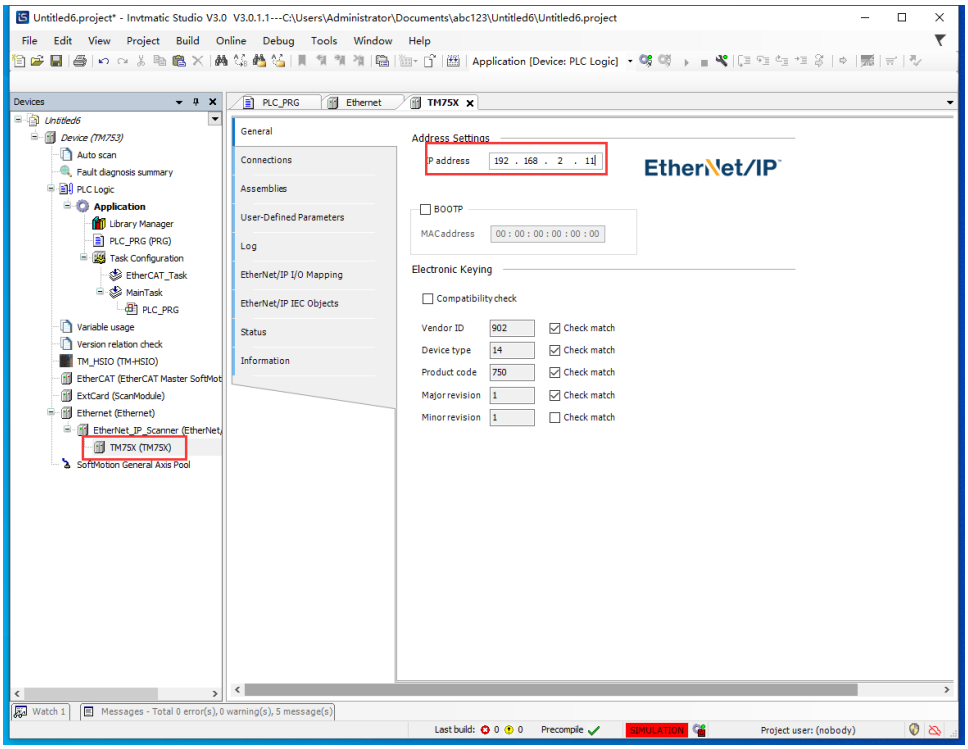
Step 2 Select **TM75X** and click **Add Device**.



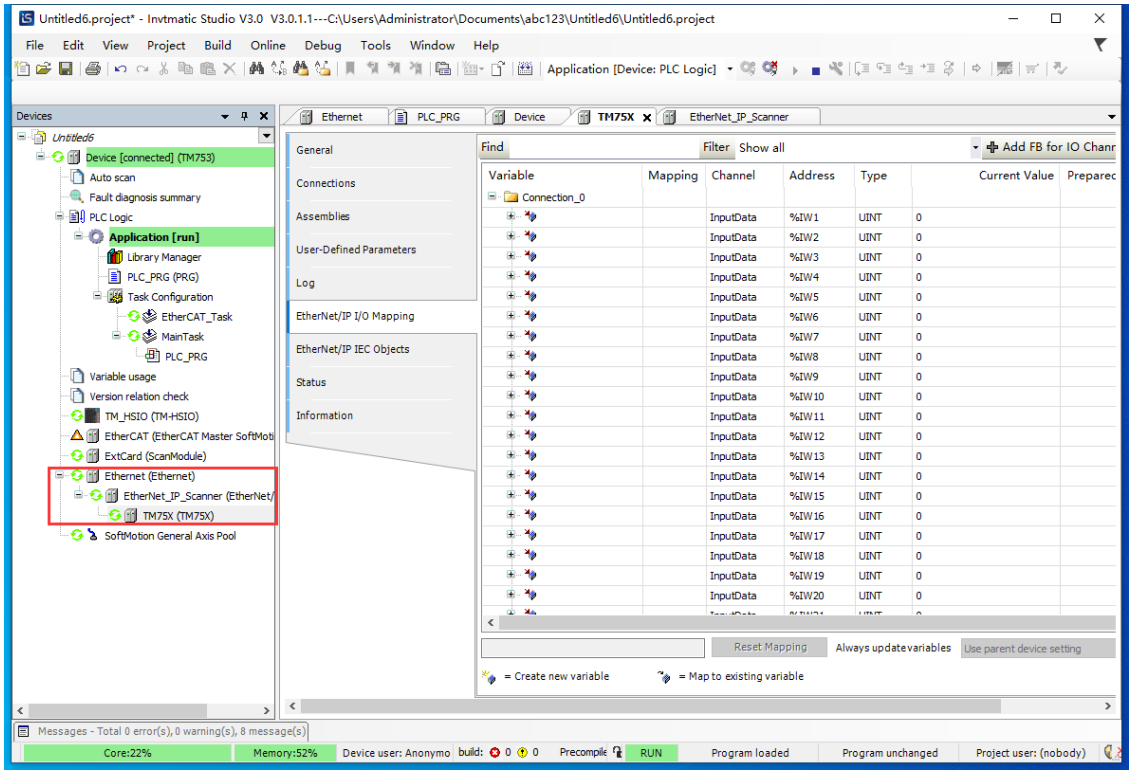
Step 3 After the master configuration is complete, double-click **Ethernet** and select **eth2** for **Network interface** (select the IP address of the master device network interface that is physically connected to the slave device).



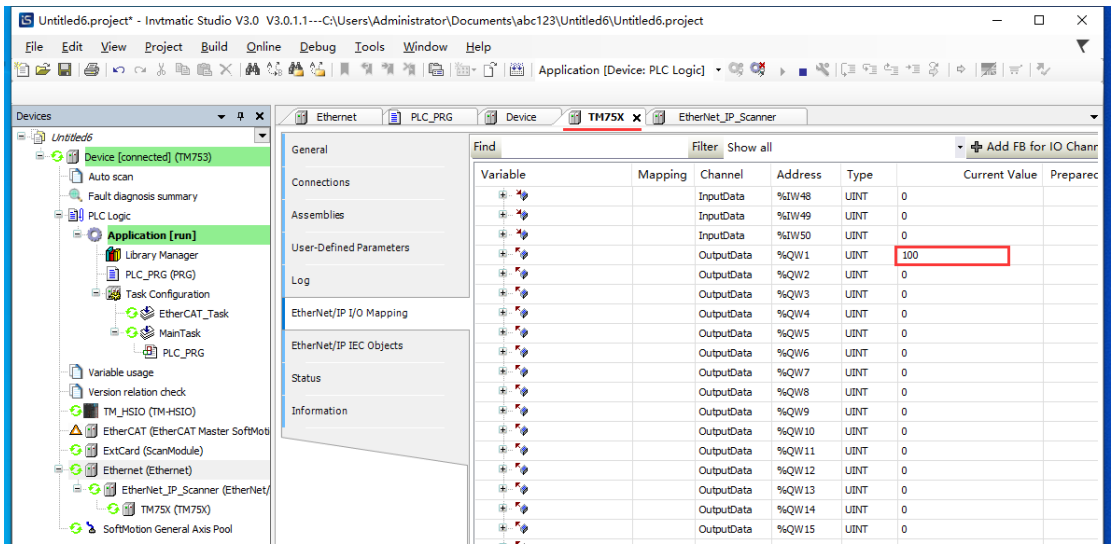
Step 4 Configure the IP address of the TM75X slave. Use the IP address of the network interface on the slave device that is physically connected to the master. This completes the TM master configuration.



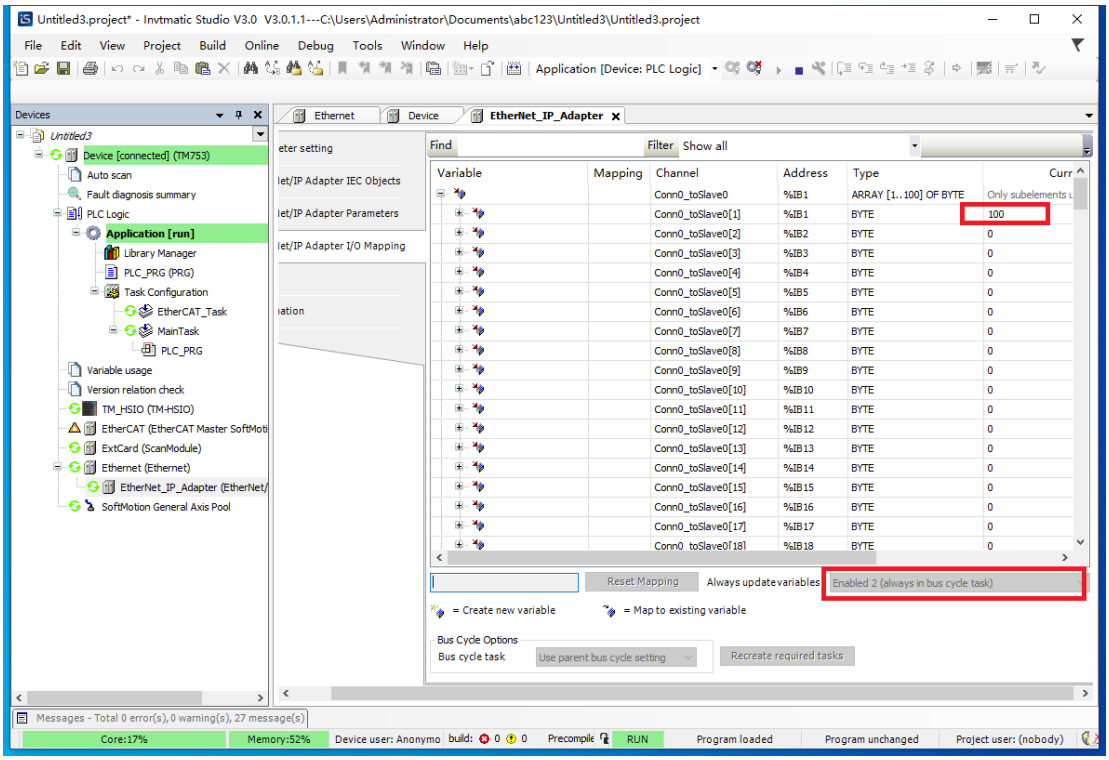
Step 5 Run the master program. The communication status is normal.



Step 6 Write 100 to the **OutputData** of the master device.



Step 7 Run the slave program and select **Enable 2 (always in bus cycle task)**. The slave displays the received value 100, indicating successful communication.



6 Diagnosis

6.1 Diagnosis Overview

Diagnosis is intended to quickly locate errors that occur during PLC operation and find solutions according to the error information and status. The diagnostic interface of Invtmatic Studio can only be obtained and displayed after logging in to the PLC. The Invtmatic Studio programming system supports the diagnosis of various communication devices and can generate fault information, offline information, and other information according to the actual running status of each communication device. The module types involved in fault diagnosis mainly include: CPU, ModbusRTU, ModbusTCP, EtherCAT, etc. The Invtmatic Studio programming system mainly provides four diagnosis routes: configuration diagnosis, diagnostic information list, device self-diagnostic information list, and diagnostic programming interface. All diagnoses are obtained through fault code analysis, and the fault codes correspond to the diagnostic programming interface.

In the configuration, different icons represent different diagnostic states of each communication module: running, standstill, and fault.

: Indicate the running state. The device has no faults.

: Indicate the standstill state. The device is not running.

: Indicate a fault state. In EtherCAT, it indicates that a fault has occurred but has been cleared. In Modbus, it is displayed as a device in a fault state.

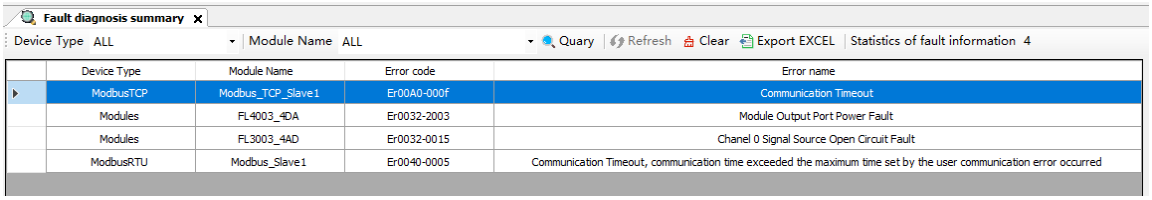
: Indicates a fault status. In CANopen, this means that a fault occurred but has been cleared.

The running state of the device can be directly viewed in the configuration.

6.1.1 Fault Diagnosis

Fault diagnosis can be used to display fault information of all devices, and provide detailed description of relevant fault information and methods for troubleshooting the causes. It can also provide more detailed diagnostic information for special situations. After the device is connected, double-click **Fault Diagnosis Summary** in the device tree to open the device fault diagnosis interface. The fault diagnosis interface is shown in Figure 6-1.

Figure 6-1 Fault Diagnosis Summary



The Device Type window displays the current fault type and provide the fault display filter function, which can display fault information by device type. Device types include CPU module, Modbus module, Modbus TCP module, and local module. You can select a different device type, and the diagnostic display list will show the corresponding type of diagnosis. All device diagnoses are displayed by default.

- Search: Search for matching fault information based on the device type or module name.
- Refresh: Used to refresh device fault information.
- Clear: Clear the fault information in the table.
- Export to EXCEL: Export fault information in the table.
- Fault Information List: mainly used to display specific module fault information, including device type, module name, and fault information. Device Type: Filter a certain type of faulty bus device;
  Module Name: Filter faulty devices with a specific name.
- Detailed Information window: When a certain piece of fault information is selected in the Fault Information List, the detailed information of the fault will be displayed in the detailed information window, which includes three options: Error Details, Troubleshooting, and In-depth Diagnosis. The first column of Error Details describes the possible cause of the fault, followed by four additional items, which are mainly used to provide more information about the fault; Troubleshooting is used to provide specific operation methods for cause investigation; and In-depth Diagnosis describes some complex errors that require more detailed information for troubleshooting.

6.2 Device Self-diagnostic Information List

6.2.1 Modbus RTU Diagnosis

Modbus RTU supports two serial ports: Modbus serial ports 0 and 1. Modbus serial port 0 or 1 can be used as a Modbus master or slave.

When the Modbus serial port is used as a master, slaves (remote slaves) can be added to the master. On the slave configuration interface, each slave has an “Error Diagnosis” interface, which details the parameter with an error.

When the Modbus serial port is used as a slave, there is also an “Error Diagnosis” interface to display communication errors between the slave and the master.

6.2.2 Modbus TCP Diagnosis

The PLC can be used as a Modbus TCP master or slave. When Modbus TCP is used as the master, slaves (remote slaves) can be added to the master. On the slave configuration interface, there is an “Error Diagnosis” interface, which details the parameter with an error.

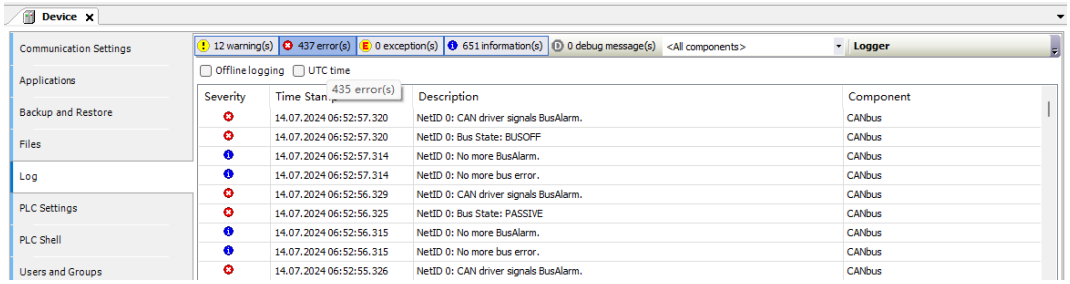
When ModbusTCP is used as a slave, there is also an “Error Diagnosis” interface to display communication errors between the slave and the master.

6.3 Online Log

An online log displays program, device, and system-related log information online in real time after connection to the PLC device. This function is used to help you quickly locate errors, solve problems timely, and ensure the normal operation of the device.

Login to the PLC: Click Login. When the software and PLC are connected, the log function will be started. The log interface is as shown in Figure 6-2.

Figure 6-2 PLC Log



The meanings of the parameters in the Toolbar of the interface are listed in the table below.

Parameter Name	Parameter Description
Warning/Error/Exception/Info/Debugging Information	Display and filter different levels of information during diagnosis.
Clear	Clear the currently displayed diagnostic information. After clearing, when there is new log information, the system will refresh the latest information.
Export	Export all log information in CSV format.
Import	Import log information.
Refresh	Refresh and view log information.
All Components	View the log information of all components, or choose to display the log information of a single component.
Offline log	Import the saved log information in CSV format in offline state.

The meanings of the parameters in the table of the interface are listed in the table below.

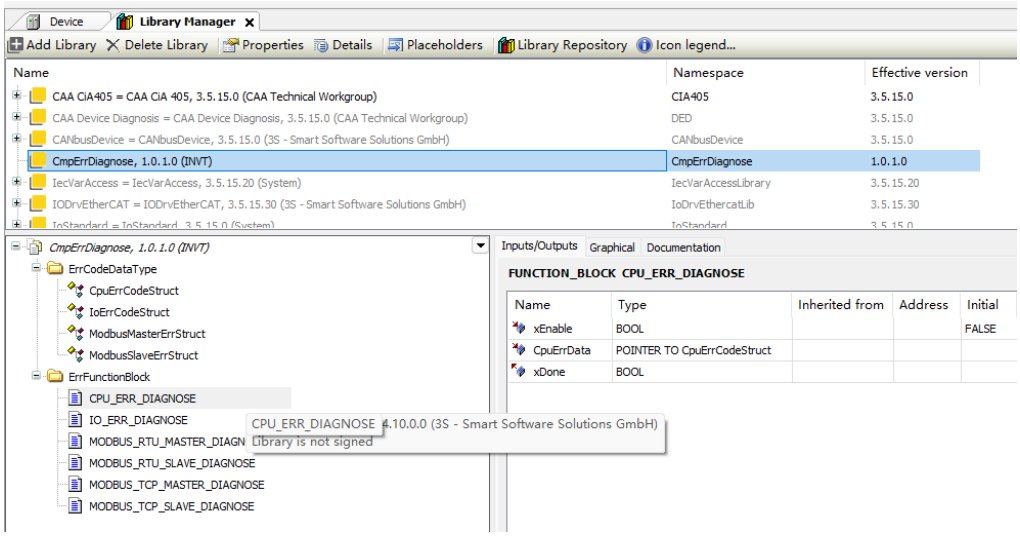
Parameter Name	Parameter Description
Class	Display the level of information.
Time	Display the time when the information is generated.
Description	Describe the phenomenon and cause of the error.
Component	Display the component where the error occurred.

6.4 Diagnostic Programming Interface

The diagnostic interface library CmpErrDiagnose provides a solution for obtaining diagnosis in the user program. It can determine the diagnostic information of each device module in the user program and make relevant processing.

The diagnostic programming interface exists in the form of a library and can be added in **Library Manager**, as shown in Figure 6-3.

Figure 6-3 Adding a Diagnostic Programming Interface Library



The diagnostic programming interfaces corresponding to CPU, ModbusRTU, and ModbusTCP are provided. Each type of diagnosis corresponds to a set of function blocks for obtaining the corresponding error codes. The custom diagnostic results and states in the diagnostic data are displayed in **ErrCodDataType** (as shown in Figure 6-3). For example, Figure 6-4 shows the Modbus master diagnostic function block, with three slaves added to the serial port 1. The strRtuDia is declared as a structure array, and each array corresponds to the fault information of a slave channel. The diagnostic result is that the slave 1 has two channel errors, and slaves 2 and 3 each have one channel error. Figure 6-5 shows the parameters of the Modbus master diagnosis function block.

Figure 6-4 Adding a Modbus Master Diagnostic Function Block

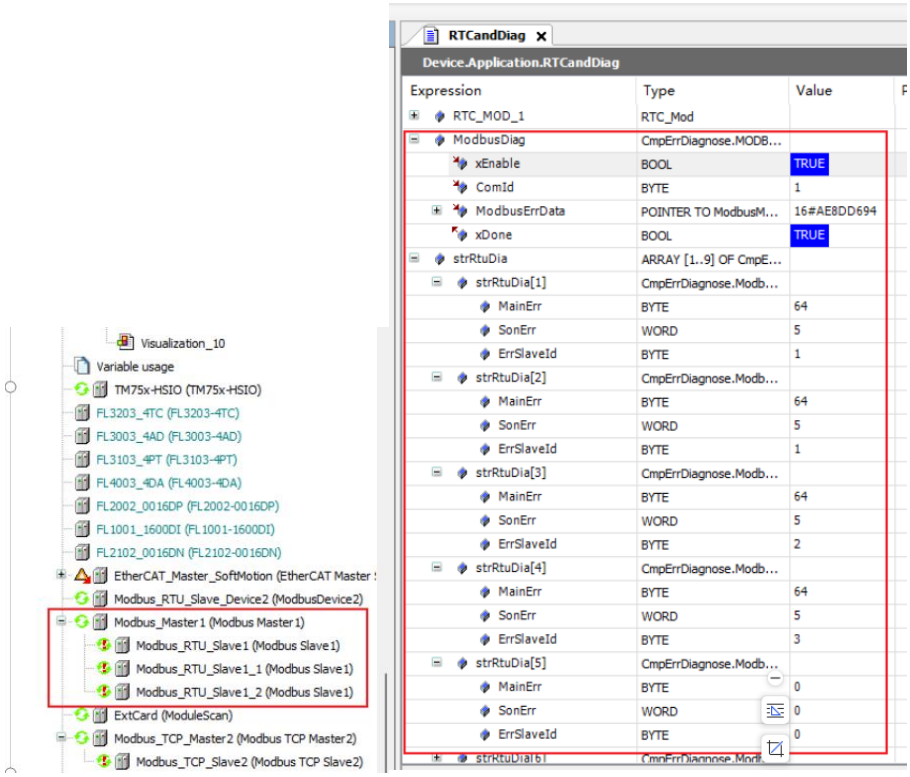


Figure 6-5 Description of Modbus Master Diagnostic Function Block Parameters

Inputs/Outputs | Graphical | Documentation

FUNCTION_BLOCK MODBUS_RTU_MASTER_DIAGNOSE

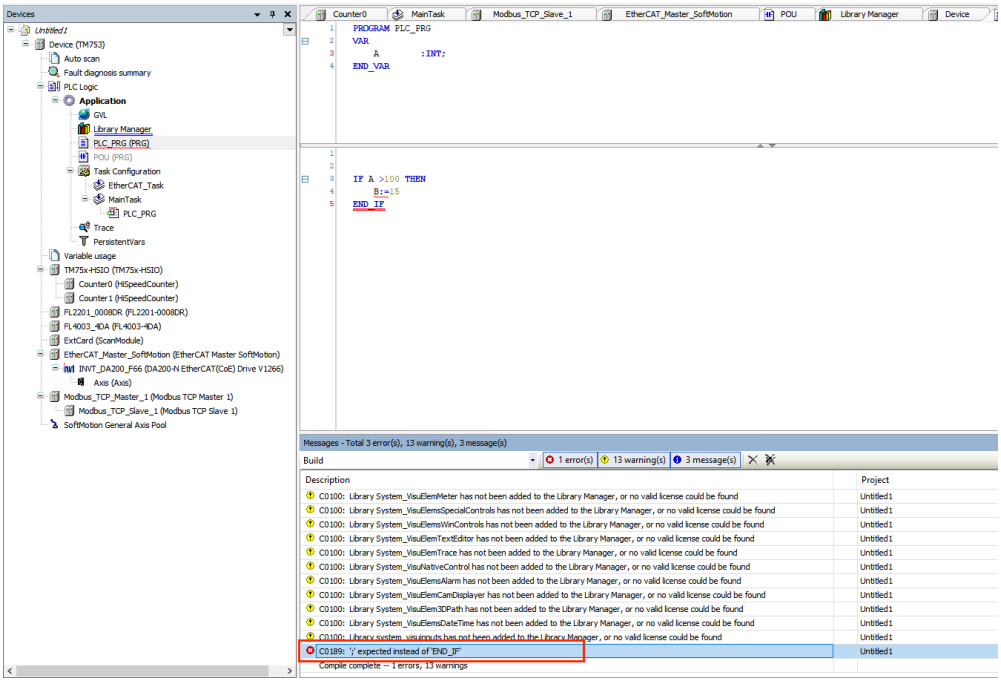
Name	Type	Inherited from	Address	Initial
xEnable	BOOL			FALSE
ComId	BYTE			
ModbusErrData	POINTER TO ModbusMasterErrStruct			
xDone	BOOL			

6.5 Application Diagnosis

The programming language used by Invtmatic Studio has the characteristics of high execution efficiency and flexible programming, but it also has high requirements on users' programming ability. When writing programs, you need to avoid abnormal operations such as illegal pointer access, division by 0, array out of bounds, implicit type conversion, and infinite loops; otherwise, the PLC system may be executed abnormally or even crash. This chapter mainly provides troubleshooting methods for exceptions that may occur in the PLC. You can refer to them according to actual conditions.

After compiling a project, the Invtmatic Studio software displays the compilation output error information by default. In the compilation output window, you can view the compilation errors. Double-click the error display line to locate the error code, as shown in Figure 6-6.

Figure 6-6 Locating an Error Code



Algorithm library error message: Switch to **Library Manager** by the same method to view the algorithm library error message, as shown in Figure 6-7. Open **Library Manager**, as shown in Figure 6-8. If this library is not used, you can delete it; if it is used, you need to install the algorithm library manually.

Figure 6-7 Library Manager-related Prompt Messages

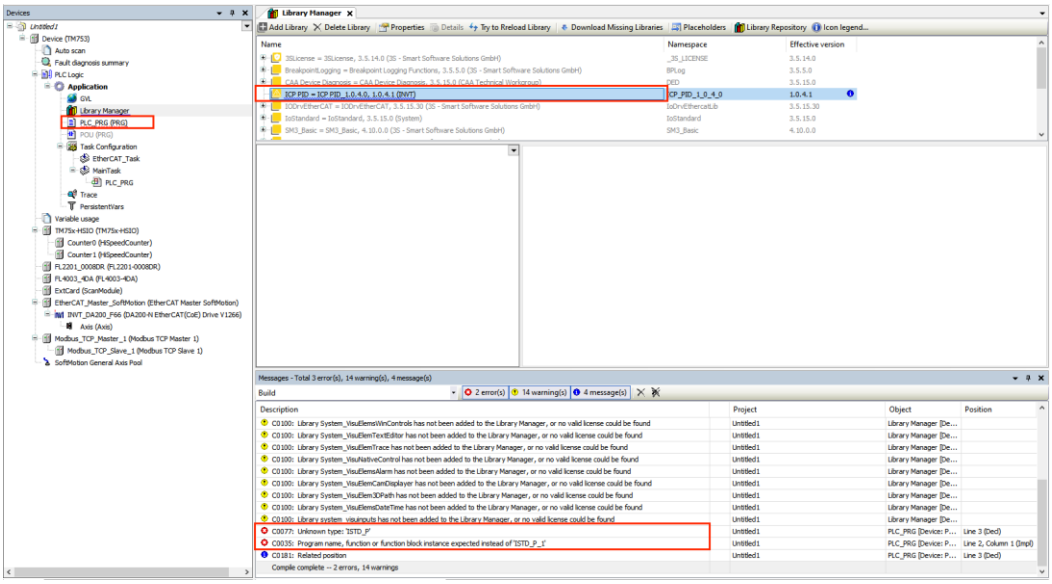
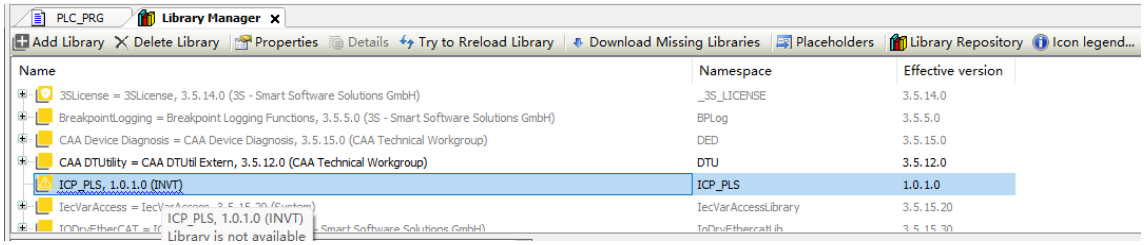


Figure 6-8 Viewing a Missing Library, and Adding or Removing a Library



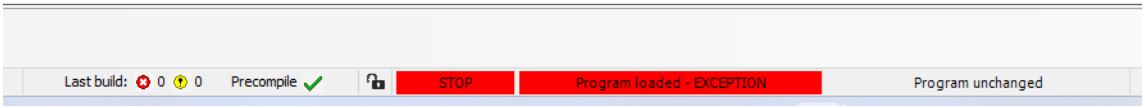
6.6 Handling of PLC Program Running Exceptions

6.6.1 Common Exceptions and Solutions

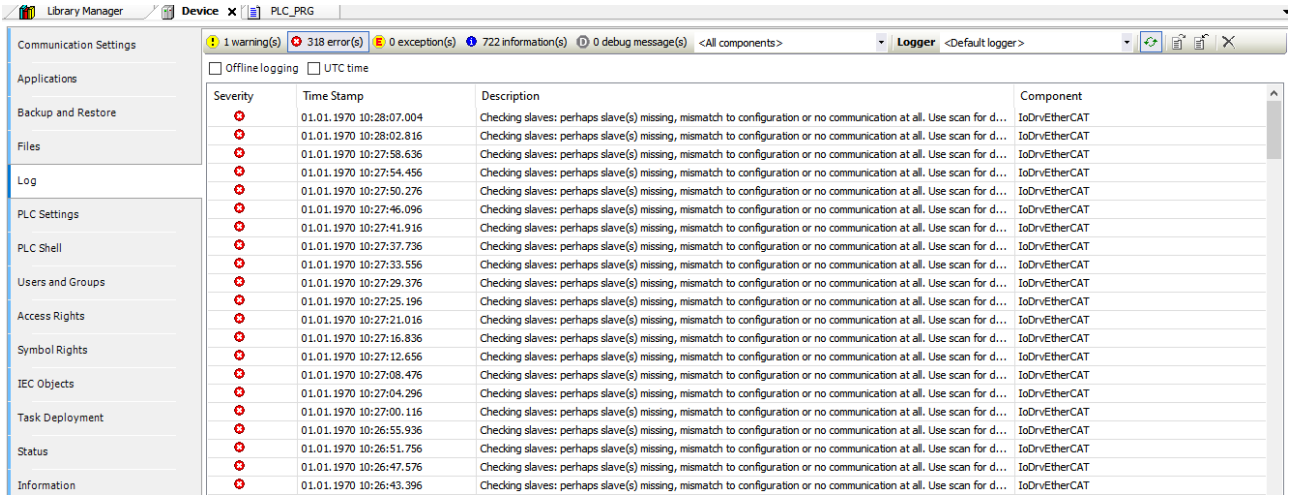
You may encounter the following exceptions during the process of writing and debugging software:

- After downloading the program, the programming software cannot scan the corresponding PLC device. In this case, you need to set the PLC toggle switch to stop and then power it on again.
- When you download a program or after running for a period of time, the PLC information display bar prompts the error “Program download exception”, as shown in Figure 6-9.

Figure 6-9 Program Download Exception

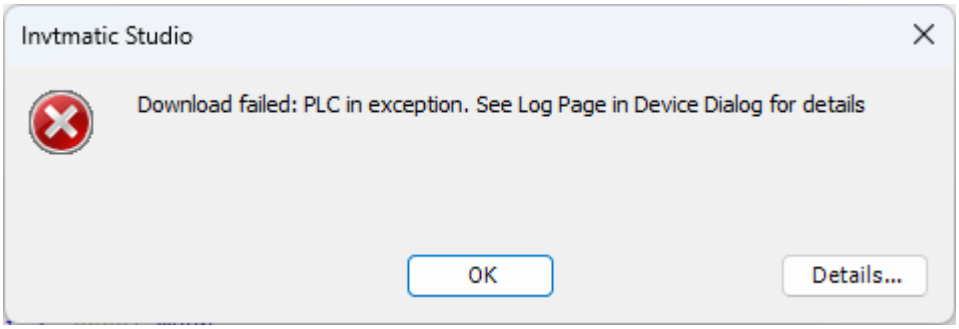


At this time, you can see that there is a type error on the log interface, and the program cannot run normally, as shown in Figure 6-10.



When you download the program, a dialog box prompting “Download Failed” pops up, as shown in Figure 6-10.

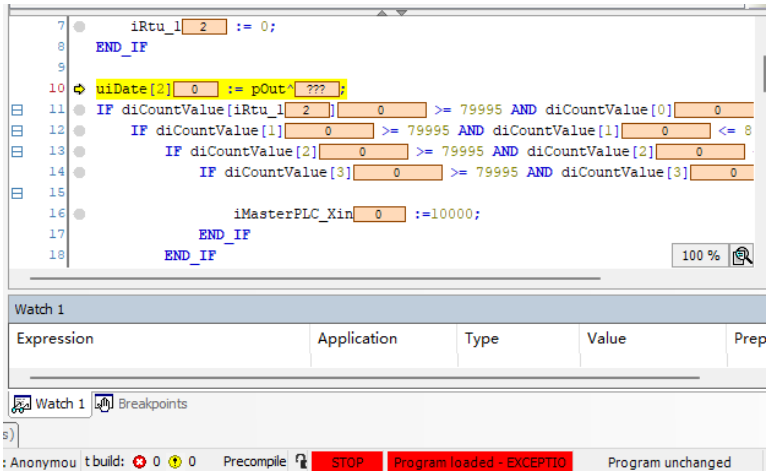
Figure 6-10 Program Download Failed Prompt Dialog Box



Three causes lead to the above exception:

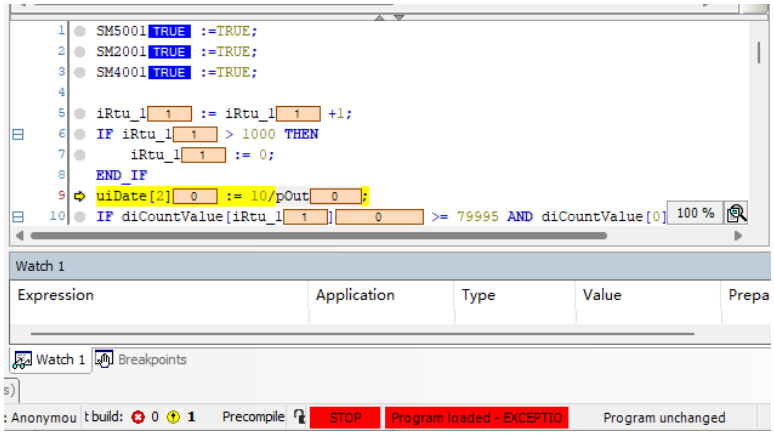
1. Null pointer (i.e. the pointer value is equal to 0).

Figure 6-11 Prompt of a Program Download Exception Due to Invalid Pointer Reference



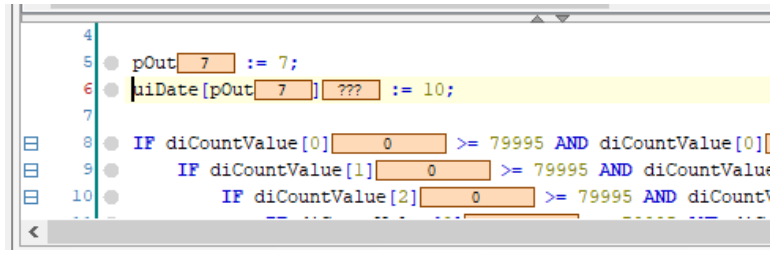
2. Division by 0.

Figure 6-12 Prompt of a Program Download Exception Due to Division by 0



3. Array out of bounds.

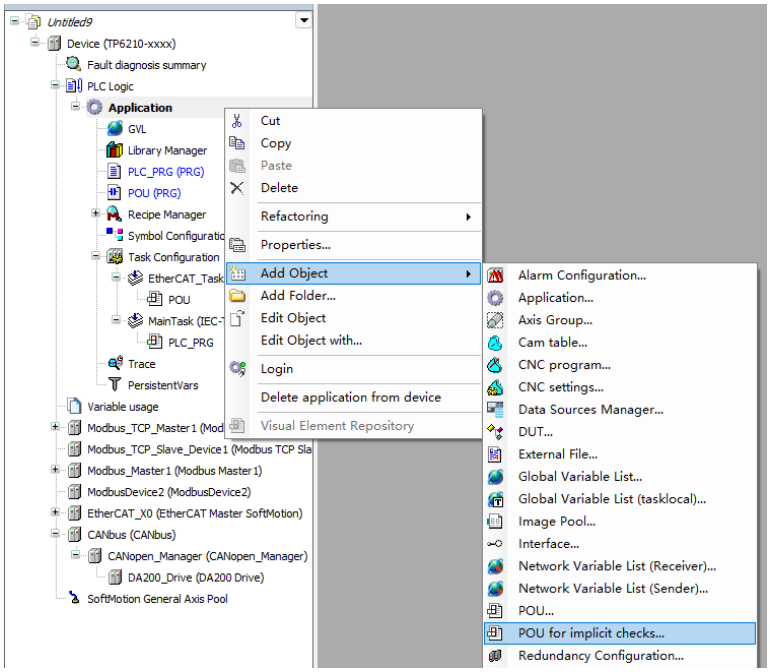
Figure 6-13 Array Out of Bounds



For a null pointer and division by 0, the monitor program will directly prompt them when running, but for the array out-of-bounds, the program can still run normally. The troubleshooting steps are as follows:

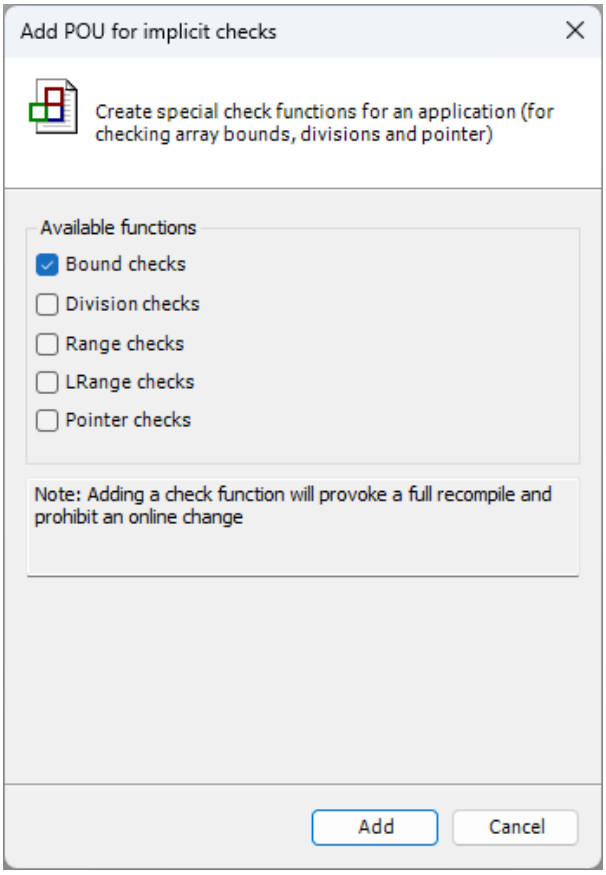
Step 1 Right-click **Application**, and select **Add Object** > **POU for implicit checks**, as shown in Figure 6-14.

Figure 6-14 Adding a POU for Implicit Checks



Step 2 On the pop-up interface, check **Bound checks** and click **Add**, as shown in Figure 6-15.

Figure 6-15 Checking "Bind Check"



Step 3 Log in to the PLC, stop the program, add breakpoints in the code “CheckBounds := lower;” and “CheckBounds := upper;” of the newly added binding function “CheckBounds”, and activate them (press the shortcut key F9).

Step 4 Run the program. When the array is detected to be out of bounds, the program enters the breakpoint, detects the out-of-bounds value (7), and defines the upper limit (3) and lower limit (1) of the array, as shown in Figure 6-16.

Figure 6-16 Program Jumping to a Breakpoint due to Array Out of Bounds

```
1  // Implicitly generated code: Only an implementation suggestion
2  IF  index < lower THEN
3      CheckBounds := lower;
4  ELSIF  index > upper THEN
5      CheckBounds := upper;
6  ELSE
7      CheckBounds := index;
8  END_IF
9
10 (*It is also possible to set a breakpoint, log messages or e.g. to halt on an exception:
11 Add CmpApp.library, SysExcept.library and SysTypes2_Itf as newest.
12 Declaration:
13 VAR
14     _pApp : POINTER TO CmpApp.APPLICATION;
15     _result   : SysTypes.RTS_IEC_RESULT;
16 END_VAR
17
18 Implementation:
19 _pApp := AppGetCurrent(pResult:=_result);
20 IF  index < lower THEN
21         CheckBounds := lower;
22         IF _pApp <> 0 THEN
```

Step 5 After locating the problematic program segment, you need to modify the program according to the actual situation to avoid exceptions.

6.6.2 Solutions for PLC Out of Control Due to Program Problems

Common PLC application problems include:

- An infinite loop or too many loops occur in the application.
- The application accesses a null pointer or a pointer out of bounds.
- The application calls an underlying function block, causing a runtime crash.

Due to improper program writing, when the program is downloaded to the PLC and started, the PLC resources may be exhausted and the Invtmatic Studio software may be unable to control the PLC, resulting in failure to scan devices, ping communication errors, connection errors, login errors, or PLC download errors. The method to restore the PLC to normal state is as follows:

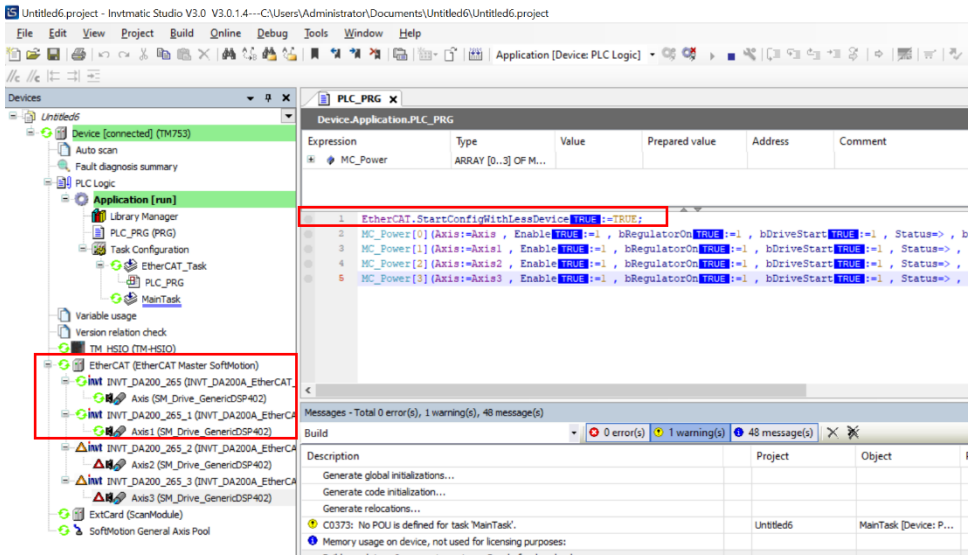
Method 1: Press and hold the **OK** button for 3 seconds to switch the Run/Stop state. Alternatively, use the physical Run/Stop input terminal on the PLC unit to trigger the STOP state. After the PLC is powered off and restarted, scan for the PLC and download a correct application.

Method 2: If the PLC network port can still be pinged, go to the Invtmatic Studio software, locate **Tool**s > **Invtmatic StudioTools**, and click **Reset** on the factory settings screen to clear the APP.

Method 3: If the PLC network port can still be pinged, use the buttons on the PLC LED screen and select **Settings** > **Clear APP** from the menu bar to clear the application.

6.7 Mismatch between EtherCAT Upper Computer Configuration and Physical Connection

As shown in the figure below, the EtherCAT upper computer is configured with 4 axes, but only 2 axes are physically connected. If you attempt to compile and log in directly, the 4 axes will not function. To resolve this, add the following code: **EtherCAT_Master_SoftMotion.StartConfigWithLessDevice:=TRUE;**. The status of the 2 physically connected axes will be restored to normal.



6.8 Device Error Codes

Note: The backplane bus only applies to the TM series PLC.

Device	Error Type	Error Location	Major Error Code	Sub-error Code	Error Description
CPU	System-related	Hardware error	0001	0001	Button cell not installed or battery voltage too low
				0002	Device supply voltage too low (less than 19V)
	System component-related	Clock system component error	0008	0001	Error in setting time
				0002	Error in writing RTC clock
				0003	Error in reading RTC clock
		IP system component error	0009	0001	IP segments of IP1 and IP2 repeated
				XXX	Reserved
				0011	Read: IP1 module - Error in opening files
				0012	Read: IP1 module - Unable to get IP information
				0013	Write: IP1 module - IP address configuration error
				0014	Write: IP1 module - Mask configuration error
				0015	Write: IP1 module - Gateway configuration error
				0016	Write: IP1 module - Repeated segments with USB
				0017	Write: IP1 module - IP and gateway in different segments
				XXX	Reserved
				0021	Read: IP2 module - Error in opening files
				0022	Read: IP2 module - Unable to get IP information
				0023	Write: IP2 module - IP address configuration error
				0024	Write: IP2 module - Mask configuration error
				0025	Write: IP2 module - Gateway configuration error
				0026	Write: IP2 module - Repeated segments with USB
				0027	Write: IP2 module - IP and gateway in different segments
Fieldbus	Modbus-related	Modbus_RTU Master1	0040	0001	Illegal function code
				0002	Illegal address
				0003	Wrong number of data
				0004	Slave device failure

Device	Error Type	Error Location	Major Error Code	Sub-error Code	Error Description
				0005	Communication timeout. An error occurs since the communication time exceeds the maximum communication time set by the user
				XXX	Reserved
				0008	Received data frame non-conforming to the Modbus protocol
				0009	CRC/LRC check error
				XXX	Reserved
				000B	The length of received data does not conform to the protocol or the number exceeds the maximum limit specified by the function code
				000C	The received slave address does not match the requested slave address
				000D	The received function code does not match the requested function code
				000E	Instruction execution failed
		Modbus_RTU Master2	0041	0001	Illegal function code
				0002	Illegal address
				0003	Wrong number of data
				0004	Slave device failure
				0005	Communication timeout. An error occurs since the communication time exceeds the maximum communication time set by the user
				XXX	Reserved
				0008	Received data frame non-conforming to the Modbus protocol
				0009	CRC/LRC check error
				XXX	Reserved
				000B	The length of received data does not conform to the protocol or the number exceeds the maximum limit specified by the function code
				000C	The received slave address does not match the requested slave address
				000D	The received function code does not match the requested function code
				000E	Instruction execution failed

Device	Error Type	Error Location	Major Error Code	Sub-error Code	Error Description
		Modbus_RTU Slave1	0042	0001	Illegal function code
				0002	Illegal address
				0003	Wrong number of data
				0004	Slave device failure
				0005	Communication timeout. An error occurs since the communication time exceeds the maximum communication time set by the user
				XXX	Reserved
				0008	Received data frame non-conforming to the Modbus protocol
				0009	CRC/LRC check error
				XXX	Reserved
				000B	The length of received data does not conform to the protocol or the number exceeds the maximum limit specified by the function code
				000C	The received slave address does not match the requested slave address
				000D	The received function code does not match the requested function code
				000E	Instruction execution failed
		Modbus_RTU Slave2	0043	0001	Illegal function code
				0002	Illegal address
				0003	Wrong number of data
				0004	Slave device failure
				0005	Communication timeout. An error occurs since the communication time exceeds the maximum communication time set by the user
				XXX	Reserved
				0008	Received data frame non-conforming to the Modbus protocol
				0009	CRC/LRC check error
				XXX	Reserved
				000B	The length of received data does not conform to the protocol or the number exceeds the maximum limit specified by the function code
				000C	The received slave address does not match the requested slave address

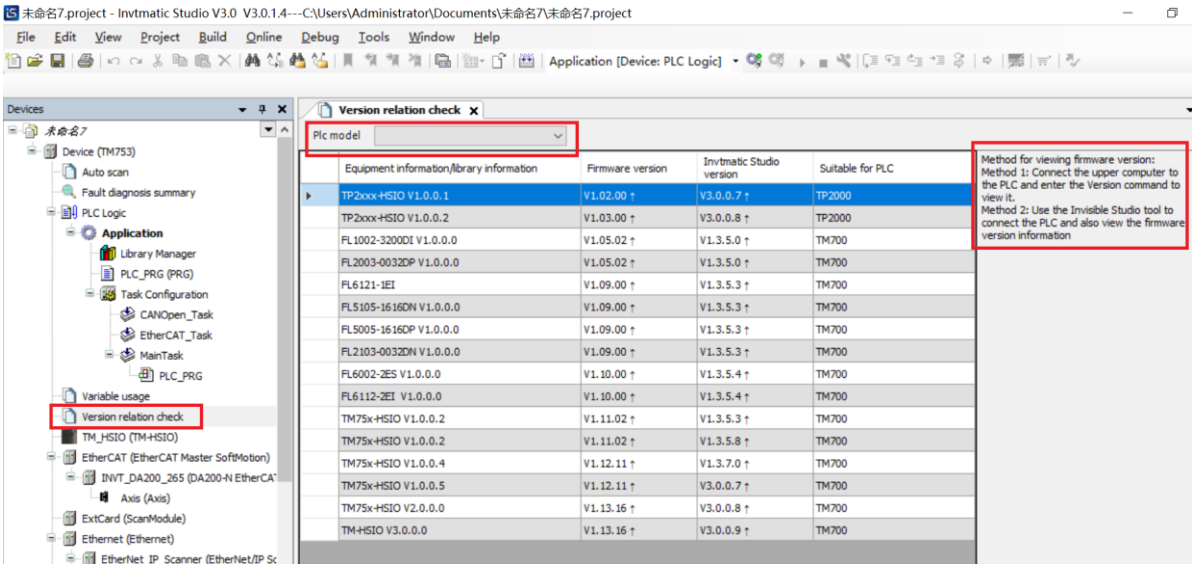
Device	Error Type	Error Location	Major Error Code	Sub-error Code	Error Description
				000D	The received function code does not match the requested function code
				000E	Instruction execution failed
	Modbus TCP-related	Modbus TCP Master1	00A0	0001	Illegal function code
				0002	Illegal address
				0003	Wrong number of data
				0004	Slave device failure
				0005	Communication timeout. An error occurs since the communication time exceeds the maximum communication time set by the user
				XXX	Reserved
				0008	Received data frame non-conforming to the Modbus protocol
				0009	CRC/LRC check error
				XXX	Reserved
				000B	The length of received data does not conform to the protocol or the number exceeds the maximum limit specified by the function code
				000C	The received slave address does not match the requested slave address
				000D	The received function code does not match the requested function code
				000E	Instruction execution failed
		Modbus TCP Master2	00A1	0001	Illegal function code
				0002	Illegal address
				0003	Wrong number of data
				0004	Slave device failure
				0005	Communication timeout. An error occurs since the communication time exceeds the maximum communication time set by the user
				XXX	Reserved
				0008	Received data frame non-conforming to the Modbus protocol
				0009	CRC/LRC check error
				XXX	Reserved

Device	Error Type	Error Location	Major Error Code	Sub-error Code	Error Description
				000B	The length of received data does not conform to the protocol or the number exceeds the maximum limit specified by the function code
				000C	The received slave address does not match the requested slave address
				000D	The received function code does not match the requested function code
				000E	Instruction execution failed
		Modbus TCP Slave1	00A2	0001	Illegal function code
				0002	Illegal address
				0003	Wrong number of data
				0004	Slave device failure
				0005	Communication timeout. An error occurs since the communication time exceeds the maximum communication time set by the user
				XXX	Reserved
				0008	Received data frame non-conforming to the Modbus protocol
				0009	CRC/LRC check error
				XXX	Reserved
				000B	The length of received data does not conform to the protocol or the number exceeds the maximum limit specified by the function code
				000C	The received slave address does not match the requested slave address
				000D	The received function code does not match the requested function code
				000E	Instruction execution failed
		Modbus TCP Slave2	00A3	0001	Illegal function code
				0002	Illegal address
				0003	Wrong number of data
				0004	Slave device failure
				0005	Communication timeout. An error occurs since the communication time exceeds the maximum communication time set by the user
				XXX	Reserved

Device	Error Type	Error Location	Major Error Code	Sub-error Code	Error Description
				0008	Received data frame non-conforming to the Modbus protocol
				0009	CRC/LRC check error
				XXX	Reserved
				000B	The length of received data does not conform to the protocol or the number exceeds the maximum limit specified by the function code
				000C	The received slave address does not match the requested slave address
				000D	The received function code does not match the requested function code
				000E	Instruction execution failed

6.9 Version Compatibility between the Upper Computer and Lower Computer

As shown in the following figure, take the TP2xxx-HSIO V1.0.0.1 library as an example. The TP2xxx-HSIO V1.0.0.1 library requires firmware version V1.02.00 or later and upper computer software version V3.0.1.4 or later.

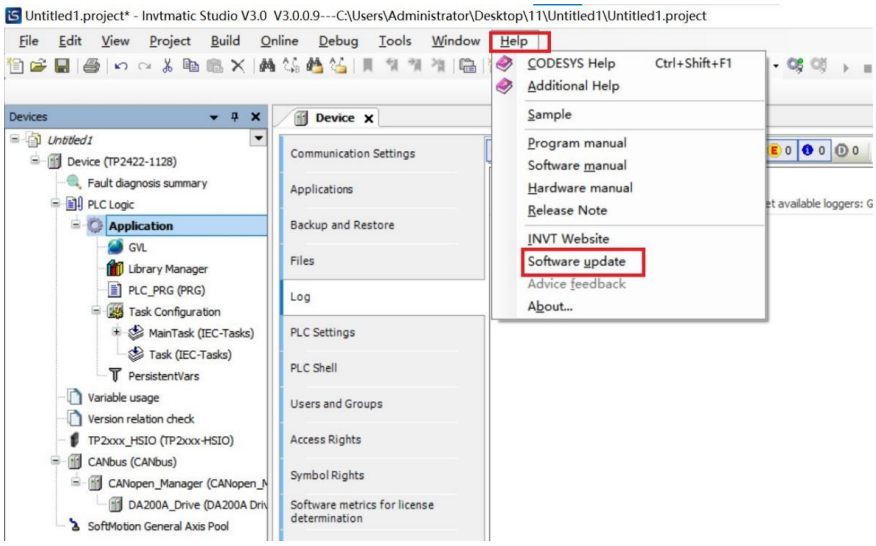


7 PLC Upgrades and Settings

7.1 Software Upgrade

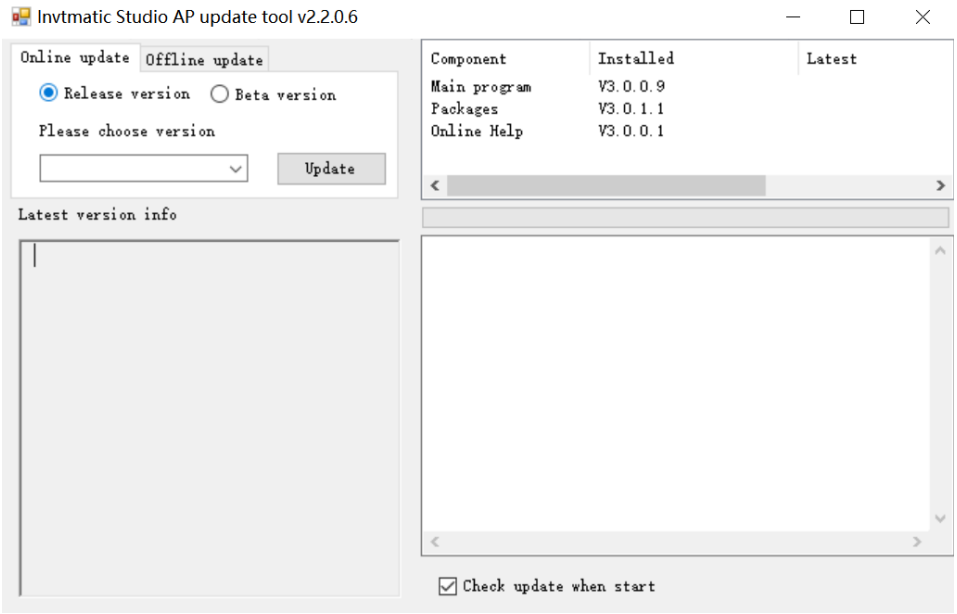
After opening the Invtmatic Studio software, click **Help** > **Software upgrade** in the menu to enable online software upgrades.

Figure 7-1 Software Upgrade Interface



If there is a new version, the software will automatically prompt you to upgrade it when you open it. After the download is complete, click **Upgrade**.

Figure 7-2 Downloading and Installing the Upgrade Package



7.2 Device Connection

Step 1 Navigate to **Tools** > **Invtmatic Studio Tool** in the menu bar to open the Invtmatic Studio Tool (hereinafter referred to as the tool).

Step 2 Once the tool is opened, the interface appears as shown in Figure 7-3. The tool provides two connection methods:

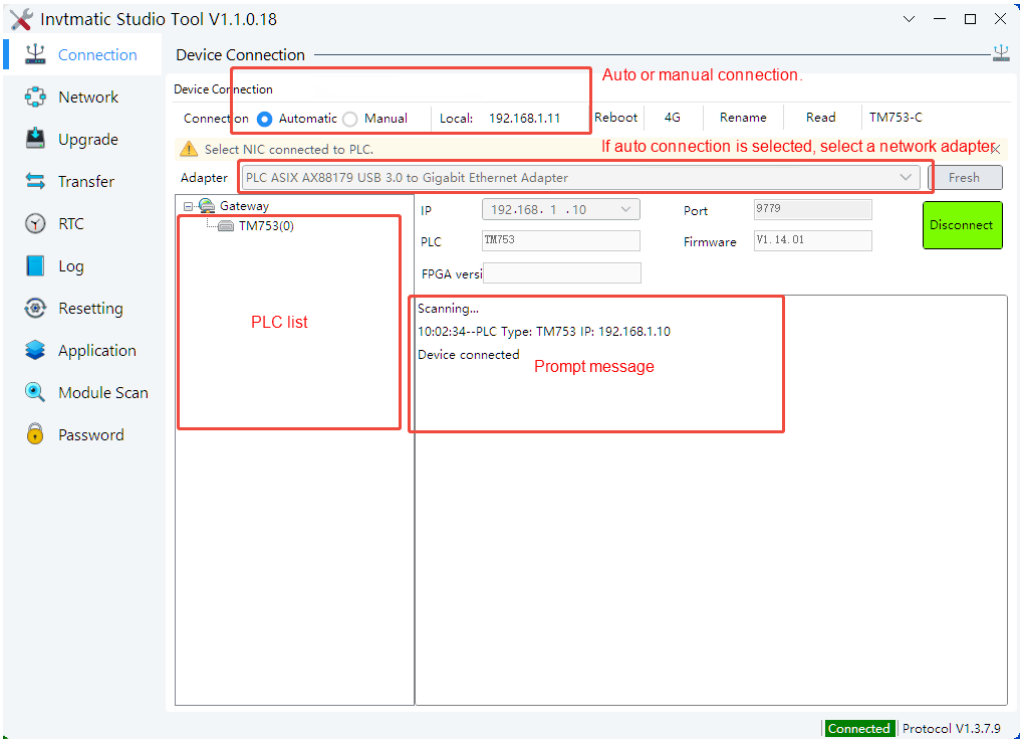
Automatic Connection: Select the corresponding network adapter and click **Connect**. All scanned PLCs will be listed in the left pane. Double-click the target PLC to establish a connection.

Manual Connection: If you know the specific IP address of the PLC, enter the IP address and port, then click **Connect**.

Step 3 During an automatic connection, if a scanned PLC cannot connect due to a subnet mismatch, a pop-up window will ask if you want to add an IP address in the corresponding network segment. You do not need to add the IP address manually, and the IP address is configurable.

If a connection fails, the tool defaults to the last successfully connected IP address.

Figure 7-3 Device Connection Interface



7.3 Firmware Upgrade

The TM/TP series supports firmware upgrades via a network port.

Step 1 After connecting to the PLC, open the firmware upgrade interface, as shown in Figure 7-4. The information of the connected PLC will be displayed.

Step 2 Click **Check**, select the firmware package corresponding to the PLC model, and start the download.

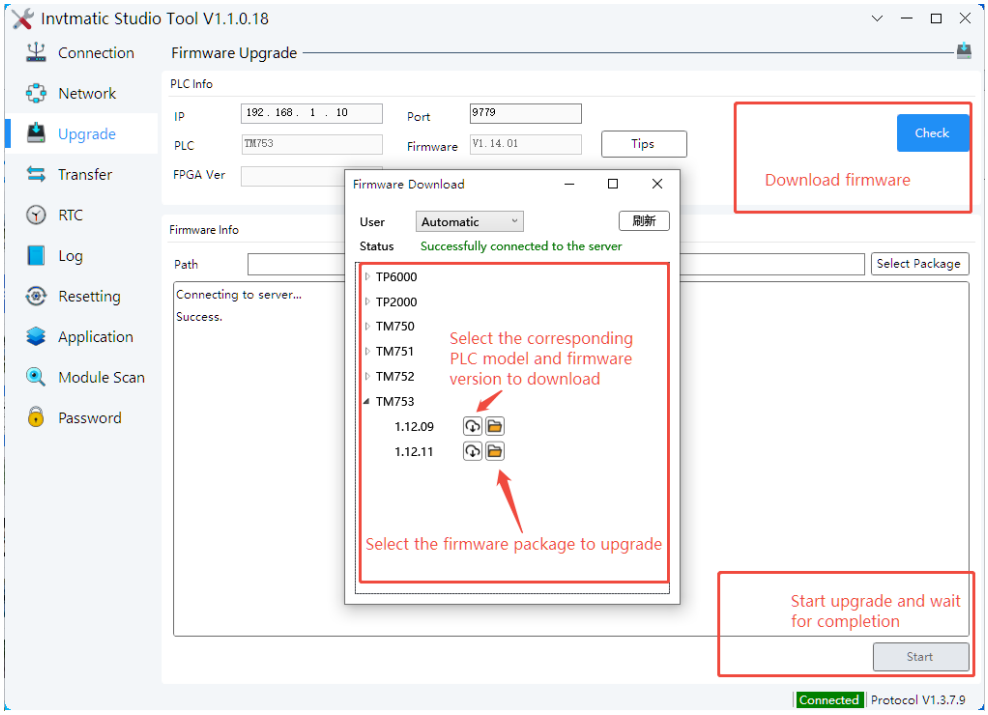
Step 3 Once the download is complete, click the file icon on the right to automatically select the downloaded package, without needing to manually select it.

Step 4 Then, click **Start** and wait for the upgrade to complete.

Step 5 After the upgrade is complete, check that the firmware version has changed.

Note: TP series PLCs also support firmware upgrades via files stored on a USB flash drive, which is convenient and easy to use.

Figure 7-4 PLC Connection Interface of the Tool



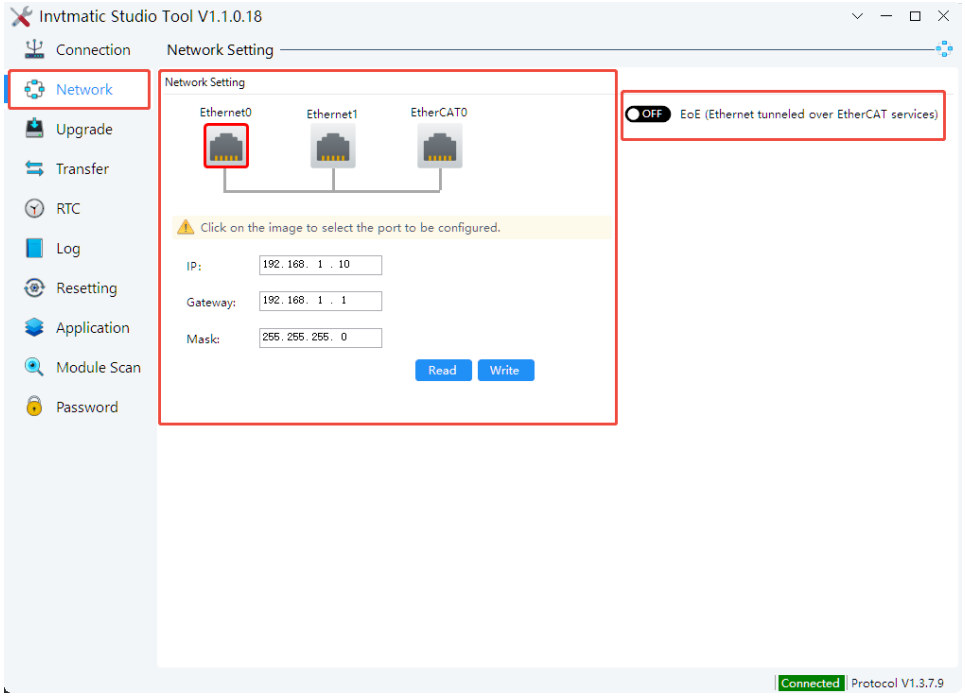
7.4 Network Settings

Through network settings, you can read or modify the IP address and subnet mask of network ports 1 and 2, as shown in Figure 7-5. You can click the icons to switch between Ethernet ports.

Note:

- Networking function switch: Allows ETH0 and ETH1 to be set to the same subnet, enabling data exchange and networking functionality.
- Ethernet over EtherCAT (EoE) function switch: EoE uses mailbox communication to encapsulate Ethernet messages in EtherCAT mailbox messages, simplifying wiring. With EoE communication, servo parameters can be quickly adjusted with a single click, greatly improving debugging and setup efficiency.
- In switch mode, do not connect two or more physical ports (such as ETH0 and ETH1) of this device to different ports on the same external switch using network cables. This can trigger a broadcast storm, causing the entire network to fail in seconds and rendering all PLCs unable to communicate.

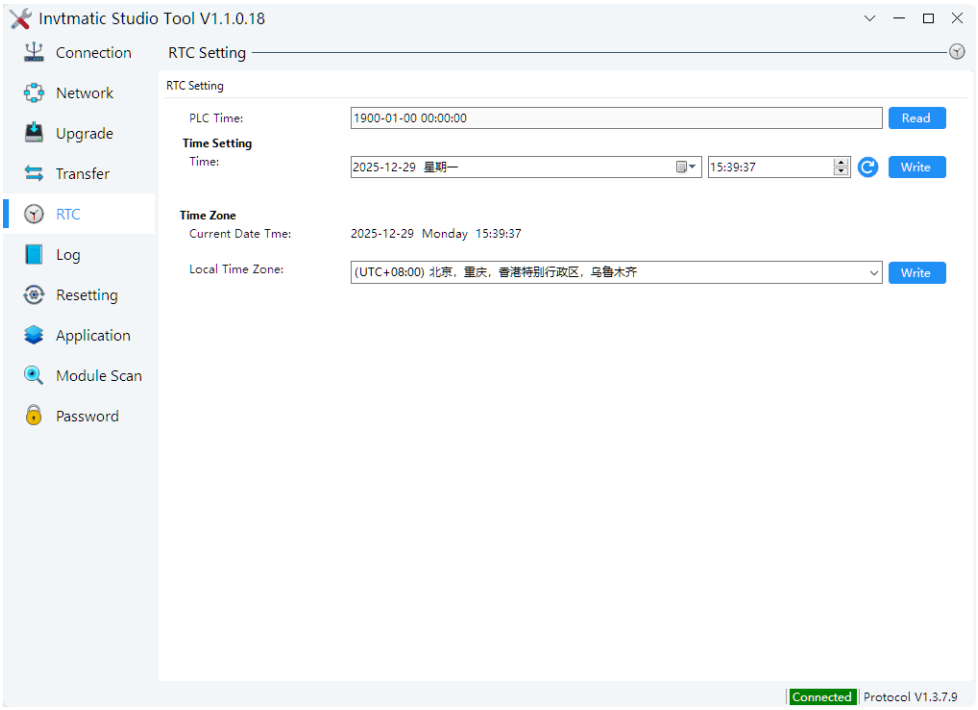
Figure 7-5 Network Setting Interface



7.5 Time Settings

You can read or modify the system time.

Figure 7-6 System Time Read/Write Interface

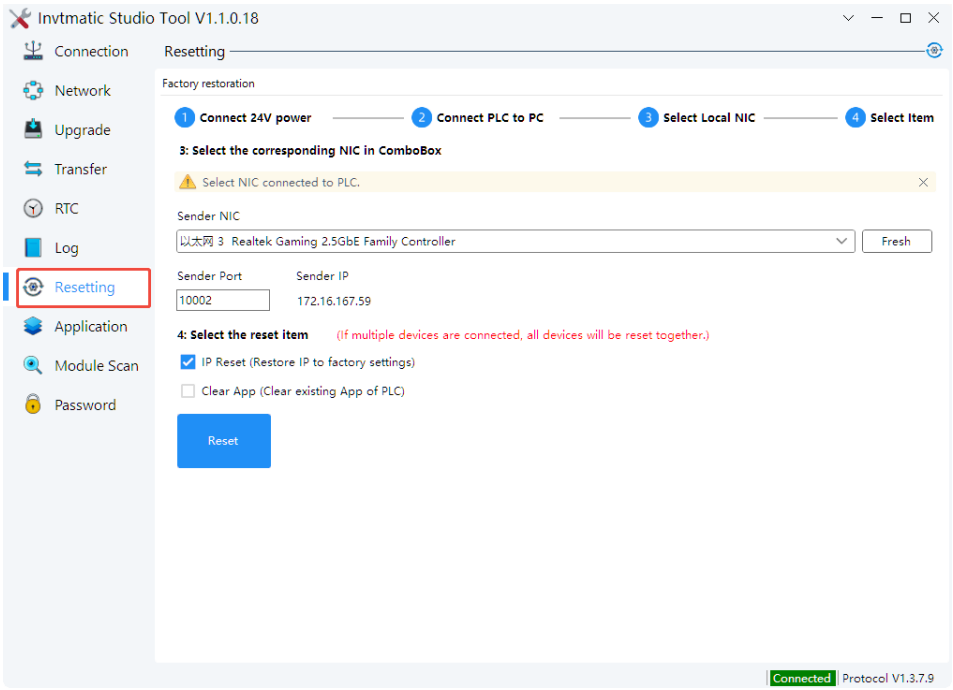


7.6 Factory Settings Reset

You can restore the PLC to its factory default settings on the **Resetting** interface, as shown in Figure 7-7. Since the PC is connected to Ethernet1 or Ethernet2 of the PLC, it is not necessary to know the IP address of the PLC. The PLC can be reset to factory settings and the APP can be cleared by one click.

Note: Restoring factory settings and clearing the APP will clear all PLCs on the same Local Area Network (LAN). Disconnect the network cables of other PLCs before executing this function.

Figure 7-7 Restoring the IP Address or Clearing the App



7.7 File Transfer

The file transfer interface is shown in Figure 7-8. The PC directory is displayed on the left, and the PLC directory is on the right. You can transfer files between them using the buttons in the middle. The TP2000 series PLC does not support the quick file transfer function. For TM series and TP6000 series PLCs, this interface allows single transfers of files or folders up to 30MB. If you need to transfer larger files or folders (exceeding 30MB), use the quick file transfer function located in the bottom right corner. Click **Quick File Transfer**. In the pop-up window, enter the PLC's IP address and click **OK** to proceed. The quick file transfer interface is shown in Figure 7-9.

Additionally, both the PC and PLC directories have operation buttons in their upper right corners. Hover your mouse over a button to view its function, or right-click.

Figure 7-8 File Transfer

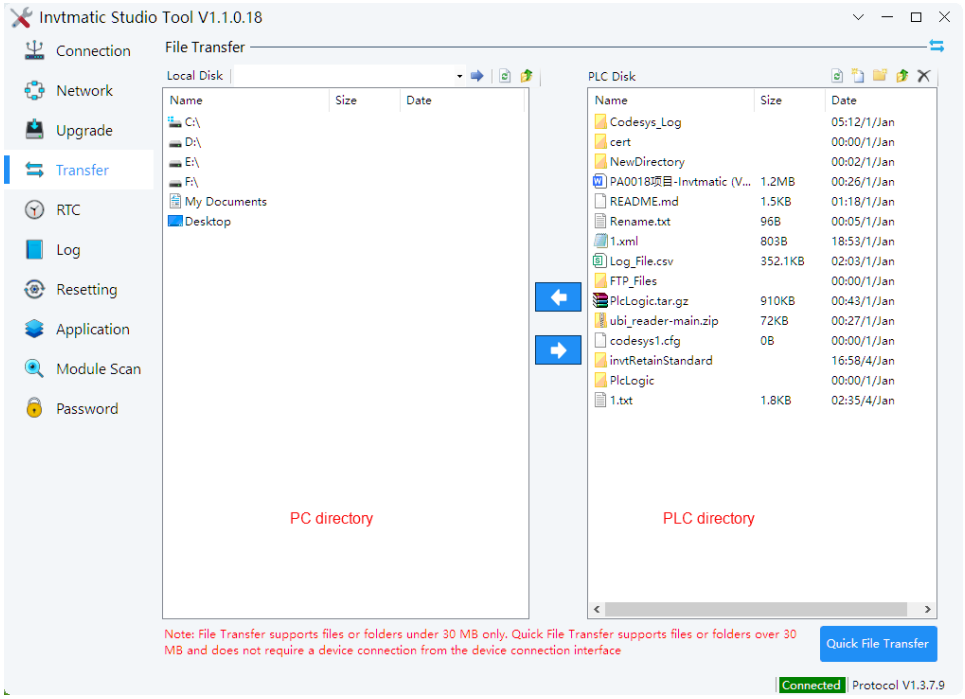
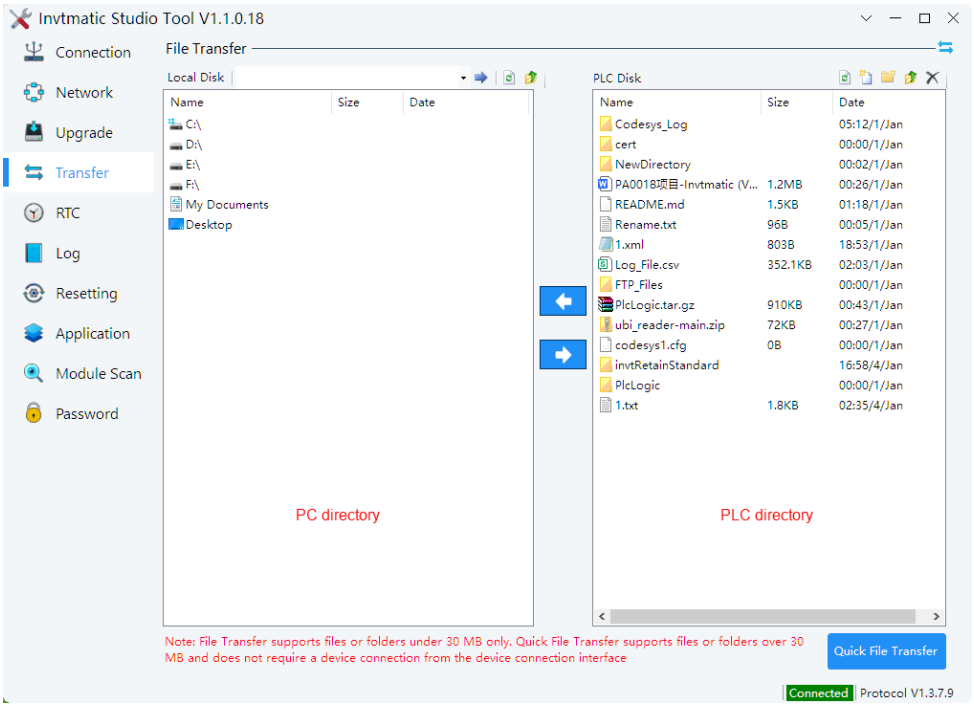


Figure 7-9 Quick File Transfer

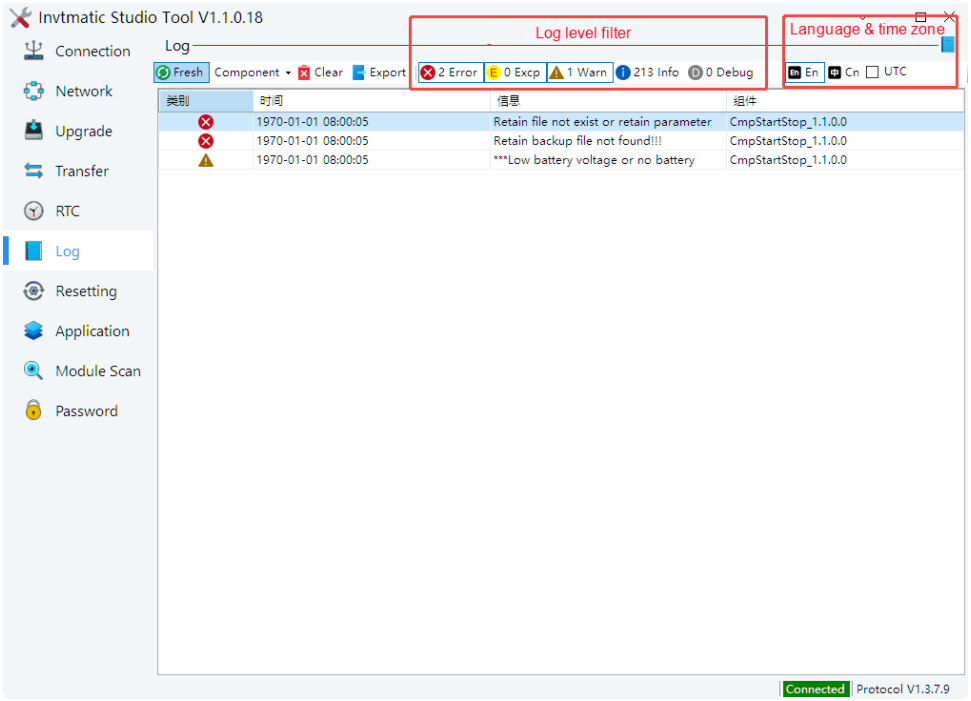


7.8 Log

The log interface is shown in Figure 7-10. Click **Fresh** to retrieve the logs. You can filter the logs by component or log level. The interface also supports switching between Chinese and English and adjusting the time zone. Logs can be exported as .CSV files.

Translation only applies to the information column of error logs.

Figure 7-10 Log Interface



7.9 Application

The application interface is shown in Figure 7-11. This function must be used in conjunction with **Build** > **Generate runtime system files** in the upper computer's toolbar. After using the **Generate runtime system files** function as described in section 3.2 Build Menu, a file with the .userprg extension will be generated. You can select this file on this interface by clicking **Select Application**. Once selected, click **Program Upgrade** to write the packaged program to the PLC.

Application Settings

Refresh to Get: Retrieves all programs currently stored in the PLC.

Save Locally: Downloads the retrieved programs to the PC.

Upload to PLC: Uploads programs from the PC to the PLC.

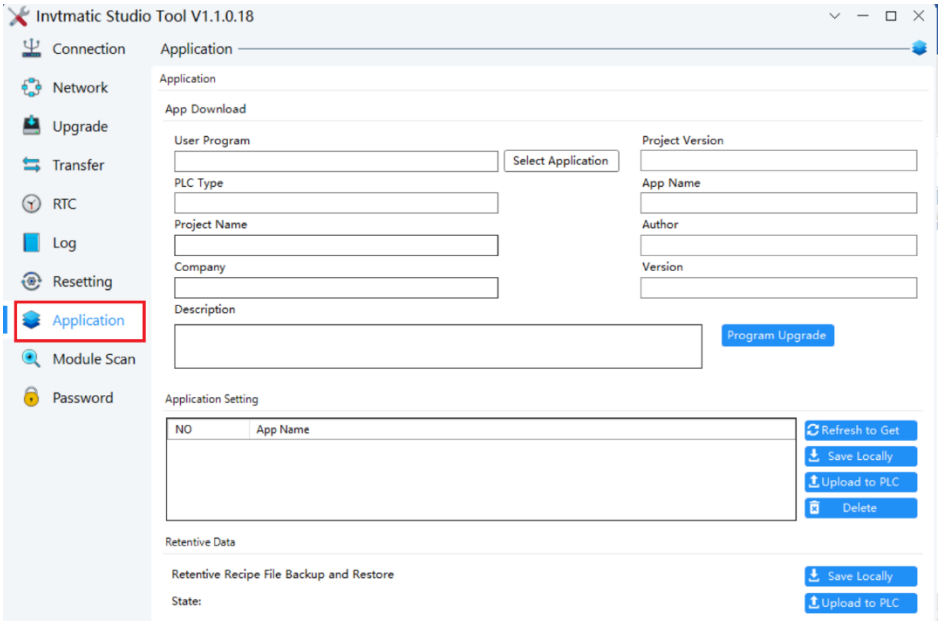
Delete: Deletes the retrieved programs.

Power-failure Retention Data

Save Locally: Downloads the power-failure retention recipes to the PC.

Upload to PLC: Uploads the power-failure retention recipes to the PLC.

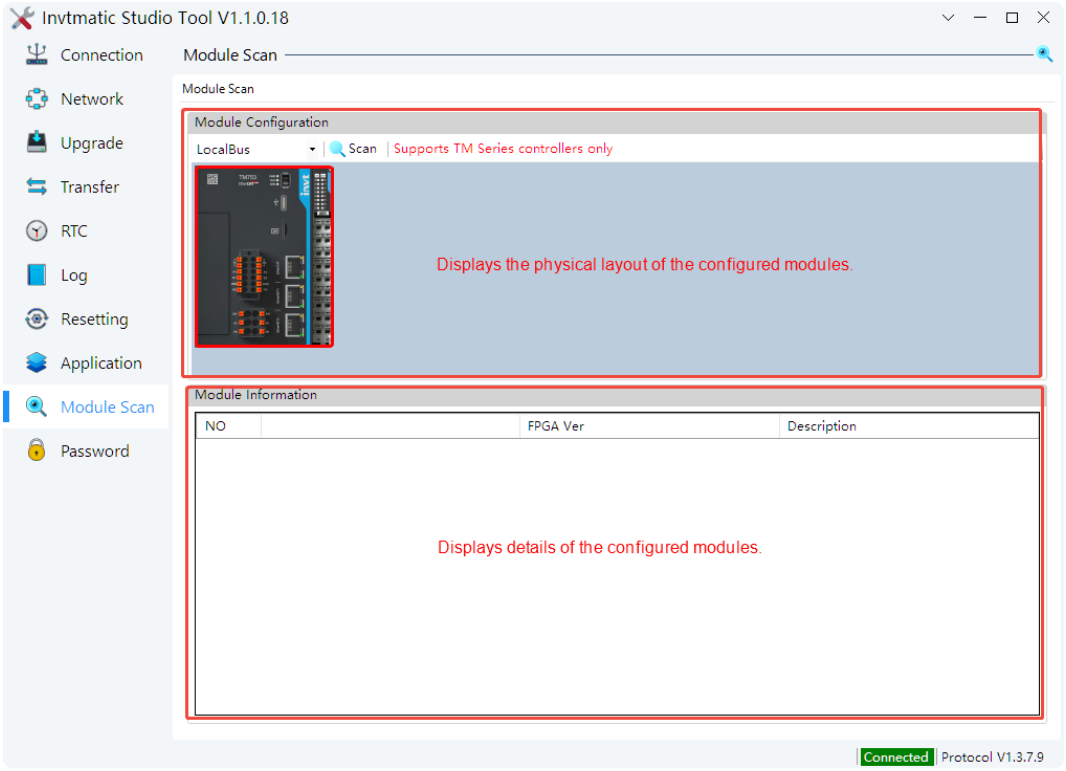
Figure 7-11 Application



7.10 Module Scanning

The module scanning interface is shown in Figure 7-12. This function is only applicable to TM series PLCs. After connecting to the PLC, click **Scan**. The **Module Information** area displays details of the configured modules, while the **Module Configuration** area shows their physical layout.

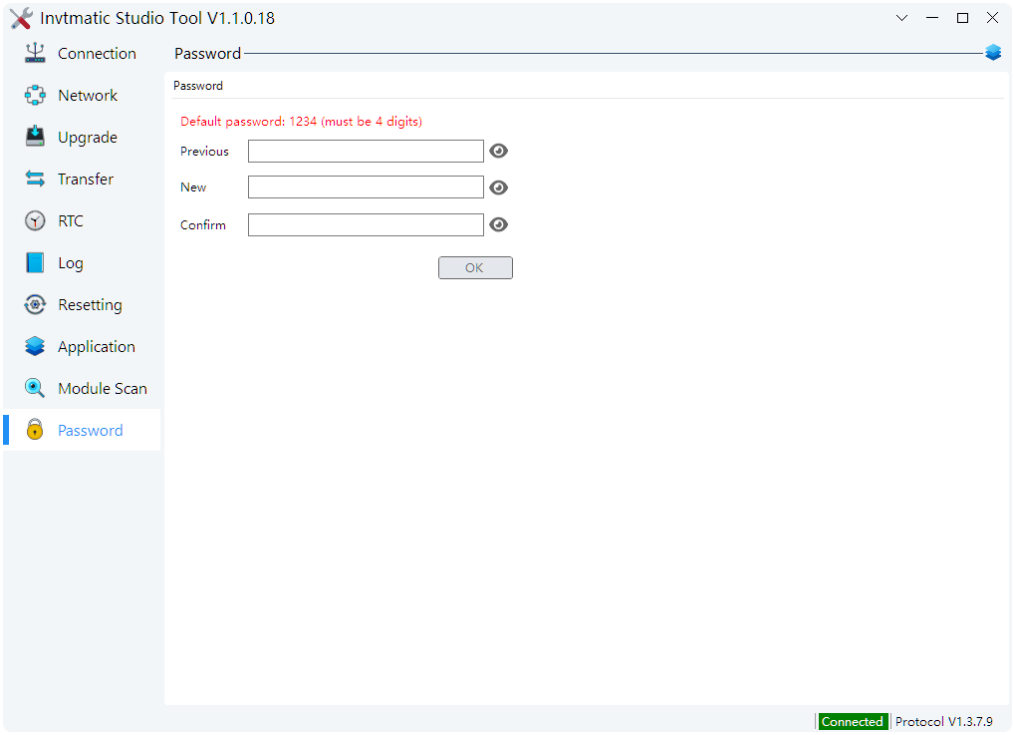
Figure 7-12 Module Scanning



7.11 Password Settings

As shown in Figure 7-13, you can reset the PLC password.

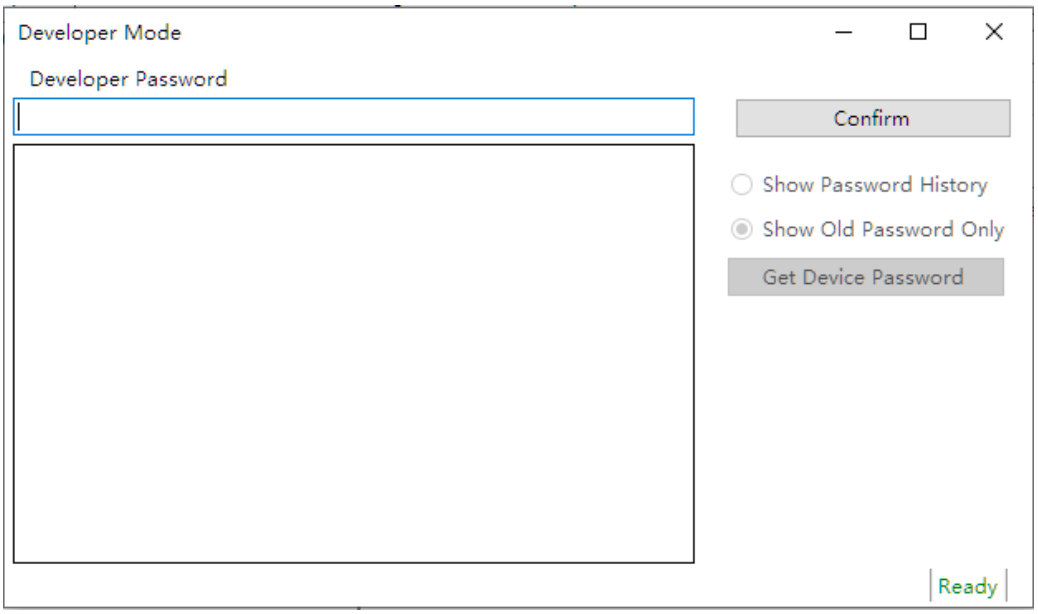
Figure 7-13 PLC Password Reset



7.12 Developer Mode

To access developer mode, click the icon in the upper right corner of the software and select **Developer Mode**. The interface is shown in Figure 7-14. This function is exclusively available for TP series PLCs and can be used to retrieve the PLC password.

Figure 7-14 Developer Mode

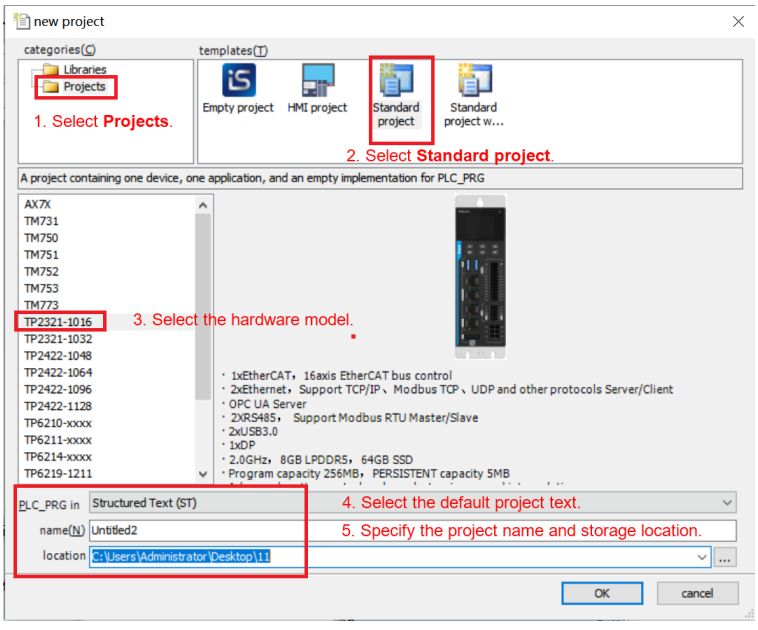


Appendix A Project Examples

A.1 Example of RS485 Communication Configuration between the Controller and Goodrive20 Series VFD

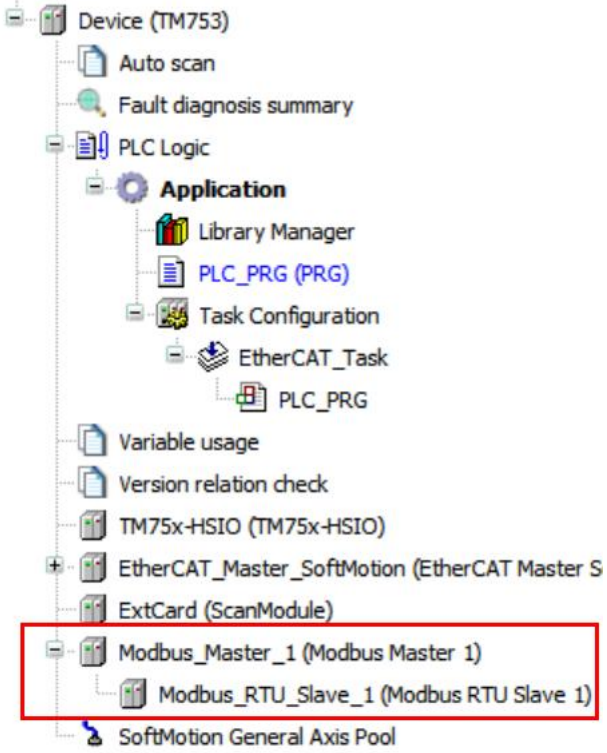
Set the TP2000 series controller as the master and Goodrive20 series VFD as the slave. The controller uses the Modbus RTU communication protocol, with the physical layer of two-wire RS485, and communicates with the VFD through the COM2 port. By writing a small program, use the upper computer to read and write the function parameters of Goodrive20 VFD. The A1 port of the PLC hardware is connected to the VFD 485+, and the B1 port is connected to the VFD 485-. The operation steps are as follows:

Step 1 Create a new project and select the menu Projects > Standard project. At this time, a new standard project has been created, the device is TP2321-1016, the programming language is structured text (ST), and the project information can be edited as needed.

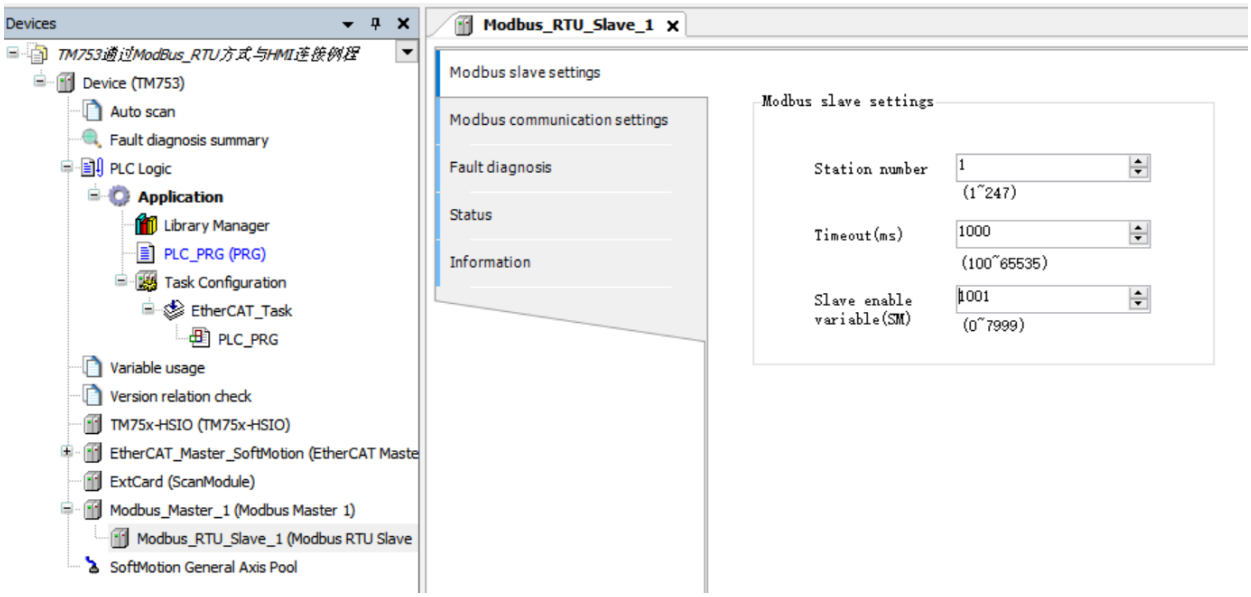


Step 2 Add Modbus_Master1 and Modbus_RTU_Slave1 following section 0

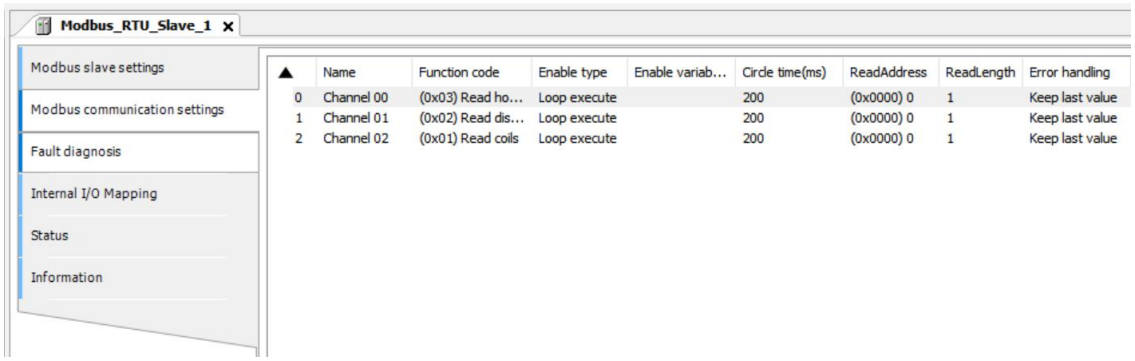
Step 3 Modbus RTU Master Configuration, set the communication parameters, and set the PLC as a Modbus_TCP slave to communicate with the HMI. This allows you to display the VFD parameters on the HMI, as shown in the figure below.



Step 4 Set the baud rate to 19200, the data bit to 8, the stop bit to 1, the check bit to EVEN (even parity), and the timeout period to 1000 ms; set the VFD node number (slave address) to 1, and the slave enable variable to 1001.



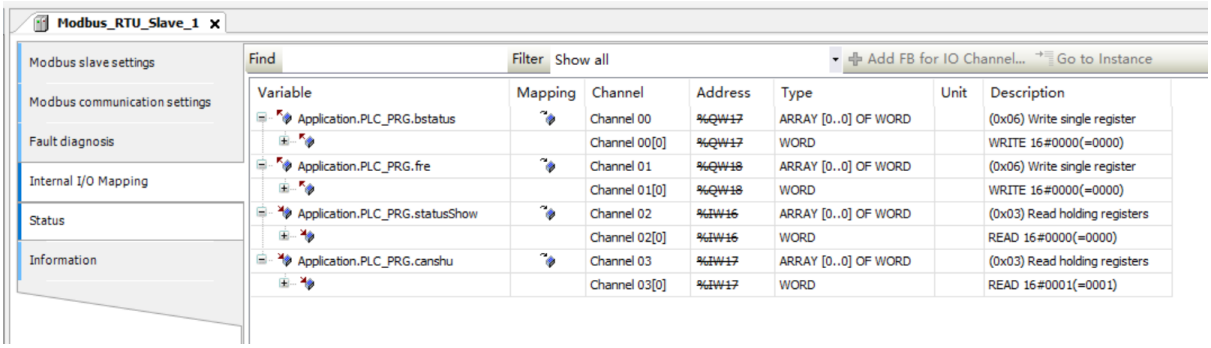
Step 5 In the **Modbus Communication Settings** window, add a Modbus slave communication configuration, as shown in the following figure.



Step 6 Double-click PLC_PRG and enter the following code in the declaration editor to bind the variable address, so as to facilitate communication with the touch screen.

```
PROGRAM PLC_PRG

VAR

brun_pos AT %MX0.0 :BOOL; // Forward

bSet        AT %MX0.1   :BOOL;// Frequency setting start

bStop        AT %MX0.2   :BOOL;// Stop

brun_neg    AT %MX0.3   :BOOL;// Reverse

breset      AT %MX0.4   :BOOL;// Error reset

bJog_Pos    AT %MX0.5   :BOOL;// Forward jog

bJog_Neg    AT %MX0.6   :BOOL;// Reverse jog


statusShow  AT %MW100   :INT;// Running state display

fre                 :INT;// Frequency

bstatus             :INT;// Running state

hfre        AT %MW10    :INT;// Touch screen frequency value setting

canshu     AT %MW200    : ARRAY[1..6] OF INT;

END_VAR
```

In the Modbus slave communication settings window, map the control variables in the program:



In Channel03, read 6 addresses from 16#3000 and map them to the canshu array variable. In the array canshu[2], i.e. the address 3002H, if the value 3335 is read, it means that the bus voltage is 333.5 V. Please refer to the VFD product manual.

In Channel02, read 1 address from 16#2100 and map it to the status Show variable. If the value is 3, it means the VFD is stopped. Please refer to the VFD product manual.

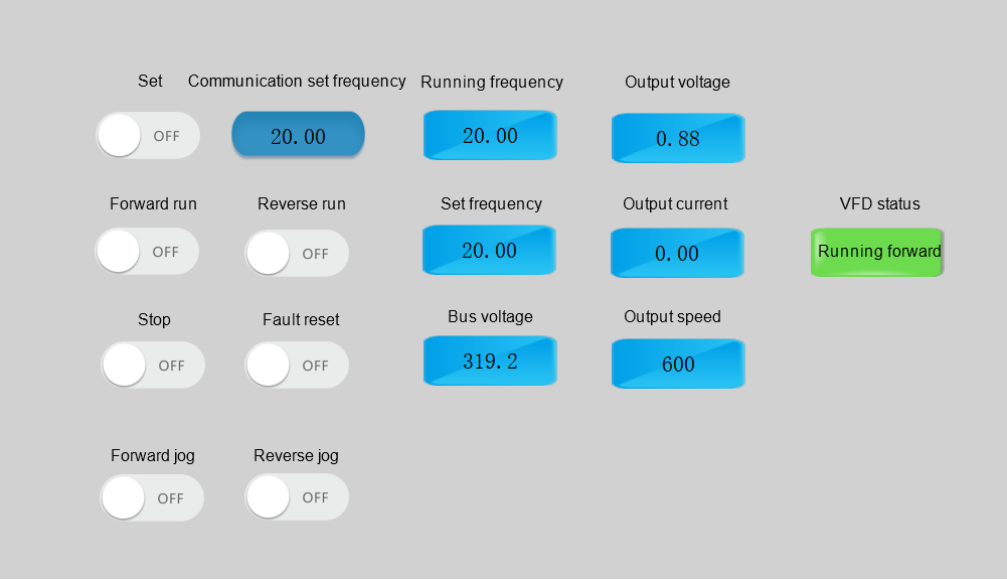
Enter the following code in the main code editor:

```
SM1001:=TRUE;// Slave enable variable
IF brun_pos THEN
  bstatus:=1;
END_IF
IF brun_neg THEN
  bstatus:=2;
END_IF
IF bJog_Pos THEN
  bstatus:=3;
END_IF
IF bJog_Neg THEN
  bstatus:=4;
END_IF
IF bStop THEN
  bstatus:=5;
END_IF
IF bSet THEN
  fre:=hfre;
END_IF
```

After connecting the VFD to the controller via two-wire RS485, start the VFD. Set the function code P00.01 to 2 through the VFD keypad so that its running instruction can be controlled by the upper computer through communication; set P00.06 to 8, that is, select the Modbus communication mode; set the serial communication parameters of the group P14 to make them consistent with the initialization setting parameters of the upper computer, including the baud rate, data bit, check bit, slave address, timeout period, etc.

Click the icon  in the Toolbar to compile the code. After completing compilation without errors, click the button  in the Toolbar to log in to the controller, and ensure that the Nixie tube of the controller has no error, Goodrive20 VFD is successfully connected to the controller, the communication is normal, and the touch screen interface is as shown in Figure A-1.

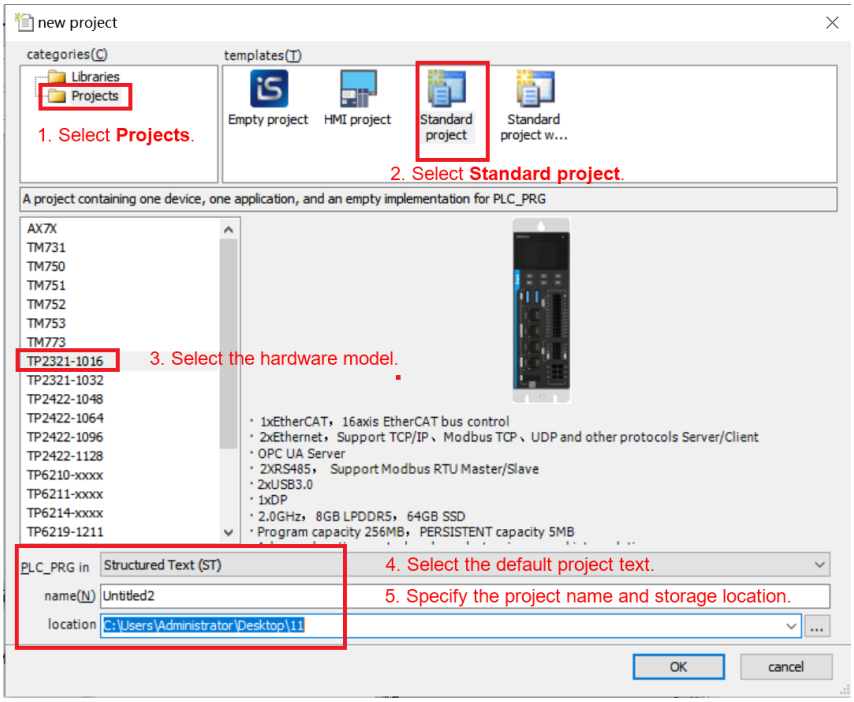
Figure A-1 HMI Screen for Communication between TP2000 and the VFD



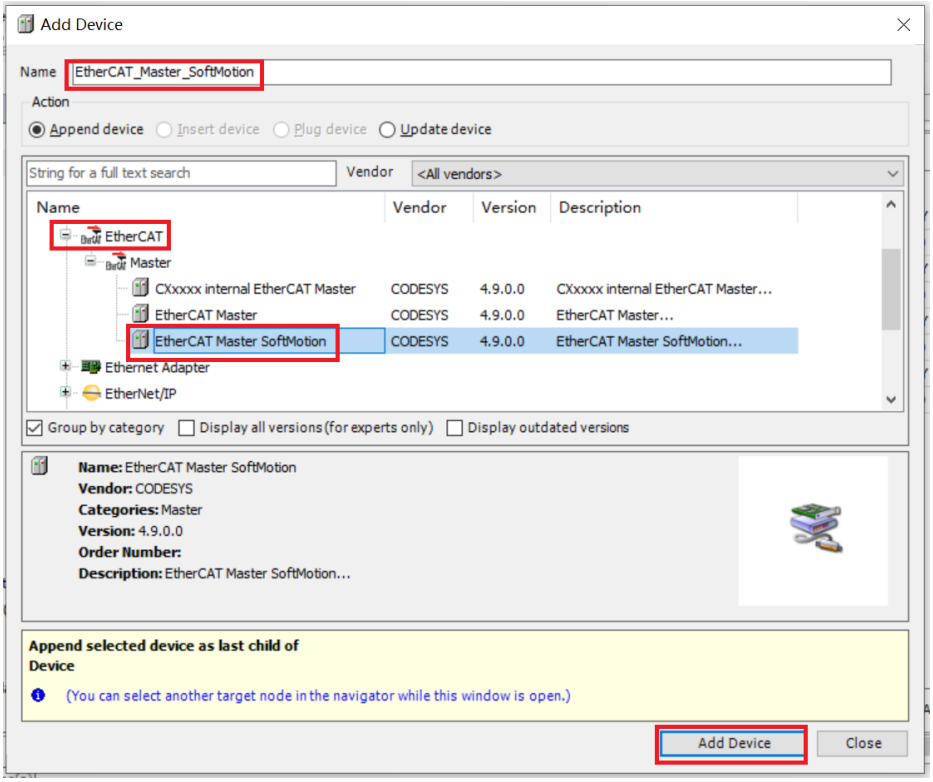
A.2 Example of Communication Configuration between the Controller and DA200 Series Servo Drive

Example: By writing a small program, control 4 DA200 series servo drives to drive 4 motor shafts to perform uniform forward and reverse motion. The operation steps are as follows:

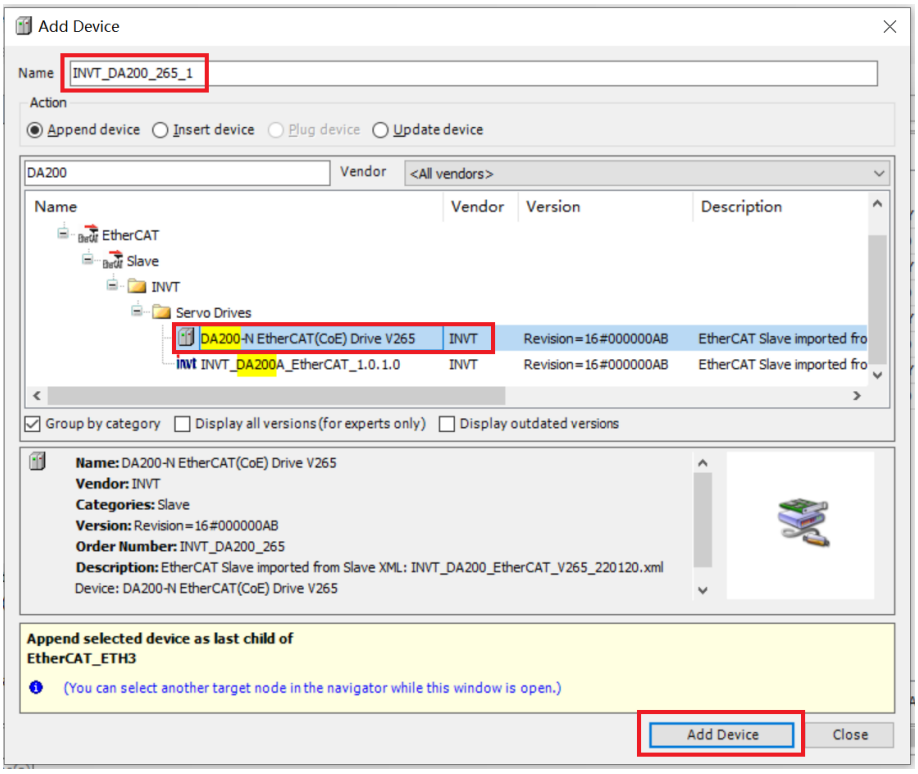
Step 1 Create a new project and select the menu **Projects** > **Standard Project**. At this time, a new standard project has been created, the device is TP2321-1016, the programming language is structured text (), and the project information can be edited as needed, as shown below.



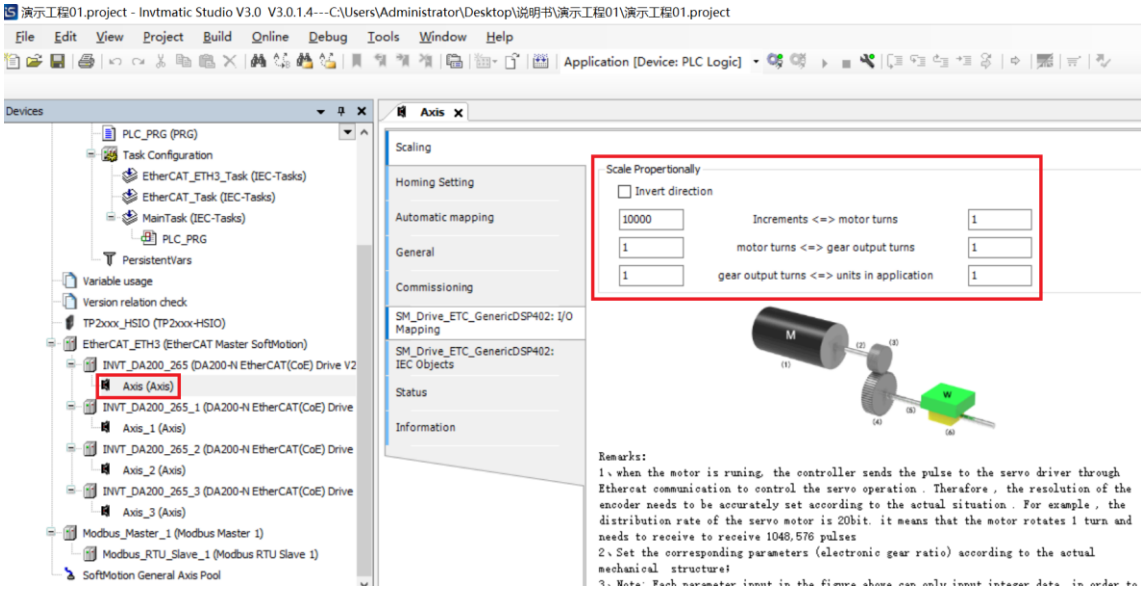
Step 2 Right-click **Device** in the device tree, and select **Add Device** to add an EtherCAT master device, **EtherCAT Master SoftMotion, Version 4.9.0.0** here.



Step 3 Right-click **EtherCAT Master SoftMotion** in the device tree, and select **Add Device** to add 4 servo drives, **DA200-N** here.



Step 4 Right-click INVT_DA200 in the device tree, add 4 servo motors, and configure the axis gear ratio for each, as shown in the figure below.



Step 5 Double-click the PLC_PRG and enter the following codes on the statement editor:

```
PROGRAM PLC_PRG

VAR

iStatus: INT;

MC_Power_0: MC_Power;

MC_Power_1: MC_Power;

MC_Power_2: MC_Power;

MC_Power_3: MC_Power;

MC_MoveAbsolute_0: MC_MoveAbsolute;

MC_MoveAbsolute_1: MC_MoveAbsolute;

MC_MoveAbsolute_2: MC_MoveAbsolute;

MC_MoveAbsolute_3: MC_MoveAbsolute;

END_VAR
```

Step 6 Enter the following code in the main code editor:

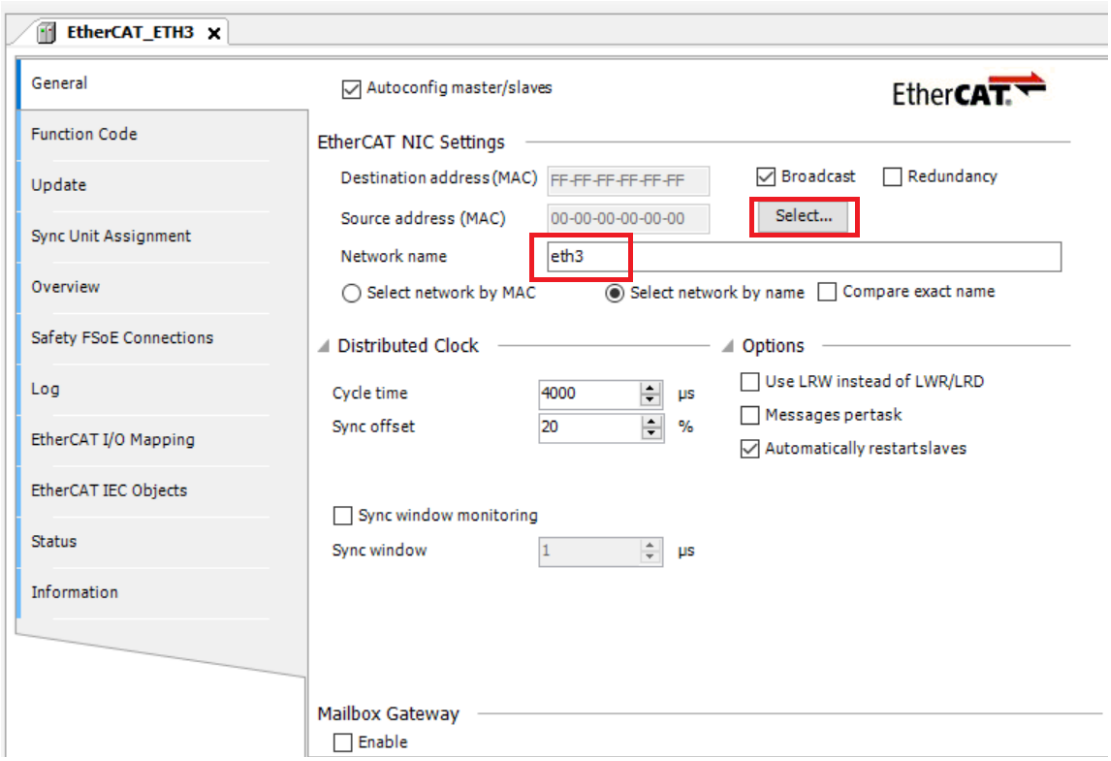
```
CASE iStatus OF

0:

MC_Power_0(Axis:= SM_Drive_GenericDSP402, Enable:= TRUE, bRegulatorOn:= TRUE,
bDriveStart:=TRUE , );

MC_Power_1(Axis:= SM_Drive_GenericDSP402_1, Enable:= TRUE, bRegulatorOn:= TRUE,
bDriveStart:=TRUE , );

MC_Power_2(Axis:= SM_Drive_GenericDSP402_2, Enable:= TRUE, bRegulatorOn:= TRUE,
bDriveStart:=TRUE , );
```

```
MC_Power_3(Axis:= SM_Drive_GenericDSP402_3, Enable:= TRUE, bRegulatorOn:= TRUE,
bDriveStart:=TRUE , );

IF MC_Power_0.Status AND MC_Power_1.Status AND MC_Power_2.Status AND
MC_Power_3.Status THEN

        iStatus:=iStatus+1;

END_IF

1:

MC_MoveAbsolute_0(Axis:=SM_Drive_GenericDSP402 , Execute:= TRUE, Position:=50 ,
Velocity:=3 , Acceleration:= 2, Deceleration:= 100,);

MC_MoveAbsolute_1(Axis:=SM_Drive_GenericDSP402_1, Execute:= TRUE, Position:=50 ,
Velocity:=3 , Acceleration:= 2, Deceleration:=100,);

MC_MoveAbsolute_2(Axis:=SM_Drive_GenericDSP402_2, Execute:= TRUE, Position:=50 ,
Velocity:=3 , Acceleration:= 2, Deceleration:=100,);

MC_MoveAbsolute_3(Axis:=SM_Drive_GenericDSP402_3, Execute:= TRUE, Position:=50 ,
Velocity:=3 , Acceleration:= 2, Deceleration:=100,);

IF MC_MoveAbsolute_0.Done AND MC_MoveAbsolute_1.Done AND MC_MoveAbsolute_2.Done
AND MC_MoveAbsolute_3.Done THEN

            MC_MoveAbsolute_0(Axis:=SM_Drive_GenericDSP402 , Execute:= FALSE,);

    MC_MoveAbsolute_1(Axis:=SM_Drive_GenericDSP402_1 , Execute:= FALSE,);

            MC_MoveAbsolute_2(Axis:=SM_Drive_GenericDSP402_2    ,    Execute:=
FALSE,);

            MC_MoveAbsolute_3(Axis:=SM_Drive_GenericDSP402_3    ,    Execute:=
FALSE,);

            iStatus:=iStatus+1;

END_IF

2:

MC_MoveAbsolute_0(Axis:=SM_Drive_GenericDSP402 , Execute:= TRUE, Position:=0 ,
Velocity:=3, Acceleration:= 2, Deceleration:= 100,);

MC_MoveAbsolute_1(Axis:=SM_Drive_GenericDSP402_1, Execute:= TRUE, Position:=0 ,
Velocity:=3 , Acceleration:= 2, Deceleration:=100,);

MC_MoveAbsolute_2(Axis:=SM_Drive_GenericDSP402_2, Execute:= TRUE, Position:=0 ,
Velocity:=3, Acceleration:= 2, Deceleration:=100,);

MC_MoveAbsolute_3(Axis:=SM_Drive_GenericDSP402_3, Execute:= TRUE, Position:=0 ,
Velocity:=3 , Acceleration:= 2, Deceleration:=100,);

IF MC_MoveAbsolute_0.Done AND MC_MoveAbsolute_1.Done AND MC_MoveAbsolute_2.Done
AND MC_MoveAbsolute_3.Done THEN

            MC_MoveAbsolute_0(Axis:=SM_Drive_GenericDSP402 , Execute:= FALSE,);

            MC_MoveAbsolute_1(Axis:=SM_Drive_GenericDSP402_1    ,    Execute:=
FALSE,);
```

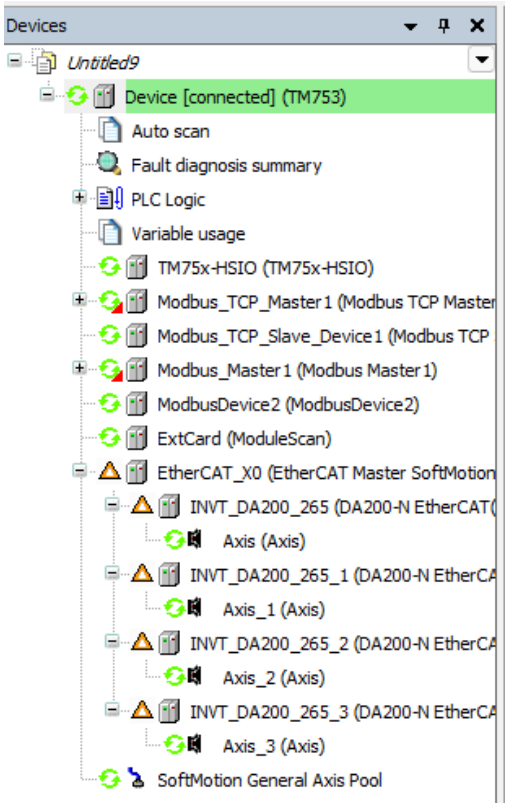
```
            MC_MoveAbsolute_2(Axis:=SM_Drive_GenericDSP402_2    ,    Execute:=
FALSE,);

            MC_MoveAbsolute_3(Axis:=SM_Drive_GenericDSP402_3    ,    Execute:=
FALSE,);

iStatus:=1;

END_IF

END_CASE
```

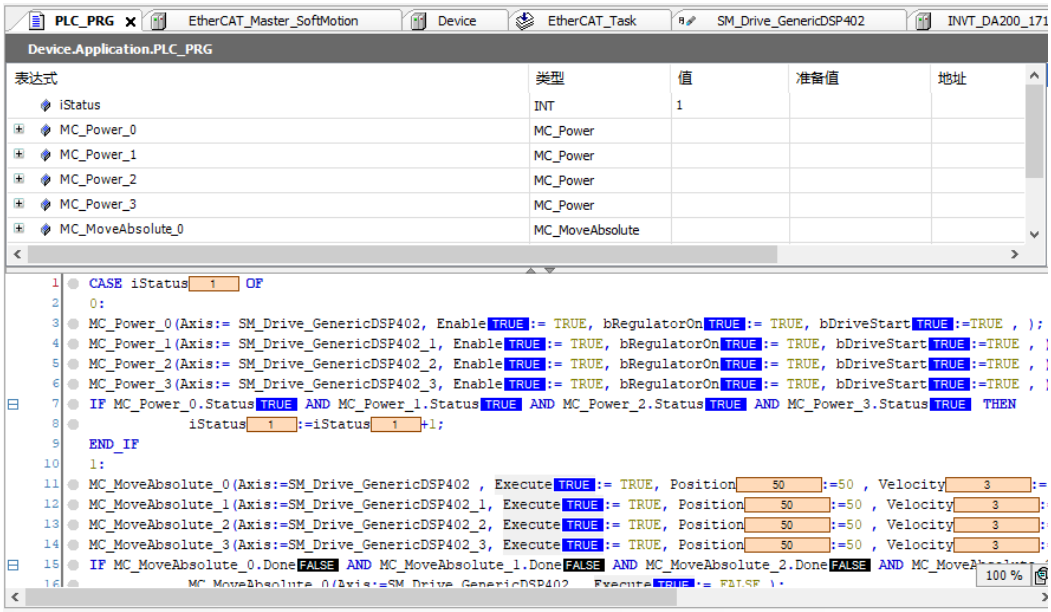
The main body of the program takes the form of a state machine that determines which part of the code to execute through the value of iStatus. When the program starts, the iStatus value is 0, and the program initializes the MC_Power function block and enables the corresponding motor shaft. If the corresponding motor shaft is enabled successfully, the iStatus value is 1 and the program enters the next state. When the iStatus value is 1, the MC_MoveAbsolute function block is executed, and the motor rotates to the specified position at the specified speed. If the motor moves normally to the specified position, the iStatus value is increased by 1, and the motor enters the next state. When the iStatus value is 2, the MC_MoveAbsolute function block is executed in the other direction. The motor continues to rotate to the specified position at the speed specified by the function block. If the motor moves normally to the specified position, the iStatus value is reset to 1. The program is executed repeatedly to implement the forward and reverse movement of the motor.

Step 7 Double-click **EtherCAT_Master_SoftMotion** from the device tree and click **Browse** to select the corresponding EtherCAT communication network named **eth3**. Select the distributed clock as needed. In this example, select 4000 µs as the cycle time, as shown below.

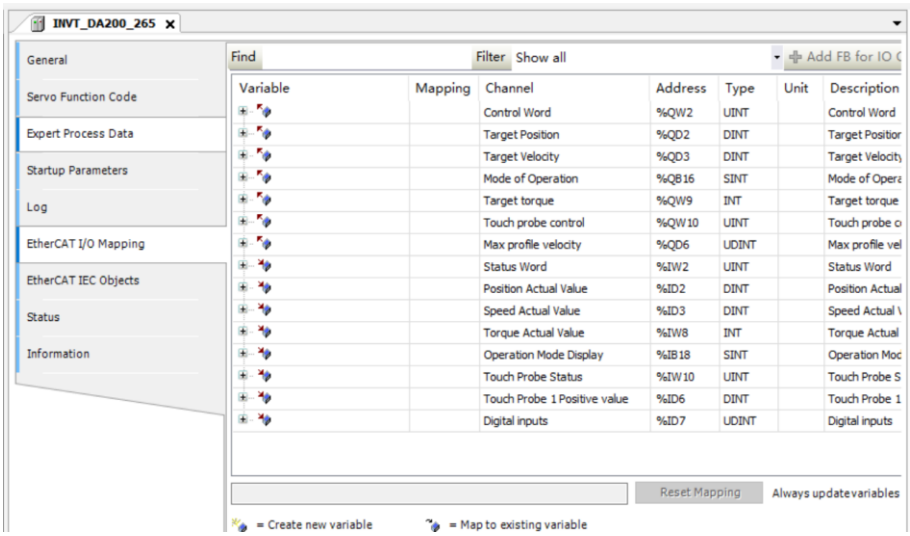


Step 8 Click the button in the Toolbar to compile the code. After compiling, click the button in the Toolbar to log in to the controller. The servo starts normally, the motor runs smoothly, and the upper computer interface is shown in the following figure.

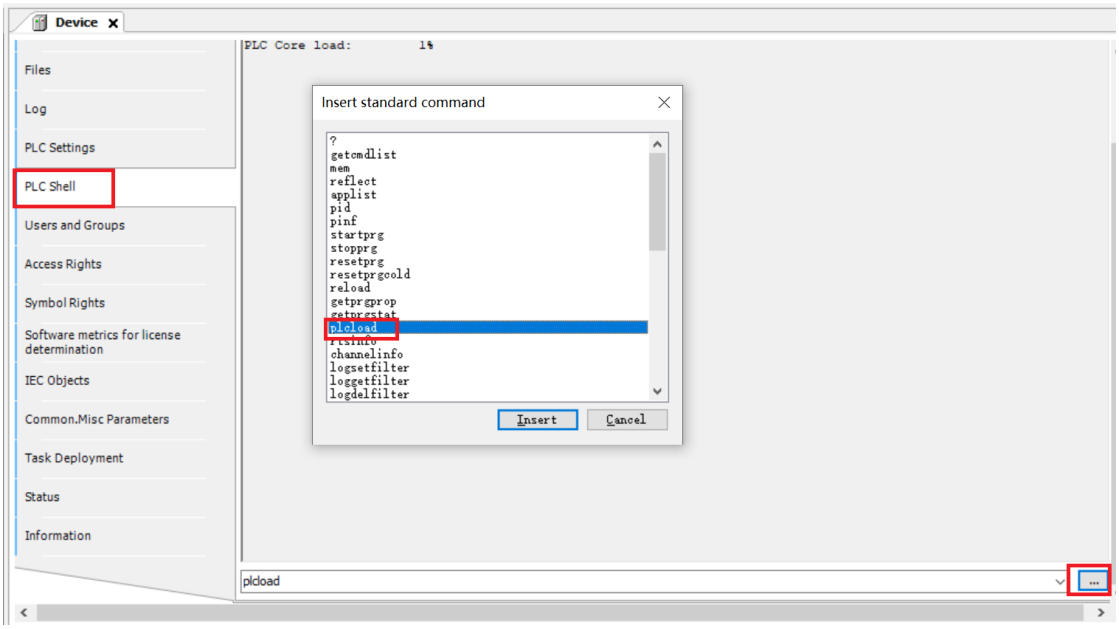




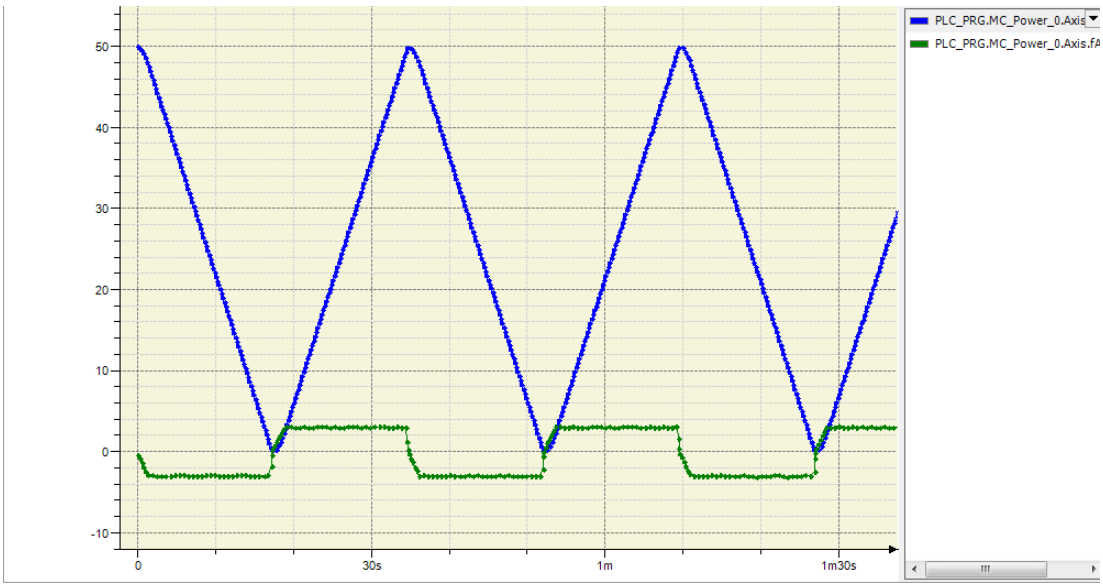
Step 9 Double-click **INVT_DA200** from the device tree to view or set the current motor running parameters in the I/O mapping interface, as shown below.



Step 10 Select **Device** > **PLC Instructions**. Click the button ... at the lower right corner and select **plcload**. Then the CPU load rate of the current controller will be shown as follows, as shown below.



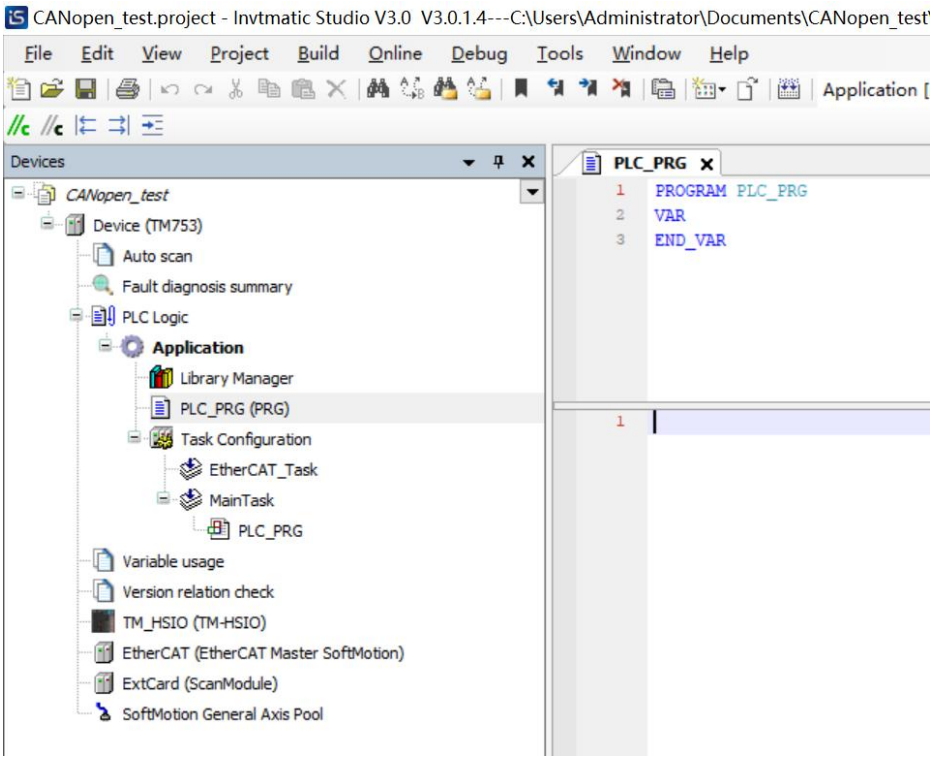
To observe the operation of the motor shaft in an intuitive way and track the actual position of the shaft, create a new trace. Right-click **Application** and select **Add Object** > **Trace**. Set the task attribute to **EtherCAT_ETH3_Task**, and add **PLC_PRG.MC_Power_0.Axis.fActPosition** and **PLC_PRG.MC_Power_0.Axis.fActVelocity** variables in **Trace**. Adjust the display properties of the coordinates appropriately to track the actual position and actual speed of the motor, as shown below.



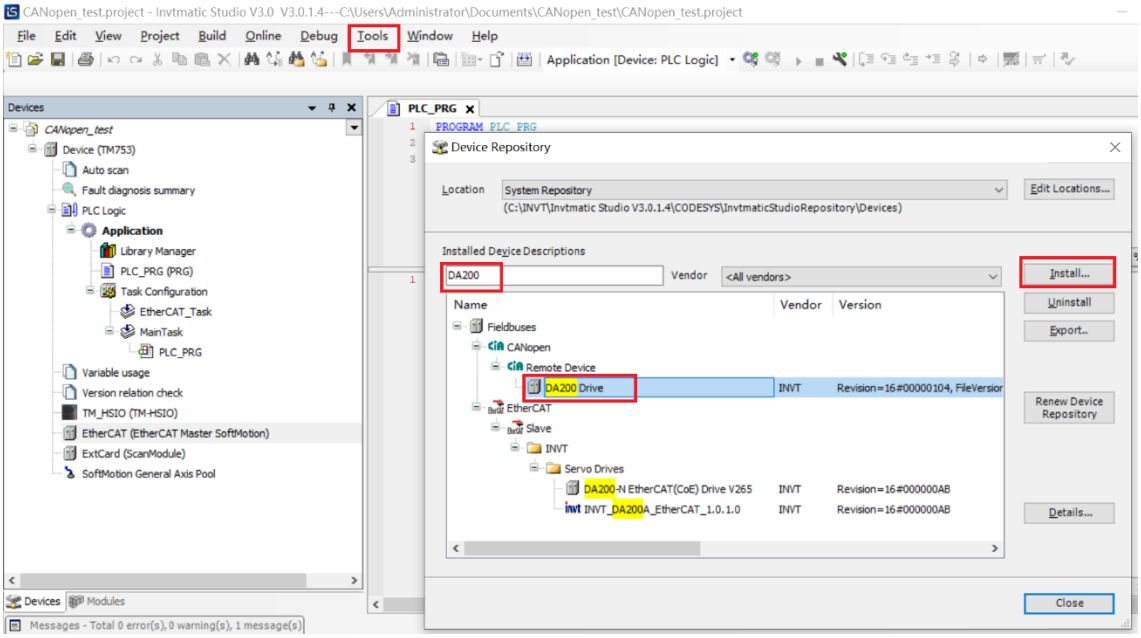
A.3 Example of CANopen Communication Configuration between the Controller and DA200 Series Servo

By writing a small program, you can implement TM700 series PLC CANopen communication to connect DA200 series servo drive. The operation steps are as follows:

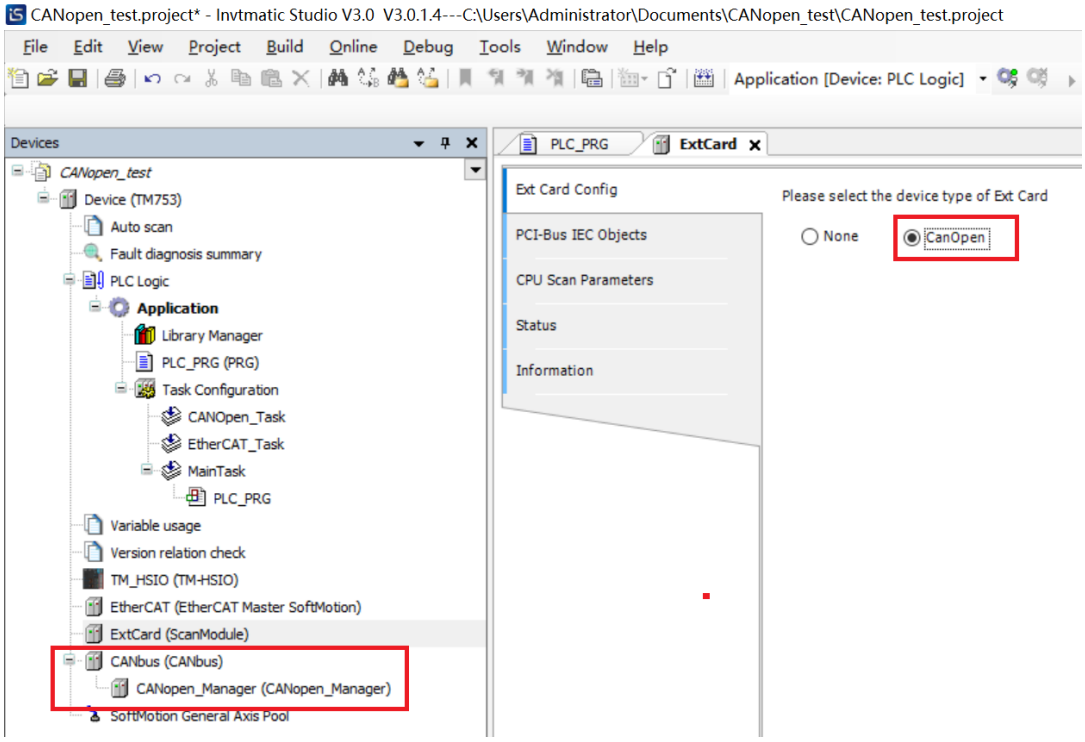
Step 1 Create the following project following section 2.1.2 Creating a New Project.



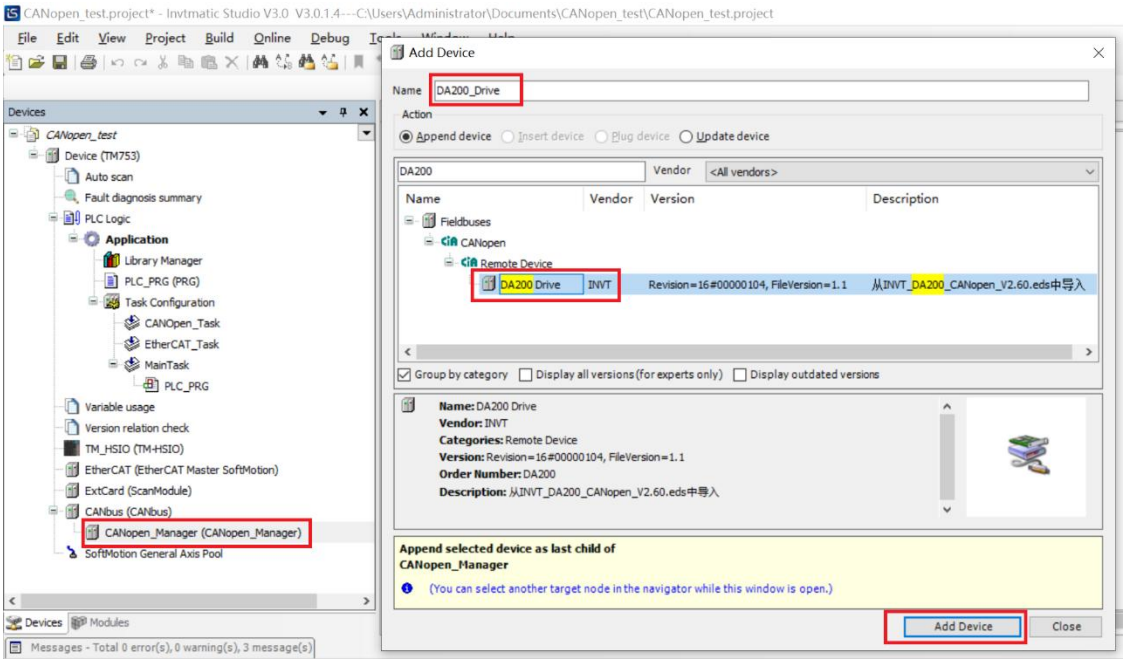
Step 2 Click **Tools** > **Device Repository**, and then click **Install**. Locate and click the device profile **INVT_DA200_CANopen.eds** to open it. At this time, DA200 CANopen device profile is added successfully.



Step 3 Double-click **ExtCard** in the device tree and enter the Settings interface. There is no device option by default. Select CANopen to automatically add CANbus and CANopen_Manager to the left device tree.



Step 4 Right-click **CANopen_Manager** in the device tree, select **Add Device**, and then select the CANopen remote device and **DA200 Drive**. Click **Add Device** in the lower right corner to add DA200 CANopen drive.

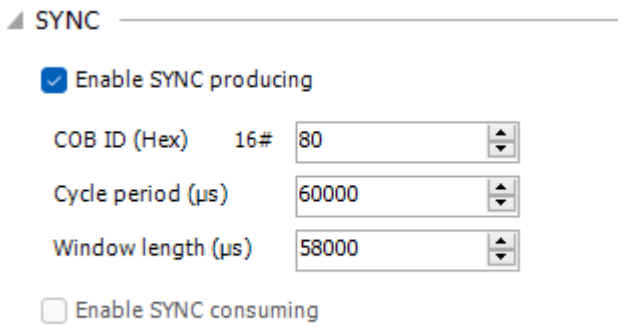


Step 5 On the “CANbus” overview interface, the baud rate configuration needs to be consistent with DA200 CANopen servo (DA200 P4.02); on the “DA200_Drive” overview interface, the node ID configuration needs to be consistent with DA200 CANopen servo (DA200 P4.05).

Step 6 After completing the physical connection of the device, download the program and log in to the device. Then, you can see that the CANopen connection to DA200 is successful.

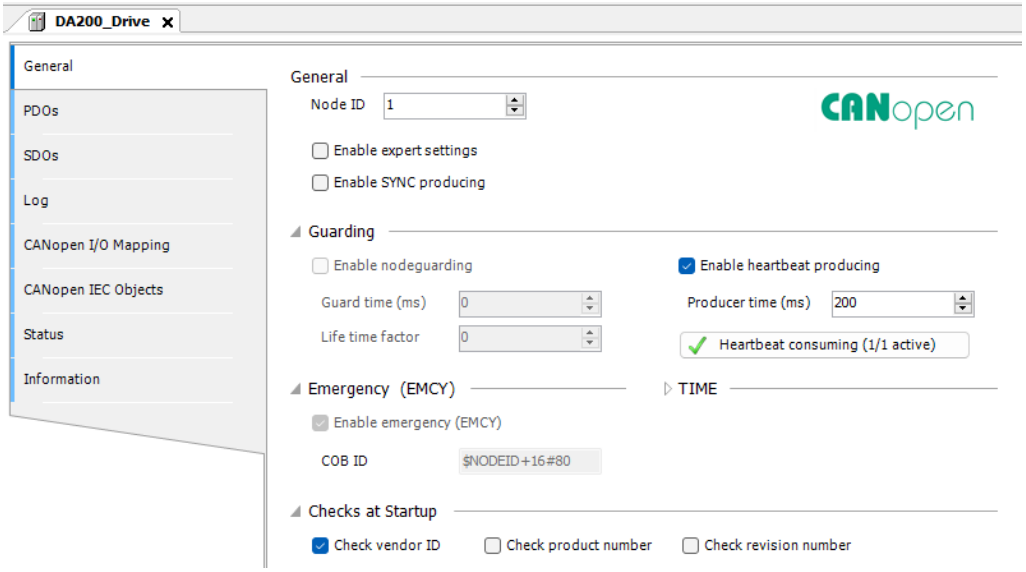
Note:

- If high real-time performance is required for data, the CAN bus load rate should be less than 30% to avoid a small delay in data transmission and reception caused by bus contention.
- For CAN buses with synchronization requirements, the window length setting value in the bus synchronization message should be slightly smaller than the cycle value.



- The cycle time of the CANopen task should be slightly longer than the actual execution time of the task.
- To ensure that the master can monitor the slave normally, check the Heartbeat Enable option in the

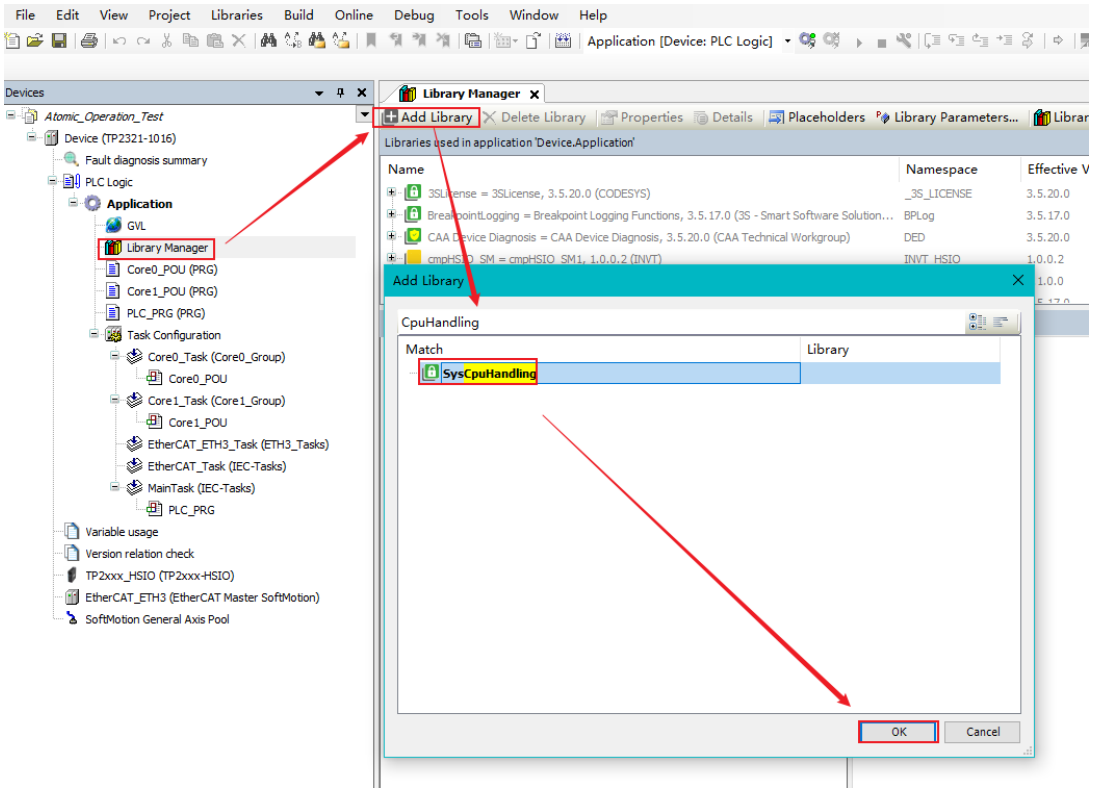
slave.

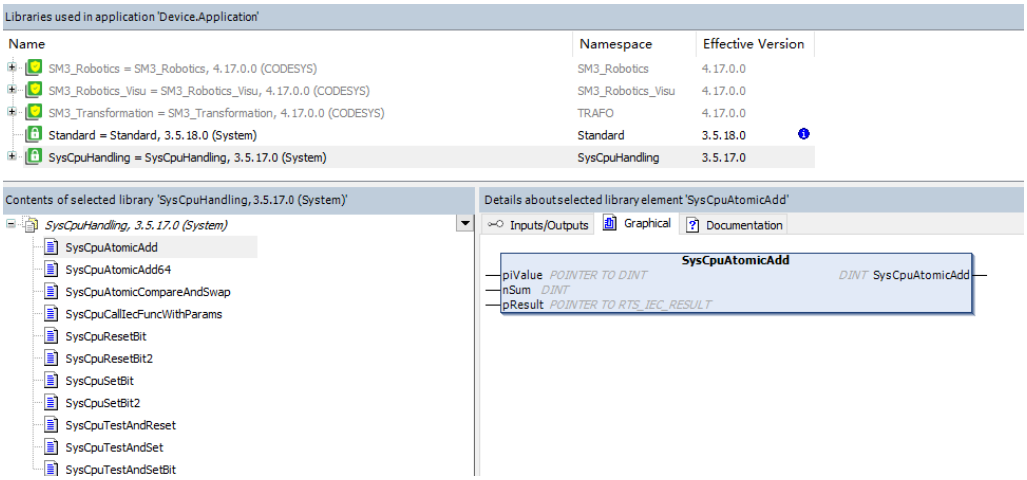


A.4 Example of Atomic Operation

Atomic operations are implemented by adding the corresponding library file and calling the library functions.

Step 1 Add the SysCpuHandling library.

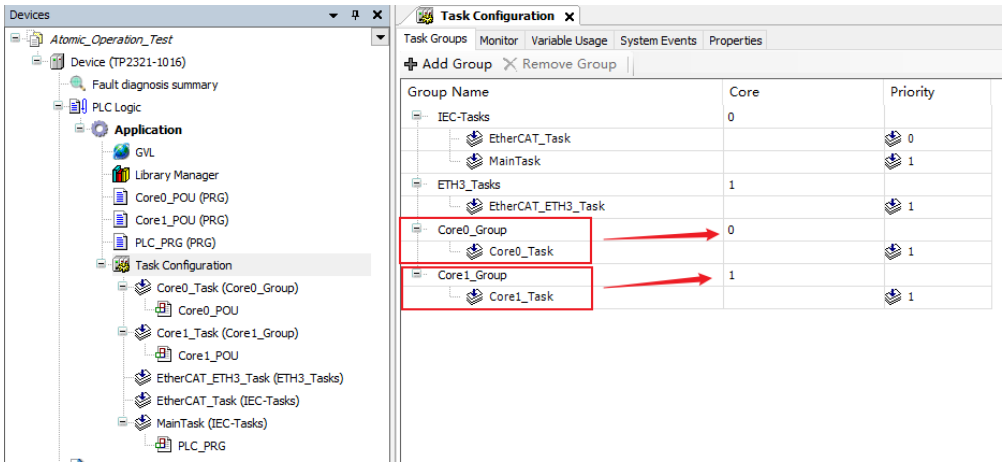




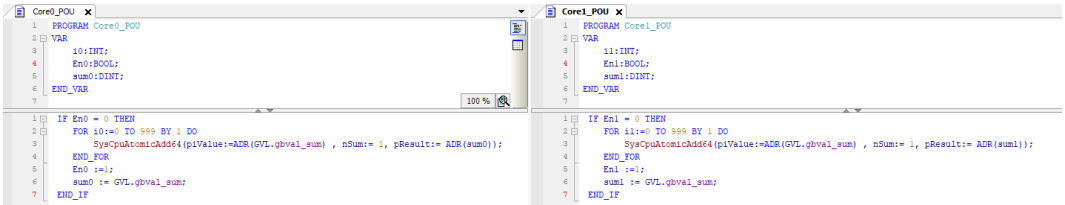
Step 2 Select a function based on the data type.

When the same variable is operated on by tasks bound to different cores, atomic operations are required to ensure that changes to the data are caused only by the current task and not by operations from other tasks.

Core0_POU and Core1_POU run on separate cores.



The global variable gbval_sum is accessed simultaneously by tasks running on two different cores. To ensure data reliability, use the atomic function SysCpuAtomicAdd64() to perform addition on the operands.



Appendix B SMC_ERROR Description

Error Number	Module	ENUM Variable	Description
0	All function blocks	SMC_NO_ERROR	No error
1	DriveInterface	SMC_DI_GENERAL_COMMUNICATION_ERROR	Communication error. For example, sercos ring has broken
2	DriveInterface	SMC_DI_AXIS_ERROR	Axis error
10	DriveInterface	SMC_DI_SWLIMITS_EXCEEDED	Position output within the allowed range (SWLimit)
11	DriveInterface	SMC_DI_HWLIMITS_EXCEEDED	Hardware end switch is active.
13	DriveInterface	SMC_DI_HALT_OR_QUICKSTOP_NOT_SUPPORTED	Drive status Halt or Quickstop is not supported.
14	DriveInterface	SMC_DI_VOLTAGE_DISABLED	The drive is not enabled.
15	DriveInterface	SMC_DI_IRREGULAR_ACTPOSITION	Current position given from the drive seems to be irregular. Check the communication
16	DriveInterface	SMC_DI_POSITIONLAGERROR	Position lag error. Difference between set and current position exceeds the given limit
20	All motion generating function blocks	SMC_REGULATOR_OR_START_NOT_SET	The controller is not enabled or the brake is applied
21	Axis in wrong controller mode	SMC_WRONG_CONTROLLER_MODE	Axis in wrong control mode
30	DriveInterface	SMC_FB_WASNT_CALLED_DURING_MOTION	The module created by motion control is not called before the motion is completed
31	All function blocks	SMC_AXIS_IS_NO_AXIS_REF	The given AXIS_REF variable is not of the type AXIS_REF
32	Axis in wrong controller mode	SMC_AXIS_REF_CHANGED_DURING_OPERATION	AXIS_REF variables have been changed while the modules being activated
33	DriveInterface	SMC_FB_ACTIVE_AXIS_DIABLED	The axis is not activated while moving (MC_Power.bRegulatorOn)
34	All motion generating function blocks	SMC_AXIS_NOT_READY_FOR_MOTION	Axis in its current state cannot execute a motion instruction
40	VirtualDrive	SMC_VD_MAX_VELOCITY_EXCEEDED	Maximum velocity (fMaxVelocity) exceeded
41	VirtualDrive	SMC_VD_MAX_ACCELERATION_EXCEEDED	Maximum acceleration (fMaxAcceleration) exceeded
42	VirtualDrive	SMC_VD_MAX_DECELERATION_EXCEEDED	Maximum deceleration (fMaxDeceleration) exceeded

Error Number	Module	ENUM Variable	Description
50	SMC_Homing	SMC_3SH_INVALID_VELACC_VALUES	Invalid velocity or acceleration values
51	SMC_Homing	SMC_3SH_MODE_NEEDS_HWLIMIT	Mode requests use of limit switches for safety reasons
70	SMC_SetControllerMode	SMC_SCM_NOT_SUPPORTED	Mode not supported
71	SMC_SetControllerMode	SMC_SCM_AXIS_IN_WRONG_STATE	The controller mode cannot be changed in the current state
75	SMC_SetTorque	SMC_ST_WRONG_CONTROLLER_MODE	The axis is under the wrong controller mode
80	SMC_ResetAxisGroup	SMC_RAG_ERROR_DURING_STARTUP	Error occurs when the axis group is activated
90	SMC_ChangeGearingRatio	SMC_CGR_ZERO_VALUES	Invalid values
91	SMC_ChangeGearingRatio	SMC_CGR_DRIVE_POWERED	The gear ratio parameters of the drive cannot be modified when it is under control
92	SMC_ChangeGearingRatio	SMC_CGR_INVALID_POSPERIOD	Invalid position period (≤ 0)
110	MC_Power	SMC_P_FTASKCYCLE_EMPTY	Axis contains no information in the scan cycle (fTaskCycle=0)
120	MC_Reset	SMC_R_NO_ERROR_TO_RESET	Axis reset without error
121	MC_Reset	SMC_R_DRIVE_DOESNT_ANSWER	Axis does not perform error-reset
122	MC_Reset	SMC_R_ERROR_NOT_RESETTABLE	Error could not be reset
123	MC_Reset	SMC_R_DRIVE_DOESNT_ANSWER_IN_TIME	Communication with the axis did not work
130	MC_ReadParameter, MC_ReadBoolParameter	SMC_RP_PARAM_UNKNOWN	Parameter number unknown
131	MC_ReadParameter, MC_ReadBoolParameter	SMC_RP_REQUESTING_ERROR	Error during parameter transmission to the drive. See error number in the Programming Manual ReadDriveParameter (SM_DriveBasic.lib)
140	MC_WriteParameter, MC_WriteBoolParameter	SMC_WP_PARAM_INVALID	Parameter number unknown or writing not allowed
141	MC_WriteParameter, MC_WriteBoolParameter	SMC_WP_SENDING_ERROR	See error number in the Programming Manual WriteDriveParameter (Drive_Basic.lib)
170	MC_Home	SMC_H_AXIS_WASNT_STANDSTILL	Axis has not been in standstill state
171	MC_Home	SMC_H_AXIS_DIDNT_START_HOMING	Error at start of homing action
172	MC_Home	SMC_H_AXIS_DIDNT_ANSWER	Communication error

Error Number	Module	ENUM Variable	Description
173	MC_Home	SMC_H_ERROR_WHEN_STOPPING	Error at stop after homing. Check whether deceleration is set
180	MC_Stop	SMC_MS_UNKNOWN_STOPPING_ERROR	Unknown error at stop
181	MC_Stop	SMC_MS_INVALID_ACCDEC_VALUES	Invalid velocity or acceleration values
182	MC_Stop	SMC_MS_DIRECTION_NOT_APPLICABLE	Direction=shortest not applicable
183	MC_Stop	SMC_MS_AXIS_IN_ERRORSTOP	Drive is in errorstop status. Stop cannot be executed
184	MC_Stop	SMC_BLOCKING_MC_STOP_WASNT_CALLED	An instance of MC_Stop with multiple calls
201	MC_MoveAbsolute	SMC_MA_INVALID_VELACC_VALUES	Invalid velocity or acceleration values
202	MC_MoveAbsolute	SMC_MA_INVALID_DIRECTION	Direction error
226	MC_MoveRelative	SMC_MR_INVALID_VELACC_VALUES	Invalid velocity or acceleration values
227	MC_MoveRelative	SMC_MR_INVALID_DIRECTION	Direction error
251	MC_MoveAdditive	SMC_MAD_INVALID_VELACC_VALUES	Invalid velocity or acceleration values
252	MC_MoveAdditive	SMC_MAD_INVALID_DIRECTION	Direction error
276	MC_MoveSuperImposed	SMC_MSI_INVALID_VELACC_VALUES	Invalid velocity or acceleration values
277	MC_MoveSuperImposed	SMC_MSI_INVALID_DIRECTION	Direction error
301	MC_MoveVelocity	SMC_MV_INVALID_ACCDEC_VALUES	Invalid velocity or acceleration values
302	MC_MoveVelocity	SMC_MV_DIRECTION_NOT_APPLICABLE	Direction=shortest/fastest not applicable
325	MC_PositionProfile	SMC_PP_ARRAYSIZE	Erroneous array size
326	MC_PositionProfile	SMC_PP_STEP0MS	Step time = t#0ms
350	MC_VelocityProfile	SMC_VP_ARRAYSIZE	Erroneous array size
351	MC_VelocityProfile	SMC_VP_STEP0MS	Step time = t#0ms
375	MC_AccelerationProfile	SMC_AP_ARRAYSIZE	Erroneous array size
376	MC_AccelerationProfile	SMC_AP_STEP0MS	Step time = t#0ms
400	MC_TouchProbe	SMC_TP_TRIGGEROCCUPIED	Trigger already active
401	MC_TouchProbe	SMC_TP_COULDNT_SET_WINDOW	Drive interface does not support the window function
402	MC_TouchProbe	SMC_TP_COMM_ERROR	Communication error
410	MC_AbortTrigger	SMC_AT_TRIGGERNOTOCCUPIED	Trigger already de-allocated
426	SMC_MoveContinuousRelative	SMC_MCR_INVALID_VELACC_VALUES	Invalid velocity or acceleration values
427	SMC_MoveContinuousRelative	SMC_MCR_INVALID_DIRECTION	Direction error

Error Number	Module	ENUM Variable	Description
451	SMC_MoveContinuousAbsolute	SMC_MCA_INVALID_VELACC_VALUES	Invalid velocity or acceleration values
452	SMC_MoveContinuousAbsolute	SMC_MCA_INVALID_DIRECTION	Direction error
453	SMC_MoveContinuousAbsolute	SMC_MCA_DIRECTION_NOT_APPLICABLE	Direction=fastest not applicable
600	SMC_CamRegister	SMC_CR_NO_TAPPETS_IN_CAM	Cam does not contain any tappets
601	SMC_CamRegister	SMC_CR_TOO_MANY_TAPPETS	Tappet group ID exceeds MAX_NUM_TAPPETS
602	SMC_CamRegister	SMC_CR_MORE_THAN_32_ACCESSES	More than 32 accesses in one CAM_REF
625	MC_CamIN	SMC_CI_NO_CAM_SELECTED	No cam selected
626	MC_CamIN	SMC_CI_MASTER_OUT_OF_SCALE	Master axis out of valid range
627	MC_CamIN	SMC_CI_RAMPIN_NEEDS_VELACC_VALUES	Velocity and acceleration values must be specified for ramp_in function
628	MC_CamIN	SMC_CI_SCALING_INCORRECT	Scaling variables fEditor/TableMasterMin/Max are not correct
640	SMC_CAMBounds, SMC_CamBounds_Pos	SMC_CB_NOT_IMPLEMENTED	Function block for the given cam format is not implemented
675	MC_GearIn	SMC_GI_RATIO_DENOM	RatioDenominator=0
676	MC_GearIn	SMC_GI_INVALID_ACC	Acceleration invalid
677	MC_GearIn	SMC_GI_INVALID_DEC	Deceleration invalid
725	MC_Phase	SMC_PH_INVALID_VELACCDEC	Velocity and acceleration/deceleration values invalid
726	MC_Phase	SMC_PH_ROTARYAXIS_PERIOD0	Rotation axis with fPositionPeriod = 0
750	All modules using MC_CAM_REF as input	SMC_NO_CAM_REF_TYPE	Type of given cam is not MC_CAM_REF
751	MC_CamTableSelect	SMC_CAM_TABLE_DOES_NOT_COVER_MASTER_SCALE	Master axis area (xStart and xEnd) from CamTable is not covered by curve data
775	MC_GearInPos	SMC_GIP_MASTER_DIRECTION_CHANGE	During coupling of slave axis, master axis has changed direction of rotation
800	SMC_BacklashCompensation	SMC_BC_BL_TOO_BIG	Gear backlash fBacklash too large (> position period/2)
1000	CNC function blocks which are supervising the licensing	SMC_NO_LICENSE	Target is not licensed for CNC

Error Number	Module	ENUM Variable	Description
1001	SMC_Interpolator	SMC_INT_VEL_ZERO	Path cannot be processed because set velocity = 0
1002	SMC_Interpolator	SMC_INT_NO_STOP_AT_END	Last object of path has Vel_End>0
1003	SMC_Interpolator	SMC_INT_DATA_UNDERRUN	Warning: GEOINFO List processed in DataIn but end of list not reached Reason: EndOfList of the queue in DataIn not be set or SMC_Interpolator faster than path generating function blocks
1004	SMC_Interpolator	SMC_INT_VEL_NONZERO_AT_STOP	Velocity at Stop > 0
1005	SMC_Interpolator	SMC_INT_TOO_MANY_RECURSIONS	Too many SMC_Interpolator recursions. SoftMotion error
1006	SMC_Interpolator	SMC_INT_NO_CHECKVELOCITIES	Input-OutQueueDataIn is not the last processed function block of SMC_CHeckVelocities
1007	SMC_Interpolator	SMC_INT_PATH_EXCEEDED	Internal/numeric error
1008	SMC_Interpolator	SMC_INT_VEL_ACC_DEC_ZERO	Velocity and acceleration / deceleration is null or too low
1009	SMC_Interpolator	SMC_INT_DWIPOTIME_ZERO	FB called with dwIpoTime = 0
1050	SMC_Interpolator2Dir	SMC_INT2DIR_BUFFER_TOO_SMALL	Data buffer too small
1051	SMC_Interpolator2Dir	SMC_INT2DIR_PATH_FITS_NOT_IN_QUEUE	Path does not go completely in queue
1100	SMC_CheckVelocities	SMC_CV_ACC_DEC_VEL_NONPOSITIVE	Velocity and acceleration/deceleration values non-positive
1120	SMC_Controlaxisbypos	SMC_CA_INVALID_ACCDEC_VALUES	Values of fGapVelocity / fGapAcceleration / fGapDeceleration non-positive
1200	SMC_NCDecoder	SMC_DEC_ACC_TOO_LITTLE	Acceleration value not allowed
1201	SMC_NCDecoder	SMC_DEC_RET_TOO_LITTLE	Deceleration value not allowed
1202	SMC_NCDecoder	SMC_DEC_OUTQUEUE_RAN_EMPTY	Data underrun. Queue has been read and is empty
1203	SMC_NCDecoder	SMC_DEC_JUMP_TO_UNKNOWN_LINE	Jump to line cannot be executed because line number is unknown
1204	SMC_NCDecoder	SMC_DEC_INVALID_SYNTAX	Syntax invalid
1205	SMC_NCDecoder	SMC_DEC_3DMODE_OBJECT_NOT_SUPPORTED	Objects are not supported in 3D mode
1300	SMC_GCodeViewer	SMC_GCV_BUFFER_TOO_SMALL	Buffer too small
1301	SMC_GCodeViewer	SMC_GCV_BUFFER_WRONG_TYPE	Buffer elements have wrong types
1302	SMC_GCodeViewer	SMC_GCV_UNKNOWN_IPO_LINE	Current line of the Interpolator could not be found
1500	All function blocks using SMC_CNC_REF	SMC_NO_CNC_REF_TYPE	Given CNC program is not of the SMC_CNC_REF type

Error Number	Module	ENUM Variable	Description
1501	All function blocks using SMC_OUTQUEUE	SMC_NO_OUTQUEUE_TYPE	Given OutQueue is not of the SMC_OUTQUEUE type
1600	CNC function blocks	SMC_3D_MODE_NOT_SUPPORTED	Function block only works with 2D paths
2000	SMC_ReadNCFile	SMC_RNCF_FILE_DOESNT_EXIST	File does not exist
2001	SMC_ReadNCFile	SMC_RNCF_NO_BUFFER	No buffer allocated
2002	SMC_ReadNCFile	SMC_RNCF_BUFFER_TOO_SMALL	Buffer too small
2003	SMC_ReadNCFile	SMC_RNCF_DATA_UNDERRUN	Data underrun. Buffer has been read and is empty
2004	SMC_ReadNCFile	SMC_RNCF_VAR_COULDNT_BE_REPLACED	Placeholder variable could not be replaced
2005	SMC_ReadNCFile	SMC_RNCF_NOT_VARLIST	Input pvl does not point to a SMC_VARLIST object
2050	SMC_ReadNCQueue	SMC_RNCQ_FILE_DOESNT_EXIST	File could not be opened
2051	SMC_ReadNCQueue	SMC_RNCQ_NO_BUFFER	No buffer defined
2052	SMC_ReadNCQueue	SMC_RNCQ_BUFFER_TOO_SMALL	Buffer too small
2053	SMC_ReadNCQueue	SMC_RNCQ_UNEXPECTED_EOF	Unexpected end of file
2100	SMC_AxisDiagnosticLog	SMC_ADL_FILE_CANNOT_BE_OPENED	File could not be opened
2101	SMC_AxisDiagnosticLog	SMC_ADL_BUFFER_OVERRUN	Buffer overrun. WriteToFile must be called more frequently
2200	SMC_ReadCAM	SMC_RCAM_FILE_DOESNT_EXIST	File could not be opened
2201	SMC_ReadCAM	SMC_RCAM_TOO_MUCH_DATA	Saved cam too big
2202	SMC_ReadCAM	SMC_RCAM_WRONG_COMPILE_TYPE	Wrong compilation mode
2203	SMC_ReadCAM	SMC_RCAM_WRONG_VERSION	Wrong file version
2204	SMC_ReadCAM	SMC_RCAM_UNEXPECTED_EOF	Unexpected end of file
3001	SMC_WriteDriveParamsToFile	SMC_WDPF_CHANNEL_OCCUPIED	SMC_WDPF_TIMEOUT_PREPARING_LIST
3002	SMC_WriteDriveParamsToFile	SMC_WDPF_CANNOT_CREATE_FILE	File could not be created
3003	SMC_WriteDriveParamsToFile	SMC_WDPF_ERROR_WHEN_READING_PARAMS	Error at reading the parameters
3004	SMC_WriteDriveParamsToFile	SMC_WDPF_TIMEOUT_PREPARING_LIST	Timeout during preparing the parameter list
5000	SMC_Encoder	SMC_ENC_DENOM_ZERO	Nominator of the conversion factor dwRatioTechUnitsDenom of the Encoder reference is 0
5001	SMC_Encoder	SMC_ENC_AXISUSEDBYOTHERFB	Other module trying to process motion on the Encoder axis
5002	DriveInterface	SMC_ENC_FILTER_DEPTH_INVALID	Filter depth invalid

Your Trusted Industry Automation Solution Provider

Shenzhen INVT Electric Co., Ltd.
Address: INVT Guangming Technology Building, Songbai Road, Matian,
Guangming District, Shenzhen, China
INVT Power Electronics (Suzhou) Co., Ltd.
Address: No. 1 Kunlunshan Road, Science & Technology Town,
Suzhou New District, Jiangsu, China
Website: www.invt.com

INVT mobile website

INVT e-manual

66001-01387

202605 (V1.1)