受　控

GCAN®广成科技

GCAN PLC入门应用指南

文档名称：GCANPLC 入门应用指南					
文档名	版本号	版本更新说明	更改日期	更改人	发布日期
GCANPLC 入门应用指南	V0.1	创建文档	2021.09.16	庞斯奇	2021.11.05

目 录

一、产品基本信息

1. GCAN-PLC 系列产品简介

1.1 功能概述

GCAN-PLC 系列产品（GCAN-PLC-400、GCAN-PLC-510、GCAN-PLC-511 等）是集成有总线控制功能的可编程逻辑控制器（PLC）。其不仅具有外观简约、高性价比的特点，还可以方便的连接入 CAN 总线系统及 Modbus 系统等，并可通过模块插片式进行扩展。

GCAN-PLC 系列产品由一个可编程的主控模块（GCAN-PLC-510 等）、若干 GC 系列 IO 模块以及一个终端端子模块（GC-0001）组成。GCAN-PLC 系列模块可连接所有的 GC 系列 IO 模块，用户可根据现场实际需求自行选择扩展 IO 模块，在电源供应充足情况下，IO 模块数量最多可扩展 32 个，GCAN-PLC 系列可根据插入 IO 模块的前后位置自动分配硬件地址，实现自动组态，用户无需在 PC 上创建组态界面及设置参数，**硬件组态详细说明将在本书第二章第 3 小节展示，本章不进行详解**。

GCAN-PLC 系列产品可使用 OpenPCS 软件对其编程，该软件支持符合 IEC-61131-3 标准中规定的五种标准编程语言，这使得程序的可移植性和复用性很强，而且该软件还具有多种调试功能（断点、单步等），调试程序更加方便。

GCAN-PLC -510 不仅可完成各种数字/模拟量的输入/输出，还集成了多种常用的工业现场总线接口，如：CAN 总线、RS232/485 总线、以太网总线，并支持常见的通信协议如：CANopen、Modbus RTU、Modbus TCP 等。

GC 系列 IO 模块包括：开关量输入及输出模块、模拟量输入及输出模块、脉冲输入及输出模块、通讯口扩展模块（RS232/485、4G、蓝牙、WiFi、Zigbee 、GPRS 等）详细选型列表请参阅本章第 2 小节。

1.2 接口描述

1.2.1 以太网接口 Ethernet RJ45

以太网接口可以通过 OpenPCS 编程，可作为 TCP SERVER、TCP CLIENT、UDP、modbus TCP 使用，在交付状态下，以太网可用来与 OpenPCS 联机，下载程序。

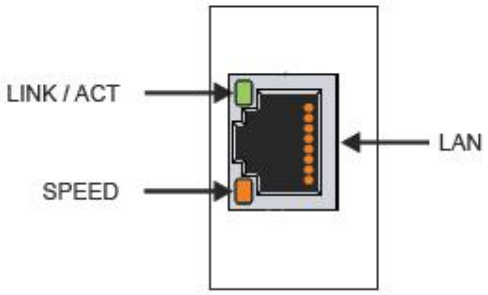


图 1.1：以太网接口

以太网接口的速度均为 100 Mbit。 接口左侧的 LED 指示连接状态。 上方的 LED（LINK / ACT）指示接口是否已连接到网络。

若已连接到网络，LED 为绿色。

下方的 LED（SPEED）为黄色。 正在进行数据传输时，Speed 闪烁。

图 1.2：以太网接口，插针编号

pin	信号	描述
1	T2 +	Pair 2
2	T2 -	
3	T3 +	Pair 3
4	T1 +	Pair 1
5	T1 -	
6	T3 -	Pair 3
7	T4 +	Pair 4
8	T4 -	

表 1.1：以太网接口引脚分配

1.2.2 串口 RS232/485 RJ45 接口

RS232 通信的接口形式是 RJ45，可以通过 OpenPCS 编程。作为 RS232 通信，modbus RTU 通信使用。我们随货附赠 RJ45 转 DB9 的线，具体引脚见下表。

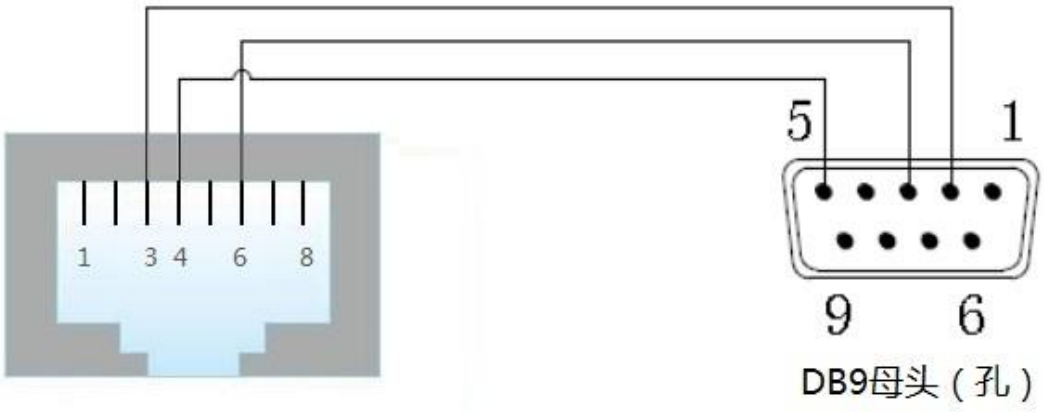


图 1.3 转接线引脚定义

端子	RJ45 端序号	DB9 端序号	含义
RS232_TX	3	2	RS232 数据发送
RS232_RX	6	3	RS232 数据接收
GND	4	5	信号地
RS485_A+	8	7	RS485 信号 A+
RS485_B-	1	8	RS485 信号 B-

表 1.2：串口 RS232/485 RJ45 接口引脚分配

1.2.3 CAN 接口

CAN 通信的接口形式是 OPEN 端子，可以通过 OpenPCS 编程。作为 CAN 通信，CANOPEN 通信，使用。具体引脚见下图。

图 1.4 CAN 引脚分配

1.2.4 供电接口

电源端子需要外部电压源，该电压源可提供 24V DC（-15％/ + 20％）。 电源端子必须在 24V 电压下提供 1A 电流，以确保 PLC 在所有情况下均能正常工作。

控制柜中 PLC 的电缆连接必须符合标准 EN 60204-1：2006 PELV =超低压保护：

•基本 CPU 模块的电压源的“PE”和“0 V”导体必须处于相同电位（连接在控制柜中）。

•标准 EN 60204-1：2006 的 6.4.1：b 节规定，电路的一侧或该电路的能量源必须连接到接地保护系统。

•PLC 与后面扩展模块在必要时需分开供电。

•为扩展模块供电的电源的电压为 24V DC（-15％/ + 20％），该电源提供的最小功率要根据扩展模块消耗电流和计算。

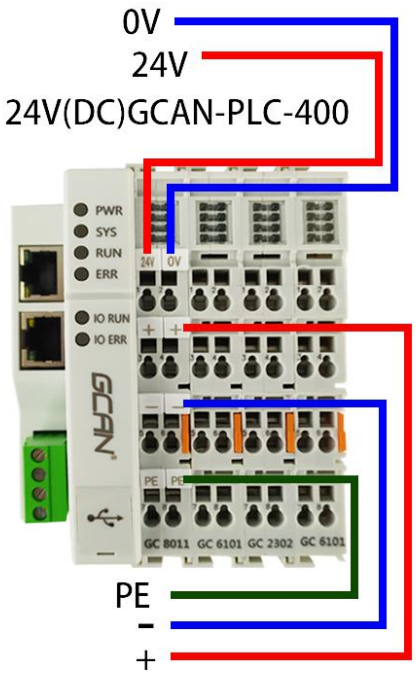


图 1.5 电源接线示意图

序号	描述
1	上面标有“24 V”和“0 V”的弹簧加载端子为 PLC 供电。
2	弹簧端子标识 “+”，“-”，通过电源触点为 IO 总线端子供电。

表 1.3 供电端子引脚分配

1.3 技术参数

	GCAN-PLC-510
尺寸（长*宽*高）	47mm×68mm×97mm
重量	180g

表 1.4　技术数据，尺寸和重量

技术参数	GCAN-PLC-510
处理器	ARM Cortex-M 系列
Flash（程序存储器）	32M 字节
SRAM（变量存储器）	16M 字节
用户数据存储区	2k 字节
供电方式	24 V DC (-15 %/+20 %)
最大能源消耗	12W
CAN/485 隔离等级	1500V

上位机软件	OpenPCS
诊断指示灯	一个电源灯，一个系统灯，一个运行灯，一个故障灯，一个 IO 总线灯，一个 IO 总线故障灯
时钟	内部电池支持时钟，用于显示时间和日期
认证	CE

表 1.5 技术数据，一般数据

技术参数	描述
本机 I/O	无，需扩展 GC 系列 IO 模块
组态方式	通过 IO 总线自动组态
扩展端子模块数量	32 个
数字量 I/O 信号	最多支持 32×8 输入/输出
模拟量 I/O 信号	最多支持 32×4 输入，32×2 输出

表 1.6 技术数据，I / O 端子

技术参数	描述
工作温度	-40° C 至+85° C
工作湿度	95%RH，无凝露
抗振性	3 轴 10 次扫频 6 Hz <f <58.1 Hz 位移 0.15 毫米，恒定振幅

	58.1 Hz <f <500 Hz 加速度 5 g，恒定振幅 符合 EN 60068-2-6
抗冲击	在 3 个轴上的每个方向上有 1000 次冲击 15 g，11 毫秒 符合 EN 60068-2-27

表 1.7 技术数据，环境条件

支持的通信	接口形式
CAN,CANOPEN	OPEN4 端子
RS232,RS485,MODBUS RTU	RJ45
TCP,UDP,MODBUS TCP	RJ45

表 1.8 技术数据、接口

2. GC-IO 模块产品简介

2.1 IO 模块分类

数字量输入模块：GC-1008、GC-1018、GC-1502

数字量输出模块：GC-2008、GC-2018、GC-2204、GC-2302

模拟量输入模块：GC-3604、GC-3624、GC-3644、GC-3654、GC-3664、GC-3674

温度采集输入模块：GC-3804、GC-3822、GC-3844、GC-3854、GC-3864

模拟量输出模块：GC-4602、GC-4622、GC-4642、GC-4652、GC-4662、GC-4672、GC-4674

通讯扩展模块：GC-6101、GC-6221、GC-6501

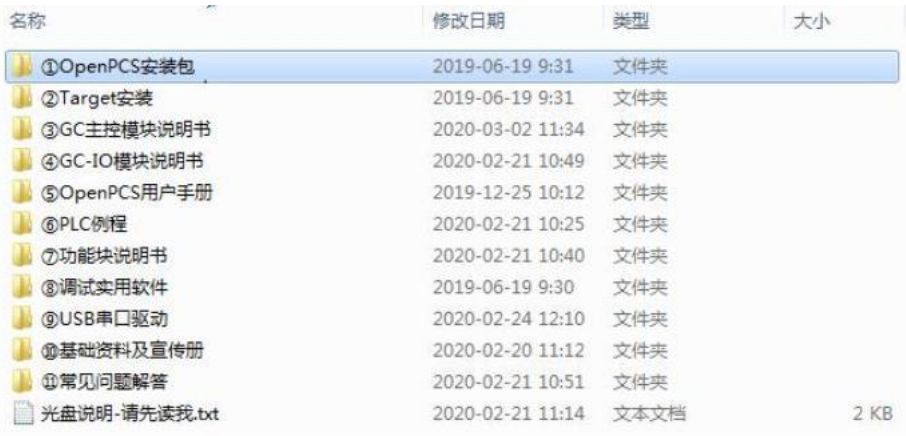
2.2 IO 模块选型表

种类	型号	特性	信号	通道数
数字量输入	GC-1008	基本数字量	24V DC	8 通道
	GC-1502	计数器（200kHz max）	-	2 通道
数字量输出	GC-2008	基本数字量	24V DC	8 通道
	GC-2204	继电器导通	-	4 通道
	GC-2302	PWM（20Hz~200kHz）	-	2 通道
模拟量输入	GC-3604	电压输入，16 位	-5V~+5V	4 通道
	GC-3624	电压输入，16 位	-10V~+10V	4 通道
	GC-3644	电流输入，16 位	0-20mA	4 通道
	GC-3654	电流输入，16 位	4-20mA	4 通道
	GC-3664	电压输入，16 位	0~+5V	4 通道
	GC-3674	电压输入，16 位	0~+10V	4 通道
	GC-3804	2 线制 PT100，16 位	热电阻	4 通道
	GC-3822	3 线制 PT100，16 位	热电阻	2 通道
	GC-3844/3854/3864	K 型/ S 型/ T 型热电偶	热电偶	4 通道
模拟量输出	GC-4602	电压输出，16 位	-5V~+5V	2 通道
	GC-4622	电压输出，16 位	-10V~+10V	2 通道
	GC-4642	电流输出，16 位	0-20mA	2 通道
	GC-4652	电流输出，16 位	4-20mA	2 通道
	GC-4662	电压输出，16 位	0~5V	2 通道
	GC-4672	电压输出，16 位	0~10V	2 通道
特殊扩展模块	GC-6101	RS232/RS485 扩展	-	-
	GC-6201	GPRS 扩展	-	-
	GC-6221	4G 扩展	-	-
	GC-6501	WiFi 扩展	-	-

3. 编程软件（OpenPCS）安装、配置

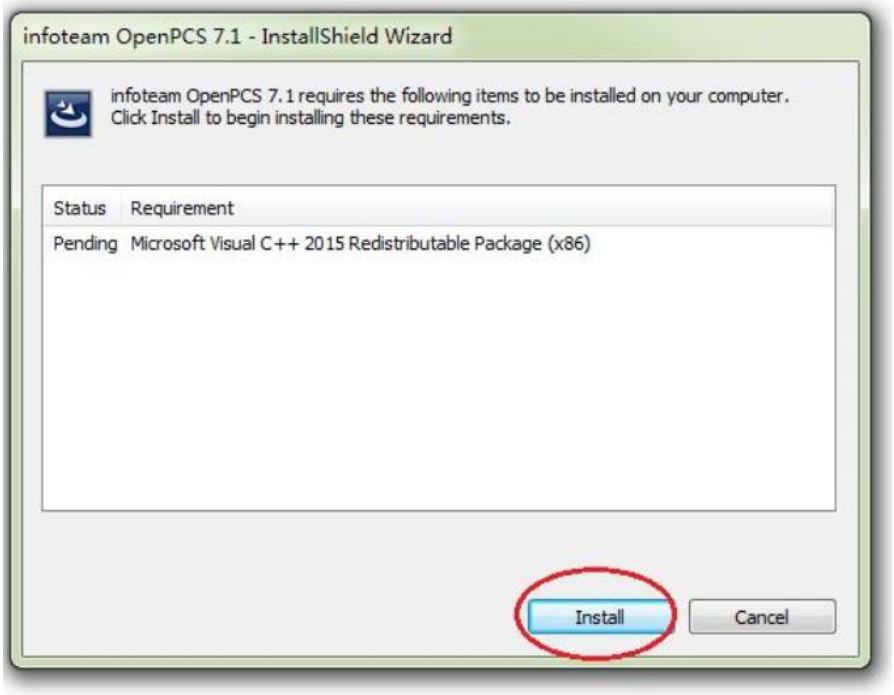
3.1 OpenPCS 安装

（1）打开网盘资料中的①号文件夹，OpenPCS 安装包

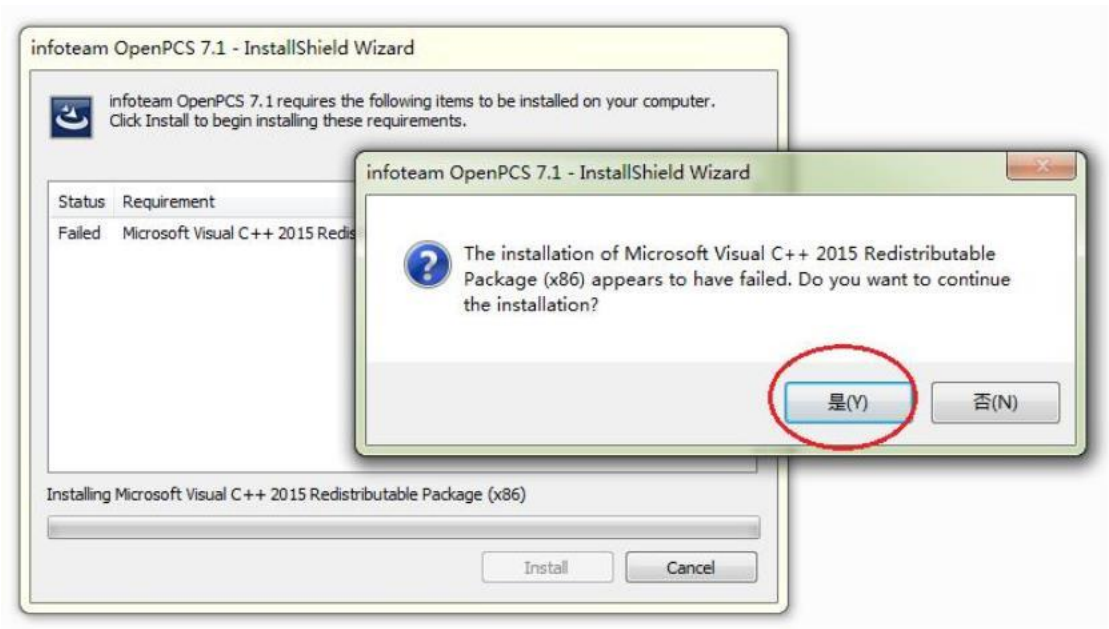


（2）选择英文版或中文版解压后双击.exe 文件安装，本文只介绍英文版，中文版安装过程与英文版相同。

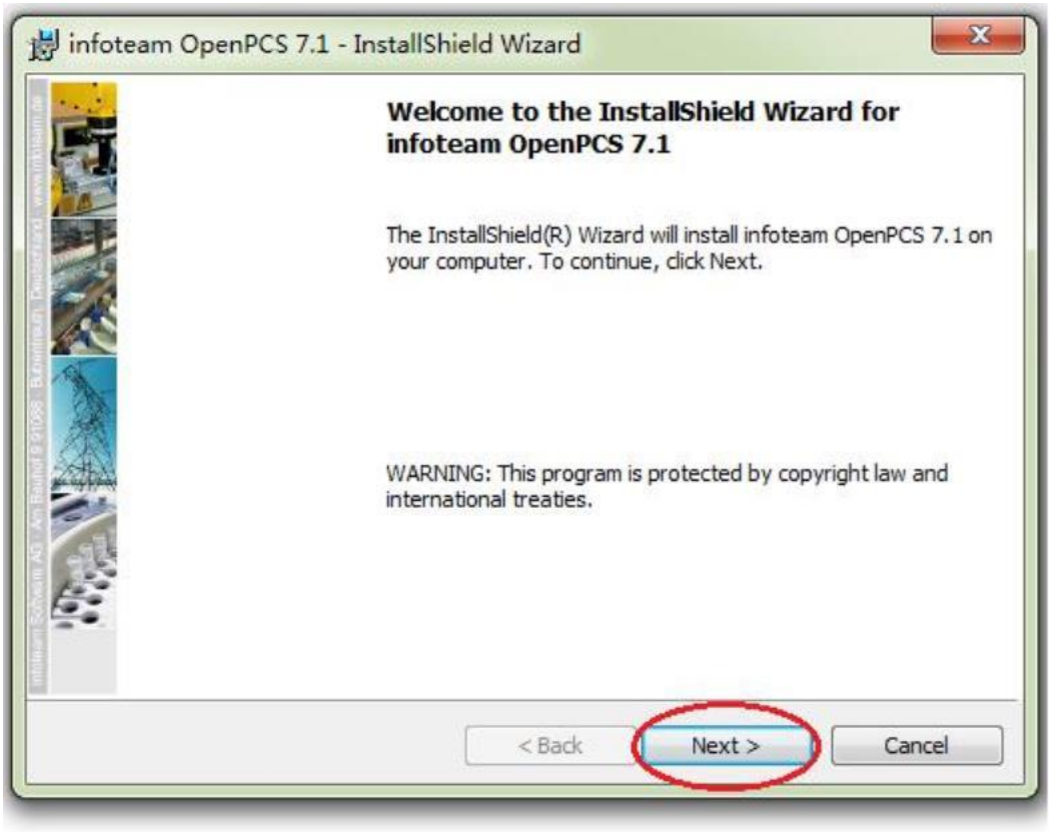
（3）出现安装界面，点击安装



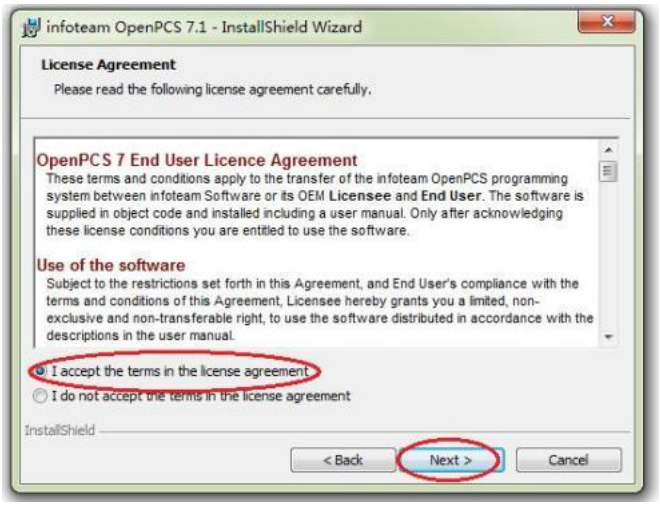
（4）安装过程中可能会出现以下提示，点击是（Y）即可



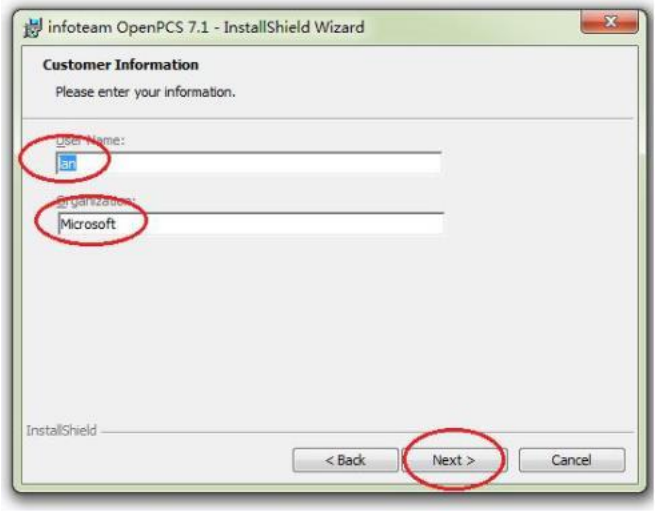
（5）点击下一步继续安装



（6）选择接受许可，点击下一步

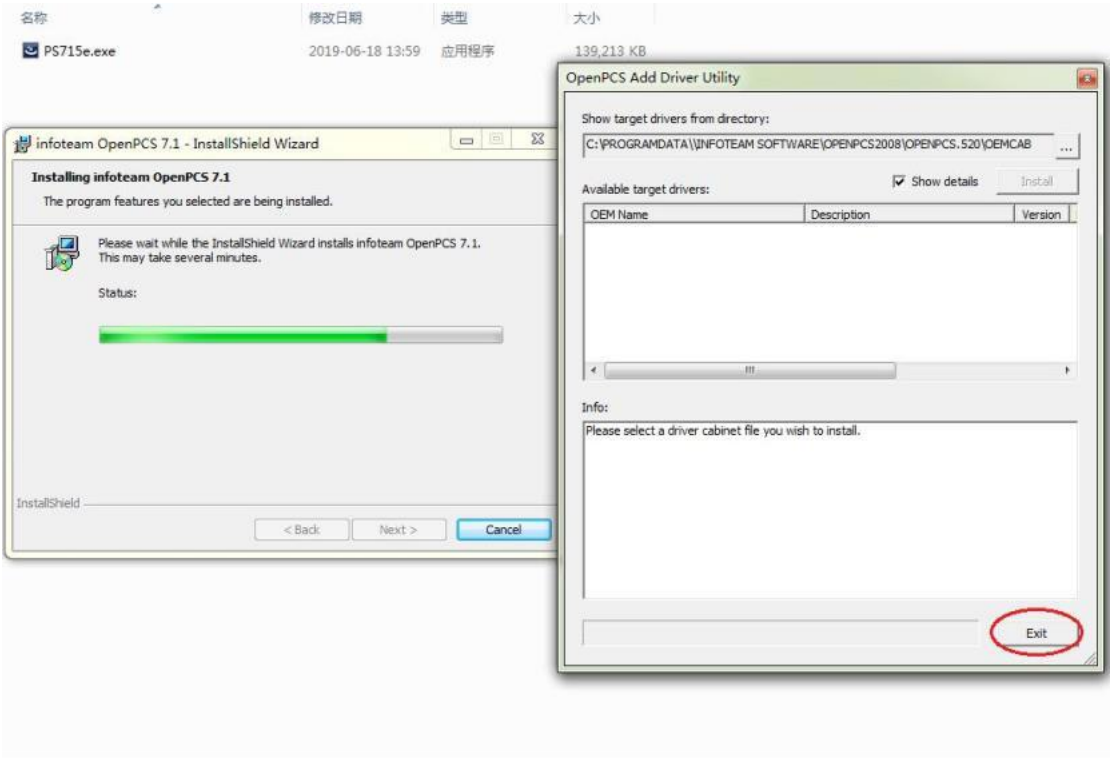


（7）填写用户名称，此处建议使用默认直接点击下一步

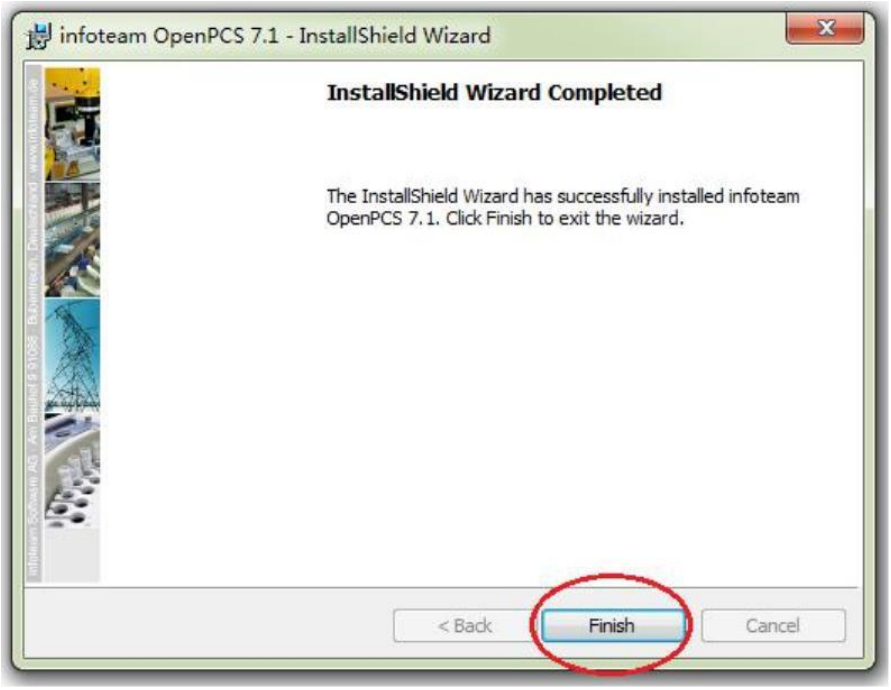


（8）选择安装路径，选好后点击下一步

（9）之后的步骤一直点击下一步即可，开始安装，并等待安装进度条，直到出现添加驱动，此处直接点击退出即可

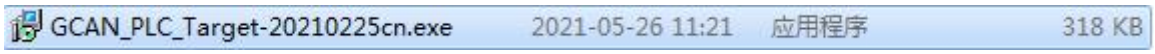


（10）安装完成



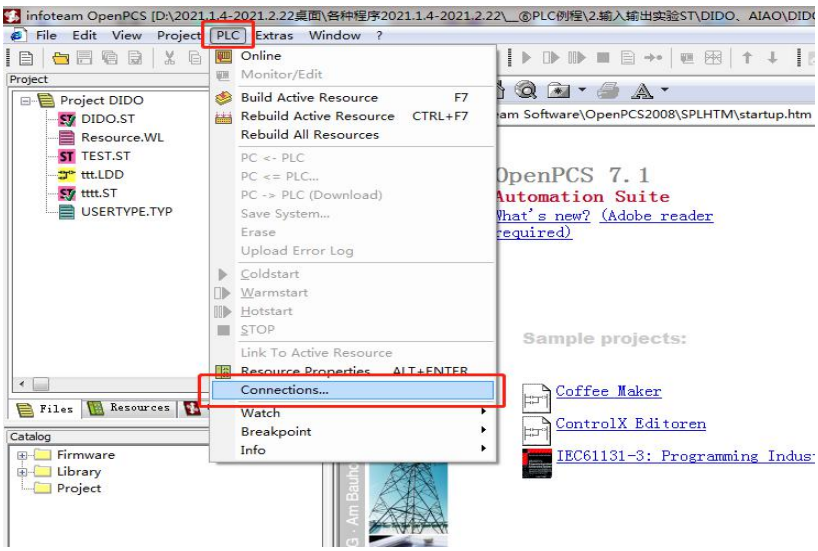
3.2 安装 TARGET 组件

打开网盘资料中的②号文件夹，TARGET 安装，选择中文版英文版均可，没有区别，只是安装向导的语言选择，选择里面的.exe 文件双击，并点击安装，安装过程只需几秒，几乎一闪而过。

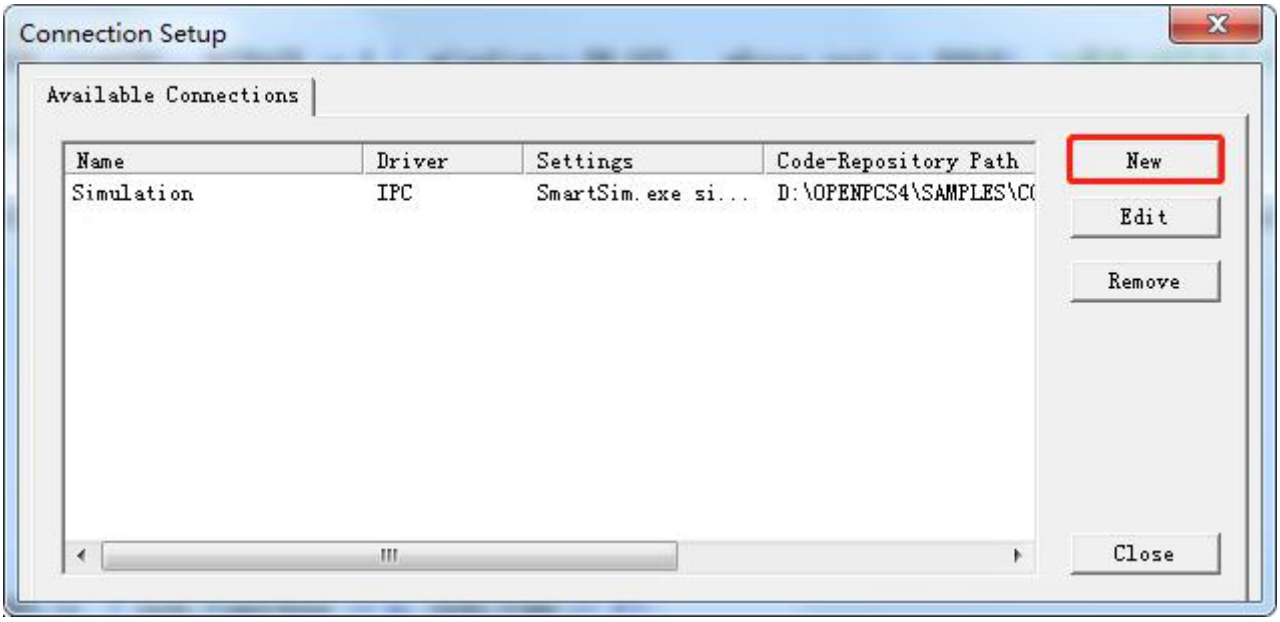


3.3 OpenPCS 联机配置

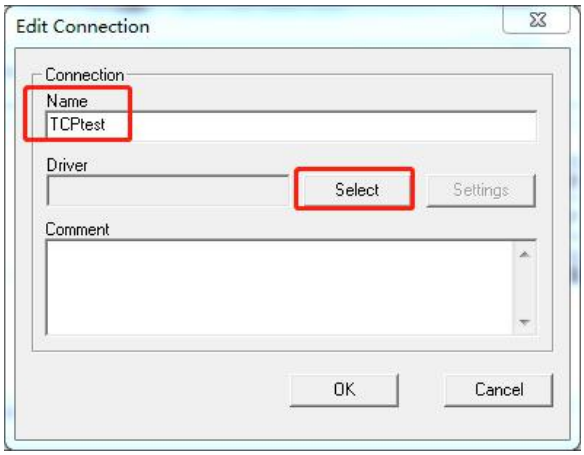
（1）选择 PLC 下拉列表的 connection 进入连接设置。



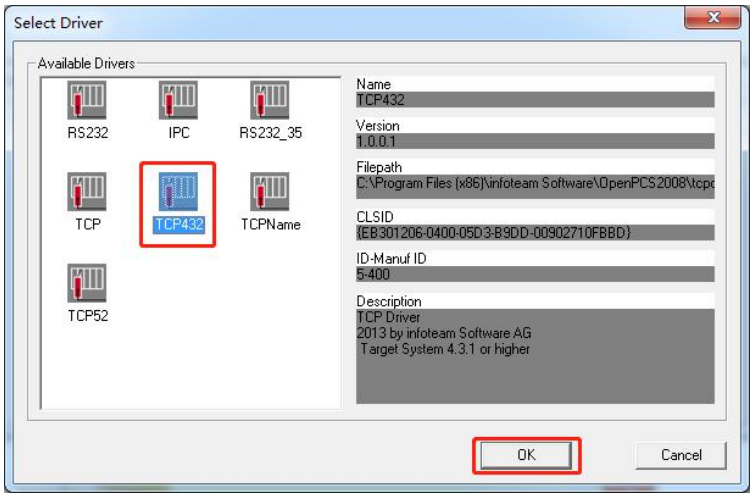
（2）点击 New 新建连接配置。



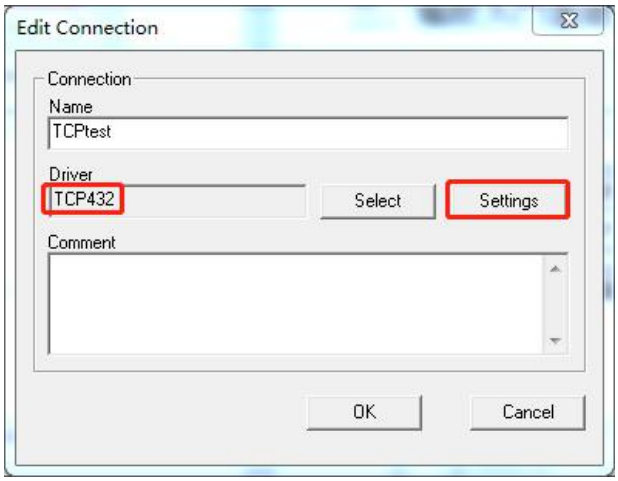
（3）为新建的连接配置命名，根据用户喜好命名，本文中命名为 TCPtest，填好命名后点击 Select 选择连接配置的驱动。



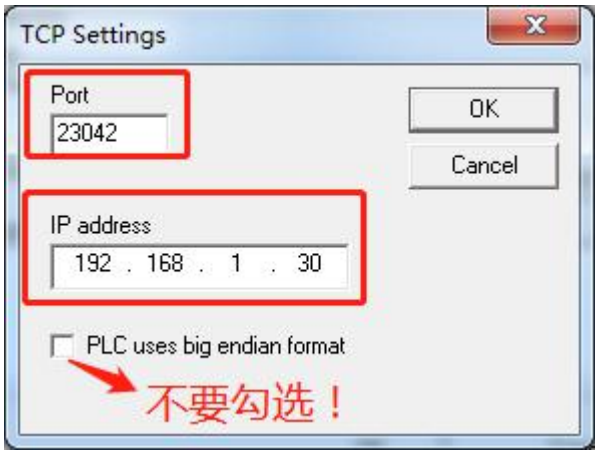
（4）在弹出的驱动选择框中选择 TCP432 驱动，并点击 OK。



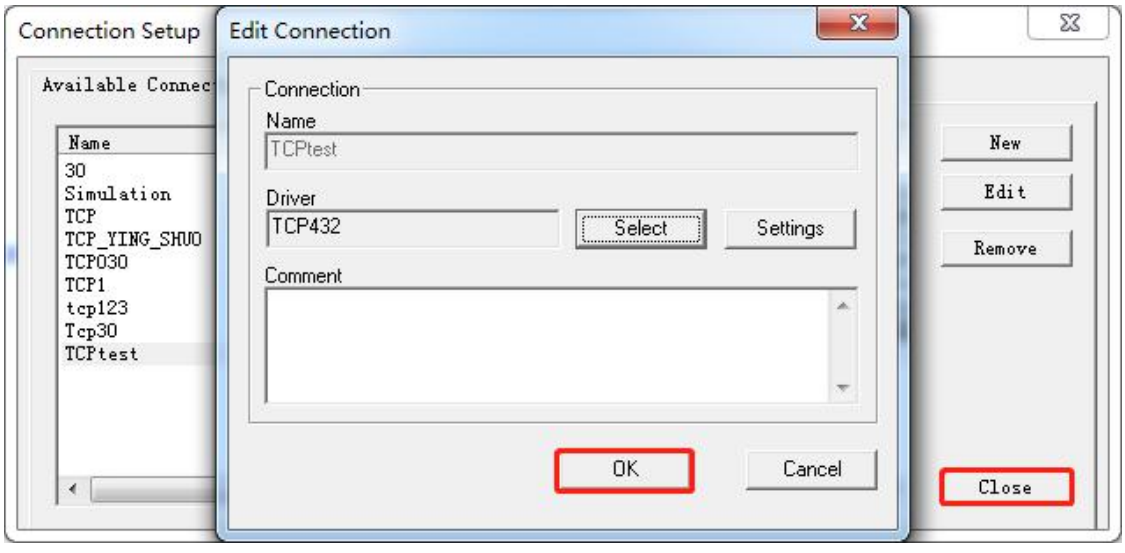
（5）选择好驱动后，Settings 按钮会变为可点状态，点击 Settings 按钮为 TCP432 驱动设置属性。



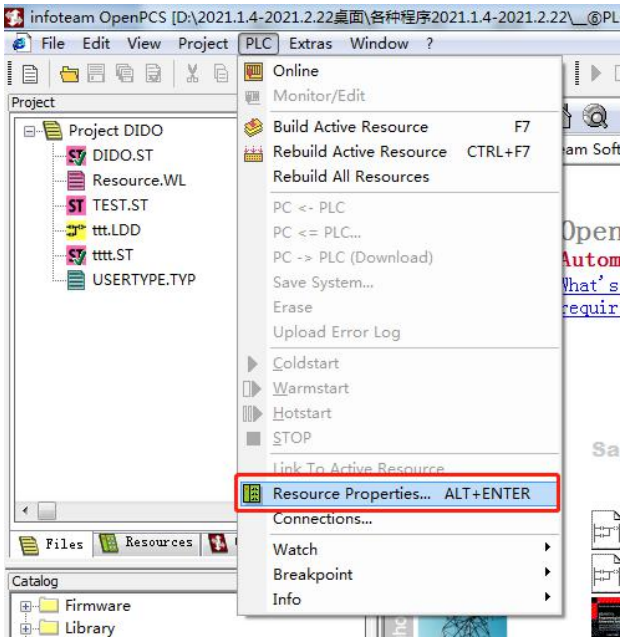
（6）设置端口号 Port 为 23042（*端口号为固定值不可更改*），IP 地址为 PLC 的设备 IP 地址（*GCANPLC 的出厂默认 IP 为 192.168.1.30，用户刚刚拿到设备时，设备 IP 即为默认 IP*）**注：下方 PLC 使用大端模式千万不要勾选**，设置好之后点击右侧 OK 按钮。



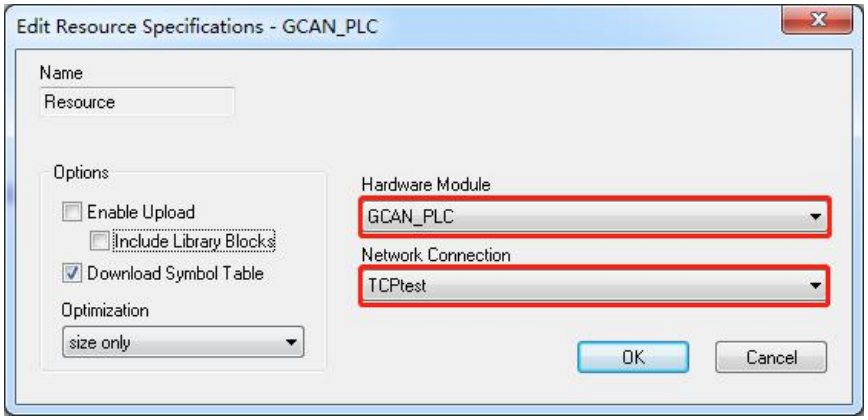
（7）连接设置编辑完毕后点击 OK，点击 OK 之后，编辑框消失，在连接设置点击关闭即可。



（8）点击菜单栏 PLC 下拉列表中的资源属性 Resource Properties。



（9）硬件模块选择 GCANPLC，网络连接选择刚刚设置的，本文中为 TCPtest，用户根据实际情况选择自己设置好的连接配置即可。



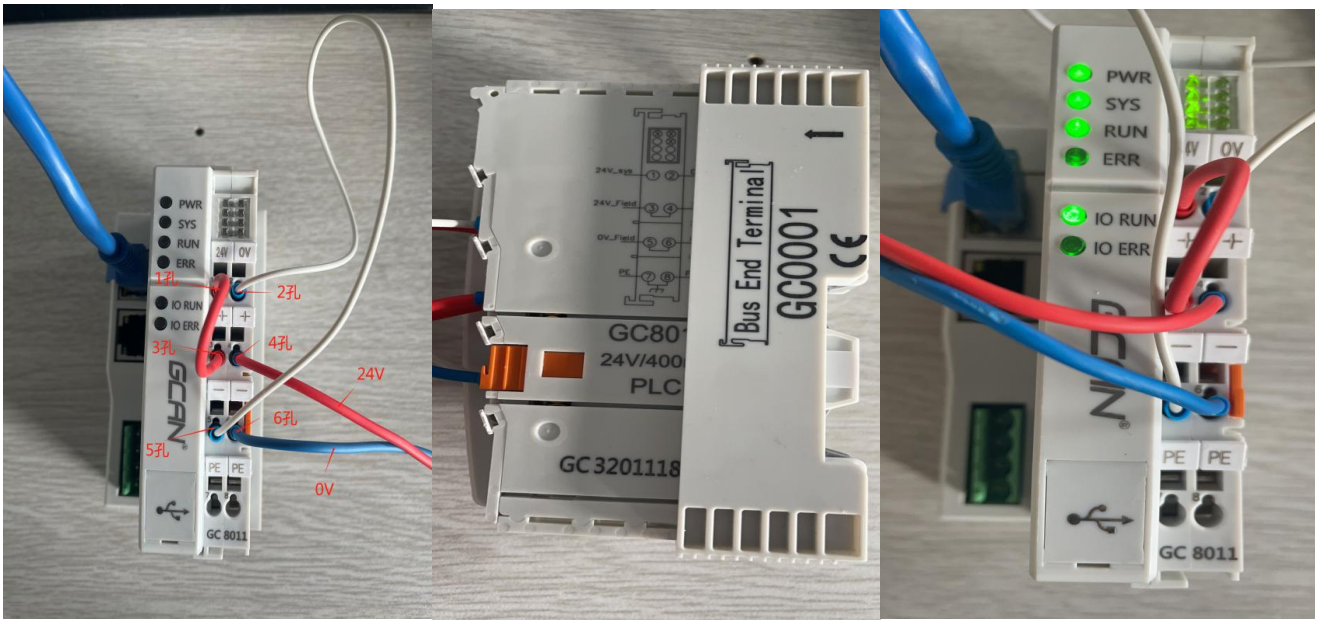
3.4 硬件连接、下载程序

3.4.1 设备接线

（1）24V 开关电源给 PLC 供电，PLC 供电共需两组，其中第一组为 1 孔接 24V，2 孔接 0V，为控制器供电；第二组为 3、4 孔接 24V，5、6 孔接 0V，为后接的 IO 模块供电，其中 3、4 孔内部短接，5、6 孔内部短接。

所以当接线时，将 1 孔与 3 孔连接，2 孔与 5 孔连接，4 孔接入电源 24V，6 孔接入电源 0V，

即可实现使用一个电源同时为控制器及 IO 模块供电。注：GC0001 应安装在最后一片 IO 模块后，若没有 IO 模块，直接安装在控制器上，形成回路，否则 IO ERR 红灯会常亮。

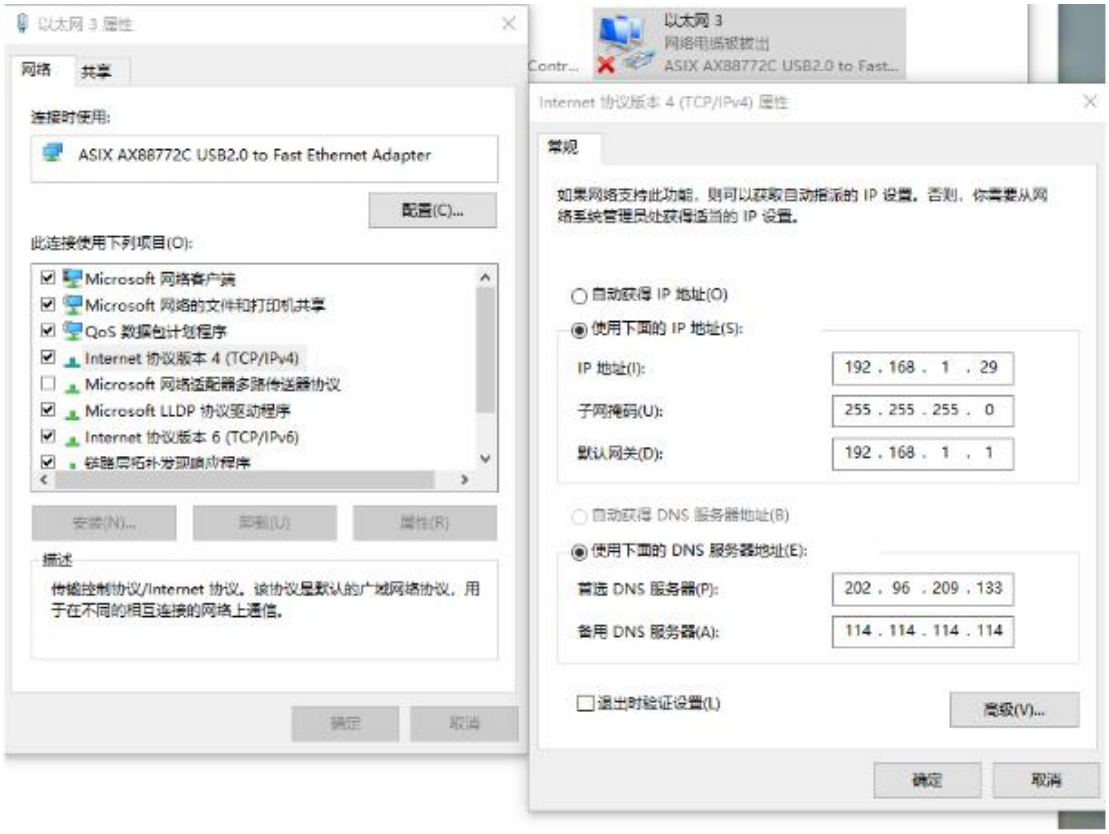


（2）使用网线连接电脑与 PLC，PLC 的 IP 地址为 192.168.1.30，需要修改电脑的 IP 地址与其前三段 IP 地址相同，即 192.168.1.XX，XX 为除 30 外的其他数，为 PLC 上电后，指示灯 PWR 常亮，SYS 慢闪，RUN 快闪，IO RUN 快闪，ERR 及 IO ERR 熄灭。

3.4.2 修改 PLC 与电脑网络连接的 IP 地址

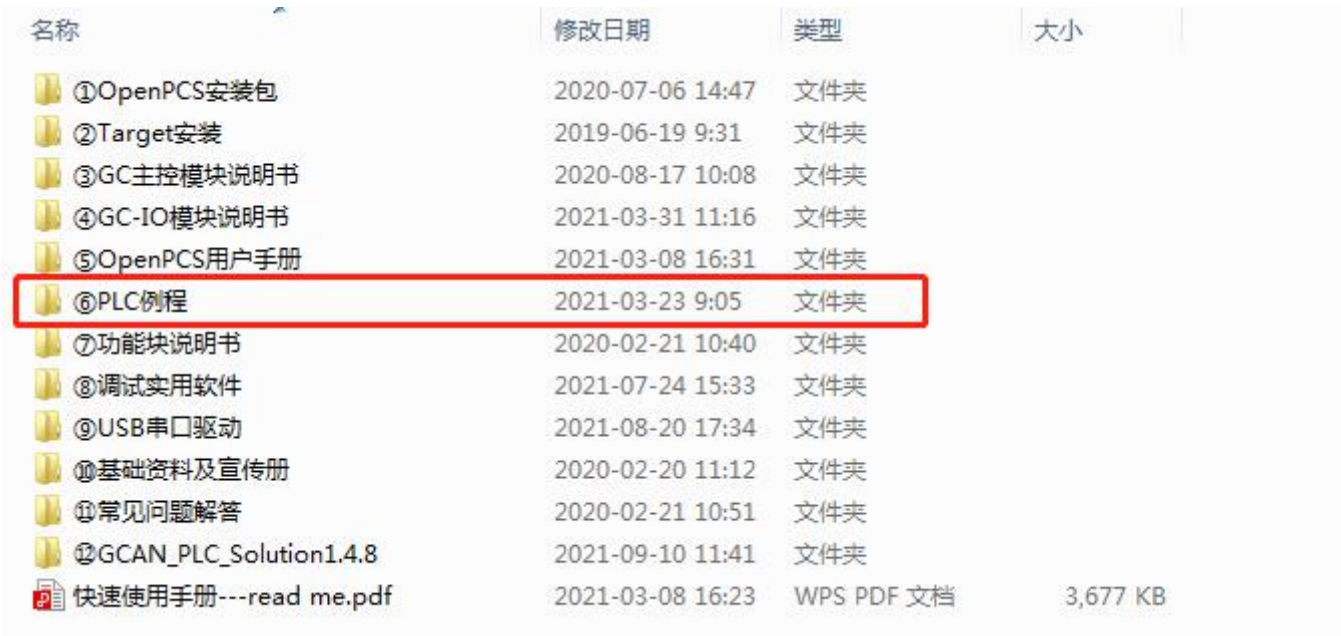
（1）打开网络与 Internet 设置，选择以太网选项

（2）点击右侧更改适配器选项，右击 PLC 与电脑之间的网络连接，点击属性，选择 Internet 协议版本 4（TCP4/IPv4），选中后点击属性，将 IP 设置为与 PLC 同一网段，如图。



3.4.3 下载程序

（1）打开网盘文件中的⑥号文件夹 PLC 例程。

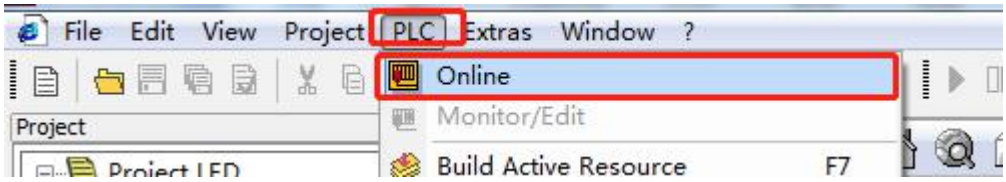


（2）打开第一个，跑马灯实验例程，打开文件夹中的 LED 文件夹。

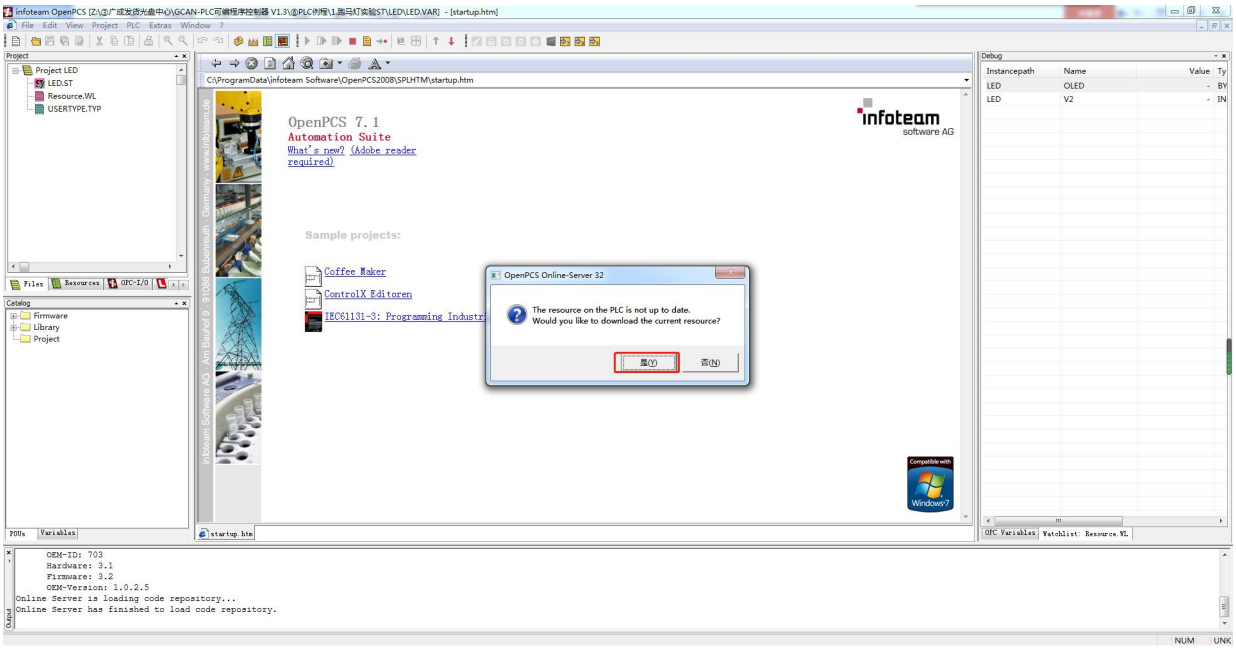
（3）双击.VAR 程序文件，打开程序

（*注：打开程序后，一定要检查 OpenPCS 的连接配置！这很重要！检查及设置步骤参照本文中第一章第三节 3.3 的 OpenPCS 连接配置！****）**

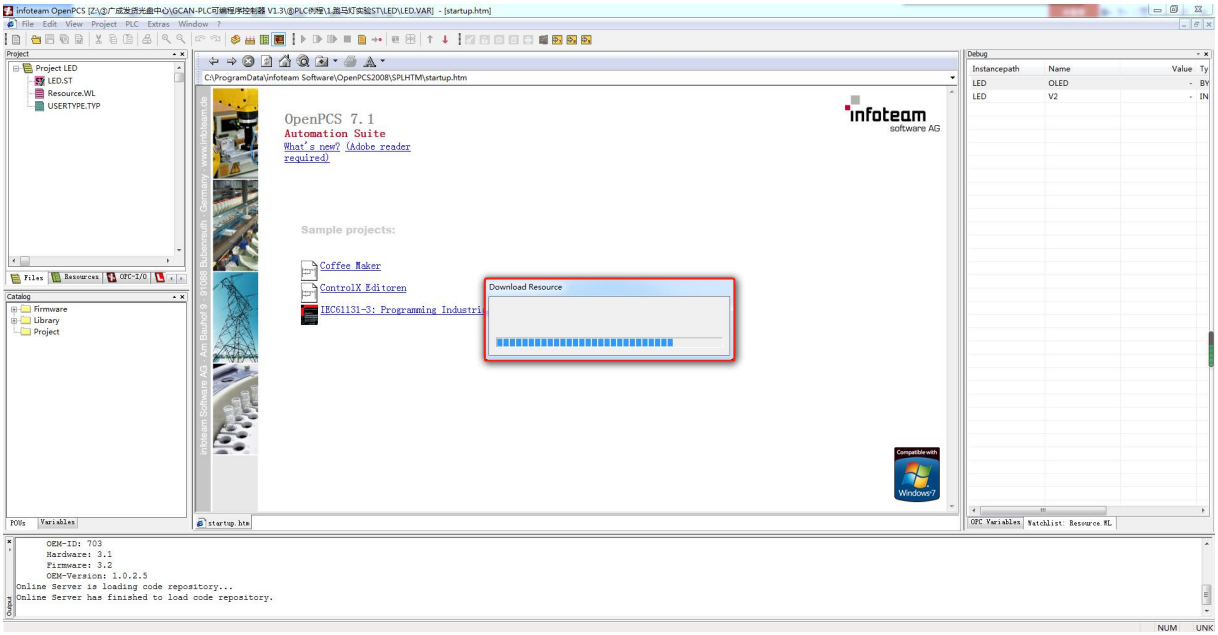
（4）打开程序后，点击菜单栏 PLC 下拉列表中的 Online



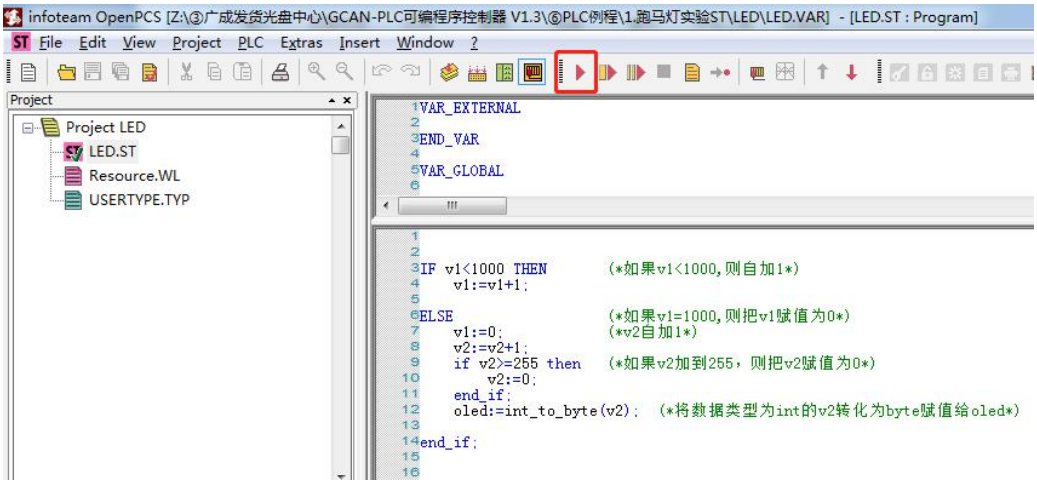
（5）下载程序前会弹出：PLC 中程序不是最新，是否下载当前程序的提示框，点击是。



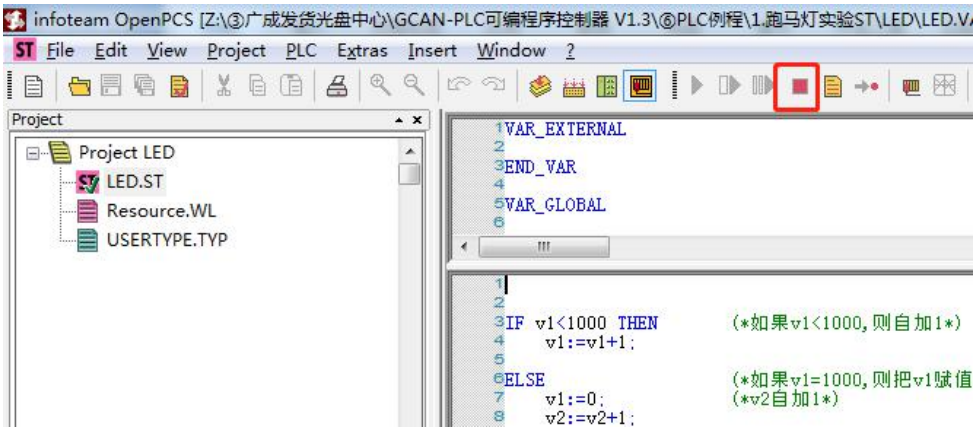
（6）等待下载进度条。



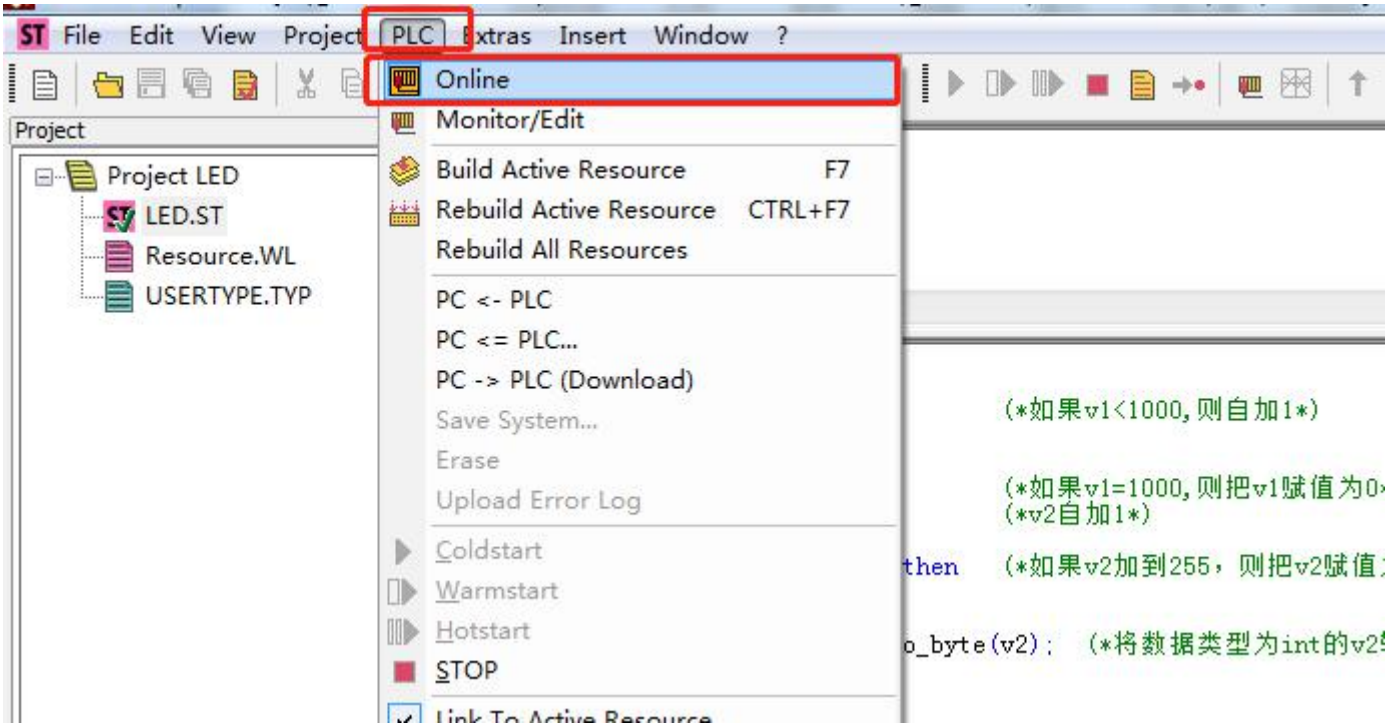
（7）下载完成后点击运行按钮运行程序。



（8）点击停止按钮可以停止程序运行。



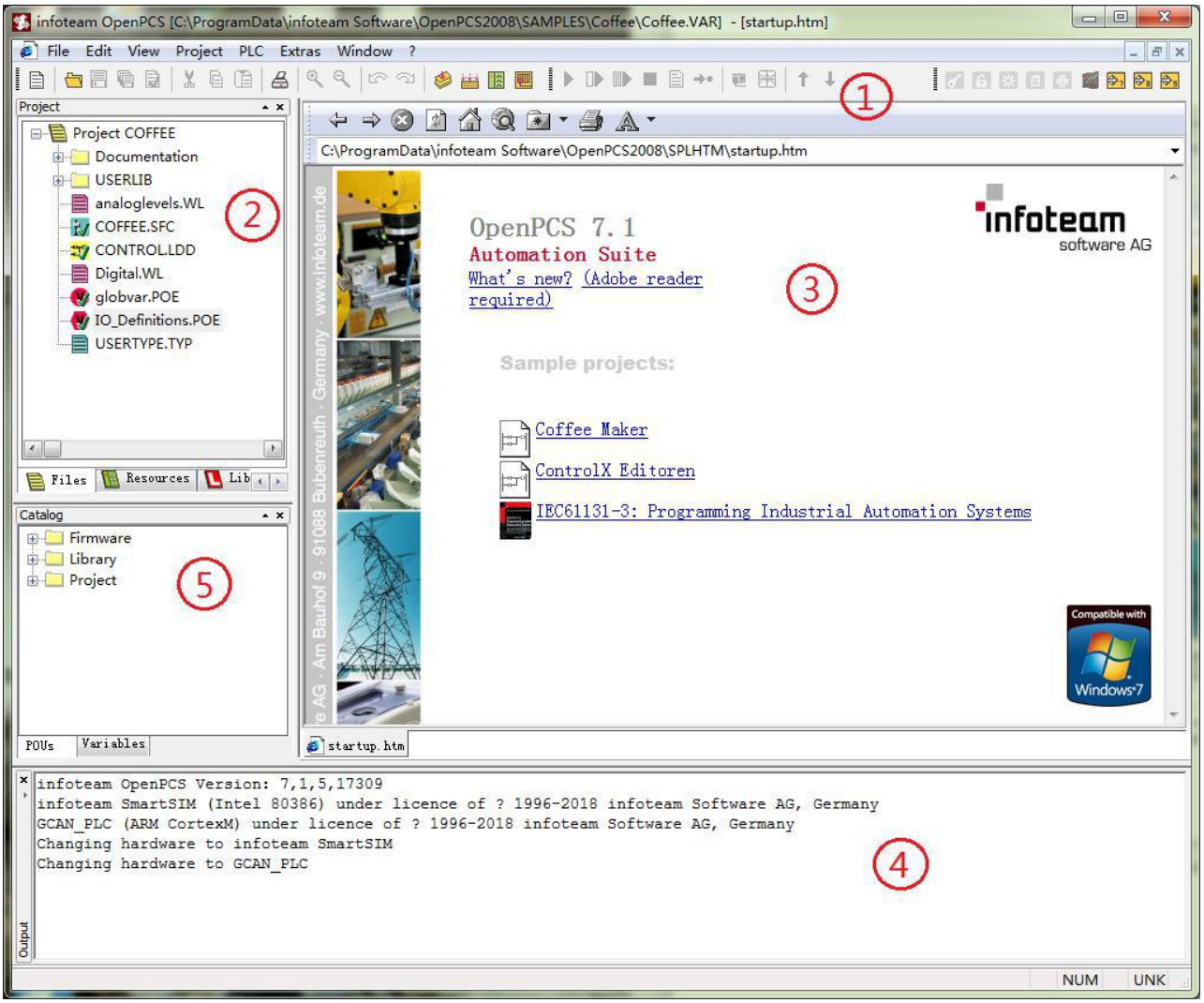
（9）再次点击 PLC 下拉列表中的 Online 可关闭联机状态，即 PLC 与电脑脱机。



二、编程入门基础

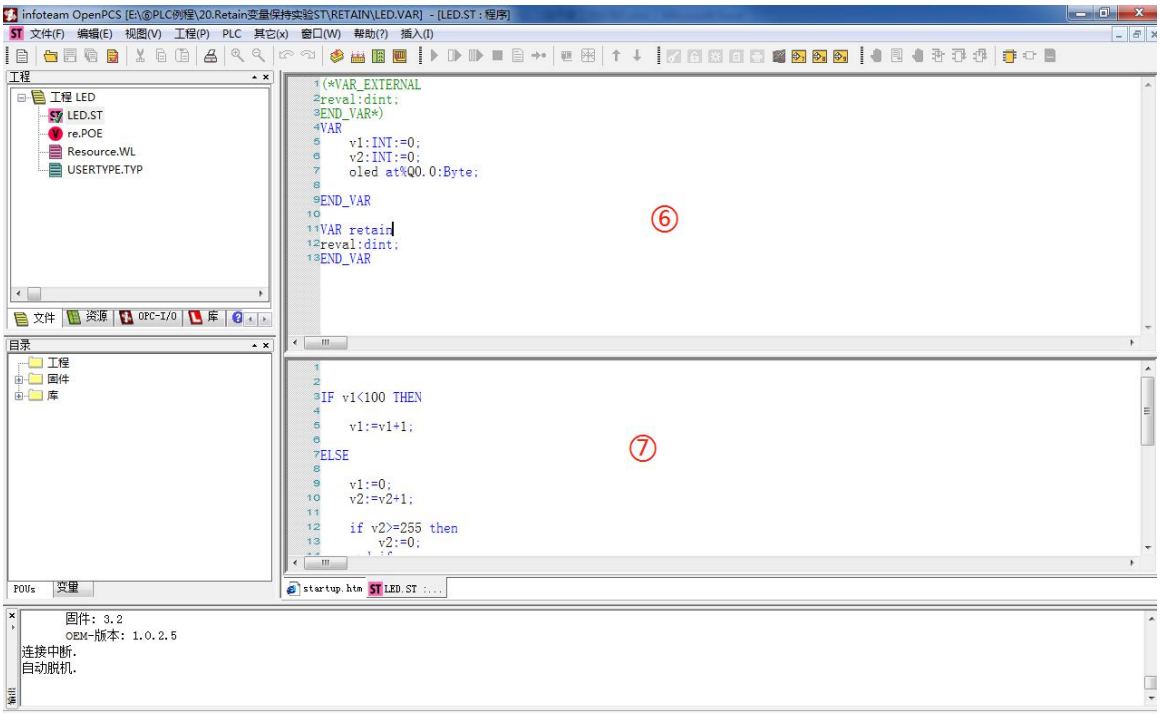
1. OpenPCS 功能简介

1.1 界面结构



①菜单栏 ②工程文件框 ③信息显示窗口 ④Debug 窗口 ⑤功能块窗口

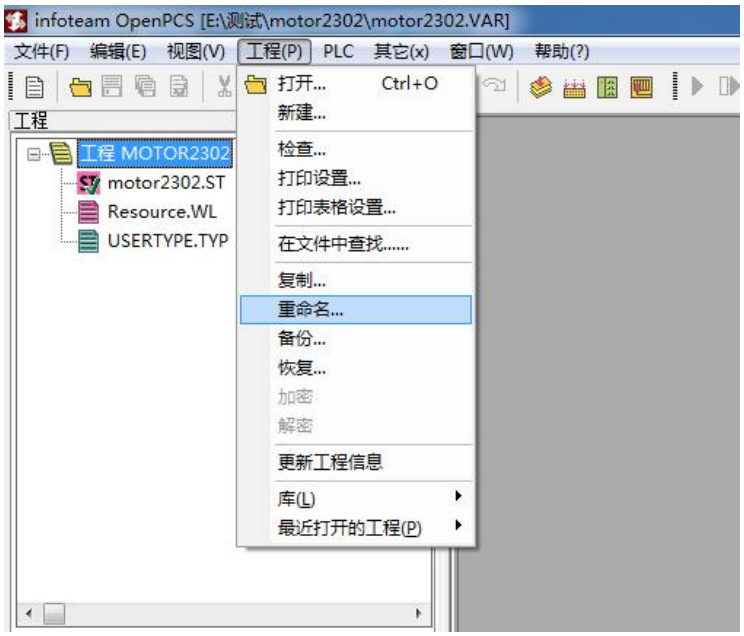
双击工程框中的程序文件，会显示如下窗口：



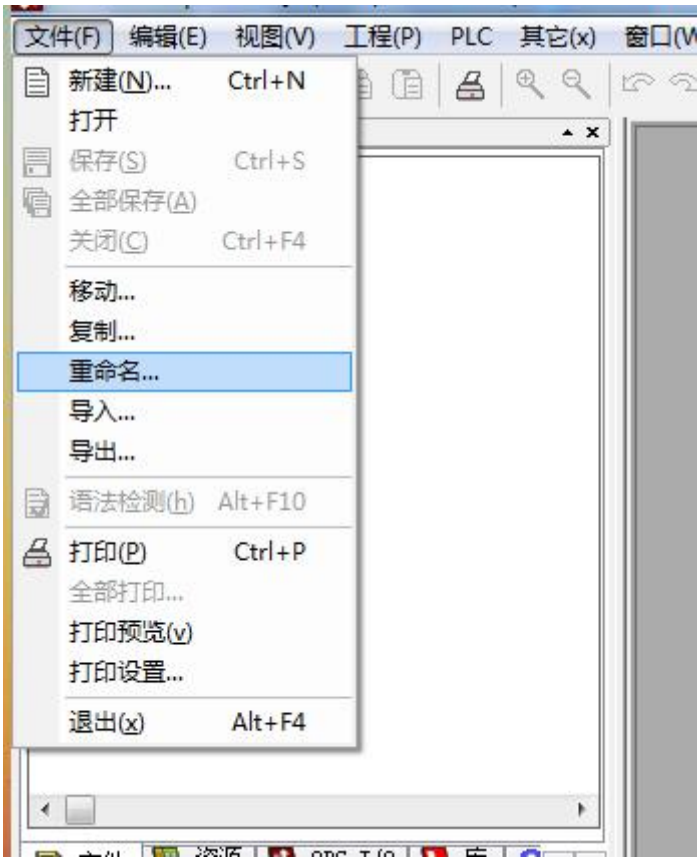
⑥变量声明区 ⑦程序编辑区

1.2 软件基本功能

（1）工程文件重命名：菜单栏-工程-重命名。

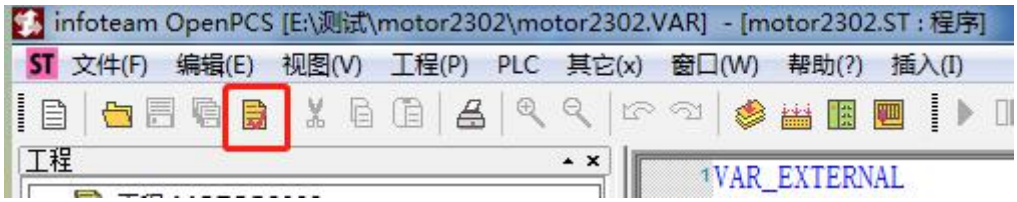


（2）程序文件重命名：选中需要重命名的文件，点击菜单栏-文件-重命名。



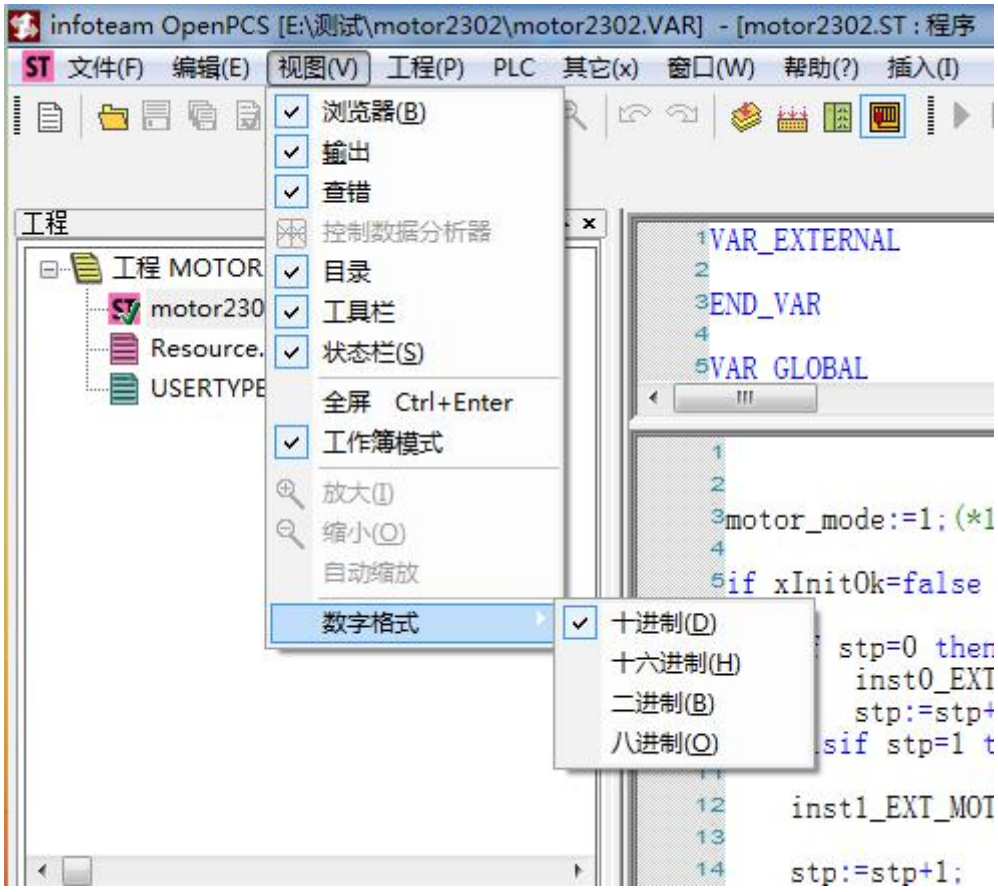
（3）删除工程文件：选中需要删除的文件，点击键盘上的 delete 键。

（4）语法检查：点击这个图标。语法检查的目的是检查程序编写有没有语法的错误。

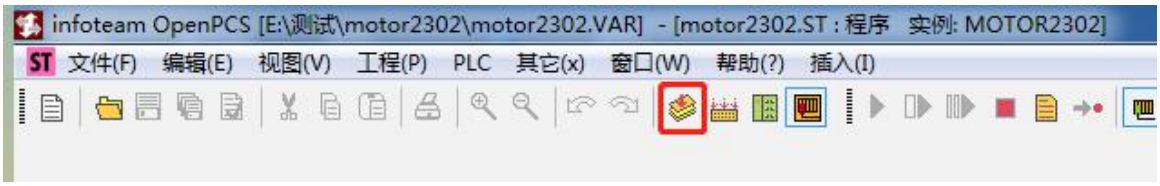


（5）添加变量到变量监测区：在工程框中点击资源，选择要监测的变量并双击。

（6）调节变量监测区数据显示格式：菜单栏-视图-数字格式。



（7）编译：点击菜单栏中的编译按钮。

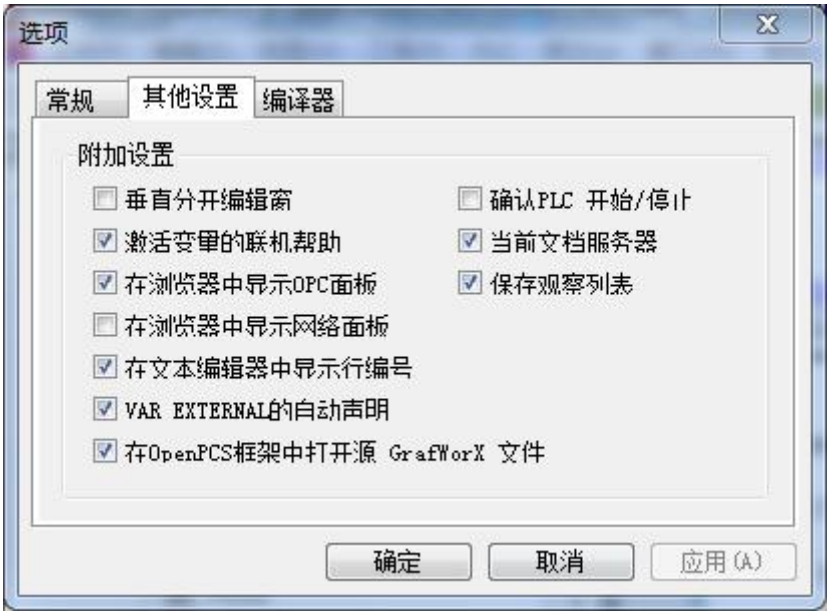


（8）重新生成当前资源：点击菜单栏中的重新生成按钮。

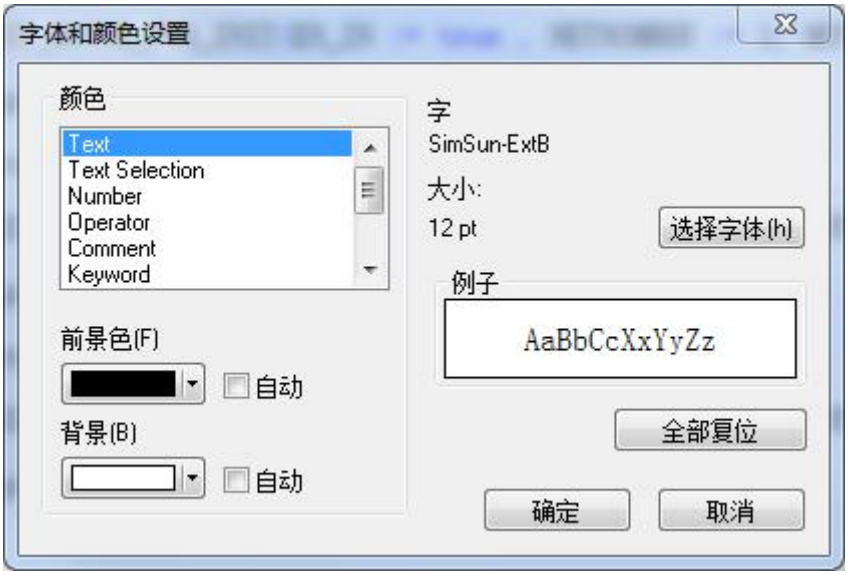
（9）断点调试：①设置断点 ②拨动断点 ③删除断点 ④步前进 ⑤步越过 ⑥步跳出。

注：设置断点的功能是让程序一行一行执行，方便找出问题所在。

（10）显示代码行数：菜单栏-其它-选项-浏览器，在弹出的窗口中选择其他设置，勾选“在文本编辑器中显示行编号”。然后点击应用，确定。



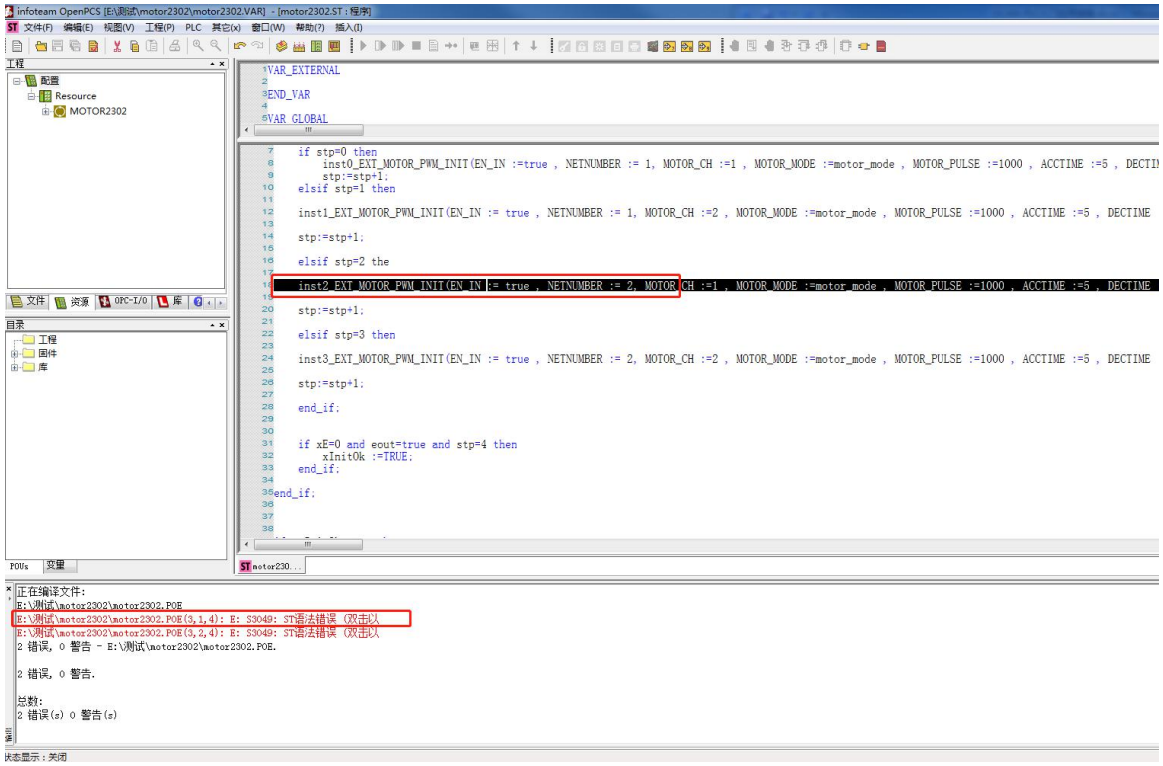
（11）调节代码字体、颜色、背景色：菜单栏-其它-字体/颜色。这里可以选择字体，字号，前景色，背景色。



（12）工程备份与恢复；1.菜单栏-工程-备份　　2.菜单栏-工程-恢复

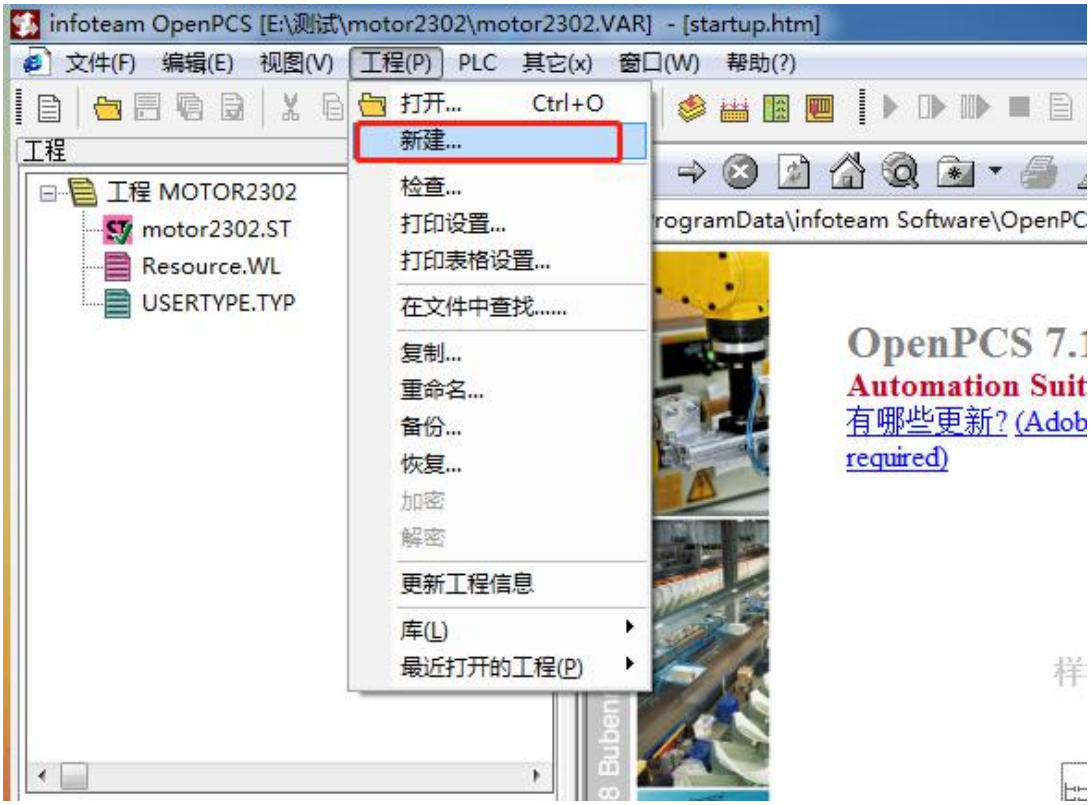
备份会将当前工程保存成一个.BAK 文件，恢复可以打开之前保存的.BAK 文件。这样可以在程序编写时随时备份，方便与最终版程序做对比。同时在给别人发送工程时，不必发送整个工程文件夹，只需要发送.BAK 文件即可。

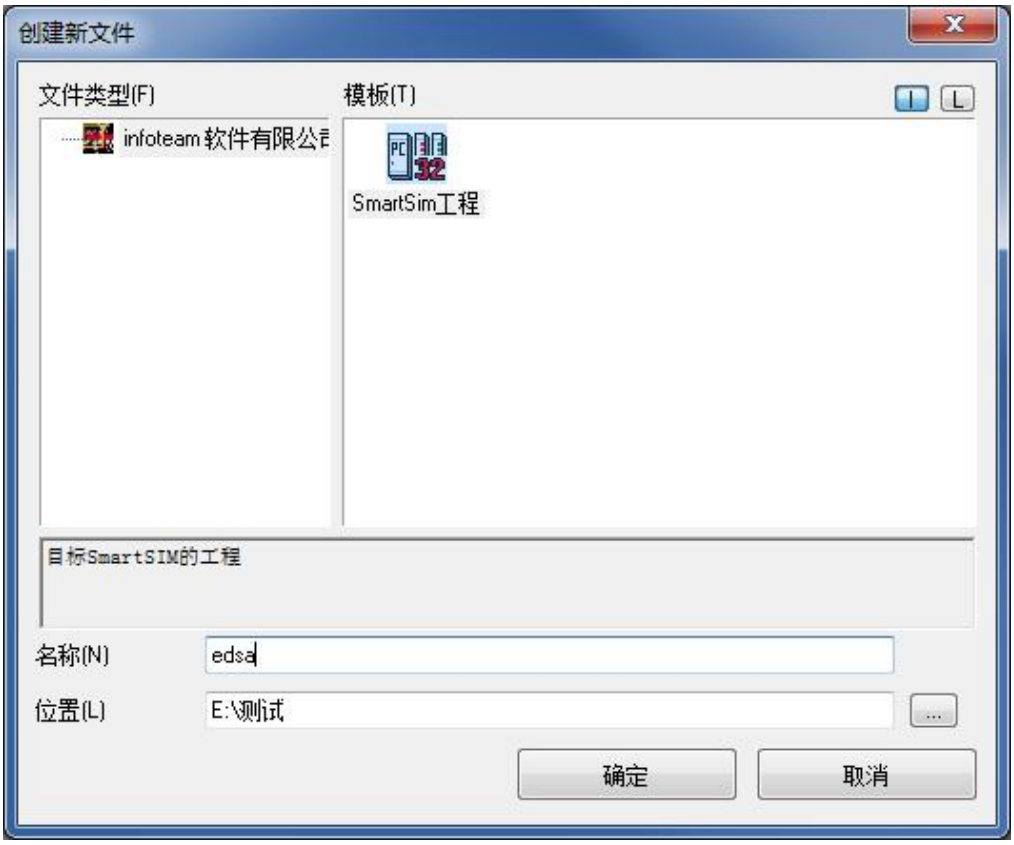
（13）语法报错检查：编译或语法检查报错时，双击报错信息，光标会跳到出错的地方。有可能是这一行，也有可能是前一行或者后一行。



1.3 新建工程

新建工程：点击菜单栏-工程-新建-命名





1.4 新建文件

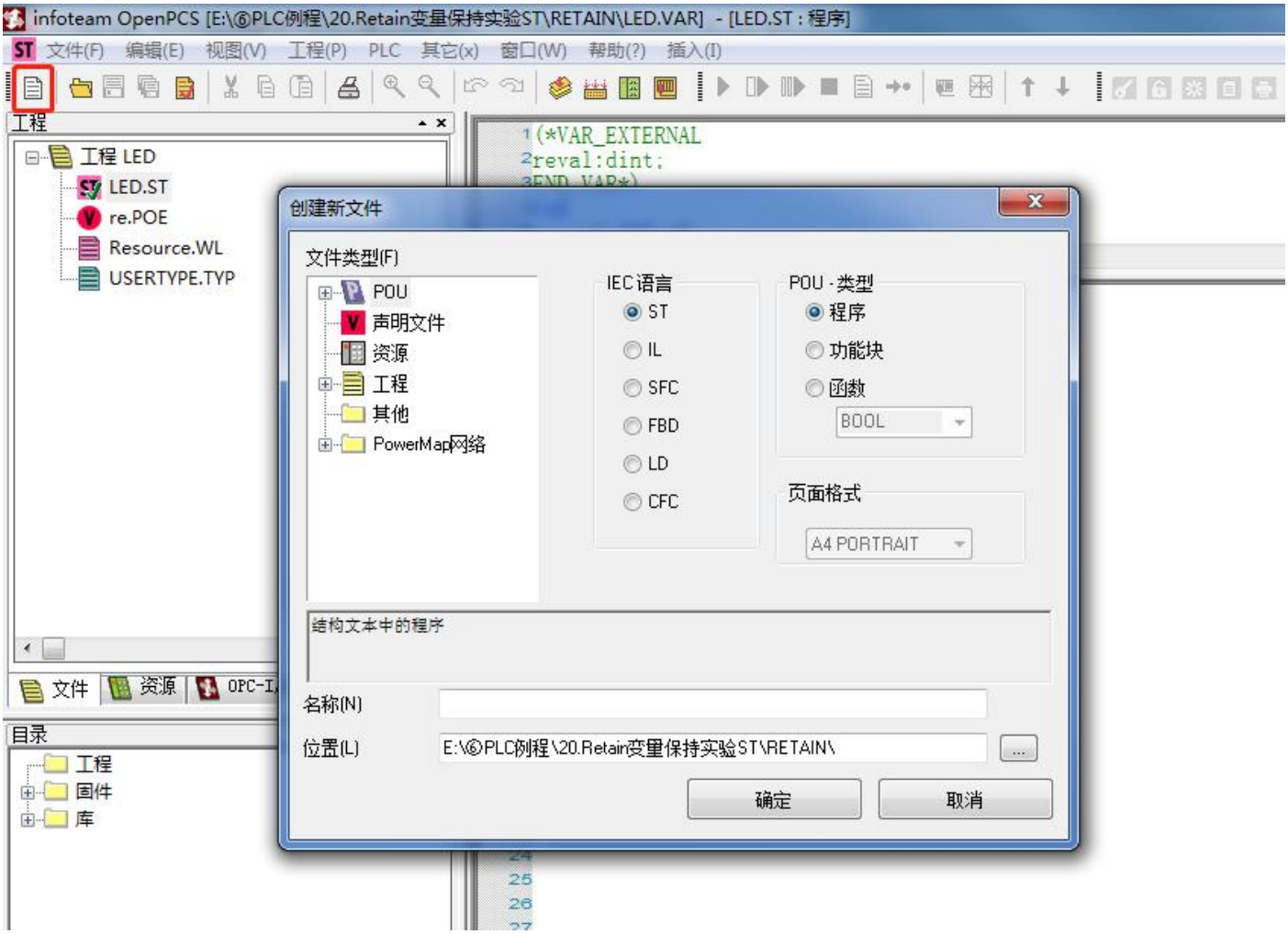
新建文件分为两种，一种是新建程序 POU，一种是新建声明文件。新建的程序文件可以选择的语言有 5 种，即为 IEC61131-3 的 5 种标准编程语言。可以选择的类型有两种：程序和功能块。

新建的声明文件分为全局变量和直接全局变量。当多个程序文件中公用相同变量，那么该变量需要在全局变量中声明；当具有直接物理地址的变量（AT%...）在当多个程序文件中公用时，那么该变量需要在直接全局变量中声明。

在命名时，文件的名称不能以数字开头，名称中不能包含中文字符。

1.4.1 新建程序

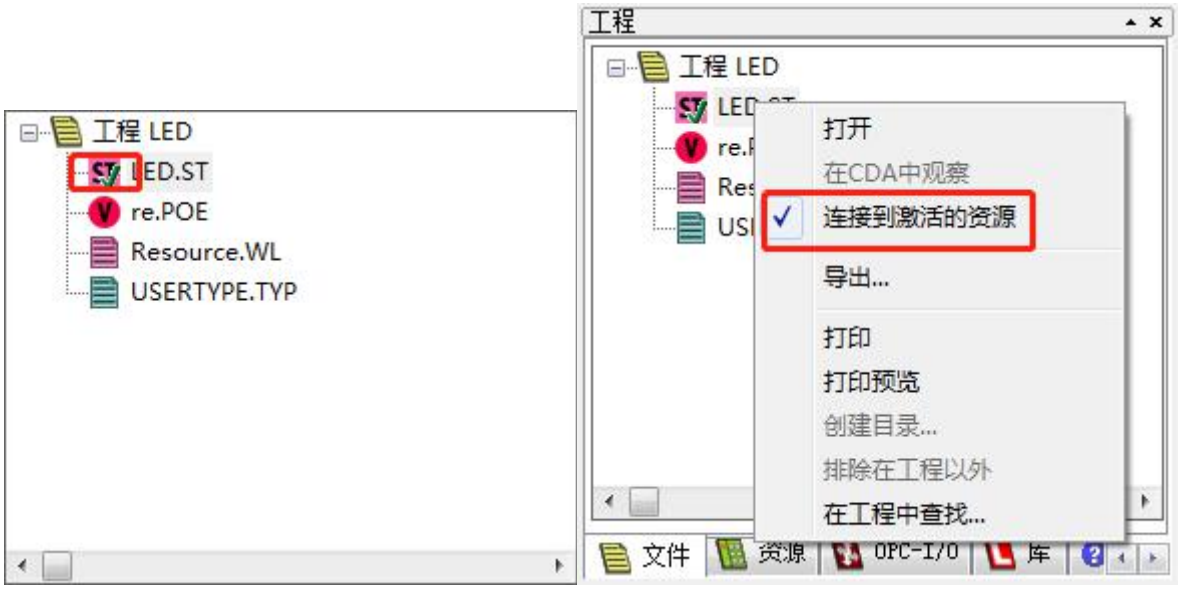
点击左上角白色纸页图标新建程序页，文件类型选择 POU，IEC 语言区选择程序页语言，POU 类型页选择程序类型，并为新程序页命名。



新建完成后点击确定，会出现如下对话框：

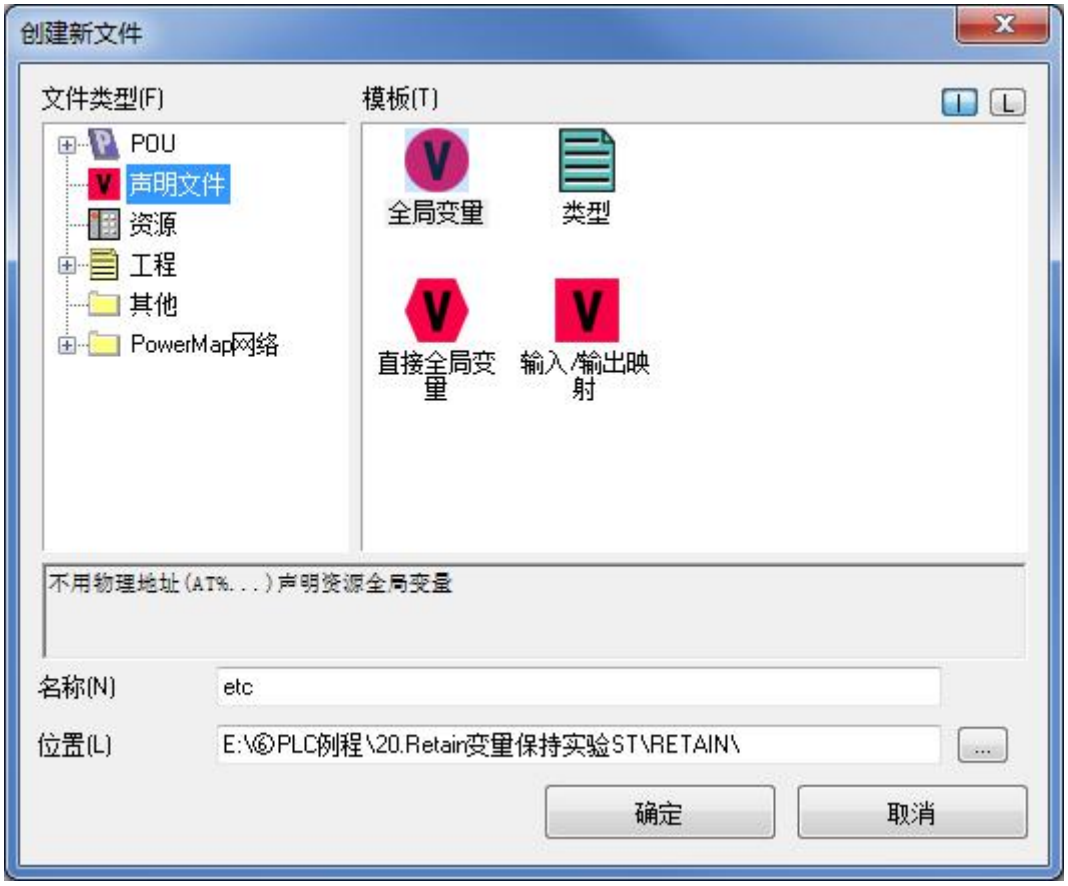
点击是，程序文件的图标上会出现绿色的对号，代表该程序文件已链接到资源。

同时，如果想取消链接到资源，就右键程序文件，取消链接到激活的资源，这样，程序文件就不会被编译和下载。



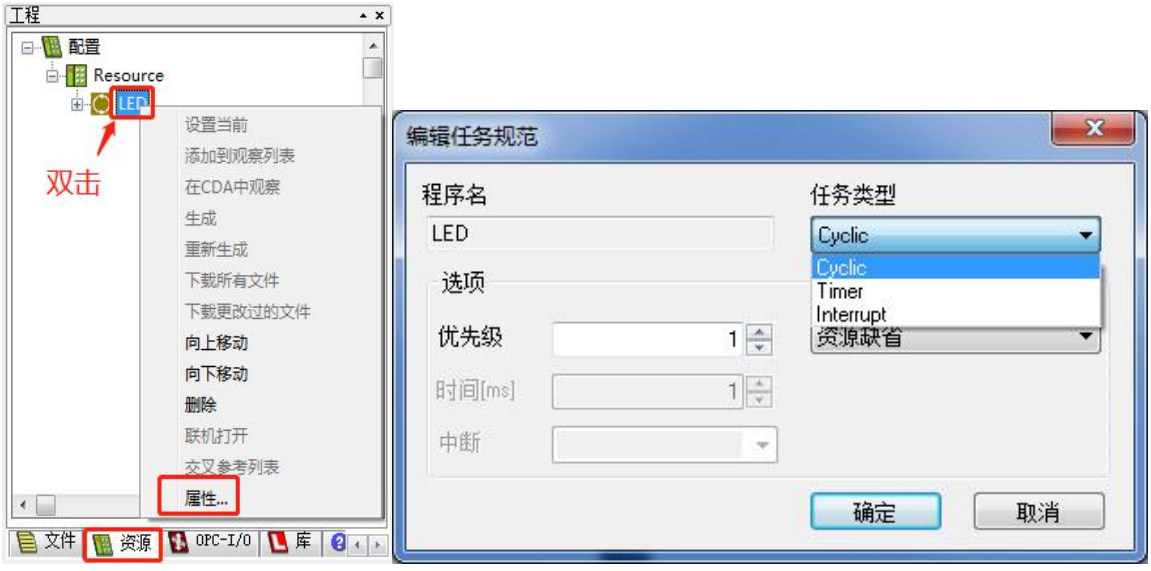
1.4.2 新建声明文件

点击左上角白色纸页图标新建程序页，文件类型选择声明文件，可建立全局变量声明页。



2. 简单编程语句入门

OpenPCS 一个工程下可以有多个程序文件，程序执行机制为同时执行。没有主程序子程序的概念。一个程序文件在 OpenPCS 中称为一个任务，任务类型分为 Cyclic 和 Timer。当任务类型为 Cyclic 时，代码 1000 行以下循环周期为 1ms；当任务类型为 Timer 时，循环时间可以自由设置，但不能低于程序可执行的最快循环周期，即 1ms。具体修改方式如下：



2.1 变量声明

注：OpenPCS 中所有使用的符号，必须为英文符号。

局部变量在

VAR

END_VAR

中声明。所有变量、基本指令、功能块均不区分大小写。

（1）中间变量声明：**变量名**：数据类型；　例：**GCANPLC**:BYTE;

（2）变量设置初始值：**变量名**：数据类型：=数值；　　例：**GCANPLC**:BYTE:=0;

（3）具有直接地址的变量声明：

变量名　AT%I0.0:数据类型；（代表变量的实际地址为 I0.0，I 代表输入）

例：**GCANPLC** AT%I0.0:BYTE;

变量名　AT%Q0.0:数据类型；（代表变量的实际地址为 Q0.0，Q 代表输出）

例：**GCANPLC** AT%Q0.0:BYTE;

（4）数组声明：

数组名 ：array[0..9] of byte；（代表数组中有 10 个元素 0~9，数组中元素的类型为 byte）

例：**GCANPLC**:array[0..9] of BYTE;　　第一个元素为 GCANPLC[0]。

（5）功能块声明：

自定义名：功能块名；（一个功能块如果要在程序中使用多次，那么必须在声明区多次声明。）

例：**inst0_ton**:TON;　　**Inst1_ton**:TON;

数据类型表格见下表：

数据类型	含义	数据长度
BOOL	布尔	1 位
BYTE	字节	8 位
WORD	字	16 位
DWORD	双字	32 位
SINT	短整	8 位
USINT	无符号短整	8 位
INT	整型	16 位
UINT	无符号整型	16 位
DINT	双整	32 位
UDINT	无符号双整	32 位
REAL	实数	32 位
CHAR	字符	
STRING	字符串	
TIME	时间	
DATE_AND_TIME	时间与日期	

2.2 编程语法

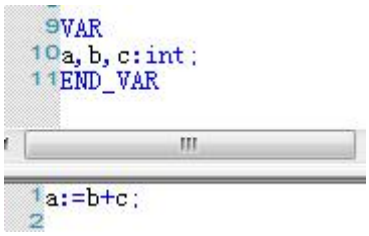
本节所有举例程序的语句并无任何实际含义及实际作用，仅为语句用法举例，如需使用语句可根据本节语法举例的用法自行编写程序。

2.2.1 数学运算符

:= 赋值运算 + 加运算 - 减运算 * 乘运算 / 除运算

例：a:=b+c; 意为将 b 加 c 的结果赋值给 a。

注：语句结束后需要加分号，且被运算的变量与被赋值的变量数据类型要一致（BYTE 无法直接进行运算，需转换成 INT、UINT、SINT、USINT 等数据类型）。



2.2.2 比较指令符

< 小于 > 大于 = 等于 <=小于等于 >=大于等于 <>不等于

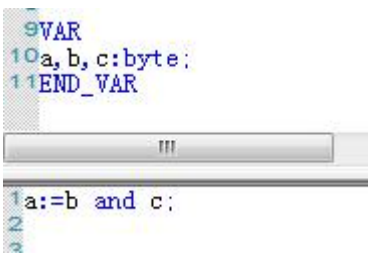
使用示例将在 2.2.4 中 IF 语句展示。

2.2.3 逻辑运算符

or 或 and、&与 not 非 orn 或非 andn 与非

例：a:=b and c; 意为将 b 和 c 的每一位做与运算后的结果赋值给 a。

注：语句结束后需要加分号，且被运算的变量与被赋值的变量数据类型要一致。

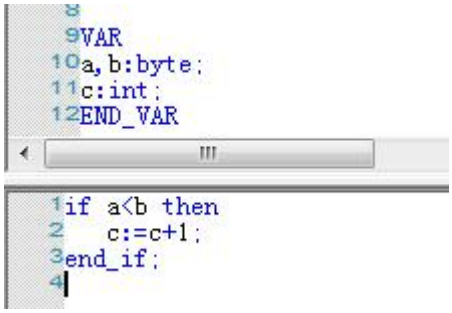


2.2.4 IF 语句

（1）IF XXX 条件 THEN 执行 XXX 命令 END_IF;

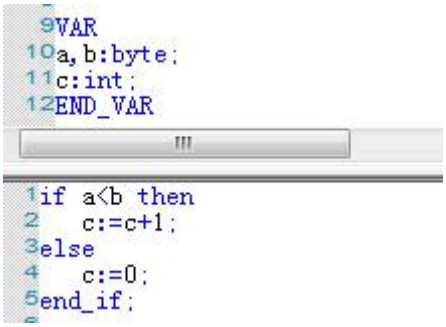
例：IF a<b THEN c:=c+1; END_IF; 意为如果 a 小于 b，则 c 执行自加 1 操作。

TIPS：BYTE 型变量虽然不可以做数学运算，但是可以做数学比较。



（2）IF XXX 条件 THEN 执行 XXX 命令 ELSE 执行 XXX 命令 END_IF;

例：IF a<b THEN c:=c+1; ELSE c:=0; END_IF; 意为如果 a 小于 b，则 c 执行自加 1 操作，否则执行将 c 赋值为 0 操作。



2.2.5 CASE 语句

CASE 语句可以根据被检查的变量的不同值，来选择执行不同的命令，尽管使用 IF 嵌套 ELSEIF 也可以实现，不过当被检查的变量可取值过多时，使用 CASE 语句将更加简洁方便。

CASE XX 变量 OF 1: 执行 XXX 命令; 2: 执行 XXX 命令 ..... N:执行 XXX 命令 END_CASE;

例：case a of 1: b:=a; 2: c:=a; 3: x:=a; end_case;

意为检查变量 a 的值，当 a=1 时，将 a 赋值给 b；当 a=2 时，将 a 赋值给 c；当 a 等于 3 时，将 a 赋值给 x。

```
VAR
a,b,c,x:byte;
END_VAR
```

```
case a of
1:
b:=a;
2:
c:=a;
3:
x:=a;
end_case;
```

2.2.6 功能块调用

本章 2.1 中介绍了如何声明功能块，编程时需要根据给功能块声明的命名来调用此功能块，例如：inst0_GPRS345G_STATUS(EN_IN := , NETNUMBER := | := STATUS, := EN_OUT);

其中括号外的 inst0_GPRS345G_STATUS 为声明时为功能块命的名；括号内，竖线 | 前面的为输入型变量，:=右面可以填数据或者变量；竖线 | 后面为输出型变量，:=左面必须填中间变量来承接功能块的输出值。具体功能块的类型及含义请参考第三章。

3. IO 模块组态简介

GCAN-PLC 系列产品的 IO 模块是自动组态方式，不需要手动组态，扩展 IO 插到 PLC 后面自动具有地址。只需要在程序声明部分创建具有实际地址的变量，那么此变量就可以表示实际的输入输出点。

例如：AI0 AT%I0.0:INT;

```
VAR
AI0 AT%I0.0:INT;
END_VAR
```

即代表 AI0 这个变量表示起始地址为 I0.0 的输入点，数据长度 16 位。GCAN 系列的模拟量 IO 每个通道所占的数据长度是 16 位，所以，AI0 正好可以表示模拟量的一个通道。

GCAN 系列扩展 IO 的地址与搭配到 PLC 后面的顺序有关。从最靠近 PLC 的 IO 开始，地址为 I0.0 或 Q0.0，后面的依次往后排。**输入不影响输出的地址，输出不影响输入的地址**。

例如，GCAN-PLC 后面搭配的 IO 依次为 1008，2008，3654，4652。那么这 4 个模块的首地址分别为 I0.0，Q0.0，I1.0，Q1.0。其中 1008 的 8 个通道地址为 I0.0~I0.7，可定义为 DI0 AT%I0.0:BOOL; 至 I0.7:BOOL;，或 DI1008 AT%I0.0:BYTE;，整片 1008 长度为 1 字节，所以 3654 的第一通道首地址为 I1.0，范围为 I1.0~I3.0，以此类推；输出同理，只是将 I 换为 Q。

各个模块通道数及每个通道所占长度见下表，方便计算 IO 地址。

产品信号	功能	通道数	每个通道所占数据长度
GC1008	数字量输入（PNP）	8	1 位
GC1018	数字量输入（NPN）	8	1 位
GC2008	数字量输出（PNP）	8	1 位
GC2018	数字量输出（NPN）	8	1 位
GC1502	脉冲计数器输入	2	计数 32 位，频率 32 位
GC2302	脉冲输出（5V）	2	速度 16 位，位置 32 位
GC2204	继电器输出	4	1 位（1 个 2204 模块占 8 位，后 4 位占空）
GC3604	模拟量输入（-5V~+5V）	4	16 位
GC3624	模拟量输入（-10V~+10V）	4	16 位
GC3644	模拟量输入（0~20mA）	4	16 位
GC3654	模拟量输入（4~20mA）	4	16 位
GC3664	模拟量输入（0V~+5V）	4	16 位
GC3674	模拟量输入（0V~+10V）	4	16 位
GC3804	PT100 温度采集（两线制）	4	16 位
GC3844	K 型热电偶采集	4	16 位
GC3822	PT100 温度采集（三线制）	2	16 位
GC3854	J 型热电偶采集	4	16 位
GC3864	T 型热电偶采集	4	16 位
GC4602	模拟量输出（-5V~+5V）	2	16 位
GC4622	模拟量输出（-10V~+10V）	2	16 位
GC4642	模拟量输出（0~20mA）	2	16 位
GC4652	模拟量输出（4~20mA）	2	16 位
GC4662	模拟量输出（0V~+5V）	2	16 位
GC4672	模拟量输出（0V~+10V）	2	16 位
GC4674	模拟量输出（0V~+10V）	4	12 位（每个通道 16 位，高 4 位占空）

三、功能块使用入门

1. 基本指令集

基础功能块索引表

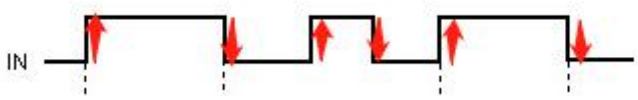
名称	功能	章节索引
CTU	以加法计算输入 CD 的脉冲数	3.1.1
CTD	以减法计算输入 CU 的脉冲数	3.1.2
CTUD	功能块 CTUD 计数上升沿和下降沿脉冲。	3.1.3
TON	计时功能块	3.1.4
DT_CLOCK	时钟功能块	3.1.5
F_TRIG	功能块 F_TRIG 检测输入操作 CLK 下降沿的状态。	3.1.6
R_TRIG	功能块 R_TRIG 检测输入操作 CLK 上升沿的状态变化。	3.1.7
SR	功能块 SR 静态转换一个数据元素（输出 Q1）为布尔值 1 或 0。	3.1.8
RS	功能块 RS 动态切换一个数据元素（输出 Q1）为布尔值 1 或 0。	3.1.9
PID	PID 控制	3.1.10
TOF	计时功能块	3.1.11
TP	计时功能块	3.1.12
CONCAT_STRING	两字符串串连	3.1.13
MOD	取余	3.1.14

基础函数索引表

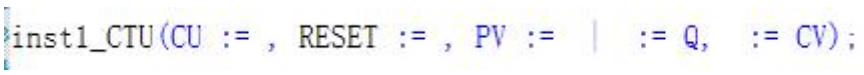
函数名	功能	章节索引
ABS（暂时无法使用）	绝对值	3.2.1
ACOS	反余弦	3.2.2
ADD	两数之和（ST 里用+）	3.2.3
ASIN	反正弦	3.2.4
ATAN	反正切	3.2.5
COS	余弦	3.2.6
DELETE	从 P 位置开始删除长度为 L 的子字符串	3.2.7
EQ（ST 中无法使用）	输入相等时为真	3.2.8
EXP	指数的幂函数	3.2.9
FIND	字符串查找	3.2.10
GE（ST 中无法使用）	比较函数（>=）	3.2.11
GT（ST 中无法使用）	比较函数（>）	3.2.12

INSERT	插入字符串	3.2.13
LE	比较函数（<=）	3.2.14
LT	比较函数（<）	3.2.15
LEFT	取字符串左边 L 个字符	3.2.16
LEN	字符串长度	3.2.17
LIMIT	限值	3.2.18
LN	以 e 为底的对数	3.2.19
LOG	以 10 为底的对数	3.2.20
MAX	所有输入值的最大值	3.2.21
MID（暂时无法使用）	取字符串中间的 N 字符	3.2.22
MIN	所有输入值的最小值	3.2.23
NE（ST 中无法使用）	不等(<>)	3.2.24
POINTER（数据类型）	指针	3.2.25
RIGHT	取字符串右边 L 个字符	3.2.26
ROL	字符串循环左移	3.2.27
ROR	字符串循环右移	3.2.28
SHL	字符串左移	3.2.29
SHR	字符串右移	3.2.30
SIN	正弦	3.2.31
SQRT	平方根	3.2.32
TAN	正切	3.2.33
XOR	异或	3.2.34

注：本章大量内容将涉及上升沿、下降沿，故在此统一讲解，上升沿为从 0 到 1 的过程，下降沿为从 1 到 0 的过程，某功能块为上升沿触发使能，说明该功能块的有效使能即不是 0，也不是 1，而是检测到使能位从 0 到 1 的过程。图中向上箭头为上升沿，向下箭头为下降沿。



3.1.1 CTU



功能块CTU以加法计算输入CU的脉冲数，初始化时，计数器置0；若RESET收到值1，计数值将重置；输入CU的每个上升沿使计数器加1；输出CV为当前计数值。若计数值小于PV上限值，输出Q将为布尔0，若大于或等于PV值，Q置1。

输入：

CU：布尔型计数器脉冲

RESET：布尔型重启计数器

PV：整型计数上限

输出：

Q：布尔型计数器是否到达上限值

CV：整型当前计数器值

3.1.2 CTD

```
inst0_CTD(CD := , LOAD := , PV :=  |  := Q,  := CV);
```

功能块CTD以减法计算输入CD的脉冲数。初始化时，计数器置0；若LOAD置1，PV值将为计数器初值。输入CD的每个上升沿将使计数器减1；输出CV为计数器当前值。若计数器值为正，输出Q将置0；若计数器值为0或负，输出Q置1。

输入：

CD：布尔型计数脉冲，上升沿

LOAD：布尔型设置条件

PV：整型初始值

输出：

Q：布尔型信号（计数值是否到0）

CV：整型计数值

3.1.3 CTUD

```
inst2_CTUD(CU := , CD := , RESET := , LOAD := , PV :=  |  := QU,  := QD,  := CV);
```

功能块CTUD计数上升沿和下降沿脉冲。初始化时，计数器置0。输入CD的每个上升沿；将使计数器加1；而CD的每个下降沿，计数器减1；若LOAD为1，PV值将预置到计数器；若RESET值为1，计

数器将重置；若计数值少于PV预置值，输出Q将显示布尔值0；若计数值到达或超过预置值，输出Q置1；若计数值为正，输出QD将为布尔值0；若计数值为0或负，输出QD将置1。

输入：

CU：布尔型计算上升、下降脉冲

CD：布尔型计算上升、下降脉冲

RESET：布尔型重置条件

LOAD：布尔型下载条件

PV：整型下载值

输出：

QU：布尔型信号（计数值是否达到PV值）

QD：布尔型信号（计数状态是否为0）

CV：整型计数状态

3.1.4 TON

```
inst3_TON(IN := , PT :=    := Q,   := ET);
```

输入IN有一个上升沿将启动定时器TON，延时时间为PT指定；计时器运行时，输出Q置值0，若时间到，Q状态变为1，且保持其值直到IN变为0；若计时器运行后PT值改变，它仅当产生下一个上升沿IN时有效；输出ET为当前定时器值。若时间到，ET保持其值，保持时间和输入IN值为1一样长；若IN变为0，ET值将为0；若输入IN为开，输出Q将经过指定的延时时间后开。

输入：

IN：开始条件

PT：time初始时间值

输出：

Q：bool计时器状态

ET： time当前时间值

3.1.5 DT_CLOCK

```
inst4_DT_CLOCK(SET_YEAR :=
               SET_MONTH :=
               SET_DAY :=
               SET_HOUR :=
               SET_MINUTE :=
               SET_SECOND :=
               SET :=

                    := YEAR,
                    := MONTH,
                    := DAY,
                    := HOUR,
                    := MINUTE,
                    := SECOND,
                    := RELTIME,
                    := ERROR);
```

此功能块为设置系统时钟功能块，SET_YEAR，SET_MONTH，SET_DAY，SET_HOUR，SET_MINUTE，SET_SECOND，分别为设置初始年，月，日，时，分，秒；SET为置位，将该参数置1再置0，该功能块生效；YEAR，MONTH，DAY，HOUR，MINUTE，SECOND分别为当前时间年，月，日，时，分，秒。RELTIME表示从计算机纪元（1970年1月1日0点）至当前时间的秒数。

ERROR表示错误信息的显示。

3.1.6 F_TRIG

```
inst5_F_TRIG(CLK :=    := Q);
```

功能块F_TRIG检测输入操作CLK的状态；通过输出Q来检测处理循环中状态由1变为0的变化；在一个处理周期内，即检测到CLK，并且下降沿指示的时候，输出为1。

输入：

CLK： 布尔型操作输入，仅对下降沿检测

输出：

Q： 布尔型操作输出，仅在CLK有下降沿时置位。

3.1.7 R_TRIG

```
inst6_R_TRIG(CLK :=  |  := Q);
```

功能块R_TRIG检测输入操作CLK的状态变化；若在处理循环中状态由0变被检测并通过输出Q为1来示意；仅当CLK状态在一个周期内的变化被检测到且产生了一个上升沿时，输出状态为1。

输入：

CLK：布尔型输入的上升沿有效

输出：

Q：布尔型输出;指示CLK上升沿

3.1.8 SR

```
inst1_SR(SET1 := , RESET :=  |  := Q1);
```

功能块SR静态转换一个数据元素（输出Q1）为布尔值1或0；0和1之间转换是根据布尔输入操作SET1和RESET；处理开始时，输出Q1初始化为0，若SET1值为1，功能块作用，输出Q1将置1；此后，SET1改变不再改变输出Q1；输入RESET为1时总是置Q1为0，如它可以复位；若两个输入都为值1，置位将起主导作用，也即Q1将置位。

输入：

Set1：bool设置条件

Reset：bool重置条件

输出：

Q1：bool双稳态元件的输出状态

3.1.9 RS

```
inst0_RS(SET :=  , RESET1 :=  |  := Q1);
```

功能块RS动态切换一个数据元素（输出Q1）为布尔值1或0。0和1之间切换是根据布尔输入操作

SET和RESET1；处理前，输出Q1初始化为0，若SET值为1，功能块作用，输出Q1将置1；此后，SET改变不改变输出Q1；输入RESET1为1时总是置Q1为0；如，它可以重置输出；若两个输入都为值1，重置将起主导作用。也即Q1将重置。

输入：

Set：bool设置条件

Reset1：bool重置条件

输出：

Q1：bool双稳态元件的输出状态

3.1.10 PID

```
inst2_PID1(ENABLE := , SPV := , APV := , KP := , KI := , KD :=  |  := READY,  := CO,  := ERROR);
```

该功能块用于实现PID控制，如果输入ENABLE处检测到上升沿，功能块接收控制参数KR，TO，TD，TI和BIAS并启动控制器。在第一次运算时，设定值和最后一次实际值被设为当前实际值。第一次运算校正变量通常会出现初始偏差值，因为比例增益、积分增益和微分增益第一次默认都为0。控制器的性能表现会受控制参数的影响，如果TI=t#0ms，控制器的积分增益不会被计算且赋值为0。如果TD=t#0ms，微分增益的值会被赋为0.如果参数KR的增益为0，比例增益就不再需要。因为KR增益与比例增益和微分增益都有关系，积分和微分增益这里被设置为1。P控制器：TI=TD=0，KR≠0，积分和微分增益为1。PI控制器：TD=0，KR，TI≠0。PID控制器：KR，TI，TD≠0。有很多种方法测定控制参数（拐点切线，震荡试验），参数也可以通过仿真工具进行确定。然而，在频率和相位响应方面，完整的控制系统被描述成一个模型是很必要的。控制系统的相关传递参数的模型可以从已知值和状态中建立并用来决定仿真过程中的参数。设定值、实际值、偏差值和校正变量被当做标准变量使用，这些标准化变量的值变化范围为0.0~1.0，在功能块开始执行时会检查偏差的有效范围，当偏差值超出有效范围时输出ERROR会报错。在程序运行过程中不会检查设定值和实际值。功能块限制了校正变量和积分和的值，如果计算结果为负值，变量值就会被设为0。如果值超过1，变量值就会被设为1.此外，积分和受校正变量的影响，也遵循以下规则：

如果校正变量的值大于1，则积分和的计算方式如下：

积分和=1-（比例增益+微分增益）

如果校正变量的值大于0，则积分和的计算方式如下：

积分增益=1-（比例增益+微分增益）

ENABLE：如果检测到输入上升沿，控制因数KR，T0，TI，TD会被系统接收，同时会计

算初值偏差的积分和。输出READY，ERROR和CO会通过令ENABLE=0来复位。功能块检查有效区域，如有必要，在参数T0和BIAS传输过程中在输出ERROR处显示溢出误差；

SPV：控制器标准设定值（控制变量），有效范围0.0~1.0，功能块会自动检测参数（但

不会检测溢出错误）

APV：控制器标准实际值（过程变量），有效范围0.0~1.0，控制器会自动检测参数（但

不会检测溢出错误）

KP：比例系数；

KI：积分系数；

KD：微分系数；

READY：PID控制器的输出状态TRUE=控制器的功能块已经完全参数化并已经进入预操作状态。FALSE=控制器的功能块未完全参数化或错误参数化（控制参数位于有效值范围外）,控制器未进入预操作模式。

CO：控制器的输出，计算控制器的校正变量（值的范围0.0~1.0）

ERROR：错误代码表示功能块执行结果的信息，可能的错误代码定义如下：

0：在功能块执行过程中未发生错误

8：指定的参数BIAS的值是无效的（小于0或大于1）

16：指定的参数T0的值是无效的（时间等于0）

3.1.11 TOF

```
inst3_TOF(IN := , PT :=   | := Q,  := ET);
```

若输入状态IN为1，它将无延时的转到输出Q；若有下降沿来临，一个计时函数将启动，延时时

间有PT决定；等到计时溢出时，操作数Q变为状态0；若PT值在启动后改变，它会在下一个上升沿IN来临后才作用；ET包含当前时间值，若时间到，ET将保持其值，直到IN值为0。若IN状态变为1。ET值也变为0;若输入IN关，输出Q在经过指定的一段延时后也将关闭。

输入：

IN：开始条件

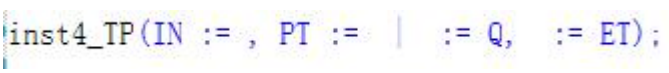
PT：time初始时间值

输出：

Q：bool计时器状态

ET：time当前时间值

3.1.12 TP



输入IN有一个上升沿将启动时间函数TP，运行时间由PT决定；计时运行时，输出Q置值为1，输入IN的任何变化将无效；若启动后，PT值改变将到下一个上升沿IN产生发生作用；输出ET为当前时间值。若时间到IN值为1后，ET将保持其值；任何上升（下降）沿产生而计时器不运行，在输出端Q将产生一个指令时间脉冲。

输入：

IN： 开始条件

PT：time 初始时间值

输出 ：

Q：bool 计时器状态

ET：time 当前时间值

3.1.13 CONCAT_STRING

```
ADC:=inst1_CONCAT_STRING(EN := , IN1 := , IN2 :=  | := ENO,  := OUT);
```

字符串IN1和IN2串连起来，成为新字符串，且装载到工作寄存器。字符串IN1和IN2从左到右顺序按升序连接起来。

输入：

In1：STRING第一个字符串

In2：STRING第二个字符串

返回：

OUT：STRING两字符串串连

3.1.14 MOD

```
inst0_MOD_USINT(EN := 1, IN1 :=122 , IN2 :=3  | ENEN := ENO, YUE := OUT);
```

MOD为取余，如图，122除以3余数为2，yue的结果为2。

3.2.1 ABS

```
RI:=abs(YUE);
```

如图，RI为YUE的绝对值。其中，RI与YUE的数据类型必须相同。

3.2.2 ACOS

```
mama:=acos(mc);
```

如图，mama为mc的反余弦值。其中，mama与mc的数据类型必须为real。

3.2.3 ADD

两数之和，相当于+

3.2.4 ASIN

```
MC:=asin(6.0);
```

反正弦：如图，其中MC与6.0必须是real类型。

3.2.5 ATAN

```
MC:=atan(6.0);
```

反正切：如图，其中MC与6.0必须是real类型。

3.2.6 COS

```
MC:=COS(6.0);
```

余弦：如图，其中MC与6.0必须是real类型。

3.2.7 DELETE

```
ADC:=DELETE(MB,1,2);
```

输入：

IN1：STRING部分需要删除的基本字符串

L：ANY_INT(根据支持)需删除部分子字符串的长度。L<0无效。

P：ANY_INT(根据支持)子字符串起始位置。P<0无效。

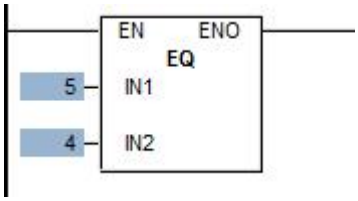
返回：

STRING缩短的字符串。IN1参数无效。

DELETE函数在给定的字符串IN1中从P位置开始删除长度为L的子字符串。

如图，删除字符串MB从第二个字符开始，长度为1的字符。

3.2.7 EQ(ST 中无法使用)



如图，如果IN1=IN2则输出1，否则输出0。

3.2.9 EXP

```
MC:=exp(6.0);
```

e的指数：如图MC的值为e的6次幂。

3.2.10 FIND

```
LS:=FIND(MB,wfe);
```

在另一个字符串中查找一个字符串。

输入：

IN1：字符串待寻找的基本字符串序列;通过工作寄存器使这个字符串有效

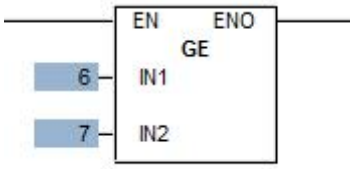
IN2：字符串要从IN1基本字符串中查找的字符串.

返回：

INT第一个出现的位置

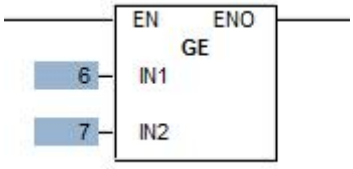
从IN1基本字符串中查找特殊字符序列。若找到，这个序列的第一个字符位置进入工作寄存器，否则，0进入工作寄存器。若找到不止一个，则首先找到的字符位置进入工作寄存器。

3.2.11 GE



如图，如果IN1>=IN2，则输出1，否则输出0。

3.2.12 GT



如图，如果IN1>=IN2，则输出1，否则输出0。

3.2.13 INSERT

```
ADC:=INSERT(MB,wfe,2);
```

输入：

IN1：STRING字符串

IN2：STRING待插入的字符串

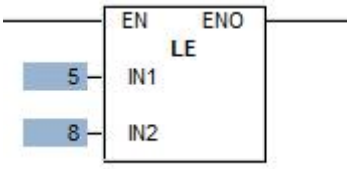
P：ANY_INT(根据支持)起始位置。P<0和P>LEN(IN1)无效

返回：

STRING组合的字符串。IN1无效参数

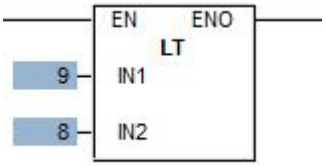
INSERT函数将字符串IN2插入字符串IN1。拼接后的字符串由IN1前P个字符，完整的IN2和剩下的IN1组成。

3.2.14 LE



如图，如果IN1<=IN2，则输出1，否则输出0。

3.2.15 LT



如图，如果IN1<IN2，则输出1，否则输出0。

3.2.16 LEFT

```
ADC:=LEFT(MB,2);
```

LEFT函数获得当前字符串的左边L个字符，并放入到工作寄存器。输入操作数L定义输入字符个数。

输入：

IN：字符串型字符串

L：ANY_INT(根据支持)要得到的字符数

返回：

STRINGIN左边L个字符，IN如果参数无效

3.2.17 LEN

```
LS:=LEN(MB);
```

函数LEN计算字符串长度(输入操作数类型为STRING)并把长度值以INT型数放入工作寄存器中。

输入：

In:STRING字符串

返回：

INT IN的长度

3.2.18 LIMIT

```
SHI:=LIMIT(56,YUE,89);
```

MN和MX值定义了上下限值。函数比较IN值与MN和MX大小。若IN在两限值之间,它将装载到工作寄存器。若IN小于MN,输出MN值.若IN大于MX,输出MX值。

输入：

MN：Any_Num下限

IN：Any_Num测试值

MX：Any_Num上限

返回：

Any_Num其中一个输入值

3.2.19 LN

```
mama:=LN(6.0);
```

以e为底的对数

如图，其中mama与6.0必须是real类型

3.2.20 LOG

```
mama:=LOG(6.0);
```

以10为底的对数：如图，其中mama与6.0必须是real类型。

3.2.21 MAX

```
NIAN:=MAX(1,2,3,4,5,6,9,8);
```

取所有数的最大值：如图，最后的输出值为9。

3.2.22 MIN

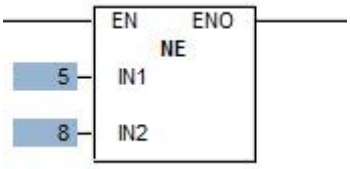
```
NIAN:=MIN(1,2,3,4,5,6,9,8);
```

取所有数的最小值：如图，最后的输出值为1。

3.2.23 MOD

详见3.1.14

3.2.24 NE



不等：如图，如果IN1<>IN2,则输出1，否则输出0。

3.2.25 POINTER

指针：POINTER是一个数据类型，为指针。在ST中，指针的作用是指向数组。例如：

PLT:=&ABUFER;

其中，PLT是指针类型的变量，ABUFER是数组变量。在某些功能块中，需要用到指针类型的变量，用来调用数组，读取或写入数据。

3.2.26 RIGHT

```
adc:=right(wfe,2);
```

输入：

IN:STRING字符串

L:ANY_INT(根据支持)要得到的字符数

返回：

STRING字符串L最右边的字符，IN如果参数无效

RIGHT函数把当前字符串的右边部分字节装载到工作寄存器.输入操作数L定义了要进入的字符个数。

3.2.27 ROL

```
b:=rol(a,2);
```

循环左移：

输入：

IN：ANY_BIT位形式

N：UINT左移的位数

返回：

ANY_BITIN,循环左移N位

左移出的位循环放到右边。

3.2.28 ROR

```
b:=ror(a,2);
```

循环右移：

输入：

IN：ANY_BIT位形式

N：UINT右移位数

返回：

ANY_BITIN,循环右移N位

右移出的位循环放到左边。

3.2.29 SHL

```
b:=shl(a,2);
```

左移：

输入：

IN：ANY_BIT位形式

N：UINT左移位数

返回：

ANY_BITIN,左移N位

最右边的位用零补

3.2.30 SHR

```
b:=shr(a,2);
```

右移：

输入：

IN：ANY_BIT位形式

N：UINT右移位数

返回：

ANY_BITIN,右移N位

最左边的位用零补

3.2.31 SIN

```
mc:=sin(5.0);
```

正弦：如图，其中，mc与5.0数据类型均为real。

3.2.32 SQRT

```
mc:=sqrt(5.0);
```

平方根：如图，其中，mc与5.0数据类型均为real。mc的值为$\sqrt{5}$。

3.2.33 TAN

```
mc:=tan(5.0);
```

正切：如图，其中，mc与5.0数据类型均为real。

3.2.24 XOR

```
b:=235 xor 119;
```

异或：如图，b的结果为99。

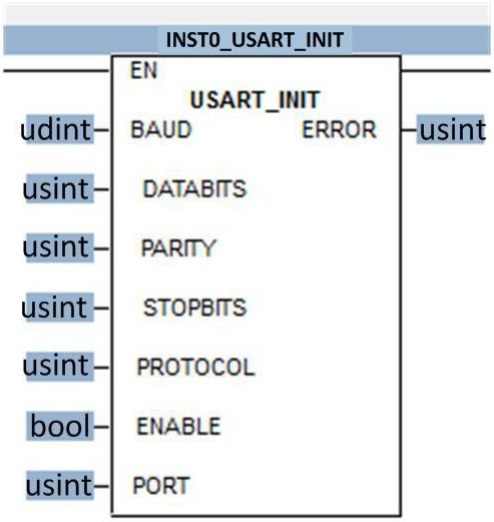
2. 扩展功能块

名称	功能	章节索引
串口功能块		
USART_INIT	串口初始化	3.3.1
USART_STATE	串口状态	3.3.2
USART_READ_BIN	串口读（二进制字符流）	3.3.3
USART_WRITE_BIN	串口写（二进制字符流）	3.3.4
USART_READ_CHR	串口读（字符）	3.3.5
USART_WRITE_CHR	串口写（字符）	3.3.6
USART_READ_STR	串口读（字符串）	3.3.7
USART_WRITE_STR	串口写（字符串）	3.3.8
EXT_USART_INIT	初始化扩展串行接口	3.3.9
EXT_USART_READ_BIN	扩展串行接口读取二进制字符流	3.3.10
EXT_USART_WRITE_BIN	扩展串行接口写入二进制字符流	3.3.11
CAN 功能块		
CAN_INIT	CAN初始化	3.3.12
CAN_MESSAGE_READ8	CAN读	3.3.13
CAN_MESSAGE_WRITE8	CAN写	3.3.14
CAN_NMT	发送NMT管理指令	3.3.15
CAN_GET_STATE	用于请求设备节点状态	3.3.16
CAN_REGISTER_COBID	用于注册或删除通过网络层接收的PDO和CAN第二层消息	3.3.17
CAN_PDO_READ8	用于读取PDO数据的功能块	3.3.18
CAN_PDO_WRITE8	用于发送PDO数据的功能块	3.3.19
CAN_SDO_READ8	通过SDO读对象字典	3.3.20
CAN_SDO_WRITE8	通过SDO写对象字典	3.3.21
CAN_SDO_READ_STR	通过SDO读对象字典中的字符串信息	3.3.22
CAN_SDO_WRITE_STR	通过SDO写对象字典中的字符串信息	3.3.23
CAN_SDO_READ_BIN	通过SDO读对象字典的二进制数据	3.3.24
CAN_SDO_WRITE_BIN	通过SDO写对象字典的二进制数据	3.3.25
CAN_RECV_EMCY	读取任意节点的紧急报文	3.3.26
CAN_RECV_EMCY_DEV	读取特定节点的紧急报文	3.3.27
CAN_WRITE_EMCY	发送特定应用程序紧急报文	3.3.28
CAN_ENABLE_CYCLIC_SYNC	激活SYNC循环同步	3.3.29
CAN_SEND_SYNC	发送同步帧（单帧）	3.3.30
CAN_ENABLE_CYCLIC_NODE_GUARD	循环发送节点保护	3.3.31
CAN_SEND_NODE_GUARD	发送节点保护（单帧）	3.3.32
CAN_RECV_BOOTUP_DEV	读取特定节点bootup消息	3.3.33
CAN_RECV_BOOTUP	读取所有节点bootup消息	3.3.34
CAN_FILTER	CAN总线滤波	3.3.35
Ethernet、GPRS 及 MQTT 功能块		
LAN_INIT	初始化以太网接口	3.3.36
LAN_GET_TCPCONNECT_SOCKET	用于建立TCP端的socket连接	3.3.37

LAN_TCP_SERVER_CREATE	创建TCP SERVER连接	3.3.38
LAN_TCP_CLIENT_CONNECT	创建TCP CLIENT连接	3.3.39
LAN_TCP_CLIENT_CLOSE	关闭TCP CLIENT连接	3.3.40
LAN_TCP_RECV_BIN	从TCP端口读取二进制字符流	3.3.41
LAN_TCP_SEND_BIN	讲二进制字符流写入TCP端口	3.3.42
LAN_UDP_CREATE_SOCKET	创建UDPsocket连接	3.3.43
LAN_UDP_CLOSE_SOCKET	关闭UDP SOCKET连接	3.3.44
LAN_UDP_RECVFROM_BIN	读取UDP数据	3.3.45
LAN_UDP_SENDTO_BIN	发送UDP数据	3.3.46
LAN_UDP_RECVFROM_STR	读取UDP数据（字符串）	3.3.47
LAN_UDP_SENDTO_STR	发送UDP数据（字符串）	3.3.48
EXT_GPRS_INIT	扩展4G功能块	3.3.49
EXT_GPRS_STATUC	扩展4G状态	3.3.50
EXT_GPRS_READ_BIN	扩展4G读	3.3.51
EXT_GPRS_WRITE_BIN	扩展4G写	3.3.52
MQTT_INIT		3.3.53
MQTT_STATUS		3.3.54
MQTT_SEND		3.3.55
Modbus 功能块		
MODBUS_SLAVE_INIT	初始化modbus-RTU从站接口	3.3.56
MODBUS_SLAVE_CTRL	执行Modbus-RTU从站控制指令	3.3.57
MODBUS_MASTER_INIT	初始化modbus-RTU主站接口	3.3.58
MODBUS_MASTER_CTRL	执行Modbus-RTU主站控制指令	3.3.59
MODBUS_TCP_SLAVE_INIT	初始化MODBUS_TCP从站接口	3.3.60
MODBUS_TCP_SLAVE_CTRL	执行MODBUS_TCP从站控制指令	3.3.61
MODBUS_TCP_MASTER_INIT	初始化MODBUS_TCP U主站接口	3.3.62
MODBUS_TCP_MASTER_CTRL	执行MODBUS_TCP主站控制指令	3.3.63
时钟、参数存储功能块		
DT_CLOCK	用于设置实时时钟并读取时间信息	3.3.64
RTC	系统时间功能块	3.3.65
GETSYSTEMDATEANDTIME	获取系统时间	3.3.66
SETSYSTEMDATEANDTIME	设置系统时间	3.3.67
GETDATESTRUCT	将日期转换为结构体	3.3.68
NVDATA_BIT	向EEPROM读取或写入数据	3.3.69
NVDATA_BIN	向EEPROM读取或写入数据（二进制流）	3.3.70
NVDATA_STR	向EEPROM读取或写入数据（字符串）	3.3.71
脉冲输出、计数功能块		
EXT_MOTOR_PWM_INIT	GC2302初始化功能块	3.3.72
EXT_MOTOR_EN	GC2302使能电机功能块	3.3.73
CTU	加法计数	3.3.74
CTD	减法计数	3.3.75
CTUD	双计数	3.3.76
INIT_TBL	初始化列表存储功能块	3.3.77
FIFO_TBL	列表成员先进先出	3.3.78

LIFO_TBL	列表成员先进后出	3.3.79
ADD_TBL	添加列表成员	3.3.80
COUNT_TBL	列表成员计数	3.3.81
其他功能块		
PID1	PID控制	3.3.82
SYS_PLC_RESET	系统复位重启功能块	3.3.83
PTO_PWM	脉冲输出PTO功能块	3.3.84
PTO	执行脉冲计时器	3.3.85
GETVARDATA	获取变量信息	3.3.86
GETVARFLATADDRESS	获取内存空间地址	3.3.87
GETTASKINFO	获取任务周期各执行时间	3.3.88

3.3.1 USART_INIT



用于初始化串行接口

操作数的定义

BAUD： 设置串口通信中使用的波特率，可设置为以下的数值：1200,2400,9600,19200,

38400,57600,115200；

DATABITS： 设置要使用的数据位数，例如：7：个数据位 ； 8:8个数据位；

PARITY： 设置用于安全数据传输的奇偶校验，例如：0:无奇偶校验

1:奇数校验

2:偶数校验；

STOPBITS： 设置要使用的停止位数，例如：1:1个停止位；2:2个停止位；

PROTOCOL：设置要使用的握手协议规则，例如：

0:无协议

1:XON/XOFF

2:硬件握手（RTS/CTS流量控制）；

ENABLE：启用或禁用串行接口：

true:启用串行接口

false:关闭串行接口；

PORT：需要使用的串行接口定义号，1为RS232，2为RS485；

ERROR：错误代码说明了有关功能块执行结果的信息。可能的错误代码定义如下：

0：在执行功能块时没有发生错误

1：在执行功能块时发生硬件错误

2：所选接口（PORT）不支持

4：所选波特率（BAUD）不支持

8：选定数量的数据位（DATABITS）不支持

16：所选奇偶校验（PARITY）不支持

32：所选择的停止位数不支持

64：所选的握手协议不支持

描述：

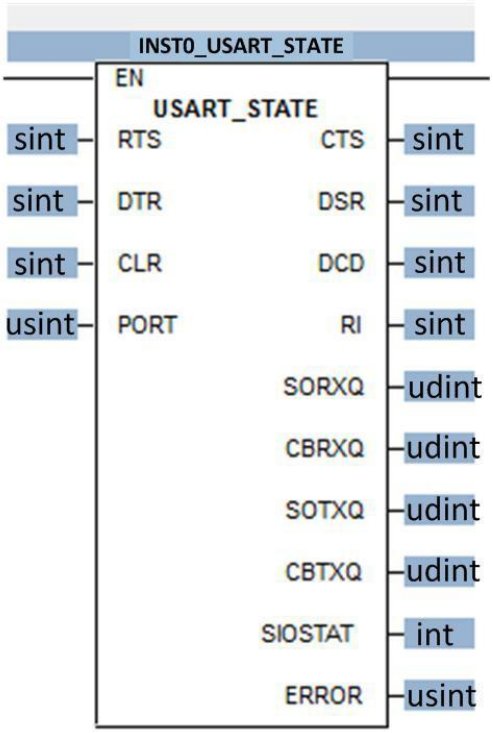
功能块使用指定的参数初始化串行接口。执行时可能会出现的错误在ERROR中显示。不同的错误代码将产生叠加，需要进行拆分（例如，6=2+4意为所选接口不支持且所选波特率不支持）。以下例程为使用功能块USART_INIT初始化串行接口，设置以下参数：串口号为RS485，9600波特率，8个数据位，无奇偶校验，1个停止位，无协议。

示例程序：

VAR

```
mPORT:USINT;
xInitOk : BOOL := FALSE; inst0_USART_INIT:USART_INIT;
xE:usint;
END_VAR


mPORT:=2;
if xInitOk=false then
inst0_USART_INIT(BAUD :=9600 , DATABITS :=8 , PARITY :=0 , STOPBITS :=1 , PROTOCOL :=0 ,
ENABLE :=1 , PORT :=mPORT| xE := ERROR);
if xE=0 then xInitOk :=TRUE
end_if;
end_if;
```

3.3.2 USART_STATE



用于设置和检索串行接口的状态信息

操作数的定义

RTS：要设置的RTS信号状态

-1：不影响当前状态

0：将信号置为无效

1：设置信号为有效

DTR：DTR信号状态待设定

-1：不影响当前状态

0：将信号置为无效

1：设置信号为有效

CLR：清除发送和接收缓冲区

-1：不影响当前状态

1：清除接收缓冲区

2：清除发送缓冲区

3：清除发送和接收缓冲区

PORT：要使用的串行接口定义号

CTS：确定CTS信号状态

-1：不支持信号

0：信号设置为无效

1：信号设置为有效

DSR：确定的DSR信号状态

-1：不支持信号

0：信号设置为无效

1：信号设置为有效

DCD：确定的DCD信号状态

-1：不支持信号

0：信号设置为无效

1：信号设置为有效

RI：确定的RI信号状态

-1：不支持信号

0：信号设置为无效

1：信号设置为有效

SORXQ：确定接收缓冲区的总体大小（Rx队列的大小）

CBRXQ：接收缓冲区中当前的字符数（Rx中的字节数队列）

SOTXQ：确定发送缓冲区的总体大小（Tx队列的大小）CBTXQ：发送缓冲区中当前的字符数（Tx

队列中的字节数）

SIOSTAT：USART特定状态寄存器（例如：超限，帧错误等;请参见各自控制）

ERROR：错误代码说明功能块执行结果的信息。可能的错误代码定义如下：

0:在执行功能块时没有发生错误

1:执行功能块时发生硬件错误

2:所选接口（PORT）不受支持

8:如果硬件握手处于活动状态，则RTS信号不受影响（USART_INIT），PROTOCOL：=2

16:如果硬件握手处于活动状态，则DTR信号不受影响（USART_INIT），PROTOCOL：=2

32:不支持直接设置RTS信号

64:不支持直接设置DTR信号

128:不支持清除发送和接收缓冲区

255:所选接口（PORT）未初始化

描述：

功能块用于设置和检索串行接口的状态信息。实际可用性或参数的支持取决于接口的相应硬件属性。请看各控制手册各自有更详细的信息。执行期间可能出现的功能块错误在输出ERROR处显示，可以同时通知几个错误（例如24=16+8=>在激活的硬件握手期间DTR信号和RTS信号不受影响）。

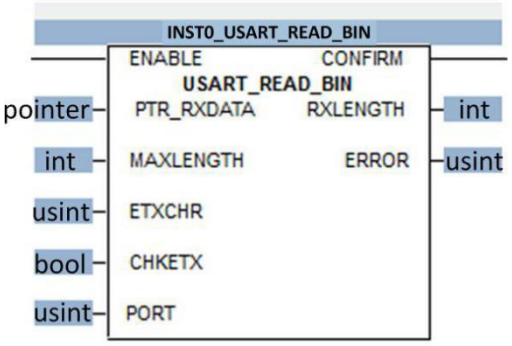
以下示例程序显示应用功能块USART_STAT确定串行接口当前状态的过程。

示例程序：

```
VAR
FB_USARTState : USART_STATE;
xStatOk : BOOL := FALSE;
siCts : SINT;
siDsr : SINT;
siDcd : SINT;
```

```
siRi : SINT;
udiSoRrQ : UDINT;
udiCbRxQ : UDINT;
udiSoTxQ : UDINT;
udiCbTxQ : UDINT;
iUSARTStat : INT;
END_VAR

(* ----- Check USART State ----- *)
CheckState:
(* read current state from serial interface *)
CAL FB_USARTState (RTS := USART_STAT_DO_NOT_CHANGE,
DTR := USART_STAT_DO_NOT_CHANGE,CLR := USART_STAT_DO_NOT_CHANGE,PORT := PORTNUM
|siCts := CTS,siDsr := DSR,siDcd := DCD,siRi := RI,udiSoRrQ := SORXQ,udiCbRxQ := CBRXQ,udiSoTxQ :=
SOTXQ,udiCbTxQ := CBTXQ,iUSARTStat := USARTSTAT)
LD    FB_USARTState.ERROR
EQ USART_STAT_ERR_SUCCESS
ST xStatOk
... RET
```

3.3.3 USART_READ_BIN



用于从串行接口读取二进制字符流

操作数的定义

ENABLE：启用或禁用功能块的输入;

PTR_RXDATA：用于接收读取数据字节对象的地址;

MAXLENGTH：读取最大字节数，如果为0，则由PTR_RXDATA确定要读取的字节数;

ETXCHR：二进制字符流的结束定界符（仅检查是否CHECKBOX=TRUE）;

CHKETX：打开/关闭结束分隔符字符检查，FALSE=未检查结束分隔符字符,TRUE=检查结束分隔符字符是否激活;

PORT：要使用的串行接口定义号，1为RS232，2为RS485;

CONFIRM：显示功能块的完成情况，FALSE=接收未成功完成或错误后终止,TRUE=接收成功完成，由PTR_RXDATA寻址的对象中RXLENGTH长度的字符可用;

RXLENGTH：读取字符数（如果ERROR：=0）;ERROR：错误代码说明有关功能块执行结果的信息。可能的错误代码定义如下：

0:在执行功能块时没有发生错误

1:执行功能块时发生硬件错误

2:所选接口（PORT）不受支持

8:没有收到终端分隔符的字符，接收MAXLENGHTH长度的字符后终止

128:指针引用不支持的数据类型的对象

255:所选接口（PORT）未初始化

描述：

功能块从串行接口读取二进制数据流。读取的字符被存储在由输入PTR_RXDATA寻址的对象中。如果输入CHKETX设置为TRUE，则检查读到的数据流对象中是否出现在输入端EOTCHR定义的结束分隔符。在识别到定义的结束分隔符后，读取操作完成，输出CONFIRM设置为TRUE。如果输入CHKETX设置为FALSE或者在读取的二进制流中未出现

定义的结束分隔符，功能块在接收到最大字节数后停止接收，（数据对象的内部尺寸由PTR_RXDATA指向的地址或MAXLENGTH定义的最大字符数而定），在这种情况下如果功能块反馈功能正常，则输出的CONFIRM也会被置为TRUE。

输出RXLENGTH显示存储在由PTR_RXDATA寻址的数据对象中的字符数。如果输出

ERROR:=0，则由输入ETXCHR定义的结束分隔符已收到。如果输出ERROR:=8，在接收到最大字符数后接收终止。

检测到输入ENABLE的上升沿之后，程序段开始字符接收。通过PLC程序重复调用功能块，直至接收到结束分隔符或MAXLENGTH个字符后终止。为此，必须设置输入

ENABLE为TRUE来使能字符接收。该功能块成功终止接收后将会令输出CONFIRM置为

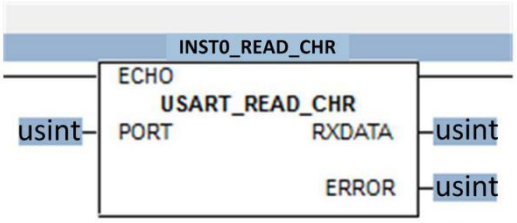
TRUE。结束接收字符串的进程后，PLC程序必须采取令ENABLE:=FALSE的方式调用功能块，用来将功能块内部重置为初始状态。随后可以通过将输入ENABLE重置为TRUE，上升沿成功使能后继续接收。可以随时令ENABLE:=FALSE来终止主动接收。

功能块执行过程中可能的错误显示在输出ERROR中。由于各种位的同时设置，可以同时显示多种错误。可通过示例程序了解功能块USART_READ_BIN读取串行接口字符流的工作过程。

示例程序：

示例程序中通过先调用USART_READ_BIN功能块来读取二进制字符流。字符流完全读取后，通过USART_WRITE_BIN功能块将其重写到接口上（参考3.3.4节示例程序）。

3.3.4 USART_WRITE_BIN



用于从串行接口读取单个字符

操作数的定义

ENABLE：启用或禁用功能块的输入；

PTR_TXDATA：要发送的二进制数据对象的地址；

TXLENGTH：要发送的数据字节数，如果数字为0，则对象的长度是通过PTR_TXDATA作为指针在内部寻址确定的；

PORT：要使用的串行接口定义号，1为RS232，2为RS485；

CONFIRM：功能块处理之后的结果，FALSE表明传输未成功完成或错误后终止，TRUE表明传输已成功完成；

ERROR：错误代码说明有关功能块执行结果的信息。可能的错误代码定义如下：

0:在执行功能块时没有发生错误

1:执行功能块时发生硬件错误

2:所选接口（PORT）不受支持

8:指针引用不支持的数据类型的对象

128:所选接口（PORT）未初始化

描述：

功能块将二进制字符流写入串行接口。要写入的二进制数据的地址必须传送到输入

PTR_TXDATA中。输入TXLENGTH指定有效字节数，如果该值为0，则由PTR_TXDATA在内部确定。

在输入ENABLE检测到上升沿之后，该块开始写入字符流。通过PLC程序重复调用功能块，直到字符传输完成。为此，必须将输入ENABLE设置为TRUE以启用字符传输。该功能块通过输出

CONFIRM:=TRUE自动显示其传输成功。在更复杂的进程中，PLC程序必须通过ENABLE:=FALSE来复位其内部状态。随后可以将输入ENABLE置为TRUE并检测上升沿来传输更多数据。主动传输随时可以通过ENABLE:=FALSE来终止。

功能块执行过程中可能的错误显示在输出ERROR中。由于各种位的同时设置，可以同时显示多种错误。

以下示例程序显示功能块USART_READ_BIN和USART_WRITE_BIN的联合应用。首先，调用块USART_READ_BIN来读取二进制字符流。字符流完全读取后，调用块

USART_WRITE_BIN将其重写到接口上。该示例程序同时添加了串行接口的初始化和程序执行的流程控制。

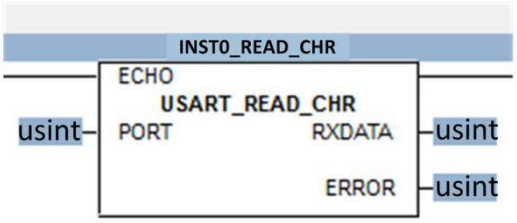
示例程序:

```
VAR
mPORT:USINT;
xInitOk : BOOL := FALSE;
inst0_USART_INIT:USART_INIT;
inst1_USART_INIT:USART_INIT;
xRdBinConfirm : BOOL := FALSE;
xWaitForReceipt : BOOL := true;
abDataBuffer : ARRAY[0..256] OF BYTE;
pDataObject : POINTER;
inst0_USART_WRITE_BIN:USART_WRITE_BIN;
xWrBinConfirm : BOOL := FALSE;
xTransmitting : BOOL := FALSE;
iRxDataSize : INT;
xReadStart:bool:=true;
```

```
xWriteStart:bool:=FALSE;
xERROR:usint;
xE:usint;
END_VAR

mPORT:=2;
if xInitOk=false then
inst0_USART_INIT(BAUD :=115200 , DATABITS :=8 , PARITY :=0 , STOPBITS :=1 , PROTOCOL :=0 ,
ENABLE :=1 , PORT :=mPORT | xE := ERROR);
if xE=0 then
xInitOk :=TRUE;
end_if;
end_if;
(* test USART_READ_BIN     USART_WRITE_BIN*)
pDataObject:=&abDataBuffer;
if xInitOk then
if xReadStart then
inst0_USART_READ_BIN(ENABLE :=0 ,     PORT :=mPORT    |     xERROR:= ERROR);
xWaitForReceipt:=true;
xWrBinConfirm:=false;
xReadStart:=false;
end_if;
```

```
if xWaitForReceipt then
inst0_USART_READ_BIN(ENABLE :=1 , PTR_RXDATA :=pDataObject ,
MAXLENGTH :=0 , ETXCHR :=6, CHKETX :=0, PORT :=mPORT | xRdBinConfirm:= CONFIRM,
iRxDataSize:= RXLENGTH, xERROR:= ERROR);
if xRdBinConfirm then xWaitForReceipt:=false; xWriteStart:=true;
end_if;
end_if;
if xWriteStart then
inst0_USART_WRITE_BIN(ENABLE :=0 , PORT :=mPORT | xERROR:= ERROR);
xRdBinConfirm:=false;
xWrBinConfirm:=false;
xTransmitting:=true;
xWriteStart:=false;
end_if;
if xTransmitting then
inst0_USART_WRITE_BIN(ENABLE :=1 ,PTR_TXDATA :=pDataObject ,TXLENGTH :=iRxDataSize,
PORT :=mPORT | xWrBinConfirm := CONFIRM, xERROR:= ERROR);
if xWrBinConfirm then
xTransmitting :=false;
xReadStart:=true;
end_if;
end_if;
end_if;
```

3.3.5 USART_READ_CHR



用于从串行接口读取单个字符

操作数的定义

ECHO：字符回显开/关,FALSE：无回显,TRUE：有回显;

PORT：要使用的串行接口定义号;

RXDATA：接收到的字符（如果ERROR：=0）;

ERROR：错误代码说明有关功能块执行结果的信息。可能的错误代码定义如下：

0:执行功能块时没有发生错误

1:执行功能块时发生硬件错误

2:所选接口（PORT）不受支持

8:接收缓冲区中没有字符可用

16:不支持字符回显

255:所选接口（PORT）未初始化

描述：

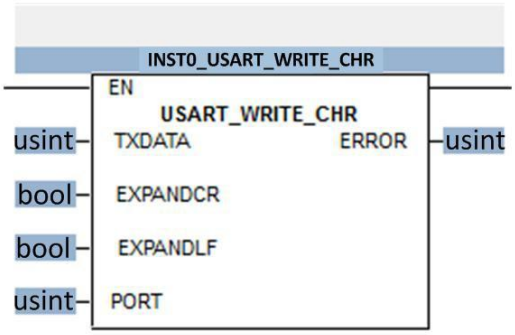
功能块从串行接口读取单个字符。如果输出ERROR：=8，则接收缓冲区没有字符可用。如果ERROR：=0，则读取的字符在输出RXDATA处可见。在这种情况下，如果输入ECHO：=TRUE，则接收到的字符会被功能块以回显的形式返回。执行期间可能出现的功能块错误在输出ERROR处显示，由于各种位的同时设置，可以同时显示多种错误。

示例程序:

功能块USART_READ_CHR和功能块USART_WRITE_CHR联合使用。首先，在程序中调用功能块USART_READ_CHR来从接口读取字符，读取成功后，调用功能块USART_WRITE_CHR在接口写入字符

（见3.3.6节）。

3.3.6 USART_WRITE_CHR



用于将单个字符写入串行接口

操作数的定义

TXDATA：输入要发送的字符；

EXPANDCR：自动扩展回车开/关

FALSE：没有自动扩展回车

TRUE：自动扩展回车

CR（'$R'=13）自动扩展到CR+LF（'$R$L'=13+10）

EXPANDLF：自动扩展换行开/关

FALSE：没有自动扩展换行

TRUE：自动扩展换行

LF（'$L'=10）扩展到CR+LF（'$R$L'=13+10）

PORT：要使用的串行接口定义号；

ERROR：错误代码说明有关功能块执行结果的信息。可能的错误代码定义如下：

0：在执行功能块时无错误

1：执行功能块时发生硬件错误

2：所选接口（PORT）不受支持

8：发送缓冲区中无可用空间，字符已丢失

16：不支持回车的自动扩展

32：不支持换行的自动扩展

255：所选接口（PORT）未初始化

描述：

该功能块将单个字符写入串行接口。如果输出ERROR：=1，则发送缓冲区中无空间可用。如果ERROR：=0，则输出TXDATA成功将字符写入接口。如果输入EXPANDCR：=TRUE，功能块会自动将该字符扩展为带回车的字符串。与此类似，如果输入EXPANDLF：=TRUE，则功能块会自动将该字符扩展为带换行的字符串。

功能块在执行过程中可能出现的错误会作为位掩码显示在输出ERROR中。由于各种位的同时设置，可以同时显示多种错误。以下例程显示功能块USART_READ_CHR和

USART_WRITE_CHR联合使用实现串行接口读写字符。首先，示例程序调用功能块

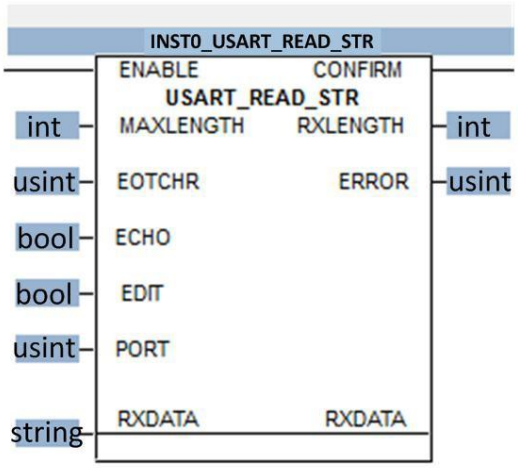
USART_READ_CHR从接口读取一个字符，成功后调用功能块USART_WRITE_CHR在接口写入字符。

示例程序：

```
VAR
xEcho : BOOL := FALSE; FB_USARTReadChr : USART_READ_CHR;
usiRxData : USINT; xRdCharSuccess : BOOL := FALSE; xExpandCR : BOOL := FALSE; xExpandLF : BOOL := FALSE;
FB_USARTWriteChr : USART_WRITE_CHR; xWrCharOk : BOOL := FALSE;
END_VAR

(* ----- Read Char ----- *)
CAL FB_USARTReadChr (ECHO := xEcho,PORT := PORTNUM|usiRxData := RXDATA)
(* check receive result *)
```

```
LD FB_USARTReadChr.ERROR (* character received ? *)
EQ USART_RCHR_ERR_SUCCESS
ST xRdCharSuccess RETCN
(* ----- Write Char ----- *)
CAL FB_USARTWriteChr (TXDATA := usiRxData,EXPANDCR :=
xExpandCR,EXPANDLF :=xExpandLF,PORT := PORTNUM)
LD FB_USARTWriteChr.ERROR EQ USART_WCHR_ERR_SUCCESS
ST xWrCharOk
...
RET
```

3.3.7 USART_READ_STR



用于从串行接口读取一个字符串

操作数的定义：

RXDATA：用于接收读取字符的字符串变量；

ENABLE：启用或禁用功能块的输入；

MAXLENGTH：限制要读取的字符数量，如果数字为0，则在内部确定缓冲区传递

的字符串的长度并用作该分隔符要读取的字符数（注意：OpenPCS中字符串的标准缓冲区大小

是32个字符）。

EOTCHR：字符串结束分隔符的字符（默认值：10='$L'），例如：0（NUL），10（'$L'=换行符），13（'$R'=回车）；

ECHO：字符回显开/关，FALSE：无回显，TRUE：返回回显；

EDIT：编辑模式开/关，FALSE:BS（8）作为普通字符存储在接收字符串中，TRUE:BS（8）被解释为校正字符；

PORT：要使用的串行接口定义号；

CONFIRM：通过功能块输出完成的消息，FALSE：接收未成功完成或错误后终止，TRUE：接收成功完成，RXLENGTH字符可用，接收缓冲区RXDATA；

RXLENGTH：读取字符串的长度（如果ERROR：=0）；

ERROR：错误代码说明有关功能块执行结果的信息。可能的错误代码定义如下：

0:在执行功能块时没有发生错误

1:执行功能块时发生硬件错误

2:所选接口（PORT）不受支持

8:没有收到终端分隔符的字符，接收MAXLENGHTH个字符后终止

16:不支持字符回显

32:不支持编辑模式

255:所选接口（PORT）未初始化

描述：

功能块从串行接口读取一个字符串。读取的字符串传递到输入RXDATA中。字符串在读取到定义为输入EOTCHR的结尾分隔符后终止接收，或者如果字符串设置为MAXLENGTH的字符数是满的（考虑到EDIT，如果MAXLENGTH：=0，传递的字符串的缓冲区长度在内部确定并用作分隔符）。在这两种情况下，返回的块显示接收已经完成，并且字符串传递给输出RXDATA，同时将输出CONFIRM设置为TRUE来表示已读取到字符串。输出RXLENGTH显示接收缓冲区中的字符数（等于LEN（RXDATA））。如果输出ERROR：=0，则表示已经收到输入EOTCHR定义的结束定界符。如果错误：=8，读取设置的

字符数后，接收终止最长长度。检测到输入ENABLE上的上升沿之后，程序段开始字符接收（第一次通过ENABLE：=TRUE）。通过PLC程序重复调用功能块，直到字符接收（结束分隔符或MAXLENGTH个字符）完成。为此，必须设置输入ENABLE作为TRUE来启用字符接收。该功能块成功终止接收后将CONFIRM置为TRUE。在处理完接收到的字符串后，PLC程序必须通过ENABLE：=FALSE调用功能块，以将块内部重置为初始状态。随后可以通过将输入ENABLE置为TRUE并检测接收信号的上升沿来继续工作。随时可以通过令ENABLE：=FALSE来终止主动接收。

如果输入ECHO：=TRUE，则功能块会自动将每个接收到的字符返回为回显。字符退格（BS=8）不作为正常字符存储，而是作为校正字符存储，如果输入EDIT：=TRUE，则在接收缓冲区中，最后接收到的字符会被删除并且输出RXLENGTH报告的已接收字符数减少。

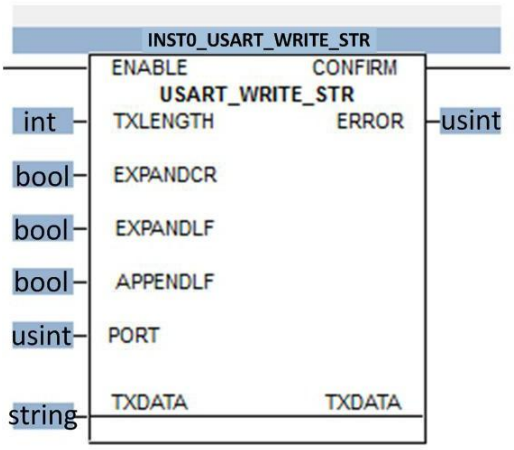
功能块在执行过程中可能的错误在输出ERROR中显示。由于各种位的同时设置，可以发信号通知几个错误。

示例程序中显示了功能块USART_READ_STR读取串行接口的字符串的工作过程。

示例程序：

示例程序显示了功能块USART_READ_STR和功能块USART_WRITE_STR的联合工作过程。首先，例程通过调用功能块USART_READ_STR来读取字符串。字符串完全读取后，再通过功能块USART_WRITE_STR将其重写到接口上。（参考3.3.8节示例程序）

3.3.8 USART_WRITE_STR



用于将字符串写入串行接口

操作数的定义：

TXDATA：要写入的字符串；

ENABLE：启用或禁用功能块的输入；

TXLENGTH：要写入的字符数，如果数字为0，则字符的长度及字符串中包含的字符TXDATA是内部确定的（等于LEN（TXDATA））；

EXPANDCR：自动扩展回车开/关

FALSE：没有自动扩展回车；

TRUE：自动扩展回车；

CR（'$R'=13）自动扩展到CR+LF（'$R$L'=13+10）；

EXPANDLF：自动扩展换行开/关

FALSE：没有自动扩展换行；

TRUE：自动扩展换行；

LF（'$L'=10）自动扩展到CR+LF（'$R$L'=13+10）；

APPENDLF：自动附加换行开/关

FALSE：没有自动附加换行，LF（'$L'=10）；

TRUE：自动附加换行，CR+LF（'RL'=13+10）；

PORT：要使用的串行接口定义号；

CONFIRM：通过功能块输出完成的消息，FALSE：传输未成功完成或错误后终止，

TRUE：传输成功完成；

ERROR：错误代码说明有关功能块执行结果的信息。可能的错误代码定义如下：

0：在执行功能块时没有发生错误

1：执行功能块时发生硬件错误

2：所选接口（PORT）不受支持

16：不支持回车的自动扩展

32：不支持换行符的自动扩展

64：不支持换行符的自动附加

255：所选接口（PORT）未初始化

描述：

功能块将字符串写入串行接口。要传输的字符串在输入TXDATA处传递。输入

TXLENGTH指定有效字符数。如果该值为0，则字符串中包含的字符串的长度TXDATA在内部确定（等于LEN（TXDATA）），并作为要处理的字符数。在这种情况下，写入整个占用的字符串内容。

在输入ENABLE（首先通过调用）检测到上升沿之后，该块开始写入字符串

（ENABLE：=TRUE）。通过PLC程序重复调用功能块，直到字符传输已经完成。为此，必须将输入ENABLE设置为TRUE以启用字符传输。该块通过令输出CONFIRM为TRUE来自动发出成功完成信号。PLC程序必须通过ENABLE：=FALSE调用块来内部复位其初始状态。随后可以通过检测输入ENABLE的上升沿开始进一步的字符传输。主动传输可以通过ENABLE：=FALSE随时终止。

如果输入EXPANDCR：=TRUE，该块将自动扩展带有ASCII码的字符串+回车符。与此类似，块还自动添加字符串的换行符，如果输入EXPANDLF：=TRUE，则扩展为字符串

+换行符。如果输入APPENDLF：=TRUE，则块在完全传输字符串后内部附加换行符传递到输出TXDATA。根据输入APPENDLF，在需要的情况下，此换行符将被转移作为由回车符+换行符组成的字符串。

功能块执行过程中可能的错误作为位掩码显示在输出ERROR中。由于各种位的同时设置，可以发信号通知几个错误。

以下示例程序显示功能块USART_WRITE_STR的应用。首先，调用块USART_READ_STR来从接口读取字符串。字符串完全读取后，块USART_WRITE_STR将其重写到接口上。该示例程序集成了初始化串行接口和程序执行的流程控制。

示例程序:

(* ----- Init USART ----- *) USARTInit:

CAL FB_USARTInit (BAUD := 9600,DATABITS := 8,PARITY := USART_INIT_PARITY_NO, STOPBITS := 1,PROTOCOL := USART_INIT_PROTOCOL_NO,ENABLE := TRUE,PORT := PORTNUM)

(* ----- Read String ----- *) ReadStringStart:

CAL FB_USARTReadStr (ENABLE := FALSE, (* Step 1: Reset FB *)RXDATA := strRxText, PORT := PORTNUM)

CAL FB_USARTReadStr (ENABLE := TRUE, (* Step 2: Start FB *)RXDATA := strRxText, MAXLENGTH := 0, (* no limit, use whole string length *)EOTCHR := usiEotChr, ECHO := xEcho,EDIT := xEdit,PORT := PORTNUM|xRdStrConfirm := CONFIRM, iRxDataSize := RXLENGTH)

(* ----- Write String ----- *) WriteStringStart:

CAL FB_USARTWriteStr (ENABLE := FALSE, (* Step 1: Reset FB *)TXDATA := strTxText, PORT := PORTNUM)

WriteStringCont:

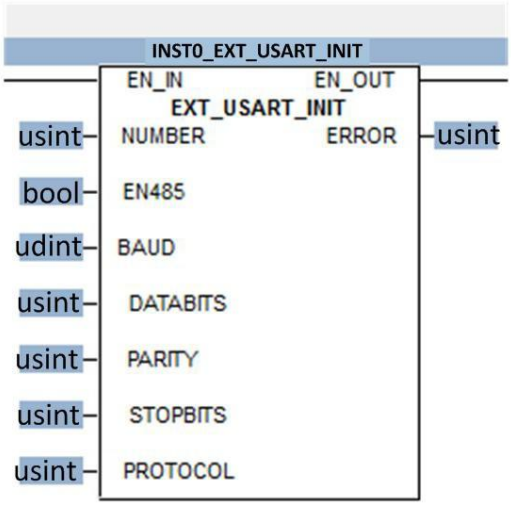
CAL FB_USARTWriteStr (ENABLE := TRUE, (* Step 2: Start FB *)TXDATA := strTxText, TXLENGTH := 0, (* no limit, transmit whole string *)EXPANDCR := xExpandCR, EXPANDLF := xExpandLF,APPENDLF := xAppendLF,PORT := PORTNUM|xWrStrConfirm := CONFIRM)

(* ----- Reset Flow Control Logic ----- *) LD FALSE

ST xTransmitting ST xWrStrConfirm ST xWaitForReceipt ST xRdStrConfirm RET

END_PROGRAM

3.3.9 EXT_USART_INIT



用于初始化扩展串行接口

操作数的定义：

EN_IN：启用或禁用串行接口：

true:启用串行接口；

false:关闭串行接口；

NUMBER：用于扩展串口的GC-6101模块的序列号，主控模块后第一个GC-6101模块序列号为1，第二个为2，以此类推；

EN485：设置串口通讯模式，1为RS485通讯，2为RS232通讯：

BAUD：设置串口通信中使用的波特率，可设置为以下的数值：1200,2400,9600,

19200,38400,57600,115200；

DATABITS：设置要使用的数据位数，例如：7:7个数据位

8:8个数据位；

PARITY：设置用于安全数据传输的奇偶校验，例如：0:无奇偶校验

1:奇数校验

2:偶数校验；

STOPBITS：设置要使用的停止位数，例如：

1:1个停止位

2:2个停止位；

PROTOCOL：设置要使用的握手协议规则，例如：

0:无协议

1:XON/XOFF

2:硬件握手（RTS/CTS流量控制）；

EN_OUT：功能块使能输出；

ERROR：错误代码说明了有关功能块执行结果的信息。可能的错误代码定义如下:

0:在执行功能块时没有发生错误

1:在执行功能块时发生硬件错误

2:所选接口（PORT）不支持

4:所选波特率（BAUD）不支持

8:选定数量的数据位（DATABITS）不支持

16:所选奇偶校验（PARITY）不支持

32:所选择的停止位数不支持

64:所选的握手协议不支持

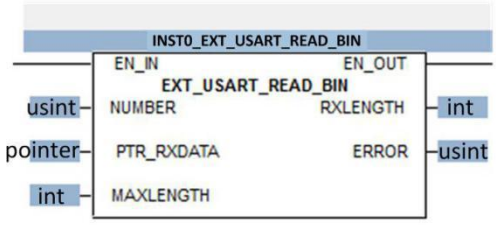
描述：

功能块使用指定的参数初始化串行接口。执行时可能会出现的错误在输出ERROR中显示。不同的错误代码将产生叠加，需要进行拆分（例如，6=2+4意为所选接口不支持且所选波特率不支持）。以下例程为使用功能块USART_INIT初始化串行接口，设置以下参数：串口号为RS485，9600波特率，8个数据位，无奇偶校验，1个停止位，无协议。

示例程序：

```
VAR mPORT:USINT;

xInitOk : BOOL := FALSE; inst0_EXT_USART_INIT:USART_INIT;

xE:usint; END_VAR

mPORT:=2;

if xInitOk=false then

inst0_EXT_USART_INIT(BAUD :=9600 , DATABITS :=8 , PARITY :=0 , STOPBITS :=1 , PROTOCOL :=0 , ENABLE :=1 , PORT :=mPORT | xE := ERROR);

if xE=0 then xInitOk :=TRUE; end_if;

end_if;
```

3.3.10 EXT_USART_READ_BIN



用于从扩展串行接口读取2进制字符流

操作数的定义：

EN_IN：启用或禁用功能块的输入;

NUMBER：用于扩展串口的GC-6101模块的序列号，主控模块后第一个GC-6101模块序列号为1，第二个为2，以此类推;

PTR_RXDATA：用于接收读取数据字节对象的地址;

MAXLENGTH：读取最大字节数，如果为0，则由PTR_RXDATA确定要读取的字节数;

EN_OUT：功能块使能输出;

RXLENGTH：读取字符数（如果ERROR：=0）;

ERROR：错误代码说明有关功能块执行结果的信息。可能的错误代码定义如下：

0:在执行功能块时没有发生错误

1:执行功能块时发生硬件错误

2:所选接口（PORT）不受支持

8:没有收到终端分隔符的字符，接收MAXLENGHTH长度的字符后终止

128:指针引用不支持的数据类型的对象

255:所选接口（PORT）未初始化

描述：

功能块从串行接口读取二进制数据流。读取的字符被存储在由输入PTR_RXDATA寻址的对象中。功能块在接收到最大字节数后停止接收，（数据对象的内部尺寸由

PTR_RXDATA指向的地址或MAXLENGTH定义的最大字符数而定），在这种情况下如果功能块反馈功能正常，则输出的EN_OUT也会被置为TRUE。

输出RXLENGTH显示存储在由PTR_RXDATA寻址的数据对象中的字符数。检测到输入

ENABLE的上升沿之后，程序段开始字符接收。通过PLC程序重复调用功能块，直至接收到结束分隔符或MAXLENGTH个字符后终止。为此，必须设置输入ENABLE为TRUE来使能字符接收。该功能块成功终止接收后将会令输出CONFIRM置为TRUE。结束接收字符串的进程后，PLC程序必须采取令ENABLE:=FALSE的方式调用功能块，用来将功能块内部重置为初始状态。随后可以通过将输入ENABLE重置为TRUE，上升沿成功使能后继

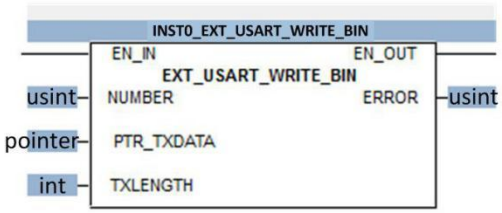
续接收。可以随时令ENABLE:=FALSE来终止主动接收。

功能块执行过程中可能的错误显示在输出ERROR中。由于各种位的同时设置，可以同时显示多种错误。可通过示例程序了解功能块EXT_USART_READ_BIN读取串行接口字符流的工作过程。

示例程序：

示例程序中通过先调用EXT_USART_READ_BIN功能块来读取二进制字符流。字符流完全读取后，通过EXT_USART_WRITE_BIN功能块将其重写到接口上（见3.3.10节）。

3.3.11 EXT_USART_WRITE_BIN



用于将二进制字符流写入扩展串行接口

操作数的定义：

EN_IN：启用或禁用功能块的输入；

NUMBER：用于扩展串口的GC-6101模块的序列号，主控模块后第一个GC-6101模块序列号为1，第二个为2，以此类推；

PTR_TXDATA：要发送的二进制数据对象的地址；

TXLENGTH：要发送的数据字节数，如果数字为0，则对象的长度是通过PTR_TXDATA作为指针在

内部寻址确定的；

EN_OUT：功能块处理之后的结果，FALSE表明传输未成功完成或错误后终止，

TRUE表明传输已成功完成；

ERROR：错误代码说明有关功能块执行结果的信息。可能的错误代码定义如下：

0:在执行功能块时没有发生错误

1:执行功能块时发生硬件错误

2:所选接口（PORT）不受支持

8:指针引用不支持的数据类型的对象

128:所选接口（PORT）未初始化

描述：

功能块将二进制字符流写入串行接口。要写入的二进制数据的地址必须传送到输入

PTR_TXDATA中。输入TXLENGTH指定有效字节数，如果该值为0，则由PTR_TXDATA在内部确定。

在输入EN_IN检测到上升沿之后，该块开始写入字符流。通过PLC程序重复调用功能块，直到字符传输完成。为此，必须将输入EN_IN设置为TRUE以启用字符传输。该功能块通过输出EN_OUT:=TRUE自动显示其传输成功。在更复杂的进程中，PLC程序必须通过EN_IN:=FALSE来复位其内部状态。随后可以将输入EN_IN置为TRUE并检测上升沿来传输更多数据。主动传输随时可以通过EN_IN:=FALSE来终止。

功能块执行过程中可能的错误显示在输出ERROR中。由于各种位的同时设置，可以同时显示多种错误。

以下示例程序显示功能块EXT_USART_READ_BIN和EXT_USART_WRITE_BIN的联合应用。首先，调用块EXT_USART_READ_BIN来读取二进制字符流。字符流完全读取后，调用块EXT_USART_WRITE_BIN将其重写到接口上。该示例程序同时添加了串行接口的初始化和程序执行的流程控制。

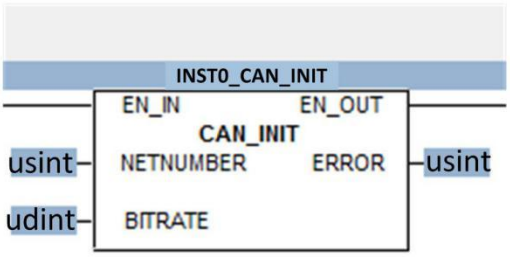
示例程序:

```
VAR
xInitOk : BOOL := FALSE; inst0_EXT_USART_INIT:EXT_USART_INIT;
```

```
xRdBinConfirm : BOOL := FALSE; xWaitForReceipt : BOOL := true;
inst0_EXT_USART_READ_BIN:EXT_USART_READ_BIN;
abDataBuffer : ARRAY[0..256] OF BYTE; pDataObject : POINTER;
inst0_EXT_USART_WRITE_BIN:EXT_USART_WRITE_BIN;
xWrBinConfirm : BOOL := FALSE; xTransmitting : BOOL := FALSE; iRxDataSize : INT;
xReadStart:bool:=true; xERROR:usint;
xE:usint; var4:INT; eout:bool;
END_VAR
if xInitOk=false then
inst0_EXT_USART_INIT(EN_IN :=TRUE , NUMBER :=1 , EN485 :=0, BAUD :=115200 , DATABITS :=8 ,
PARITY :=0 , STOPBITS :=1 , PROTOCOL :=0      | eout := EN_OUT,     xE:=
ERROR);
if xE=0 and eout=true then xInitOk :=TRUE;
end_if; end_if;
(* test USART_READ_BIN USART_WRITE_BIN*)
pDataObject:=&abDataBuffer; if xInitOk then
if xReadStart then
inst0_EXT_USART_READ_BIN(EN_IN :=0 , NUMBER :=1      |     xERROR := ERROR);
xWaitForReceipt:=true; xRdBinConfirm:=false; xReadStart:=false;
end_if;
if xWaitForReceipt then inst0_EXT_USART_READ_BIN(EN_IN := 1, NUMBER := 1 ,
PTR_RXDATA :=pDataObject , MAXLENGTH := 0 | xRdBinConfirm := EN_OUT, iRxDataSize :=
RXLENGTH,xERROR:= ERROR);
if xRdBinConfirm then var4:=iRxDataSize; xWaitForReceipt:=false;
```

```
end_if; END_if;

if xRdBinConfirm then

inst0_EXT_USART_WRITE_BIN(EN_IN :=0 , NUMBER :=1    | xERROR:= ERROR);

xRdBinConfirm:=false; xTransmitting:=true;

end_if;

if xTransmitting then abDataBuffer[0]:=49;

(* abDataBuffer[1]:=1; abDataBuffer[2]:=2; iRxDataSize:=3;*)

inst0_EXT_USART_WRITE_BIN(EN_IN := 1, NUMBER :=1 ,

PTR_TXDATA :=pDataObject, TXLENGTH :=iRxDataSize |    xWrBinConfirm := EN_OUT, xERROR :=
ERROR);

if xWrBinConfirm then

xTransmitting :=false; xReadStart:=true;

end_if; end_if;

end_if;
```

3.3.12 CAN_INIT



用于初始化CAN接口

操作数的定义：

BITRATE：波特率以比特/秒为单位，分类如下：BITRATE：=0表示1MBit/s

BITRATE：=1表示840kBit/s

BITRATE：=2表示700kBit/s

BITRATE：=3表示500kBit/s

BITRATE：=4表示400kBit/s

BITRATE：=5表示250kBit/s

BITRATE：=6表示200kBit/s

BITRATE：=7表示125kBit/s

BITRATE：=8表示100kBit/s；

BITRATE：=9表示80kBit/s

BITRATE：=10表示50kBit/s

BITRATE：=11表示40kBit/s

BITRATE：=12表示20kBit/s

BITRATE：=13表示10kBit/sAMR：CAN控制器接收屏蔽寄存器（AMR）的配置值;允许在CAN控制器中对CAN标识符

进行硬件过滤（所有CAN消息处理的标准值：AMR：=16＃FFFFFFFF）；ACR：CAN控制器接收代码寄存器（ACR）的配置值，允许在CAN控制器中对CAN标识符

进行硬件过滤（所有CAN消息处理的标准值：ACR：=16＃00000000）；NETNUMBER：网络号；

ENABLE：启用或禁用功能块的输入；

CONFIRM：通过功能块完成的消息输出；

ERROR：错误代码涉及数据类型“CAN_ERROR”，可能的错误代码定义如下：

0:无错误

1:其他错误

2:无效的网络号

3:无效的参数

4:无消息

5:不支持的波特率

6:初始化失败

7:设备繁忙

8:TX缓冲区溢出

9:无自由通道

10:COBID已被注册

11:指针类型不支持

描述：

功能块CAN_INIT用于初始化CAN接口。这需要停用该接口的CANopen功能。如果PLC通过WEBFrontend支持配置，通常可以将相关接口的“使能状态”设置为“禁用”。因此，接口并没有为CANopen自动初始化，可通过PLC在程序中调用功能块实现功能。

在输入BITRATE时，Bitrate的值必须以Bit/s的形式指定，例如125000为125

kBit/s。输入AMR（屏蔽寄存器）和ACR（代码寄存器）转移到CAN控制器的AMR或ACR寄存器后两个值的组合是一个接收过滤器，只能传递满足过滤条件的带CAN标识符的

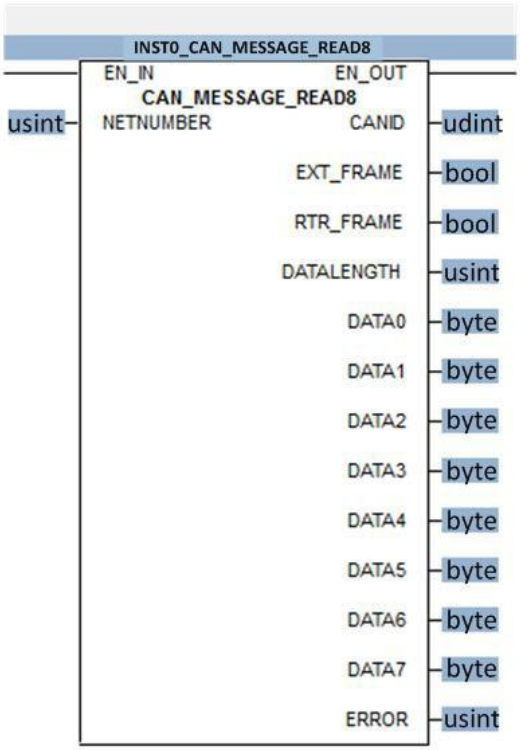
CAN消息。通过AMR和ACR的适当设置，所有不相关的干扰消息均可从CAN控制器中的接收中排除，减少控制器的CPU负载，并可以防止接收缓冲区的溢出。各个CAN控制器可以从各自的相关控制表中获取AMR和ACR的相关性以及实现的过滤器值。通常情况下，滤波器以CAN-Controller的方式接收所有CAN消息。为此，AMR和ACR必须设置如下：AMR:=16＃FFFFFFFF;ACR:=16＃00000000;

示例程序：

```
VAR canCH:USINT;

inst0_CAN_INIT:CAN_INIT;

mCanConfirm:BOOL; mCanInit:BOOL:=false; xERROR:USINT;

END_VAR

canCH:=2;

if not mCanInit then
```

```
inst0_CAN_INIT(BITRATE :=0 , AMR :=0 , ACR :=0 , NETNUMBER :=canCH ,
ENABLE :=true |     mCanConfirm:= CONFIRM, xERROR := ERROR);
if xERROR=0 then
    mCanInit:=true;
end_if;
end_if;
```

3.3.13 CAN_MESSAGE_READ8



用于读取接收的CAN消息（在功能块输出端传输数据）

操作数的定义：

NETNUMBER：网络号；

ENABLE：启用或禁用功能块的输入；CANID：CAN标识符

EXT_FRAME：TRUE：发送CAN扩展帧（29位帧ID）；FALSE：发送CAN标准帧（11位帧ID）；

RTR_FRAME：TRUE：发送远程帧；FALSE：发送数据帧；

DATA0-DATA7：CAN-Message接收到的数据字节；DATALENGTH：CAN-Message接收的长度；

CONFIRM：通过功能块完成的消息输出；

ERROR：数据类型“CAN_ERROR”的错误代码，可能的错误代码定义如下：

0:无错误

1:其他错误

2:无效的网络号

3:无效的参数

4:无消息

5:不支持的波特率

6:初始化失败

7:设备繁忙

8:TX缓冲区溢出

9:无自由通道

10:COBID已被注册

11:指针类型不支持

描述：

功能块CAN_MESSAGE_READ8用于从CAN接口的接收缓冲区中读取接收到的CAN消息。数据在功能块输出端逐字节传输。输出DATALENGTH的显示值为有效数据字节数（从DATA0开始）。

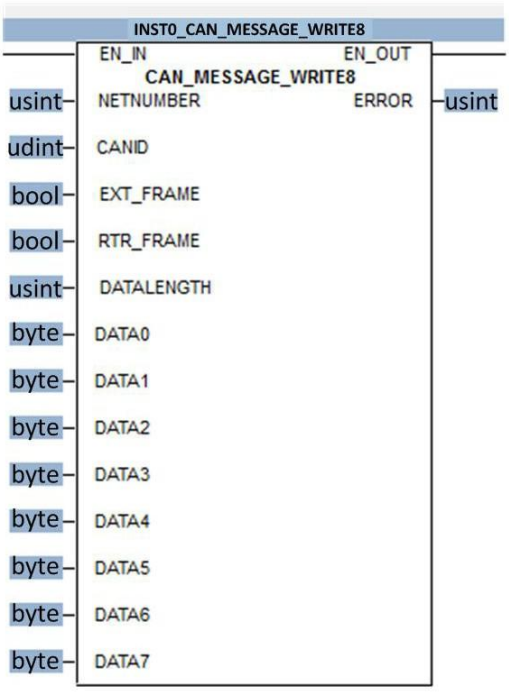
如果在功能块返回期间输出CONFIRM被置为TRUE，则数据0到数据7包含消息的单个字节，输出数据长度显示有效的字节数（从数据0开始），虽然此时CONFIRM已复位为FALSE，但CAN接口不包含消息，在CONFIRM的帮助下，可以区分是否接收到长度为0字节的有效消息，如果没有消息可用，则通过输出ERROR处的错误代码显示。

示例程序：

示例程序显示了功能块CAN_MESSAGE_READ8和功能块CAN_MESSAGE_WRITE8的联合工作过程。

首先，例程通过调用功能块CAN_MESSAGE_READ8来读取CAN接口消息。在消息完全读取后，再通过功能块CAN_MESSAGE_WRITE8将其重写到接口上(参考3.3.13节)。

3.3.14 CAN_MESSAGE_WRITE8



用于将需要发送的CAN信息写到CAN总线上

操作数的定义：

CANID：要发送的CAN消息的标识符；

EXT_FRAME：TRUE：CAN标识符标记扩展帧（29位）；FALSE：CAN标识符标记标准帧（11位）；

RTR_FRAME：TRUE：CAN对象作为RTR-Frame传递；FALSE：CAN对象不作为RTR-Frame传递；

CHANNEL：如果CAN对象已经被功能块CAN_DEFINE_CANID定义，则此功能块返回的通道号为CAN对象传递。如果CAN对象已经被CAN_DEFINE_CANID_RANGE定义了功能，那么0必须被传递（注意：通过CAN_DEFINE_CANID_RANGE定义的对象不能用于RTR-消息）；

DATA0-DATA7：要发送的CAN消息的数据字节；

DATALENGTH：要发送的CAN消息的长度；

NETNUMBER：网络号；

ENABLE：启用或禁用功能块的输入；

CONFIRM：通过功能块完成的消息输出；

ERROR：数据类型“CAN_ERROR”的错误代码，可能的错误代码定义如下：

0:无错误

1:其他错误

2:无效的网络号

3:无效的参数

4:无消息

5:不支持的波特率

6:初始化失败

7:设备繁忙

8:TX缓冲区溢出

9:无自由通道

10:COBID已被注册

11:指针类型不支持

描述：

功能块CAN_MESSAGE_WRITE8用于发送CAN数据。要发送的数据在功能块的输入端逐字节出现，以单字节的形式写在DATA0到DATA7中。输入DATALENGTH指定有效数据字节数（从DATA0开始）。当调用功能块CAN_MESSAGE_WRITE8时，要发送的消息存储在CAN接口的发送缓冲区中。如果没有发生错误（消息正确存储在发送缓冲区），功能块输出CONFIRM为TRUE。

以下示例程序显示功能块CAN_MESSAGE_WRITE8的应用。首先，调用块

CAN_MESSAGE_READ8从CAN接口读取数据。数据完全读取后，CAN_MESSAGE_WRITE8将其重写到接口上。该示例程序集成了初始化CAN接口和程序的流程控制。

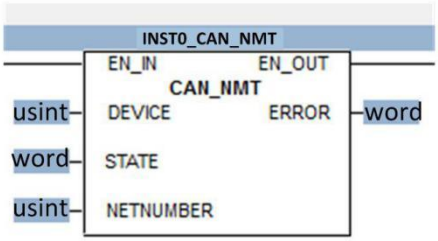
示例程序：

```
VAR canCH:USINT;
```

```
inst0_CAN_INIT:CAN_INIT;
mCanConfirm:BOOL; mCanInit:BOOL:=false; xERROR:USINT;
inst1_CAN_MESSAGE_READ8:CAN_MESSAGE_READ8;
inst2_CAN_MESSAGE_WRITE8:CAN_MESSAGE_WRITE8; mCANID:UDINT;
mEXT_FRAME:BOOL; mRTR_FRAME:BOOL; mDATA0:BYTE;
mDATA1:BYTE; mDATA2:BYTE; mDATA3:BYTE; mDATA4:BYTE; mDATA5:BYTE; mDATA6:BYTE;
mDATA7:BYTE; mDATALENGTH:USINT; mCONFIRM:BOOL; canID:UDINT;
canDAT0:Byte; canDAT1:Byte; canDAT2:Byte; canDAT3:Byte; canDAT4:Byte; canDAT5:Byte;
canDAT6:Byte; canDAT7:Byte; RecCount:INT; first:bool:=true; END_VAR


canCH:=2;
if not mCanInit then
inst0_CAN_INIT(BITRATE :=0 , AMR :=0 , ACR :=0 , NETNUMBER :=canCH ,
ENABLE :=true |     mCanConfirm:= CONFIRM, xERROR := ERROR); if xERROR=0 then
mCanInit:=true; end_if;
else
inst1_CAN_MESSAGE_READ8(
NETNUMBER :=canCH , ENABLE :=true    |     mCANID:= CANID,     mEXT_FRAME:= EXT_FRAME,
mRTR_FRAME:= RTR_FRAME, mDATA0:= DATA0,   mDATA1:= DATA1,     mDATA2:=
DATA2,
mDATA3:= DATA3,   mDATA4:= DATA4,     mDATA5:= DATA5,     mDATA6:= DATA6, mDATA7:=
DATA7,mDATALENGTH:= DATALENGTH,
mCanConfirm:= CONFIRM,
xERROR:= ERROR);
```

```
if mCanConfirm=1 then canID:=mCANID; canDAT0:=mDATA0; canDAT1:=mDATA1; canDAT2:=mDATA2;
canDAT3:=mDATA3; canDAT4:=mDATA4; canDAT5:=mDATA5; canDAT6:=mDATA6; canDAT7:=mDATA7;

RecCount:=RecCount+1; end_if;

if mCanConfirm=1 then mCANID:=mCANID+1;

inst2_CAN_MESSAGE_WRITE8(CANID := mCANID, EXT_FRAME := mEXT_FRAME, RTR_FRAME :=
mRTR_FRAME, CHANNEL := 1, DATA0 :=mDATA0 , DATA1 :=mDATA1 , DATA2 :=mDATA2 , DATA3 :=mDATA3 ,
DATA4 :=mDATA4 , DATA5 :=mDATA5 , DATA6 :=mDATA6 , DATA7 :=mDATA7 ,
DATALENGTH :=mDATALENGTH , NETNUMBER :=canCH , ENABLE := true |     mCONFIRM:= CONFIRM,

xERROR:= ERROR);

end_if;

(*

if first=true then

inst2_CAN_MESSAGE_WRITE8(CANID := 2, EXT_FRAME :=0, RTR_FRAME :=0,

CHANNEL := 1, DATA0 :=12 , DATA1 :=13 , DATA2 :=14 , DATA3 :=15 ,

DATA4 :=16 , DATA5 :=17 , DATA6 :=18 , DATA7 :=19 , DATALENGTH :=8 , NETNUMBER :=canCH ,
ENABLE := true |     mCONFIRM:= CONFIRM,xERROR:=

ERROR);

first:=false;

end_if;

*)

end_if;
```

3.3.15 CAN_NMT



用于发送NMT命令（网络管理命令）的功能块

操作数的定义：

ENABLE： 启用或禁用功能块的输入；

DEVICE： 需要控制的节点地址(1-127或0，0代表控制所有节点)；

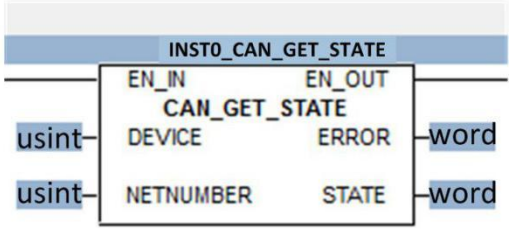
NETNUMBER： 网络号；

STATE： 节点状态；

CONFIRM： 通过功能块输出完成的消息，FALSE说明传输未成功完成或错误后终止，TRUE说明传输成功完成；

ERROR： 错误代码说明有关功能块执行结果的信息。

3.3.16 CAN_GET_STATE



用于请求设备节点状态的功能块

操作数的定义：

ENABLE： 启用或禁用功能块的输入；

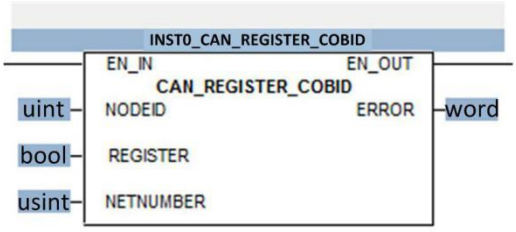
DEVICE： 需要请求的节点地址(1-127 或 0，0 代表请求所有节点)；

NETNUMBER： 网络号；

CONFIRM：通过功能块输出完成的消息，FALSE 说明传输未成功完成或错误后终止，TRUE 说明传输成功完成；

STATE：返回的当前节点状态。

3.3.17 CAN_REGISTER_COBID



用于注册或删除通过网络层接收的PDO和CAN第2层消息

操作数的定义：

EN_IN：启用或禁用功能块的输入；

NODEID：要输入到注册表或从注册表中删除的新信息的COBID（CAN标识符）

REGISTER：TRUE=输入COBID到注册表；FALSE=从注册表中删除COBID；

NETNUMBER：网络号（注：如果PLC只支持一路CANopen接口，该路输入即可跳过，因为变量数值根据IEC61131标准已经被设置成初始值0；

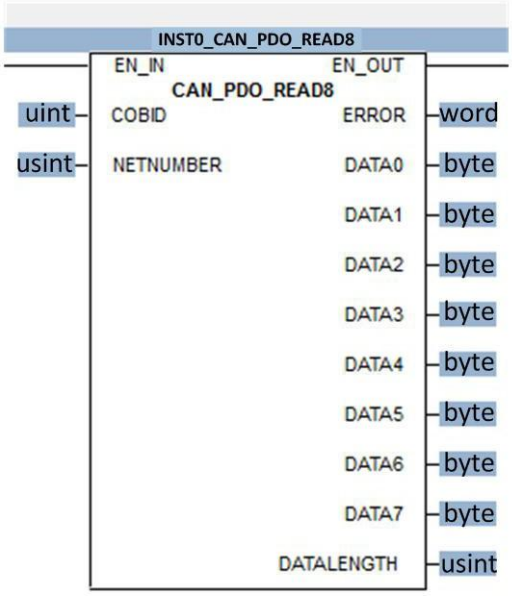
EN_OUT：功能块输出工作完成消息；

ERROR：功能块执行出错的报警信息

描述：

该功能块可以用于注册一个PDO或者CAN第二层的信息以供网络层接收，或者删除这样一个注册。当输入REGISTER为1时调用功能块，网络层接收到消息的指定COBID就会被注册，当输入REGISTER为0时调用功能块，相应COBID的注册表信息就会被删除。当REGISTER为0且COBID为0时调用功能块将删除所有的注册信息及所有储存在网络层缓冲区中的信息。事实上，网络层只支持通过调用功能块CAN_PDO_READ8访问PDO和CAN第二层消息并进行信息注册。

3.3.18 CAN_PDO_READ8



用于读取PDO数据的功能块

操作数的定义：

ENABLE：启用或禁用功能块的输入；

COBID：需要读取的PDO数据的COBID(即CAN帧ID)；

NETNUMBER：网络号；

DATA0-DATA7：接收的CAN数据；

DATALENGTH：接收的CAN数据长度；

ERRORINFO：预留；

CONFIRM：通过功能块输出完成的消息，FALSE说明传输未成功完成或错误后终止，TRUE说明传输成功完成；

ERROR：错误代码说明有关功能块执行结果的信息,可能的错误代码定义如下：

16#0000(= 00 dec)：没有错误

16#0001(= 01 dec)：其他错误

16#0002(= 02 dec)：数据溢出

16#0003(= 03 dec)：超时

16#0010(= 16 dec)：CAN 总线关闭

16#0011(= 17 dec)：CAN 总线被动错误

16#0021(= 33 dec)：预留（错误）

16#0022(= 34 dec)：无效的功能块

16#0023(= 35 dec)：没有主站模式

16#0024(= 36 dec)：无效的设备

16#0025(= 37 dec)：转发拥堵（transfer busy）

16#0030(= 48 dec)：没有可用的 SDO 通道

16#0031(= 49 dec)：SDO 拥堵

16#0032(= 50 dec)：SDO 初始化错误

16#0033(= 51 dec)：SDO 长度错误

16#0034(= 52 dec)：SDO 错误（ERRORINFO 中的 SDO 中止代码）

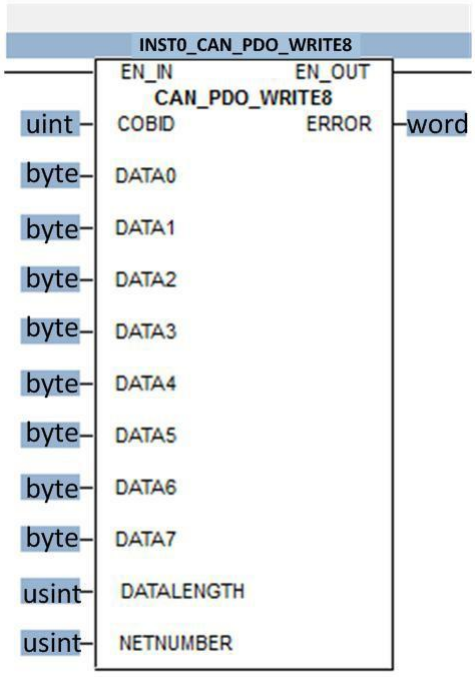
16#0040(= 64 dec)：没有有效的数据

16#0041(= 65 dec)：此 COBID 已被注册

16#0042(= 66 dec)：没有可用的 COBID 表入口

16#0043(=	67	dec)：没有此 COBID 被注册
16#0044(=	68	dec)：没有可用的接收通道
16#0045(=	69	dec)：CAN 数据长度不得为 0

3.3.19 CAN_PDO_WRITE8



用于发送PDO数据的功能块

操作数的定义：

ENABLE：启用或禁用功能块的输入；

COBID：要发送PDO报文的COBID(即CAN帧ID)；

DATA0-DATA7：需要发送的数据；

DATALENGTH：需要发送的CAN数据长度；

NETNUMBER：网络号；

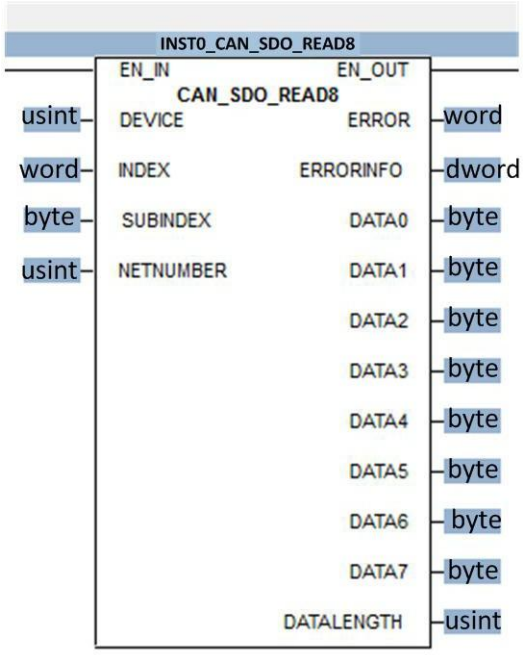
ERRORINFO：预留；

CONFIRM：通过功能块输出完成的消息，FALSE说明传输未成功完成或错误后终止，TRUE说明

传输成功完成；

ERROR：错误代码说明有关功能块执行结果的信息。

3.3.20 CAN_SDO_READ8



通过SDO（服务数据对象）传输，读对象字典功能块

操作数的定义：

ENABLE：启用或禁用功能块的输入；

DEVICE：需要读取的节点地址(1-127或0，0代表写入所有节点)；

SUBINDEX：需要读取的子索引项；

INDEX：需要读取的索引项；

NETNUMBER：网络号；

DATA0-DATA7：需要读取的数据；

DATALENGTH：需要读取的CAN数据长度；

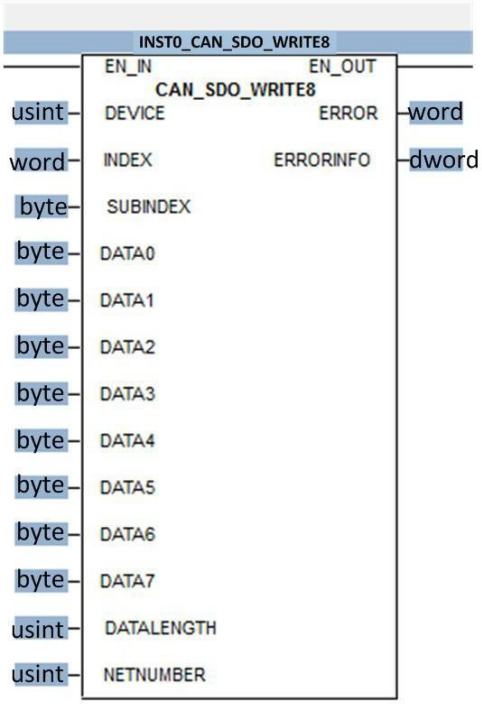
ERRORINFO：SDO中止代码；

CONFIRM：通过功能块输出完成的消息，FALSE说明传输未成功完成或错误后终止，TRUE

说明传输成功完成；

ERROR：错误代码说明有关功能块执行结果的信息。

3.3.21 CAN_SDO_WRITE8



通过SDO（服务数据对象）传输，写对象字典功能块

操作数的定义：

ENABLE：启用或禁用功能块的输入；

DEVICE：需要写入的节点地址(1-127 或 0，0 代表写入所有节点)；

SUBINDEX：需要写入的子索引项；

INDEX：需要写入的索引项；

NETNUMBER：网络号；

DATA0-DATA7：需要写入的数据；

DATALENGTH：需要写入的数据长度；

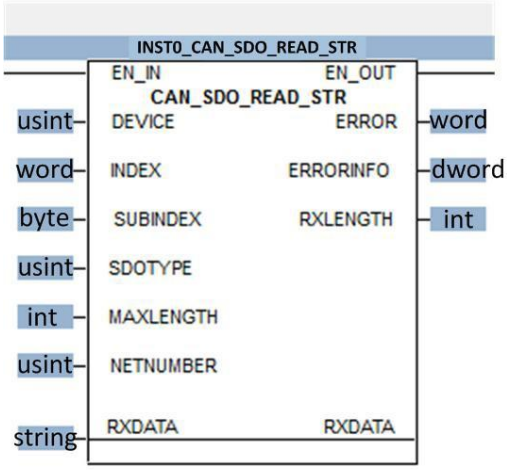
ERRORINFO：SDO 中止代码；

CONFIRM：通过功能块输出完成的消息，FALSE 说明传输未成功完成或错误后终止，TRUE 说

明传输成功完成；

ERROR：错误代码说明有关功能块执行结果的信息。

3.3.22 CAN_SDO_READ_STR



用于通过SDO传输读取节点对象字典中的字符串信息

操作数的定义：

EN_IN：启用或禁用功能块的输入；

DEVICE：要读取的节点地址（本地OD为1-127或0）；

INDEX：要读取的主索引号；

SUBINDEX：要读取的子索引号；

SDOTYPE：根据数据类型定的SDO传输模式类型；注：如果该输入没有明确的值（没有输入或未使用)，网络层会根据传输的数据量自动选择最合适的SDO传输模式。

MAXLENGTH：要读取的字符数限制；如果该值为0，则由系统内部自动确定传输字符串的缓冲长度,（注：OpenPCS中字符串的标准缓冲长度为32个字符）。

NETNUMBER：网络号；

RXDATA：用于接收读取字符的字符串变量；

EN_OUT：功能块功能完成输出消息；

ERROR：错误代码说明有关功能块执行结果的信息。

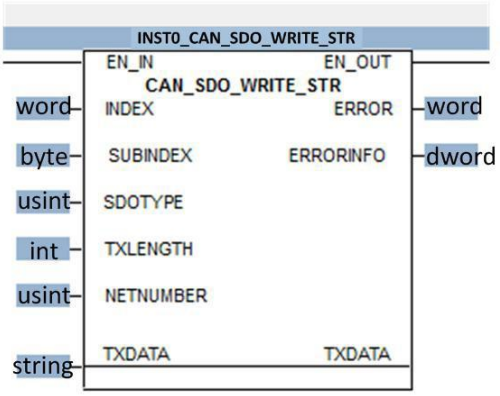
ERRORINFO：通信伙伴的SDO终止代码。

RXLENGTH：字符串信息的长度；

描述：

功能块使用SDO传输从节点的对象字典中读取到的字符串，SDO传输总是在后台进行。因此，3.1.3节中描述的程序必须通过参数ENABLE和CONFIRM来同步功能块和PLC程序。如果当功能块执行完成后输出的CONFIRM为1，则作为元素RXDATA传输的字符串中包含输入对象的字符串特征。输出RXLENGTH表示读取到的字符数量（等于LEN（RXDATA）），每一次网络层通过PLC程序只能支持有限数量的SDO传输（标准情况下最多5次传输，如果PLC手册中没有对输入引脚做任何更改），在通过设置ENABLE为1开始SDO传输后，相关的SDO通道会被关闭且不会被其他功能块使用，这种关闭的状态一直持续到SDO功能块通过令ENABLE为0重新调用。如果跳过令ENABLE为0调用功能块，则资源会一直保持关闭且不会被PLC使用，可通过令DEVICE为0调用功能块来访问PLC的本地对象字典，同样的方式也可以用来读取您本地的OD值。

3.3.23 CAN_SDO_WRITE_STR



用于通过SDO传输将字符串发送到相应节点的对象字典中

操作数的定义：

EN_IN：功能块使能输入；

INDEX：待发送数据的主索引号；

SUBINDEX：待发送数据的子索引号；

SDOTYPE：根据数据类型“CAN_SDO_Type”使用的SDO传输模式，注意：如果没有明确指定给

该输入的值（输入开放或未使用），功能块使用默认模式，网络层根据要传输的数据量，分别选择最合适的SDO传输方式。

TXLENGTH：要写入的字符数量，如果数量是0，则字符串TXDATA包含的对象字符的长度是由内部确定的（等于LEN（TXDATA）），并被用作要写入的字符数量。

NETNUMBER：网络号（注意：如果PLC只支持一路CANopen接口，则这一项输入的设置可以被跳过，因为根据IEC61131标准，变量的数值已经被设置为初值0。

TXDATA：要传递的字符串。

EN_OUT：功能块执行完成后的输出消息。

ERROR：根据数据类型CIA405_CANOPEN_KERNEL_ERROR生成的错误代码。

ERRORINFO：根据数据类型CIA405_SDO_ERROR生成的通讯对象SDO终止代码。

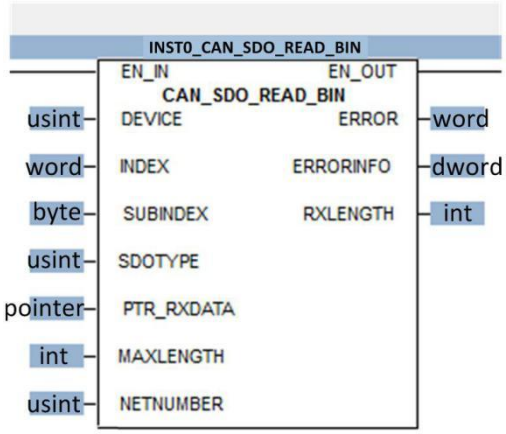
描述：

功能块CAN_SDO_WRITE_STR被用来通过SDO传输将字符串写入节点的对象字典中。

SDO传输通常在后台进行，因此4.1.3节中描述的程序必须通过使用ENABLE和CONFIRM来同步功能块和PLC程序，写入对象字典中的字符串必须被转移到元素TXDATA中。

输入TXLENGTH表示有效的字符数量。如果这一项值为0，则字符串TXDATA中包含的字符数量是由内部确定的，并被用作要写入的字符数量。在这种情况下，字符串包含的内容将全部被写入。网络层通过PLC程序在一次只能支持有限数量的SDO传输（标准：如果PLC手册并未对输入项更改，最多5次传输），在令ENABLE为1开始SDO传输后，各路SDO通道会被关闭，不会被其他功能块使用，这种关闭状态会一直持续到功能块再次通过输入ENABLE为0调用才会打开。如果跳过令ENABLE为0调用功能块，则资源将会永远关闭且不会再被PLC使用。PLC的本地对象字典可以通过令DEVICE为0调用功能块来访问。这种方式也可以用来将值写入您自己的对象字典中。

3.2.24 CAN_SDO_READ_BIN



功能块通过SDO传输读取节点对象字典中的二进制数据

操作数的定义：

EN_IN：功能块使能输入

DEVICE：读到的节点地址(1到127或本地对象字典0)

INDEX：读到的主索引号

SUBINDEX：读到的子索引号

SDOTYPE：使用的SDO传输模式（参照数据类型CAN_SDO_TYPE），注意：如果这一项输入没有明确值（输入开放或未被使用），功能块将使用SDO类型自动分配模式。网络层将根据传输数据的数量自动选择最合适的SDO传输模式。

PTR_RXDATA：读取到的数据字节的对象地址

MAXLENGTH：读取字节数的限制，如果值为0，则通过PTR_RXDATA定义地址的对象长度是由内部定义的并被用作读取的字符数（读取的最大字节数等于对象占据的字节数）

NETNUMBER：网络号（注意:如果PLC只支持1路CANopen接口，这一项的设置可以跳过，因为根据IEC61131标准，变量的数值已经被设置为初值0）。

EN_OUT：输出功能块执行结果

ERROR：参考数据类型CIA405_CANOPEN_KERNEL_ERROR生成的错误代码。

ERRORINFO：根据数据类型CIA405_SDO_ERROR生成的通讯对象SDO中断代码。

RXLENGTH：读取数据字节的数量

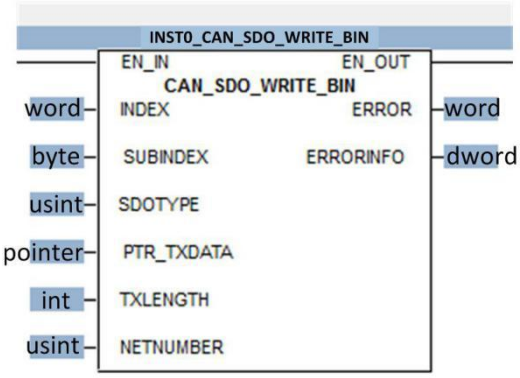
描述：

功能块CAN_SDO_READ_STR用来通过SDO传输读取节点对象字典中的二进制数据。

SDO传输通常在后台进行，因此4.1.3节中描述的程序必须通过使用ENABLE和CONFIRM来同步功能块和PLC程序。如果功能块执行结束后输出CONFIRM为1，则通过PTR_RXDATA定义地址的对象中包含要读取对象的二进制数据，输出RXLENGTH表示读取字节数的数量。网络层通过PLC程序在一次只能支持有限数量的SDO传输（标准：如果PLC手册并未对输入项更改，最多5次传输），在令ENABLE为1开始SDO传输后，各路SDO通道会被关闭，不会被其他功能块使用，这种关闭状态会一直持续到功能块再次通过输入

ENABLE为0调用才会打开。如果跳过令ENABLE为0调用功能块，则资源将会永远关闭且不会再被PLC使用。PLC的本地对象字典可以通过令DEVICE为0调用功能块来访问。这种方式也可以用来将值写入您自己的对象字典中。

3.3.25 CAN_SDO_WRITE_BIN



功能块用于通过SDO传输将二进制数据写入节点的对象字典中

操作数的定义：

EN_IN：功能块使能输入

INDEX：写入的主索引号

SUBINDEX：写入的子索引号

SDOTYPE：使用的SDO传输模式（参照数据类型CAN_SDO_TYPE），注意：如果这一项输入没有明确值（输入开放或未被使用），功能块将使用SDO类型自动分配模式。网络层将根据传输数据的

数量自动选择最合适的SDO传输模式。

PTR_TXDATA：要写入的二进制数据的对象地址

TXLENGTH：要写入的字节数，如果值为0，通过PTR_TXDATA定义地址的对象长度是由内部确定的并被用作要写入的字节数。

NETNUMBER：网络号（注意:如果PLC只支持1路CANopen接口，这一项的设置可以跳过，因为根据IEC61131标准，变量的数值已经被设置为初值0）。

EN_OUT：输出功能块执行结果

ERROR：参考数据类型CIA405_CANOPEN_KERNEL_ERROR生成的错误代码。

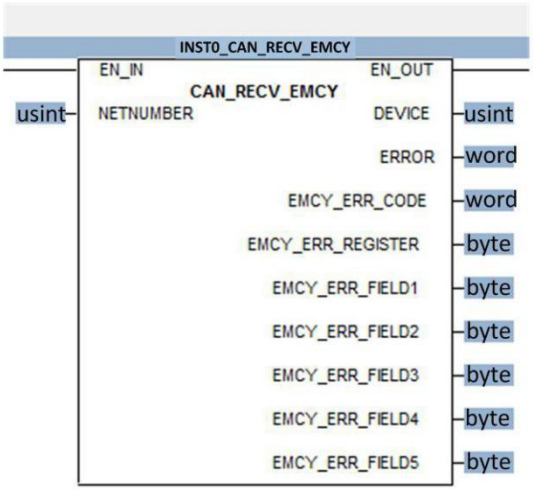
ERRORINFO：根据数据类型CIA405_SDO_ERROR生成的通讯对象SDO中断代码。

描述：

功能块CAN_SDO_WRITE_STR通过SDO传输将二进制数据写入节点对象字典中的。SDO传输通常在后台进行，因此3.1.3节中描述的程序必须通过使用ENABLE和CONFIRM来同步功能块和PLC程序。

要被写入二进制数据的对象字典所在的对象地址必须传递到参数PTER_TXDATA中，输入TXLENGTH指定有效的字节数，如果值为0，对象的长度会被内定并被用作要写入的字节数。网络层通过PLC程序在一次只能支持有限数量的SDO传输（标准：如果PLC手册并未对输入项更改，最多5次传输），在令ENABLE为1开始SDO传输后，各路SDO通道会被关闭，不会被其他功能块使用，这种关闭状态会一直持续到功能块再次通过输入ENABLE为0调用才会打开。如果跳过令ENABLE为0调用功能块，则资源将会永远关闭且不会再被PLC使用。PLC的本地对象字典可以通过令DEVICE为0调用功能块来访问。这种方式也可以用来将值写入您自己的对象字典中。

3.3.26 CAN_RECV_EMCY



用于读取网络层接收缓冲区中的任意节点紧急报文（Emergency）的功能块

操作数的定义：

ENABLE：启用或禁用功能块的输入；

NETNUMBER：网络号；

DEVICE：接收紧急报文（Emergency）的节点地址(1-127)；EMCY_ERR_CODE

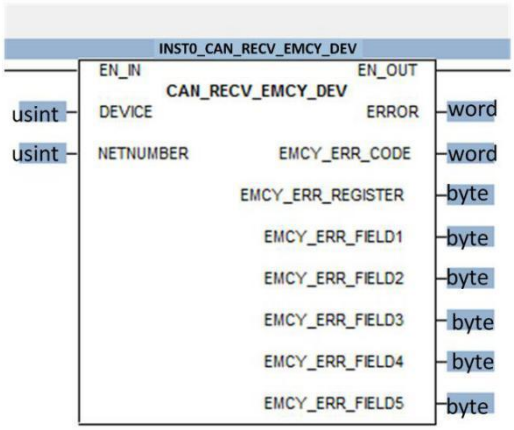
EMCY_ERR_REGISTER

EMCY_ERR_FIELD1-EMCY_ERR_FIELD5：紧急错误信息；

CONFIRM：通过功能块输出完成的消息，FALSE说明传输未成功完成或错误后终止，TRUE说明传输成功完成；

ERROR：错误代码说明有关功能块执行结果的信息。

3.3.27 CAN_RECV_EMCY_DEV



用于从网络层接收缓冲区读取特定节点紧急报文（Emergency）的功能块

操作数的定义：

ENABLE：启用或禁用功能块的输入；

DEVICE：需要控制的节点地址(1-127或0，0代表控制所有节点)；

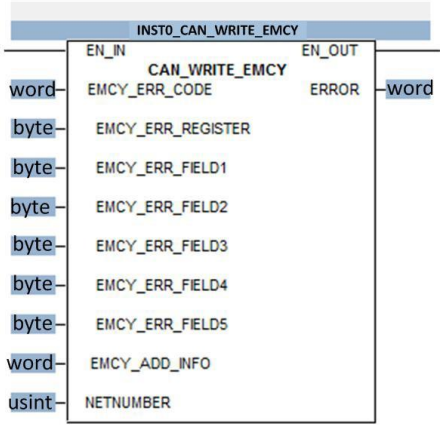
NETNUMBER：网络号；EMCY_ERR_CODEEMCY_ERR_REGISTER

EMCY_ERR_FIELD1EMCY_ERR_FIELD5：紧急错误信息；

CONFIRM：通过功能块输出完成的消息，FALSE说明传输未成功完成或错误后终止，TRUE说明传输成功完成；

ERROR：错误代码说明有关功能块执行结果的信息。

3.3.28 CAN_WRITE_EMCY



通过网络层，发送特定应用程序紧急报文（Emergency）的功能块

操作数的定义：

ENABLE：启用或禁用功能块的输入；

NETNUMBER：网络号； EMCY_ERR_CODE EMCY_ERR_REGISTER

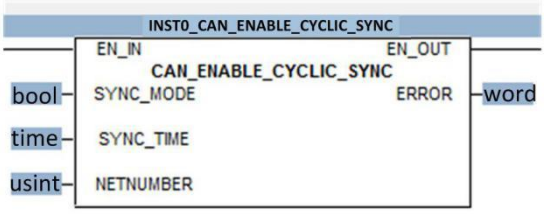
EMCY_ERR_FIELD1 - EMCY_ERR_FIELD5：紧急错误信息；

EMCY_ADD_INFO：额外的用户特定紧急错误信息，请注意，它不是需要发送的紧急错误消息的一部分，而是用于诊断目的，因此可以是 0。

CONFIRM：通过功能块输出完成的消息，FALSE 说明传输未成功完成或错误后终止，TRUE 说明传输成功完成；

ERROR：错误代码说明有关功能块执行结果的信息。

3.3.29 CAN_ENABLE_CYCLIC_SYNC



功能块用于激活循环SYNC同步信息

操作数的定义：

EN_IN：功能块使能输入

SYNC_MODE：TRUE=激活循环SYNC同步消息的生成,FALSE=停用循环SYNC同步消息

SYNC_TIME：两次连续的SYNC同步消息的时间间隔或0表示在每次PLC扫描周期后才发送SYNC同步消息。

NETNUMBER：网络号（注意:如果PLC只支持1路CANopen接口，这一项的设置可以跳过，因为根据IEC61131标准，变量的数值已经被设置为初值0）。

EN_OUT：输出功能块执行结果

ERROR：参考CIA405_CANOPEN_KERNEL_ERROR生成的错误代码。

描述：

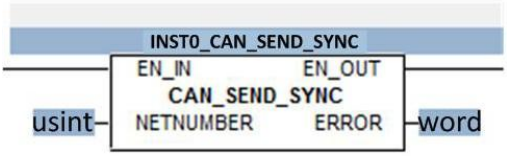
功能块CAN_ENABLE_CYCLIC_SYNC用来激活或停用循环SYNC同步消息，当激活后，如果上一次SYNC同步消息距离现在的时间间隔大于输入SYNC_TIME，则控制系统会在两次连续的PLC周期间生成一个SYNC同步消息，如果上一次SYNC同步消息距离现在的时间间隔小于输入SYNC_TIME,则不会有SYNC同步消息生成。如果输入值SYNC_TIME为0将会使控制系统以SYNC同步消息结束每个PLC扫描周期。在这种情况下，网络层的过程映像区中的信息交流将同时在PLC的本地输入、输出过程映像区中进行。

在每个PLC扫描周期末，控制系统都会检查时间明细来发送一个SYNC同步消息，因此输入SYNC_TIME指定的时间就是两个连续的SYNC同步消息之间的最小间隔，在误差最大的系统中，两个连续SYNC同步消息的实际时间间隔会随着PLC扫描周期的时间而变化:

SYNC实际时间:=SYNC_TIME输入时间+PLC扫描周期

该功能块只可被用于CANopen主站控制系统中，并可作为功能块CAN_SEND_SYNC的可替换功能块使用。

3.3.30 CAN_SEND_SYNC



用于发送独立的同步报文（SYNC）的功能块

操作数的定义：

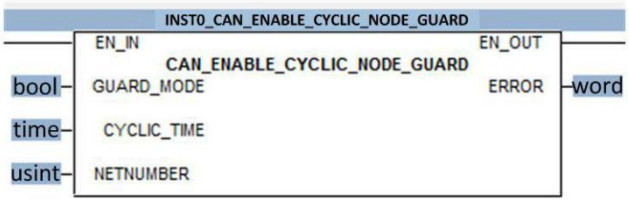
ENABLE：启用或禁用功能块的输入；

NETNUMBER：网络号；

CONFIRM：通过功能块输出完成的消息，FALSE说明传输未成功完成或错误后终止，TRUE说明传输成功完成；

ERROR：错误代码说明有关功能块执行结果的信息。

3.3.31 CAN_ENABLE_CYCLIC_NODE_GUARD



用于激活或停用循环同步报文（SYNC）的功能块

操作数的定义：

ENABLE： 启用或禁用功能块的输入；

NETNUMBER： 网络号；

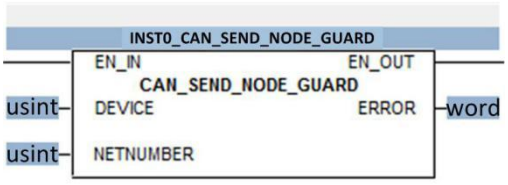
SYNC_MODE： TRUE=激活循环同步报文，FALSE=禁用循环同步报文；

SYNC_TIME： 两个连续同步报文之间的时间间隔；

CONFIRM： 通过功能块输出完成的消息，FALSE 说明传输未成功完成或错误后终止，TRUE 说明传输成功完成；

ERROR： 错误代码说明有关功能块执行结果的信息。

3.3.32 CAN_SEND_NODE_GUARD



使能CANopen从站发送节点保护

操作数的定义：

EN_IN： 功能块使能输入；

DEVICE： CANopen从站节点号；

NETNUMBER： PLC的CANopen端口号；

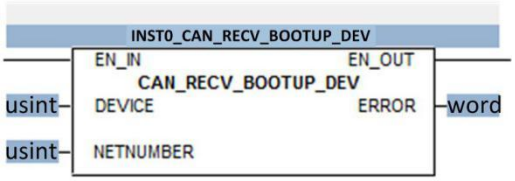
EN_OUT： 功能块使能输出；

ERROR： 错误代码说明了有关功能块执行结果的信息；

描述：

本功能块用于使能CANopen从站发送节点保护报文。

3.3.33 CAN_RECV_BOOTUP_DEV



用于从网络层的接收缓冲区中，读取特定节点Bootup消息的功能块

操作数的定义：

ENABLE：启用或禁用功能块的输入；

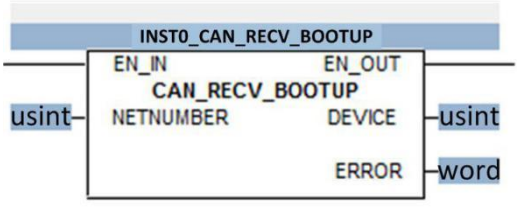
NETNUMBER：网络号；

DEVICE：用来检查Bootup消息接收的节点号(1-127)；

CONFIRM：通过功能块输出完成的消息，FALSE说明传输未成功完成或错误后终止，TRUE说明传输成功完成；

ERROR：错误代码说明有关功能块执行结果的信息。

3.3.34 CAN_RECV_BOOTUP



用于从网络层的接收缓冲区中，读取总线节点Bootup消息的功能块

操作数的定义：

ENABLE：启用或禁用功能块的输入；

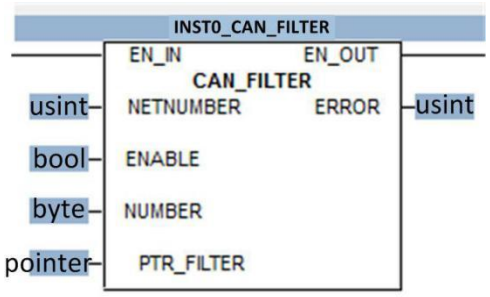
NETNUMBER： 网络号；

DEVICE： 收到Bootup消息的节点的地址(1-127)；

CONFIRM： 通过功能块输出完成的消息，FALSE说明传输未成功完成或错误后终止，TRUE说明传输成功完成；

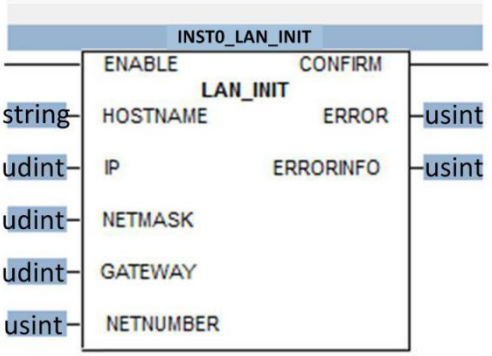
ERROR： 错误代码说明有关功能块执行结果的信息。

3.3.35 CAN_FILTER



用于CAN总线滤波

3.3.36 LAN_INIT



用于初始化以太网接口

操作数的定义：

ENABLE： 启用或禁用功能块，true为启用，false为禁用；

HOSTNAME： 设置主机名称，例如：’PLCCore’；

IP： 规定以太网端的IP地址，例如：’192.168.1.30’；

NETMASK： 规定以太网端的子网掩码，例如：’255.255.255.0’；

GATEWAY：规定以太网端的网关，例如：’192.168.1.1’；

NETNUMBER：网络号；

CONFIRM：false为功能块执行失败，true为功能块执行成功；

ERROR：错误代码说明了有关功能块执行结果的信息。可能的错误代码定义如下：

0:在执行功能块时没有发生错误

1:在执行功能块时发生硬件错误

2:填入的主机名称不支持

4:填入的IP地址不支持

8:填入的子网掩码不支持

16:填入的网关不支持

32:所选择的以太网接口不支持

ERRORINFO：二级错误信息。

描述：

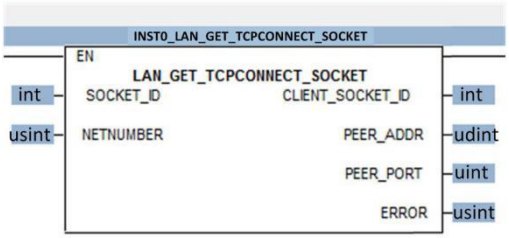
功能块可使用指定的参数初始化以太网接口。执行时如果出现错误，功能块会通过输出的ERROR进行显示，错误代码可查阅表15。以下示例程序显示功能块LAN_INIT初始化以太网接口的应用程序，设置以下参数：IP地址为192.168.1.30，子网掩码为255.255.255.0，网关192.168.1.1。

示例程序：

```
VAR
lanInit:bool:=false; inst0_LAN_INIT:LAN_INIT;
mConfirm:bool; mError:usint; mErrorinfo:usint; mIP:udint; mNetMask:udint; mGateWay:udint;
mSocket:int;
END_VAR
if lanInit=false then
mIP := LAN_ASCII_TO_INET('192.168.1.30'); mNetMask := LAN_ASCII_TO_INET('255.255.255.0');
```

```
mGateWay := LAN_ASCII_TO_INET('192.168.1.0');
    inst0_LAN_INIT(ENABLE :=true , HOSTNAME := 'PLCCore', IP := mIP, NETMASK :=mNetMask,
GATEWAY :=mGateWay ,NETNUMBER:=1
    | mConfirm:= CONFIRM,    mError:= ERROR, mErrorinfo:= ERRORINFO); lanInit:=true;
    end_if;
```

3.3.37 LAN_GET_TCPCONNECT_SOCKET



用于建立TCP端的socket连接

操作数的定义

SOCKET_ID：生成的socket索引值；

NETNUMBER：网络号；

CLIENT_SOCKET_ID:对端socket的索引值；

PEER_ADDR:对端的地址，和对端处于连接状态时可获得；

PEER_PORT:对端的端口号，和对端处于连接状态时可获得；

ERROR：错误代码说明了有关功能块执行结果的信息。可能的错误代码定义如下：

0：在执行功能块时没有发生错误

1：在执行功能块时发生硬件错误

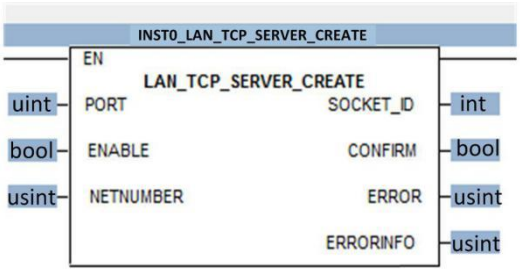
2：填入的socket索引值不支持

4：选择的以太网接口不支持

描述：

功能块可使用指定的参数初始化以太网接口。执行时如果出现错误，功能块会通过输出的ERROR进行显示，错误代码可查阅表16。以下示例程序显示功能块LAN_INIT初始化以太网接口的应用程序，设置以下参数：IP地址为192.168.1.30，子网掩码为255.255.255.0，网关192.168.1.1。

3.3.38 LAN_TCP_SERVER_CREATE



用于初始化以太网TCP连接的SERVER端

操作数的定义：

PORT：要使用的以太网端口号；

ENABLE：启用或禁用功能块，true为启用，false为禁用；

NETNUMBER：网络号；

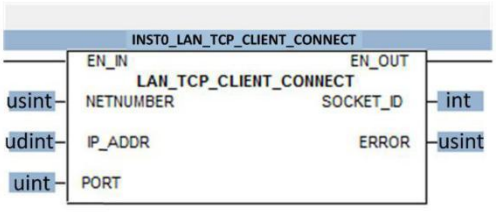
SOCKET_ID：生成的socket索引值；

CONFIRM：false为功能块执行失败，true为功能块执行成功；

ERROR：错误代码说明了有关功能块执行结果的信息。

ERRORINFO：二级错误信息。

3.3.39 LAN_TCP_CLIENT_CONNECT



用于建立TCP_CLIENT连接

操作数的定义：

EN_IN：功能块使能输入；

NETNUMBER：网络号；

IP_ADDR：IP地址；

PORT：端口号；

EN_OUT：功能块使能输出；

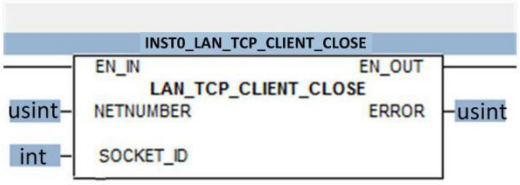
SOCKET_ID：SOCKET连接的ID；

ERROR：错误代码说明了有关功能块执行结果的信息

描述：

本功能块用于新建TCP_CLIENT的socket连接。

3.3.40 LAN_TCP_CLIENT_CLOSE



用于关闭TCP_CLIENT连接

操作数的定义：

EN_IN：功能块使能输入；

NETNUMBER：网络号；

SOCKET_ID：SOCKET连接ID；

EN_OUT：功能块使能输出；

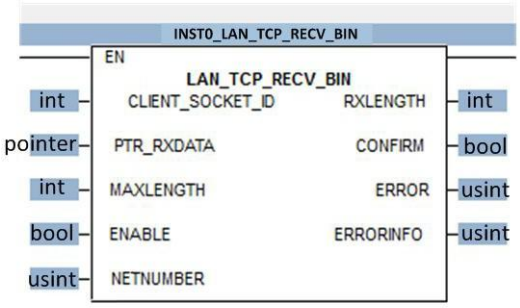
ERROR：错误代码说明了有关功能块执行结果的信息；

描述：

本功能块用于关闭TCP_CLIENT的socket连接。

本功能块执行时需要预先通过功能块LAN_TCP_CLIENT_CONNECT开启socket连接。

3.3.41 LAN_TCP_RECV_BIN



用于从TCP端口读取二进制字符流

操作数的定义：

CLIENT_SOCKET_ID：对端socket的索引值；

PTR_RXDATA：用于接收读取字节对象的地址；

MAXLENGTH：要读取的字节数限制，如果为0，则对象的长度由PTR_RXDATA确定

ENABLE：启用或禁用功能块，true：启用，false：禁用；

NETNUMBER：网络号；

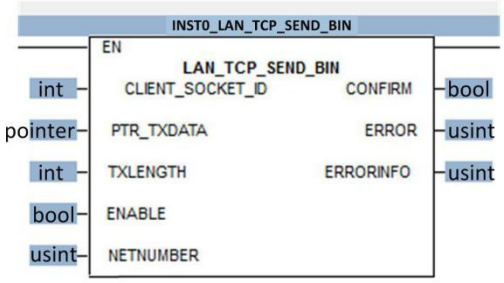
RXLENGTH：读取字符数（如果ERROR：=0）；

CONFIRM：false：功能块执行失败；true：功能块执行成功；

ERROR：错误代码说明了有关功能块执行结果的信息；

ERRORINFO：二级错误信息。

3.3.42 LAN_TCP_SEND_BIN



用于将二进制字符流写入TCP端口

操作数的定义：

CLIENT_SOCKET_ID:对端socket的索引值；

PTR_RXDATA：用于接收读取数据字节对象的地址；

TXLENGTH：要写入的字符数，如果值为0，则字符的长度及字符串中包含的字符TXDATA是内部确定的（等于LEN（TXDATA））

ENABLE：启用或禁用功能块，true：启用，false：禁用；

NETNUMBER：网络号；

CONFIRM：false：功能块执行失败，true：功能块执行成功；

ERROR：错误代码说明了有关功能块执行结果的信息；

ERRORINFO：二级错误信息；

示例程序：

```
VAR
lanInit:bool:=false; inst0_LAN_INIT:LAN_INIT;
mConfirm:bool; mError:usint;mErrorinfo:usint; mIP:udint;mNetMask:udint;
mGateWay:udint;mSocket:INT; inst4_LAN_TCP_SERVER_CREATE:LAN_TCP_SERVER_CREATE;
mTcpCreateOK:bool:=false; mClientSocket:INT; mPeer:udint; mPeerPort:uint;
inst5_LAN_GET_TCPCONNECT_SOCKET:LAN_GET_TCPCONNECT_SOCKET;
inst6_LAN_TCP_RECV_BIN:LAN_TCP_RECV_BIN; inst7_LAN_TCP_SEND_BIN:LAN_TCP_SEND_BIN;
mRxLen:int;
abDataBuffer : ARRAY[0..1000] OF BYTE;
pDataObject : POINTER;
END_VAR


if lanInit=false then
```

mIP := LAN_ASCII_TO_INET('192.168.1.30'); mNetMask := LAN_ASCII_TO_INET('255.255.255.0'); mGateWay := LAN_ASCII_TO_INET('192.168.1.0');

inst0_LAN_INIT(ENABLE :=true , HOSTNAME := 'PLCCore',IP := mIP, NETMASK :=mNetMask,GATEWAY :=mGateWay ,NETNUMBER:=1

|mConfirm:= CONFIRM,mError:= ERROR, mErrorinfo:= ERRORINFO); lanInit:=true;

else

if mTcpCreateOK=false then

inst4_LAN_TCP_SERVER_CREATE(PORT :=8089 ,ENABLE := true, NETNUMBER :=

1|

mSocket:= SOCKET_ID, mConfirm:= CONFIRM, mError:= ERROR, mErrorinfo:= ERRORINFO);

mTcpCreateOK:=true; end_if;

end_if; pDataObject:=&abDataBuffer; if mTcpCreateOK=true then

inst5_LAN_GET_TCPCONNECT_SOCKET(SOCKET_ID := mSocket,

NETNUMBER := 1| mClientSocket:= CLIENT_SOCKET_ID, mPeer:= PEER_ADDR, mPeerPort:= PEER_PORT, mError:= ERROR);

if mClientSocket>=0 then

inst6_LAN_TCP_RECV_BIN(CLIENT_SOCKET_ID :=mClientSocket , PTR_RXDATA :=pDataObject , MAXLENGTH := 1500, ENABLE :=1 , NETNUMBER := 1|mRxLen:= RXLENGTH, mConfirm:= CONFIRM, mError:= ERROR,

mErrorinfo:= ERRORINFO); if mRxLen>0 then

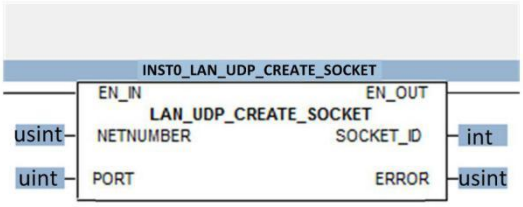
inst7_LAN_TCP_SEND_BIN(CLIENT_SOCKET_ID :=mClientSocket , PTR_TXDATA :=pDataObject , TXLENGTH := mRxLen, ENABLE :=1 ,

NETNUMBER :=1| mConfirm:= CONFIRM, mError := ERROR, mErrorinfo:= ERRORINFO);

end_if; end_if;

end_if;

3.3.43 LAN_UDP_CREATE_SOCKET



可建立一个UDP的socket连接，用于发送和接收数据

操作数的定义：

PORT：需要使用的以太网端口号；

ENABLE：启用或禁用功能块，true为启用，false为禁用；

NETNUMBER：网络号；

SOCKET_ID：生成的socket索引值（由PLC的UDP层在内部分配引用）；

CONFIRM：false：功能块执行失败，true：功能块执行成功；

ERROR：错误代码说明了有关功能块执行结果的信息。

ERRORINFO：二级错误信息。

描述：

功能块LAN_UDP_CREATE_SOCKET可建立一个UDP的socket连接，用于发送和接收数据。如果连接被用作接收数据，在输入PORT处必须为有效的端口号。只有在端口有效的前提下，PLC才会在UDP层调用内部函数，才能够在与其端口号匹配的IP地址处接收到数据包，在大多数系统中，小于1024的端口号只被内部特权程序使用，范围在1024到49151的端口号仍为默认应用保留并被IANA（互联网号码分配机构）管理，因此最好使用从49152到65535的端口号来进行PLC的UDP通讯。

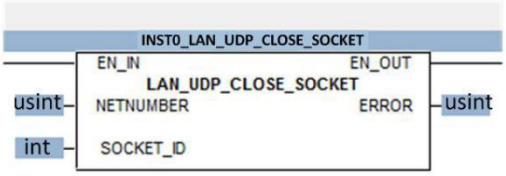
如果创建的连接只用来发送数据，则端口号是任意的，如果输入PORT设为0，PLC会在UDP层的内部专用空间调用一个空闲的端口号来发送数据，此时UDP层调用的内部函数就不是必要的，但是，被定义用来发送数据的端口号的说明却是必不可少的，例如网络中的活动防火墙只将数据转发到特定端口。

在功能块正确执行后，PLC会在内部引用UDP层并在输出处生成SOCKETID，这个SOCKETID的值会

在随后调用的相关功能块中进行引用，例如功能块LAN_UDP_SENDTO_STR或LAN_UDP_RECVFROM_STR。

不再需要的socket连接可通过功能块LAN_UDP_CLOSE_SOCKET来关闭。

3.3.44 LAN_UDP_CLOSE_SOCKET



用于关闭一个不再需要的UDP socket连接

操作数的定义：

SOCKET_ID：需要关闭的socket索引值；

ENABLE：启用或禁用功能块，true为启用，false为禁用；

NETNUMBER：网络号；

CONFIRM：false为功能块执行失败，true为功能块执行成功；

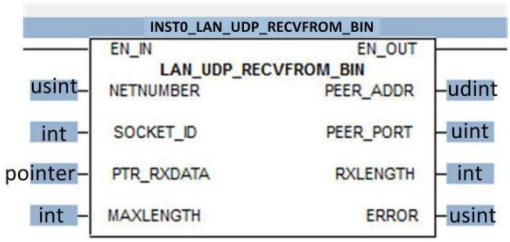
ERROR：错误代码说明了有关功能块执行结果的信息；

ERRORINFO：二级错误信息。

描述：

SOCKETID是在调用功能块LAN_UDP_CREATE_SOCKET时由PLC的UDP层在内部引用的，所有未关闭的socket连接在PLC程序退出时，都会自动关闭。

3.3.45 LAN_UDP_RECVFROM_BIN



用于从UDP层接收缓存区读取UDP数据包

操作数的定义：

SOCKET_ID：需要被轮询的 socket 索引值；

PTR_RXDATA：存放数据字节对象（接收）的地址；

MAXLENGTH：要读取的字节数限制；

ENABLE：启用或禁用功能块，true 为启用，false 为禁用；

NETNUMBER：网络号；

PEER_ADDR:对端（远程端）的地址，和对端处于连接状态时可获得；

PEER_PORT:对端（远程端）的端口号，和对端处于连接状态时可获得；

RXLENGTH：读取的字符数；

CONFIRM：false：功能块执行失败，true：功能块执行成功；

ERROR：错误代码说明了有关功能块执行结果的信息。

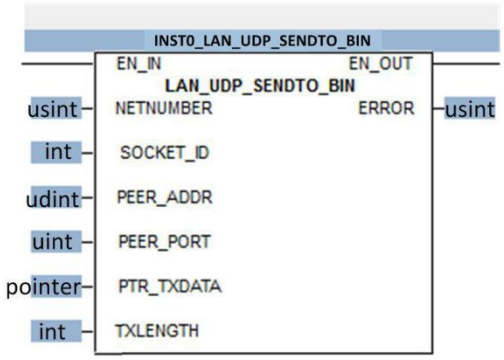
ERRORINFO：二级错误信息。

描述：

本功能块用于从 UDP 层的接收缓冲区中读取 UDP 数据包，功能块成功执行后，confirm 会被置为 TRUE。PTR_RXDATA 包含接收到的数据字节，输出 RXLENGTH 指定读取的数据字节数。如果功能块执行失败，则不会有数据字节输出。

当成功接收到数据包（CONFIRM 为 TRUE），输出 PEER_ADDR 和 PEER_PORT 用于存放对端（远程端）的 IP 地址与端口号，如果 PLC 对接收到的数据包作出响应，则对端地址和对端端口必须用作后续功能块调用的目标规范，本功能块执行时需要预先通过功能块 LAN_UDP_CREATE_SOCKET 开启 socket 连接。

3.3.46 LAN_UDP_SENDTO_BIN



用于发送UDP数据包

操作数的定义：

SOCKET_ID：用于发送的socket索引值；

PEER_ADDR：对端（远程端）的IP地址，和对端处于连接状态时可获得；

PEER_PORT：对端（远程端）的端口号，和对端处于连接状态时可获得；

PTR_TXDATA：要发送的二进制数据的对象的地址；

TXLENGTH：要发送的数据字节数；

ENABLE：启用或禁用功能块，true为启用，false为禁用；

NETNUMBER：网络号；

CONFIRM：false：功能块执行失败，true：功能块执行成功；

ERROR：错误代码说明了有关功能块执行结果的信息。

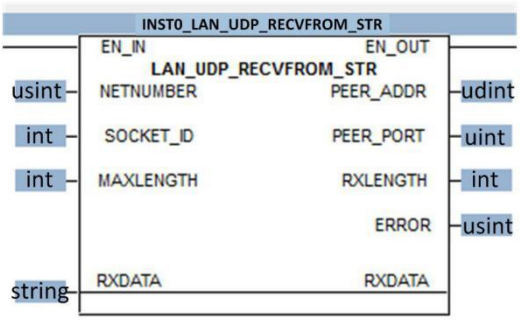
ERRORINFO：二级错误信息。

描述：

本功能块用于发送UDP数据包。PEER_ADDR和PEER_PORT用于设置对端（远程端）的IP地址与端口号，与功能块LAN_UDP_RECVFROM_BIN配套使用。

本功能块执行时需要预先通过功能块LAN_UDP_CREATE_SOCKET开启socket连接。

3.3.47 LAN_UDP_RECVFROM_STR



用于从UDP层的接收缓存区读取UDP数据包

操作数的定义：

RXDATA：用于接收读取到的字符串的变量；

SOCKET_ID：需要被轮询的socket索引值；

MAXLENGTH：要读取的字节数限制；

ENABLE：启用或禁用功能块，true为启用，false为禁用；

NETNUMBER：网络号；

RXDATA：用于接收读取到的字符串的变量；

PEER_ADDR:对端（远程端）的地址，和对端处于连接状态时可获得；

PEER_PORT:对端（远程端）的端口号，和对端处于连接状态时可获得；

RXLENGTH：读取字符串的长度；

CONFIRM：false：功能块执行失败，true：功能块执行成功；

ERROR：错误代码说明了有关功能块执行结果的信息。可能的错误代码在表27中进行了详细的说明；

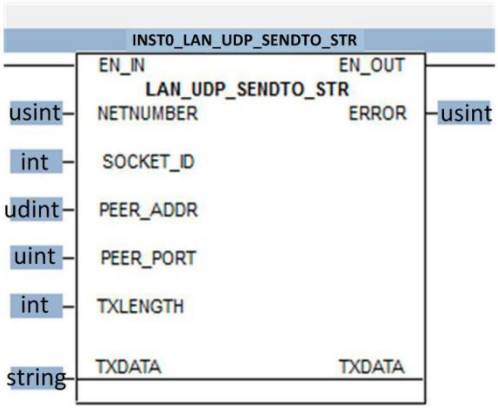
ERRORINFO：二级错误信息。

描述：

本功能块用于从UDP的接收缓冲区读取UDP数据包。PEER_ADDR和PEER_PORT用于存放对端（远程端）的IP地址与端口号。

本功能块执行时需要预先通过功能块LAN_UDP_CREATE_SOCKET开启socket连接。

3.3.48 LAN_UDP_SENDTO_STR



用于发送UDP数据包

操作数的定义：

TXDATA：要发送的字符串变量；

SOCKET_ID：用于发送的socket索引值；

PEER_ADDR：对端（远程端）的IP地址，和对端处于连接状态时可获得；

PEER_PORT：对端（远程端）的端口号，和对端处于连接状态时可获得；

TXLENGTH：要发送的数据字节数，如果数字为0，则对象的长度由PTR_TXDATA内部寻址确定；

ENABLE：启用或禁用功能块，true：启用，false：禁用；

NETNUMBER：网络号；

TXDATA：要发送的字符串变量；

CONFIRM：false：功能块执行失败，true：功能块执行成功；

ERROR：错误代码说明了有关功能块执行结果的信息。可能的错误代码定义如下：

0:在执行功能块时，没有发生错误

1:当调用函数块时，设置了无效的网络号

2:当调用函数块时，设置了一个无效的参数

3:在对PLC进行UDP初始化时出错

4:在新增、发送、或接收socket连接时，PLC的UDP层报错

5:没有可用的socket连接

6:设置了一个无效的socketID7socketID对应的socket无法正常使用

8:发送缓存区过大，数据包大小被最大字节数所限制

9:发送缓存区过小，没有数据可发送

10:设置的主机出现错误

11:指针指向了一个不支持的数据类型

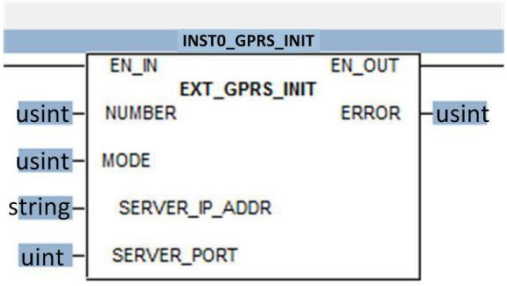
ERRORINFO： 二级错误信息。

描述：

本功能块用于发送UDP数据包。PEER_ADDR和PEER_PORT用于设置对端（远程端）的IP地址与端口号。

本功能块执行时需要预先通过功能块LAN_UDP_CREATE_SOCKET开启socket连接。

3.3.49 EXT_GPRS_INIT



扩展GPRS通讯初始化功能块

操作数的定义：

EN_IN： 功能块使能输入；

NUMBER： 网络号；

MODE： 通讯模式，1 为 TCP，2 为 UDP；

SERVER_IP_ADDR:服务器 IP 地址；

SERVER_PORT:服务器端口号；

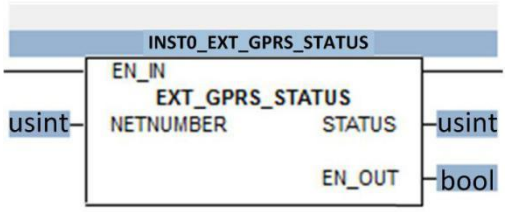
EN_OUT：功能块使能输出；

ERROR：错误代码说明了有关功能块执行结果的信息

描述：

本功能块用于初始化扩展 GPRS 通讯。

3.3.50 EXT_GPRS_STATUC



获取GPRS通讯状态功能块

操作数的定义：

EN_IN：功能块使能输入；

NETNUMBER：网络号；

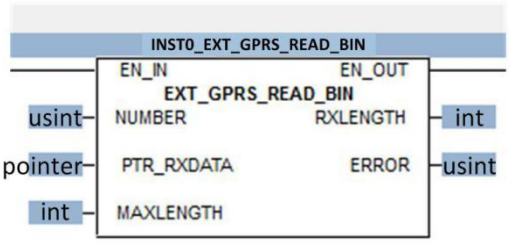
STATUS：GPRS通讯状态；

EN_OUT：功能块使能输出；

描述：

本功能块用于获取GPRS通讯状态。

3.3.51 EXT_GPRS_READ_BIN



GPRS通讯读取数据

操作数的定义：

EN_IN：功能块使能输入；

NETNUMBER：网络号；

PTR_RXDATA：读取数据后存放的数组指针；

MAXLENGTH：允许读取数据的最大长度；

EN_OUT：功能块使能输出；

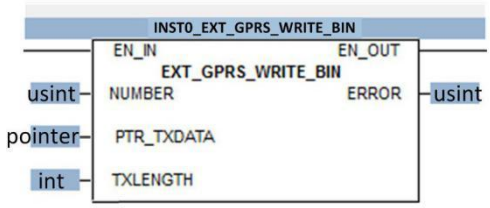
RXLENGTH：实际读取的数据长度；

ERROR：错误代码说明了功能块执行结果错误的信息；

描述：

本功能块用于GPRS通讯读取数据。

3.3.52 EXT_GPRS_WRITE_BIN



GPRS通讯发送数据

操作数的定义：

EN_IN：功能块使能输入；

NETNUMBER：网络号；

PTR_RXDATA：待发送数据存放的数组指针；

TXLENGTH：发送数据的长度；

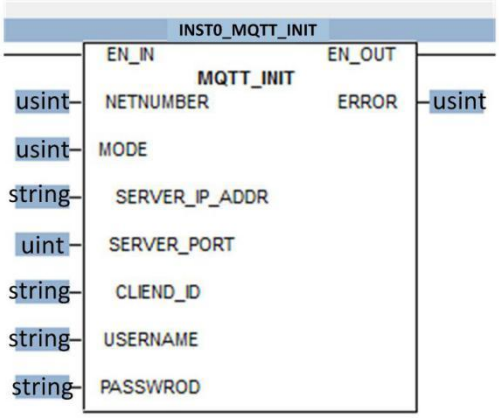
EN_OUT：功能块使能输出；

ERROR：错误代码说明了功能块执行结果错误的信息；

描述：

本功能块用于GPRS通讯发送数据。

3.3.53 MQTT_INIT



MQTT初始化功能块

操作数的定义：

EN_IN：功能块使能输入；

NETNUMBER：网络号；

MODE：功能模式；

SERVER_IP_ADDR：服务器IP地址；

SERVER_PORT：服务器端口号；

CLIEND_ID：客户端ID名称；

USERNAME：用户名；

PASSWROD：密码；

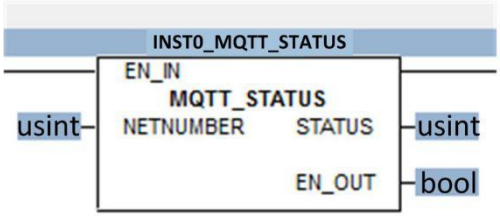
EN_OUT：功能块使能输出；

ERROR：错误代码说明了功能块执行结果错误的信息；

描述：

本功能块用于MQTT通讯初始化。

3.3.54 MQTT_STATUS



MQTT状态功能块

操作数的定义：

EN_IN：功能块使能输入；

NETNUMBER：网络号；

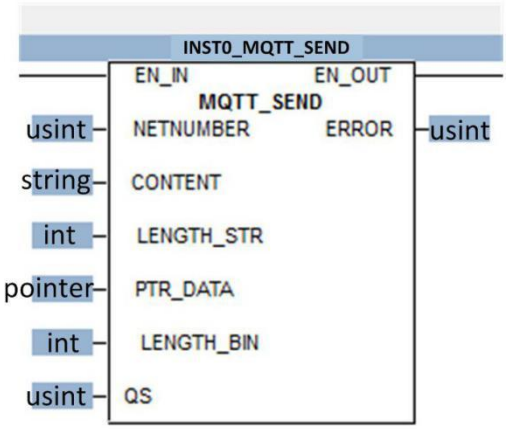
STATUS：MQTT 通讯状态；

EN_OUT：功能块使能输出；

描述：

本功能块用于获取 MQTT 状态。

3.3.55 MQTT_SEND



MQTT发送数据功能块

操作数的定义：

EN_IN：功能块使能输入；

NETNUMBER：网络号；

CONTENT：数据结构；

LENGTH_STR：待发送字符串长度；

PTR_DATA：待发送数据存放数组的指针；

LENGTH_BIN：待发送字符流的长度；

QS：字符流参数；

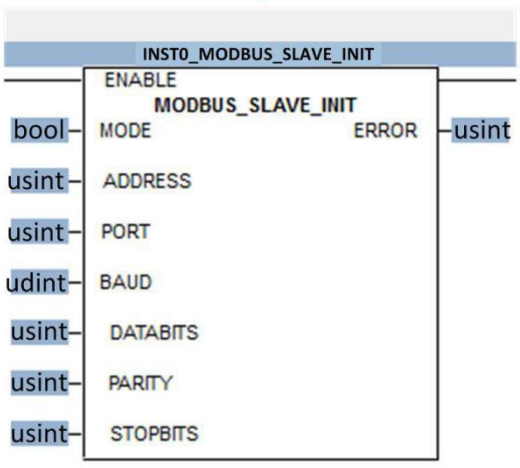
EN_OUT：功能块使能输出；

ERROR：错误代码说明了功能块执行结果的信息；

描述：

本功能块用于 MQTT 发送数据。

3.3.56 MODBUS_SLAVE_INIT



用于初始化Modbus-RTU从站接口

操作数的定义：

ENABLE：启用或禁用功能块的输入；

MODE：选择通讯协议：输入1值将Port定义为Modbus协议并使能该协议，输入值为0将Port定义为PPI并禁止Modbus协议。

ADDRESS：设置本站地址

PORT：要使用的串行接口定义号，1为RS232，2为RS485BAUD：设置波特率。1200、2400、4800、

9600、19200、38400、57600、115200

DATABITS：数据位数

PARITY：设置校验

0：无校验,1：奇校验,2：偶校验

STOPBITS：停止位

ERROR：错误代码说明功能块执行结果的信息。可能的错误代码定义如下：

0：无错误

1：存储区范围错误

2：非法波特率或校验

3：非法从站地址

4：Modbus参数的非法值

5：保持寄存器与Modbus从站符号地址重复

6：接收校验错误

7：接收CRC错误

8：非法功能请求/不支持的功能

9：请求中有非法存储区地址

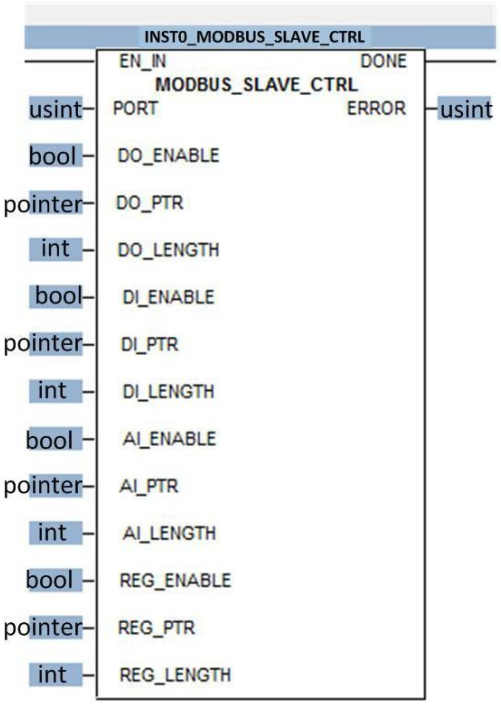
10：从站功能未使能

描述：

MODBUS_SLAVE_INIT指令用于使能和初始化或禁止Modbus通讯。

MODBUS_SLAVE_INIT指令必须无错误的执行，然后才能够使用MBUS_SLAVE指令。在继续执行下一条指令前，MODBUS_SLAVE_INIT指令必须执行完。MODBUS_SLAVE_INIT指令应该在每次通讯状态改变时只执行一次。因此，ENABLE输入端应使用边沿检测元素以脉冲触发，或只在第一个循环周期内执行一次。

3.3.57 MODBUS_SLAVE_CTRL



执行Modbus-RTU从站接口的控制指令

操作数的定义：

PORT：要使用的串行接口定义号；

DO_ENABLE：数字量输出使能；

DO_PTR：数字量输出存储地址；

DO_LENGTH：数字量输出长度；

DI_ENABLE：数字量输入使能；

DI_PTR：数字量输入存储地址；

DI_LENGTH：数字量输入数据长度；

AI_ENABLE：模拟量输入使能；

AI_PTR：模拟量输入存储地址；

AI_LENGTH：模拟量输入数据长度；

REG_ENABLE：寄存器使能；

REG_PTR：寄存器存储地址；

REG_LENGTH：寄存器存储数据长度；

DONE:完成标志位；

ERROR：根据数据类型“CAN_ERROR”的错误代码，可能的错误代码定义如下：

0:无错误

1:响应校验错误

2:未用

3:接收超时（从站无响应）

4:请求参数错误

5:Modbus/自由口未使能

6:Modbus正在忙于其它请求

7:响应错误（响应不是请求的操作）

8:响应CRC校验和错误

描述：

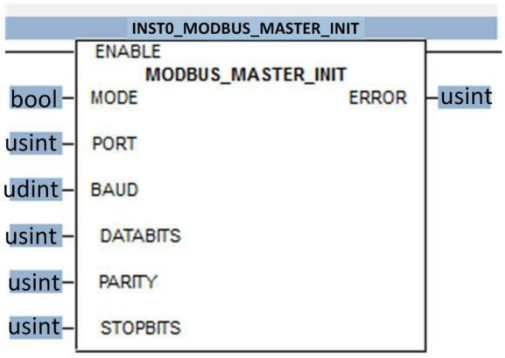
调用功能块MODBUS_SLAVE_CTRL可实现MODBUS-RTU从站的数据收发。

示例程序：

```
Var; inst1_MODBUS_SLAVE_CTRL:MODBUS_SLAVE_CTRL;
modbusDOBuf : ARRAY[0..5] OF byte; DO_Ptr : POINTER;
modbusDIBuf : ARRAY[0..5] OF byte; DI_Ptr : POINTER;
modbusAIBuf : ARRAY[0..10] OF int; AI_Ptr : POINTER;
modbusRegBuf : ARRAY[0..127] OF int; mPtr : POINTER;
xDONE:BOOL;
END_VAR
```

```
if xInitOk then mPtr:=&modbusRegBuf; DO_Ptr:=&modbusDOBuf; DI_Ptr:=&modbusDIBuf;
AI_Ptr:=&modbusAIBuf;

inst1_MODBUS_SLAVE_CTRL(DO_ENABLE :=1 ,DO_PTR :=DO_Ptr , DO_LENGTH :=5 , DI_ENABLE :=1 ,
DI_PTR :=DI_Ptr , DI_LENGTH :=5 , AI_ENABLE :=1 ,  AI_PTR :=AI_Ptr ,AI_LENGTH :=10 , REG_ENABLE :=1 ,
  REG_PTR :=mPtr , REG_LENGTH :=127 | xDONE := DONE, xERROR := ERROR);

end_if;
```

3.3.58 MODBUS_MASTER_INIT



用于初始化Modbus-RTU主站接口

操作数的定义：

ENABLE：启用或禁用功能块，true为启用，false为禁用；

MODE：模式选择；

PORT：端口号，1为232,2为485；

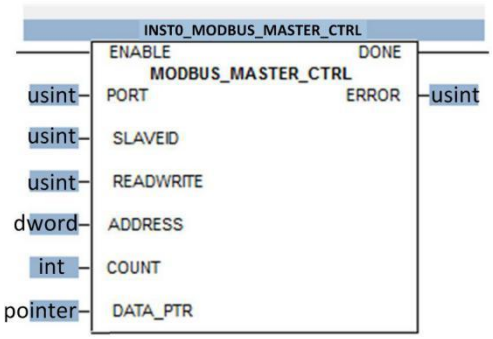
BAUD：波特率，设置波特率。1200、2400、4800、9600、19200、38400、57600、115200；

DATABITS：数据位数PARITY：设置校验

0：无校验，1：奇校验，2：偶校验STOPBITS：停止位

ERROR：错误代码说明功能块执行结果的信息。

3.3.59 MODBUS_MASTER_CTRL



执行Modbus-RTU主
站接口的控制指令（功能码）

操作数的定义：

ENABLE：功能块使能

PORT：要使用的串行接口定义号，1为232,2为485；

SLAVEID：Modbus_RTU从站的ID；

READWRITE：读写使能；

ADDRESS：从站地址；

COUNT：读取或写入的数量；

DATA_PTR：读取或写入的数组指针；

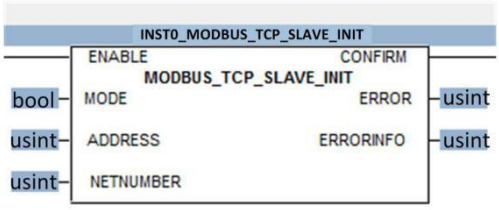
DONE：完成标志位；

ERROR：错误信息；

描述：

调用功能块MODBUS_MASTER_CTRL可实现MODBUS-RTU主站的数据收发。

3.3.60 MODBUS_TCP_SLAVE_INIT



用于初始化Modbus-TCP从站接口

操作数的定义：

ENABLE：启用或禁用功能块，true 为启用，false 为禁用；

MODE：模式选择；

ADDRESS：规定以太网 TCP 端的从站地址；

NETNUMBER：网络号；

CONFIRM：false：功能块执行失败，true：功能块执行成功；

ERROR：错误代码说明了有关功能块执行结果的信息。可能的错误代码定义如下:

0:在执行功能块时，没有发生错误

1:在执行功能块时，出现硬件错误

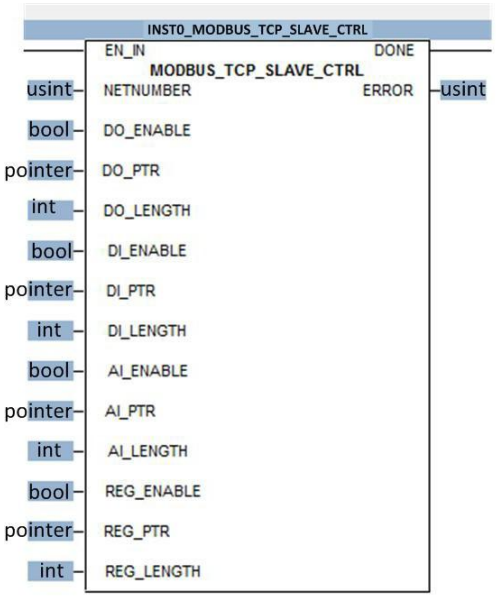
2:当调用函数块时，设置了无效的模式

4:设置的地址太大，超出了最大可用存储区域

8:当调用函数块时，设置了无效的以太网接口

ERRORINFO：二级错误信息

3.3.61 MODBUS_TCP_SLAVE_CTRL



用于执行Modbus-TCP从站接口的控制指令

操作数的定义：

NETNUMBER： 网络号；

DO_ENABLE： 数字量输出使能；

DO_PTR： 数字量输出存储地址；

DO_LENGTH： 数字量输出长度；

DI_ENABLE： 数字量输入使能；

DI_PTR： 数字量输入存储地址；

DI_LENGTH： 数字量输入数据长度；

AI_ENABLE： 模拟量输入使能；

AI_PTR： 模拟量输入存储地址；

AI_LENGTH： 模拟量输入数据长度；

REG_ENABLE： 寄存器使能；

REG_PTR： 寄存器存储地址；

REG_LENGTH：寄存器存储数据长度；

DONE：完成标志位；

ERROR：错误代码说明了有关功能块执行结果的信息。可能的错误代码定义如下：

0:在执行功能块时，没有发生错误

1:在执行功能块时，出现硬件错误

2:当调用函数块时，设置了无效的模式

4:设置的地址太大，超出了最大可用存储区域

8:当调用函数块时，设置了无效的以太网接口

示例程序：

```
VAR
lanInit:bool:=false; inst0_LAN_INIT:LAN_INIT;
mConfirm:bool; mError:usint; mErrorinfo:usint; mIP:udint; mNetMask:udint; mGateWay:udint;
mSocket:INT;
inst0_MODBUS_TCP_SLAVE_INIT:MODBUS_TCP_SLAVE_INIT;
mModbusInitOK:bool:=false; modbusDOBuf : ARRAY[0..5] OF byte; DO_Ptr : POINTER;
modbusDIBuf : ARRAY[0..5] OF byte; DI_Ptr : POINTER;
modbusAIBuf : ARRAY[0..10] OF int; AI_Ptr : POINTER;
modbusRegBuf : ARRAY[0..127] OF int; mPtr : POINTER;
xDONE:BOOL; inst4_MODBUS_TCP_SLAVE_CTRL:MODBUS_TCP_SLAVE_CTRL;
xError:usint;
mDI at %I0.0:byte; mAI at %I1.0:int; mDO at %Q0.0:byte;
END_VAR
```

if lanInit=false then

mIP := LAN_ASCII_TO_INET('192.168.1.31'); mNetMask := LAN_ASCII_TO_INET('255.255.255.0'); mGateWay := LAN_ASCII_TO_INET('192.168.1.1');

inst0_LAN_INIT(ENABLE :=true , HOSTNAME := 'PLCCore'， IP := mIP, NETMASK :=mNetMask, GATEWAY :=mGateWay ,NETNUMBER:=1

| mConfirm:= CONFIRM,mError:= ERROR， mErrorinfo:= ERRORINFO); lanInit:=true;

else

if mModbusInitOK=false then

inst0_MODBUS_TCP_SLAVE_INIT(ENABLE := true, MODE := 1, ADDRESS :=1 ,

NETNUMBER :=1 | mConfirm:= CONFIRM, mError:= ERROR, mErrorinfo:= ERRORINFO);

mModbusInitOK:=true; end_if;

end_if; mDO:=modbusDOBuf[0]; modbusDIBuf[0]:=mDI; modbusAIBuf[0]:=mAI;

modbusRegBuf[0]:=byte_to_int(mDI); modbusRegBuf[1]:=mAI; modbusRegBuf[2]:=byte_to_int(mDO);

mPtr:=&modbusRegBuf; DO_Ptr:=&modbusDOBuf; DI_Ptr:=&modbusDIBuf; AI_Ptr:=&modbusAIBuf;

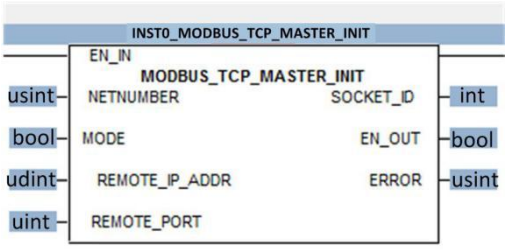
if mModbusInitOK=true then

inst4_MODBUS_TCP_SLAVE_CTRL(NETNUMBER :=1,DO_ENABLE :=1 , DO_PTR :=DO_Ptr ,

DO_LENGTH :=5 , DI_ENABLE :=1 ,DI_PTR :=DI_Ptr , DI_LENGTH :=5 , AI_ENABLE :=1 , AI_PTR :=AI_Ptr , AI_LENGTH :=10 , REG_ENABLE :=1 , REG_PTR :=mPtr , REG_LENGTH :=127 | xDONE := DONE, xError := ERROR);

end_if;

3.3.62 MODBUS_TCP_MASTER_INIT



用于初始化Modbus-TCP主站接口

操作数的定义：

EN_IN：启用或禁用功能块，true为启用，false为禁用；

MODE：模式选择；

REMOTE_IP_ADDR:远端IP地址

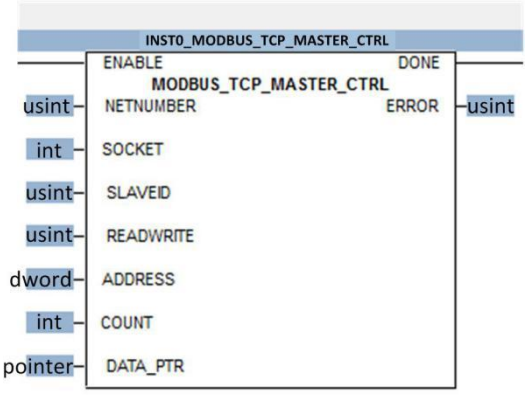
REMOTE_PORT:远端端口号

EN_OUT:完成标志位；

SOCKET_ID:网络ID

ERROR:错误代码说明了有关功能块执行结果的信息。

3.3.63 MODBUS_TCP_MASTER_CTRL



用于执行Modbus-TCP主
站接口的控制指令（功能码）

操作数的定义

ENABLE：功能块使能；

NETNUMBER：网络号；

SOCKET：网络ID；

SLAVEID：从站ID；

READWRITE：读写使能；

ADDRESS：从站地址；

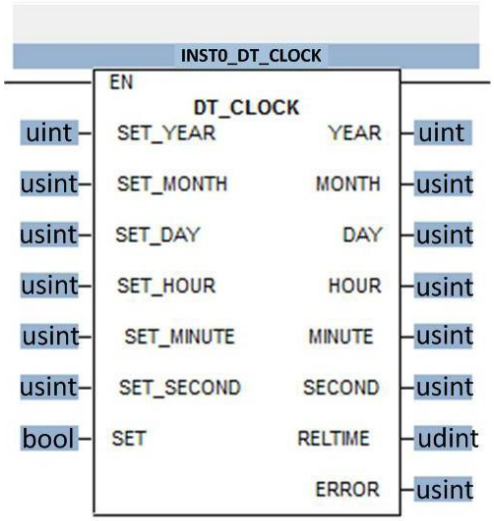
COUNT：读写的数量；

DATA_PTR：存放读写变量数组的指针；

DONE：完成标志位；

ERROR：错误代码说明了有关功能块执行结果的信息。

3.3.64 DT_CLOCK



用于设置实时时钟RTC并从PLC的RTC中读取日期和时间。这个功能块仅在配有RTC功能的控制器上可用

操作数的定义：

SET_YEAR、SET_MONTH、SET_DAY：年/月/日的设定

SET_HOUR、SET_MINUTE、SET_SECOND：小时/分/秒的设定

SET：TRUE：允许输入SET_YEAR，SET_MONTH和SET_DAY对日期进行更改，允许输入

SET_HOUR，SET_MINUTE和SET_SECOND对具体时间进行修改，同时，可以在各自的输出读取设定的日期和时间。

FALSE：只读取RTC的当前日期和时间，不可更改日期和时间。

YEAR、MONTH、DAY：年/月/日的输出日期

HOUR、MINUTE、SECOND：小时/分/秒的输出时间

RELTIME：读取日期和时间的相对形式（自01.01.1980以来的秒数）

ERROR：错误代码说明功能块执行结果的信息。可能的错误代码定义如下：

在执行功能块时没有发生错误

执行功能块时发生硬件错误

4功能块通话期间的模式（MODE）无效

8电源故障，读取时间无效

16读取绝对时间和日期无效

描述：

如果通过设置输入SET为TRUE调用功能块，则输入日期（SET_YEAR，SET_MONTH及SET_DAY）和输入时间（SET_HOUR，SET_MINUTE及SET_SECOND）可按照各自的输入值传递给PLC的RTC。同时，设定的日期和时间可以在各自的输出读取。但是，如果通过设置输入SET为FALSE调用了功能块，则只能读取当前日期和时间，RTC不受影响，设置输入被丢弃，功能块执行过程中可能出现的错误显示在输出端ERROR。

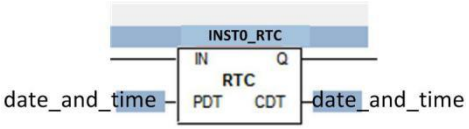
如果在执行功能块DT_CLOCK后输出为ERROR=3，则电源为RTC已中断（电源故障或电池为空），读取时间无效。这个错误状态一直保持到PLC的RTC复位（功能块通过输入SET：=TRUE调用）或PLC通过复位开关复位。

以下示例程序显示功能块DT_CLOCK设置和查看系统RTC的过程。

示例程序：

```
VAR
Year : UINT; Month : USINT; Day : USINT; Hour : USINT;
Minute : USINT; Second : USINT; RelTime : UDINT;
ErrorCode : ARRAY [0..1] OF USINT;
FB_DtClock : DT_CLOCK; END_VAR
LD 0
ST ErrorCode[0] ST ErrorCode[1]
CAL FB_DtClock (SET_YEAR := 2003,SET_MONTH := 8,SET_DAY := 6,SET_HOUR := 12, SET_MINUTE :=
3,SET_SECOND := 0,SET := TRUE|Error[0] := ERROR)
CAL FB_DtClock (SET :=FALSE)
LD FB_DtClock.YEAR ST Year
LD FB_DtClock.MONTH ST Month
LD FB_DtClock.DAY ST Day
LD FB_DtClock.HOUR ST Hour
LD FB_DtClock.MINUTE ST Minute
LD FB_DtClock.SECOND ST Second
LD FB_DtClock.RELTIME ST RelTime
LD FB_DtClock.ERROR ST    ErrorCode[1] RET
END_PROGRAM
```

3.3.65 RTC



用于当IN=1时设置输出
CDT等于输入PDT，否则
CDT无效

操作数的定义：

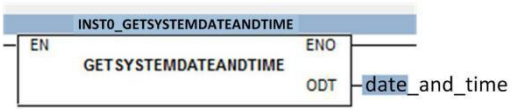
IN：功能块使能输入

PDT：系统日期和时间

Q：与IN状态相同

CDT：当IN=1时，输出系统日期和时间

3.3.66 GETSYSTEMDATEANDTIME



操作数的定义：

EN:使能输入

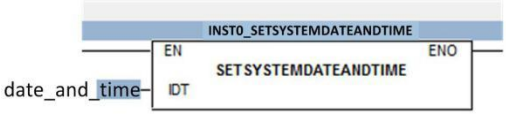
ENO:使能输出

ODT:系统时间

描述：

该功能块用于在输出ODT处显示实际系统时间，该功能块未被IEC61131-3标准定义。

3.3.67 SETSYSTEMDATEANDTIME



操作数的定义：

EN:使能输入

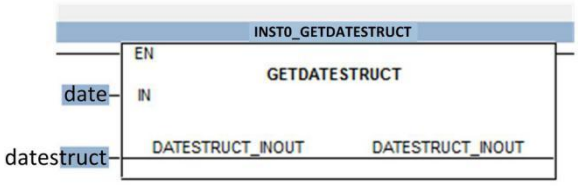
IDT:系统日期和时间的设置值

ENO:使能输出

描述：

该功能块在输入IDT处设置实际系统时间，该功能块未被IEC61131-3标准定义。

3.3.68 GETDATESTRUCT



由日期类型转为结构体类型

操作数的定义:

EN:使能输入

IN：须转换的日期

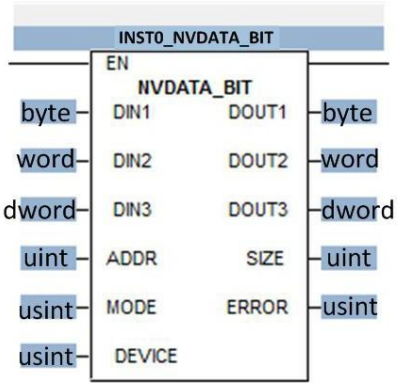
DATESTRUCT_INOUT:转换得到的日期结构体,如：

datestruct：structYear：uint；Month：usintDay：usintend_struct

描述：

GETDATESTRUCT将一个给定的日期转换为结构体日期，以单独整型变量分别表示日，月，年。

3.3.69 NVDATA_BIT



用于在读取PLC存储单元（EEPROM）中数据的同时，写入过程数据(这些数据可以是1字节，或1个字，或4个字节)

操作数的定义：

DIN1：写入一个字节的数据；

DIN2：写入两个字节的数据；

DIN3：写入四个字节的数据；

ADDR：存储器中的地址，用于读取和写入数据(需设置参数模式)；

MODE：执行读或写操作的设置，表31-1中包含所支持的所有模式；

DEVICE：设备号，需要设置为0；

DOUT1：读取一个字节的数据；

DOUT2：读取两个字节的数据；

DOUT3：读取四个字节的数据；

SIZE：MODE不为0时，用于输出读取或写入的字节的数量，

MODE为0时，用于输出可使用的存储器大小；

ERROR：错误代码说明了有关功能块执行结果的信息。可能的错误代码定义如下：

0:在执行功能块时，没有发生错误

1:在执行功能块时，出现硬件错误

2:当调用函数块时，设置了无效的设备号

4:当调用函数块时，设置了无效的模式

8:设置的地址太大，超出了最大可用存储区域

16:指针指向了一个不支持的数据类型

功能块调用模式:

16#00确定可用的存储器大小

16#01在DOUT1中，读取一个字节的数据

16#02在DOUT2中，读取两个字节的数据

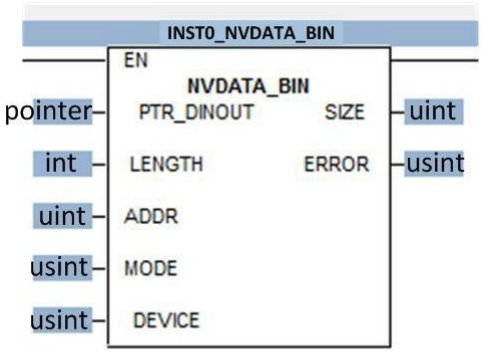
16#03在DOUT3中，读取四个字节的数据

16#81在DIN1中，写入一个字节的数据

16#82在DIN2中，写入两个字节的数据

16#83在DIN3中，写入四个字节的数据

3.3.70 NVDATA_BIN



用于在读取PLC存储单元（EEPROM）中数据的同时，写入过程数据（这些数据为二进制数据）

操作数的定义：

PTR_DINOUT：数字量输入输出存储地址（用于读写数据）；

LENGTH：读/写字节数的长度；

ADDR：存储器中的地址，用于读取和写入数据(需设置参数模式)；

MODE：执行读或写操作的设置，表30-1中包含所支持的所有模式；

DEVICE：设备号，需要设置为0；

SIZE：MODE不为0时，用于输出读取或写入的字节的数量，

MODE为0时，用于输出可使用的存储器大小；

ERROR：错误代码说明了有关功能块执行结果的信息。可能的错误代码定义如下：

0:在执行功能块时，没有发生错误

1:在执行功能块时，出现硬件错误

2:当调用函数块时，设置了无效的设备号

4:当调用函数块时，设置了无效的模式

8:设置的地址太大，超出了最大可用存储区域

16:指针指向了一个不支持的数据类型

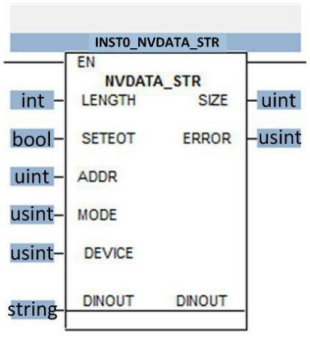
模式功能：

16#00:确定可用的存储器大小

16#09:从存储器中读取二进制数据,地址位于PTR_DINOUT

16#89:将二进制数据写入存储器中，地址位于PTR_DINOUT

3.3.71 NVDATA_STR



该功能块用于将字符串过程数据写入PLC非易失性存储区中或从PLC非易失性存储区中读取字符串过程数据

操作数的定义：

EN：功能块使能输入

LENGTH：读或写的字符串数量限制。如果值为0，则字符串的长度是内部定义的（等于LEN（DINOUT））；注意：在OpenPCS中标准的字符串缓冲区长度是32个字符。

SETEOT：TRUE:带终止字符的字符串存储FALSE:不带终止字符的字符串存储（默认:TRUE）

ADDR：存储器中的地址，用于读取和写入数据

MODE：读或写的操作模式设置，列表12包括所有支持的模式。

DEVICE：设备编号，这一参数由各自的控制系统决定。（注意：大多数的控制系统只支持设备号0，而设备号0会被默认为初值，因此不必单独设置）

DINOUT：用来读写字符串值的输入、输出

SIZE：用来表示读或写的字节数（MODE≠0）或非易失性存储区的可用大小（MODE=0）

ERROR：错误代码表示功能块执行结果的信息，可能的错误代码定义如下:

功能块的调用方式：

16#00确定可用的非易失性存储区的大小

16#08在数据输出端口DINOUT读取一个非易失性缓冲区中的字符串

16#88在数据输入端口DINOUT写一个字符串到非易失性缓冲区中

描述：

可以通过功能块对非易失性存储区的字符串数据或读或写，根据读或写的数据，符合列表12的对应模式必须在输入MODE处设置。此种情况下，参数DINOUT作输入还是作输出取决于模式的类型，输入ADDR的值是执行读或写的基本地址。如果寻址范围超过最大值，功能块会报相应的错误。PLC程序用来划分可获取的存储器区域并确保输入ADDR的值不会造成存储数据的重叠。读或写的字节数量会显示给输出SIZE。这个值会被用来计算下一个空闲地址（ADDRnew：=ADDRold+SIZE），输入LENGTH指定写入过程中有效的字符数，如果该值为0，字符串的长度会由内部确定（等于LEN（DINOUT））并被用作写入的字符数，在这种情况下，全部被使用的字符串内容都会被写入。输入LENGTH也会在读取时使用来限制待处理的字符数量到指定值。

使用输入SETEOT来设置是否字符串的结束符也被存储（默认是TRUE），如果字符串是带结束符被完整储存到非易失性存储器中的，则读取过程中输入LENGTH的值就不是必须的（LENGTH=0），功能块会接收所有的字符数据直到在DINOUT中检测到结束分隔符。当令SETEOT=false调用功能块时结束符的存储区域是空的。因此，在非易失性存储区中每个字符串会被占用的区域小于一个字节。但是这种情况字符串的长度必须已知且在读取过程中指定到输入LENGTH上。如果结束符已经被写入，当处理过的字符在输出SIZE处被指定时，结束符的长度仍然会被考虑进去。因此，当令SETEOT=TRUE调用功能块时，输出SIZE的值等于LEN（DINOUT）+1。当令MODE=0调用功能块时会确定非易失性存储器的可用区域大小，此种情况下，自输入ADDR起的遗留的剩余区域大小会反馈到输出SIZE上（SIZE：=NVDATAFullSize-ADDR）。通过令ADDR=0调用功能块将确定非易失性存储器的全部大小。功能块执行过程中可能的错误信息将会显示到输出ERROR上，并在表10中描述（错误代码与功能块NVDATA_BIT相同）。

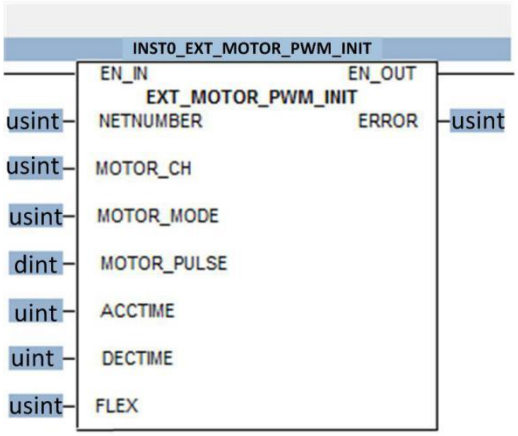
下面的示例程序显示了功能块NVDATA_STR的应用。开始时，从地址30处写入一个字符串，随后又从同样的地址读取该字符串，标准设置保持相同，且默认设备号为0。

示例程序:

```
VAR CONSTANT

NVDSTR_MODE_GET_SIZE : USINT := 16#00; NVDSTR_MODE_RD_STRING : USINT := 16#08;
NVDSTR_MODE_WR_STRING : USINT := 16#88; NVDATA_ERROR_SUCCESS : USINT := 0;
NVDATA_ERROR_HW_ERROR : USINT := 1; NVDATA_ERROR_UNKNOWN_DEVICE :
USINT := 2; NVDATA_ERROR_INVALID_MODE : USINT := 4; NVDATA_ERROR_OUT_OF_MEM :
USINT := 8;

END_VAR

VAR

WriteDataString : STRING; WriteDataSize : UINT; ReadDataString : STRING; ReadDataSize : UINT;

Error : ARRAY[0..1] OF USINT;

FB_NvDataStr : NVDATA_STR;
```

```
END_VAR

(* write a STRING value into EEPROM *) LD    'HelloWorld'

ST      WriteDataString

CAL    FB_NvDataStr ( DINOUT := WriteDataString,

LENGTH := 0,          (* save whole string   *)          SETEOT := TRUE,  (* include termination character *)

   ADDR := 30,

MODE := NVDSTR_MODE_WR_STRING

|

WriteDataSize := SIZE, Error[0] := ERROR)

(* read a STRING value from EEPROM *) CAL  FB_NvDataStr (

DINOUT := ReadDataString,

LENGTH := 0,   (* read whole string   *) ADDR := 30,

MODE := NVDSTR_MODE_RD_STRING

|

ReadDataSize := SIZE, Error[1] :=    ERROR) RET


END_PROGRAM
```

3.3.72 EXT_MOTOR_PWM_INIT



GC-2302模块初始化脉冲输出功能块

操作数的定义：

EN_IN：功能块使能输入；

NETNUMBER：GC-2302模块的排列序号，主控模块或耦合器后接的第一个GC-2302模块为1，第二个GC-2302模块为2，以此类推；

MOTOR_CH：每个模块脉冲输出的排列序号，每个GC-2302模块有两路脉冲输出，第一路脉冲输出序号为1，第二路脉冲输出序号为2；

MOTOR_MODE：GC-2302模块脉冲输出的模式：1为速度模式，2为位置模式，3为脉冲模式

MOTOR_PULSE：速度模式和位置模式下电机转动一周所需的脉冲数，脉冲模式下无意义；

ACCTIME：加速时间，脉冲模式下无意义

DECTIME：减速时间，脉冲模式下无意义

FLEX：坡度系数：

EN_OUT：功能块使能输出；

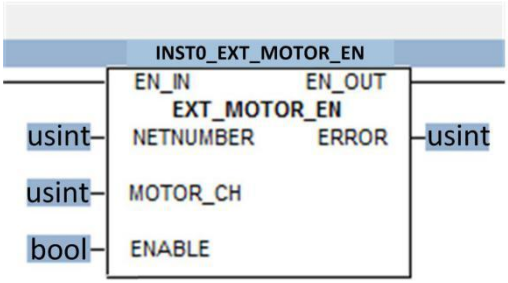
ERROR：错误代码说明了有关功能块执行结果的信息

描述：

本功能块用于初始化GC-2302模块的脉冲输出，脉冲输出模式共有3种，在MOTOR_MODE为1和2时，ACCTIME、DECTIME、FLEX这3项对电机进行配置的参数才有意义，可在程序中对输出地址进行

定义并调用功能块EXT_MOTOR_EN实现对电机参数的配置，当MOTOR_MODE为3时，端子模块输出为纯脉冲，同样可在程序中对输出地址进行定义实现对脉冲占空比和频率的配置。

3.3.73 EXT_MOTOR_EN



GC2302模块使能电机功能块

操作数的定义：

EN_IN：功能块使能输入；

NETNUMBER：GC-2302 模块的排列序号，主控模块或耦合器后接的第一个 GC-2302 模块为 1，第二个 GC-2302 模块为 2，以此类推；

MOTOR_CH：每个模块脉冲输出的排列序号，每个 GC-2302 模块有两路脉冲输出，第一路脉冲输出序号为 1，第二路脉冲输出序号为 2；

MOTOR_MODE：GC-2302 模块脉冲输出的模式：

1 为速度模式，2 为位置模式，3 为脉冲模式

ENABLE：电机使能信号

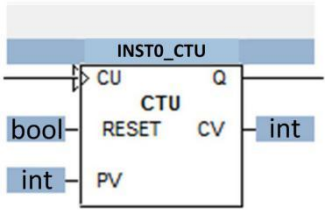
EN_OUT：功能块使能输出；

ERROR：错误代码说明了有关功能块执行结果的信息

描述：

本功能块用于 GC-2302 脉冲输出模块使能电机。在程序中需要预先调用功能块 EXT_MOTOR_PWM_INIT 对输出端子模块进行初始化并配置参数，在初始化成功后调用该功能块对电机进行使能，电机会按照定好的参数运行。

3.3.74 CTU



加计数

操作数的定义：

CU：计数脉冲

RESET：复位计数器

PV：计数限制值

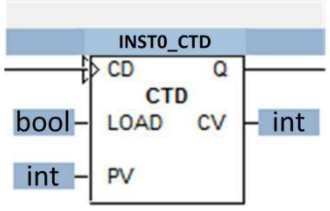
Q：显示是否计数器达到限制值

CV：当前计数器的值

描述：

功能块用于计算从输入CU接收到的脉冲数，初始化时，计数器的值将设置为0，如果输入RESET设为1，则计数器的值将被复位，输入CV处每个检测到的上升沿都会让计数器的值+1，输出CV为计数器的当前值。如果计数器的值小于PV，则输出Q为0，如果计数器的值大于等于PV，则输出Q为1。

3.3.75 CTD



减计数

操作数的定义：

CD:计数脉冲

LOAD:设置初值

PV:初值

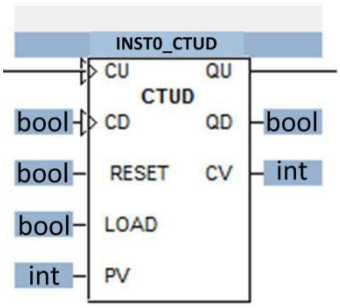
Q:计数器值为0时输出1

CV:计数器的值

描述：

功能块CTD采集输入CD处的脉冲信号做减计数运算，如果参数LOAD的值为1，则操作数PV的值就会被装载到计数器中，输入CD处每检测到一个上升沿都会让计数器的值减1，输出CV为计数器当前值，如果计数器的值为正数，则输出Q的值为0.如果计数器的值小于等于0，则输出Q的值为1，功能块已被IEC61131-3标准定义。

3.3.76 CTUD



加、减计数

操作数的定义：

CU:上升沿加计数

CD:上升沿减计数

RESET:复位计数器

LOAD:装载初值

PV: 初值

QU: 显示计数器的值是否大于PV

QD: 显示计数器的值是否大于0

CV: 计数器当前值

描述：

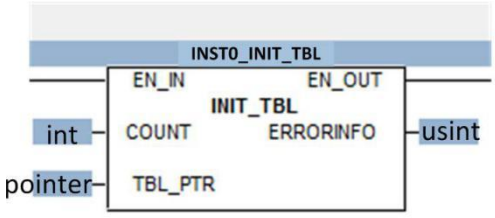
功能块CTUD用来加减计数。初始化时计数器的值为0.每个输入CU处的上升沿将会使计数器加1，而每个输出CD处的上升沿将会使计数器的值减1。

如果操作数LOAD的值为1，则PV的值将会被装载到计数器中

如果操作数RESET的值为1，则计数器的值将为0.如果RESET保持为0，则计数和装载的正常操作将不会受到影响。

输出CV为计数器的当前值。如果计数器的值小于PV，则输出QD将为0.如果计数器的值大于等于PV，则输出QD将为1.如果计数器的值为正数，则输出QD将为0，如果计数器的值小于等于0，则输出QD将为1.该功能块已在IEC61131-3标准中定义。

3.3.77 INIT_TBL



初始化列表存储功能块

操作数的定义：

EN_IN:使能初始化列表存储功能块；

COUNT:定义列表内成员的数量；

TBL_PTR:列表成员的数组指针；

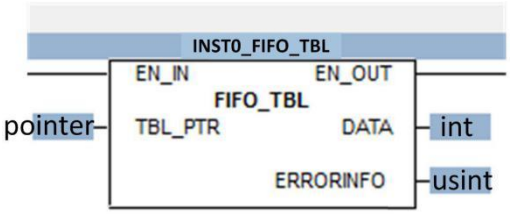
EN_OUT:功能块使能输出；

ERRORINFO：功能块执行结果的错误信息

描述：

该功能块用于初始化列表存储功能，由输入COUNT定义列表成员数量，TBL_PTR定义列表成员所在的数组指针。

3.3.78 FIFO_TBL



列表成员先进先出

操作数的定义：

EN_IN:使能列表成员先进先出功能块；

TBL_PTR:列表成员的数组指针；

EN_OUT:功能块使能输出；

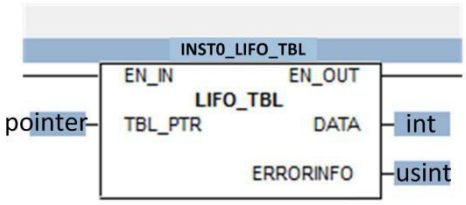
DATA:先进先出的列表成员；

ERRORINFO：功能块执行结果的错误信息

描述：

该功能块用于列表成员的先进先出，由输入EN_IN使能功能块，TBL_PTR定义列表成员所在的数组指针。

3.3.79 LIFO_TBL



列表成员先进后出

操作数的定义：

EN_IN:使能列表成员先进后出功能块；

TBL_PTR:列表成员的数组指针；

EN_OUT:功能块使能输出；

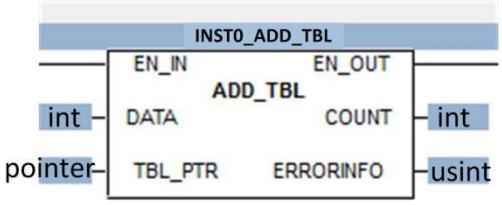
DATA:先进后出的列表成员；

ERRORINFO：功能块执行结果的错误信息

描述：

该功能块用于列表成员的先进后出，由输入EN_IN使能功能块，TBL_PTR定义列表成员所在的数组指针。

3.3.80 ADD_TBL



添加列表成员

操作数的定义：

EN_IN:使能添加列表成员功能块； TBL_PTR:列表成员的数组指针；

DATA:添加的列表成员；

EN_OUT:功能块使能输出；

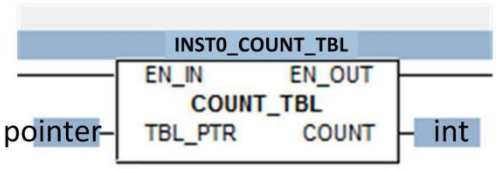
COUNT:列表成员的数量；

ERRORINFO：功能块执行结果的错误信息

描述：

该功能块用于列表成员的添加，由输入 EN_IN 使能功能块，TBL_PTR 定义列表成员所在的数组指针。

3.3.81 COUNT_TBL



列表成员计数

操作数的定义：

EN_IN:使能列表成员计数功能块；

TBL_PTR:列表成员的数组指针；

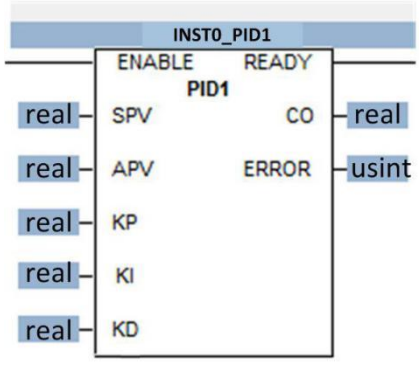
EN_OUT:功能块使能输出；

COUNT：列表成员的数量；

描述

该功能块用于统计列表成员的数量，由输入EN_IN使能功能块，TBL_PTR定义列表成员所在的数组指针。

3.3.82 PID1



该功能块用于实现PID控制

操作数的定义

ENABLE：如果检测到输入上升沿，控制因数KR，T0，TI，TD会被系统接收，同时会计算初值偏差的积分和。输出READY，ERROR和CO会通过令ENABLE=0来复位。功能块检查有效区域，如有必要，在参数T0和BIAS传输过程中在输出ERROR处显示溢出误差；

SPV：控制器标准设定值（控制变量），有效范围0.0~1.0，功能块会自动检测参数（但不会检测溢出错误）

APV：控制器标准实际值（过程变量），有效范围0.0~1.0，控制器会自动检测参数（但不会检测溢出错误）

KP：比例系数；

KI：积分系数；

KD：微分系数；

READY：PID控制器的输出状态TRUE=控制器的功能块已经完全参数化并已经进入预操作状态。FALSE=控制器的功能块未完全参数化或错误参数化（控制参数位于有效值范围外）,控制器未进入预操作模式。

CO:控制器的输出，计算控制器的校正变量（值的范围0.0~1.0）

ERROR:错误代码表示功能块执行结果的信息，可能的错误代码定义如下：

0:在功能块执行过程中未发生错误

8:指定的参数BIAS的值是无效的（小于0或大于1）

16:指定的参数T0的值是无效的（时间等于0）

描述：

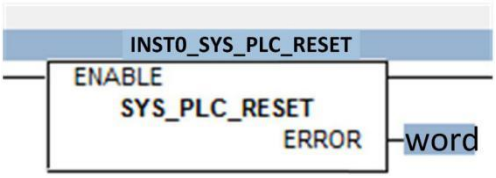
该功能块用于实现PID控制，如果输入ENABLE处检测到上升沿，功能块接收控制参数KR，TO，TD，TI和BIAS并启动控制器。在第一次运算时，设定值和最后一次实际值被设为当前实际值。第一次运算校正变量通常会出现初始偏差值，因为比例增益、积分增益和微分增益第一次默认都为0。控制器的性能表现会受控制参数的影响，如果TI=t#0ms，控制器的积分增益不会被计算且赋值为0。如果TD=t#0ms，微分增益的值会被赋为0.如果参数KR的增益为0，比例增益就不再需要。因为KR增益与比例增益和微分增益都有关系，积分和微分增益这里被设置为1。P控制器：TI=TD=0，KR≠0，

积分和微分增益为1。PI控制器：TD=0，KR，TI≠0。PID控制器：KR，TI，TD≠0。有很多种方法测定控制参数（拐点切线，震荡试验），参数也可以通过仿真工具进行确定。然而，在频率和相位响应方面，完整的控制系统被描述成一个模型是很必要的。控制系统的相关传递参数的模型可以从已知值和状态中建立并用来决定仿真过程中的参数。设定值、实际值、偏差值和校正变量被当做标准变量使用，这些标准化变量的值变化范围为0.0~1.0，在功能块开始执行时会检查偏差的有效范围，当偏差值超出有效范围时输出ERROR会报错。在程序运行过程中不会检查设定值和实际值。功能块限制了校正变量和积分和的值，如果计算结果为负值，变量值就会被设为0。如果值超过1，变量值就会被设为1.此外，积分和受校正变量的影响，也遵循以下规则：

如果校正变量的值大于1，则积分和的计算方式如下：积分和=1-（比例增益+微分增益）

如果校正变量的值大于0，则积分和的计算方式如下：积分增益=1-（比例增益+微分增益）

3.3.83 SYS_PLC_RESET



PLC系统复位重启功能块

操作数的定义：

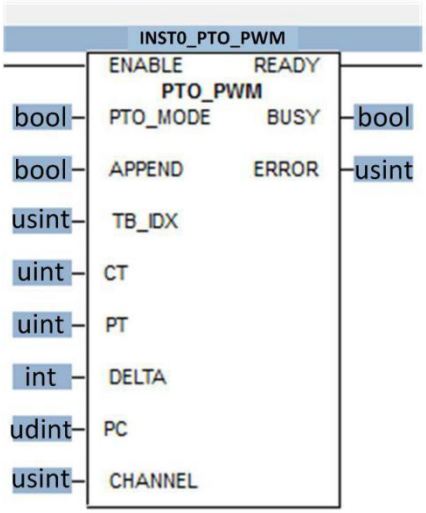
ENABLE:使能PLC系统复位重启功能块；

ERROR：错误代码说明了有关功能块执行结果的信息；

描述：

该功能块用于PLC系统复位重启，当输入ENABLE为1时，PLC会停止当前进程，重新启动。

3.3.84 PTO_PWM



脉冲输出PTO功能块

操作数的定义：

ENABLE：启用或禁用功能块，true为启用，false为禁用；

PTO_MODE：模式选择

1：PTO发生器（脉冲计数器输出，一次性脉冲串）

0：PWM发生器（脉宽调制输出，连续脉冲串）

在输入ENABLE为1时模式发生改变会终止先前模式下设置的各项功能

APPEND：用于控制扩展参数

1：接受当前配置的参数作为进一步的参数集

0：仅更新功能块的输出状态，丢弃之前配置的参数

TB_IDX：用于设置脉冲发生器的基本周期，这项参数取决于相应的控制模式有效值如下：0=800ns基本周期 1=1ms基本周期

基本周期只有在输入ENABLE接收到上升沿后才会被设置

CT：PTO模式：持续时间

PWM模式：周期时间

持续时间和周期时间取决于输入TB_IDX指定的基本周期TB_IDX:=0:

125...65535(100µs-52428µs)TB_IDX:=1: 2...65535(2ms-65535ms)

PT：PTO模式：未使用

PWM模式：脉冲持续时间，数值范围：0...65535

DELTA：PTO模式：两个脉冲之间持续性的周期变化，数值范围:-32768...+32767

PWM模式：未使用

PC：PTO模式：脉冲数，数值范围:1...4294967295

CHANNEL：使用到的通道号

READY：脉冲发生器的输出状态

1：脉冲发生器已完全参数化，发生器已准备好运行

0：脉冲发生器未参数化，或功能块因错误而终止，发生器未准备好

BUSY：脉冲发生器的状态输出：

1:脉冲发生器激活状态（正在生成脉冲串）；发生器控制数字输出

0:脉冲发生器未激活状态（脉冲串已经生成）；过程映像控制数字量输出（PLC程序直接映像输出）

ERROR：错误代码表示功能块执行结果的信息，可能的错误代码定义如下：

0：功能块执行过程中无错误

1：功能块执行过程中发生了硬件错误

2：选择的通道号不支持

8：选择的基本周期的索引不支持

16：考虑DELTA时重新计算持续时间的溢出错误

32：数据记录缓冲区中无可用空间，数据记录已被丢弃

描述：

功能块可直接参数化脉冲发生器PTO（脉冲串输出）和PWM（脉宽调制输出）。脉冲输出模式是数字量输出的替代功能。如果输入ENABLE=0，则过程映像直接影响对应的数字量输出。如果输入

ENABLE=1，则脉冲发生器控制输出。

PTO发生器（PDO_MODE=1,脉冲计数器输出，一次性脉冲串）：PTO发生器创建一次性脉冲串以控制数字量输出。脉冲序列由一个参数集描述，该参数集包括周期持续时间（初始值），周期持续时间的增量（两个连续脉冲之间的变化值）以及要生成的脉冲数。脉冲宽度设置为周期持续时间的50%（采样比1:1）。考虑到DELTA，周期持续时间Tn计算如下：Tn=(CT+n*DELTA)*tB （0<=n<=PC）

如果tB=1ms（tB_IDX:=1）和CT:=1000，则初始周期持续时间（n=0）为：Tn=1ms*1000=1秒通过功能块，可以将具有不同周期持续时间、增量（周期持续时间的变化）和脉冲数量值的脉冲串连接起来。为此，需要令APPEND=1调用功能块，使每个参数集都可扩展。

这样最多可以扩展255个参数集。直到ERROR=32（数据记录缓冲区无可用空间，数据记录已被丢弃）。所有参数集都基于相同的时间标准。只有在脉冲发生器停用的时候才能更改时间标准（输入TB_IDX）。只有在输入ENABLE检测到上升沿时才会启动时间标准索引。如果脉冲串已完全传输且没有进一步的参数设置可用，则脉冲发生器PTO自动关闭，过程映像再次控制数字量输出。因此，PLC程序必须在过程映像中存储脉冲发生器停用后所需的数字输出状态。如果在后续的脉冲发生期间出现上溢或下溢错误，则PTO发生器也会自动关闭。如果Tn大于65535或小于0，则为此种情况。功能块显示ERROR=32（溢出错误），由于是包含DELTA的累积运算，所以ERROR的结果是经过一系列连续的成功运算后得出的。

PWM发生器（PDO_MODE=0,脉冲持续时间输出，连续的脉冲串):在PWM发生器功能中，在数字量输出处会生成连续的脉冲串。这种情况下，周期持续时间和脉冲持续时间可以设置为基本周期的数量。如果输入ENABLE为1，则PWM生成器在采集到参数后直接被激活。如果脉冲持续时间PT的值为0，则在整个周期持续时间内，各个输出均保持不活动状态。但如果脉冲持续时间PT的值大于或等于周期持续时间，则输出在整个周期持续时间内均处于活动状态。持续的占空时间总是异步变化，当前占空状态会因接收新值而中断。脉冲的存续时间是同步变化的并在下一个周期持续时间开始时接受。通过令APPEND=1调用功能块来改变参数，令ENABLE=0来终止连续脉冲串的生成。如果在执行过程中发生错误，功能块会自动关闭。只有通过ENABLE=0重置功能块后，才能重新使用该功能块。

输出READY表示功能块已完全参数化并进入预操作状态，参数TB_IDX（用来设置脉冲发生器基本周期的索引）不会被改变（TB_IDX只有在输入ENABLE处检测到上升沿时才会被读取），如果通过ENABLE=0调用功能块，则输出READY=0。

如果功能块输出BUSY=1，则表示脉冲发生器已被激活并控制相应的数字量输出（PTO模式：会传输参数化脉冲串，PWM模式：会生成连续的脉冲串）。BUSY=0表示发生器未被激活并由PLC程序

通过过程映像直接影响相应的数字输出。

功能块执行过程中可能的错误信息在ERROR中显示。

下面的示例程序显示了功能块在PTO模式下生成一次性脉冲串以及在PWM模式下生成连续脉冲串的应用。由于参数选择的特殊性，不需要额外的技术或测试设备来观察脉冲串在输出PWM处的LED灯状态。通过输出xStartButton的上升沿启动PTO循环模式。脉冲串以1s的脉冲开始（CT*TB=1000*1ms=1s），随后的每个脉冲缩短50ms（Delta=-50）。共产生15个脉冲（PC=15）。通过输入xStartButtonPwm的上升沿启动PWM模式。产生的脉冲串持续时间为500ms（CT*TB=500*1ms=0.5s->2Hz），每个脉冲的on-time为150ms

（PT*TB=150*1ms=150ms）。

示例程序：

```
VAR    CONSTANT

(* Definition of TimeBase-Index *) PTO_TB_IDX_800_US:USINT:=0;       (* TimeBase-Index 800us *)
PTO_TB_IDX_1_MS:USINT:=1;     (* TimeBase-Index 1ms     *) (* Error Codes of FB PTO_TAB *)
PTOTAB_ERROR_SUCCESS:SINT=0;

PTOTAB_ERROR_HW_ERROR: USINT:=1;

PTOTAB_ERROR_UNKNOWN_CHANNEL:USINT:=2; PTOTAB_ERROR_UNKNOWN_TB_IDX:USINT:= 8;
PTOTAB_ERROR_DELTA_OVERFLOW:USINT:=16; PTOTAB_ERROR_INVALID_TAB:USINT:=64;
PTO_PWM_CHANNEL:USINT:=0;

END_VAR VAR

xStartButtonPto AT %IX0.0 : BOOL; xStartButtonPwm AT %IX0.1 : BOOL; xPtoPwmOut AT %QX2.4 :
BOOL;

FB_RTrigPto     : R_TRIG; FB_RTrigPwm     : R_TRIG; usiPtoTbIdx: USINT := 1; uiPtoCt   : UINT    := 1000;

iPtoDelta   : INT      := -50;

udiPtoPc    : UDINT := 15; usiPwmTbIdx    : USINT := 1; uiPwmCt      : UINT    := 500;

uiPwmPt    : UINT    := 150;

xPtoAppend     : BOOL   := TRUE; xPtoReady    : BOOL   := FALSE;
```

```
xPtoBusy   : BOOL   := FALSE;

xPwmAppend  : BOOL   := TRUE; xPwmReady : BOOL   := FALSE;

xPwmBusy      : BOOL   := FALSE;

FB_PtoPwm     : PTO_PWM; usiPtoPwmError : USINT;

END_VAR


(* ----- Wait for Start ----- *) WaitForStart:

CAL    FB_RTrigPto

LD     FB_RTrigPto.Q JMPC  StartPtoMode CAL     FB_RTrigPwm

LD     FB_RTrigPwm.Q JMPC      StartPwmMode LD     xPtoBusy

JMPC  RunPtoMode

LD     xPwmBusy

JMPC  RunPwmMode

JMP    ProgExit


(* ----- Run PTO Mode ----- *)

StartPtoMode:

LD         FALSE (* preset output state, this state is    *) ST         xPtoPwmOut         (* used when PTO
Generator isn't running *) LD         FALSE (* reset state flags *)

ST     xPtoReady

ST     xPtoBusy

ST     xPwmReady

ST     xPwmBusy
```

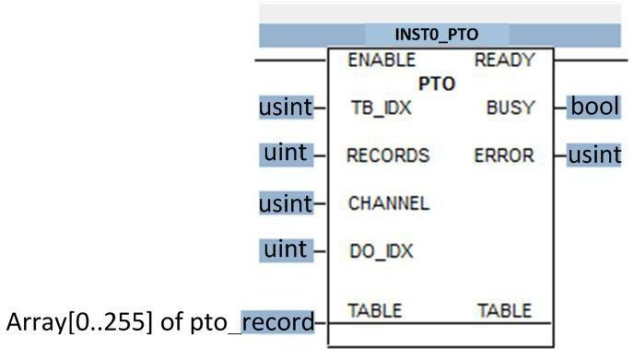
```
CAL    FB_PtoPwm ( ENABLE := FALSE,
CHANNEL := PTO_PWM_CHANNEL)
CAL    FB_PtoPwm ( ENABLE := TRUE, PTO_MODE := TRUE,
APPEND := xPtoAppend, TB_IDX := usiPtoTbIdx, CT := uiPtoCt,
DELTA := iPtoDelta, PC := udiPtoPc,
CHANNEL := PTO_PWM_CHANNEL
|
xPtoReady := READY, xPtoBusy := BUSY, usiPtoPwmError := ERROR)

RunPtoMode:
CAL    FB_PtoPwm ( ENABLE := TRUE, PTO_MODE := TRUE, APPEND := FALSE,
CHANNEL := PTO_PWM_CHANNEL
|
xPtoReady := READY, xPtoBusy := BUSY, usiPtoPwmError := ERROR)
JMP    ProgExit

(* ----- Run PWM Mode ----- *)
StartPwmMode:

                FALSE                preset output state, this state is
        D                *                                              )
                xPtoPwmO         used when PTO Generator isn't
            ut                   running
        T                *                                              )
```

```
                FALSE                reset state flags *)
        D                *

                xPtoReady
        T

                xPtoBusy
        T


ST      xPwmReady

ST      xPwmBusy

CAL     FB_PtoPwm ( ENABLE := FALSE,

CHANNEL := PTO_PWM_CHANNEL)

CAL     FB_PtoPwm    ( ENABLE := TRUE, PTO_MODE := FALSE,

APPEND := xPwmAppend, TB_IDX := usiPwmTbIdx, CT := uiPwmCt,

PT := uiPwmPt,

CHANNEL := PTO_PWM_CHANNEL

|

xPwmReady := READY, xPwmBusy := BUSY, usiPtoPwmError := ERROR)

RunPwmMode:

CAL     FB_PtoPwm    ( ENABLE := TRUE, PTO_MODE := FALSE, APPEND := FALSE,

CHANNEL := PTO_PWM_CHANNEL

|

xPwmReady := READY, xPwmBusy := BUSY, usiPtoPwmError := ERROR)

JMP     ProgExit
```

(* ----- Cycle End ----- *) ProgExit:

RET END_PROGRAM

3.3.85 PTO



用于执行脉冲计时器。典型的应用例子是控制步进电机的单脉冲，实现步进电机控制的斜坡函数

操作数的定义：

TABLE：参数设置表，包含要生成的脉冲序列；

ENABLE：启用或禁用功能块，true 为启用，false 为禁用；；

TB_IDX：为脉冲发生器设置基准周期，0=800ns，1=1ms；

RECORDS：存储器中的地址，用于读取和写入数据(需设置参数模式)；

CHANNEL：需要使用的通道号；

DO_IDX：设备号，需要设置为 0；

TABLE：读取一个字节的数据；

READY：表示脉冲发生器的输出状态，True 代表已初始化，False 代表未初始化；

BUSY：表示脉冲发生器的输出状态，True 代表脉冲发生器忙，False 代表脉冲发生器空闲；

ERROR：错误代码说明了有关功能块执行结果的信息。可能的错误代码定义如下:

0:在执行功能块时，没有发生错误

1:在执行功能块时，出现硬件错误

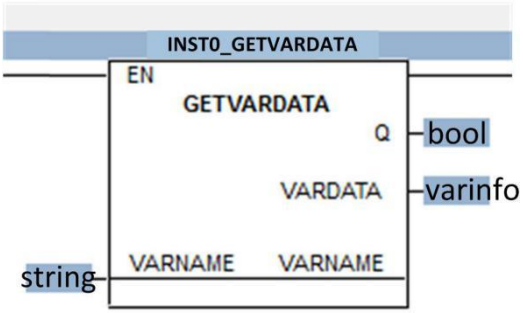
2:当调用函数块时，设置了无效的设备号

4:当调用函数块时，设置了无效的模式

8:设置的地址太大，超出了最大可用存储区域

16:指针指向了一个不支持的数据类型

3.3.86 GETVARDATA



操作数的定义：

EN：使能输入

VarName：待访问变量的字符串名称

Q：如果变量是有效的，则输出1

VarData：有效的变量信息

描述：

输入处指定的变量位于内存地址空间中并在输出处显示变量的信息，如果变量无法寻址，则输出Q为0。OpenPCS必须具有按名称寻址的功能，这项功能实现的前提是必须在资源选项中生成一个映射文件。

3.3.87 GETVARFLATADDRESS

查询变量地址

操作数的定义：

EN：使能输入。

VARNAME：待查询变量的字符串名称。

Q：若变量是有效的，则输出1。

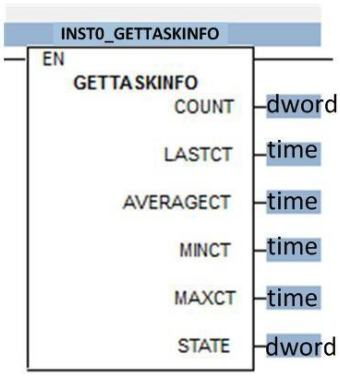
VARADDRESS：待查询变量的地址。

描述：

如果输入处指定的变量在内存地址空间中，则会输出内存空间的地址。如果变量不在内存地址空间中，输出Q就会为0

为了OpenPCS能够通过名称寻址，必须生成一个内存映射文件（在资源选项中）。内存地址空间只可在当前程序周期中使用，不可被其他程序存储或调用。

3.3.88 GETTASKINFO



获取任务周期各执行时间

操作数的定义：

COUNT：执行此任务的周期数；

LASTCT：上一个周期所需的时间；

AVERAGECT：执行所需的平均时间；MINCT：执行所需的最小时间；

MAXCT：执行所需的最大时间；

STATE：尚未使用；

描述：

该功能块返回有关当前任务的最后一个周期的执行时间的信息。此功能块没有输入参数。

四、基础功能应用例程讲解

因前文已经介绍过如何打开程序文件、编译程序、下载程序等操作，故本章不再详细说明，本章内容仅对例程的编程思路、编程逻辑以及例程实现的功能进行解读。

1. 跑马灯实验例程讲解

```
VAR

    v1:INT:=0;          (*定义整形变量，初始值为0*)
    v2:INT:=0;
    oled at%Q0.0:Byte;  (*输出Q0.0~Q0.7共8位*)

END_VAR
```

```

IF v1<1000 THEN         (*如果v1<1000,则自加1*)
    v1:=v1+1;

ELSE                    (*如果v1=1000,则把v1赋值为0*)
    v1:=0;              (*v2自加1*)
    v2:=v2+1;
    if v2>=255 then     (*如果v2加到255，则把v2赋值为0*)
        v2:=0;
    end_if;
    oled:=int_to_byte(v2);  (*将数据类型为int的v2转化为byte赋值给oled*)

end_if;
```

该程序中，V1 变量初始值为 0，当 V1 小于 1000 时，V1 自加 1，当 1000 个扫描周期过后，V1 值为 1000，进入 ELSE，V1 归 0，V2 开始自加 1，由于 V1 已经归零，所以下个周期继续进入 V1 自加 1，该段程序实际效果为 V1 每加到 1000 一次，V2 就加 1。

当 V2 每自加到 255 时，将 V2 重新归零，并每个周期都将 V2 的数据类型转换成 BYTE 赋值给 oled 来输出跑马灯。

进行数据转换是因为 BYTE 不能直接数学运算，所以需要使用 INT 来自加，实现跑马灯的现象。

2. 输入输出实验

```
    DI0 AT%I0.0:BOOL; (* 数字量输入1 *)
    DI1 AT%I0.1:BOOL; (* 数字量输入2 *)
    DI2 AT%I0.2:BOOL; (* 数字量输入3 *)
    DI3 AT%I0.3:BOOL; (* 数字量输入4 *)
    DI4 AT%I0.4:BOOL; (* 数字量输入5 *)
    DI5 AT%I0.5:BOOL; (* 数字量输入6 *)
    DI6 AT%I0.6:BOOL; (* 数字量输入7 *)
    DI7 AT%I0.7:BOOL; (* 数字量输入8 *)

    DO0 AT%Q0.0:BOOL; (* 数字量输出1 *)
    DO1 AT%Q0.1:BOOL; (* 数字量输出2 *)
    DO2 AT%Q0.2:BOOL; (* 数字量输出3 *)
    DO3 AT%Q0.3:BOOL; (* 数字量输出4 *)
    DO4 AT%Q0.4:BOOL; (* 数字量输出5 *)
    DO5 AT%Q0.5:BOOL; (* 数字量输出6 *)
    DO6 AT%Q0.6:BOOL; (* 数字量输出7 *)
    DO7 AT%Q0.7:BOOL; (* 数字量输出8 *)

    AI0 AT%I0.0:int; (* 模拟量输入1 *) (* int形式的变量用于定义-32767~+32767的输入 *)
    AI1 AT%I2.0:int; (* 模拟量输入2 *)
    AI2 AT%I4.0:int; (* 模拟量输入3 *)
    AI3 AT%I6.0:int; (* 模拟量输入4 *)

    AO0 AT%Q1.0:uint; (* 模拟量输出1 *) (* uint形式的变量用于定义0~+65535的输出 *)
    AO1 AT%Q3.0:uint; (* 模拟量输出2 *)

    AO2 AT%Q5.0:int;  (* 模拟量输出3 *) (* int形式的变量用于定义-32767~+32767的输出 *)
    AO3 AT%Q7.0:int;  (* 模拟量输出4 *)

    v1:bool;(* 中间变量1 *)
    v2:bool;(* 中间变量2 *)
```

```
DO0:=DI0; (* 数字量输出=数字量输入 *)
DO1:=DI1;
DO2:=DI2;
DO3:=DI3;
DO4:=DI4;
DO5:=DI5;
DO6:=DI6;
DO7:=DI7;

if AI0>5000 then (* 电压/电流输入换算值大于5000 *)

 v1:=1;
 v2:=0;

else              (* 电压/电流输入换算值小于等于5000 *)

 v1:=0;
 v2:=1;

end_if;

AO0:=65535;       (* 输出换算值为65535的电压/电流 *)
AO1:=50000;       (* 输出换算值为50000的电压/电流 *)

AO2:=32767;       (* 输出换算值为32767的电压/电流 *)
AO3:=-32767;      (* 输出换算值为-32767的电压/电流 *)
```

```
(*IO模块定义表*)
(*1008  8路数字量输入*)
(*2008  8路数字量输出*)
(*3604  4路模拟量输入-5~+5V   16位即-32768~+32767*)
(*3624  4路模拟量输入-10~+10V 16位即-32768~+32767*)
(*3644  4路模拟量输入0~+20mA  16位即0~+65535*)
(*3654  4路模拟量输入4~+20mA  16位即0~+65535*)
(*3664  4路模拟量输入0~+5V    16位即0~+65535*)
(*3674  4路模拟量输入0~+10V   16位即0~+65535*)
(*4602  2路模拟量输出-5~+5V   16位即-32768~+32767*)
(*4622  2路模拟量输出-10~+10V 16位即-32768~+32767*)
(*4642  2路模拟量输出0~+20mA  16位即0~+65535*)
(*4652  2路模拟量输出4~+20mA  16位即0~+65535*)
(*4662  2路模拟量输出0~+5V    16位即0~+65535*)
(*4672  2路模拟量输入0~+10V   16位即0~+65535*)
(*1502  2路计数器输入         32位即0~4294967295  注：计数和频率均为32位*)
(*2204  4路继电器输出         8位                 注：2204位宽为8位，但只用到了前4位，后4位为空*)
(*2302  2路PWM输出  具体应用参考2302使用例程*)
```

该程序中，变量声明区体现了带有 IO 模块硬件地址的变量如何声明，程序编辑区体现了简单的语句使用，以及 IO 模块对应的数据类型及取值范围。

3. CAN 收发数据实验 ST

```
(*该程序实现了将每个读到的CAN帧信息在CANID加1后重新写出去*)

canCH:=2;                                   (*使用的CAN端口号，PLC-400/510的CAN端口号为2，PLC-core的CAN端口号为1或2*)

if not mCanInit then                        (*如果未初始化CAN口*)

       inst0_CAN_INIT(EN_IN :=1 ,           (*PLC的CAN口初始化功能块，EN_IN为使能；NETNUMBER为端口号，PLC-400/510为2；初始化功能块只在上电第一个周期启动，若*)
        NETNUMBER :=canCH ,
        BITRATE :=0 |
        xERROR:= ERROR);                    (*BITRATE: = 0表示1 MBit/s   BITRATE: = 1表示840 kBit/s   BITRATE: = 2表示700 kBit/s   BITRATE: = 3表示500 kBit/s   BITRATE: = 4表示400 kBit/s*)
        if xERROR=0 then                    (* BITRATE: = 5表示250 kBit/s   BITRATE: = 6 表示200 kBit/s   BITRATE: = 7 表示125 kBit/s   BITRATE: = 8 表示100 kBit/s   BITRATE: = 9 表示80 kBit/s*)
        mCanInit:=true;                     (* BITRATE: = 10表示50 kBit/s   BITRATE: = 11表示40 kBit/s   BITRATE: = 12表示20 kBit/s   BITRATE: = 13表示10 kBit/s*)
       end_if;                              (*如果CAN初始化功能块执行过程无错误，则CAN初始化完成标志置为1*)

else

      inst1_CAN_MESSAGE_READ8(EN_IN :=can_read  (*如果CAN初始化成功，则启动CAN读功能块*)
      , NETNUMBER := canCH |
      mCanConfirm := EN_OUT                     (*CAN读功能块读到CAN帧数据的标志*)
```

该段程序为 CAN 初始化功能块，功能块的使用在第三章已有讲解，本节不再详细赘述，调节波特率的参数可见程序注释，该段程序逻辑为如果 CAN 初始化完成标志位为 FALSE 则执行初始化，初始化 ERROR 为 0 则标志位变为 TRUE，且下次判断 IF，进入 ELSE 程序段。

```
16
17else
18
19      inst1_CAN_MESSAGE_READ8(EN_IN :=can_read  (*如果CAN初始化成功，则启动CAN读功能块*)
20      , NETNUMBER := canCH |
21      mCanConfirm := EN_OUT,                    (*CAN读功能块读到CAN帧数据的标志*)
22      mCANID := CANID,                          (*将读到的CANID存放到变量中*)
23      mEXT_FRAME := EXT_FRAME,                  (*判断读到的CAN帧是否为扩展帧，若为扩展帧则值为1，若为标准帧则值为0*)
24      mRTR_FRAME := RTR_FRAME,                  (*判断读到的CAN帧是否为远程帧，若为远程帧则值为1，若为数据帧则值为0*)
25      mDATALENGTH := DATALENGTH,                (*读到的CAN帧数据长度，即包含多少字节*)
26      mDATA0:= DATA0,                           (*将读到的CAN数据第1个字节存放到变量中*)
27      mDATA1:= DATA1,                           (*将读到的CAN数据第2个字节存放到变量中*)
28      mDATA2 := DATA2,                          (*将读到的CAN数据第3个字节存放到变量中*)
29      mDATA3:= DATA3,                           (*将读到的CAN数据第4个字节存放到变量中*)
30      mDATA4:= DATA4,                           (*将读到的CAN数据第5个字节存放到变量中*)
31      mDATA5:= DATA5,                           (*将读到的CAN数据第6个字节存放到变量中*)
32      mDATA6 := DATA6,                          (*将读到的CAN数据第7个字节存放到变量中*)
33      mDATA7:= DATA7,                           (*将读到的CAN数据第8个字节存放到变量中*)
34      xERROR:= ERROR);                          (*CAN读功能块执行过程中的错误信息*)
35
36      if mCanConfirm=1 then                     (*只有在CAN读完成标志为1时读到的ID和数据才是实际值*)
37       canID:=mCANID;                           (*获取实际的CANID*)
38       canDAT0:=mDATA0;                         (*获取实际CAN数据的第1个字节*)
39       canDAT1:=mDATA1;                         (*获取实际CAN数据的第2个字节*)
40       canDAT2:=mDATA2;                         (*获取实际CAN数据的第3个字节*)
41       canDAT3:=mDATA3;                         (*获取实际CAN数据的第4个字节*)
42       canDAT4:=mDATA4;                         (*获取实际CAN数据的第5个字节*)
43       canDAT5:=mDATA5;                         (*获取实际CAN数据的第6个字节*)
44       canDAT6:=mDATA6;                         (*获取实际CAN数据的第7个字节*)
45       canDAT7:=mDATA7;                         (*获取实际CAN数据的第8个字节*)
46       can_read:=0;
47       can_write:=1;                            (*CAN成功读取后，将读指令赋值0，写指令赋值1*)
48      end_if;
49
50
51    if can_write=1 then                         (*如果写指令为1，则将之前读到的CANID加1*)
52        mCANID:=mCANID+1;
53    end_if;
54
```

进入 ELSE 程序段后，执行 CAN_READ，CAN 读取功能块，其中 mDATA0-7 为读到的数据，只有在读到的一瞬间为实际数据，其余时间为随机数，mCanConfirm 为读完成标志位，只有在读到数据的一瞬间置 TRUE，下一周期马上回到 FALSE，所以做了个判断当标志位为 1 那个周期，立即把读到的数据赋值给另一个变量来将数据保存住，且关闭读取使能，打开写出使能。写出使能为 1 时将 CANID+1，避免写出时因 ID 相同与接收数据撞帧。（注：此处关闭读取使能是例程的逻辑，CAN 读取功能块使能常置 1 可实现一直读取。）

```
50
51    if can_write=1 then                       (*如果写指令为1，则将之前读到的CANID加1*)
52        mCANID:=mCANID+1;
53    end_if;
54
55        inst2_CAN_MESSAGE_WRITE8(             (*调用CAN写功能块将读到的CAN帧数据的CANID加1后写出去*)
56        EN_IN :=can_write ,                   (*CAN写功能块的使能条件为can_write=1*)
57        NETNUMBER :=canCH ,                   (*PLC-400/510的CAN写功能块端口号为2，PLC-core的CAN写端口号为1或2*)
58        CANID :=mCANID ,                      (*将CAN读到的帧ID加1写出去，将读和写的帧ID作区分*)
59        EXT_FRAME := mEXT_FRAME,              (*CAN写的帧信息，1位扩展帧，0为标准帧*)
60        RTR_FRAME := mRTR_FRAME ,             (*CAN写的帧信息，1为远程帧，0为数据帧*)
61        DATALENGTH :=mDATALENGTH ,            (*CAN写的帧长度，CAN数据的字节数*)
62        DATA0 :=mDATA0 ,                      (*CAN写数据的第1个字节*)
63        DATA1 :=mDATA1 ,                      (*CAN写数据的第2个字节*)
64        DATA2 :=mDATA2 ,                      (*CAN写数据的第3个字节*)
65        DATA3 :=mDATA3 ,                      (*CAN写数据的第4个字节*)
66        DATA4 :=mDATA4 ,                      (*CAN写数据的第5个字节*)
67        DATA5 :=mDATA5 ,                      (*CAN写数据的第6个字节*)
68        DATA6 :=mDATA6 ,                      (*CAN写数据的第7个字节*)
69        DATA7 :=mDATA7 |                      (*CAN写数据的第8个字节*)
70        mCONFIRM := EN_OUT,                   (*CAN写成功的标志位*)
71        xERROR:= ERROR);                      (*CAN写过程中的错误信息*)
72
73     if mCONFIRM=1 then                       (*如果CAN写成功后，将CAN读指令置为1，重新读取CAN帧信息；将CAN写指令置为0，关闭写通道*)
74
75        can_read:=1;                          (*程序在下一个扫描周期重新执行先读后写的过程*)
76        can_write:=0;
77     end_if;
78
79end_if;
80
81
```

CAN 读程序段处理结束且 CAN ID+1 后，进入 CAN 写功能块，该功能块的使能触发条件为上升沿触发，关于上升沿触发的含义详见本书第 39 页，本节不再赘述，CAN 写功能块识别到使能变量的上升沿之后将数据发出去，当该帧数据发送完毕，CAN 写成功标志位将置 1（与 CAN 读成功标志位相同

只有在本周期为 1，下一个周期又会变回 0），当标志位置 1 时关闭 CAN 写使能以便下次触发上升沿，打开 CAN 读使能开启读取。

4. 串口 RS232、485 实验 ST

```
(*本示例程序实现了串口数据的先读后写，且写的内容为先前读到的数据*)

mPORT:=2;  (*串行端口号，1为RS232,2为RS485*)

if xInitOk=false then  (*如果未初始化串口*)
inst0_USART_INIT(BAUD :=115200 , DATABITS :=8 , PARITY :=0 , STOPBITS :=1 , PROTOCOL :=0 , ENABLE :=1 , PORT :=mPORT  | xE := ERROR);  (*启用串口初始化功能块，BAUD为波特率，
                                                                                                                                    DATABITS为数据位数，PARITY为奇偶校验，0表示无校验*)
    if xE=0 then       (*如果串口初始化功能块执行无错误*)
        xInitOk :=TRUE; (*将串口初始化状态置1，不再执行串口初始化*)

    end_if;
end_if;
```

该例程实现了将串口收到的数据原封不动的发回去，此段程序为串口初始化，所使用的串口初始化功能块参数再第三章已有讲解，各参数含义也可见注释，该段程序含义为，如果初始化未完成，则调用初始化功能块，若初始化无错误，则初始化完成。

```
pDataObject:=&abDataBuffer;  (*指针指向数组*)

if xInitOk then    (*在初始化成功前提下*)


    if xReadStart then  (*如果启动串口读取模式*)

        inst0_USART_READ_BIN(ENABLE :=0 ,  PORT :=mPORT  |   xERROR:= ERROR); (*以使能ENABLE为0调用串口读取功能块，与接下来以ENABLE为1调用功能块相配合，生成使能ENABLE的上升沿指令*)
        xWaitForReceipt:=true;   (*置位串口预读取标志*)
        xRdBinConfirm:=false;    (*复位串口读取功能块执行完成标志*)
        xReadStart:=false;       (*复位串口模式的启动指令*)
    end_if;

    if xWaitForReceipt then     (*如果串口预读取标志为1*)
        inst0_USART_READ_BIN(ENABLE :=1 , PTR_RXDATA :=pDataObject , MAXLENGTH :=0 , ETXCHR :=10, CHKETX :=1, PORT :=mPORT  | xRdBinConfirm:= CONFIRM,  iRxDataSize:= RXLENGTH,  xERROR:= ERROR);
                             (*以使能ENABLE为1调用功能块，将读取到的数据存放到数组abDataBuffer中，读取的最大长度为20个字节，实际读取长度赋值到iRxDataSize中*)
        if xRdBinConfirm then   (*如果串口读取功能块执行完成*)
            var4:=iRxDataSize;  (*将串口读到的实际数据长度赋值到该变量*)
            xWaitForReceipt:=false;  (*复位串口预读取标志*)
        end_if;

    end_if;

    if xRdBinConfirm then       (*如果串口读取功能块执行完成*)

        inst0_USART_WRITE_BIN(ENABLE :=0 ,  PORT :=mPORT |  xERROR:= ERROR);  (*以使能ENABLE为0调用串口发送功能块，与接下来以ENABLE为1调用功能块相配合，生成使能ENABLE的上升沿指令*)

        xRdBinConfirm:=false;    (*复位串口读取功能块完成标志*)
        xTransmitting:=true;     (*启动串口发送模式*)

    end_if;

    if xTransmitting then       (*如果串口发送模式已启动*)

        inst0_USART_WRITE_BIN(ENABLE :=1 , PTR_TXDATA :=pDataObject , TXLENGTH :=iRxDataSize, PORT :=mPORT | xWrBinConfirm := CONFIRM,  xERROR:= ERROR);
                             (*以使能ENABLE为1调用串口发送功能块，将先前读到数组abDataBuffer中的数据发送出去，发送长度为串口先前读取数据的实际长度*)
        if xWrBinConfirm then   (*如果串口发送完成*)
            IW:=IW+1;
            xTransmitting :=false;  (*关闭串口发送模式*)
            xReadStart:=true;       (*启动串口读取模式，继续先读后写的循环*)
        end_if;

    end_if;

end_if;
```

初始化完成后进入串口工作模式，第一行为指针型变量指向数组，ST 语言中的指针变量只能指向数组，因数组无法在功能块中直接使用，所以需要功能块的指针引脚指向数组，来使用数组调用数据。

进入串口读取模式，使能为 1 时等待读取数据，当读取数据完毕，读取完成标志位会置 1，此时调用串口写功能块（使能为 0），调用完毕将使能置 1 再次调用触发上升沿将数组中的数据写出。

5. TCP SERVER 通信实验

该例程实现了 PLC 作为 TCP SERVER，与 TCP CLIENT 设备通信，并将收到的数据原封不动的发回去。

```
(*本示例程序实现了PLC作为TCP_SERVER与TCP_CLIENT通讯，将读取到的数据原样发送出去*)


if lanInit=false then       (*如果网口未初始化*)

mIP := LAN_ASCII_TO_INET('192.168.1.30');              (*将PLC的IP地址转化成ASCII码*)
mNetMask := LAN_ASCII_TO_INET('255.255.255.0');        (*将子网掩码转化成ASCII码*)
mGateWay := LAN_ASCII_TO_INET('192.168.1.1');          (*将网关转化成ASCII码*)
    inst0_LAN_INIT(ENABLE :=true ,                     (*调用网口初始化功能块*)
            HOSTNAME := 'PLCCore',                     (*主机名*)
            IP := mIP,                                 (*PLC的IP地址*)
            NETMASK :=mNetMask,                        (*PLC的子网掩码*)
            GATEWAY :=mGateWay ,                       (*PLC的网关*)
            NETNUMBER:=1                               (*网络端口号*)
            |  mConfirm:= CONFIRM,                     (*初始化功能块执行完成标志*)
             mError:= ERROR,                           (*功能块执行完成报的错误信息*)
             mErrorinfo:= ERRORINFO);                  (*功能块执行过程中发生终止报的错误信息*)
            lanInit:=true;                             (*将网口初始化状态置1，不再进行网口初始化*)
else                            (*如果网口已初始化*)
```

该段程序为网口初始化功能块，各个参数含义可见程序注释，其中 mIP 代表 PLC 的 IP，如果没有对 PLC 的 IP 进行过修改，则此处填写 PLC 的出厂默认 IP：192.168.1.30 即可，如果进行过修改，则按照用户所修改的 IP 填写（注：此处并非 IP 填写多少，PLC 的设备 IP 就会被修改为多少，如需修改 IP，请联系广成科技技术工程师）。子网掩码和网关也按照用户的组网需求填写即可。

```
            lanInit:=true;                             (*将网口初始化状态置1，不再进行网口初始化*)
else                            (*如果网口已初始化*)

    if mTcpCreateOK=false then    (*如果未建立TCP_SERVER*)
        inst4_LAN_TCP_SERVER_CREATE(PORT :=8089 ,   (*生成一个TCP_SERVER,端口号8089*)
         ENABLE := true,
         NETNUMBER := 1                                (*网络端口号为1*)
         |
         mSocket:= SOCKET_ID,                          (*mSocket存放SOCKET_ID*)
         mConfirm:= CONFIRM,
         mError:= ERROR,                               (*执行完成报的错误信息*)
         mErrorinfo:= ERRORINFO);                      (*执行过程中终止报的错误信息*)

         if mSocket>=0 then                            (*如果SOCKET_ID成功生成，表示TCP_SERVER成功建立*)
         mTcpCreateOK:=true;                           (*置位TCP_SERVER建立成功标志，后续不再建立TCP_SERVER*)
         end_if;

    end_if;

end_if;
```

初始化成功后，创建一个 TCP SERVER，可设置端口号，创建完毕后，SOCKET ID 会大于 0，此时跳出创建，进入主程序。

```
pDataObject:=&abDataBuffer;     (*指针指向网口读写数据的地址*)

if mTcpCreateOK=true then       (*如果TCP_SERVER建立成功*)


    inst5_LAN_GET_TCPCONNECT_SOCKET(SOCKET_ID := mSocket, NETNUMBER := 1 |  mClientSocket:= CLIENT_SOCKET_ID,  mPeer:= PEER_ADDR,  mPeerPort:= PEER_PORT,  mError:= ERROR);
    (*建立TCP_SERVER与TCP_CLIENT的SOCKET连接,SOCKET_ID为先前生成的ID，mClientSocket用于存放TCP_CLIENT的SOCKET_ID，mPeer存放网络对端地址，mPeerPort存放网络对端的端口号*)

    if mClientSocket>=0 then   (*如果TCP_CLIENT的SOCKET_ID获取成功*)

        inst6_LAN_TCP_RECV_BIN(CLIENT_SOCKET_ID :=mClientSocket ,   (*调用网口读取数据功能块，指针指向数据存储地址，最大读取长度为1500，实际读取长度存到mRxLen中*)
            PTR_RXDATA :=pDataObject ,
            MAXLENGTH := 1500,
            ENABLE :=1 ,
            NETNUMBER := 1 |  mRxLen:= RXLENGTH,  mConfirm:= CONFIRM,  mError:= ERROR,  mErrorinfo:= ERRORINFO);

        if mRxLen>0 then       (*如果网口读取数据成功*)
            tmplen:=mRxLen;
            inst7_LAN_TCP_SEND_BIN(CLIENT_SOCKET_ID :=mClientSocket ,  (*调用网口发送数据功能块，数据内容与读取到的内容相同，数据长度为实际读到数据的长度*)
                PTR_TXDATA :=pDataObject ,
                TXLENGTH := mRxLen,
                ENABLE :=1 ,
                NETNUMBER :=1  |  mConfirm:= CONFIRM, mError := ERROR,  mErrorinfo:= ERRORINFO);

        end_if;
    end_if;
end_if;
```

该段程序为该例程的主程序，首先建立与 TCP CLIENT 的连接，当有 CLIENT 连接时，mClientSocket 会大于 0，此时调用 TCP 读，读取该 CLIENT 发送来的数据，保存在指针指向的数组里，当收到数据的长度大于 0 时启动 TCP 写功能块，将指针指向的数组里的数据发出去。（视频讲解详情可见 https://www.bilibili.com/video/BV1Sq4y1E7Cn?spm_id_from=333.999.0.0 该链接讲解包含了 TCP SERVER/TCP CLIENT 以及 UDP 通讯案例，故本文不再过多赘述）。