固高科技（深圳）有限公司

地　　址：深圳市高新技术产业园南区深港产学研基地西座二层 W211 室

电　　话：0755-26970823　26970819　26970824

传　　真：0755-26970821

电子邮件：support@googoltech.com

网　　址：http://www.googoltech.com.cn

Googol Technology (HK) Ltd

Addr: Room 3639,Annex Building Hong Kong University of Science and Technology, Hong Kong

Tel: (852) 2358-1033

Fax: (852) 2358-4931

E-mail: info@googoltech.com

Web: http://www.googoltech.com

OtoStudio 在线帮助手册

（2012-02-28）

务必将此手册交给用户

- 非常感谢您选购 CPAC 控制器
- 在您使用之前，请仔细阅读此手册，确保正确使用。
- 请将此手册妥善保存，以备随时查阅。

版权声明

 注意	***运动中的机器有危险！使用者有责任在机器中设计有效的出错处理和安全保护机制，固高科技没有义务或责任对由此造成的附带的或相应产生的损失负责。***

前言

感谢选用固高运动控制器

为回报客户，我们将以品质一流的运动控制器、完善的售后服务、高效的技术支持，帮助您建立自己的控制系统。

固高产品的更多信息

固高科技的网址是 http://www.googoltech.com.cn 。在我们的网页上可以得到更多关于公司和产品的信息，包括：公司简介、产品介绍、技术支持、产品最新发布等等。

您也可以通过电话（0755－26970839）咨询关于公司和产品的更多信息。

技术支持和售后服务

您可以通过以下途径获得我们的技术支持和售后服务：

- 电子邮件：support@googoltech.com；
- 电话：(0755) 26970835
- 发函至：深圳市高新技术产业园南区园深港产学研基地西座二楼 W211 室
  固高科技（深圳）有限公司
  邮编：518057

用户手册的用途

用户通过阅读本手册，能够了解 CPAC 计算机可编程自动化控制器的控制功能，掌握运动控制指令的基本用法，熟悉特定控制功能的编程实现。最终，用户可以根据自己特定的控制系统，开发相应的应用软件，实现控制要求。

用户手册的使用对象

本编程手册适用于，具有 PLC 编程基础或 Windows 环境下使用 C 语言的基础。同时具有一定 PLC，或者运动控制工作经验，对伺服或步进电机控制的基本结构有一定了解的工程开发人员。

相关文件

关于 CPAC 控制器的安装和调试，请参见随产品配套的《OtoStudio 运动控制库编程手册》。可视化编程请参考《OtoStudio 可视化用户手册》，编程实例请参考《OtoStudio 用户编程实例》。

CPAC-OtoBox 系列自动化控制器用户手册

第一章　概述

1.1 简介

CPAC-OtoBox 系列自动化控制器，是将 PC 技术与运动控制技术相结合的产物。它以 Intel 标准 X86 架构的 CPU 和芯片组为系统处理器，采用高性能 DSP 和 FPGA 作为运动控制协处理器。在延续了固高科技运动控制器可以实现高性能多轴协调运动控制和高速点位运动控制的同时，可以实现普通 PC 机的所有基本功能，是客户理想的嵌入式一体化解决方案。它适用于广泛的应用领域，包括机器人、数控机床、木工机械、印刷机械、装配生产线、电子加工设备、激光加工设备以及 PCB 钻铣设备等。

CPAC-OtoBox 系列自动化控制器，提供计算机常见接口（如 PS2、USB、VGA，LAN）及运动控制专用接口（具体定义参见第 3 章）。CPAC-OtoBox 系列自动化控制器提供专用的软件开发与诊断工具 CPAC-OtoStudio, 它遵循 IEC61131-3 国际标准定义的编程语言，这包含了定义架构内部程序构架的语言的顺序功能表，和内部可控制的四种语言：指令表、梯形图、功能图和结构文本。通过模块化的编程方式，增加程序的重用性，降低错误率和提高效率。并且，IEC61131-3 标准建立了一个控制系统的架构。此外，CPAC-OtoStudio 软件包中还包括诸如任务配置和硬件配置，远程诊断，人机界面开发，以及数据采集等模块。

1.2 CPAC-OtoBox 系列自动化控制器型号及含义

该平台的总体型号为 CPAC-OtoBox，具体型号描述见下图:

1.3 电机控制系统的基本组成

使用固高科技生产的 CPAC-OtoBox 系列自动化控制器，搭建一个完整的电机控制系统时，一般需要以下几部分器件组成：

1、CPAC-OtoBox 系列自动化控制器；
2、步进电机或伺服电机；
3、步进电机或伺服电机驱动器；
4、驱动器电源；
5、+24V 直流电源（用于 CPAC-OtoBox 系列自动化控制器及端子板电源）；
6、原点开关、正/负限位开关（根据系统需要可选）；
7、显示器、输入设备或专用人机界面（根据系统需要可选）；
8、IO 扩展模块（根据系统需要可选）；

伺服电机既可以选择交流伺服电机也可以选择直流伺服电机。

控制伺服电机时：

如果使用的是 CPAC-OtoBox-X00-XXV 运动控制器的模拟量输出功能时，电机驱动器应设置为速度控制方式。

如果使用的是 CPAC-OtoBox-X00-XXV 运动控制器的脉冲量输出功能时，电机驱动器应设置为位置控制方式，且控制器和驱动器的脉冲模式设置要一致。

如果使用的是 CPAC-OtoBox-X00-XXG 运动控制器，电机驱动器应选为位置控制方式，且控制器和驱动器的脉冲模式设置要一致。如果还有疑问，可咨询您的电机供应商或与固高公司联系。

对于控制步进电机：

运动控制器提供两种不同的控制信号：正脉冲/负脉冲、脉冲+方向。在控制步进电机时，控制模式为开环控制，不需要编码器的反馈信号，运动控制器处于脉冲输出方式下时默认关闭编码器反馈通道。若用户外接编码器来监控实际位置，可在编程中调用相关函数打开编码器反馈通道。

采用CPAC-OtoBox系列自动化控制器的控制系统典型组成部分见图1-1：

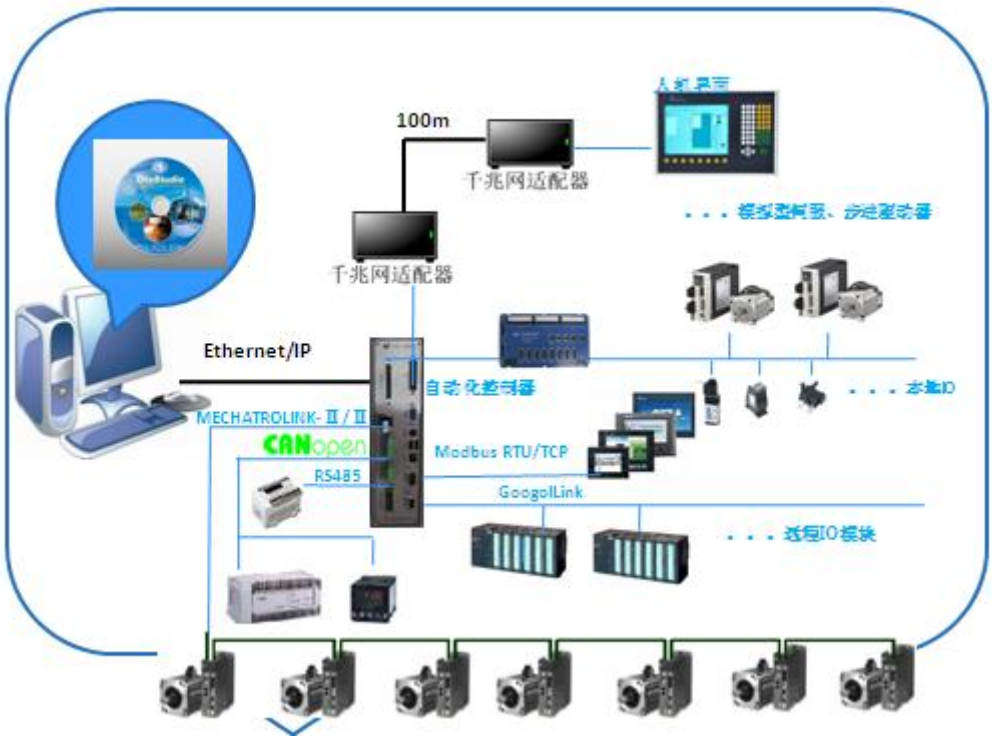


图 1-1 采用 CPAC-OtoBox 系列自动化控制器组成的控制系统框图

CPAC-OtoBox 系列自动化控制器典型应用，见图 1-2：

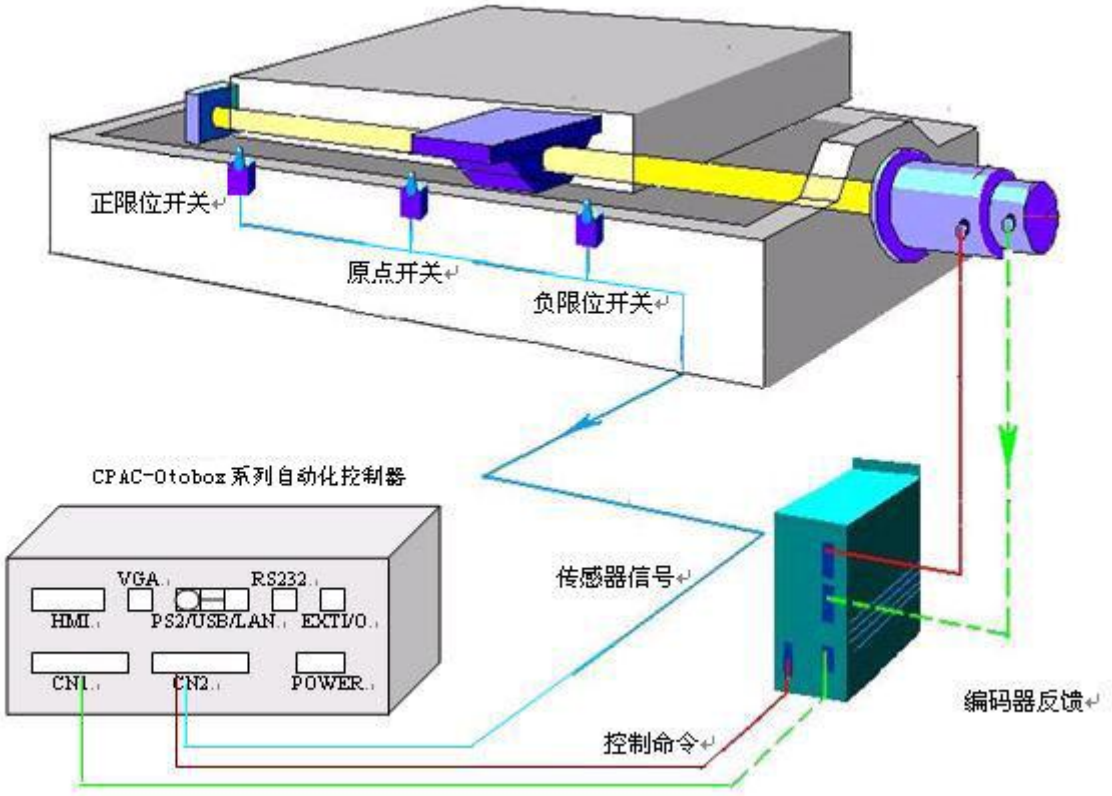


图 1-2 CPAC-Otobox 系列自动化控制器典型应用

第二章　安装与测试

2.1 开箱检查

打开包装前，请先查看外包装标明的产品型号是否与订购的产品一致。打开包装后，请首先检查运动控制器的表面是否有机械损坏，然后按照装箱清单或订购合同仔细核对配件是否齐备。如果运动控制器表面有损坏，或产品内容不符合，请不要使用，立即与固高科技或经销商联系。

注意	为了防止静电损害运动控制器，请在接触控制器电路或插/拔控制器之前触摸有效接地金属物体以释放身体所携带的静电荷。

2.2　CPAC-Otobox 系列自动化控制器的外形结构

CPAC-Otobox 系列自动化控制器的外形结构如图 2-1 所示：

图 2-1　CPAC-Otobox 系列自动化控制器外形结构图

注：此外形结构图为示意图，用户需根据所选用的具体型号对照接口定义检查是否正确。详细的接口定义请查看“第三章 CPAC-Otobox 系列自动化控制器接口定义”。

2.3 系统安装

请按照以下安装步骤建立控制系统：

步骤 1：将 CPAC-Otobox 系列自动化控制器连接标准输入输出设备、+24V 直流电源，系统上电。

步骤 2：建立主机与运动控制器的通讯

步骤 3：连接电机和驱动器

步骤 4：连接运动控制器和端子板（只针对-L2 运动控制器）

步骤 5：连接驱动器、系统输入/输出和 CPAC-Otobox 自动化控制器（或端子板）

2.3.1 步骤 1：连接标准输入输出设备、+24V 直流电源，系统上电

CPAC-Otobox 系列自动化控制器为用户提供了构成 PC 系统的标准输入输出设备接口，如 VGA、PS2、USB 等，用户可将显示器、键盘，鼠标等通用输入输出设备连接到这些接口上以组成 PC 系统。此外用户还需要提供一个 24V，至少 3A 的直流电源为其供电。直流电源接到控制器 POWER 接口上，接通后控制器上的 2 个 LED 指示灯亮起，表明 CPAC-Otobox 自动化控制器已上电工作。

另外在电源接口上提供了一个与 CPAC-Otobox 自动化控制器外壳连通的 PE（保护地）接口，用户可根据自己的电器系统需要，将其与其它外部地（机壳地、大地等）和（或）运动控制器内部地（数字地、+24V 参考地）连通。电源连接如图 2-2 所示。

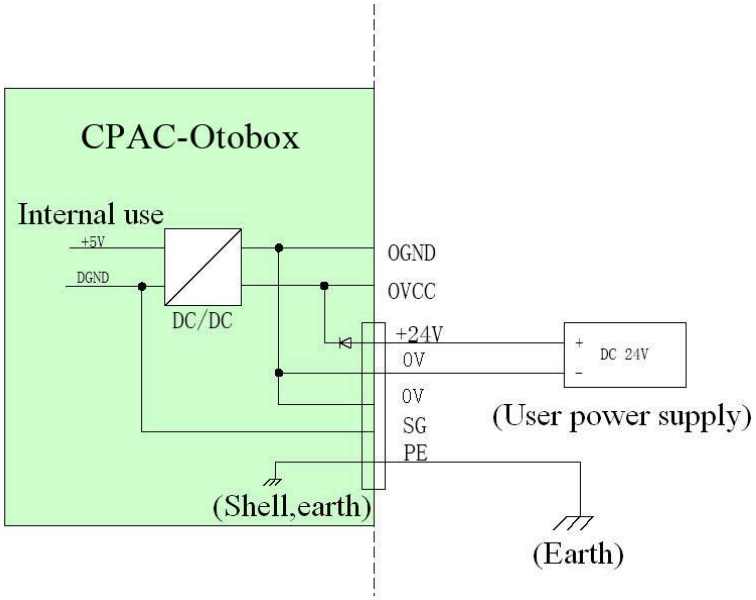


图 2-2 CPAC-Otobox 自动化控制器电源连接图

注意	为了防止人身触电事故、保证电气设备正常运行，请务必将电源端子的 PE 端通过接地线与大地保持良好连接！

2.3.1.1 用户定制 Windows CE 操作系统恢复方法

1. 用户需准备可作启动盘的 U 盘一个，并拷贝产品光盘中的 GHOST.exe 及 WINCE.gho 文件至此 U 盘。
2. 将 U 盘与 CPAC-Otobox 系列自动化控制器通过 USB 口连接，系统上电。
3. 在启动过程中按下 DEL 键进入 BIOS，将“First Boot Device”设置为“USB-HDD”启动。
4. 保存 BIOS 设置，重启电脑。
5. 待 CPAC-Otobox 自动化控制器再次启动后，执行 GHOST.exe 文件，进入 GHOST 安装界面（如图 2-3 所示）。

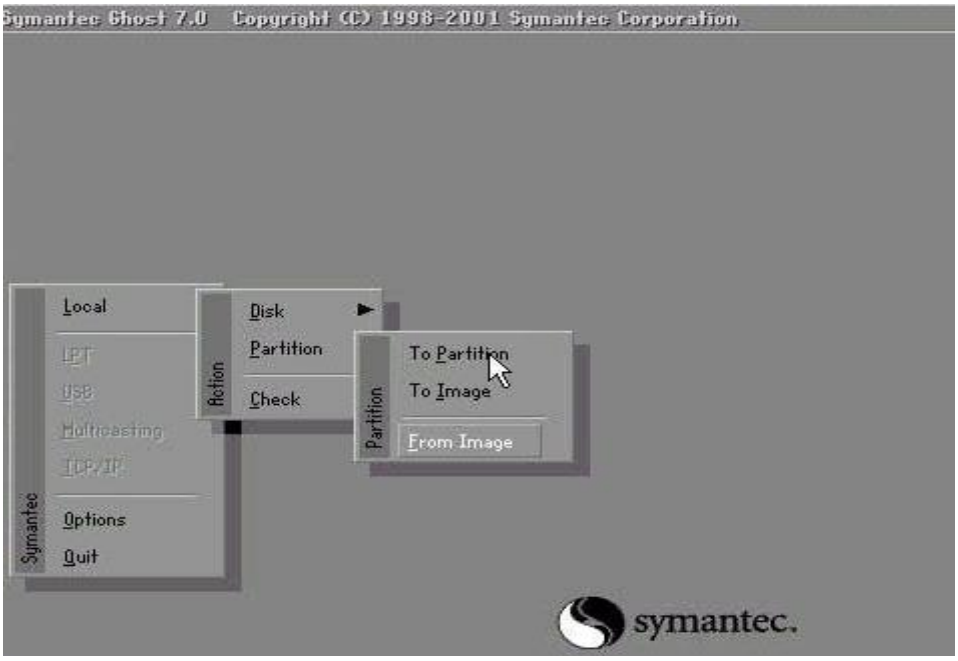


图 2-3 GHOST 安装界面

依次选择“Local”->“Disk”->“Form Image”按回车，再选择“*.gho”文件（如图 2-4 所示）。

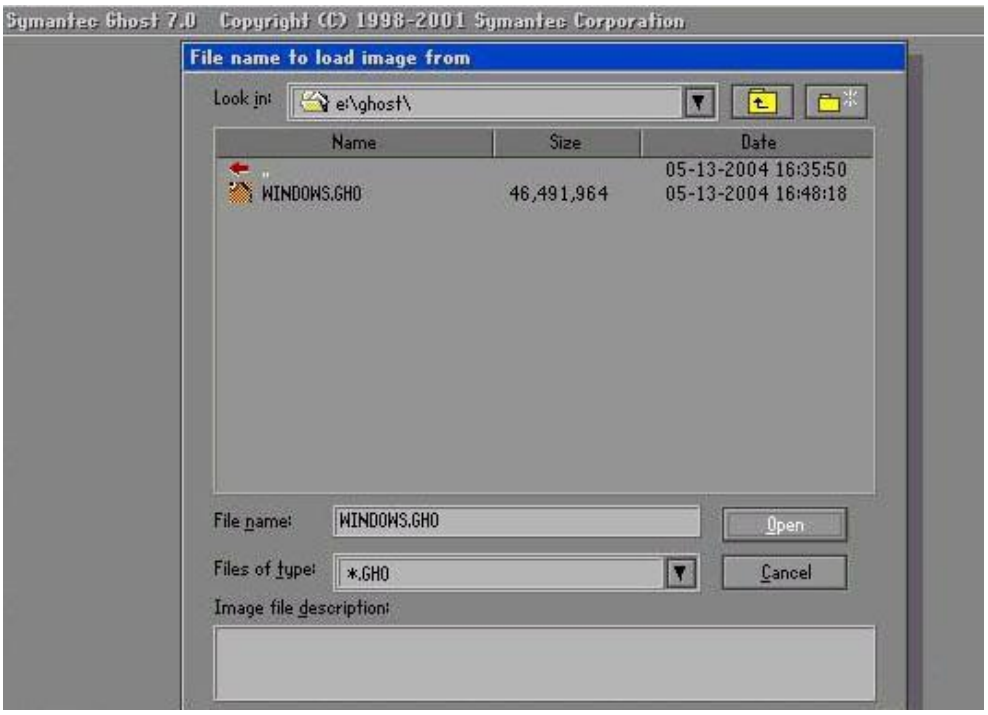


图 2-4 选择“*.gho”文件界面

之后一直选择“YES”，直到进入系统安装界面（如图 2-5 所示）。

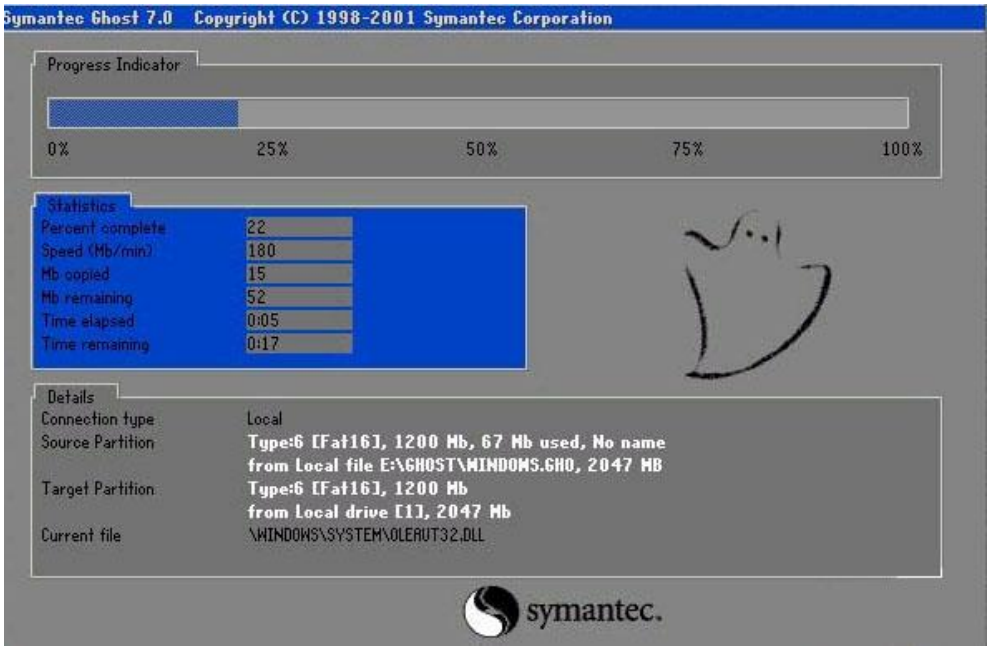


图 2-5　系统安装界面

6．待安装完成之后重启电脑，即可进入 WINCE 操作系统界面。

注：只有客户要求定制 WINCE 系统时，产品光盘中才会出现 GHOST.exe 及 WINCE.gho 文件。标准产品不提供以上文件。客户如需要定制的 WINCE 系统时，请与固高科技联系洽谈。

2.3.2 步骤 2：建立主机与运动控制器的通讯

使用 CPAC-Otobox 系列自动化控制器，使用 CPAC-Otostudio 软件，其中的联机选项。正确设置“通讯参数”，IP 地址设置为控制器的 IP，之后选择“联机”->“登录”选项，可以测试是否和自动化控制器建立了通讯。

如果“联机”选项下的“下载”，“运行”等选项为可选择，则证明控制器通讯正常。否则会提示“通讯错误”。如果出现通讯错误，用户可以参考*附录 C 故障处理*，确定问题所在，排除故障后重新测试。如果需要，请按照封面的公司信息与我们联系。

2.3.3 步骤 3：连接电机和驱动器

注意	为安全起见，建议用户初次使用控制器时，务必将电机与负载脱离开，在未完成控制系统的安装、调试前，**不要**将电机与任何机械装置连接。待调整控制器以及驱动器参数使得电机受控后，方可进行系统的连接，否则可能造成严重的后果。

在驱动器没有与运动控制器连接之前，连接驱动器与电机。用户必须详细的阅读驱动器的说明书，正确连接。按照驱动器说明书的要求测试驱动器与电机，确保其工作正常。

2.3.4 步骤 4：连接运动控制器和端子板（只针对-L2 型运动控制器）

关闭电源，取出产品附带的两条屏蔽电缆。一条屏蔽电缆连接 CPAC-Otobox 系列 L2 型

自动化控制器的 CN1 与端子板的 CN1，另一条屏蔽电缆连接 CPAC-Otobox 系列 L2 型自动化控制器的 CN2 与端子板的 CN2。为保证外部电路正常运行，必须连接这两条屏蔽电缆。应特别注意接口的对应，以及线缆和接口标识的匹配。

连接方式见图 2-6 和 2-7。

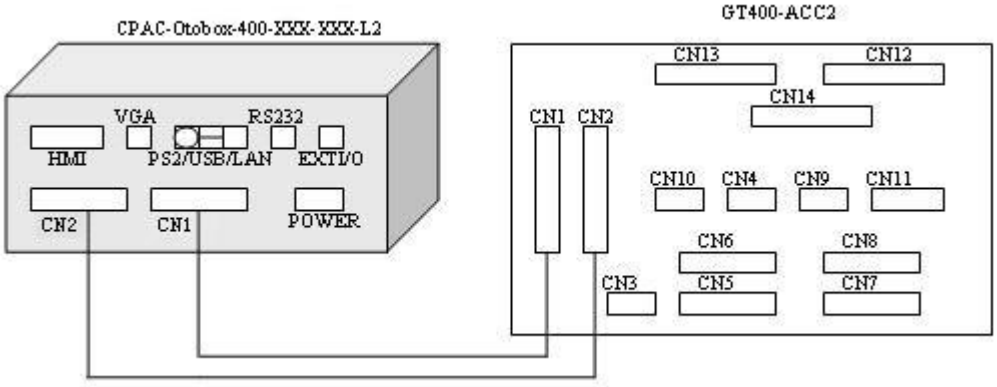


图 2-6 CPAC-Otobox-400-EXX-XXX-L2 型运动控制器与 GT-400-ACC2 端子板联接示意图

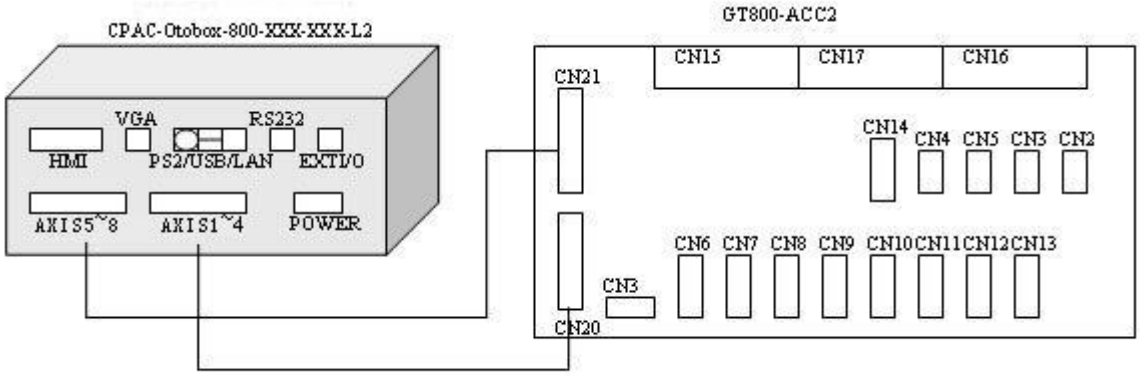


图 2-7 CPAC-Otobox -800-TXX-XXX-L2 型运动控制器与 GT-800-ACC2 端子板联接示意图

2.3.5 步骤 5：连接驱动器、系统输入/输出和 CPAC-Otobox 自动化控制器（或端子板）

2.3.5.1 连接电源

对于 CPAC-Otobox 系列-L2 型运动控制器，除将运动控制器与外部+24V 电源连接外，还需将外接端子板的 CN3 与外部 24V 电源连接，CPAC-Otobox 自动化控制器与端子板可共用同一电源，也可分开各自单独供电。详细的电源接口定义请参看第三章。接线图见图 2-8。

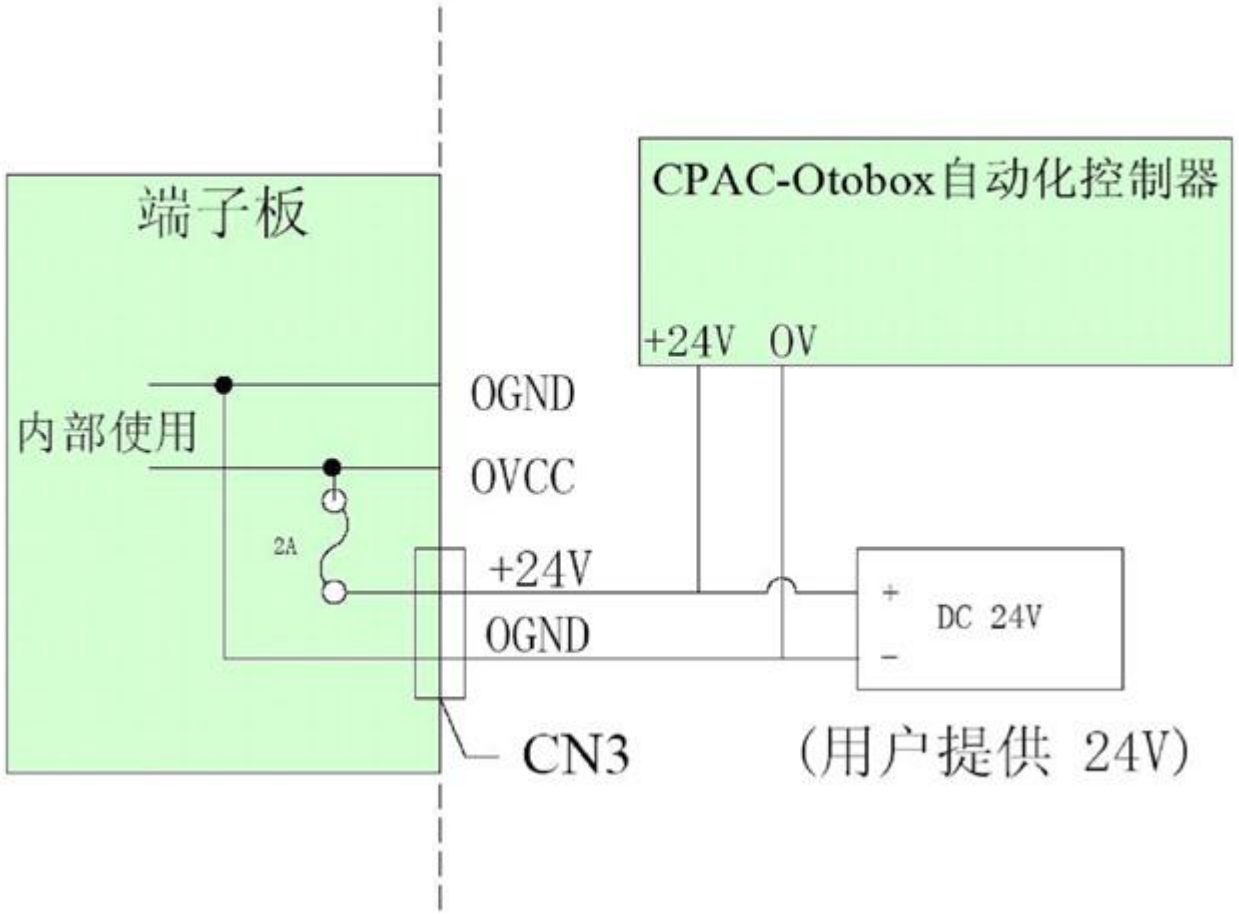


图 2-8 端子板电源连接图

注意	仔细了解运动控制器的接口信号和电机驱动器的接口定义，妥善连线并避免带电插拔接口。否则，**信号连接错误**或**带电操作**可能导致系统正反馈或硬件损坏使系统不能正常工作。

2.3.5.2 专用输入、输出连接方法

CPAC-Otobox 系列自动化控制器所提供的专用输入包括：驱动报警信号、原点信号和限位信号，专用输出包括：驱动允许，驱动报警复位。详细的专用 IO 接口定义请参看第三章。

专用输入输出的连接原理见图 2-9。

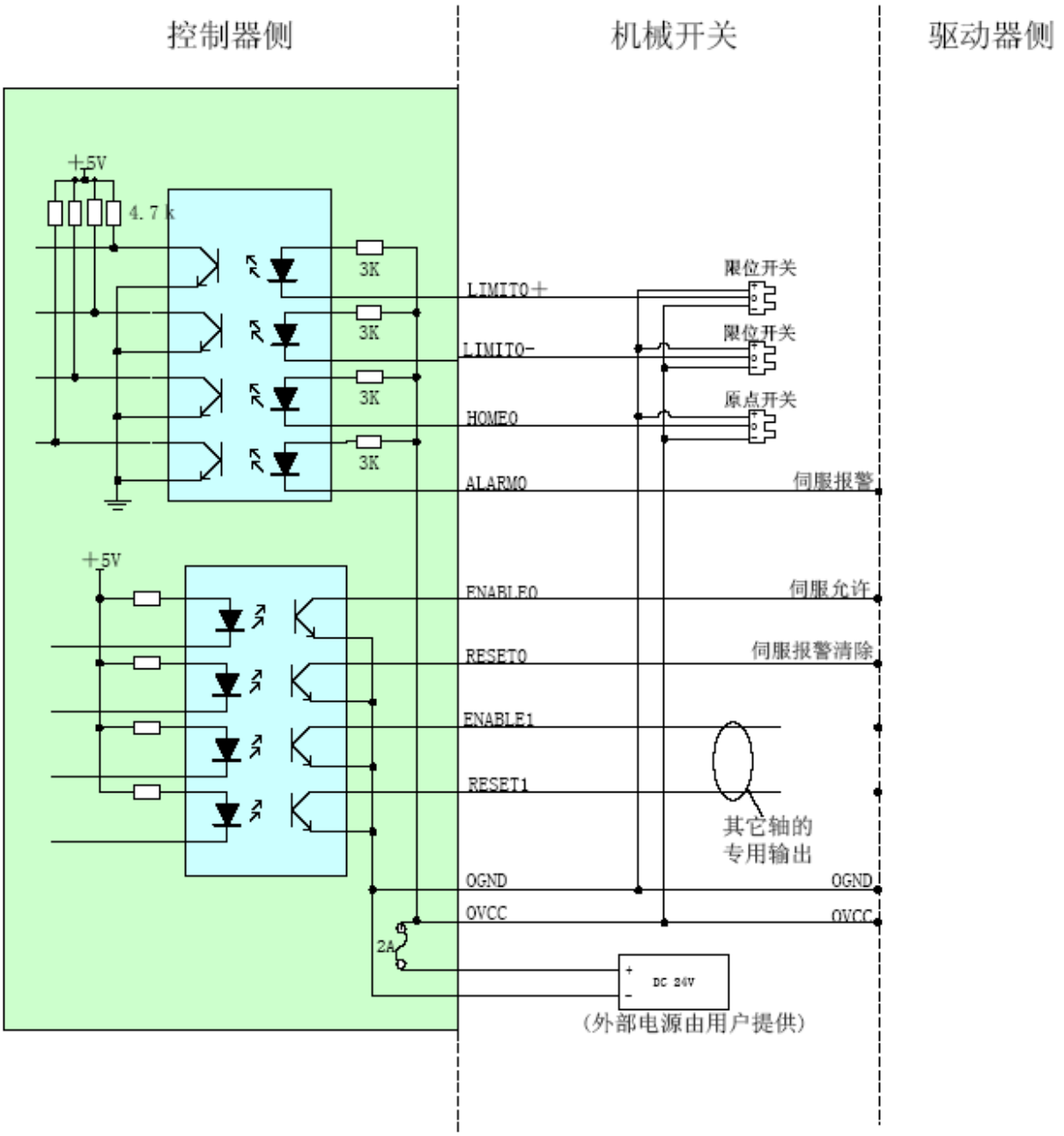


图 2-9 专用输入、输出信号连接图

注意	根据安全标准: 1. 驱动器报警输入信号为常闭状态,(用户不用时,请将该输入对 OGND 短接); 2. 系统的限位开关须接成常闭状态; 3. 原点开关为常开状态。

2.3.5.3 编码器输入连接方法

CPAC-Otobox 系列自动化控制器在每个轴上各提供一个编码器接口，此外还提供了 1~2 个辅助编码器接口（辅助编码器接口数量视具体型号而定）。详细的编码器接口定义请参看第三章

以上编码器接口可兼容双端输入和单端输入，连接方法见图 2-10 和 2-11。

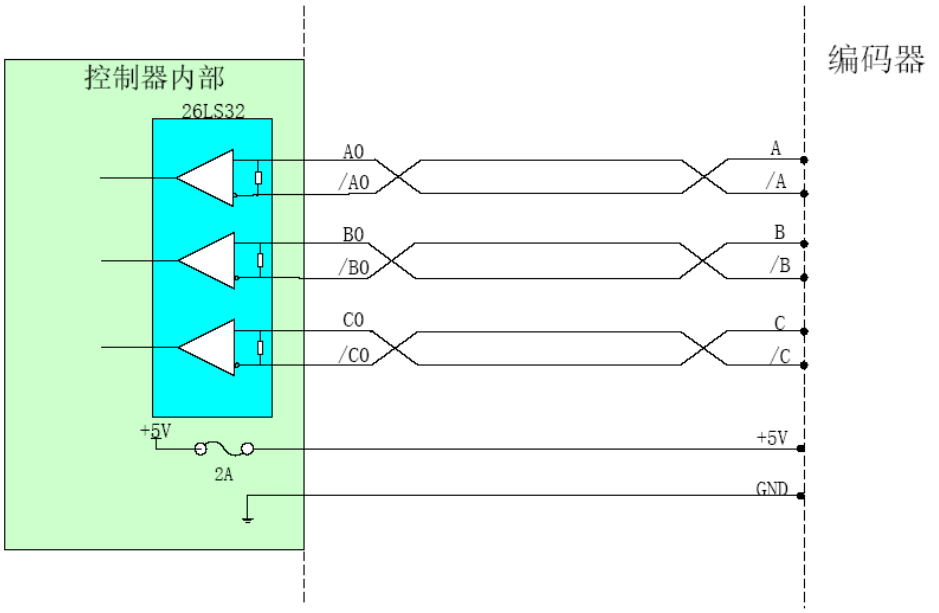


图 2-10 编码器双端输入信号连接图

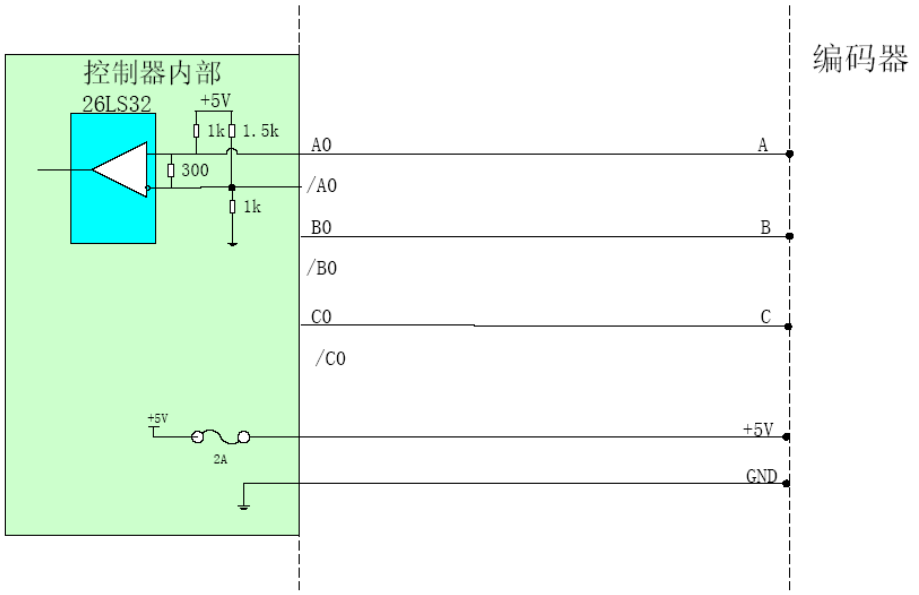


图 2-11 编码器单端输入信号连接图

2.3.5.4 控制输出信号连接方法

CPAC-Otobox 系列自动化控制器控制电机驱动器时，可以工作于脉冲量或模拟量输出模式（视具体型号而定），用户可以通过软件指令在 2 种输出模式之间相互切换（具体请参看编程手册）。详细的控制输出信号接口定义请参看第三章。

（1） 模拟量输出连接方法

模拟量输出通过驱动器接口的 PIN8 输出。参考地为+5V 电源地，驱动器接口的引脚定义请参见第三章，电气接线图参见图 2-12。

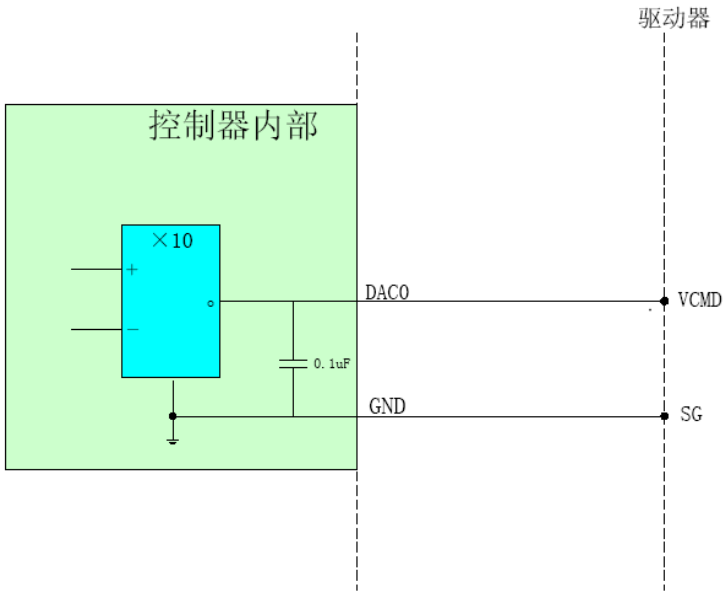


图 2-12 模拟量输出模式的电气接线图

(2) 脉冲输出连接方法

脉冲/方向输出信号通过驱动器接口的 9、22、23、11 脚输出，参考地为+5V 电源地。

在脉冲信号输出方式下，有两种工作模式，一种是脉冲+方向信号模式，另一种是正/负脉冲信号模式。默认情况下控制器输出脉冲+方向信号模式。用户可以通过软件指令在两种脉冲输出方式之间相互切换（具体请参看编程手册）。

在脉冲+方向信号模式下，引脚 23、11 输出差动的脉冲控制信号，引脚 9、22 输出差动的运动方向控制信号。

在正/负脉冲模式下，引脚 9、22 输出差动的正转脉冲控制信号，引脚 23、11 输出差动的反转脉冲控制信号。

如果驱动器需要的信号不是差动信号，将相应信号接于上述差动信号输出的正信号端（即引脚 9、23），**负信号端悬空**。信号连接方法见图 2-13、输出波形见图 2-14。

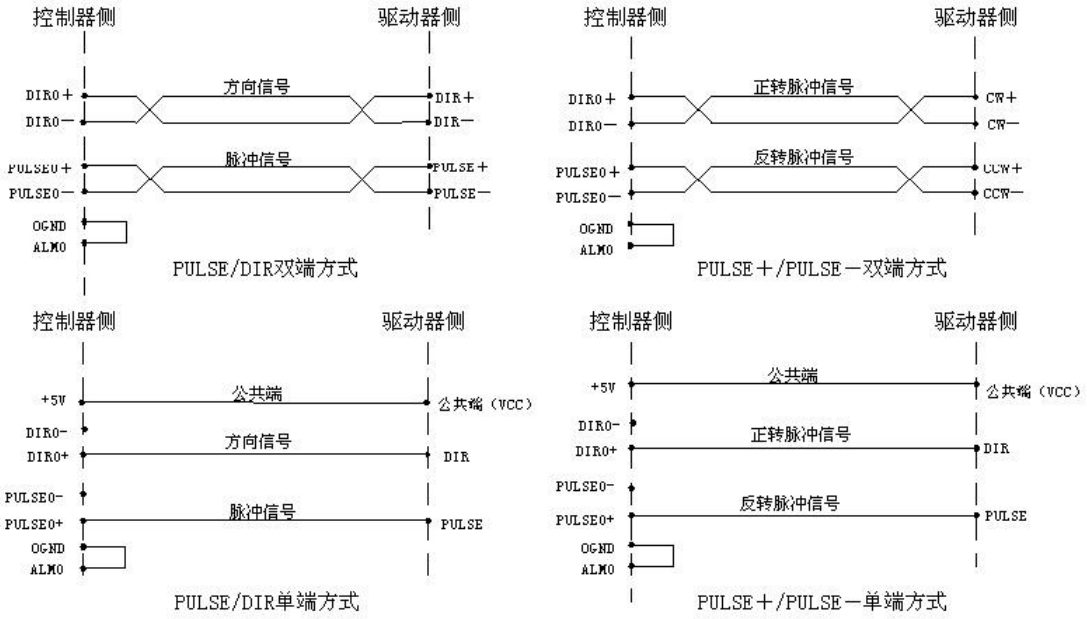


图 2-13 脉冲量控制输出信号连接图

输出方式	引脚	正转	反转
-PULSE + +PULSE	23-11 9-22		
PULSE + DIR	23-11 9-22		

图 2-14 脉冲量控制输出信号波形

2.3.5.5 通用数字量输入/输出连接方法

CPAC-Otobox 系列自动化控制器提供多路的通用数字量输入输出（具体路数视不同型号而定），通用输入输出的接口定义请参看第三章，连接方法见图 2-15。

当通用 I0 的输出接感性负载时，应考虑连接用于反电势泄放的二极管。

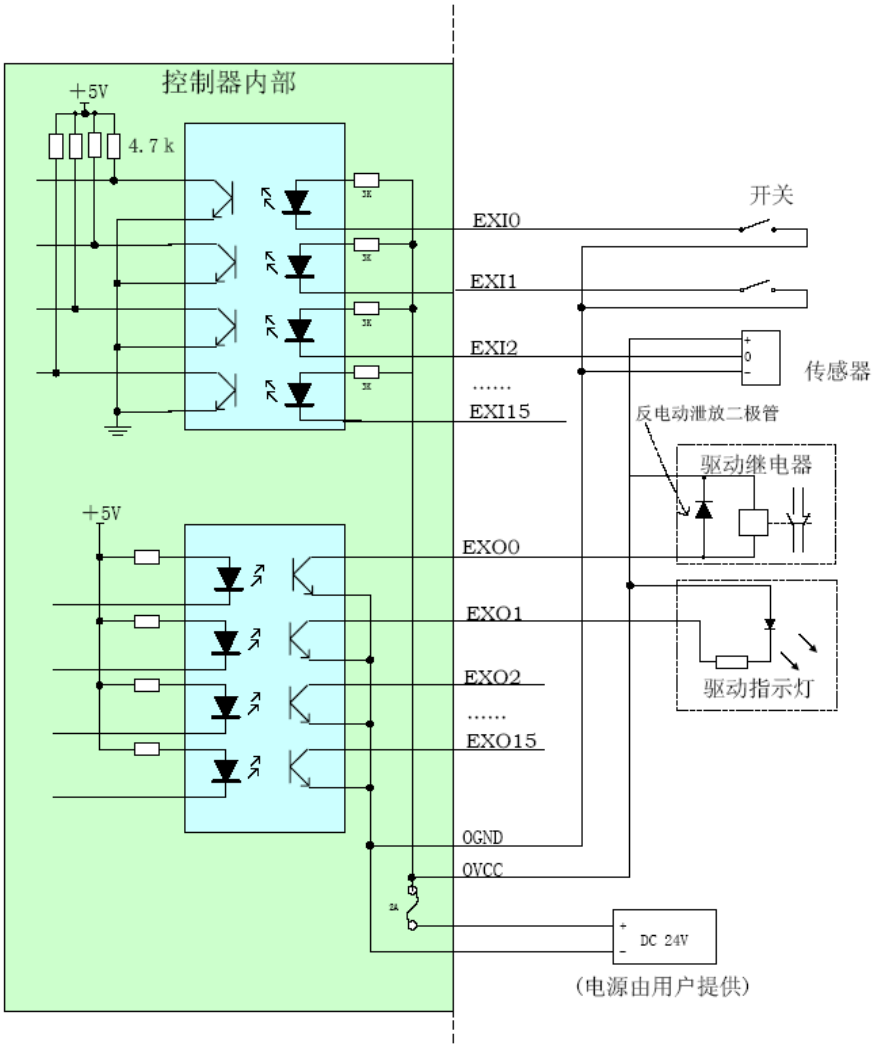


图 2-15 通用输入/输出信号连接图

2.3.5.6 模拟量输入连接方法

CPAC-Otobox 系列自动化控制器提供可选的模拟量输入功能（视具体型号而定）。模拟量输入信号的接口定义请参看第三章。连接方法见图 2-16。

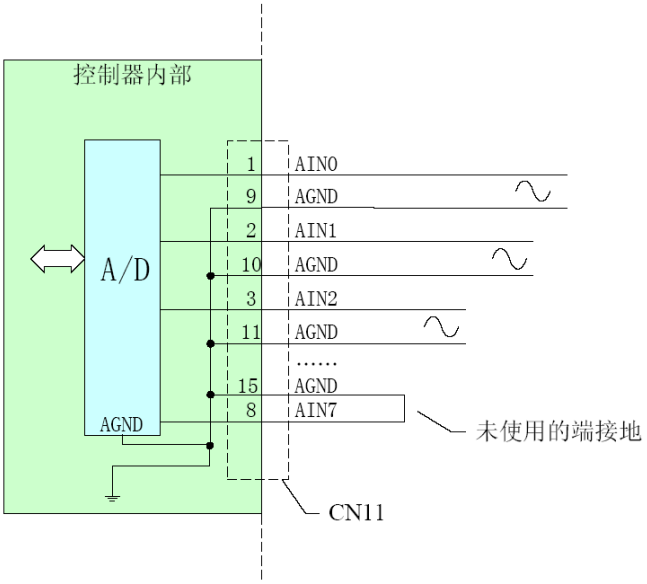


图 2-16 模拟输入信号连接图

2.3.6 CPAC-Otobox 控制器安装方式

为了更好地散热，形成通畅的空气对流，建议 CPAC-Otobox 自动化控制器的安装位置：垂直方向安装，如图 2-17。

图 2-17 CPAC-Otobox 自动化控制器垂直方向安装（散热效果好）

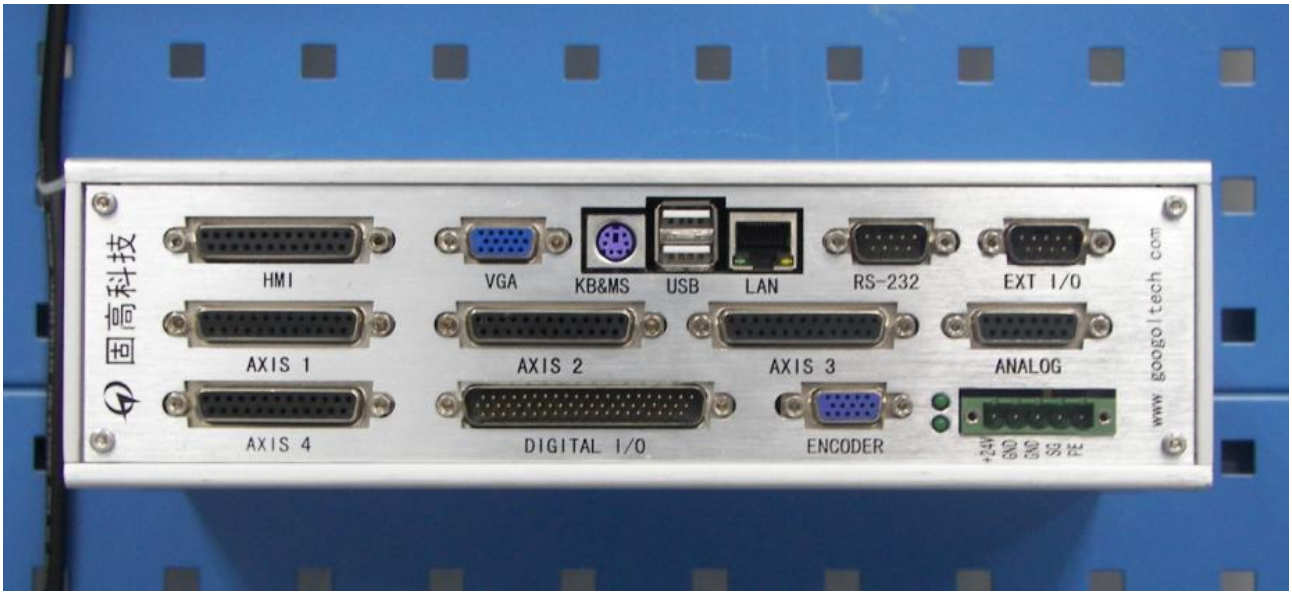


图 2-20 CPAC-Otobox 自动化控制器水平方向安装（散热效果差）

2.4 硬件测试

具体的使用方法请参见 OtoStudio 中的 PLC 浏览器。在 CPAC-Otostudio 中的 PLC 浏览器中键入 Help，会出现控制器所支持的调试指令，并有指令参数的详细介绍。

为安全起见，建议用户在系统调试过程中，**不要**将电机与任何机械装置连接。请检查电机确实没有负载。

第三章 CPAC-Otobox 系列自动化控制器接口定义

CPAC-Otobox 系列自动化控制器目前型号为：

CPAC-OtoBox-UCT2-8PV
CPAC-OtoBox-UCT2-8PG
CPAC-OtoBox-UCT2-4PV
CPAC-OtoBox-UCT2-4PV
CPAC-OtoBox-UCT2-8PV-Mini
CPAC-OtoBox-UCT2-8PG-Mini
CPAC-OtoBox-UCT2-4PV-Mini
CPAC-OtoBox-UCT2-4PV-Mini
以上这些型号的硬件接口定义与 GUC-X00-TXX-M01-L2 一致。详见 3.1

CPAC-OtoBox-UCTNI-4PV
CPAC-OtoBox-UCTNI-4PG
CPAC-OtoBox-UCTNI-000
以上这些型号的硬件接口定义与 GUC-X00-TN1-M01 一致。详见 3.2

L2 型运动控制器需要外接端子板才能与驱动器及外部输入/输出设备相连。因此本章节将具体介绍各型号的 CPAC-Otobox 自动化控制器的接口定义，以及所需外接的端子板的具体型号及接口定义。

3.1 GUC-X00-TXX-M01-L2 型运动控制器接口定义

GUC-X00-TXX-M01-L2 型运动控制器是固高科技推出的一款 4/8 轴通用型运动控制器。它需要外接固高科技生产的 GT2-800-ACC2 八轴端子板或 GT2-400-ACC2 四轴端子板才能与驱动器及外部输入/输出设备相连。GUC-X00-TXX-M01-L2 型运动控制器的接口列表参见表 3-1，接口定义见表 3-2。GT2-800-ACC2 八轴端子板的接口列表参见表 3-3，运动控制器及端子板的各接口定义见表 3-4、表 3-5、表 3-6、表 3-7、表 3-8、表 3-9、表 3-10。4 轴端子板接口定义请查看表 3-11 ~表 3-16。

表 3-1 GUC-X00-TXX--M01-L2 型运动控制器接口列表

接口标识	功能
HMI	HMI 接口
VGA	标准 VGA 接口
KB/MS	键盘、鼠标接口
USB	双层 USB 接口
LAN	以太网接口
RS232	通用串行口
EXT IO	高速 IO 扩展接口
AXIS（1~4）	控制器与端子板接口 1
AXIS（5~8）	控制器与端子板接口 2
POWER	电源接口

表 3-2 运动控制器电源接口（POWER）定义

接口标识	说明
24V	+24V 输入

0V	+24V 参考地
0V	+24V 参考地
SG	GUC 控制器内部数字地
PE	保护地（与大地相连）

表 3-3　GT2-800-ACC2 八轴端子板接口列表

接口标识	功能
CN1	驱动器接口
CN2	驱动器接口
CN3	驱动器接口
CN4	驱动器接口
CN5	驱动器接口
CN6	驱动器接口
CN7	驱动器接口
CN8	驱动器接口
CN9	通用/专用 IO 输入接口
CN11	通用/专用 IO 输入接口
CN10	通用 IO 输出接口
CN12	RS232 接口（内部使用）
CN13	辅助编码器接口
CN14	高速 IO 口（保留）
CN15	扩展 IO 口
CN16	端子板电源接口
CN17	控制器与端子板接口 1（BASIC PORT）
CN18	控制器与端子板接口 2（EXT PORT）

表 3-4　八轴端子板驱动器接口（CN1~CN8）定义

引脚	信号	说明	引脚	信号	说明
1	OGND	+24V 电源地	14	OVCC	+24V 电源输出
2	ALM	驱动报警	15	RESET	驱动报警复位
3	ENABLE	驱动允许	16	保留	保留
4	A-	编码器输入	17	A+	编码器输入
5	B-	编码器输入	18	B+	编码器输入
6	C-	编码器输入	19	C+	编码器输入
7	+5V	+5V 电源输出	20	GND	+5V 电源地
8	DAC	模拟输出	21	GND	+5V 电源地
9	DIR+	步进方向输出	22	DIR-	步进方向输出
10	GND	+5V 电源地	23	PULSE+	步进脉冲输出
11	PULSE-	步进脉冲输出	24	GND	+5V 电源地
12	保留	保留	25	保留	保留
13	GND	+5V 电源地			

表 3-5　八轴端子板通用/专用 IO 输入接口（CN9）定义

引脚	信号	说明	引脚	信号	说明
1	HOME0	1 轴原点信号输入	11	LIMIT3+	4 轴正向限位输入

引脚	信号	说明	引脚	信号	说明
2	HOME1	2 轴原点信号输入	12	LIMIT3-	4 轴负向限位输入
3	HOME2	3 轴原点信号输入	13	EXI0	通用输入
4	HOME3	4 轴原点信号输入	14	EXI1	通用输入
5	LIMIT0+	1 轴正向限位输入	15	EXI2	通用输入
6	LIMIT0-	1 轴负向限位输入	16	EXI3	通用输入
7	LIMIT1+	2 轴正向限位输入	17	EXI4	通用输入
8	LIMIT1-	2 轴负向限位输入	18	EXI5	通用输入
9	LIMIT2+	3 轴正向限位输入	19	EXI6	通用输入
10	LIMIT2-	3 轴负向限位输入	20	EXI7	通用输入

表 3-6　八轴端子板通用/专用 IO 输入接口（CN11）定义

引脚	信号	说明	引脚	信号	说明
1	HOME4	5 轴原点信号输入	11	LIMIT7+	8 轴正向限位输入
2	HOME5	6 轴原点信号输入	12	LIMIT7-	8 轴负向限位输入
3	HOME6	7 轴原点信号输入	13	EXI8	通用输入
4	HOME7	8 轴原点信号输入	14	EXI9	通用输入
5	LIMIT4+	5 轴正向限位输入	15	EXI10	通用输入
6	LIMIT4-	5 轴负向限位输入	16	EXI11	通用输入
7	LIMIT5+	6 轴正向限位输入	17	EXI12	通用输入
8	LIMIT5-	6 轴负向限位输入	18	EXI13	通用输入
9	LIMIT6+	7 轴正向限位输入	19	EXI14	通用输入
10	LIMIT6-	7 轴负向限位输入	20	EXI15	通用输入

表 3-7　八轴端子板通用 IO 输出接口（CN10）定义

引脚	信号	说明	引脚	信号	说明
1	EXO0	通用输出	11	EXO10	通用输出
2	EXO1	通用输出	12	EXO11	通用输出
3	EXO2	通用输出	13	EXO12	通用输出
4	EXO3	通用输出	14	EXO13	通用输出
5	EXO4	通用输出	15	EXO14	通用输出
6	EXO5	通用输出	16	EXO15	通用输出
7	EXO6	通用输出	17	OVCC	+24V 电源输出
8	EXO7	通用输出	18	OVCC	+24V 电源输出
9	EXO8	通用输出	19	OGND	+24V 电源地
10	EXO9	通用输出	20	OGND	+24V 电源地

表 3-8　八轴端子板辅助编码器接口（CN13）定义

引脚	信号	说明	引脚	信号	说明
1	A+	编码器输入	6	A-	编码器输入
2	B+	编码器输入	7	B-	编码器输入
3	C+	编码器输入	8	C-	编码器输入
4			9	GND	数字地
5	+5V	电源输出			

表 3-9　八轴端子板辅助编码器接口（CN15）定义

引脚	信号	说明	引脚	信号	说明
1	NC	无连接	6	NC	无连接
2	TX+	串行通讯下行数据正相	7	TX-	串行通讯下行数据负相
3	RX+	串行通讯上行数据正相	8	RX-	串行通讯上行数据负相
4	NC	无连接	9	NC	无连接
5	NC	无连接			

表 3-10　八轴端子板电源接口（CN16）定义

接口标识	说明
+24V	+24V 输入
OGND	+24V 参考地
FG	保护地（与大地相连）

表 3-11　GT2-400-ACC2　　四轴端子板接口定义

接口端子	功能
CN1~ CN4	控制轴接口
CN9-1	限位、原点及通用输入 0~7 接口
CN9-2	通用输入 8~15 以及通用输出 0~15 接口
CN12	辅助编码器接口
CN13	辅助编码器接口
CN14	内部接口
CN15	扩展模块接口
CN16	端子板电源接口
CN17	运动控制器连接接口
CN19	模拟量输入接口

表 3-12　　四轴端子板 CN1~CN4 接口定义

引脚	信号	说明	引脚	信号	说明
1	OGND	外部电源地	14	OVCC	+24V 输出
2	ALM	驱动报警	15	RESET	驱动报警清除
3	ENABLE	驱动允许	16	ARRIVE	电机到位
4	A-	编码器输入	17	A+	编码器输入
5	B-	编码器输入	18	B+	编码器输入
6	C-	编码器输入	19	C+	编码器输入
7	+5V	电源输出	20	GND	数字地
8	DAC	模拟输出	21	GND	数字地
9	DIR+	步进方向输出	22	DIR-	步进方向输出
10	GND	数字地	23	PULSE+	步进脉冲输出
11	PULSE-	步进脉冲输出	24	GND	数字地
12	AIN	模拟输入	25	保留	保留
13	GND	数字地			

表 3-13　四轴 端子板 CN9-1 的接口定义

引脚	信号	说明	引脚	信号	说明
1	OVCC	+24V 电源	13	LIMIT2+	3 轴正向限位
2	OVCC	+24V 电源	14	LIMIT2-	3 轴负向限位
3	OGND	外部电源地	15	LIMIT3+	4 轴正向限位
4	OGND	外部电源地	16	LIMIT3-	4 轴负向限位
5	HOME0	1 轴原点输入	17	EXI0	通用输入
6	HOME1	2 轴原点输入	18	EXI1	通用输入
7	HOME2	3 轴原点输入	19	EXI2	通用输入
8	HOME3	4 轴原点输入	20	EXI3	通用输入
9	LIMIT0+	1 轴正向限位	21	EXI4	通用输入
10	LIMIT0-	1 轴负向限位	22	EXI5	通用输入
11	LIMIT1+	2 轴正向限位	23	EXI6	通用输入
12	LIMIT1-	2 轴负向限位	24	EXI7	通用输入

表 3-14　四轴端子板 CN12 和 CN13 接口定义

引脚	信号	说明	引脚	信号	说明
1	A+	编码器输入	6	A-	编码器输入
2	B+	编码器输入	7	B-	编码器输入
3	C+	编码器输入	8	C-	编码器输入
4			9	GND	数字地
5	+5V	电源输出			

表 3-15　四轴端子板 CN9-2 的接口定义

引脚	信号	说明	引脚	信号	说明
1	EXI8	通用输入	13	EXO4	通用输出
2	EXI9	通用输入	14	EXO5	通用输出
3	EXI10	通用输入	15	EXO6	通用输出
4	EXI11	通用输入	16	EXO7	通用输出
5	EXI12	通用输入	17	EXO8	通用输出
6	EXI13	通用输入	18	EXO9	通用输出
7	EXI14	通用输入	19	EXO10	通用输出
8	EXI15	通用输入	20	EXO11	通用输出
9	EXO0	通用输出	21	EXO12	通用输出
10	EXO1	通用输出	22	EXO13	通用输出
11	EXO2	通用输出	23	EXO14	通用输出
12	EXO3	通用输出	24	EXO15	通用输出

表 3-16　四轴端子板 CN19 的接口定义

引脚	信号	说明
1	模拟输入通道 1	模拟输入
2	模拟输入通道 2	模拟输入
3	模拟输入通道 3	模拟输入
4	模拟输入通道 4	模拟输入
5	模拟输入通道 5(AXIS4)	模拟输入

6	模拟输入通道 6(AXIS3)	模拟输入
7	模拟输入通道 7(AXIS2)	模拟输入
8	模拟输入通道 8(AXIS1)	模拟输入
9	GND	模拟地
10	GND	模拟地
11	GND	模拟地
12	GND	模拟地
13	GND	模拟地
14	GND	模拟地
15	GND	模拟地

模拟量输出通过 CN19 的 1~8 引脚输入，参考地为数字地。其中，CN1~ CN 4 上的 12 引脚分别与 CN19 上的 5~8 引脚一一对应，分别为同一输入通道。内部连接如图 3-1。

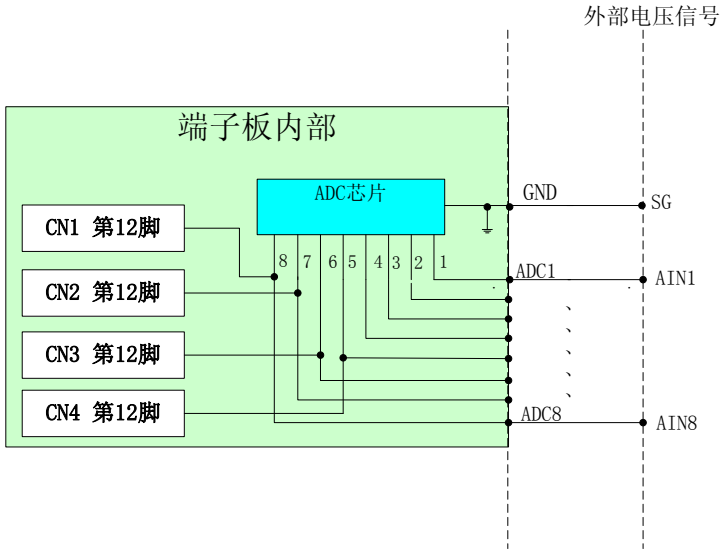


图 3-1　四轴端子板 AD 内部连接

3.2　GUC-X00-TN1-M01-L2 型运动控制器接口定义

GUC-400-TN1-(OX)-M0X-FXG 型运动控制器是固高科技推出的一款网络型运动控制器。其接口列表参见表 3-1。各接口定义见表 3-2、表 3-3、表 3-4。

表 3-1　GUC-400-TN1-(OX)-M0X-FXG 型运动控制器接口列表

接口标识	功能
HMI	HMI 接口
VGA	标准 VGA 接口
KB&MS	键盘、鼠标接口
USB	双层 USB 接口
LAN	以太网接口
RS-232	通用串行口
EXT I/O	高速 IO 扩展接口
AXIS (1~4)	端子板驱动器接口
MII A/B	MII 驱动器、终端电阻接口
CAN	CAN 接口

485	485 接口
POWER	电源接口

表 3-2　电源接口（POWER）定义

接口标识	说明
24V	+24V 输入
0V	+24V 参考地
0V	+24V 参考地
SG	GUC 控制器内部数字地
PE	保护地（与大地相连）

表 3-3　485 接口定义

接口标识	说明
LR1	终端电阻
HR1	终端电阻
COM1	485 地
485+	485 数据线
485-	485 数据线

表 3-4　CAN 接口定义

接口标识	说明
LR2	终端电阻
HR2	终端电阻
VsUp	5V 电源输出
CAN_H	CAN 数据线
SHLD	CAN 屏蔽地
CAN_L	CAN 数据线
COM2	CAN 地

表 3-11　GT2-400-ACC2　　四轴端子板接口定义

接口端子	功能
CN1~ CN4	控制轴接口
CN9-1	限位、原点及通用输入 0~7 接口
CN9-2	通用输入 8~15 以及通用输出 0~15 接口
CN12	辅助编码器接口
CN13	辅助编码器接口
CN14	内部接口
CN15	扩展模块接口（不可用）
CN16	端子板电源接口
CN17	运动控制器连接接口
CN19	模拟量输入接口

表 3-12　　四轴端子板 CN1~CN4 接口定义

引脚	信号	说明	引脚	信号	说明
1	OGND	外部电源地	14	OVCC	+24V 输出
2	ALM	驱动报警	15	RESET	驱动报警清除
3	ENABLE	驱动允许	16	ARRIVE	电机到位
4	A-	编码器输入	17	A+	编码器输入
5	B-	编码器输入	18	B+	编码器输入
6	C-	编码器输入	19	C+	编码器输入
7	+5V	电源输出	20	GND	数字地
8	DAC	模拟输出	21	GND	数字地
9	DIR+	步进方向输出	22	DIR-	步进方向输出
10	GND	数字地	23	PULSE+	步进脉冲输出
11	PULSE-	步进脉冲输出	24	GND	数字地
12	AIN	模拟输入	25	保留	保留
13	GND	数字地			

表 3-13　四轴 端子板 CN9-1 的接口定义

引脚	信号	说明	引脚	信号	说明
1	OVCC	+24V 电源	13	LIMIT2+	3 轴正向限位
2	OVCC	+24V 电源	14	LIMIT2-	3 轴负向限位
3	OGND	外部电源地	15	LIMIT3+	4 轴正向限位
4	OGND	外部电源地	16	LIMIT3-	4 轴负向限位
5	HOME0	1 轴原点输入	17	EXI0	通用输入
6	HOME1	2 轴原点输入	18	EXI1	通用输入
7	HOME2	3 轴原点输入	19	EXI2	通用输入
8	HOME3	4 轴原点输入	20	EXI3	通用输入
9	LIMIT0+	1 轴正向限位	21	EXI4	通用输入
10	LIMIT0-	1 轴负向限位	22	EXI5	通用输入
11	LIMIT1+	2 轴正向限位	23	EXI6	通用输入
12	LIMIT1-	2 轴负向限位	24	EXI7	通用输入

表 3-14　四轴端子板 CN12 和 CN13 接口定义

引脚	信号	说明	引脚	信号	说明
1	A+	编码器输入	6	A-	编码器输入
2	B+	编码器输入	7	B-	编码器输入
3	C+	编码器输入	8	C-	编码器输入
4			9	GND	数字地
5	+5V	电源输出			

表 3-15　四轴端子板 CN9-2 的接口定义

引脚	信号	说明	引脚	信号	说明
1	EXI8	通用输入	13	EXO4	通用输出
2	EXI9	通用输入	14	EXO5	通用输出
3	EXI10	通用输入	15	EXO6	通用输出
4	EXI11	通用输入	16	EXO7	通用输出
5	EXI12	通用输入	17	EXO8	通用输出

引脚	信号	说明	引脚	信号	说明
6	EXI13	通用输入	18	EXO9	通用输出
7	EXI14	通用输入	19	EXO10	通用输出
8	EXI15	通用输入	20	EXO11	通用输出
9	EXO0	通用输出	21	EXO12	通用输出
10	EXO1	通用输出	22	EXO13	通用输出
11	EXO2	通用输出	23	EXO14	通用输出
12	EXO3	通用输出	24	EXO15	通用输出

表 3-16　四轴端子板 CN19 的接口定义

引脚	信号	说明
1	模拟输入通道 1	模拟输入
2	模拟输入通道 2	模拟输入
3	模拟输入通道 3	模拟输入
4	模拟输入通道 4	模拟输入
5	模拟输入通道 5(AXIS4)	模拟输入
6	模拟输入通道 6(AXIS3)	模拟输入
7	模拟输入通道 7(AXIS2)	模拟输入
8	模拟输入通道 8(AXIS1)	模拟输入
9	GND	模拟地
10	GND	模拟地
11	GND	模拟地
12	GND	模拟地
13	GND	模拟地
14	GND	模拟地
15	GND	模拟地

模拟量输出通过 CN19 的 1~8 引脚输入，参考地为数字地。其中，CN1~ CN 4 上的 12 引脚分别与 CN19 上的 5~8 引脚一一对应，分别为同一输入通道。内部连接如图 3-1。

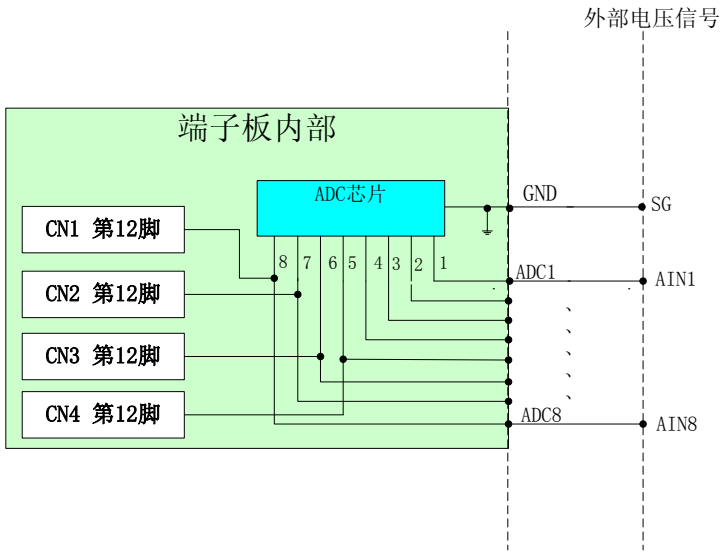


图 3-1　四轴端子板 AD 内部连接

附录 A　技术参数

CPAC-Otobox 系列自动化控制器

工业计算机

CPU　　X86 架构处理器，主频100MHZ /600MHZ /1G/

通讯方式

PCI 总线通讯

控制周期

TXX ： 200us 可调

模拟量输出

通道数　3/4/8　（每轴 1 路）
范围　　-10V ~ +10V
分辨率　16bit

脉冲输出

通道数　　3/4/8　（每轴 1 路）
输出频率　　TXX　：最大频率 1MHz

输出方式　　RS-422 线驱动器，+/-20mA
占空比　　50%
非线性　　<1%

编码器输入

轴编码器　　2、3、4、8 路（A，B，C，　其中 A，B 正交，每轴 1 路）
辅助编码器　TXX：　　1 路（A，B 其中 A，B 正交）
编码器信号　RS-422 线接收器，兼容单端输入
输入频率　　TSX、EPX：最大频率　　8MHz
　　ESX：最大频率　　4MHz

专用数字量输入输出

专用输入（每轴）：　LIMIT （POS）　（正限位）
　　LIMIT （NEG）　（负限位）
　　HOME　　（原点）
　　ALARM　　（驱动器报警）

专用输出（每轴）：　ENABLE　（伺服允许）
　　RESET　（驱动器复位）

通用数字量输入输出

通用输入：　16 路
　　光电隔离
通用输出：　16 路
　　光电隔离
　　集电极开路输出
　　驱动能力 200mA

电源要求

+24V　　Icc = 3A

外形尺寸

296mm x 160mm x 77mm

工作温度

0-60℃　（32℉-140℉）

相对湿度

5%-90% 非凝结

端子板

光耦隔离 I/O

光耦的输入规格：
　　隔离电压　5000V RMS
　　输入电压　+12V～+24VDC
　　输入电流　3.7mA～7.6mA
　　传输延迟　H→L　5us
　　L→H　3us

光耦输出规格为：
　　隔离电压　5000V RMS
　　集电极开路输出,无上拉电阻
　　Vceo　≤ 50V
　　Veco　≤ 5V
　　Ic　≤ 30mA（标准型）

Ic　　≤ 200mA (-R 型)
平均输出延迟　8us

A/D (可选项)

经同步串行口与运动控制器相连
输入路数　8 路　(单端 双极性)
输入范围　-10V～+10V
分辩率　　12bit
精度　　+/-1bit
最高采样速率　50KHz (单路)

外部电源

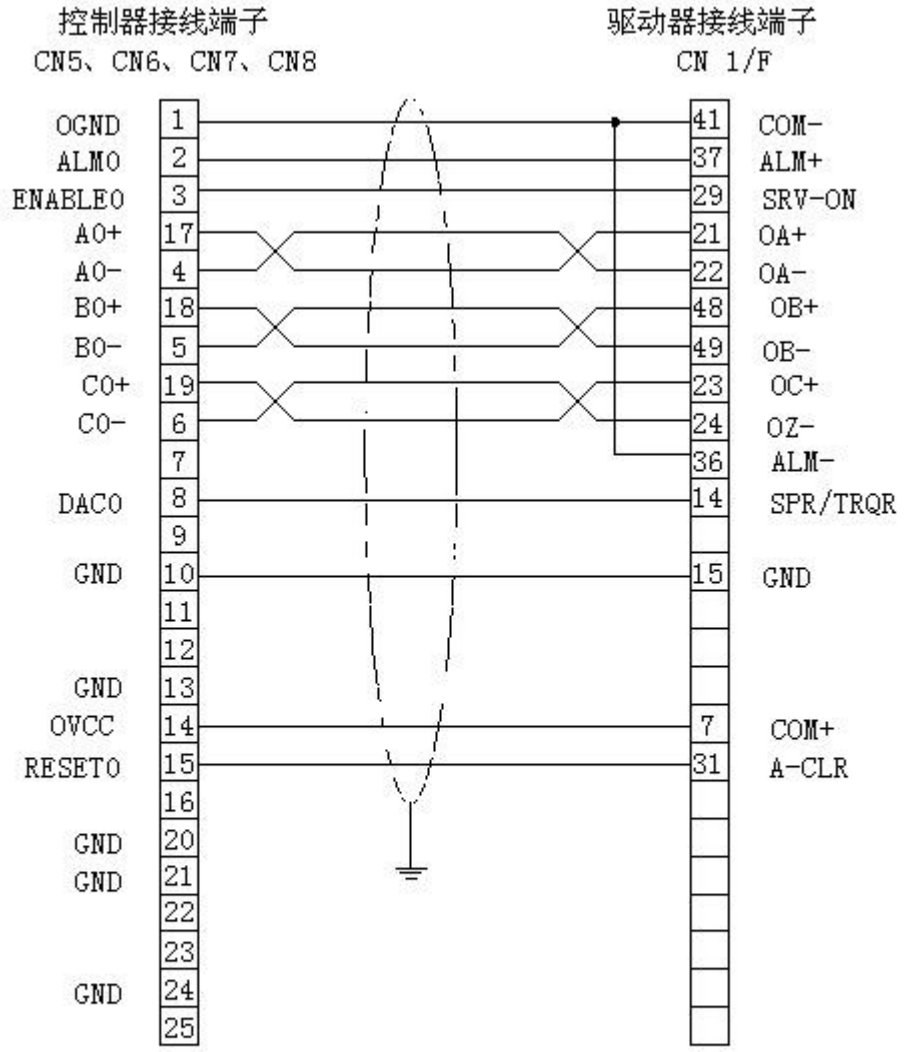
+24V DC　Icc=1.8A

外形尺寸

220mm x 132mm

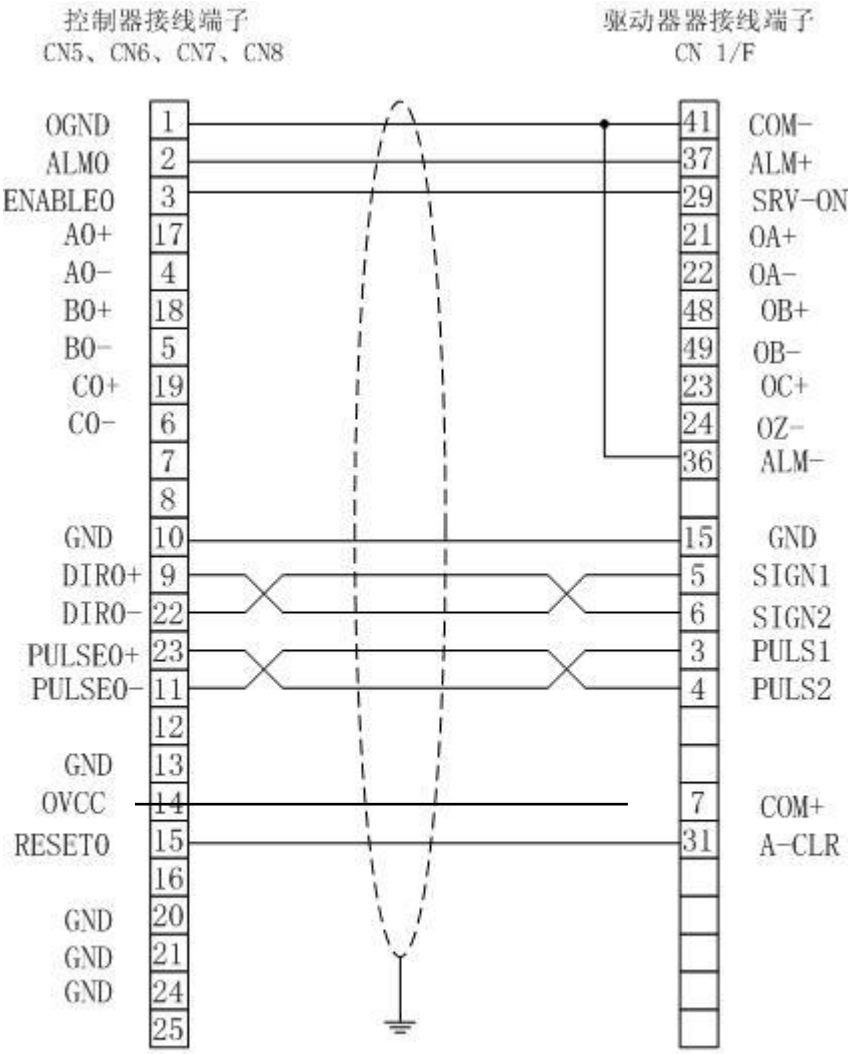
附录 B　典型接线

B.1 控制器与 Panasonic MSDA 系列驱动器速度控制方式接线



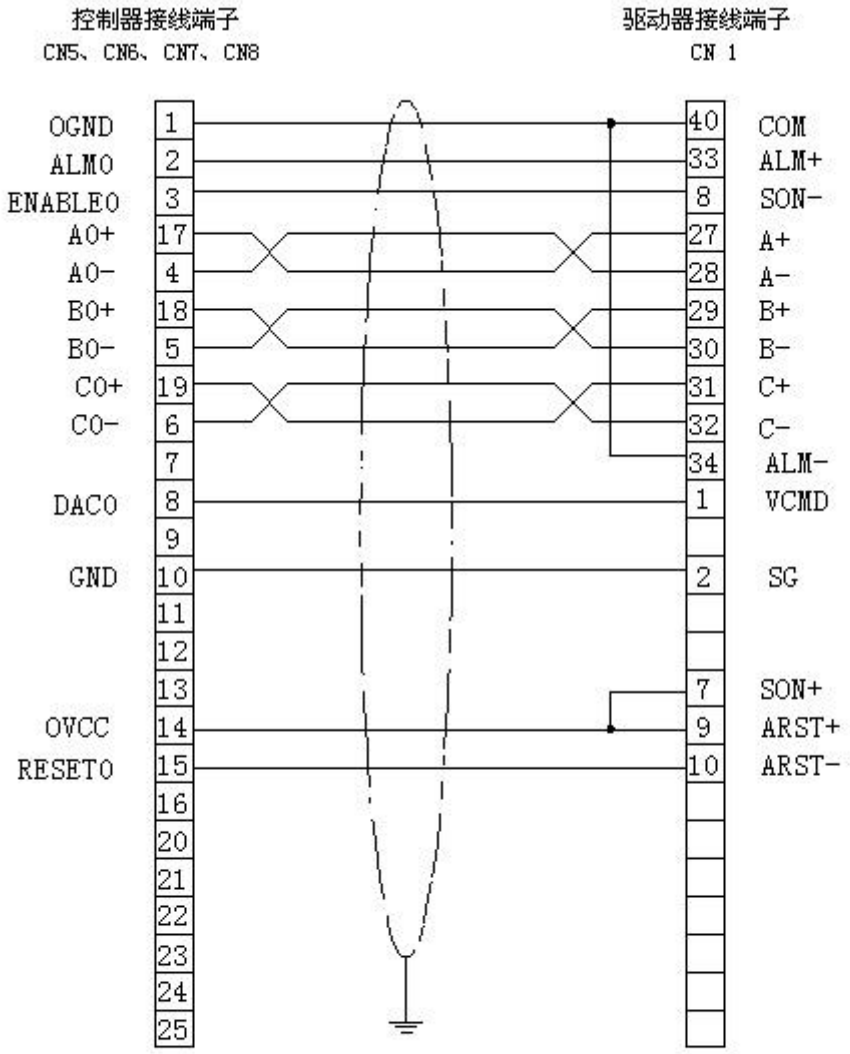
Panasonic MSDA系列驱动器速度控制方式接线

B.2 控制器与 Panasonic MSDA 系列驱动器位置控制方式接线



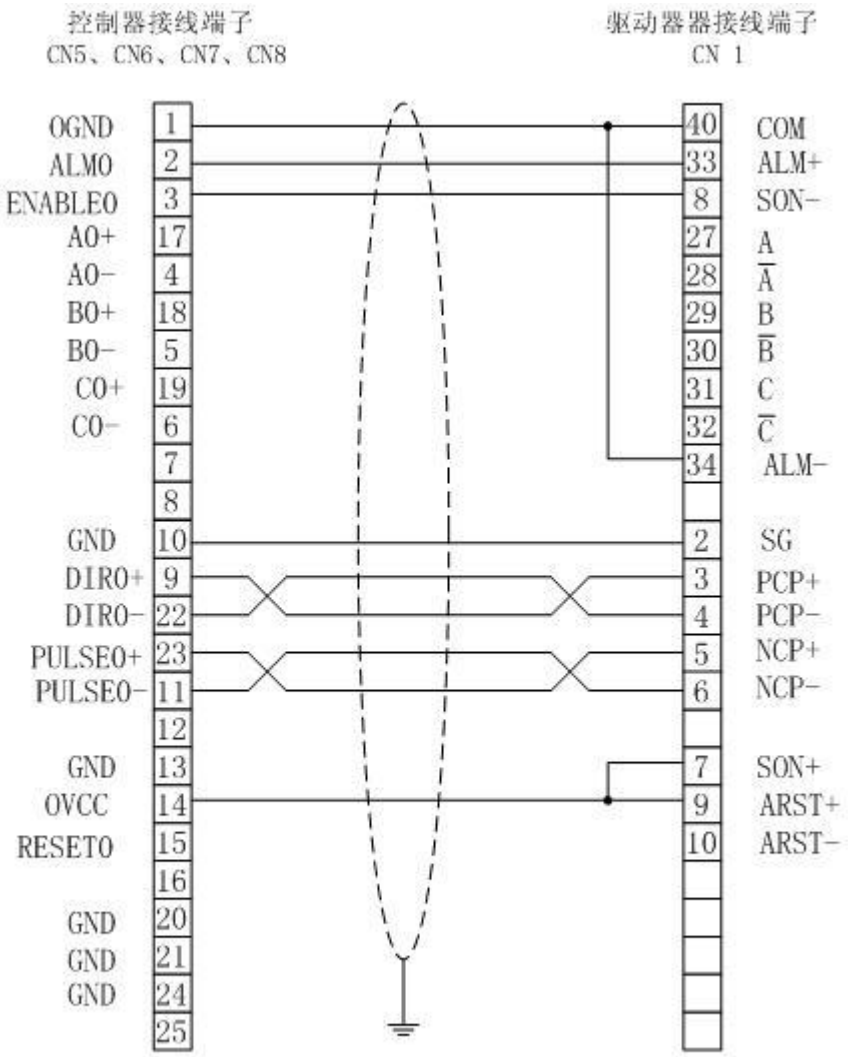
Panasonic MSDA 系列驱动器位置控制方式接线图

B.3 控制器与 SANYO DENKI PV1 系列驱动器速度控制方式接线



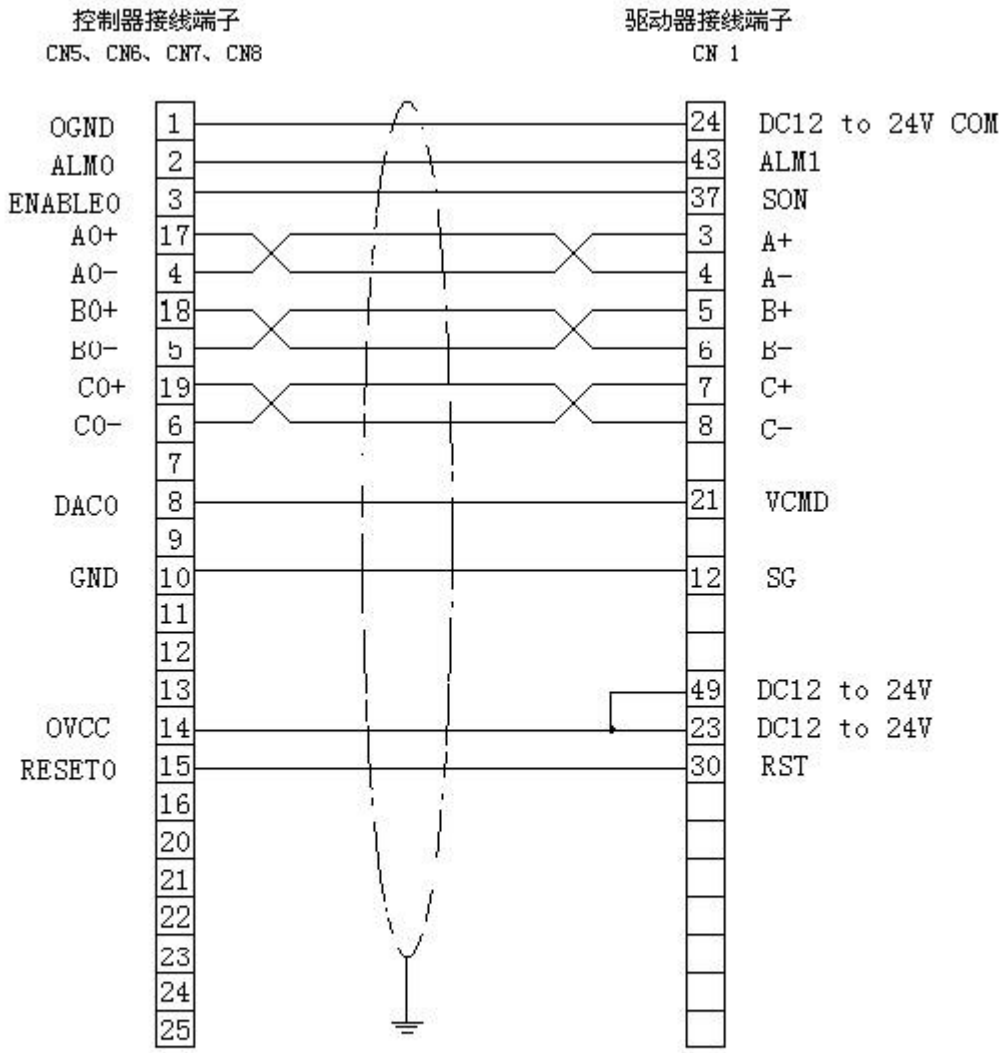
SANYO DENKI PV1 系列驱动器速度控制方式接线

B.4 控制器与 SANYO DENKI PV1 系列驱动器位置控制方式接线



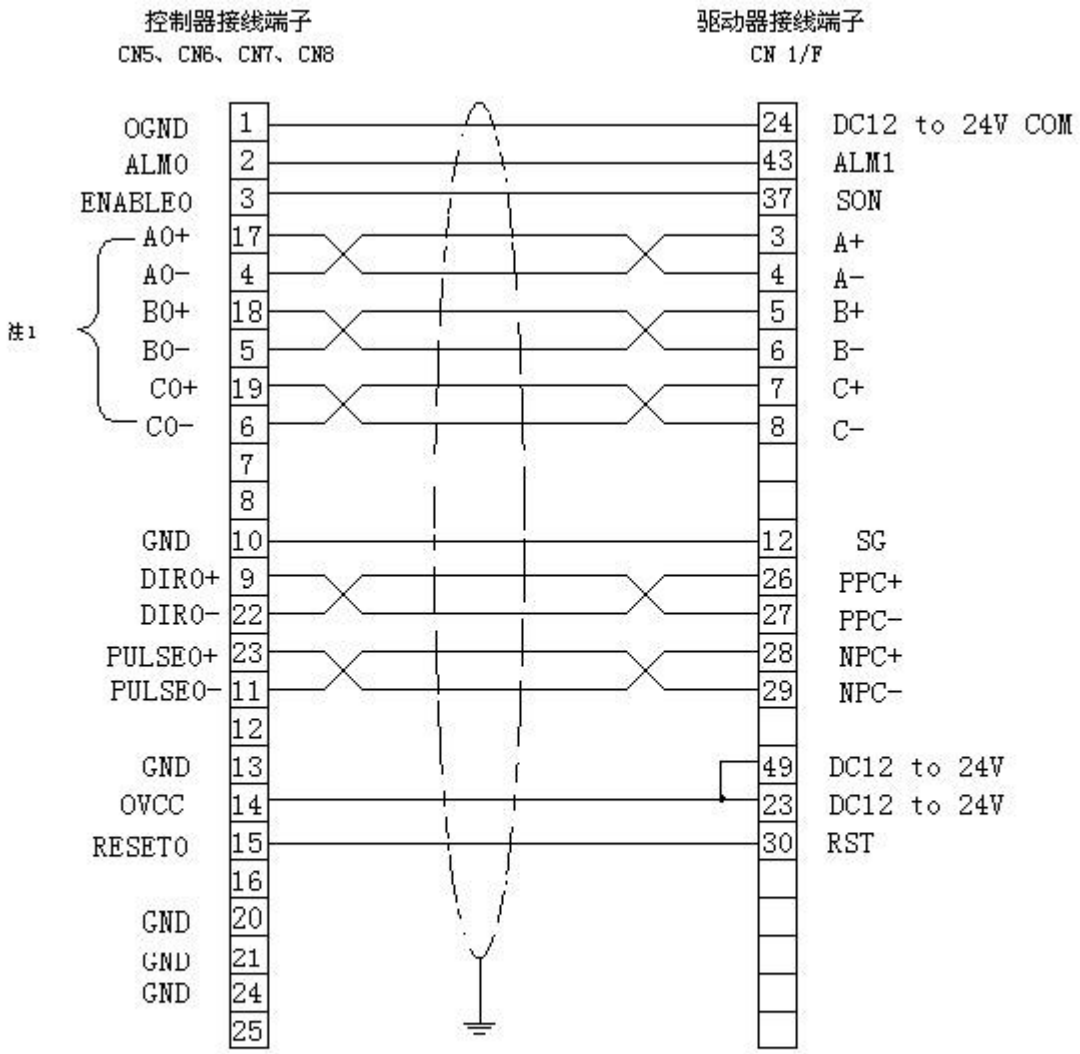
SANYO DENKI PV1 系列驱动器位置控制方式接线图

B.5 控制器与 SANYO DENKI PY0/PY2 系列驱动器速度控制方式接线



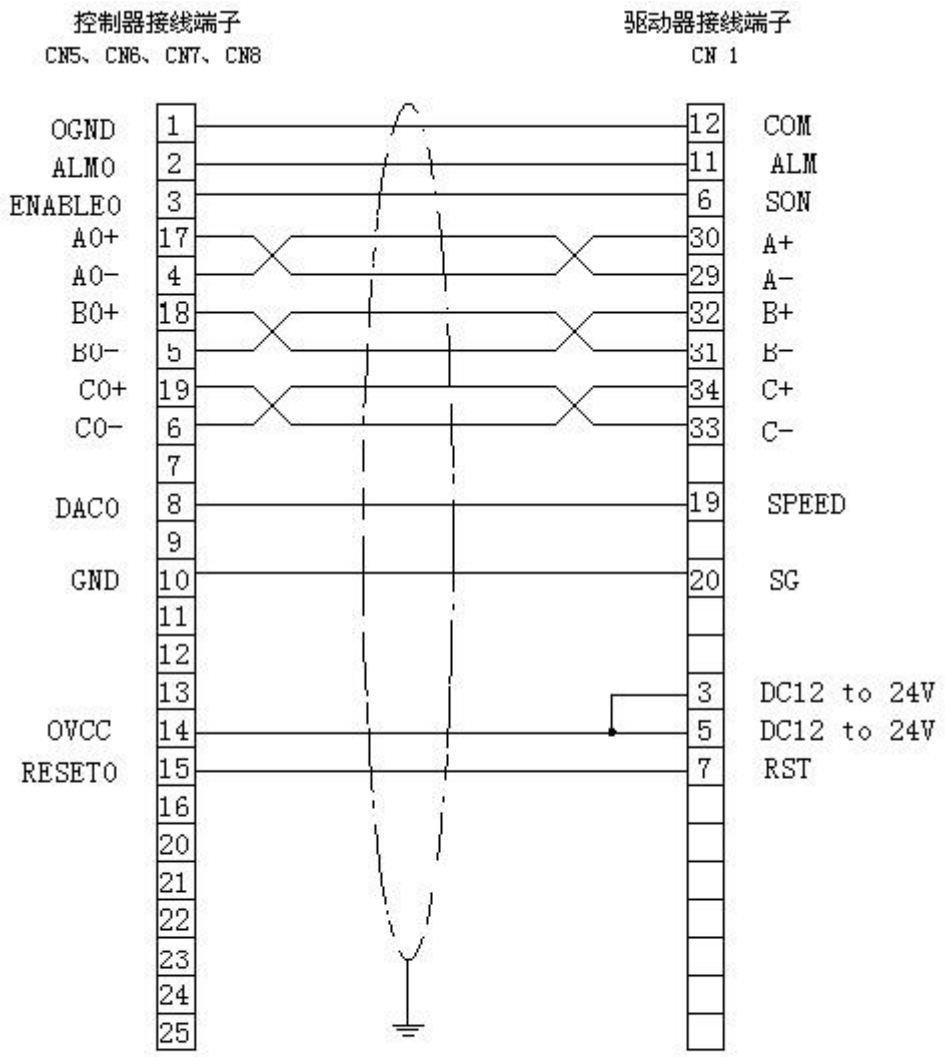
SANYO DENKI PY0/PY2系列驱动器速度控制方式接线

B.6 控制器与 SANYO DENKI PY0/PY2 系列驱动器位置控制方式接线



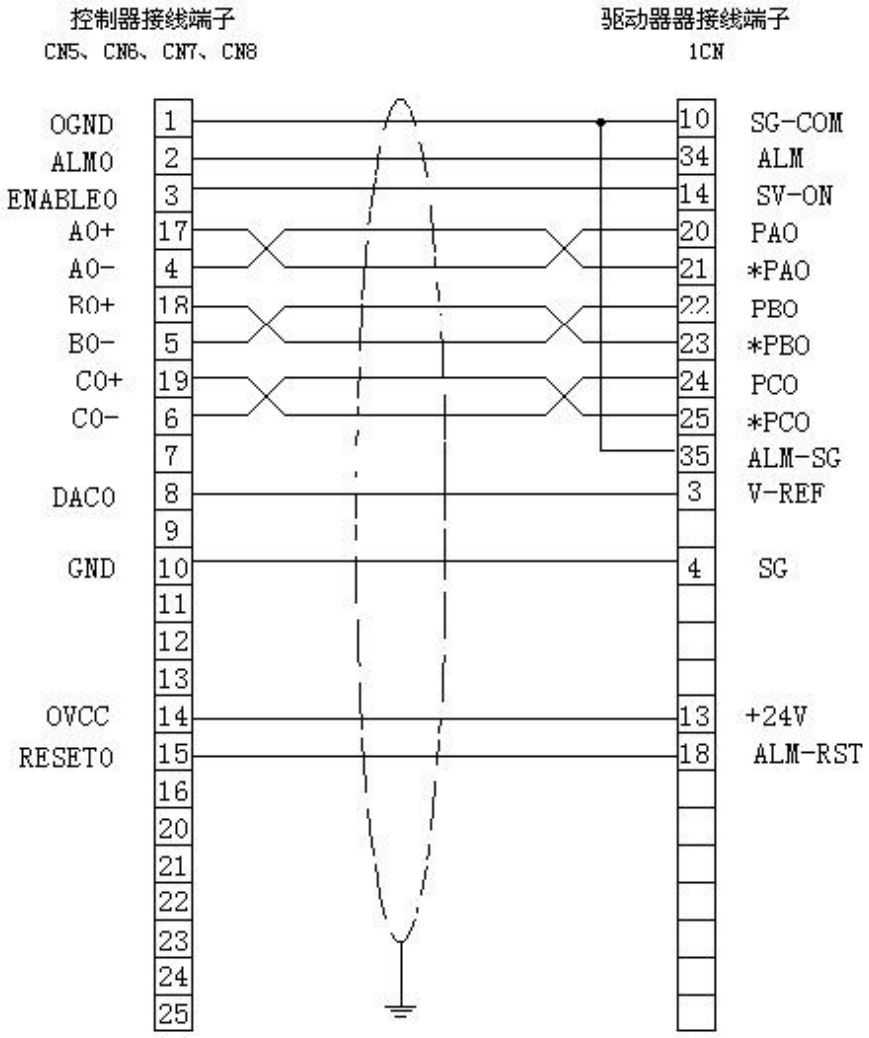
SANYODENKI PY0/PY2系列驱动器位置控制方式接线

B.7 控制器与 SANYO DENKI PU 系列驱动器速度控制方式接线



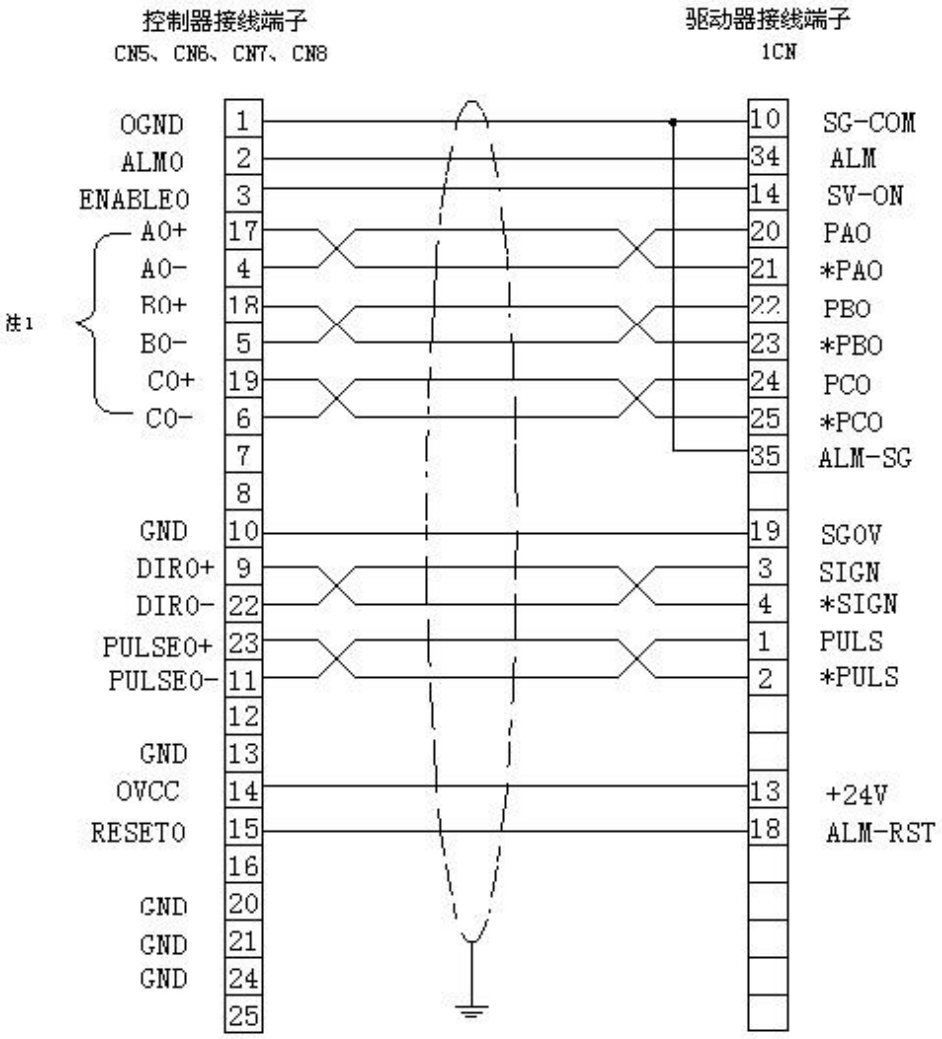
SANYODENKI PU系列驱动器速度控制方式接线

B.8 控制器与 YASKAWA SERVOPACK 系列 驱动器速度/力矩控制方式接线



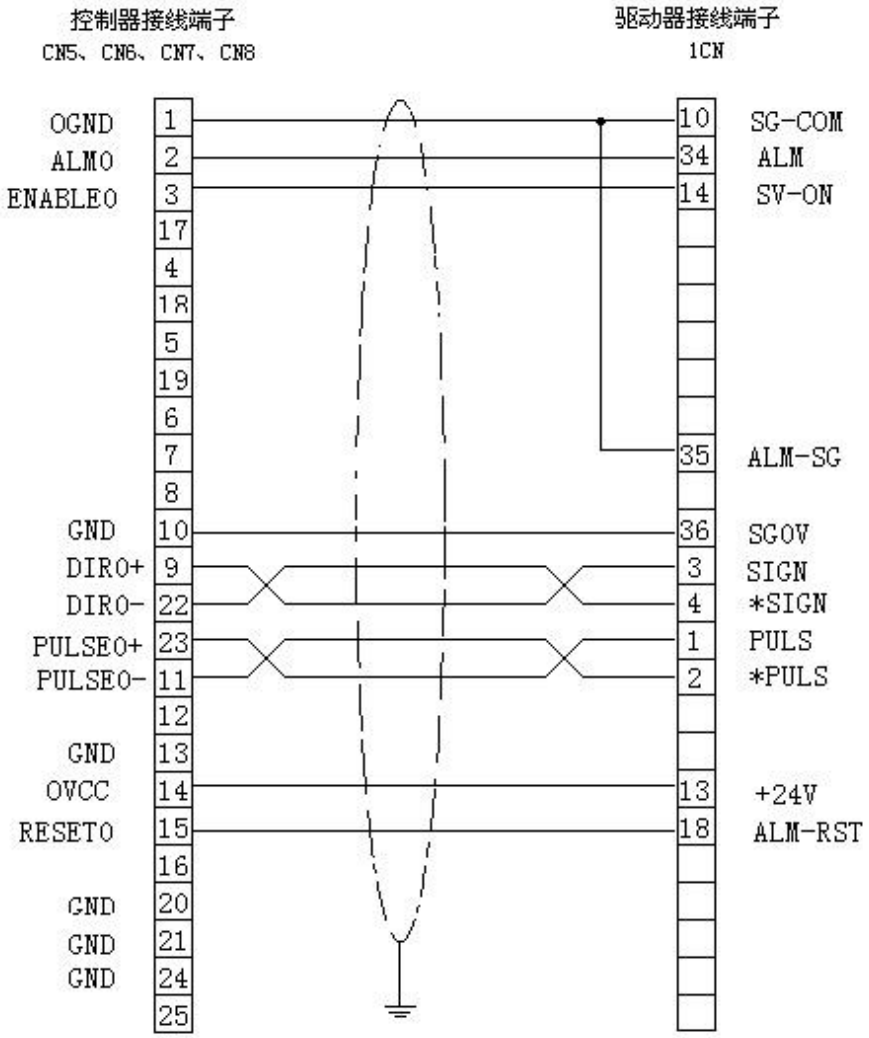
YASKAWA SERVOPACK SGDA-xxxS系列驱动器速度/力矩控制方式接线

B.9 控制器与 YASKAWA SERVOPACK 系列驱动器位置控制方式接线



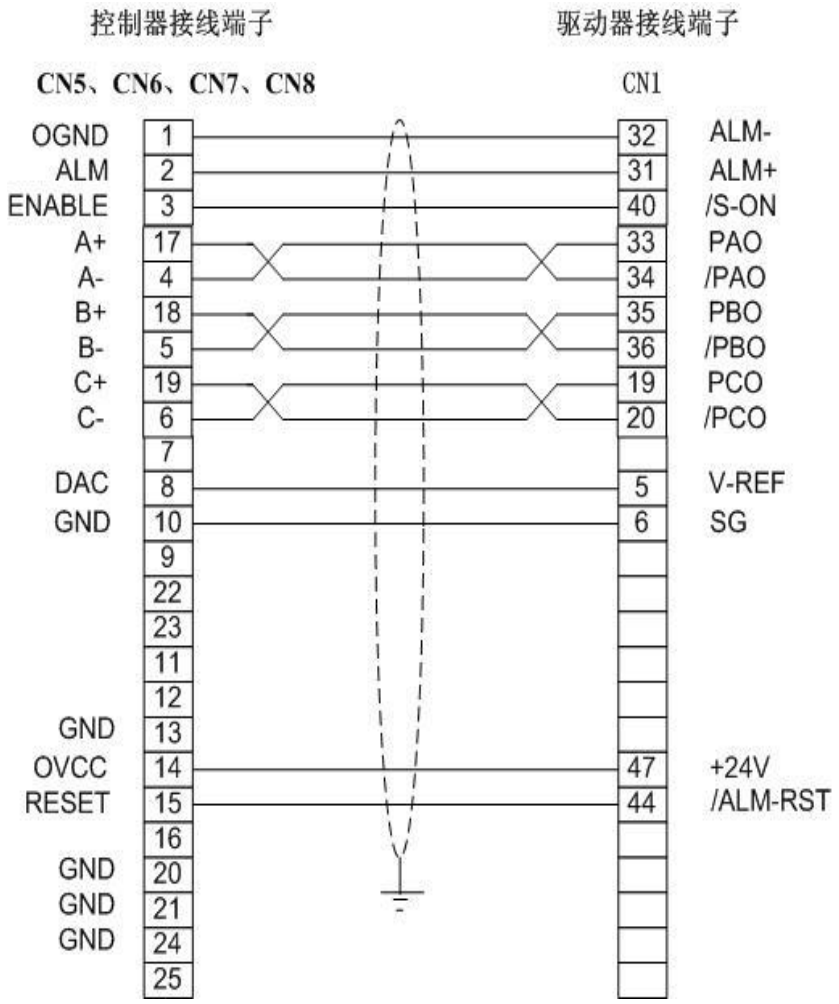
YASKAWA SERVOPACK SGDA-xxxP系列驱动器位置控制方式接线

B.10 控制器与 YASKAWASGDE 系列驱动器位置控制方式接线



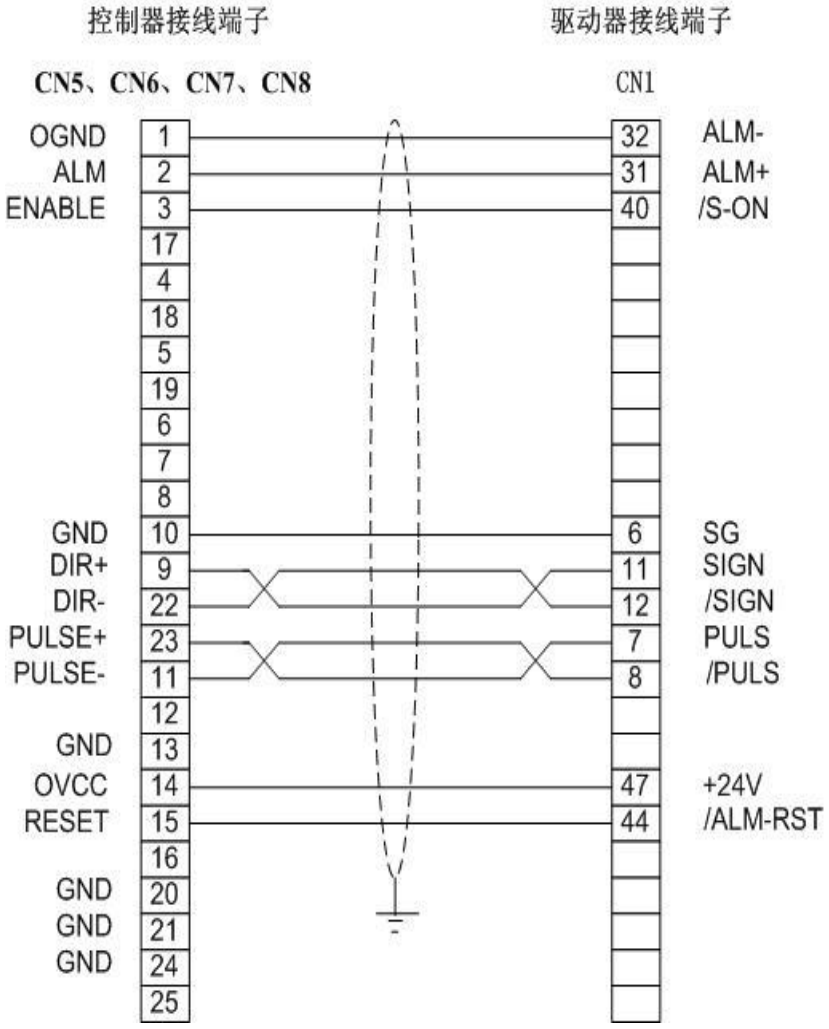
YASKAWA SERVOPACK SGDE-xxxP系列驱动器位置控制方式接线

B.11 控制器与 YASKAWA SGDM 型驱动器速度方式接线



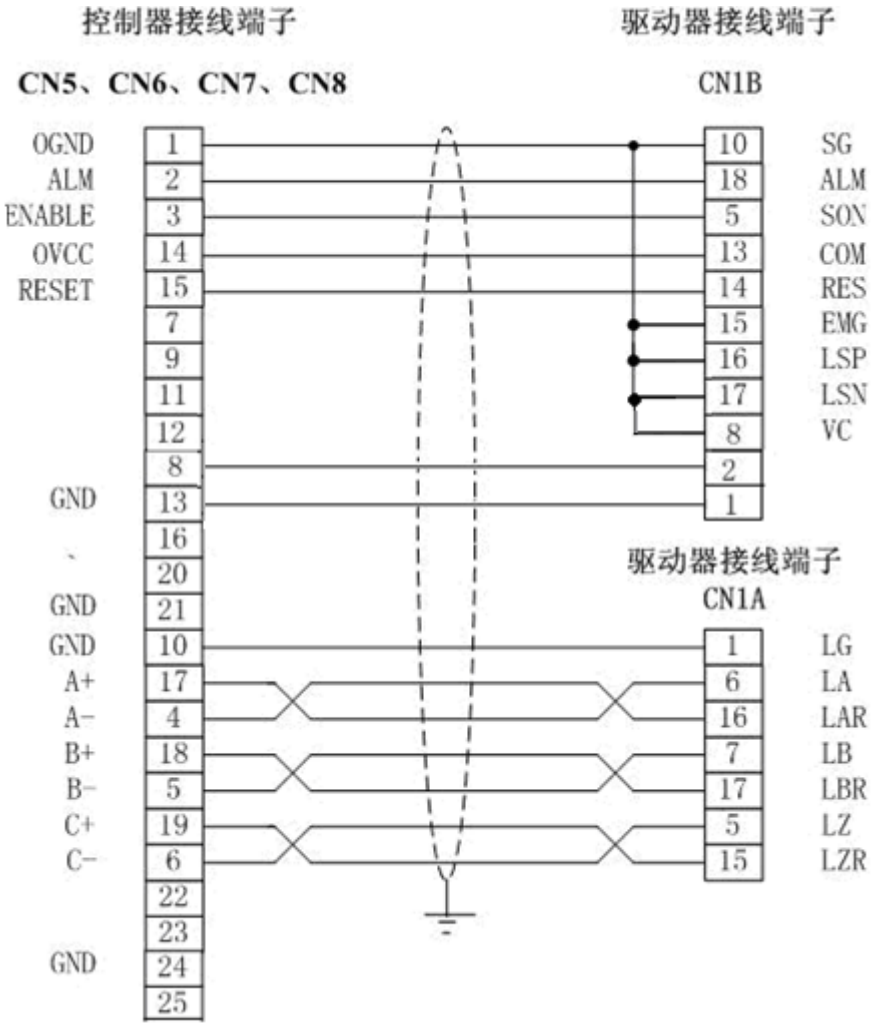
注：本图为 YASKAWA SGDM 型伺服放大器与控制器的接线方式。

B.12 控制器与 YASKAWA SGDM 型驱动器位置控制方式接线

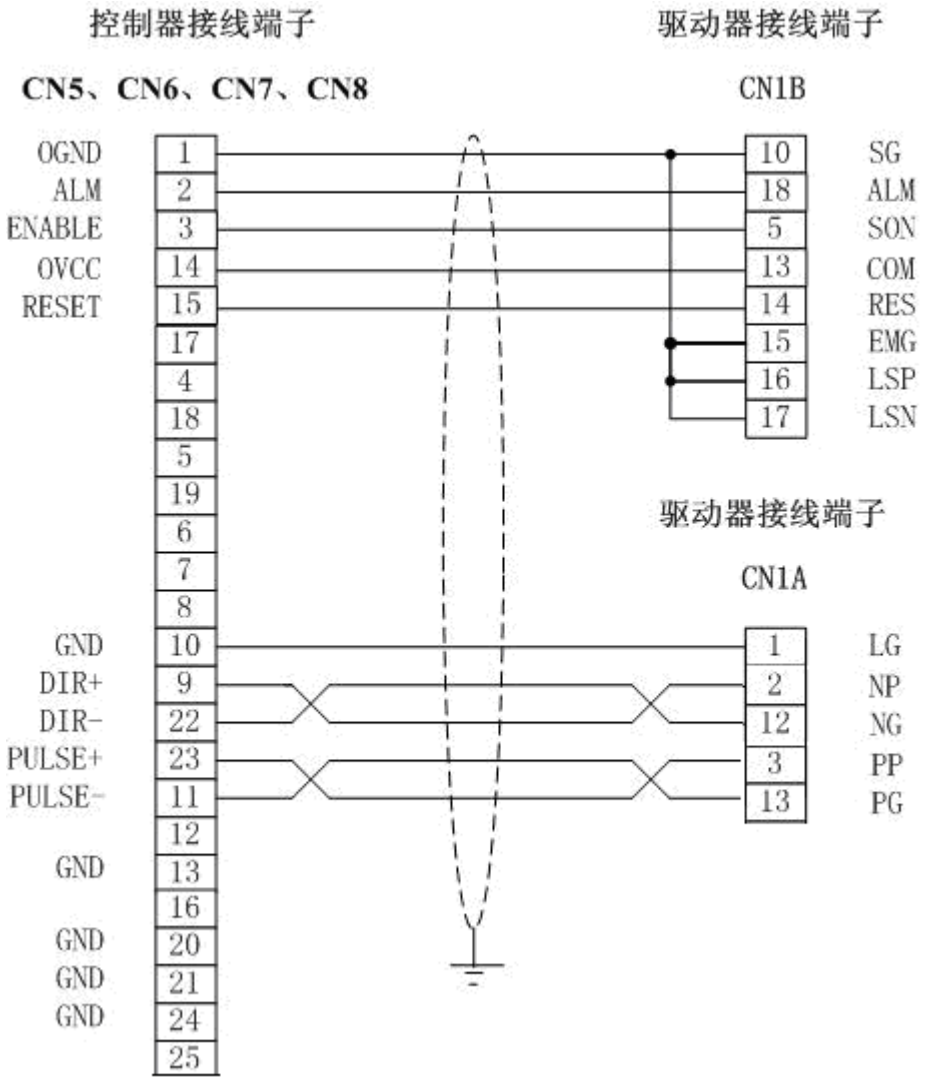


注：本图为 YASKAWA SGDM 型伺服放大器与控制器的接线方式。

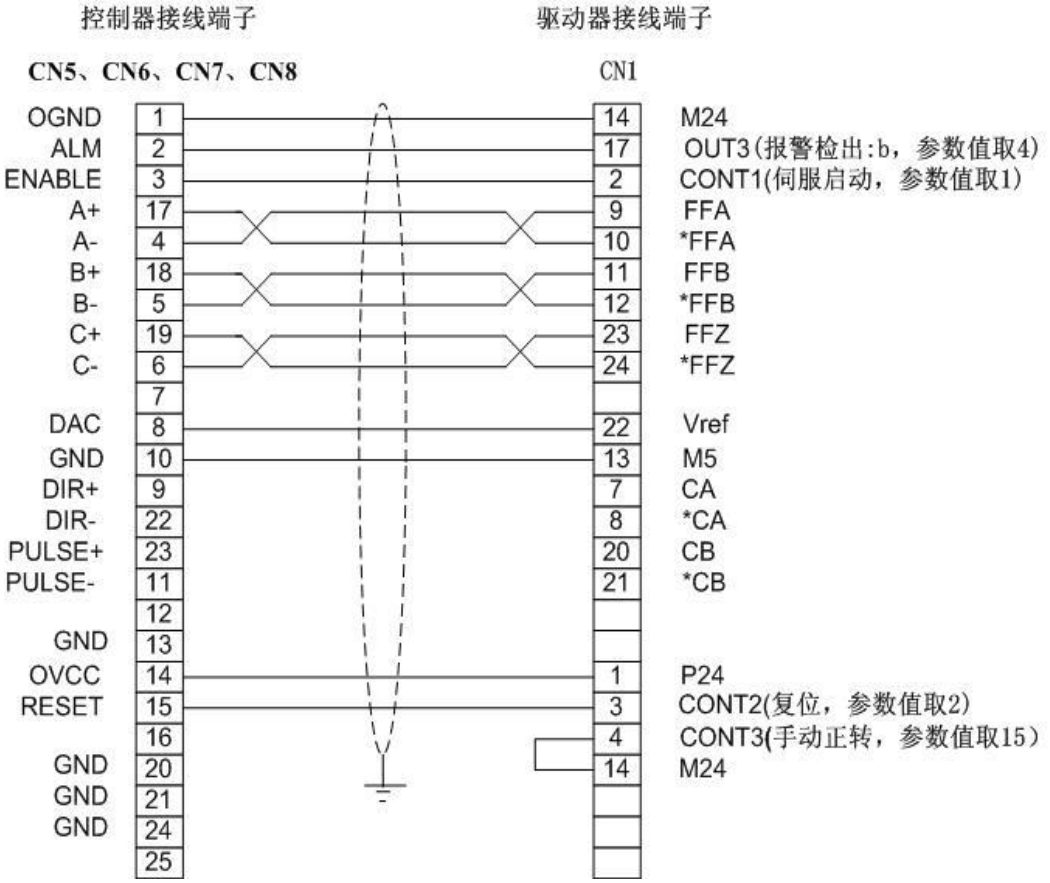
B.13 控制器与三菱 MELSERVO-J2-Super 系列驱动器速度控制方式接线



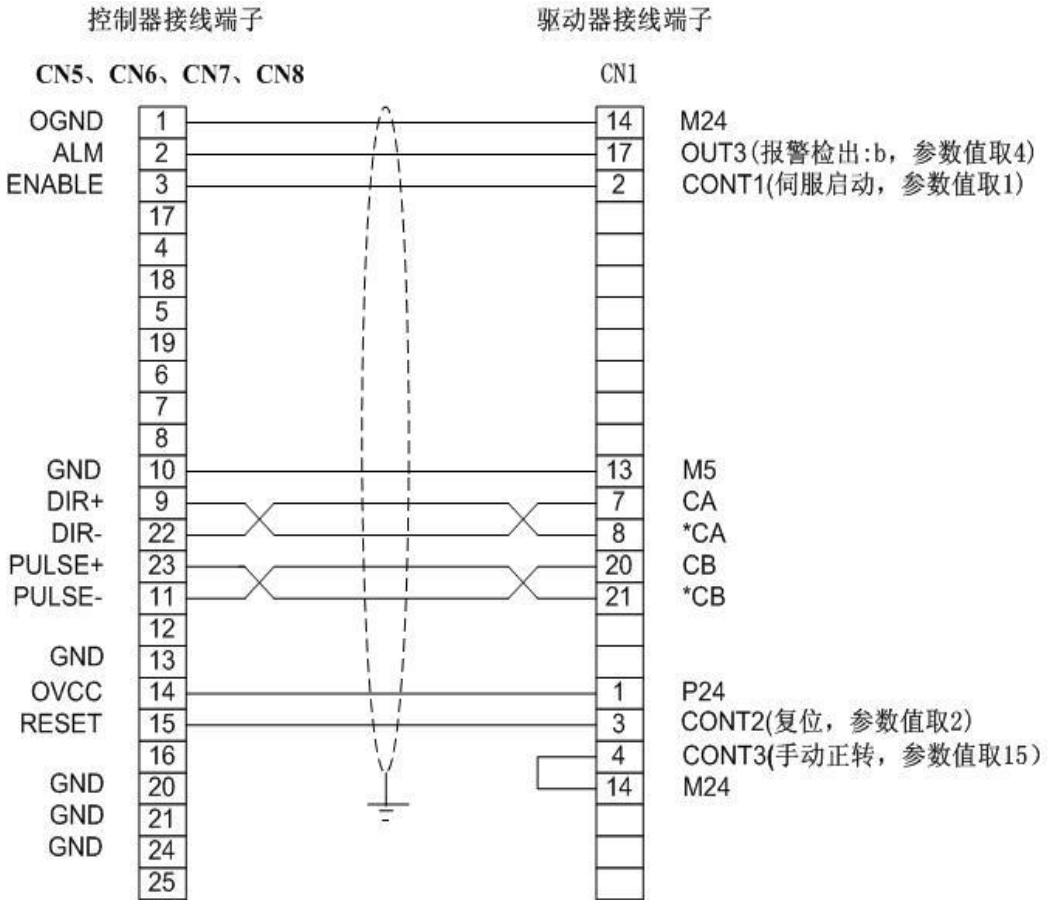
B.14 控制器与三菱 MELSERVO-J2-Super 系列驱动器位置控制方式接线



B.15　控制器与富士 FALDIC-W 系列驱动器速度控制方式接线



B.16 控制器与富士 FALDIC-W 系列驱动器位置控制方式接线



注：本图为富士 FALDIC-W 系列 RYC101D3-VVT2 的伺服驱动器与控制器的接线方式。

附录 C　故障处理

	故　障	原　因	处 理 办 法
1	VGA 不显示	刷新频率设置不正确。 部分 LCD 显示器最大的刷新频率是 60HZ，大于这个刷新频率可能会导致显示错误或不显示。	连接 CRT 显示器或支持较高刷新频率的 LCD 显示器，把刷新频率设为要使用的 LCD 显示器支持的刷新频率。
		显卡驱动问题。 若系统正常启动，进入操作系统前显示正常。进入操作系统后显示故障。按 F8 进入安全模式可用，则是显示模式出问题。信号输出切换到了 LVDS 上。	1、 在进入 XP 之后按 Ctrl+Alt+F1 即可切换到 VGA 显示。 2、 用光盘提供的显卡驱动程序，可以解决此问题。
		BIOS 设置信息丢失。 无法启动，键盘没有反应。则计算机没有启动起来。这种情况可能是 BIOS 出问题。	断电。用牙签等物品点面板上的复位孔，进行 BIOS 放电。
2	LVDS 屏不显示	刷新频率设置不正确	接 VGA 显示器。在 BIOS 下设置适当的分辨率。(与所用的 LVDS 屏匹配) Advanced Chipset Features —>Panle Type(LVDS)—>1024*768*18bit
3	LVDS 屏显示有雪花	接地问题	将电源座子的 SG 与 PE 信号用短接器或粗导线短接
4	USB 设备工作不正常	USB 下端口接外接 USB 光驱不稳定。	使用上端口接光驱
		HMI 口有一路 USB 与 USB 口的下端口共用同一个通道，二者只能用一个。	使用上端口。
		USB 鼠标启动后找不到。 芯片组问题。	重新插上可以工作或更换到另一个 USB 口
5	U 盘启动盘不能启动	BIOS 设置不正确。	BIOS 下需要设置成 USB-HDD 启动模式
		个别 U 盘不能做启动盘	更换其它型号 U 盘
6	插 U 盘后系统不能正常启动	主板芯片组兼容性问题	系统启动过程不能插 U 盘
7	主机与运动控制器通信出错	运动控制器芯片损坏	更换运动控制器
		运动控制器软硬件不配套	更换运动控制器或更换配套软件

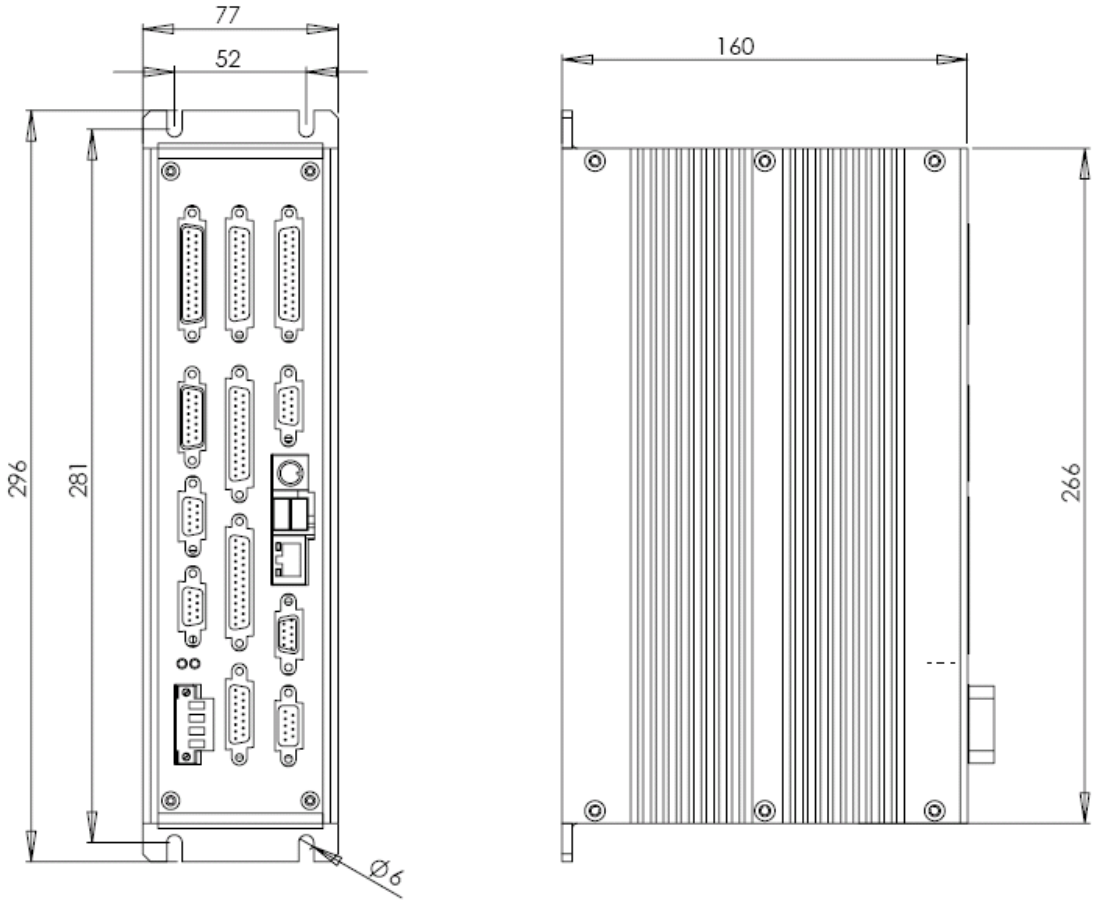
	故　障	原　因	处 理 办 法
8	SV 运控卡复位后，DAC 输出不为零	由于具体工作环境和系统造成初始输出偏差	调整驱动器零漂参数或 调用 GT_SetMtrBias()命令补偿该偏差
9	不能正常读取编码器信号	编码器接线错误	检查编码器接线
		电气噪声	采用带屏蔽的编码器连线、 采用差动输入方式，减小编码器连线长度
		编码器信号频率太高	运动控制器编码器输入信号最高频率不大于 8MHz，选择其它编码器降低分辨率
		编码器不能工作	检查编码器信号
		控制器错误	更换运动控制器
10	电机飞车(SV 卡)	编码器 A，B 相接反	重新连接 A，B 信号连线或者调用 GT_EncSns()命令将 A，B 反相
11	电机震动(SV 卡)	PID 参数设定不正常	调整 PID 参数
12	电机不能控制	运动控制器读到正负限位开关状态均为触发状态，即限位开关触发电平设置不对	重新设定限位开关触发电平
		驱动未使能	调用 GT_AxisOn()，驱动使能
		控制模式设置不匹配	检查驱动器的控制模式，确保与运动控制器设置模式匹配
		电机驱动器报警	检查电机驱动器报警原因，复位电机驱动器。如驱动器无报警输出信号，将 CN5 的 1、2 脚短接或调用相关函数关闭报警信号输入。
		运动控制器有工作异常的状态	检查状态，并加以更正
		电机连线不正确	按说明书检查接线
		接地不正确	按说明书检查接地
		电机力矩输出太小	检查电机驱动器
13	电机位置漂移（SV 卡）	运动控制器处于开环状态	设置成闭环状态
		PID 参数设置不正确，通常 P 参数过小	调整 PID 参数，尤其是加大 P 参数
14	电机驱动器(没有伺服打开信号线)带电的情况下，给	在运动控制器上电和断电时刻处于不定状态，而电机处于工作状态	在给主机上电之前，确保电机驱动器已经断电（即先上弱电、再上强电）

故　障		原　因	处 理 办 法
	主机上电时，电机突然转动		
15	运动控制器输入/输出信号不正确	接线错误	检查接线
		没有提供外部接口电源	检查外部电源供电
		接地错误	重新连接地线
		运动控制器输入/输出接口损坏	更换运动控制器
16	工作不稳定	供电电源功率不够	更换大功率电源

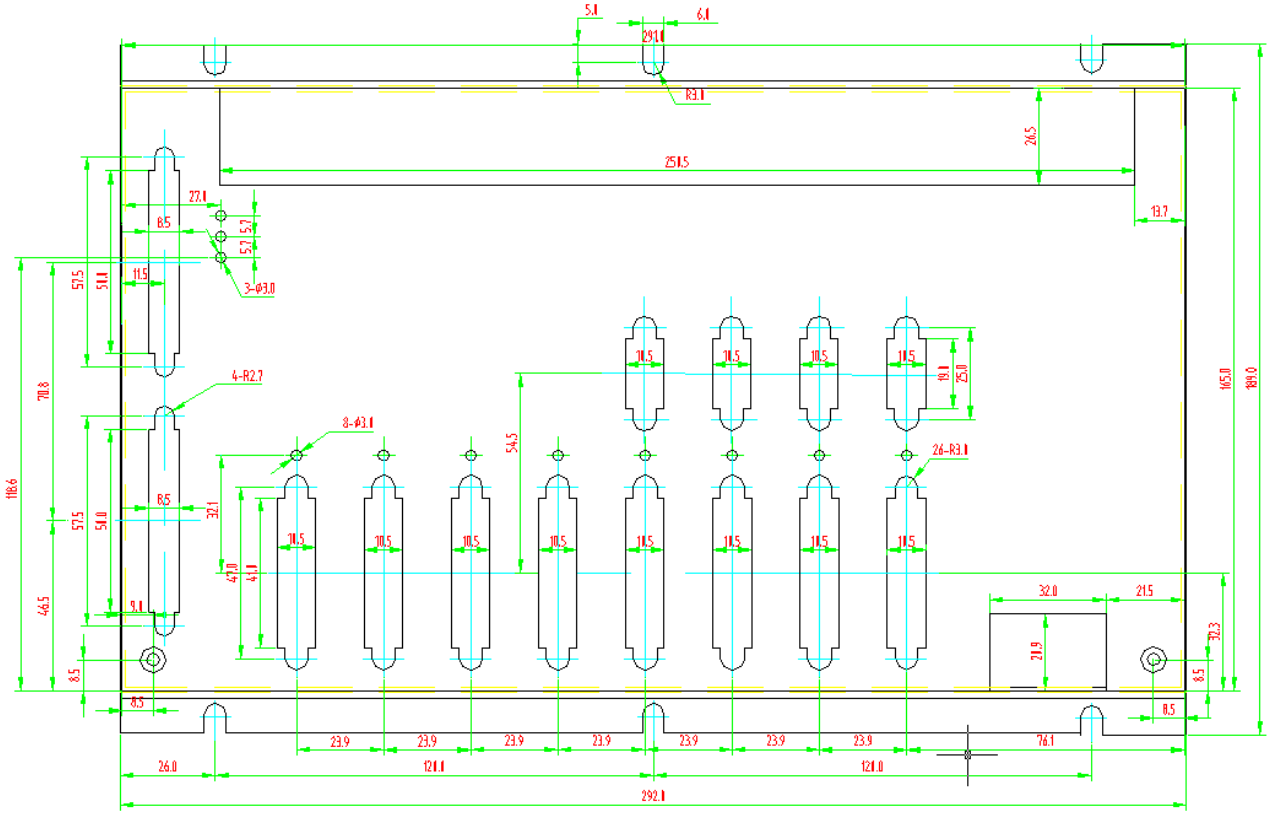
注：1．USB 上、下端口是指 USB 标识字体放正时的上下端口。

附录 D　机械尺寸图

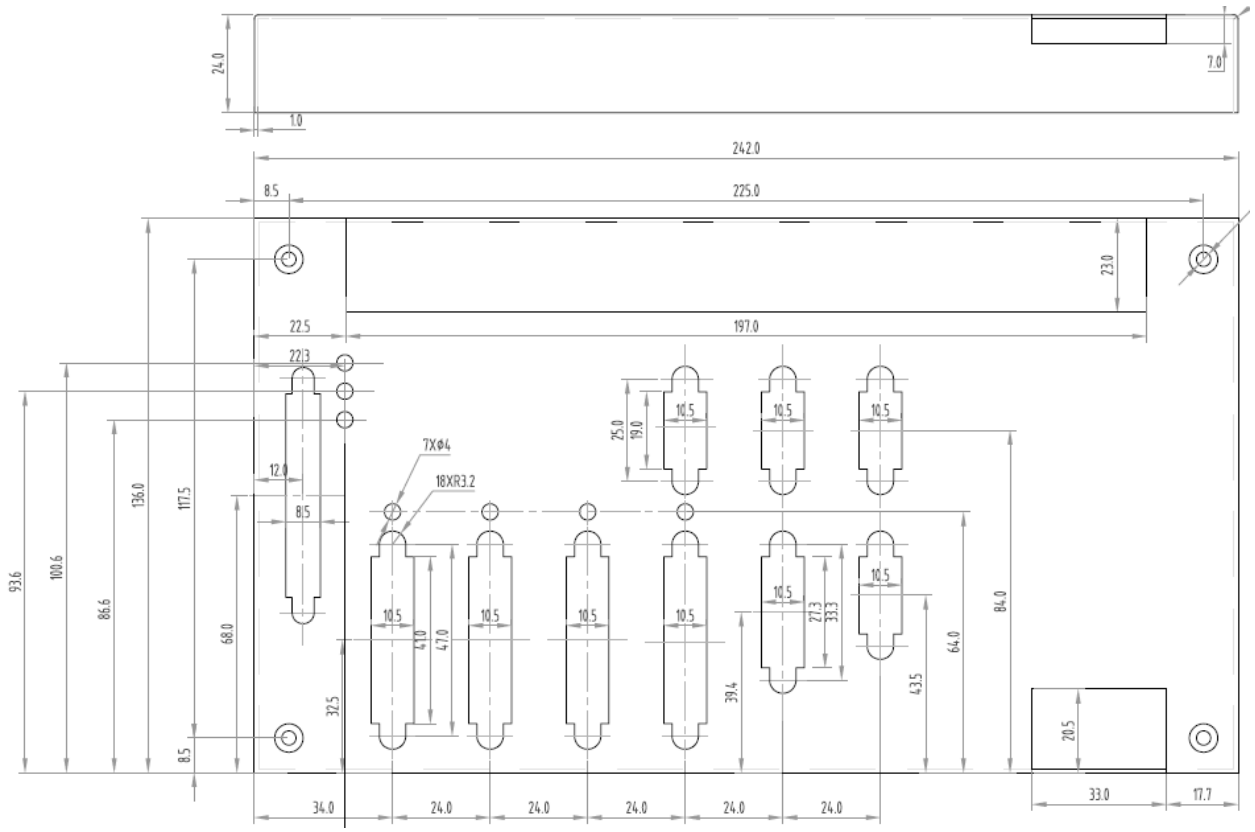
D.1 控制器尺寸



D.2 八轴端子板 GT2-800-ACC2 尺寸



D.3 四轴端子板 GT2-400-ACC2 尺寸



CPAC-OtoStudio 安装指导手册

第一章　概要

1.1 背景和意义

伴随着工业处理器技术的提升和小型化的趋势，工作自动化控制面临着重大挑战。传统的 PLC 控制有许多的优点，比如：在过程控制和输入输出控制中的广泛应用，实时控制能力，易于编程，稳定可靠。但是他同时也有许多缺点，比如：封闭的体系结构、在运动领域中的低应用，与其他的设备的兼容性低，并且人机界面简陋。出于以上原因，固高科技（深圳）有限公司推出了基于 PC 的实时控制器，这种控制器被称为 CPAC，它具有实时 I/O 控制，IEC61131-3 标准编程环境，开放的架构，丰富的人机界面的优点。

CPAC 的软件编程环境 OtoStudio 完全遵从 IEC61131-3 国际标准。这包含了定义架构内部程序构架的语言的顺序功能表，和内部可控制的四种语言：指令表、梯形图、功能图和结构文本。通过模块化的编程方式，增加程序的重用性，降低错误率和提高效率。并且，IEC61131-3 标准建立了一个控制系统的架构。此外，OtoStudio 软件包中还包括诸如任务配置和输入/输出配置在内的程序单元运行环境编辑器。在人机交互方面，OtoStudio 软件提供了可视化的人机界面编写模块，更支持多重界面显示方式。

1.2 OtoStudio 简介

OtoStudio 是用于控制系统应用程序的开发平台。OtoStudio 包含有控制方案编辑器和仿真调试器，是一套完整控制编辑和调试软件包，它为程序员提供了一种操作简单，功能强大的编程语言，实现各种控制方案，并提供功能强大的离线仿真调试工具。它为 CPAC 提供了相应的软件开发平台。

1.3 软件技术的特点和功能

OtoStudio/GRT 是 CPAC 的编程环境及运行引擎。它是一种基于 IEC61131—3 国际标准的工业控制器开发工具。OtoStudio 不仅功能和结构先进而且易于掌握，因而使它成为自动化产品市场上首屈一指的编程工具。软件环境如下图：

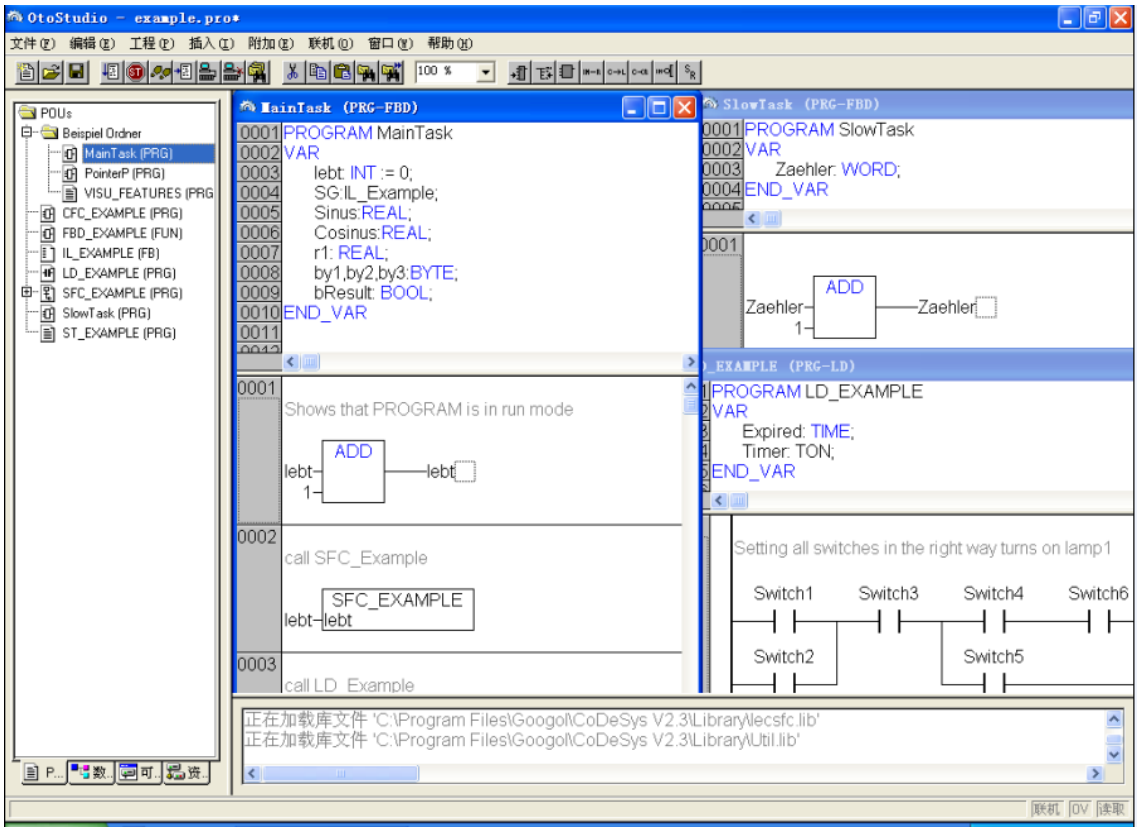


图 1-1 OtoStudio 编程环境

其主要特点有：

✧ **OtoStudio IEC61131－3 编程系统，自动化软件系统的内核**

OtoStudio 软件包中包括符合 IEC61131－3 标准的五种编程语言：即指令表语言（IL）、功能块图（FBD）/连续功能图（CFC）、梯形图（LD）、结构化文本（ST）、顺序功能图（SFC）， 同时也可支持 VC++开发语言开发的动态链接库。此外，OtoStudio 软件包中还包括诸如任务配置和输入/输出配置在内的程序单元运行环境编辑器。

✧ **GRT －Googol Runtime 在 Windows CE 5.0 平台上运行的实时内核**

GRT 软件包可以在 Windows CE5.0 平台上运行。它具有良好的实时特性而对计算机的硬件和软件没有特殊的扩展要求。实时性能保证在 1 毫秒以内，适应几乎所有控制系

统要求，如过程控制，PLC 控制，运动控制等。

✧ **HMI (Human Machine Interface) －集成在 PLC 编程系统中的可视化功能**

HMI 软件包中集成了可视化的功能。所生成的画面可以在编程系统内部显示，也可以在直接显示在 Windows 平台的目标控制器或内部浏览器上。

✧ **Motion Control Function Block－运动控制功能模块**

Motion Control 软件包把运动控制模块直接集成包含了点位运动，插补运动，同步运动等功能块。此外，还包括 PLCopen 的标准功能。

✧ **Web Server －用于自动化网络控制的远程数据接口**

Web Server 软件包中可以提供一个 web 接口，遵循 TCP/IP 通讯协议，可在通过局部以太网，英特网，与各地 CPAC 用户，控制器进行数据，人机交互。并可通过 website 实现网络人机界面功能。

第二章　软件的安装

2.1 安装要求

推荐设备

OtoStudio 软件包是基于 Windows 的应用软件，使用 OtoStudio 时，应具有以下的条件：

✓ □ 一台 PC 机，CPU 为 Pentium II 或更高的处理器，128M 内存；

✓ □ VGA 显示器，或 Microsoft Windows 支持的其他显示器（分辨率 640X480 以上）

✓ □ 至少 150M 的硬盘空间

✓ □ Windows 98/2000/NT、WindowsXP、32 位 Windows Vista 和 32 位 Windows 7 操作系统

✓ □ Microsoft Windows 所支持的鼠标设备

2.2 安装 OtoStudio 软件平台

安装前：

请检查是否安装过旧版本，如果安装了，请先卸载旧版软件，再进行安装过程操作。否则在安装过程中，会提示“程序已经存在，无法安装”。

安装过程：

按照以下的步骤 OtoStudio 软件：

1、将 CD 盘插入光盘驱动器

2、找到 OtoStudio 软件的安装程序 Setup.exe，双击开始软件的安装

Setup.exe

3、按照在线安装向导提示完成软件的安装工作，具体如下：启动 Setup 安装程序，用该程序可自动地进行安装。用户按照屏幕弹出的指南信息一步一步地完成整个安装步骤。

注意：

[1] 推荐安装在默认路径下，即：C:\Program Files\Googol

[2] 如果安装了瑞星，请在安装 OtoStudio 软件前，关闭瑞星。

[3] 在安装过程中，如果防火墙（如 360 安全卫士）提示快捷方式修改，请全部同意，我们向您承诺这些都是安全的。

4、在安装结束后，在开始菜单中选择 Googol->CPAC Platform->OtoStudio.exe，即可打开软件，默认安装好的软件是中文版，也可手动切换语言。 例如切换到英文版本：打开 OtoStudio.exe，选择“Project”下的“Option”，如下图：

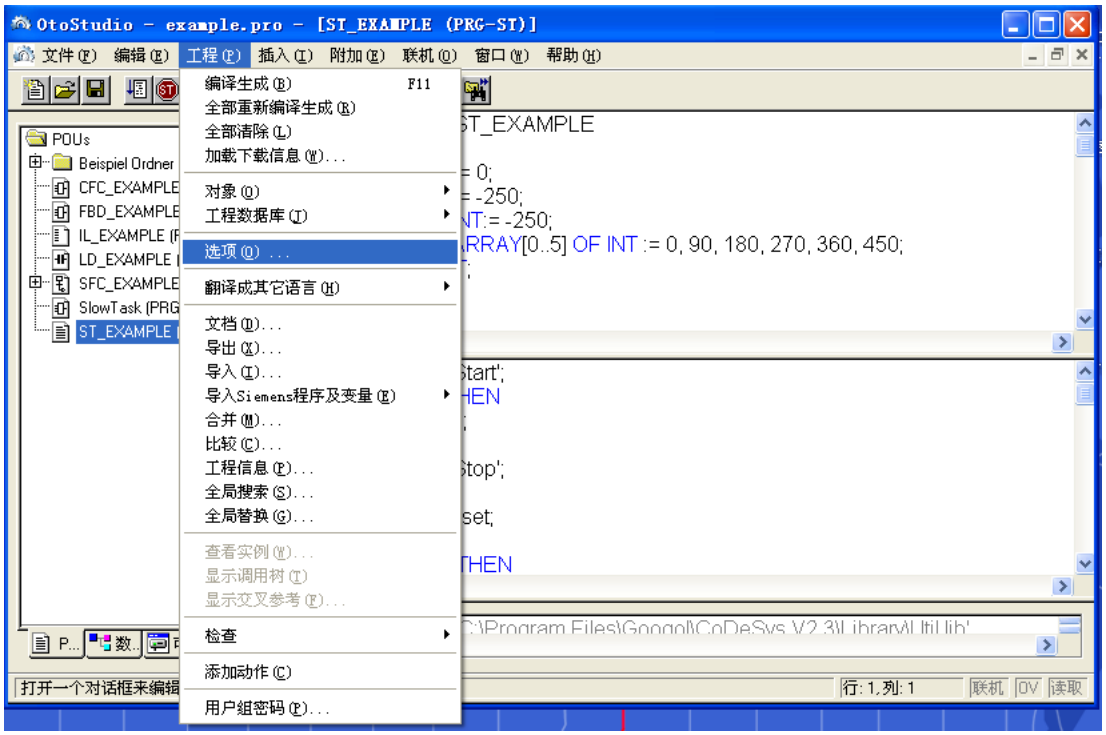


图 2-1

点击 Option 进入对话框，选择 Desktop，在 Language 下选择 English，点击“OK”，如图：

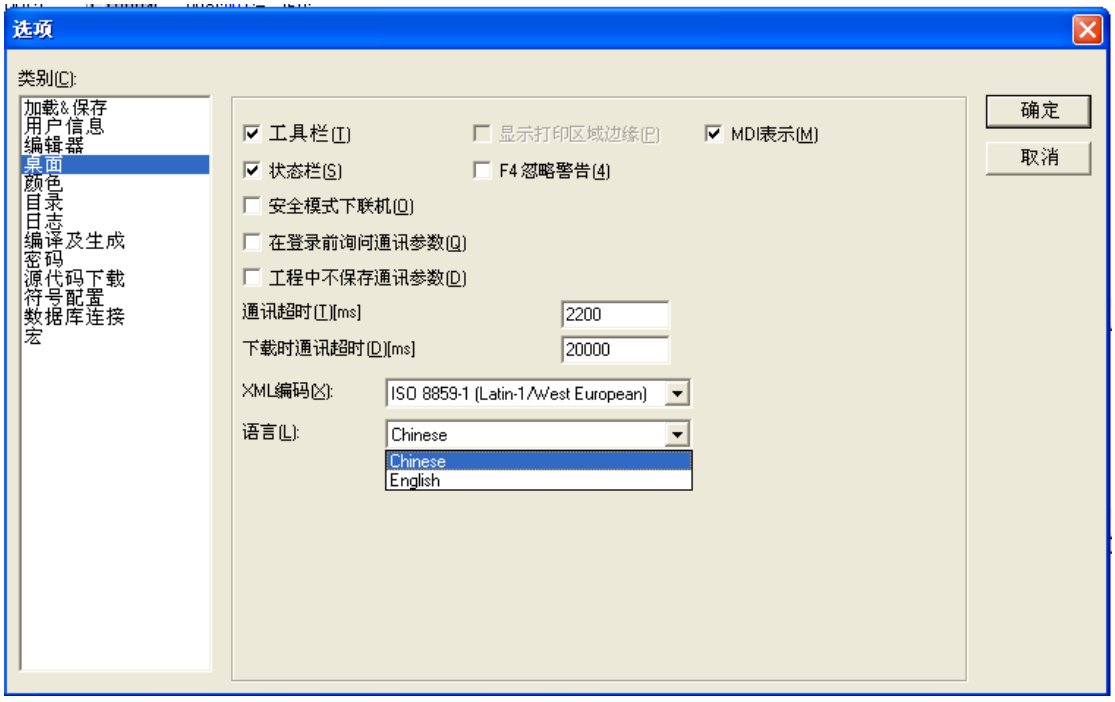


图 2-2

有关 OtoStudio 软件的信息，可以参考随软件附带的光盘或浏览我们的网站（http://www.googoltech.com/ ）

第三章　如何应用 OtoStudio 开发项目

3.1 OtoStudio 的组成

3.1.1 工程组件

3.1.1.1 工程

一个工程包含了 PLC 程序中的所有对象，工程存储在以工程命名的文件中，工程中包含下列对象：POU，数据类型，可视化，资源和库。

3.1.1.2 POU（程序组织单元）

功能、功能块、程序是程序组织单元，它们能够通过动作来增补，每一个程序组织单元都包含一个定义部分和主体部分，主体部分可以用 IEC 的语言来编写，这些语言包括指令列表，结构化文本，顺序功能图，功能模块图，梯形图或连续功能图表。OtoStudio 支持所有 IEC 标准的 POU，如果你想在你的工程文件中使用这些 POU，你必须在你的工程文件中包含标准库文件 standard.lib。POU 可以调用其它的 POU，但递归调用是不允许的。

3.1.1.3 功能

FUNCTION

一个功能(**FUNCTION**)是一个 POU，它正确地产生一个数据元素（可以包含若干元素，比如，字段或者结构体）在处理过程中，可以用文本化语言中的表达式中的一个操作数来调用它。在声明一个功能的时候，一定要给它一个类型，这就是说，在功能名后面加上一个冒号然后跟一个数据类型。

一个正确的功能声明可以参考下面的例子：

FUNCTION Fct: INT

另外，必须分配给功能一个结果，即把功能名作为一个输出变量功能的声明从关键字 FUNCTION 开始。推荐的声明方式。

下例是在指令列表（IL）中的一个功能，它声明了三个输入变量：前两个变量的相乘然

后除以第三个变量。功能返回此操作的结果。

声明部分：

```
FUNCTION Fct: INT
VAR_INPUT
    PAR1:INT;
    PAR2:INT;
    PAR3:INT;
END_VAR
```

程序部分：

```
  LD PAR1
  MUL PAR2
  DIV PAR3
  ST Fct
```

在结构文本中功能的调用可以作为表达式中的一个操作数。功能不会有任何内部条件，这就是说，调用带有相同的输入变量功能将会返回相同的输出结果。功能不会保持内部状态，对于不包含全局变量和地址的功能，每次在它被调用的过程中，给它传递相同的输入变量，它将返回相同的值 。

如果一个局部变量在一个功能中被声明为 RETAIN，这也没有任何影响，为此变量将不会写到保留区。在 SFC 中，一个功能的调用只能发生在一个单步操作或变换之内。

3.1.1.4 功能块

FUNCTION_BLOCK

一个功能块是一个程序组织单元，在程序中提供一个或多个值，与功能相反，一个功能块没有返回值。其中功能块的声明用关键字 **FUNCTION_BLOCK** 开始.推荐的声明方式。可以创建功能块的复制或实例。调用功能块是通过功能块实例实现的。

下面是一个在指令列表中功能块的例子，在指令列表中功能块中包含两个输入变量和两个输出变量，一个输出的是两个输入变量的乘积，另一个是两个输入变量的是否相等的比较。

声明部分：

```
FUNCTION_BLOCK FUB
VAR_INPUT;
    PAR1:INT;
    PAR2:INT;
END_VAR;
VAR_OUTPUT
    MELERG:INT;
    VERGL:BOOL;
END_VAR;
```

在 IL 的执行部分：

```
  LD PAR1
  MUL PAR2
  ST MULERG

  LD PAR1
  EQ PAR2
  ST VERGL
```

关于功能块的使用，详见 3.1.1.5 功能块实例

3.1.1.5 功能块实例

可以创建功能块的复制或实例。每一个实例都有它自己的标识符，并且一个数据结构体中包含它的输入输出和内部变量，实例可以象变量一样被声明为局部变量或全局变量，然而功能块的名称表示标识符的类型。推荐的声明方式。

例如名为 INSTANCE 功能块 PUB 实例：

```
fubInstance ： FUB；
```

功能块通常是通过上述的实例来调用的。从此功能块实例的外部只能访问它的输入输出变量，不能访问它的内部变量调用。

下面是一个访问输入变量的例子 

The function block FB has an input variable in1 of the type INT.

```
PROGRAM prog
    VAR
```

```
    inst1:fb;
    END_VAR
    LD 17
    ST inst1.in1
    CAL inst1
END_PROGRAM
```

功能块 FB 有一个整型的输入变量 in1，功能块和程序的声明部分能够包含实例的声明，实例的声明不能包含在功能之中。访问功能块实例仅限于它被声明的 POU 中，除非它被声明为全局变量。

3.1.1.6 调用一个功能块

通过建立一个功能块的实例并且用下面的语法来规定要求的变量，可以从其它的 POU 访问这个功能块的输入和输出变量。

<实例名>.<变量名>

在调用时为变量赋值：

如果你喜欢在调用功能块的时候再设置输入或输出变量，你可以用指令列表和结构化文本语言。通过在功能块实例名后面的括号中为变量赋值来进行（对输入变量的赋值就象在声明位置的变量初始化一样，使用“：=”来分配变量的值。）如果在 ST 或 IL POU 的执行窗口中使用选项 With arguments，并通过输入帮助（〈F2〉）来插入实例，它将根据这个句式显示所有的变量，但不必为这些变量赋值。

例如：

FBINST 是一个功能块类型的局部变量，它包含了输入变量 xx 和输出变量 yy。当 FBINST 是通过输入帮助插入到了 ST 程序中，将显示如下的调用：FBINST1(xx:= , yy=>)。

在调用输入输出变量时：

请注意，功能块的输入输出变量作为指针来处理。因此在调用一个功能块时，常量是不能赋予 VAR_IN_OUT 并且从外部没有读和写的权限。

例如：

在 ST 模式下调用 fubo 功能块的一个 VAR_IN_OUT 变量 inout1：

```
VAR
    fuboinst:fubo;
    iVar1:int;
```

```
END_VAR
    iVar1:=2;
    fuboinst(iInout1:=var1);
```

下列在这种情况下下列语句示不允许的

```
    fuboinst(iInout1:=2);
```

或

```
fuboinst.iInout1:=2;
```

下面举例说明调用功能块 FUB：

关于功能块 FUB，参照上述‘功能块’部分

声明:

```
FUNCTION_BLOCK FUB
VAR_INPUT
    PAR1:INT;
    PAR2:INT;
END_VAR
VAR_OUTPUT
    MELERG:INT;
    VERGL:BOOL;
END_VAR
```

ST 语言实现:

```
LD PAR1
MUL PAR2
ST MULERG

LD PAR1
EQ PAR2
ST VERGL
```

乘法运算的结果被存储在变量 ERG 中，比较的结果存储在 QUAD 中，FUB 的实例被声明为 INSTANCE

下面是功能块的实例在指令列表中调用的例子

IL 中调用 FUB:

声明部分:

```
PROGRAM AWLaufruf
VAR
    QUAD : BOOL;
    INSTANZ :FUB;
    ERG:INT:=0;
END_VAR
```

执行部分：

```
CAL INSTANZ(PAR1:=5;PAR2:=5);
LD INSTANZ.VERGL
```

ST QUAD
LD INSTANZ.MULERG
ST ERG

下面是功能块的实例在结构化文本中调用的例子（声明部分与指令列表部分相同）

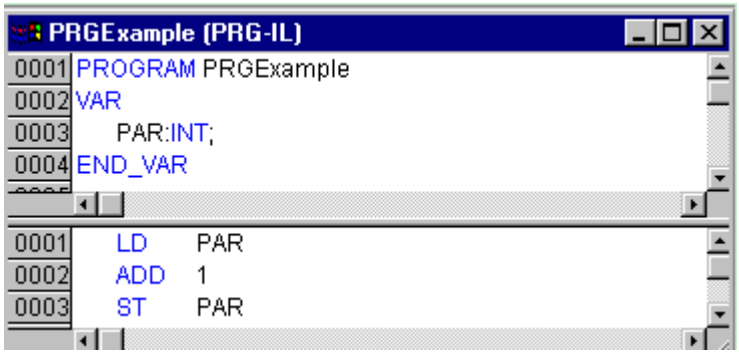
PROGRAM STaufruf
INSTANZ(PAR1:=5;PAR2:=5); bzw. INSTANZ;
QUAD:=INSTANZ.VERGL;
ERG:=INSTANZ.MULERG;

3.1.1.7 程序

PROGRAM

一个程序是一个 POU，它在操作过程中返回多个值，程序在工程文件中是全局的。程序的所有值将保留到下一个程序开始运行。

下面是程序的一个例子  :



程序可以被调用，在一个功能中调用程序是不允许的，同时也不存在程序的实例。

如果一个 POU 调用一个程序，并且如果程序的值发生了变化，那么这些变化将保留到下一次程序的调用时。即使是其它的 POU 内部调用了它。

这和调用功能块不同，那里只有给定的功能块实例中的特定的值才会变化。

当相同的实例被调用时，这些变化才会发挥重要的作用。推荐的声明方式。

程序的声明开始于关键字 PROGRAM 结束于 END_PROGRAM

如果你喜欢在调用程序的时候再设置输入或输出变量，你可以用文本语言如指令列表和结构化文本来做这些。在功能块的实例名后面的括号中为变量赋值（对输入变量的赋值就象在声明位置的变量初始化一样，使用“：=”来分配变量的值。）

在结构化文本或者指令列表程序组织单元的执行窗口中，如果程序是通过带 With arguments 选项的输入帮助插入的，根据这个语法，程序和它的所有变量将自动的显示出来。

但是你不必给这些变量赋值。

下面是程序调用的例子：

IL 中:

```
CAL PRGexample2
LD PRGexample2.out_var
ST ERG
CAL PRGexample2(in_var:=33, out_var=>erg )
```

ST 中:

```
PRGexample2;
Erg := PRGexample2.out_var;
PRGexample2(in_var:=33, out_var=>erg );
```

PLC_PRG 调用顺序的例子：

```
请参照本页之上的程序 PRGexample
LD0
STPRGexample.PAR(*Default setting for PAR is 0*)
CAL IL call(*ERG in IL call results in 1*)
CALST call(*ERG in ST call results in 2*)
CALFBD call(*ERG in FBD call results in 3*)
```

如果程序 PRGexample 中的变量 PAR 在初始化时被主程序赋予 0 值，随后用上面命名的程序调用一个接一个的调用。那么程序中 ERG 的结果会有 1，2 和 3，如果改变了调用的顺序，那么给出的结果变量的值也会相应的跟着变化。

3.1.1.8 PLC_PRG

PLC_PRG

PLC_PRG 是一个特殊的预定义的 POU，每一个工程文件中必须包含一个这样的特殊的程序。实际上这个 POU 在每个控制循环中只调用一次

在一个新工程文件创建之后，将首次使用“工程”“添加对象”命令，在 POU 的对话框的缺省项目是一个名为 PLC_PRG 的程序类型的 POU。你不能更改这些默认的设置。

如果定义了任务  ，那么工程中可以不包含 PLC_PRG，因为在这种情况下，程序的时序依赖于任务的分配。

 注意	**不要删除或者重命名程序组织单元 PLC_PRG(假如你没有使用任务配置)PLC_PRG 是一个单任务程序中的主程序。**

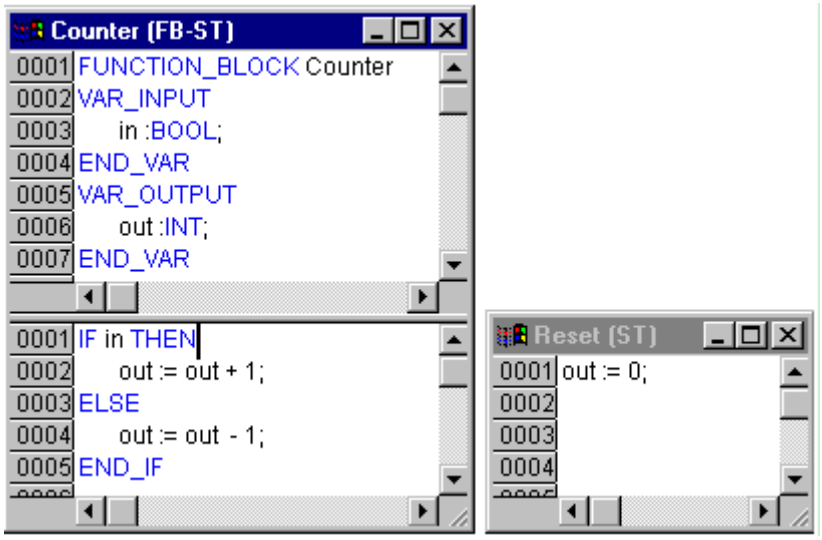
3.1.1.9 动作

ACTION

动作能够被定义并分配给功能块和程序，动作代表了一个另外的执行，它可以用其它的语言进行创建，每一个动作都有一个名称。

每一个动作都是和功能块或者程序中的数据一起工作的，动作使用和标准执行相同的输入/输出变量和局部变量。

下面是一个功能块的动作的例子：



在上面的例子中，调用一个功能块计数器增加或减少输出变量 out 的值，它依赖输入变量 in 的值，调用功能块的复位来设置输出变量为零，相同的变量 out 写到了两个例子中。

调用一个动作：

调用一个动作是通过<程序名>.<动作名>或<功能块实例名>.<动作名>，注意在 FBD 中的注释（看下例）！如果需要在自己的模块中调用这个动作，只需要在文本编辑器和图形界面中使用动作的名称来构成功能块的调用而不必需要实例的信息。

下面是一个从其他的程序组织单元调用上述动作的例子：

声明:

```
PROGRAM PLC_PRG
VAR
    Inst : Counter;
END_VAR
```

采用 IL 编程方式,用另一个 POU 调用 'Reset':

```
CALInst.Reset(In := FALSE)
LDInst.out
STERG
```

采用 ST 编程方式,用另一个 POU 调用 'Reset':

```
Inst.Reset(In := FALSE);
Erg := Inst.out;
```

采用 FBD 编程方式,用另一个 POU 调用 'Reset':

动作在顺序功能图中发挥重要的作用，参照顺序功能图，IEC 标准只认可顺序功能图中的动作，除此之外都不认可。

3.1.1.10 资源

你需要用资源来配置和组织你的工程文件和追踪变量的值

•工程文件或网络中使用的全局变量。

•添加库文件到工程文件中的库文件管理器

•记录在线期间工作的日志文件

•在工程中为报警处理进行报警配置

•配置可编程控制器的 PLC 配置资源

•通过任务来引导创建程序的任务配置

•显示变量值和添加默认变量值的监控和配方管理器

•选择目标设置和必要时的确定的目标系统的最终配置

•作为工程选项的工作空间，根据在 OtoStudio 中作出的目标系统和目标设置，在你的工程中也要用的到下列资源。

•用于变量图形显示的采样追踪

•用于在同一个网络中与其它控制器交换数据的变量管理器

•作为控制监视的 PLC 浏览器

•工具箱、可用性依赖对象系统，用于在 OtoStudio 内部调用它外部的工具程序

3.1.1.11 库

你可以在你的工程文件中包含一系列的库文件，你可以象使用用户定义的变量一样使用库文件的程序组织单元，数据类型，和全局变量，库文件中的标准库文件和 util.lib 是标准的部件并且你经常使用它。

更多的知识请参照“库文件管理器”。

3.1.1.12 数据类型

参照标准的数据类型，用户可以定义自己的数据类型，可以建立结构体枚举类型和引用。

详见“数据类型”

3.1.1.13 可视化

OtoStudio 提供了可视化界面的编程，因此你可以显示工程的变量，通过可视化的帮助你可以在离线的情况下绘制几何元素，在联机模式下能够响应特定变量的值从而改变他们的形式，颜色和文本输出。

可视化的界面可以用作带 OtoStudio 的 HMI 的 PLC 纯操作接口，或者作为一个网页可视化或通过因特网与 PLC 直接连接的对象可视化。

详见“OtoStudio 可视化”用户手册

3.1.2 语言

OtoStudio 支持 IEC_61131 国际标准所描述的所有语言

文本化的语言：

- 指令表
- 结构文本

图形化的语言：

- 顺序功能流程图
- 功能模块图
- 梯形图

还可采用基于功能模块图的连续功能编辑器（CFC）.

3.1.2.1 指令表

IL

指令表中包含一系列的指令，依赖于操作的类型,每一条指令在一个新行开始并且包含运算符号和一个或多个用逗号隔开的操作数。在一个指令前面，还可以有一个标号，后缀一个冒号。注释部分在一行的最后，指令与指令之间可以插入空行。

例如：

LD 17
ST lint (* Kommentar *)
GE 5

JMPC next

LD idword

EQ istruct.sdword

STN test

next:

请参考：
指令表中的修饰符和操作符

指令表中的修饰符和操作符

在指令列表中将用到下面的操作符和修饰符：
修饰符：

C	与操作符 JMP, CAL RET 连用	当前面的表达式处理的结果为 TRUE 时，才执行此指令。
N	与操作符 JMPC, CALC,RETC 连用	当前面的表达式处理的结果为 FALSE 时，才执行此指令。

下面是操作符和它们可能的修饰符以及相关的意思：

操作符	修饰符	含义
LD	N	使当前的值等于操作数
ST	N	在操作数的位置保存当前值
S		当前的值为 TRUE 时，把布尔型操作数置为 TRUE
R		当当前的值为 TRUE 时，把布尔型操作数置为 FALSE
AND		位逻辑运算符号“与”
OR	N,(	位逻辑运算符号“或”
XOR	N,(	位逻辑运算符号“异或”
ADD	N,(	加法
SUB	(	减法
MUL	(	乘法
DIV	(	除法
GT	(	>
GE	(	>=
EQ	(	=
NE	(	<>
LE	(	<=
LT	(	<
JMP	CN	跳转到标号
CAL	CN	调用程序功能块
RET	CN	离开 POU 并返回到调用的地方
)		执行延时操作

在 IL 中也可以在操作之后放一个圆括号。圆括号内的值被认为是一个操作数。
例如：
LD 2
MUL 2
ADD 3
Erg
这里 Erg 的值为 7，但是如果加一个圆括号：
LD 2
MUL (2
ADD 3
)
ST Erg
Erg 的结果是 10，当到达")"时操作 MUL 才开始计算；此时对操作数计算 MUL 5。

3.1.2.2 结构文本

结构化文本中包含一系列的指令，这些用高级语言编写的指令能够被执行（例如 IF……THEN……ELSE）或者在循环（WHILE..DO）。

例如：

```
IF value < 7 THEN
  WHILE value < 8 DO
      value:=value+1;
  END_WHILE;
END_IF;
```

参照：
表达式

表达式

表达式是一个在运算后返回一个值的结构。
表达式由运算符和操作数组成，操作数可以是常量、变量、功能调用或其它表达式。

表达式的计算

依照一定的规则来处理运算符号可以计算出表达式的值，约束力最高的运算符首

先参加运算，然后是约束力稍高的运算符，直到所有的运算符都被处理为止。
运算符号“=”号的处理是从左到右的顺序。
下面是结构文本中运算符号约束力的级别排列

操作	符号	约束力
放入圆括号	(表达式)	最强约束
功能调用	Function name (parameter list)	
求幂	EXPT	
取反	NOT	
乘法	*	
除法	/	
取模	MOD	
加法	+	
减法	-	
比较	<,>,<=,>=	
等于	=	
不等于	<>	
布尔运算与	AND	
布尔运算异或	XOR	最弱约束
布尔运算或		

下面这些是结构化文本中的其它指令，和例子一起安排在一个表中。

赋值

对操作数赋值
“=”号左边是一个操作数（变量，地址），它的右边是赋予它的表达式的值
例如：
Var1 ：=Var2*10
在运算结束后，变量 Var1 就得到了 Var2 的 10 倍值。

指令类型	例子
赋值	A:=B; CV := CV + 1; C:=SIN(X);

功能块调用

在结构化文本中调用功能块
通过写 功能块的实例名和随后在括号中给参数分配值来调用一个 功能块 ，在下面的例子中，通过给两个参数 IN 和 PT 赋值来调用一个定时器，然后结果变量 Q 的值赋予变量 A 结果变量，就象在指令表中，被表示为功能块名称后跟一个小点和变量的名字。

```
CMD_TMR(IN := %IX5, PT := 300);
A:=CMD_TMR.Q
```

指令类型	例子
调用一个功能块并使用功能块输出	CMD_TMR(IN := %IX5, PT := 300); A:=CMD_TMR.Q

RETURN

返回指令可以用来按照条件离开一个 POU（程序组织单元）。

指令类型	例子
返回	RETURN;

IF

IF 指令可以检验一个条件，根据这个条件，执行指令。

IF <Boolean_expression1> THEN
<IF_instructions>
{ELSIF <Boolean_expression2> THEN
<ELSIF_instructions1>
ELSIF <Boolean_expression n> THEN
<ELSIF_instructions n-1>
ELSE
<ELSE_instructions>}
END_IF;

在{}中的部分是可选的。

如果布尔运算表达式<Boolean expression>返回 TRUE，只有 if 指令部分执行，其它部分不执行。

否则，布尔运算表达式从<Boolean expression 2>开始，一个接一个的计算，直到某个布尔表达式返回为 TRUE，然后，在这个布尔运算表达式 2 之后，ELSE 或 ELSE IF 之前的部分被计算。

如果没有任何一个布尔运算表达式返回 TRUE，那么只计算 ELSE 下的指令

例如：

```
IF temp<17
THEN heating_on := TRUE;
ELSE heating_on := FALSE;
END_IF;
```

这里当温度降到 17 度以下时加热开始，否则保持关闭状态。

指令类型	例子

IF 语句	D:=B*B;
	IF D<0.0 THEN
	　　C:=A;
	ELSIF D=0.0 THEN
	　　C:=B;
	ELSE
	　　C:=D;
	END_IF

CASE

使用 CASE 指令，可以在一个结构中，用同一个条件变量组合多个有条件的指令。

句式：

CASE <Var1> OF

<Value1>: <Instruction 1>

<Value2>: <Instruction 2>

<Value3, Value4, Value5>: <Instruction 3>

<Value6 .. Value10>: <Instruction 4>

...

<Value n>: <Instruction n>

ELSE <ELSE instruction>

END_CASE;

•CASE 指令根据下面的模式来处理

•如果变量 Var1 有值 Value1，那么执行指令 Instruction1。

•如果变量 Var1 不是所指明的值，那么执行 ELSE Instruction。

•如果有多个变量值要执行同一个指令，那么这些条件执行一个公共指令

•如果对于一个变量在一个值的范围内执行同一个指令，那么在初始值和最后值之间用两个句点隔开，所以你可以规定公共条件。

例如：

CASE INT1 OF

1, 5: BOOL1 := TRUE；

BOOL3 := FALSE；

2: BOOL2 := FALSE；

BOOL3 := TRUE；

10..20: BOOL1 := TRUE；

BOOL3:= TRUE；

ELSE

BOOL1 := NOT BOOL1；

BOOL2 := BOOL1 OR BOOL2；

END_CASE；

指令类型	例子
CASE 语句	CASE INT1 OF 1: BOOL1 := TRUE; 2: BOOL2 := TRUE; ELSE BOOL1 := FALSE; BOOL2 := FALSE; END_CASE;

FOR

通过 FOR 循环程序可以编写重复执行的过程。

句式：

INT_Var :INT；

FOR <INT_Var> := <INIT_VALUE> TO <END_VALUE> {BY <Step size>} DO

<Instructions>

END_FOR；

{}内的部分是可选的。

只要计数器 INT_Var 不大于 END_VALUE，指令 Instructions 就一直执行，在执行 Instructions 之前首先检查计数器的值，如果 INIT_VALUE 比 END_VALUE 大的话 Instructions 将不在执行。

当 Instructions 执行后，INT_Var 通常要增加一个 Step size，Step size 可以是任何整型值，如果没有 Step size，它将设置为 1，当 INT_Var 大到一定值时，循环结束。

例如：

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

我们假设 Var1 的默认值是 1，那么在循环结束后它将得到值 32

注意：END_VALUE 一定不要大于等于与计数器 INT_VAR 的极限值，例如：如果变量计数器是一个 SINT 类型并且 END_VALUE 为 127，那么这将是一个死循环。

指令类型	例子

FOR 语句	J:=101;
	FOR I:=1 TO 100 BY 2 DO
	IF ARR[I] = 70 THEN
	J:=I;
	EXIT;
	END_IF;
	END_FOR;

WHILE

WHILE 循环

WHILE 循环可以象 FOR 循环那样使用，不同之处在与 WHILE 循环的退出条件可以是任何布尔型表达式，当条件满足时，就会执行循环。

句式：

```
WHILE <Boolean expression>
<Instructions>
END_WHILE;
```

只要 Boolean_expression 返回 TRUE，那么就重复执行 Instructions 如果 Boolean_expression 在首次计算出 FALSE，那么指令将不再执行，如果 Boolean_expression 从不出现 FALSE，Instructions 将没完没了的重复执行。

注意：程序员必须保证不出现死循环，这可以通过改变循环中指令部分的条件来实现，例如：可以通过计数器增加或减少。

例如：

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

对于 WHILE 和 REPEAT 个循环在循环之前不必知道循环的次数，从这个意义上来说，这两种循环要比 FOR 要强大一些。因此在这种情况下，可以用这两种循环。如果循环数比较明确，那么 FOR 循环因为没有死循环而更好一点。

指令类型	例子
WHILE 语句	J:=1; WHILE J<= 100 AND ARR[J] <> 70 DO J:=J+2; END_WHILE;

REPEAT

REPEAT 循环和 WHILE 循环的不同之处在于它的中断条件是在循环执行之后才被检查，这就是说，循环至少要执行一次，不管中断是什么条件

句式：

REPEAT

<Instructions>

UNTIL <Boolean expression>

END_REPEAT；

Instructions 一直执行到 Boolean expression 返回 TRUE

如果 Boolean expression 第一次就赋予真值，Instructions 只执行一次，否则 Instructions 将重复执行将会导致时间延迟。

注意：程序员可以通过改变循环中指令部分的条件来保证没有死循环出现，例如：可以通过计数器增加或减少。

例如：

REPEAT

Var1 := Var1*2

Counter := Counter-1；

UNTIL

Counter=0

END_REPEAT；

指令类型	例子
REPEAT 语句	J:=-1; REPEAT J:=J+2; UNTIL J= 101 OR ARR[J] = 70 END_REPEAT;

EXIT

如果在 FOR WHILE 或 REPEAT 循环中有 EXIT 指令，那么内循环就结束，不管中断是什么条件。

指令类型	例子
EXIT 语句 退出当前循环，相当于 C 语言中的”break”	EXIT;

空指令

指令类型	例子
空指令	;

3.1.2.3 顺序流程图

顺序功能图是基于图形化的语言，用它可以描述一个程序中不同动作的先后顺序。因为这些动作分配给单步元素，通过变迁元素来控制处理的顺序。

3.1.2.4 功能模块图

功能模块图是一种基于图形的编程语言，它用一串网络来工作，每一个网络包含一个算术或逻辑表达式、功能块的调用、跳转或返回指令的结构。

3.1.2.5 梯形图

梯形图也是一种基于图形化的编程语言，它接近于电子电路的结构，一方面，梯形图很适合构建逻辑开关，另一方面，它也能创建象 FBD 中的网络图，所以梯形图在控制调用其它程序组织单元的时候是很有用的。

梯形图包含了一系列的网络，左右两边各有一个垂直的电流线，网络图仅限制于左右两母线之间的范围内，在中间是由线圈触点和连接线组成的电路图。

每一个网络包含左边的一系列触点，这些触点根据布尔变量值的 TRUE 和 FALSE 来传递从左到右的开和关的状态。每一个触点是一个布尔变量，如变量值为 TRUE，电路从左到右通过连接线就连通。否则右边接收到“关“的值。

3.1.2.6 连续功能编辑器

连续功能图表编辑器不象功能模块图表那样操作，但是可以自由放置元素，它允许使用反馈。

3.1.3 调试，联机功能

1）采样追踪

采样追踪允许你追踪变量的连续变化的值，它依赖于所谓的触发事件，触发事件是先前定义的布尔变量（触发变量）的上升沿或下降沿。OtoStudio 允许对 20 个变量进行追踪，每一个变量可以追踪 500 个值。

2）调试

OtoStudio 的调试功能可以让你很容易的找到错误。为了调试，运行‘工程’‘选项’命令并且在生成选项对话框中选择动态调试。

3）断点

断点是程序处理过程中停止的位置，因而它可以在程序中的特定位置观察变量值的变化。

断点可以在编辑器中设置，在文本编辑器中断点在行的编号处设置,在连续功能图中和梯形图中是在网络编号处设置,在 CFC 中是在程序组织单元处设置,在 SFC 中是在步处设置，在功能模块图的实例中不能设置断点。

4）单步

单步是:

在指令表中:执行程序直到运行 CAL LD 和 JMP 命令。

在结构化文本中：执行下一条指令。

在功能模块图 梯形图中： 执行下一步网络。

在顺序功能图中：继续目前的动作 直到下一个步开始。

通过一步一步的运行你可以检查程序中的逻辑错误。

5）单循环

如果选择了单循环，每一个循环结束，执行也就结束。

6) 联机模式下改变值

在操作过程中，变量可以设置为一个特定的值（写入新值）或者在每一个循环之后重新定义为特定的值（强制新值）。在联机模式下可以通过双击变量的值来改变它的值，布尔变

量从 TRUE 变为 FALSE 或从 FALSE 变为 TRUE。对于每一种类型的变量都可以打开写入变量对话框。在这里可以编辑变量的真实值。

7）监控

在联机模式下，所有的显示变量从控制器中读出并及时的显示。你可以在定义和程序编辑器中找到这些显示。你也可以在观察和接收器中读出变量的当前值并且可以看到它们。如果要监视功能模块的实例中的变量,相应的实例块必须已经打开。在监视 VAR_IN_OUT 变量时，不引用的值将输出。在监视指针时，指针和不引用的值都将在声明部分输出。在程序部分，只有指针输出：

+ --pointervar = '<'pointervalue'>'

在不引用值中的 POINTER 也相应的显示。在行上双击或在交叉上单击，显示或是展开或是收缩。在执行部分，显示指针的值。对于不引用，将显示不引用的值。

监视数组元素：数组元素除了由常量指出的之外，还有由变量指出的：

anarray[1] = 5

anarray[i] = 1

如果索引中包含有表达式(例如， [i+j] or [i+1])，元素不能显示出来。

请注意：如果已经达到了被监视变量的最大编号，对于随后的变量不是显示当前的值，而是显示字符串“监视的变量太多”。

8）仿真

在模拟过程中，创建的 PLC 程序不在实际的 PLC 中运行，而是 OtoStudio 系统中的计算器中运行。所有的联机功能都是可用的。它允许你在无需 PLC 硬件的情况下检测逻辑的正确性。

注意：外部库文件的 POU 是不能运行在模拟的模式的。

9）日志

日志记录着用户的操作、内部进程、状态变换和联机模式处理过程中发生的意外的情形。它用来监视和跟踪错误。

3.2 开发步骤

在 OtoStudio 安装完成后，接下来就可以在计算机上进行相应项目的程序开发工作了，

进行一个项目的完整开发过程应该按下列步骤顺序进行：

- 首先打开 OtoStudio 软件；
- 新建一个项目；
- 进行目标系统设置，同时进行相应的设置；
- 选择主程序的编程语言（IL、ST、FBD、LD、SFC、CFC）；
- 在资源的库文件管理器里加载相应的库文件；
- 在资源的 PLC 配置里进行相应的 PLC 配置；
- 在 POUS 中进行程序的开发；
- 在可视化界面里进行可视化界面的编写；
- 按照需要进行相应的任务配置；
- 对程序进行编译、仿真调试；
- 编译通过后设置通讯参数；
- 登录进行程序的下载；
- 运行工程；
- 设置密码保护。

3.3 程序开发的具体操作

3.3.1 如何建立新工程

1）打开OtoStudio.exe,，选择新建项目；在如图3-1 中显示的窗口中，选择硬件控制器“CPAC GUC-XOO-TPX”，或者“CPAC GUC-XOO-TPX Mini”如图3-2 所示窗口。

图 3-1

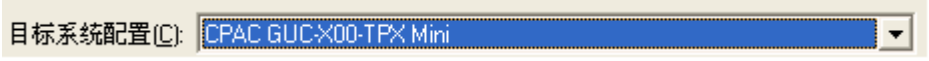


图 3-2

注：“CPAC GUC-XOO-TPX”和“CPAC GUC-XOO-TPX Mini”两个平台支持的扩展总线不同。在 PLC 配置中会详细介绍。

2）在所有的选项卡中选择默认值，单击“确定”，弹出如图3-3 所示窗口。

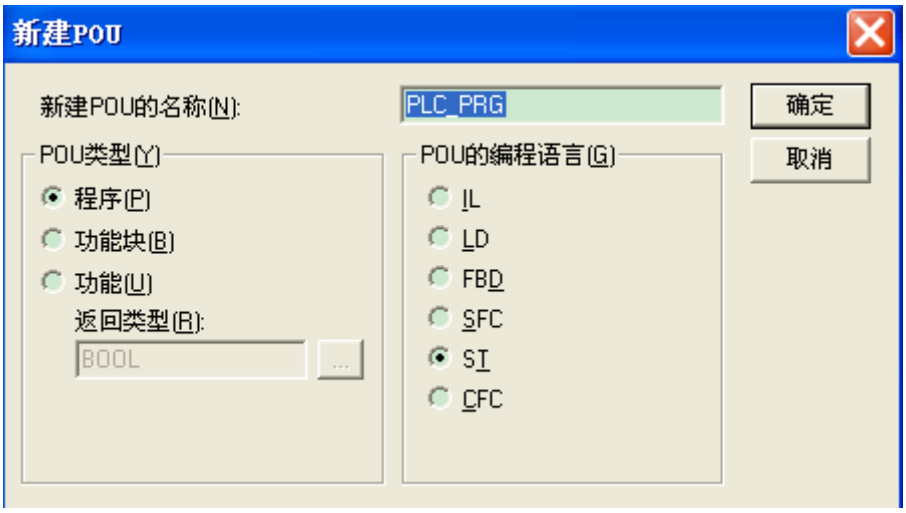


图 3-3

3）键入程序名称，选择程序类型，以及程序语言(IL、ST、FBD、LD、SFC、CFC)。

3.3.2 库文件导入

在编辑程序之前，系统会自动导入需要的库文件。如果用户想手动添加，可如图3-4 所示，在“资源”选项卡中，双击“库文件管理器”，在空白处右键单击选择“添加库文件”，弹出图3-5 所示的窗口，选择所需的库文件。最基本的库文件为“Standard.lib”，“CPAC GUC-X00-TPX.lib”。

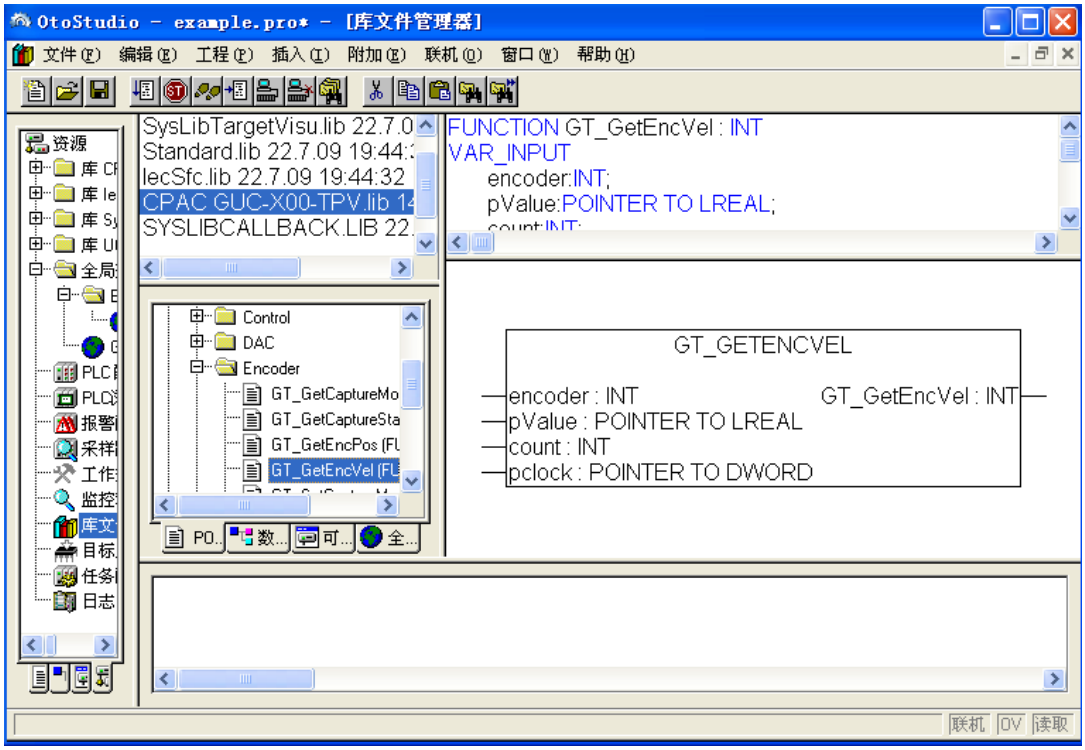


图 3-4

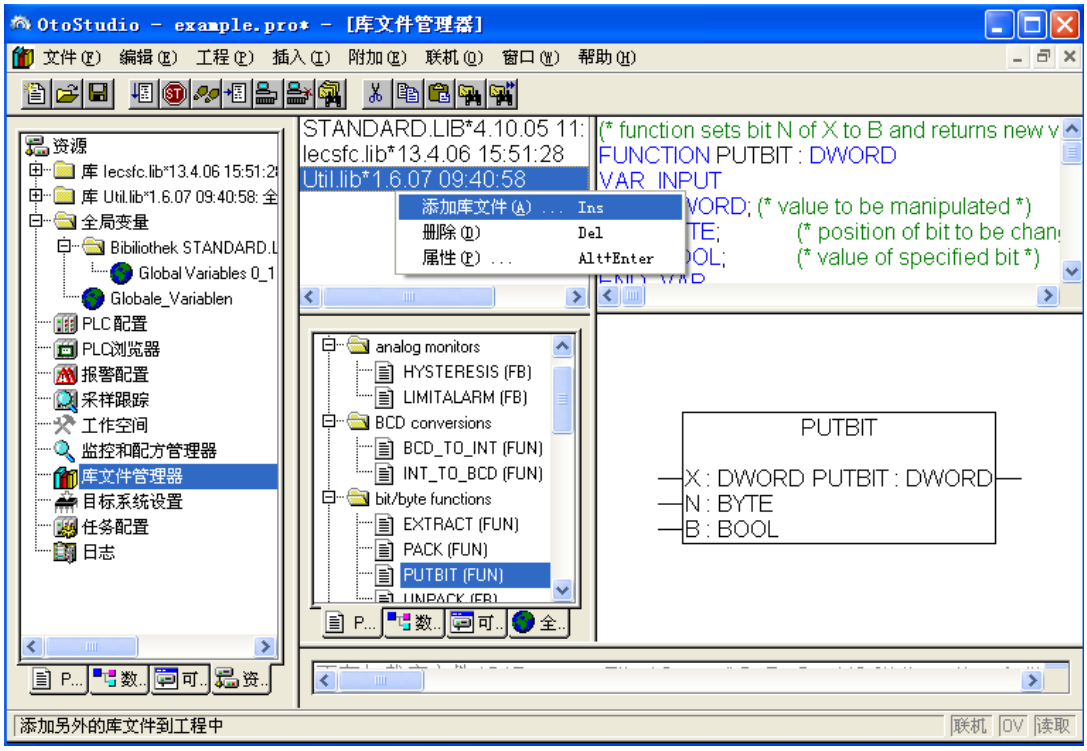

OtoStudio - example.pro* - [库文件管理器]
文件(F) 编辑(E) 工程(P) 插入(I) 附加(E) 联机(O) 窗口(W) 帮助(H)
资源
库 Util.lib*1.6.07 09:40:58: 全
全局变量
Global Variables 0_1
Globale_Variablen
PLC 配置
PLC浏览器
报警配置
采样跟踪
工作空间
监控和配方管理器
库文件管理器
目标系统设置
任务配置
日志
STANDARD.LIB*4.10.05 11:
lecsfc.lib*13.4.06 15:51:28
Util.lib*1.6.07 09:40:58
添加库文件(A) ... Ins
删除(D) Del
属性(P) ... Alt+Enter
(* function sets bit N of X to B and returns new v
FUNCTION PUTBIT : DWORD
VAR_INPUT
(* value to be manipulated *)
(* position of bit to be chan
(* value of specified bit *)
analog monitors
HYSTERESIS (FB)
LIMITALARM (FB)
BCD conversions
BCD_TO_INT (FUN)
INT_TO_BCD (FUN)
bit/byte functions
EXTRACT (FUN)
PACK (FUN)
PUTBIT (FUN)
PUTBIT
X : DWORD PUTBIT : DWORD
N : BYTE
B : BOOL
添加另外的库文件到工程中
联机
OV
读取

图 3-5

3.3.3 PLC 配制

1)同样在“资源”选项卡中，双击“PLC配置”，如图3-6所示，右键单击选择“添加子元件”中的“IO module”。

针对”CPAC GUC-X00-TPX”平台，可以添加IM/SM系列IO模块，也可以添加ACC系列IO模块，以及HMI操作面板，组态出IO如图3-6

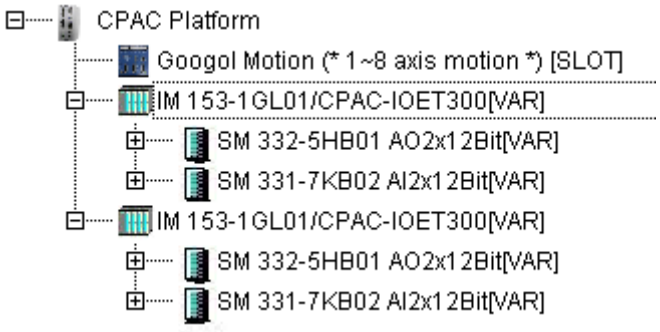


图 3-6

- 注意：必须先配置所有的 300 模块，然后再配置其他模块。（站号可以分别设置）

针对”CPAC GUC-X00-TPX Mini”平台，只可以添加ACC系列IO模块，组态出IO如图3-7

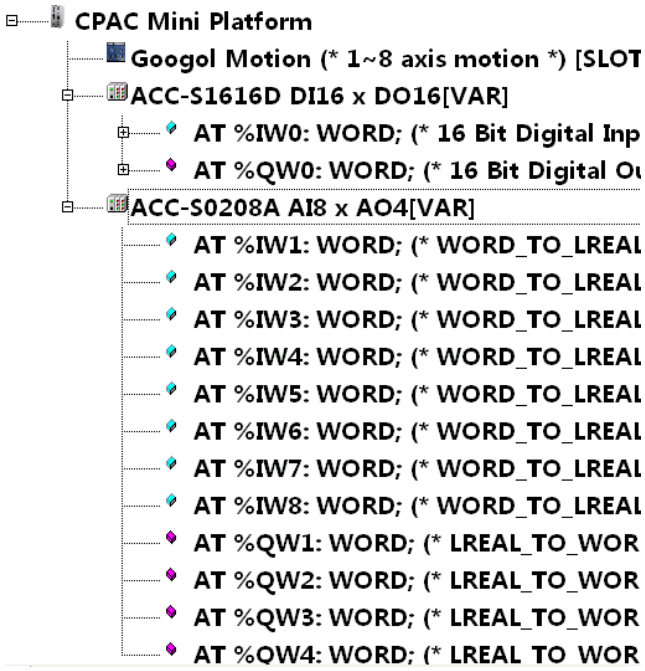


图 3-7

3.3.4 程序的编写

完成建立新项目的准备工作之后，保存文件，即可开始编写程序。

1）函数声明：如图3-8，PLC_PRG的变量声明在上方对话框中定义

```
0001 PROGRAM PLC_PRG
0002 VAR
0003     rtn: INT;
0004     sts: WORD;
0005     i: INT;
0006     a: BOOL := 1;
0007     prfPnt: ARRAY [1..4] OF LREAL;
0008     POS_X: DINT;
0009     POS_Y: DINT;
0010 END_VAR
0011
0012
```

图 3-8

2）函数体书写：在下方空白处 F2，调用 CPAC GUC-X00-TPV.lib 的函数，如图 3-9。

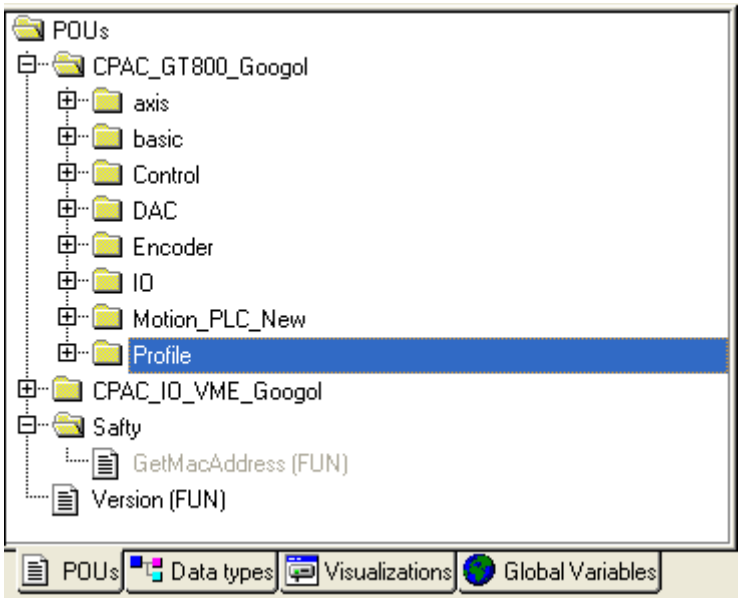


图 3-9

按照这些操作，可得出下面程序（具体函数功能可参考函数编程手册）

```
PROGRAM PLC_PRG
VAR
    rtn: INT;
    sts: DWORD;
    Prm: TTrapPrm;
```

```
    vel: LREAL := 3;
    a: BOOL := TRUE;
    Pos: DINT := 20000;
    axis: INT := 1;
    PrfPos: LREAL;
    PrfVel: LREAL;
    card_Num: INT:=0;
    i: INT;
    Timer: INT:=0;
END_VAR
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
IF a THEN
FOR i:=1 TO 8 BY 1DO
rtn:=GT_ClrSts(i,1);
rtn:=GT_GetSts(i, ADR(sts), 1, 0);
rtn:=GT_AxisOn(i);
END_FOR
END_FOR
rtn:=GT_SetCardNo(0);(*0: card 1; 1: card 2*)
rtn:=GT_PrfTrap(axis);
Prm.acc:=0.001;
Prm.dec:=0;
Prm.velStart:=0.5;
Prm.smoothTime:=0;

rtn:=GT_SetTrapPrm(axis, ADR(Prm));
rtn:=GT_GetTrapPrm(axis, ADR(Prm));

rtn:=GT_SetVel(axis, vel);
rtn:=GT_SetPos(axis,Pos);
rtn:=GT_Update(SHL(DWORD#1,(axis-1)));
a:=FALSE;
END_IF

rtn:=GT_GetPrfVel(axis, ADR(PrfVel), 1, 0);
rtn:=GT_GetPrfPos(axis, ADR(PrfPos), 1, 0);
```

3.3.5 人机界面编写

OtoStudio的另一功能是能够进行人机界面的模拟，方便客户调试，步骤如下。

1）在“可视化”选项卡中，在空白处右键，选择“添加对象”，键入自定义名称。

2）在如图3-10 所示中，单击“示波器”图标，在空白处画出要添加的对象的区域后，

双击示波器，弹出对话框，如图 3-11

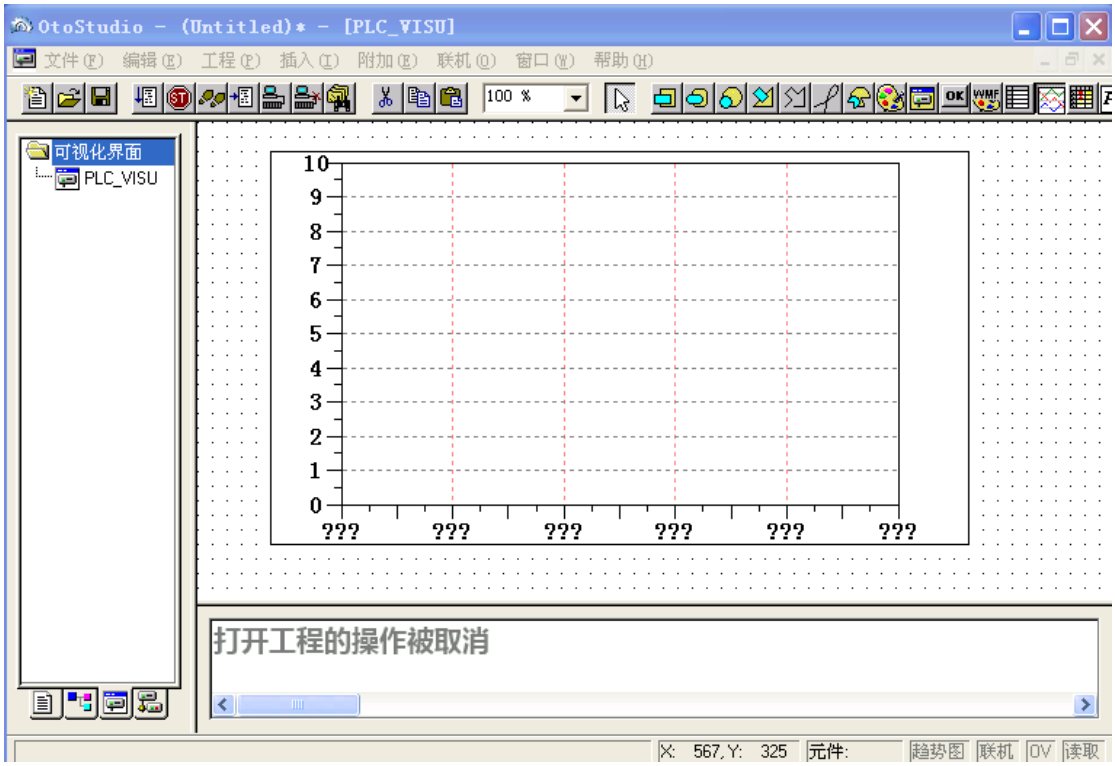


图 3-10

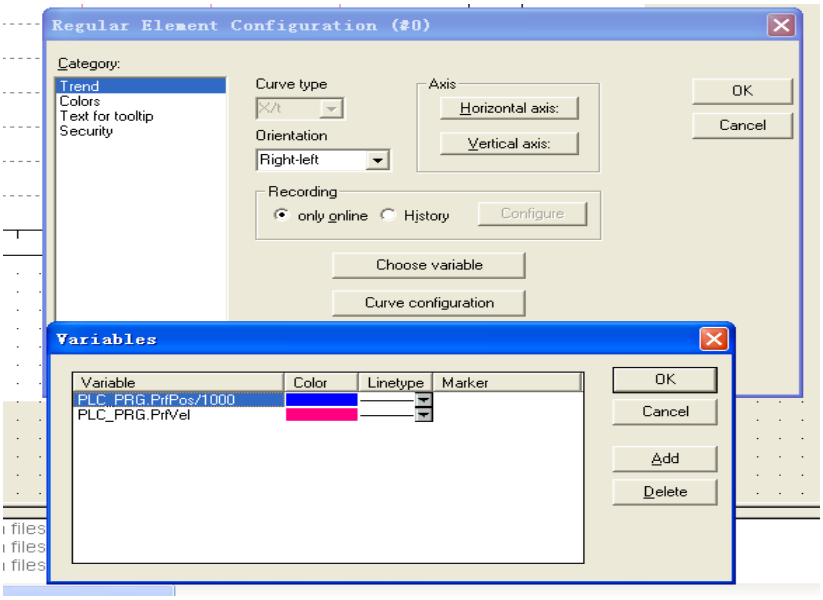


图 3-11

点Choose variable，按快捷键“F2”，打开输入助手，选择程序中与之相对应的变量，单击“OK”完成。如图3-12

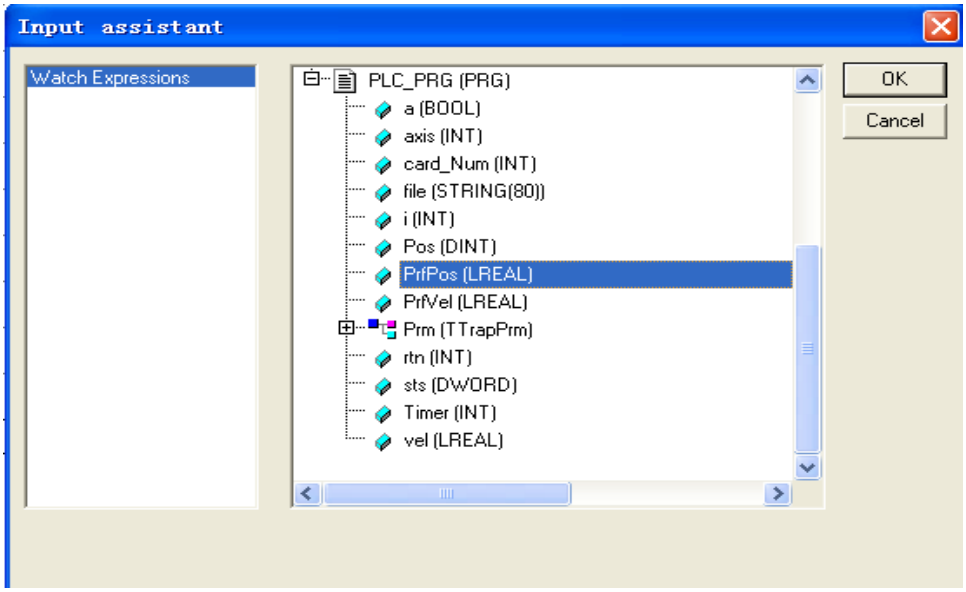


图3-12

同样的操作也可继续添加对应的变量，单击“OK”完成。

Otostudio 软件支持：当程序中有多个可视化界面时，默认的主界面的名称为“PLC_VISU”。

3.3.6 任务配置

1）库文件导入之后，在相同的选项卡中，双击“任务配置”，如图3-13，右键单击选择“添加任务”。

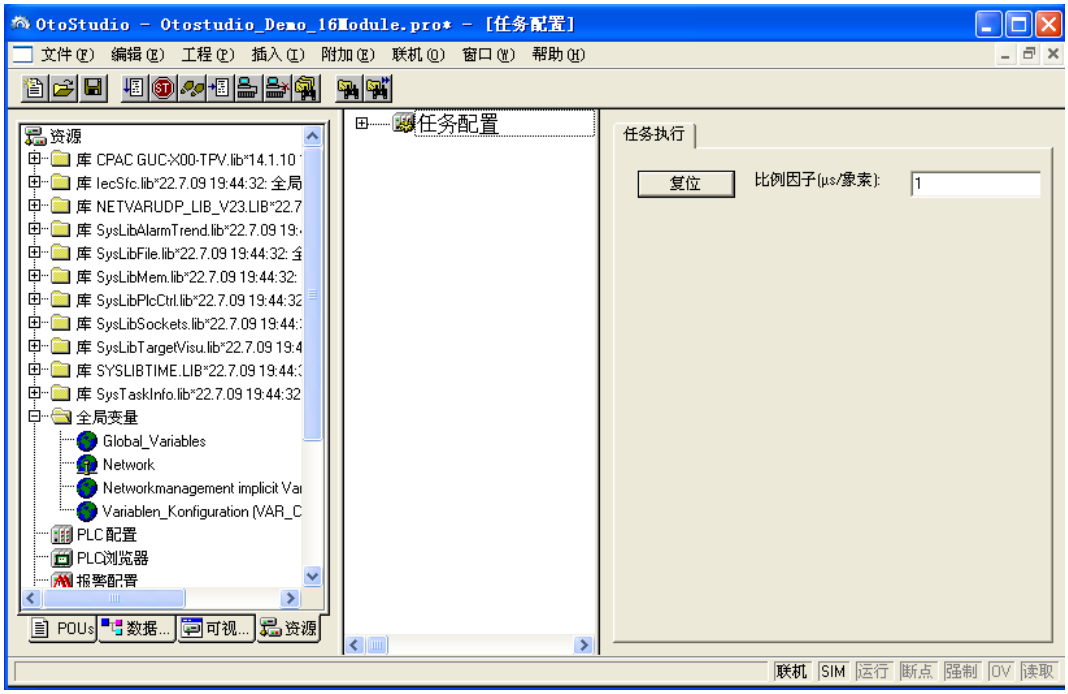


图 3-13

2） “New Task”的设置如图3-14 所示。

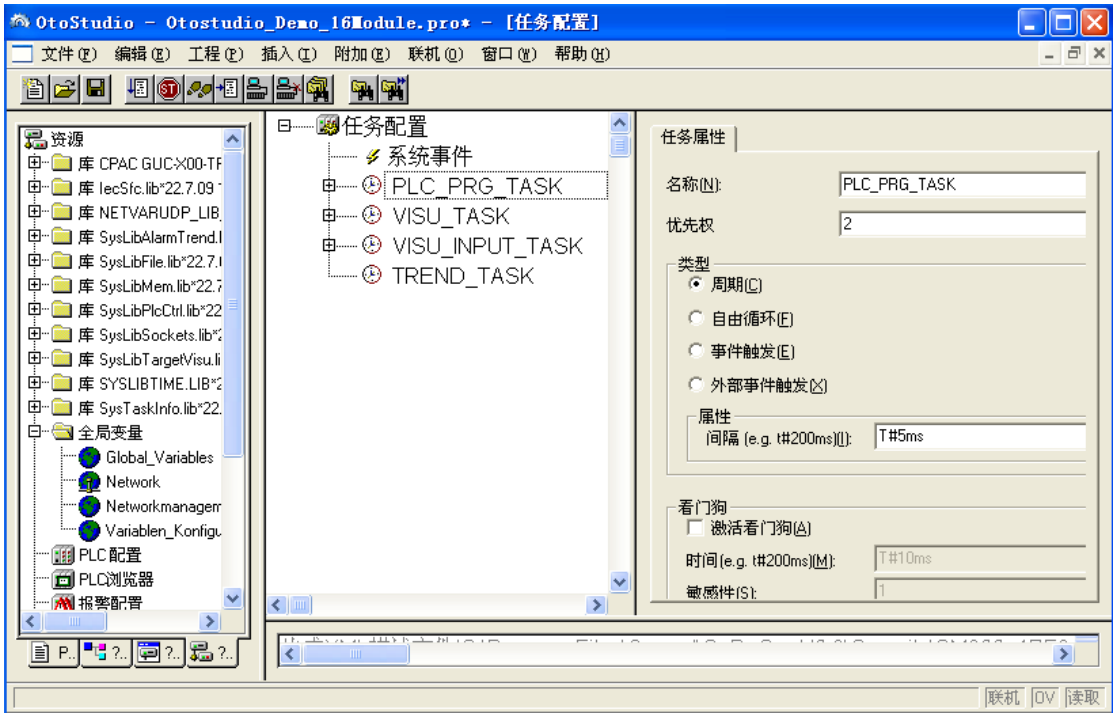


图 3-14

3）右键单击“添加程序调用”添加程序，如图3-15。

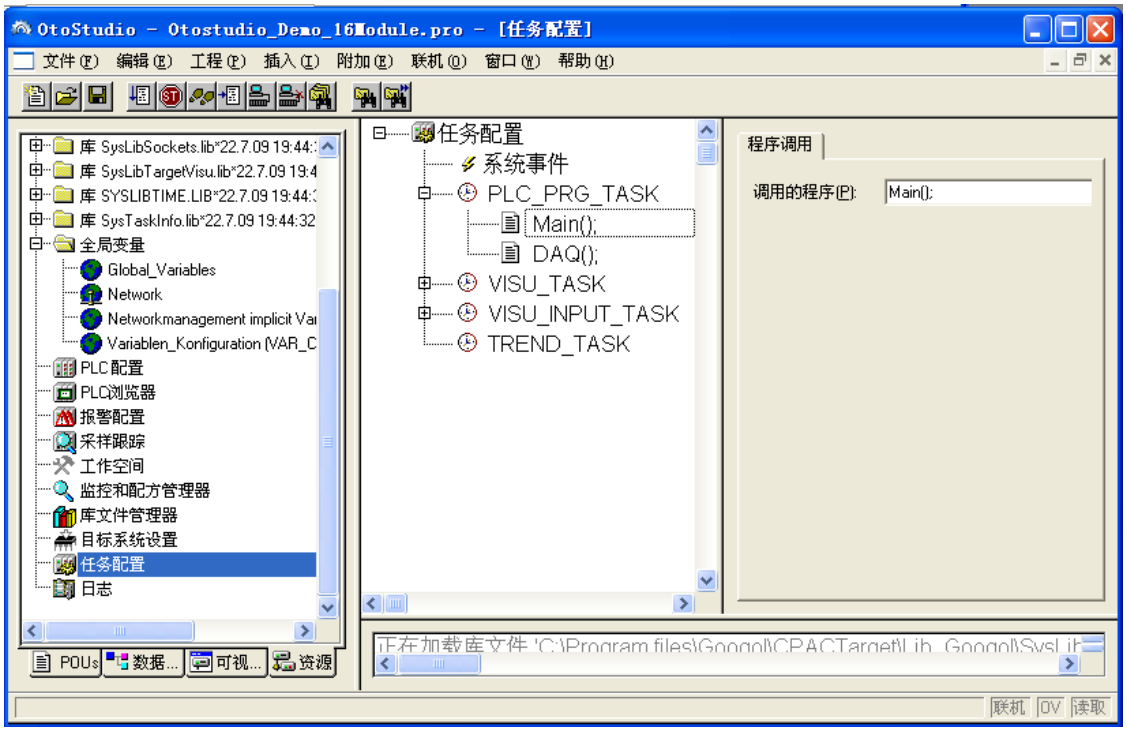


图 3-15

4） 单击“…”弹出如图3-16 所示的窗口，选择程序，单击“确定”即可。

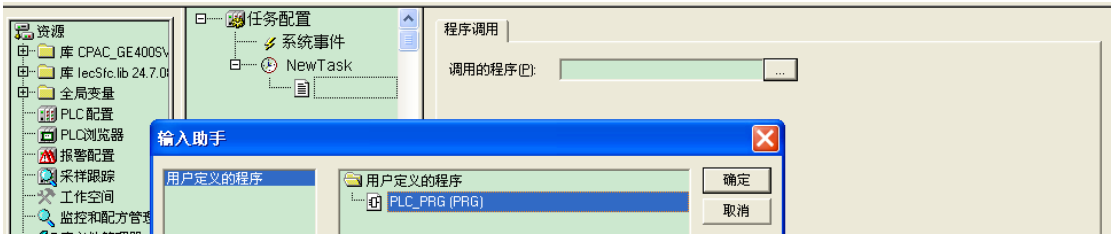


图 3-16

OtoStudio 在默认情况下，会自动分配一个主任务 PLC_PRG_TASK，并且默认认为程序 PLC_PRG 为主程序。

3.3.7 程序编译

程序编写完成后，须对程序进行编译，在“工程”中，单击“编译生成”，或按快捷键“F11”，如图3-17所示。

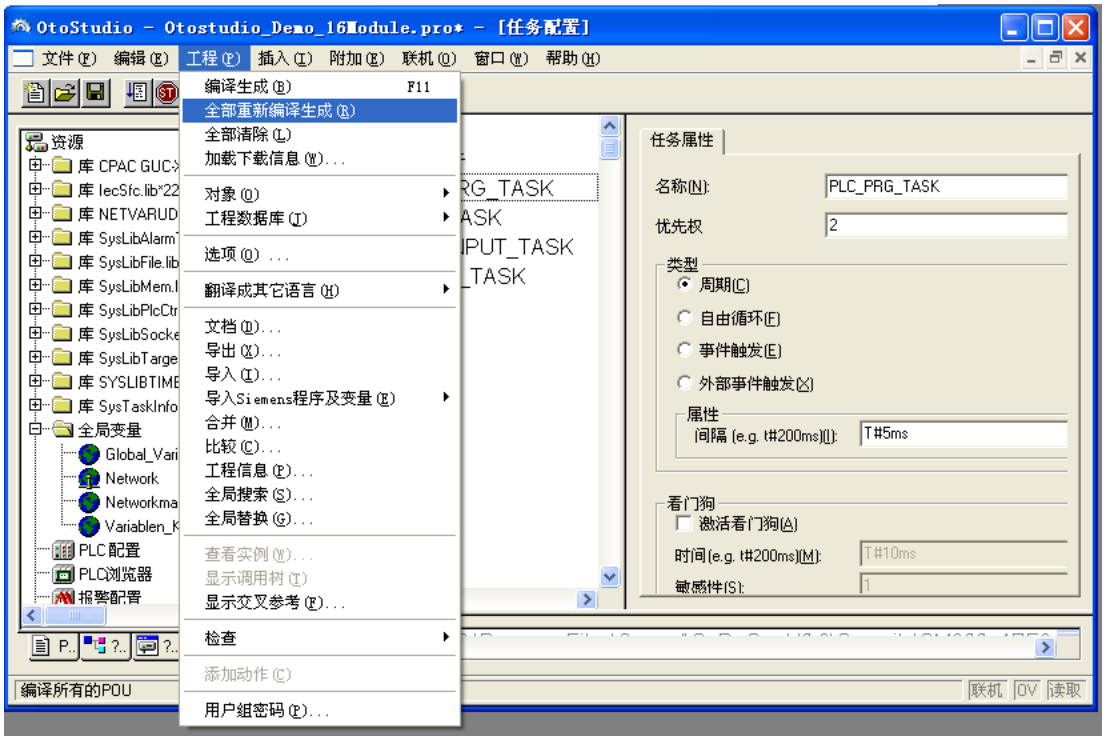
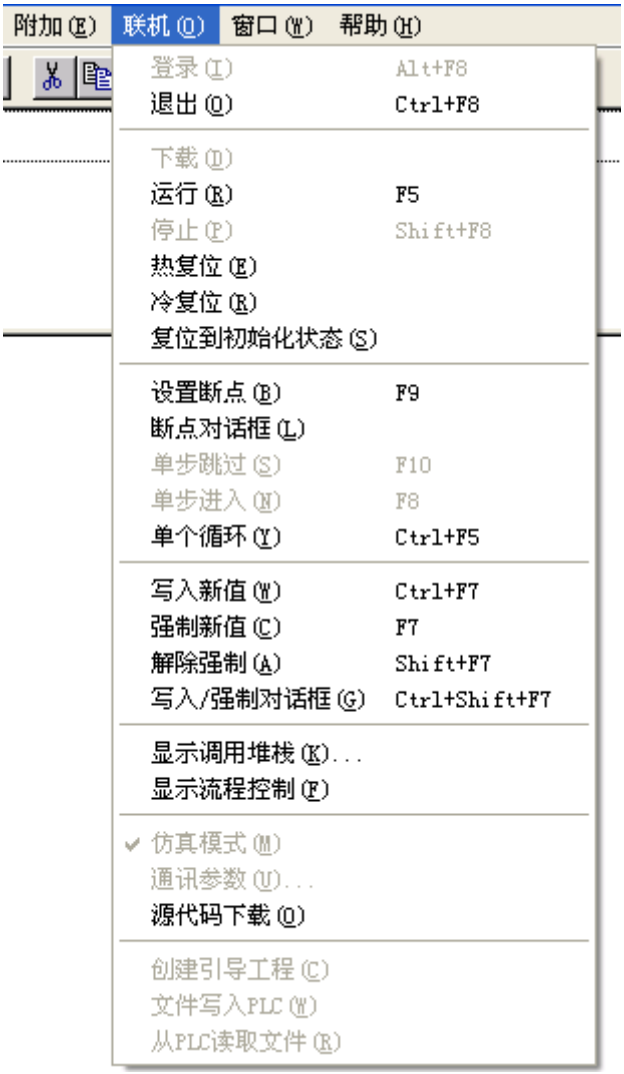


图 3-17

3.3.8 程序调试

程序编译完成后，选择“联机”中的“仿真模式”，然后“登陆”，单击“联机”中的“运行”，或按快捷键“F5”，使程序运行，即可通过如前所述的模拟界面进行调试。



其中F9是设置断点，如图3-18，F8和F11是单步跳入和单步跳出

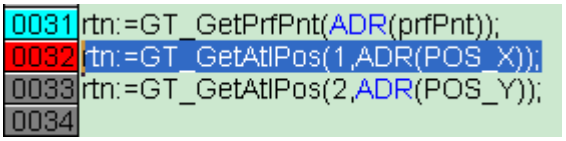


图 3-18

3.3.9 打开 CPAC-GRT 实时内核并设置通讯参数

当仿真程序通过后，进行程序下载运行：将计算机与控制器用点对点网线连接，系统默认为点对点网络传输模式。当控制器上电后，GoogolRuntime实时内核程序会自动运行。如图3-19所示，Start,Reset和Shutdown分别控制该实时内核程序的运行，复位，和关闭。注意默认是点对点网络传输模式下IP是：192.168.0.2，当切换到局域网传输模式下，IP地址自动获取，在对话框的

TCP/IP-Settings中，可以看到当前控制器的IP地址，如：192.168.0.68

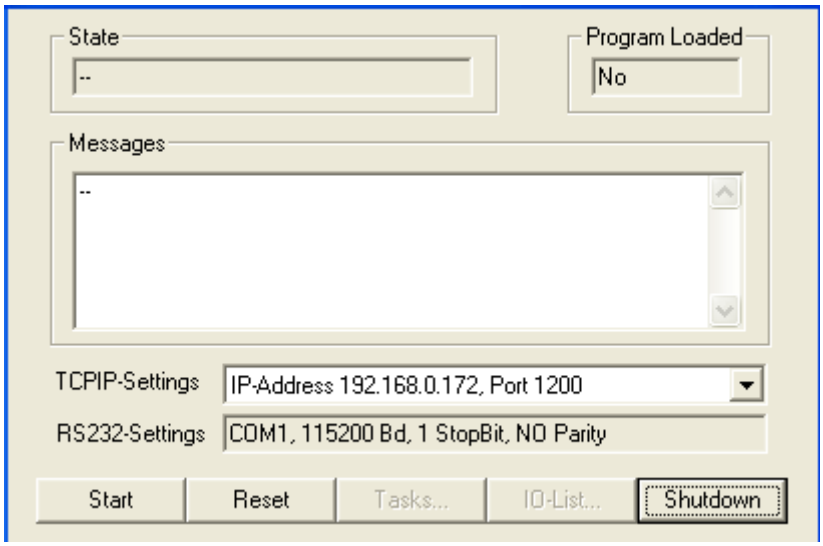


图 3-19

3.3.10 程序下载

1）在“联机”中取消“仿真模式”，然hoou选择“通讯参数”，弹出图3-20 所示的窗口。

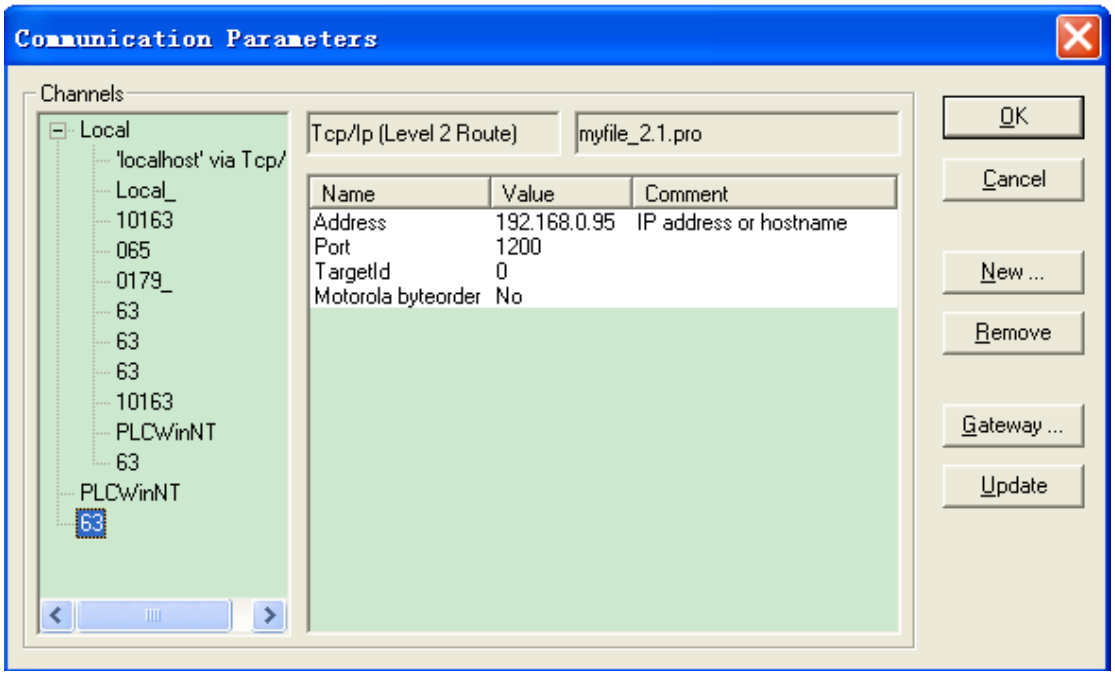


图 3-20

2）单击“New…”，弹出图3-21 所示的窗口，选择“TCP/IP”，单击“OK”

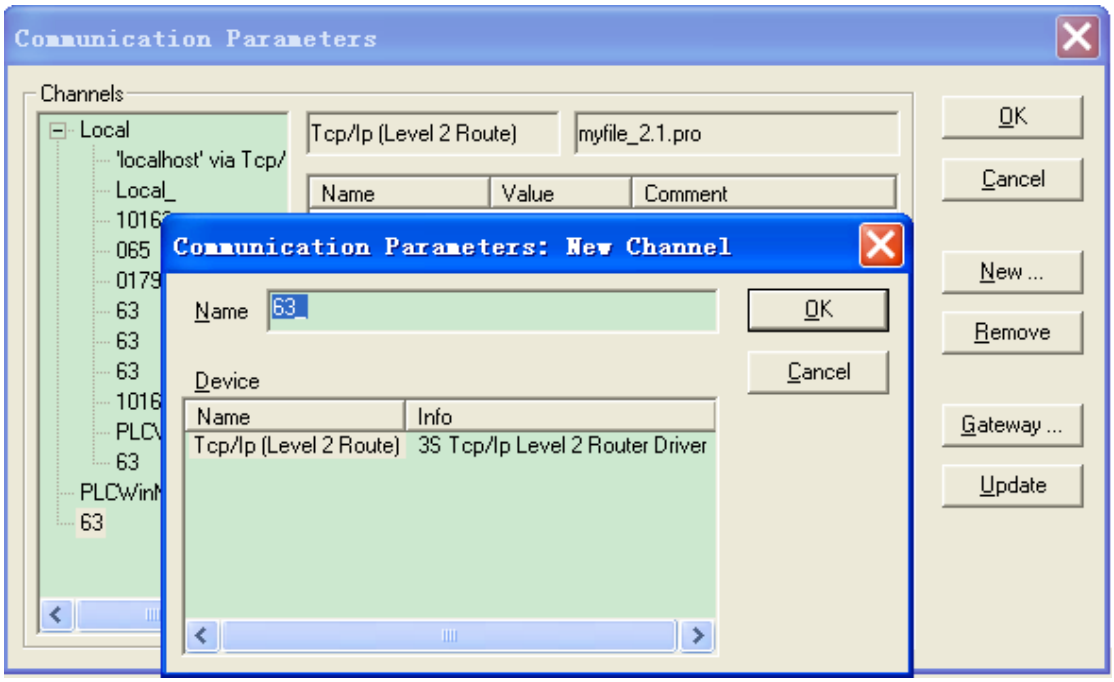


图3-21

3）在图3-22 所示的窗口中，设置“Address”值后回车,使其与控制器的IP地址相同，单击“OK”完成通讯参数设置。

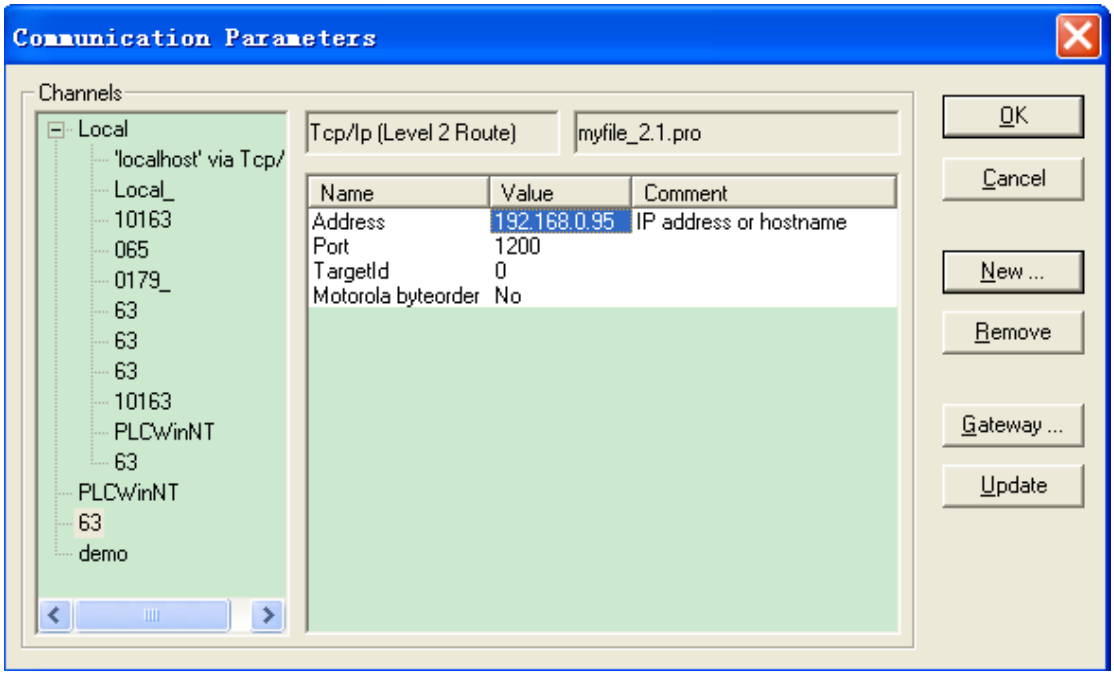


图 3-22

4）在“联机”中选择“登入”或者Alt+F8快捷键，进行程序下载。如图3-23

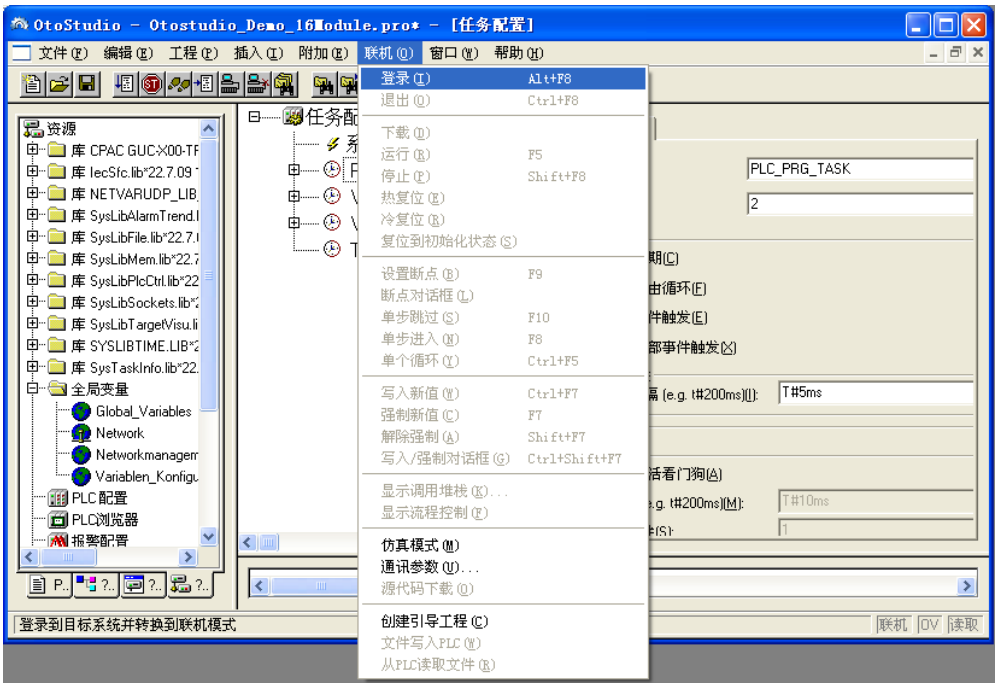


图3-23

3.3.11 目标人机界面

如果用户下位机接了 VGA/LVDS 显示器，希望把设计好的人机界面显示在下位显示运行，需要在“联机”程序前，在“目标系统设置”里选择“可视化”，可以修改人机界面的尺寸，默认为 640x480 与控制其默认配置得分辨率一致，如图 3-24 所示。

图 3-24

如需要修改，请同时修改控制其得分辨率，控制其的分辨率修改方法如下：在 Hard disk\Autoexec.bat 中，将对应的分辨率的值修改。同时修改 BIOS 中的分辨率，重启生效。

3.4 用户编程实例

3.4.1 交通信号灯控制单元

现在让我们编写一个简单的控制交通信号单元的小程序，用它来控制十字路口的交通信号灯，和现实中情况一样，红/绿信号灯交替作用，绿灯通过黄色警示灯过渡到红色。

在这个例子中，我们将使用标准的 IEC61131-3 标准化语言来显示时间依赖程序是怎样工作的，怎样能在 Otostudio 帮助下编辑不同的标准化语言，并且熟悉 Otostudio 仿真的情况下很容易的来连接它们。

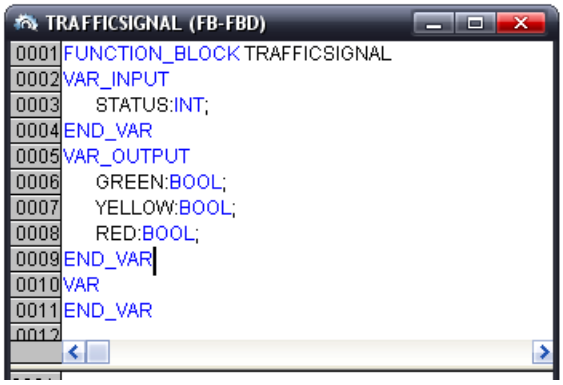
3.4.1.1 创建 POU

打开 Otostudio 并选择“文件”“新建”，在出现的对话框中，最先的 POU 已经给予默认的名字 PLC_PRG，保持这个名字不变，这个 POU 定义为一个程序，每个工程文件都需要一个这样名字的程序。在这个例子中，我们选择 CFC 作为这个 POU 的编程语言。

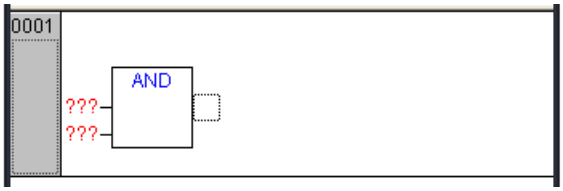
现在创建三个对象，右键点击 POUs 列表空白处，选择用命令“添加对象”，选择“功能块”，“SFC”语言，POU 名为“SEQUENCE”，同样的步骤建立 POU“WAIT”，IL 语言的功能块；“TRAFFICSIGNAL”，FBD 语言的功能块。

3.4.1.2 POU TRAFFICSIGNAL

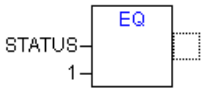
通过该 POU，我们可以通过状态变量来控制信号灯的状态。在声明编辑器中定义输入变量 STATUS 为整型变量（在关键字 VAR_INPUT 和 VAR_OUTPUT 之间）。STATUS 有三个可能的状态，分别对应交通信号状态中绿、红、黄。对应三个输出，在声明编辑器中，这些变量也应该被声明。TRAFFICSIGNAL 的声明部分如下：



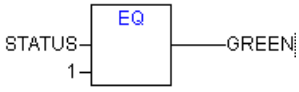
TRAFFICSIGNAL 由输入 STATUS 的值来决定输出变量，先进入 POU 的主体，单击第一个网络图左边的区域（编号为 0001 的灰色区域）。现在选中了第一个网络，选择菜单项“插入”“框”（快捷键 Ctrl+B）在第一个网络插入一个具有两个输入端和操作符号 AND 的方框：



单击文本 AND，显示选中状态并改变值为 EQ，然后分别选择两个输入端，并赋予它们值“STATUS”和“1”。

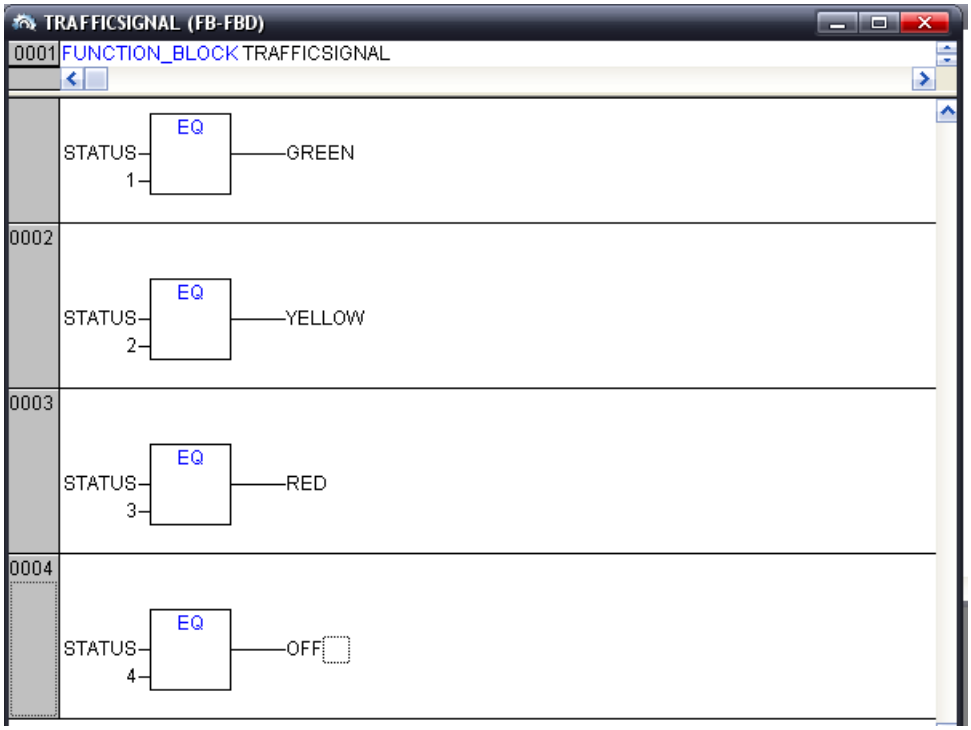


在方框 EQ 的后面上单击，选中 EQ 操作的输出端，选择“插入”“赋值”（快捷键 Ctrl+A），标记输出量为 GREEN，就得到以下结构图。



该结构的作用是对输入 STATUS 和 1 进行比较，结果赋给 GREEN。完成之后右键点击网络中的空白处，选择“网络（插入在当前网络之后）”（快捷键 Ctrl+T）。这时出现了编号 0002 的网络，按照相同的步骤，建立通过 STATUS 和 2、3 比较，输出变量 YELLOW 和 RED 的网络。建立输出变量为 OFF 的网络时，在弹出的对话框中选择 VAR_GLOBAL。

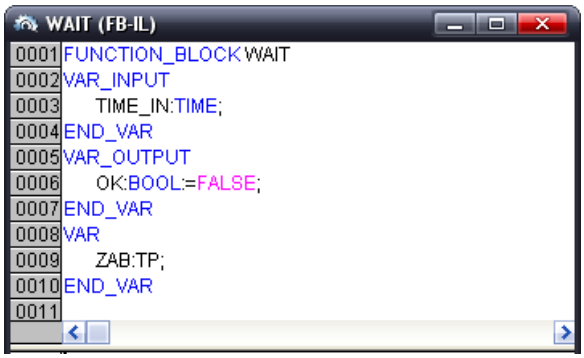
变量 OFF 作为控制信号灯停止工作的信号，变量 STATUS 作为 INT 类型，取值 1、2、3、4 对应绿、黄、红和停止信号。最终网络应如下图所示。



3.4.1.3 POU WAIT

为了使用定时器 WAIT，我们需要 POU 中的一个标准库文件。打开“资源窗口”“库文件管理器”，右键点击库文件管理器的库文件列表空白处，选择“添加库文件”，选择“Standard.lib”。这个 POU 用来作为一个计时器来控制每一个 TRAFFICSIGNAL 状态的时间长短。我们的 POU 接收一个 TIME 类型的时间变量作为输入变量，输出是初始值为 FALSE 的布尔型变量 OK，变量 ZAB 作为时钟发生器，声明为 PT 类型。

WAIT 的声明部分如下：



TP 包括两个输入端 IN 和 PT，两个输出 Q 和 ET，IN 是触发信号，PT 是持续时间。当

IN 输入为 FALSE，ET 输出 0，Q 输出 FALSE。当 IN 输入 TRUE，计时器开始以毫秒为单位计时，当 ET 达到 PT，计时器停止计时，即在接收到 IN 的信号后，计时器会保持输出 TRUE 状态，持续时间为 PT，当计时达到 PT，计时器将输出 FALSE。详见 Standard.lib 中 POU 介绍。

WAIT 的定义部分如下：

```
0001    LD      ZAB.Q
0002    JMPC    mark
0003
0004    CAL         ZAB(IN:=FALSE)
0005    LD      TIME_IN
0006    ST      ZAB.PT
0007    CAL         ZAB(IN:=TRUE)
0008    JMP         END
0009
0010mark:
0011    CAL         ZAB
0012end:
0013    LDN         ZAB.Q
0014    ST      OK
0015    RET
```

首先检查 Q 的值是否为 TRUE（即使已经开始计数），如果是，我们不改变 ZAB 的输入并且在无输入的情况下调用 ZAB（为了检查计时是否结束）；否则我们设置变量 IN 为 FALSE，这样 ET 为 0，Q 为 FALSE。所有的值都设置为期望的初始状态。现在我们把变量 TIME 赋给 PT，并调用 ZAB，IN 输入 TRUE。在 POU ZAB 中开始计时直到它达到 TIME 的时间，随后 Q 被设置为 FALSE。Q 的值取反保存在 OK 变量中，Q 为 FALSE，OK 即为 TRUE，计时器在这个点上结束。

3.4.1.4 POU SEQUENCE

和所有的POU一样，我们要先声明变量。如下图所示：

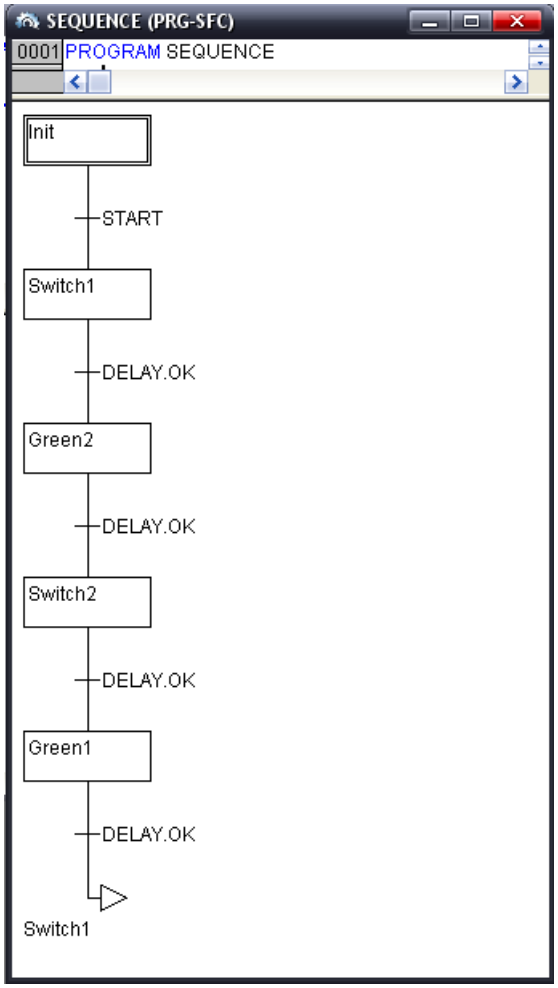
SEQUENCE (PRG-SFC)

```
0001PROGRAM SEQUENCE
0002VAR_INPUT
0003    START:BOOL;
0004END_VAR
0005VAR_OUTPUT
0006    TRAFFICSIGNAL1:INT;
0007    TRAFFICSIGNAL2:INT;
0008END_VAR
0009VAR
0010    COUNTER:INT;
0011    DELAY:WAIT;
0012END_VAR
0013
```

在一个 SFC 语言描述的 POU 中，图表一般包含一个动作“Init”和一个伴随转变“Trans0”

和返回 Init 的跳转。在我们编写各个动作和转变之前，我们先决定一下图表的结构。我们需要为每个 TRAFFICSIGNAL 状态分配一个步，选中标志 Trans0 并选择“插入”“步转换(插入在当前行后)”来插入步，重复这个动作来插入三个步。单击步或转变名字，可以对其进行重命名。命名 Init 之后的第一个转变为 START，其他的转变为“DELAY.ok”。当 START 的值为 TRUE 并且其它所有开关通过 OK 中的 DELAY 都输出 TRUE 时，第一个变换开关接通，例如，当设定的时间段结束。

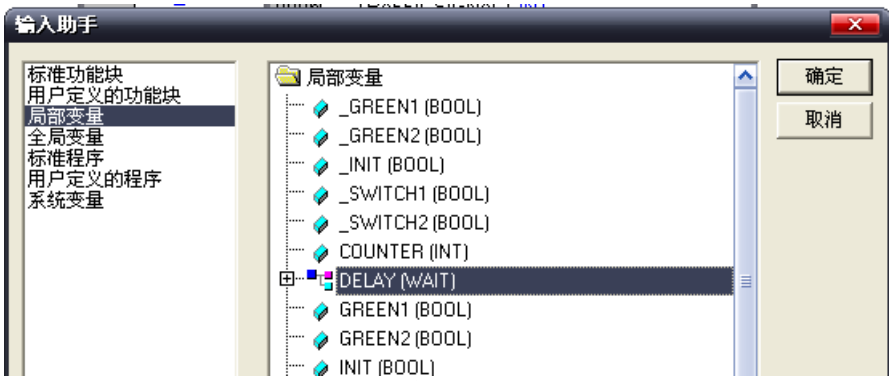
从上到下依次命名步为 Switch1，Green2，Switch2，Green1。只有初始化过程保留它的名字，“Switch”应当包括一个黄色的状态，在 Green1，TRAFFICSIGNAL1 将变为绿色，在 Green2，TRAFFICSIGNAL2 将变为绿灯。最后在开关 Switch1 后返回到初始化的值，如图所示：



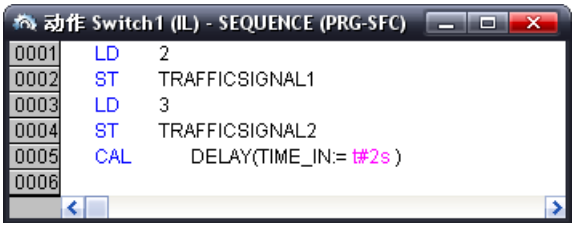
在建立好流程图之后，双击一个步上，打开动作的对话框，可以通过不同语言来对步进行定义。我们使用IL来定义初始化步和扩展的4个步：

Init：信号灯1为红灯，信号灯2为绿灯

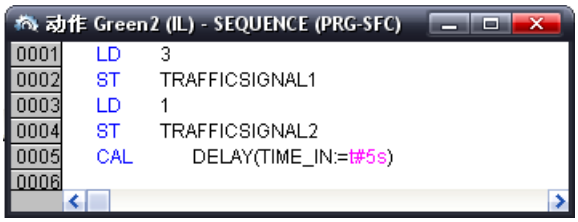
Switch1：信号灯1过渡黄灯，添加延迟2秒。第5行，输入CAL，空格，按F2输入帮助，选择“局部变量”“DELAY”，点击确定，如下图。



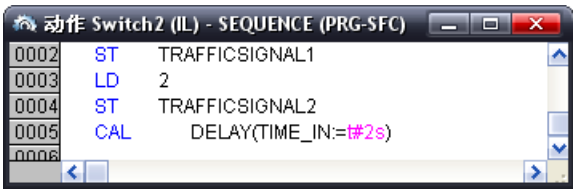
在“TIME_IN:=”后输入“t#2s”，步中不涉及输出，删除后面的“OK=>”。



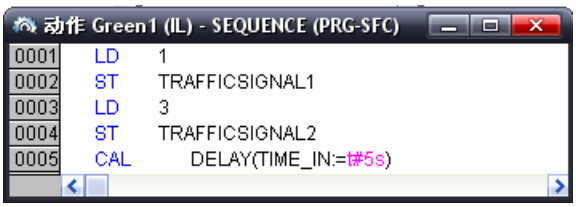
同样的步骤，定义Green2：



Switch2：

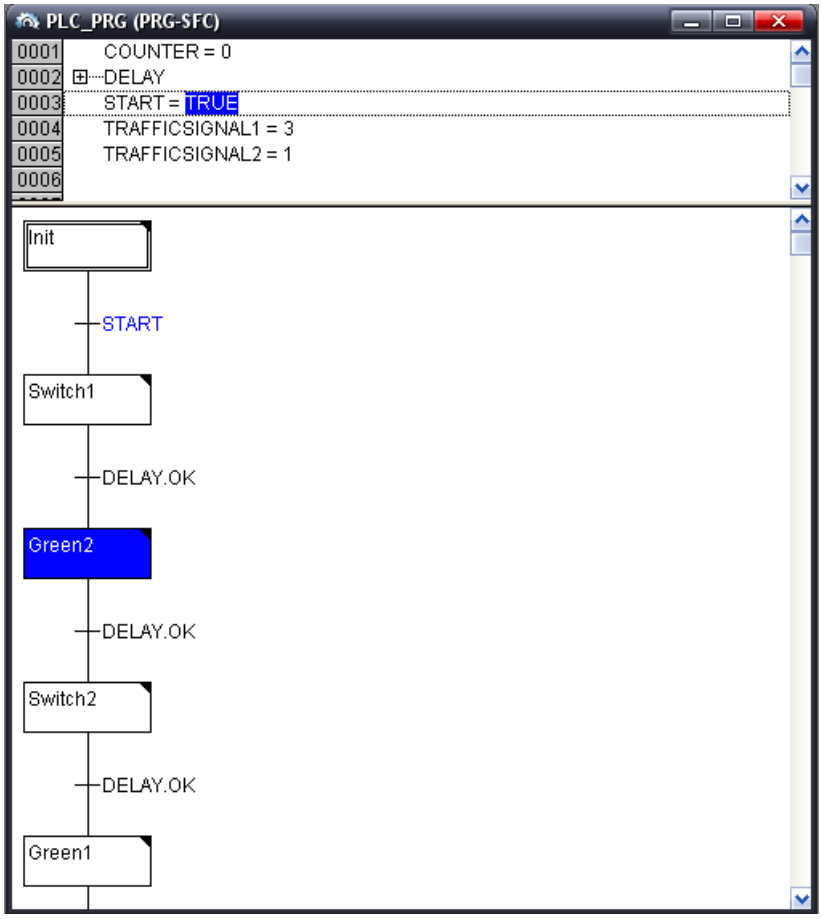


Green1：



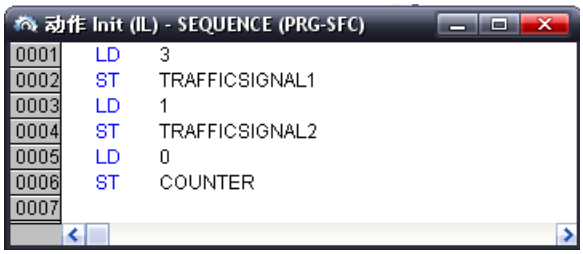
程序的展开部分已经完成，现在可以在仿真模式来测试POU SEQUENCE了（如果要在此测试，须将SEQUENCE重命名为PLC_PRG，因为一个工程需要一个主程序入口）。通过菜

单“工程”“编译生成” 编译工程，在信息窗口中应该得到0个错误和0个警告。现在检查“联机”“仿真模式”是否激活，用命令“联机”“登录”进入仿真，用“联机“运行”来启动程序，程序启动后，必须把变量START设置为TRUE。程序启动后，在变量窗口双击START变量后面的值，这将给它赋以TRUE（准备值），现在选择命令“联机”“写入新值”来把新值赋给变量。在顺序功能图中 START 将以蓝色显示，当前激活步中的正在处理的步将标记为蓝色，如下图：



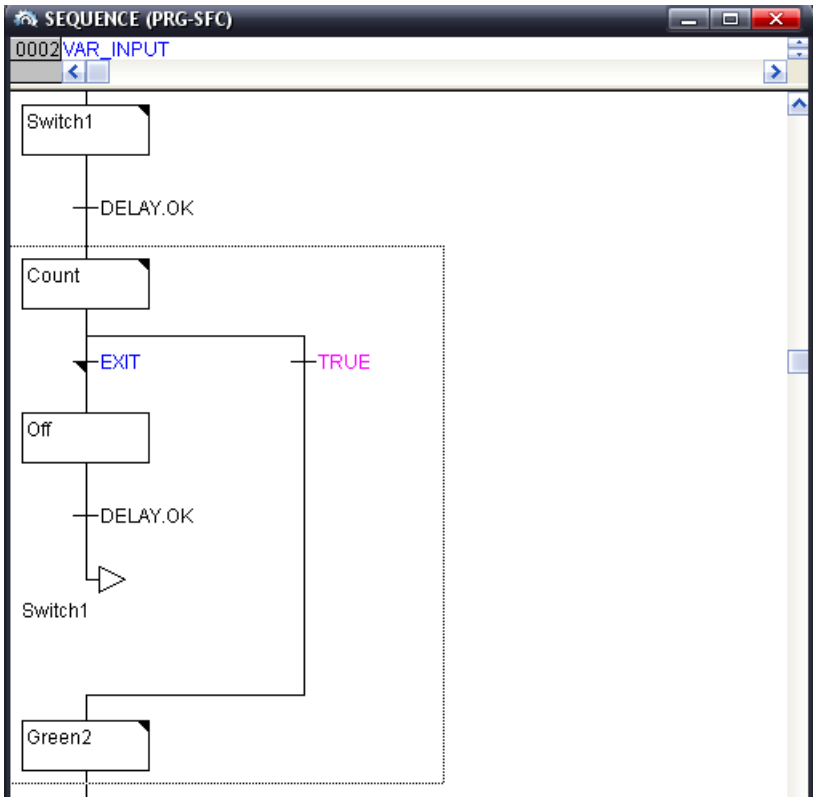
当你完成了这些中间的测试，通过使用命令“联机” “退出”来退出模拟模式,继续编写程序。为了保证我们的图表有至少一个可选择的分支，并且信号灯可以关闭，我们现在在程序中编写一个计数器，在一定数的TRAFFICSIGNAL循环之后，关闭交通灯。

首先我们需要一个新的整型变量COUNTER，在SEQUENCE的声明部分定义这个变量，并初始化使它为零，如下图所示。



现在选择Switch1 后的 转变条件并插入一个步和一个转变，选择结果变换并在它的右边插入一个可供选择的分支，在左边的变换条件之后插入一个步和一个转变。在新转变条件

之后，在Switch1 之后插入一个跳转。对新部分命名如下：上面两个新步命名为“Count”，下面的叫“Off”，转变名为 EXIT、TRUE 和DELAY.OK 。新部分应该象下面用虚线框标注的部分一样。



为新加入的步和转变定义，下面是Count的定义：

EXIT转变的条件：计数5次，关闭信号灯

Off的定义：把状态4送给信号灯，COUNTER清零，进行一个10秒的等待

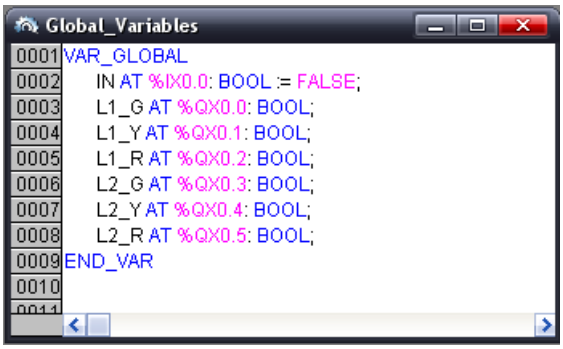
在我们假定在5个交通信号循环之后信号灯停止，经过10秒的延迟，信号灯又再次开始

工作，整个过程不断的重复。

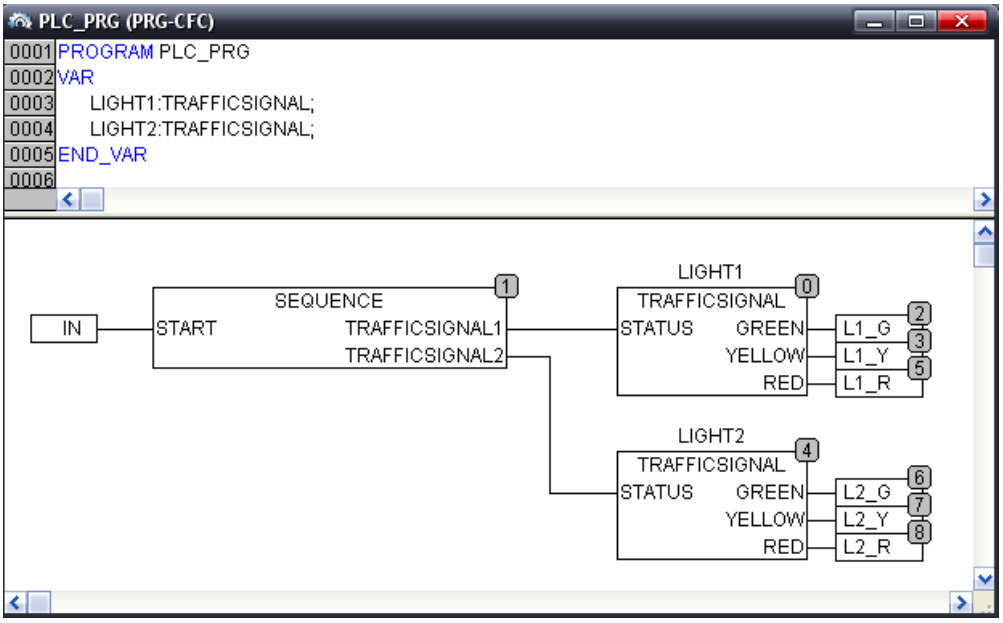
3.4.1.5 PLC_PRG

我们已经在模块 SEQUENCE 中为两套交通灯定义和关联了各个阶段的时间序列，但是，实际的交通灯系统是一个总线系统的一个模块，例如CAN BUS。我们必须在模块PLC_PRG中可利用输入和输出变量，我们希望用ON开关上打开交通灯系统，并且为SEQUENCE的各个步分配给6个灯相应的“信号命令”，现在为这6个输出和1个输入变量定义布尔类型，在编辑器中编写程序之前，为它们分配值，同时分配相应的IEC地址。下面步是定义Phases类型变量LIGHT1和LIGHT2。LIGHT1 和LIGHT2 的声明

这些为模块SEQUENCE的各个步传递6个灯的BOOL值给6个输出变量。我们在全局变量窗中定义：IN表示开关控制，L1_G、L1_Y、L1_B表示信号灯的绿、黄、红，L2表示第二组信号灯。



AT跟在变量名后面，%号符号开始是IEC地址。I代表输入，Q代表输出。X（在例子中使用）代表按位操作。各个模块的位数用0.0（0.1，0.2 等等）表示。在这个例子中，我们不需要配置控制器，因为它依赖于你计算机中的可利用的目标包，更多的信息请查看PLC配置部分。接下来进入编辑窗口，选择连续功能图表编辑器，从菜单栏下面的工具栏中可以找到一个CFC的工具符号（查看连续功能图表编辑器）。



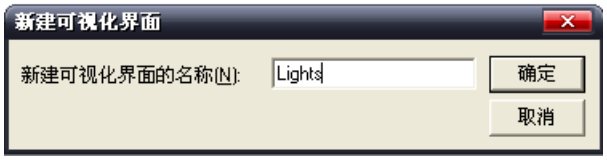
插入输入，选择框内，用输入助手，选择全局变量IN；插入框，点击AND，用输入助手选择用户定义的程序，选择SEQUENCE；同样的方法，插入用户定义的功能块TRAFFICSIGNAL和输出，并把输出选择为6个全局变量。

现在模拟程序中测试这个程序。编译（如果出现错误“无效地址”，请检查是否配置了模拟仿真系统，或者可以在资源窗口，目标系统配置项中点击“一般设置”选项卡，点击“不进行地址检查”）并加载，通过“联机”“运行”来启动程序，然后设置变量ON为TRUE。在CFC编辑器中的输入框的条目“IN”上双击，变量将设置为TRUE。然后按Ctrl＋F7或命令“联机”“写入新值”来设置值。现在SEQUENCE中的变量START（在程序的开始阶段我们手动设置为TRUE）从变量IN上获得到值。这用来运行交通灯循环。PLC_PRG 转换为一个监视窗口，在声明编辑器中的加号上双击，变量将顺序显示，你能看到变量的各自的值。

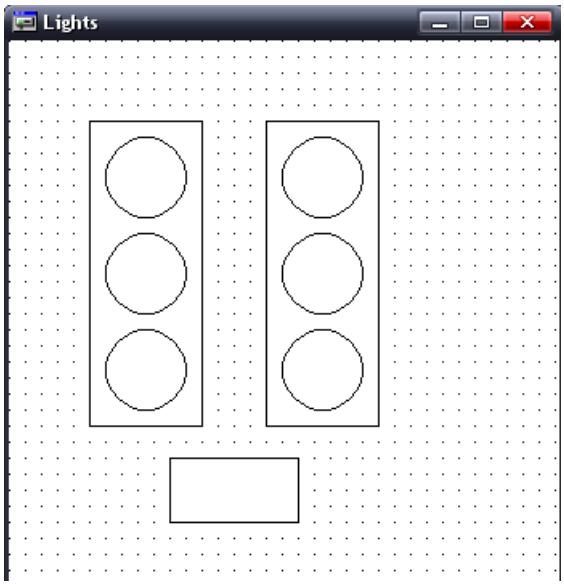
3.4.2 可视化交通信号单元

通过Otostudio的可视化。你可以很容易的把各式各样的程序应用到实际中。我们现在通过两个交通灯的模型和一个控制开关，反映出4个POU中数据的变化。

首先点击左下“可视化界面”选项卡，在空白处右键点击，选择“添加对象”，命名为Lights。



界面建立后，使用上方可视化工具栏中的工具，绘制出如下效果：

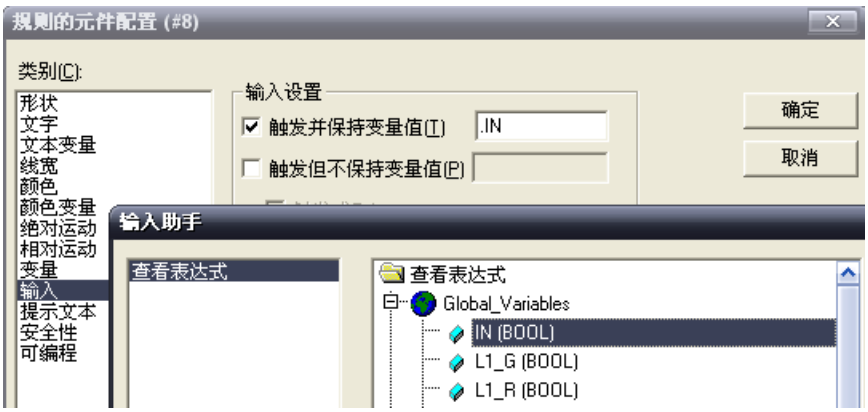


双击左上第一个圆形，出现规则原件配置对话框，选取颜色项，选定内部颜色为黑色，报警颜色为绿色，即在无信号的时候保持黑色，有输入情况下为报警色。点击变量项，在“改变颜色”栏使用输入助手，选取变量 L1_G，如图：

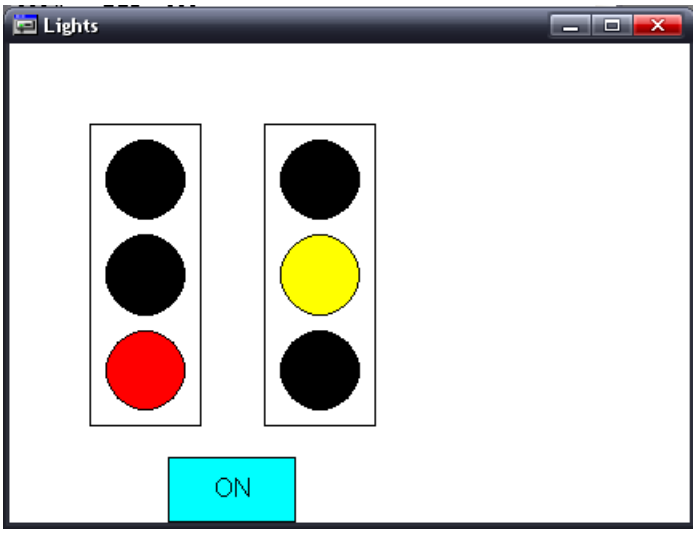
同样的方法，设置左边第二灯为黄灯，变量为 L1_Y，第三灯为 L1_R；右侧信号灯和左边方法一致，变量替换为 L2。

配置下方的控制开关，选择文字选项，

为了使按键可以触发 IN 变量，选择输入项，选择“触发但不保持变量值”，输入助手选择全局变量 IN，点击确定。



这样，一个简单的可视化界面就完成了，可以登录，运行，观察变量，结果如下：



第四章　指令系统

4.1 操作块 Operator

1、运算符：

ADD	（加）
MUL	（乘）
SUB	（减）
DIV	（除）
MOD	（求余）
MOVE	（赋值）
INDEXOF	（查找POU索引）
SIZEOF	（获取长度）

ADD

相加变量的类型: BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL 和 LREAL.

两个时间量也可以相加得出另一个时间量。

(例如： t#45s + t#50s = t#1m35s)

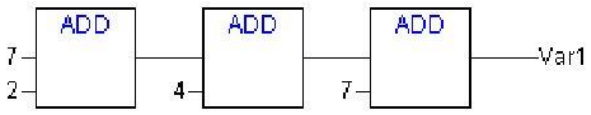
IL 例子:

```
LD 7
ADD 2,4,7
ST Var1
```

ST 例子:

```
var1 := 7+2+4+7;
```

FBD 例子:



MUL

相乘变量的类型: BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL 和 LREAL.

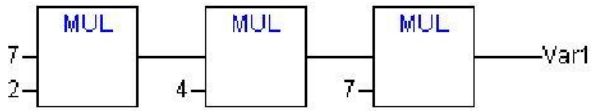
IL 例子:

```
LD 7
MUL 2,4,7
ST  Var1
```

ST 例子:

```
var1 := 7*2*4*7;
```

FBD 例子:



SUB

相减变量的类型: BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL 和 LREAL.

一个时间变量与另一个时间变量相减得出第三个时间类型变量。注意负时间值无意义。

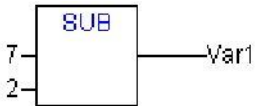
IL 例子:

```
LD 7
SUB 2
ST  Var1
```

ST 例子:

```
var1 := 7-2;
```

FBD 例子:



DIV

相除变量的类型: BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL 和 LREAL.

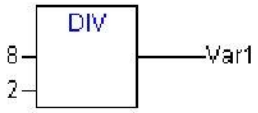
IL 例子:

```
LD  8
DIV 2
ST Var1 (*  结果是  4 *)
```

ST 例子:

```
var1 := 8/2;
```

FBD 例子:



MOD

模块分割的两个变量类型: BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT. 此功能的结果则是此项分割的余数，其结果将是整个数字.

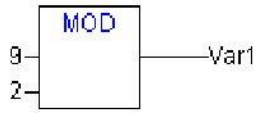
IL 例子:

```
LD 9
MOD 2
ST   Var1 (* Result is 1 *)
```

ST 例子:

```
var1 := 9 MOD 2;
```

FBD 例子:

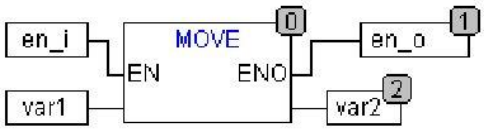


MOVE

一个适当的类型从一个变量分配到另一个变量中.由于 Move 在图表编辑 LD，CFC 的框盒子里, 此处(未加锁的) EN/EN0 功能 也可以用于变量分配中，在 FBD 编辑器里这个则是不可能的。

CFC 例子带 EN/EN0 功能:

只有当 en_i 是 TRUE, var1 将赋值给 var2.



IL 例子:

```
LD ivar1
MOVE
ST ivar2 (* Result: ivar2 gets assigned value of ivar1 *)
```

(! 你会得到相同的结果:

```
LD ivar1
ST ivar2 )
```

ST 例子:

```
ivar2 := MOVE(ivar1);
```

(! 你会得到相同的结果: ivar2 := ivar1;)

INDEXOF

此功能未列入 IEC61131-3 标准.执行这项功能用来找到 POU 中的内部索引。

ST 例子:

```
var1 := INDEXOF(POU2);
```

SIZEOF

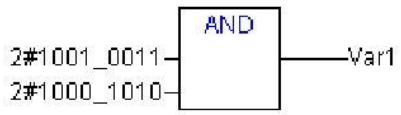
此功能未列入 IEC61131-3 标准.执行此功能块可以判断所给变量要求达到的字符数。
IL 例子:
arr1:ARRAY[0..4] OF INT;
Var1 INT
LD arr1
SIZEOF
ST Var1 (* 结果是 10 *)
ST 例子:
var1 := SIZEOF(arr1);

2、逻辑操作：

AND （与）
OR （或）
XOR （异或）
NOT （非）

AND

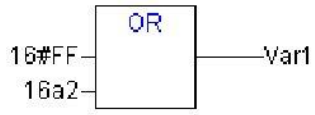
位操作符的 AND。其类型应该属于 BOOL, BYTE, WORD 或 DWORD.
IL 例子:
Var1 BYTE
LD 2#1001_0011
AND 2#1000_1010
ST Var1 (* 结果是 2#1000_0010 *)
ST 例子:
var1 := 2#1001_0011 AND 2#1000_1010
FBD 例子:



OR

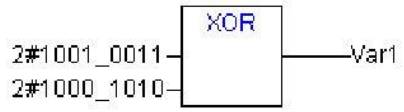
位操作符的 OR。其类型应该属于 BOOL, BYTE, WORD 或 DWORD.
IL 例子:
var1 :BYTE;
LD 2#1001_0011

OR 2#1000_1010
ST　var1 (* 结果是 2#1001_1011 *)
ST 例子:
Var1 := 2#1001_0011 OR 2#1000_1010
FBD 例子:



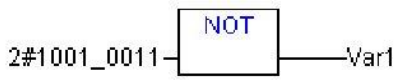
XOR

位操作符的 XOR。其类型应该属于 BOOL, BYTE, WORD 或 DWORD.
注意: 把 XOR 的作用行为视为其延伸形式, 也即如果出现 2 次以上的输入. 输入符号会被成对检查，所得的数据结果也会再次被成对检查核对。 (这与标准相一致,但也许并不是用户所期望的).
IL 例子:
Var1 :BYTE;
LD 2#1001_0011
XOR 2#1000_1010
ST Var1 (* 结果是 2#0001_1001 *)
ST 例子:
Var1 := 2#1001_0011 XOR 2#1000_1010
FBD 例子:



NOT

位操作符的 NOT。其类型应该属于 BOOL, BYTE, WORD 或 DWORD.
IL 例子:
Var1 :BYTE;
LD 2#1001_0011
NOT
ST Var1 (* 结果是 2#0110_1100 *)
ST 例子:
Var1 := NOT 2#1001_0011
FBD:



3、移位操作：

SHL （左移）
SHR （右移）
ROL （循环左移）
ROR （循环右移）

SHL

操作数的位左移 : erg:= SHL (in, n)
in 向左位移 n 个位. 如果 n > 大于数据类型的宽度, BYTE, WORD 和 DWORD 将用 0 来填补. 但是如果使用有符号的数据类型, 如 e.g. INT, 那么此种情况下会执行一次运算位移, 也即它会由最上边的位值占位.
注意: 请注意位的数量, 它是用来作算术计算的, 然而却是以输入变量的数据类型的形式出现的。若输入变量是个常数则被视为最小数据类型。输入变量的数据类型对算术计算毫无影响。
注意: 见如下例子中十六进制位符号，根据输入变量数据类型的不同（BYTE 或 WORD）会得到 erg_byte 和 erg_word 的不同结果，尽管此时的 in_byte 和 in_word 输入变量的值相同。
ST 例子:

```
PROGRAM shl_st
VAR
in_byte : BYTE:=16#45;
in_word : WORD:=16#45;
erg_byte : BYTE;
erg_word : WORD;
n: BYTE :=2;
END_VAR
erg_byte:=SHL(in_byte,n); (* 结果是 16#14 *)
erg_word:=SHL(in_word,n); (* 结果是 16#01141 *)
```

FBD 例子:

IL 例子:

```
LD   16#45
SHL 2
ST erg_byte
```

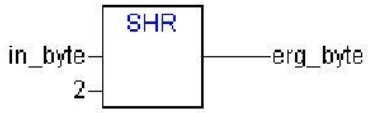
SHR

操作数的位右移 : erg:= SHR (in, n)

in 向右位移 n 个位. 如果 n > 大于数据类型的宽度, BYTE, WORD 和 DWORD 将用 0 来填补. 但是如果使用有符号的数据类型, 如 e.g. INT, 那么此种情况下会执行一次运算位移, 也即它会由最上边的位值占位.
注意: 请注意位的数量, 它是用来作算术计算的, 然而却是以输入变量的数据类型的形式出现的。若输入变量是个常数则被视为最小数据类型。输入变量的数据类型对算术计算毫无影响。
见如下例子中十六进制位符号，根据输入变量数据类型的不同（BYTE 或 WORD），会发现算术运算的结果。
ST 例子:

```
PROGRAM shr_st
VAR
in_byte : BYTE:=16#45;
in_word : WORD:=16#45;
erg_byte : BYTE;
erg_word : WORD;
n: BYTE :=2;
END_VAR
erg_byte:=SHR(in_byte,n); (* 结果是 11 *)
erg_word:=SHR(in_word,n); (* 结果是 0011 *)
```

FBD 例子:



IL 例子:

```
LD   16#45
SHR 2
ST   erg_byt
```

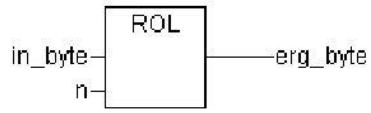
ROL

操作数的位循环左移: erg:= ROL (in, n)
erg, in 和 n 应该为 BYTE, WORD 或 DWORD 类型. in 将向左方向移动一个位，移动 n 次，而距离左边最远的位符，将被再次从右边插入。
注意: 请注意位的数量, 它是用来作算术计算的, 然而却是以输入变量的数据类型的形式出现的。若输入变量是个常数则被视为最小数据类型。输入变量的数据类型对算术计算毫无影响。
见如下例子中十六进制位符号，根据输入变量数据类型的不同（BYTE 或 WORD），会得到 erg_byte 和 erg_word 的不同结果，尽管此时的 in_byte 和 in_word 输入变量的值相同。
ST 例子:

```
PROGRAM rol_st
VAR
in_byte : BYTE:=16#45;
in_word : WORD:=16#45;
erg_byte : BYTE;
```

```
erg_word : WORD;
n: BYTE :=2;
END_VAR
erg_byte:=ROL(in_byte,n); (* 结果是 16#15 *)
erg_word:=ROL(in_word,n); (* 结果是 16#0114 *)
```

FBD 例子:



IL 例子:

```
LD 16#45
ROL 2
ST  erg_byte
```

ROR

操作数的位循环右移: erg = ROR (in, n)

erg, in 和 n 应该为 BYTE, WORD 或 DWORD 类型. in 将向右方向移动一个位，移动 n 次，而距离右边最远的位符，将被再次从左边插入。

注意: 请注意位的数量, 它是用来作算术计算的, 然而却是以输入变量的数据类型的形式出现的。若输入变量是个常数则被视为最小数据类型。输入变量的数据类型对算术计算毫无影响。

见如下例子中十六进制位符号，根据输入变量数据类型的不同（BYTE 或 WORD），会得到 erg_byte 和 erg_word 的不同结果，尽管此时的 in_byte 和 in_word 输入变量的值相同。

ST 例子:

```
PROGRAM ror_st
VAR
in_byte : BYTE:=16#45;
in_word : WORD:=16#45;
erg_byte : BYTE;
erg_word : WORD;
n: BYTE :=2;
END_VAR
erg_byte:=ROR(in_byte,n); (* 结果是 16#51 *)
erg_word:=ROR(in_word,n); (* 结果是 16#4011 *)
```

FBD 例子:

IL 例子:

```
LD 16#45
ROR 2
ST erg_byte
```

4、选择操作：

SEL (选择)
MAX （最大值）
MIN （最小值）
LIMIT （比较选择）：
IN>MAX：OUT:=MAX；
IN<MIN：OUT:=MIN；
MUX （多路选择）：
OUT:=MUX(IN0,…,INk,…INn)；
OUT:=Ink；

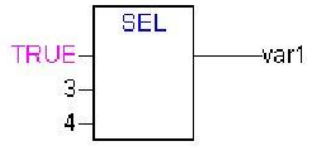
SEL

位选择.
OUT := SEL(G, IN0, IN1) 也就是说:
OUT := IN0 if G=FALSE;
OUT := IN1 if G=TRUE.
IN0, IN1 和 OUT 可为任何变量类型, G 必须是 BOOL 型. 选择结果如果 G 是 FALSE 则为 IN0,如果 G 为 TRUE，结果则为 IN1。

IL 例子:
LD TRUE
SEL 3,4
(* IN0 = 3, IN1 =4 *)
ST Var1 (* 结果是 4 *)
LD FALSE
SEL 3,4
ST Var1 (* 结果是 3 *)

ST 例子:
Var1:=SEL(TRUE,3,4); (* 结果是 4 *)

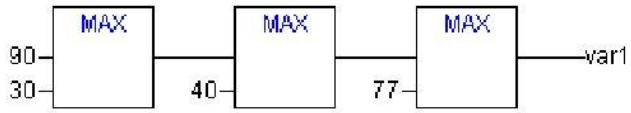
FBD 例子:



注意: 如果 IN1 或 IN2 前出现符号而 IN0 是 TRUE，则此符号不会被处理。

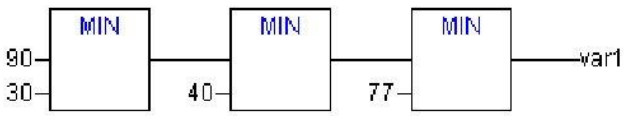
MAX

操作符: 最大作用. 返回两个位当中的最大值。
OUT := MAX(IN0, IN1)
IN0, IN1 和 OUT 可以是任何变量数据类型。
IL 例子:
LD 90
MAX 30
MAY 40
MAX 77
ST Var1 (* 结果是 90 *)
ST 例子:
Var1:=MAX(30,40); (* 结果是 40 *)
Var1:=MAX(40,MAX(90,30)); (* 结果是 90 *)
FBD 例子:



MIN

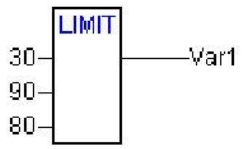
操作符: 最小作用. 返回两个位当中的最小值。
OUT := MIN(IN0, IN1)
IN0, IN1 和 OUT 可以是任何变量数据类型。
IL 例子:
LD 90
MIN 30
MIN 40
MIN 77
ST Var1 (* 结果是 30 *)
ST 例子:
Var1:=MIN(90,30); (* 结果是 30 *);
Var1:=MIN(MIN(90,30),40); (* 结果是 30 *);
FBD 例子:



LIMIT

操作符: 限制

OUT := LIMIT(Min, IN, Max) 也就是:
OUT := MIN (MAX (IN, Min), Max)
结果中 Max 为上限，Min 为下限。如果 IN 值超过上限 MAX，LIMIT 将会返回 Max. 如果 IN 低于 Min, 其结果仍为 Min.
IN 和 OUT 可以是任何变量数据类型。
IL 例子:
LD 90
LIMIT 30,80
ST Var1 (* 结果是 80 *)
ST 例子:
Var1:=LIMIT(30,90,80); (* 结果是 80 *);
FBD 例子:



MUX

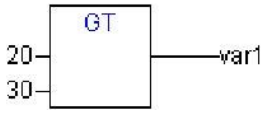
操作符: 多路器
OUT := MUX(K, IN0,...,INn) 也就是:
OUT := INK.
IN0, ...,INn 和 OUT 可以是任何变量类型。 K 必须是 BYTE, WORD, DWORD, SINT, USINT, INT, UINT,
DINT 或 UDINT.在一组数中 MUX 选择 Kth 值。
Example in IL:
LD 0
MUX 30,40,50,60,70,80
ST Var1 (* 结果是 30 *)
Example in ST:
Var1:=MUX(0,30,40,50,60,70,80); (* 结果是 30 *);

5、比较运算：

GT （大于）
LT （小于）
LE （小于等于）
GE （大于等于）
EQ （等于）
NE （不等于）

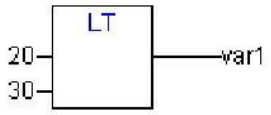
GT

操作符: 大于
当第一个操作数值大于第二个时，布尔算子将会返回 TRUE 值，操作数可以是 BOOL, BYTE, WORD, DWORD,SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME 和 STRING.
IL 例子:
LD 20
GT 30
ST Var1 (* 结果是 FALSE *)
ST 例子:
VAR1 := 20 > 30 > 40 > 50 > 60 > 70;
FBD 例子:



LT

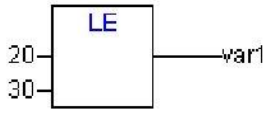
操作符: 小于
当第一个操作数值小于第二个时，布尔算子将会返回 TRUE 值，操作数可以是 BOOL, BYTE, WORD, DWORD,SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME 和 STRING.
IL 例子:
LD 20
LT 30
ST Var1 (* 结果是 TRUE *)
ST 例子:
VAR1 := 20 < 30;
FBD 例子:



LE

操作符: 小于或等于
当第一个操作数值小于或等于第二个时，布尔算子将会返回 TRUE 值，操作数可以是 BOOL, BYTE, WORD,DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME 和 STRING.
IL 例子:
LD 20

LE 30
ST Var1 (* 结果是 TRUE *)
ST 例子:
VAR1 := 20 <= 30;
FBD 例子:



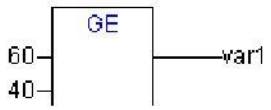
GE

操作符: 大于或等于
当第一个操作数值大于或等于第二个时，布尔算子将会返回 TRUE 值，操作数可以是 BOOL, BYTE, WORD,DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME 和 STRING.

IL 例子:
LD 60
GE 40
ST Var1 (* 结果是 TRUE *)

ST 例子:
VAR1 := 60 >= 40;

FBD 例子:



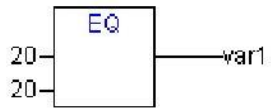
EQ

操作符: 等于
当第一个操作数值等于第二个时，布尔算子将会返回 TRUE 值，操作数可以是 BOOL, BYTE, WORD, DWORD,SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME 和 STRING.

IL 例子:
LD 40
EQ 40
ST Var1 (* 结果是 TRUE *)

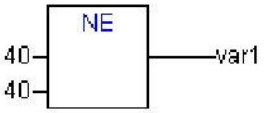
ST 例子:
VAR1 := 40 = 40;

FBD 例子:



NE

操作符: 不等于
当第一个操作数值不等于第二个时，布尔算子将会返回 TRUE 值，操作数可以是 BOOL, BYTE, WORD, DWORD,SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME 和 STRING.
IL 例子:
LD 40
NE 40
ST Var1 (* 结果是 FALSE *)
ST 例子:
VAR1 := 40 <> 40;
FBD 例子:



6、地址：

ADR

操作符: 地址功能未被列入 IEC61131-3 标准。
ADR 返回到 DWORD 中其自变数的地址。此地址会被传到加工功能处作为指针进行处理，或被分配到工程中作为指针。
dwVar:=ADR(bVAR);
IL 例子:
LD bVar
ADR
ST dwVar
man_fun1
注意: 在在线更改之后，还可以对某些地址方面的数据加以更改。

ADRINST

操作符: 地址功能未被列入 IEC61131-3 标准。
功能块程序实例中的 ADRINST 把 DWORD 中程序的地址归位。此地址会被传到加工功能处作为指针进行处理，或被分配到工程中作为指针。
ST 例子 (带有功能块的实例):
dvar:=ADRINST();
(* 实例中的地址被写到变量 dvar 中 *)
fun(a:=ADRINST()); (* 实例中的地址作为功能块 fun 的输入变量 *)

IL 例子:
ADRINST
ST dvar
ADRINST
Fun

BITADR

操作符: 地址功能未被列入 IEC61131-3 标准。
BITADR 将 DWORD 中片断内的位偏移较正归位。注意偏移值对应地址目标设定中选项固定器是否被激活而定。
VAR
var1 AT %IX2.3:BOOL;
bitoffset: DWORD;
END_VAR
ST 例子:
bitoffset:=BITADR(var1); (* Result if byte addressing=TRUE: 19, if byte addressing=FALSE: 35 *)
IL 例子:
LD Var1
BITADR
ST Var2
注意: 在在线更改之后，还可以对某些地址方面的数据加以更改。

^

内容操作符
IEC 操作符: 指示器可以采用在操作符“^”后增加内容符的方法废弃。
ST 例子:
pt:POINTER TO INT;
var_int1:INT;
var_int2:INT;

pt := ADR(var_int1);
var_int2:=pt^;
注意: 在在线更改之后，还可以对某些地址方面的数据加以更改。

7、调用操作:

CAL

IEC 操作符: 呼叫功能程序模块或程序。
用 IL 中的 CAL 来调出功能模块程序。把用作输入变量的变量放在功能模块程序名称之后的括号内。
例子:
从功能程序模块中调取 Inst 程序时，输入变量的 Par1 和 Par2 分别为 0 和 TRUE 。
CAL INST(PAR1 := 0, PAR2 := TRUE)

8、数据类型强制转换:

BOOL_TO_～	（布尔值转型）（INT/STRING/TIME/～D/DATE/DT 等）
～_TO_BOOL	（转型成布尔值）（BYTE/INT/TIME/STRING等）
INT_TO_～/SINT_TO_～	（整数类型转换）
REAL_TO_～/LREAL_TO_～	（实数型/长实数型转型）（INT 等）
TIME_TO_～/TIME_OF_DAY_TO_	（时间转型） （STRING/DWORD/SINT 等）
DATE_TO_～/DT_TO_～	（日期转型） （BOOL/INT/BYTE/STRING 等）
STRING_TO_～	（字符串转型） （BOOL/WORD/TIME 等）
TRUNC	（取整）

TRUNC

IEC 操作符: REAL 变为 INT. 整个值的数字部分将会被用到。
类型变换从大到小，会出现数据丢失的状况。
IL 例子:
LD 2.7
TRUNC
GE %MW8
ST 例子:
i:=TRUNC(1.9); (* 结果是 1 *)
i:=TRUNC(-1.4); (* 结果是 -1 *).

9、数据计算功能：

ABS	（取绝对值）
SQRT	（开方）
LN	（取自然对数）
LOG	（取对数）
EXP	（e 求幂）
SIN	（正弦）
COS	（余弦）
TAN	（正切）
ASIN	（反正弦）
ACOS	（反余弦）
ATAN	（反正切）
EXPT	（求幂）

ABS

操作符: 返回一个数字的绝对值。 ABS(-2) 等于 2。
可以将下列的输出输入变量进行类型合并:

IL 例子:
LD -2
ABS
ST i (* 结果是 2 *)

ST 例子:
i:=ABS(-2);

FBD 例子:

SQRT

操作符: 返回一个数字的平方根, 类型为 REAL。
IN 可以为类型 BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT.

IL 例子:
LD 16
SQRT
ST q (* 结果是 4 *)

ST 例子:
q:=SQRT(16);

FBD 例子:

LN

操作符: 返回一个数字的自然对数, 类型为 REAL。
IN 可以是类型 BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT.

IL 例子:
LD 45
LN
ST q
(* 结果是 3.80666 *)

ST 例子:
q:=LN(45);

FBD 例子:

LOG

IEC 操作符: 返回一个根为 10 的数字的对数, 类型为 REAL。
IN 可以是类型 BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT.

IL 例子:
LD 314.5
LOG
ST q
(* 结果是 2.49762 *)

ST 例子:
q:=LOG(314.5);

FBD 例子:

EXP

IEC 操作符: 返回一个以输入为指数, 自然数 e 为底数的值, 类型为 REAL。
IN 可以是类型 BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT.

IL 例子:
```
LD   2
EXP
ST   q (* 结果是 7.389056099 *)
```

ST 例子:
```
q:=EXP(2);
```

FBD 例子:

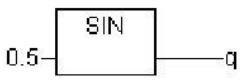
SIN

IEC 操作符: 返回一个数字的正弦值, 类型为 REAL。
输入值 IN 以弧度为单位进行计算。它可以是类型 BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT,UINT, UDINT.

IL 例子:
```
LD   0.5
SIN
ST   q (* 结果是 0.479426 *)
```

ST 例子:
```
q:=SIN(0.5);
```

FBD 例子:



TAN

IEC 操作符: 返回一个数字的正切值, 类型为 REAL。
输入值 IN 以弧度为单位进行计算。它可以是类型 BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT,UINT, UDINT.

IL 例子:
LD 0.5
TAN
ST q (* 结果是 0.546302 *)

ST 例子:
q:=TAN(0.5);

FBD 例子:

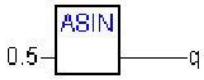
ASIN

IEC 操作符: 返回数字的反正弦值, 类型为 REAL, 单位: 弧度。 (正弦的反运算)
IN 可以是类型 BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT,

IL 例子:
LD 0.5
ASIN
ST q (* 结果是 0.523599 *)

ST 例子:
q:=ASIN(0.5);

FBD 例子:



ACOS

IEC 操作符: 返回一个数字的反余弦值, 类型为 REAL, 单位: 弧度。 (余弦的反运算)
IN 可以是类型 BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT,

IL 例子:

LD 0.5
ACOS
ST q (* 结果是 1.0472 *)

ST 例子:
q:=ACOS(0.5);

FBD 例子:

ATAN

IEC 操作符: 返回一个数字的反正切, 类型为 REAL, 单位: 弧度。 (正切的反运算)
IN 可以是类型 BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT.

IL 例子:
LD 0.5
ATAN
ST q (* 结果是 0.463648 *)

ST 例子:
q:=ATAN(0.5);

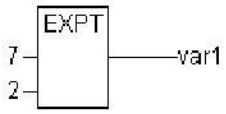
FBD 例子:

EXPT

IEC 操作符: 返回两个输入值的幂, 类型为 REAL: OUT = $IN1^{IN2}$.
IN1 和 IN2 可以是类型 BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT 必须是类型 REAL.
IL 例子:
LD 7
EXPT 2
ST var1 (* 结果是 49 *)

ST 例子:
var1 := EXPT(7,2);

FBD 例子:



4.2 库文件 Library

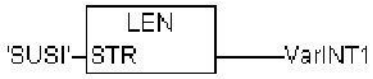
4.2.1 Standard.lib 标准库

LEN

返回字符串长度。
输入 STR 便是类型 STRING，功能的返回值是 INT 类型。
IL 例子:

```
LD 'SUSI'
LEN
ST VarINT1 (* 结果是 4 *)
```

FBD 例子:



ST 例子:

```
VarSTRING1 := LEN ('SUSI');
```

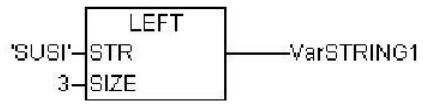
注意: 字符串功能并非〝thread safe〞：使用任务时，字符串功能只可用在单任务中。如果不同任务中使用相同的功能，便有被覆盖的危险。

LEFT

Left 返回，已知字符串为初始化字符串。
输入 STR 即类型 STRING，SIZE 为 INT 类型，此功能的返回值为 STRING 类型。
LEFT(STR,SIZE)意思是：在字符串 STR 从右边取第一个 SIZE 字节。
IL 例子:

```
LD      'SUSI'
LEFT 3
ST      VarSTRING1 (* 结果是 'SUS' *)
```

FBD 例子:



ST 例子:

VarSTRING1 := LEFT ('SUSI',3);

注意: 字符串功能并非〝thread safe〞：使用任务时，字符串功能只可用在单任务中。如果不同任务中使用相同的功能，便有被覆盖的危险。

RIGHT

Right 返回，已知字符串为初始化字符串。

RIGHT(STR,SIZE)意思是：从字符串 STR 的右边取第一个 SIZE 字节。

IL 例子:

```
LD          'SUSI'
RIGHT       3
ST          VarSTRING1 (* 结果是 'USI' *)
```

FBD 例子:

ST 例子:

VarSTRING1 := RIGHT ('SUSI',3);

注意: 字符串功能并非〝thread safe〞：使用任务时，字符串功能只可用在单任务中。如果不同任务中使用相同的功能，便有被覆盖的危险。

MID

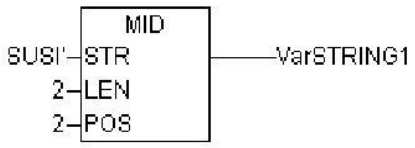
MID 从一个字符串中返回一部分字符串。

输入 STR 为类型 STRING，而 LEN 和 POS 为类型 INT，功能的返回值为类型 STRING

IL 例子:

```
LD   'SUSI'
MID 2,2
ST VarSTRING1 (* 结果是 'US' *)
```

FBD 例子:



ST 例子:

VarSTRING1 := MID ('SUSI',2,2);

注意: 字符串功能并非〝thread safe〞：使用任务时，字符串功能只可用在单任务中。如果不同任务中使用相同的功能，便有被覆盖的危险。

CONCAT

两个字符串的连接(合并)
输入变量 STR1 和 STR2 以及功能的返回值都是类型 STRING。
IL 例子:

```
  LD        'SUSI'
  CONCAT 'WILLI'
  ST        VarSTRING1 (* 结果是 'SUSIWILLI' *)
```

FBD 例子:

ST 例子:

```
    VarSTRING1 := CONCAT ('SUSI','WILLI');
```

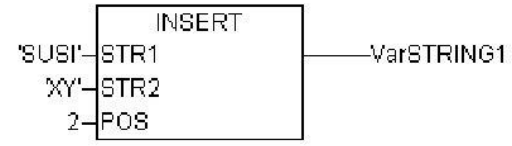
注意: 连接 CONTACT 功能如果嵌套三层以上便失灵。

INSERT

INSERT 可以在定义的点把一个字符串插入到另一个字符串中。
输入变量 STR1 和 STR2 为 STRING 类型，POS 是 INT 类型，功能返回值是 STRING 类型。
INSERT(STR1,STR2,POS)意思是：在 POS 之后将 STR2 插入到 STR1 中。
IL 例子:

```
  LD        'SUSI'
  INSERT 'XY',2
  ST        VarSTRING1 (* 结果是 'SUXYSI' *)
```

FBD 例子:



ST 例子:

```
    VarSTRING1 := INSERT ('SUSI','XY',2);
```

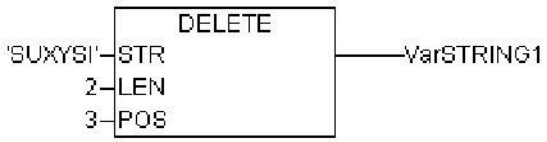
注意: 字符串功能并非〝thread safe〞：使用任务时，字符串功能只可用在单任务中。如果不同任务中使用相同的功能，便有被覆盖的危险。

DELETE

DELETE 可以在定义的位置把一个较大的字符串的一部分删除掉。
输入变量 STR 为 STRING 类型，LEN 和 POS 为 INT 类型，功能的返回值是 STRING 类型。
DELETE(STR,L,P)意思是：从以 P 位置开始的字节处将 L 字节删除。
IL 例子:

```
LD      'SUXYSI'
DELETE 2,3
ST      Var1 (* 结果是 'SUSI' *)
```

FBD 例子:



ST 例子:

```
Var1 := DELETE ('SUXYSI',2,3);
```

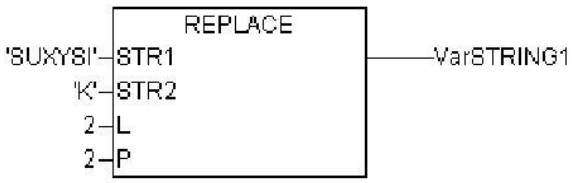
注意: 字符串功能并非〝thread safe〞：使用任务时，字符串功能只可用在单任务中。如果不同任务中使用相同的功能，便有被覆盖的危险。

REPLACE

REPLACE 可用第三个字符串从较大的字符串中替换掉部分字符串。
输入变量 STR1 和 STR2 为 STRING 类型，LEN 和 POS 为 INT 类型。
REPLACE(STR1,STR2,L,P)意思是：从以 P 位置开始的 STR2 把 STR1 从 L 字节处替换。
IL 例子:

```
LD          'SUXYSI'
REPLACE 'K',2,2
ST          VarSTRING1 (* 结果是 'SKYSI' *)
```

FBD 例子:



ST 例子:

```
VarSTRING1 := REPLACE ('SUXYSI','K',2,2);
```

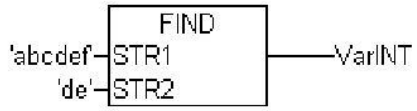
注意: 字符串功能并非〝thread safe〞：使用任务时，字符串功能只可用在单任务中。如果不同任务中使用相同的功能，便有被覆盖的危险。

FIND

FIND 在一个字符串中查找到某部分字符串。
输入变量 STR1 和 STR2 都是 STRING 类型，功能的返回值是 INT 类型。
FING(STR1,STR2)意思是：查找 STR1 第一次出现在 STR2 处时，后者第一个字节的位置。
IL 例子:

```
  LD          'abcdef'
  FIND 'de'
  ST          VarINT1 (* 结果是 '4' *)
```

FBD 例子:



ST 例子:

```
    arINT1 := FIND ('abcdef','de');
```

注意: 字符串功能并非〝thread safe〞：使用任务时，字符串功能只可用在单任务中。如果不同任务中使用相同的功能，便有被覆盖的危险。

注意：当使用FIND比较文件名时，文件名为纯数字的，比较的后缀需要为大写。例如 FIND(‘123.txt’,’TXT’);非纯数字的文件不分大写小。

SR

重置双稳态功能模块
Q1 = RS (SET, RESET1) 的意思是:
Q1 = NOT RESET1 AND (Q1 OR SET)
输入变量 SET 和 RESET1 以及输出变量 Q1 都是 BOOL 类型。
声明如下所示:
RSInst : RS ;
IL 例子:

```
  CAL RSInst(SET:= VarBOOL1,RESET1:=VarBOOL2)
  LD RSInst.Q1
  ST VarBOOL3
```

FBD 例子:

ST 例子:

```
  RSInst(SET:= VarBOOL1 , RESET1:=VarBOOL2 );
```

```
   VarBOOL3 := RSInst.Q1 ;
```

RS

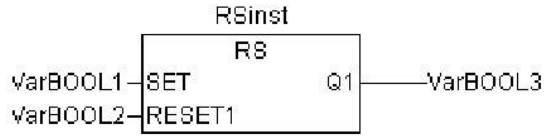
重置双稳态功能模块
Q1 = RS (SET, RESET1) 的意思是:
Q1 = NOT RESET1 AND (Q1 OR SET)
输入变量 SET 和 RESET1 以及输出变量 Q1 都是 BOOL 类型。
声明如下所示:

```
RSInst : RS ;
```

IL 例子:

```
   CAL RSInst(SET:= VarBOOL1,RESET1:=VarBOOL2)
   LD RSInst.Q1
   ST VarBOOL3
```

FBD 例子:



ST 例子:

```
   RSInst(SET:= VarBOOL1 , RESET1:=VarBOOL2 );
   VarBOOL3 := RSInst.Q1 ;
```

SEMA

软件信号量(可断续的)
BUSY = SEMA(CLAIM, RELEASE) 的意思是:

```
BUSY := X;
IF CLAIM THEN X:=TRUE;
ELSE IF RELEASE THEN BUSY := FALSE; X:= FALSE;
END_IF
```

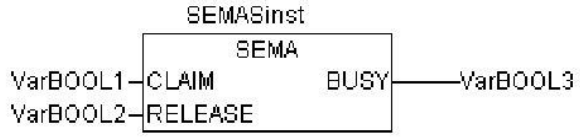
X 为内部布尔变量，初始化时是 FALSE。输入变量 CLAIM 和 RELEASE 以及输出变量 BUSY 都是 BOOL 变量。
如果调用 SEMA 时，BUSY 为 TURE，即 SEMA 已经被分配了值(SEMA 被调用时，CLAIM＝TRUE)。若 BUSY 是FALSE，不调用 SEMA 或已被启动(RELEASE=TRUE)。
声明如下所示:

```
SEMAInst : SEMA ;
```

IL 例子:

```
   CAL SEMAInst(CLAIM:=VarBOOL1,RELEASE:=VarBOOL2)
   LD SEMAInst.BUSY
   ST VarBOOL3
```

FBD 例子:



ST 例子:

```
SEMAInst(CLAIM:= VarBOOL1 , RELEASE:=VarBOOL2 );
VarBOOL3 := SEMAInst.BUSY;
```

R_TRIG

R_TRIG 功能块触发一个上升的边界。

```
FUNCTION_BLOCK R_TRIG
VAR_INPUT
  CLK : BOOL;
END_VAR
VAR_OUTPUT
  Q : BOOL;
END_VAR
VAR
  M : BOOL := FALSE;
END_VAR
Q := CLK AND NOT M;
M := CLK;
```

只要输入变量 CLK 为 FALSE，输出 Q 和帮助变量 M 依然是 FALSE。一旦 CLK 返回 TRUE，Q 首先返回 TRUE，那么 M 就会被设置为 TRUE。此意味着，每调用一次此功能，Q 便会返回 FALSE，直到 CLK 边界已下降，随即边界再次上升。

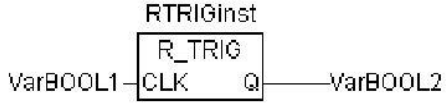
声明如下所示:

```
RTRIGInst : R_TRIG ;
```

IL 例子:

```
  CAL RTRIGInst(CLK := VarBOOL1)
  LD RTRIGInst.Q
  ST VarBOOL2
```

FBD 例子:



ST 例子:

```
  RTRIGInst(CLK:= VarBOOL1);
  VarBOOL2 := RTRIGInst.Q;
```

F_TRIG

F_TRIG 功能块触发一个下降的边界。

```
FUNCTION_BLOCK F_TRIG
VAR_INPUT
   CLK: BOOL;
END_VAR
VAR_OUTPUT
   Q: BOOL;
END_VAR
VAR
   M: BOOL := FALSE;
END_VAR
Q := NOT CLK AND NOT M;
M := NOT CLK;
```

只要输入变量 CLK 返回 TRUE，输出 Q 和帮助变量 M 依然是 FALSE。一旦 CLK 返回 FALSE， 首先返回 TRUE，那么 M 就会被设置为 TRUE。此意味着，每调用一次此功能，Q 便会返回 FALSE，直到 CLK 边界已上升，随即边界再次下降。

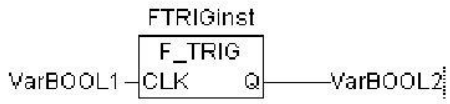
声明如下所示:

```
FTRIGInst : F_TRIG ;
```

IL 例子:

```
   CAL FTRIGInst(CLK := VarBOOL1)
   LD   FTRIGInst.Q
   ST VarBOOL2
```

FBD 例子:



ST 例子:

```
   FTRIGInst(CLK:= VarBOOL1);
   VarBOOL2 := FTRIGInst.Q;
```

CTU

功能模块增量器:

输入变量 CU 和 RESET 以及输出变量 Q 都是 BOOL 类型，输入变量 PV 和输出变量 CV 是 WORD 类型。

计数器变量 CV 在 RESET 为 TURE 时被初始化为 0，如果 CU 从 FALSE 到 TURE 为上升边界，CV 将被增加 1，Q 在 CV 大于或等于 PV 上限时，会返回 TRUE。

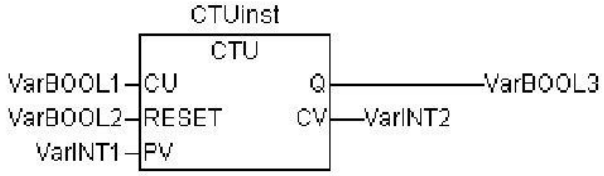
声明如下所示:

```
CTUInst : CTU ;
```

IL 例子:

```
CAL   CTUInst(CU := VarBOOL1, RESET := VarBOOL2, PV := VarINT1)
LD CTUInst.Q
ST VarBOOL3
LD CTUInst.CV
ST VarINT2
```

FBD 例子:



ST 例子:

```
CTUInst(CU:= VarBOOL1, RESET:=VarBOOL2 , PV:= VarINT1);
VarBOOL3 := CTUInst.Q ;
VarINT2 := CTUInst.CV;
```

CTD

功能模块减数器:

输入变量 CD 和 LOAD 以及输出变量 Q 都是 BOOL 类型，输入变量 PV 和输出变量 CV 为 WORD 类型。

当 LOAD_为 TRUE 时，如果 CV 大于 0(即，它不致让值低于 0).，CV 则会下降 1。

当 CVis 等于 0 时 Q 返回 TRUE。

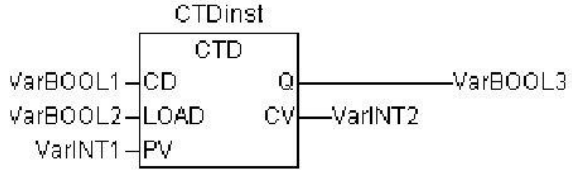
声明如下所示:

```
CTDInst : CTD ;
```

IL 例子:

```
CAL   CTDInst(CD := VarBOOL1, LOAD := VarBOOL2, PV := VarINT1)
LD CTDInst.Q
ST VarBOOL3
LD CTDInst.CV
ST VarINT2
```

FBD 例子:



ST 例子:

```
CTDInst(CD:= VarBOOL1, LOAD:=VarBOOL2 , PV:= VarINT1);
VarBOOL3 := CTDInst.Q ;
VarINT2 := CTDInst.CV;
```

CTUD

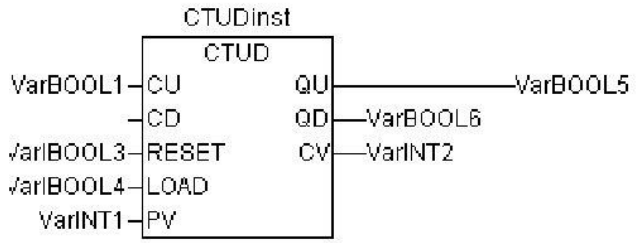
功能模块增量器/减量器
输入变量 CU，CD，RESET，LOAD 以及输出变量 QU 和 QD 都是 BOOL 类型，PV 和 CV 是 WORD 类型。
若 RESET 有效，计数变量 CV 以 0 值进行初始化。若 LOAD 有效，CV 则以 PV 进行初始化。若 CU 从 FALSE 到 TRUE 边界下降，则 CV 上 1。如果 CD 工 FALSE 到 TRUE 为上升边界，如果这不会使数值降到 0 以下，CV 则会降 1。CV 大于等于 PV 时，QU 返回 TRUE。CV 等于 0 时，QD 返回 TRUE。
声明如下所示:

```
CTUDInst : CUTD ;
```

IL 例子:

```
  CAL CTUDInst(CU:=VarBOOL2, RESET:=VarBOOL3,
  LOAD:=VarBOOL4, PV:=VarINT1)
  LD CTUDInst.Q
  ST VarBOOL5
  LD CTUDInst.QD
  ST VarBOOL5
  LD CTUInst.CV
  ST VarINT2
```

FBD 例子:



ST 例子:

```
  CTUDInst(CU := VarBOOL1, CU:= VarBOOL2, RESET :=
  VarBOOL3, LOAD:=VarBOOL4 , PV:= VarINT1);
  VarBOOL5 := CTUDInst.QU ;
  VarBOOL6 := CTUDInst.QD ;
  VarINT2 := CTUDInst.CV;
```

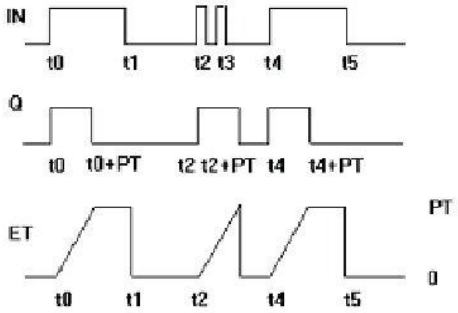
TP

功能块定时器是一个触发器。TP（IN，PT，Q，ET）意思是:
IN 和 PT 分别是类型为 BOOL 和 TIME 的输入变量。Q 和 ET 分别是类型为 BOOL 和 TIME 的输出变量。若 IN是 FALSE，Q 是 FALSE，ET 为 0。

在 IN 变成 TRUE 时，ET 中的时间便开始以毫秒计数，直到其值等于 PT。然后便保持恒定。当 IN 为 TRUE 并且 ET 小于或等于 PT 时，Q 为 TRUE，。否则 Q 是 FALSE。

Q 返回一个 PT 中已知的时间段信号。

定时器时间顺序的图表显示：



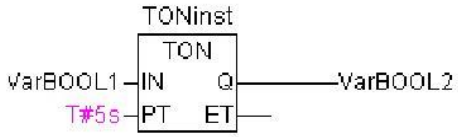
声明如下所示:

TPInst : TP ;

IL 例子:

```
CAL TPInst(IN := VarBOOL1, PT := T#5s)
LD   TPInst.Q
ST   VarBOOL2
```

FBD 例子:



ST 例子:

```
TPInst(IN := VarBOOL1, PT:= T#5s);
VarBOOL2 :=TPInst.Q;
```

TON

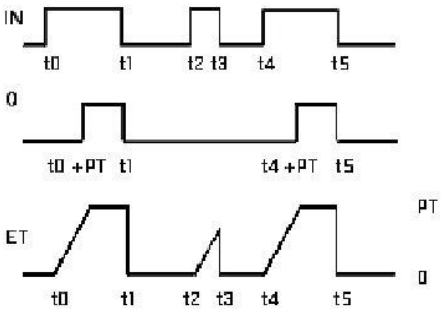
功能块 Timer On Delay 实现开启延迟。

TON(IN, PT, Q, ET)意思是：

IN 和 PT 分别是类型为 BOOL 和 TIME 的输入变量。Q 和 ET 分别是类型为 BOOL 和 TIME 的输出变量。若 IN是 FALSE，Q 则是 FALSE，ET 则为 0。

IN 一旦变成 TRUE 时，ET 中的时间便开始以毫秒计数，直到其值等于 PT。然后便保持恒定。当 IN 为 TURE并且 ET 等于 PT 时，Q 为 TRUE。否则 Q 为 FALSE。这样，当 PT 所示的以毫秒为单位的时间耗尽时，Q 呈现一个上升的边界。

TON 时间行为图表如图：



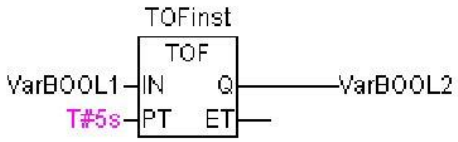
声明如下所示:

TONInst : TON ;

IL 例子:

```
CAL TONInst(IN := VarBOOL1, PT := T#5s)
LD   TONInst.Q
ST   VarBOOL2
```

FBD 例子:



ST 例子:

```
TONInst(IN := VarBOOL1, PT:= T#5s);
```

TOF

功能块 TOF 实现关闭延迟。

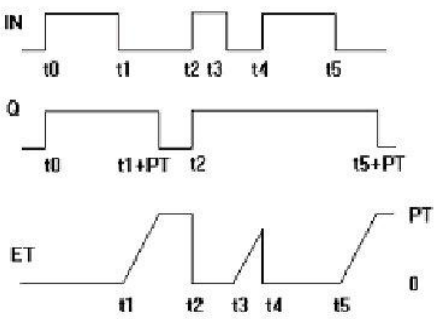
TOF(IN, PT, Q, ET)意思是:

IN 和 PT 分别是类型为 BOOL 和 TIME 的输入变量。Q 和 ET 分别是类型为 BOOL 和 TIME 的输出变量。若 IN是 TRUE，则输出变量分别为 TRUE 和 0。

一旦 IN 变为 FALSE，ET 中时间便开始以毫秒为单位进行计时，直到值等于 PT 为止。然后便保持恒定。

当 IN 为 FALSE 并且 ET 等于 PT 时，Q 为 FALSE。否则 Q 为 TRUE。这样，当 PT 以毫秒为单位所示的时间耗尽时，Q 的边界下降。

TOF 时间行为图表如图:



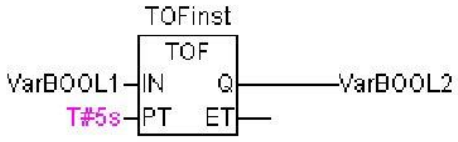
声明如下所示:

TOFInst : TOF ;

IL 例子:

```
CAL TOFInst(IN := VarBOOL1, PT := T#5s)
LD TOFInst.Q
ST VarBOOL2
```

FBD 例子:

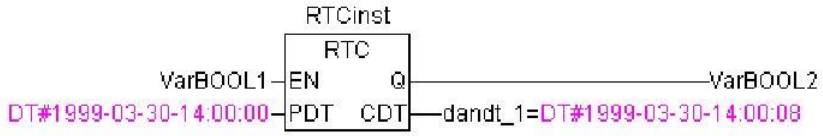


ST 例子:

```
TOFInst(IN := VarBOOL1, PT:= T#5s);
VarBOOL2 :=TOFInst.Q;
```

RTC

定时开始，功能块 Runtime Clock 返回当前时间和日期。



RTC(EN, PDT, Q, CDT)意思是：

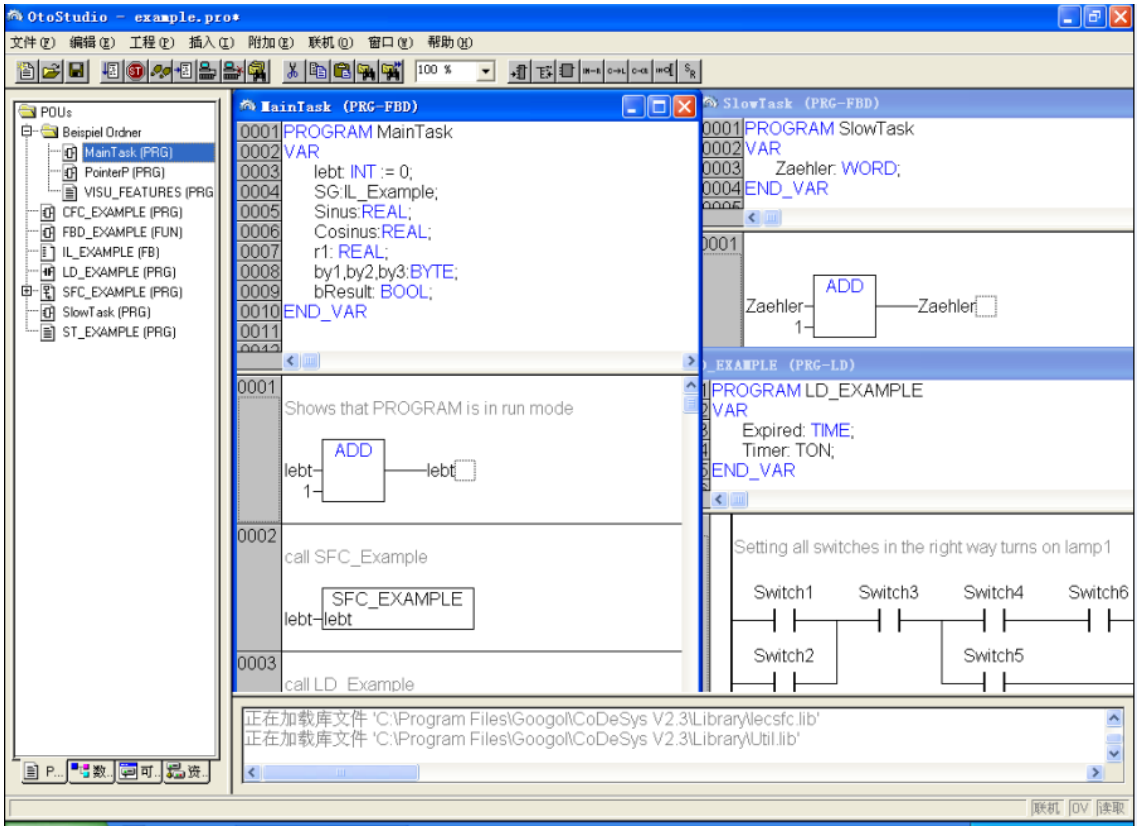
EN 和 PDT 是类型为 TIME 输入变量，Q 和 CDT 分别是类型为 BOOL 和 DATE_AND_TIME 输出变量。若 EN 为FALSE 时，输出变量 Q 和 CDT 则分别时 FALSE 和DT#1970-01-01-00：00：00。一旦 EN 变为 FALSE，PDT 时间已定，并以秒为单位计时。

只要 EN 变成 TRUE，CDT 则复原(见上例图示)。每当 EN 重设成 FALSE，则 CDT 又重设为初始值DT#1970-01-01-00：00：00。请注意，PDT 的时间只能被上升边界设定。

第五章 OtoStudio 各个单独的组件

5.1 主窗口

主窗口的组成



OtoStudio 的主窗口从上到下包括以下组件

主菜单；

工具栏（可选择）：快速选择菜单的命令的按钮；

带程序组织单元、数据类型、可视化和资源选项卡的对象管理器；

在对象管理器和 OtoStudio 工作空间之间的垂直屏幕分隔器；

用于包含编辑器窗口的工作空间；

信息窗口（可选择）；

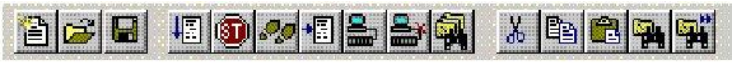
状态栏（可选）：显示当前工程文件的状态的信息。

以上内容也可参看：上下文菜单

主菜单

菜单条位于主窗口的最上边，它包含了所有的菜单命令。

工具栏



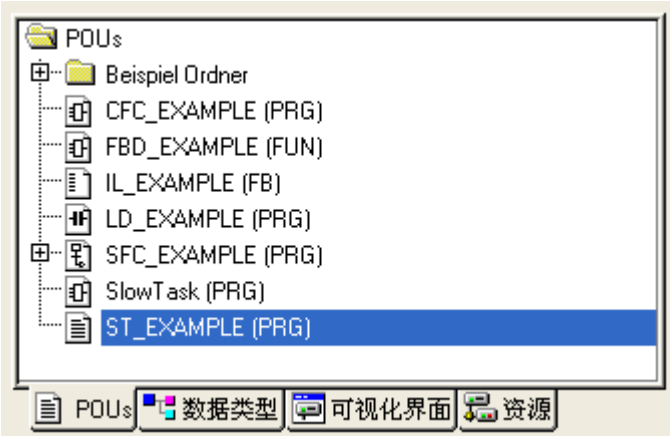
通过用鼠标在符号上单击，就可以更迅速的选择一个菜单命令，所选用的符号能自动的出现于活动窗口。

当鼠标键按在工具按钮上单击，然后释放后，命令才执行。

如果用鼠标指针在工具按钮上停留几秒钟，在工具条中就显示按钮的信息。

为了能看到工具栏中每个按钮的描述，在帮助编辑器中选择你想要的信息，单击你感兴趣的工具栏符号。工具栏的显示是可选择（参看“工程”“选项”中“桌面”部分的说明）

对象管理器



对象管理器通常位于 OtoStudio 的左边，在底部有四个选项卡，它们是：POUs、数据类型、可视化界面、资源。单击或使用键盘的左或右箭头能在相应的选项卡之间变换。

在管理对象章节中你会学到在对象管理器中怎样使用对象。

屏幕分割器

屏幕分割器是指两个非重叠窗口之间的边界。OtoStudio 中，在主窗口的工作区和对象管理器之间有一个屏幕分割器。在界面（声明部分）和程序组织单元的执行（指令部分）以及工作区和信息窗口之间都有一个屏幕分割器。你可以用鼠标指针来移动屏幕分割器，按住鼠标的左键来拖动到合适的位置。

在窗口尺寸发生变化时，应确保屏幕分割器的绝对位置。如果屏幕分割器好象不见了，那么只要放大你的窗口就会重新出现。

工作区

工作区位于 OtoStudio 中主窗口的右边，在这个区域可以打开对象的所有编辑器和库文件管理器。当前的对象名显示在标题栏中，在它后面的括号中显示 POU 中一个缩略的 POU 类型和当前使用的编程语言。在编辑器章节将详细讲述 编辑器的功能。

在“窗口”菜单下，可以得到所有 窗口管理命令。

信息窗口

消息窗口位于主窗口的 工作区的 屏幕分割器的下面。它包含了所有来自先前的编译、

检查或比较的信息。搜索结果和交叉引用列表也能从这里输出。

如果在消息窗口的消息上双击鼠标或者是按回车键，编辑器将打开这个对象，对象的相应的行被选中。用命令 '编辑' '下一个错误' 和“'编辑' '前一个错误' ，能迅速的在错误消息之间跳转。

状态栏

OtoStudio 的主窗口的窗口框架底部的状态栏给出了当前工程文件和菜单命令的信息。如果一个项目激活，那么它的名称将在状态栏的右边以黑色字迹显示，否则以灰色字迹显示。当你在联机模式时， Online 以黑色字迹显示状态栏中，在离线模式下，它以灰色字迹显示。在联机模式下，可以从状态栏中看到是否在 仿真模式（SIM）、程序是否在运行（RUNS）、是否设置了断点（BP）或是否有 强制赋值（FORCE）。

在文本编辑器中，可以显示鼠标指针的行和列的位置。（例如， 行:5,列.:11）。联机模式下‘OV’在状态栏中显示为黑色，按〈Ins〉键在覆盖和插入之间转换。

如果鼠标指针是可视化的,当前鼠标指针的 X 和 Y 位置将以与屏幕左上角相关的象素表示，如果鼠标指针是在元素上,或元素正在处理中,那么将会显示出它的数值， 如果插入了对象,那么它也会显示出来（例如：矩形）。

如果你已经选择了一个菜单命令但是还没有执行它,那么在状态栏中会出现一个简短的描述。

状态栏的显示是可选择的（参看“工程”“选项”中“桌面”部分的说明）。

上下文菜单

快捷方式：<Shift>+<F10>

可以用鼠标右键来代替使用菜单栏执行命令，菜单包含了对象或活动编辑器常用的命令，根据激活窗口的不同出现不同的内容菜单。

5.2 工程选项

点击”工程”->”选项”，可以打开设置选项的对话框。这些选项被分为不同的类别。可以使用鼠标点击，或者是使用箭头键，选择位于对话框左侧的你所要求的选项，或者对位于对话框右侧的选项进行更改。

在资源标签中的组件 “工作空间”中，可以找到已经设定的当前工程文件选项信息。在主窗口中，可以看到已经设定的相关信息。除非特殊指定，这些信息都存储在文件“OtoStudio.ini”中，并在下一次启动 OtoStudio 软件的时候，重新恢复。

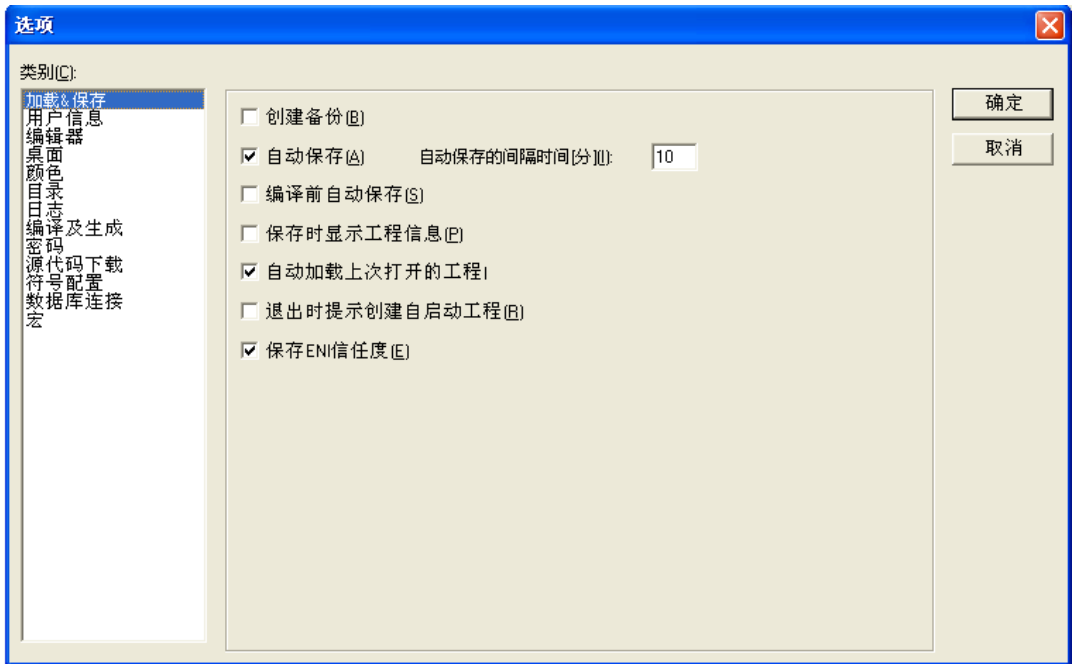
可以对以下的类别进行配置：

	stored in OtoStudio	stored in project
加载和保存		
用户信息		
编辑器		
桌面		
颜色		
目录		
编译及生成		

密码		
源代码下载		
符号配置		
数据库连接		
宏		

5.2.1 加载与保存

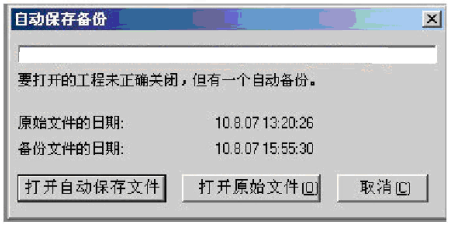
在选项对话框中选择了这个选项将出现下面的对话框：



当激活一个选项时，选项前面会出现一个√。

创建备份：OtoStudio 在每一次保存文件的时候，都创建一个扩展名".bak"的备份文件，和下面自动保存创建的扩展文件名*.asd 不同，这个扩展名为".bak"的文件在关闭工程文件以后一直保存着，因此你可以恢复最近保存的工程文件的版本。

自动保存：根据设定的时间间隔（自动保存间隔），工程文件将保存到扩展名为".asd"的一个临时文件中，这个文件在程序正常退出之后删除。任何情况下，OtoStudio 没有正常的关闭（例如，因为电源故障），那么文件将不会删除。当你再次打开文件的时候会出现下列信息：



此时，你可以决定是否想打开原始文件或自动保存文件。

编辑前自动保存：在每次编译之前，工程文件将自动保存，在这个过程中创建一个扩展名为".asd"的文件，这个文件的作用像上面在选项“自动保存”中讲述的一样。

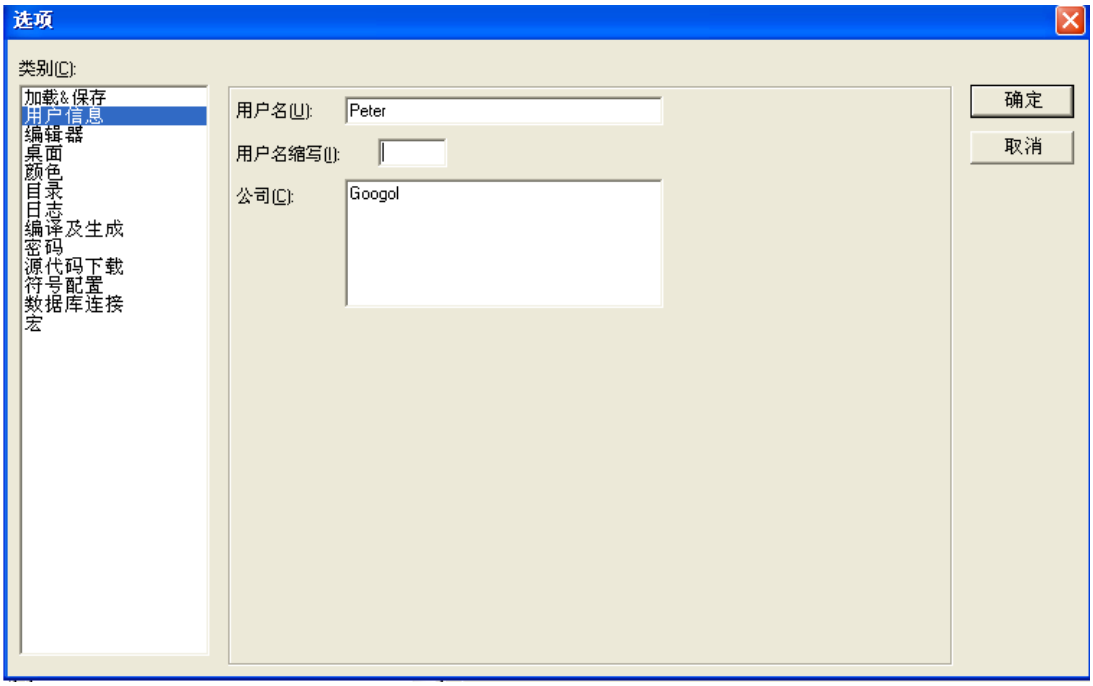
询问工程信息：当保存一个新的工程文件或另存为一个新文件名字的时候，系统自动访问工程文件的信息，可以通过命令“工程”“工程信息”对文件信息可视化并处理它。

自动加载：在下次 OtoStudio 启动的时候自动加载最近打开的工程文件，可以在命令行中输入工程文件，以便在启动 OtoStudio 软件时，加载相应的文件。

导入命令提醒：如果工程文件已经被修改并且没有创建新的导入文件，那么在用户退出工程之前，系统弹出一个对话框，提醒用户 “上次下载后没有创建导入工程文件。要退出吗？”

5.2.2 用户信息

如果在 选项对话框中选择了这个类别，就会出现如下的对话框：



在这个对话框中，用户名、名字的大写字母缩写和他的工作的公司都属于用户信息，每个条目都可以进行修改，这些设置将会应用到任何在本地计算机上由 OtoStudio 创建的工程文件中。

5.2.3 编辑器

如果在 选项对话框中选择了这个类别，就会看到以下的对话框：
可以为编辑器进行下列的设置：

自动声明：如果激活这个选项，在输入一个没有声明的变量后，所有的编辑器中出现声明所需变量的对话框。

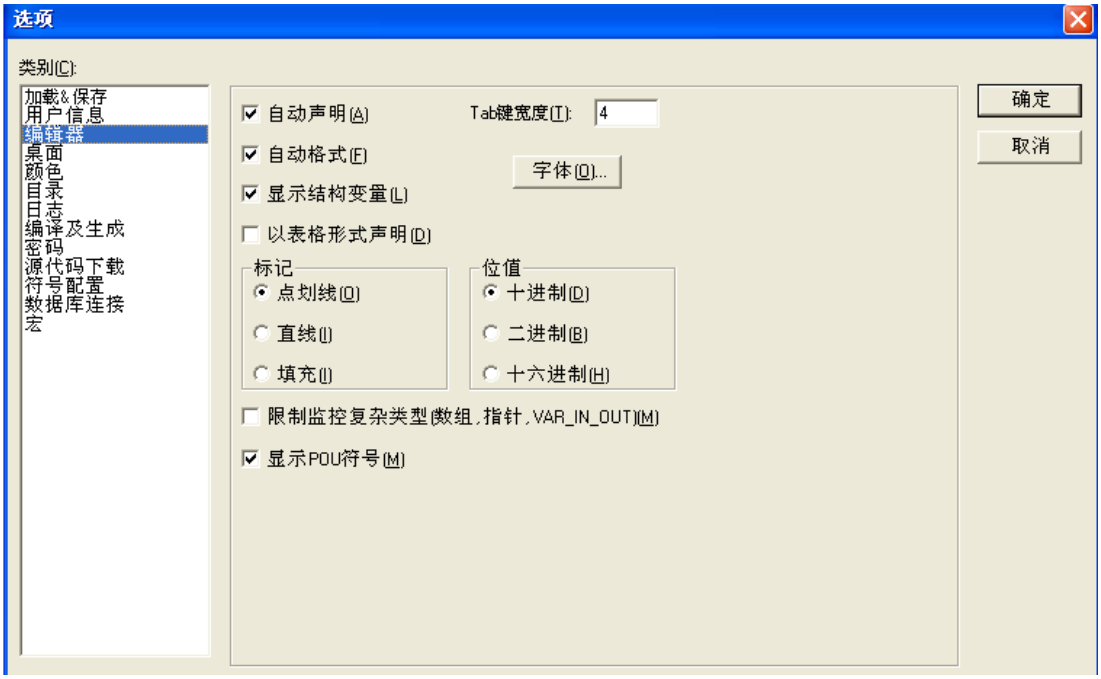
自动格式：如果激活这个选项，那么 OtoStudio 软件将自动对 指令表语言和 声明编辑器进行格式化处理。

当你完成一行时，产生下列的格式：

1. 以小写字母写的操作数以大写字母显示；

2. 制表符插入到列中。

显示结构变量：如果激活这个选项，系统可以提供智能化的输入帮助功能。如果在一个

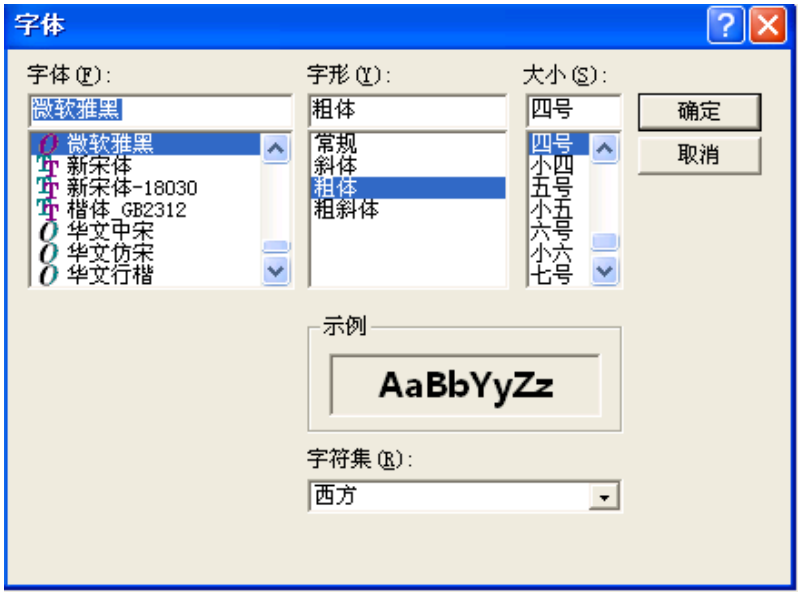


应该插入工程符号的地方插入一个小圆点，系统就会自动打开一个选择列表。列表中提供了工程用到的所有全局变量。如果你插入了功能模块实例的名称，那么你会得到实例功能模块的所有输入和输出的列表。在编辑器中、在观察和接收管理器中、在可视化中和在采样追踪中都可以使用这个智能化的功能。

以表格形式声明：如果激活这个选项，可以在表格中编辑变量而代替使用通常的声明编辑器，这个表格象一个纸牌盒那样存储，在这个纸牌盒中，你可以找到用于输入变量、输出变量、局部变量和输入输出变量的各种符号。对于每一个变量，都有编辑区进行名称，地址，类型，大写和注释等内容的编辑。

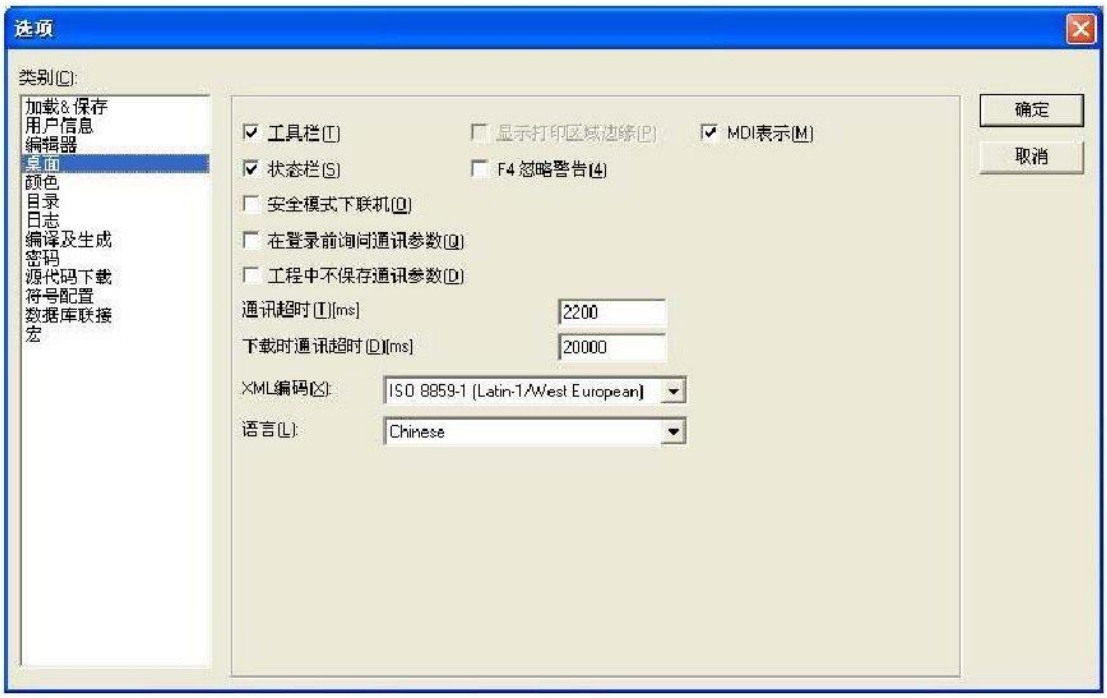
Tab 键宽度：在编辑器中按 Tab 键后跳过的字符数，默认的设置是四个字符，字符的宽度依赖于所选择的字体。

字体：在对话框中编辑类别中字体按钮上单击，可以在所有的 OtoStudio 编辑器中选择字体，对所有的制图操作来说字体的大小是基本的单元，选择一个大字体尺寸能放大字体。在输入命令之后，出现选择字体、字型和字号的字体对话框。



5.2.4 桌面

如果在选项对话框中选择了这个类别，将会出现如下的对话框：



当激活一个选项，在它的旁边出现一个√。

工具栏：在菜单栏下面的快速选择菜单命令的工具栏的按钮变为可视。

状态栏：在 OtoStudio 主窗口中底部边界的状态栏变为可视。

安全模式下的联机：在联机模式下，有以下命令： '运行', '停止', '复位' '设置断点', '单循环', '写入新值', '强制新值' 和 '强制赋值'，并出现一个带有确认请求命令是否确实执行的对话框。如果目标系统支持，当你想从设计系统加载一个实际工程到 PLC 中，并且这个实际工程已经存在时，就可得到一个扩展的对话框。这个对话框既可显示已有工程的工程信息，

又可显示当前要加载的工程的信息。当在 PLC 中已经有一个工程，再创建一个导入工程时，也要用到这些有关工程的信息。

这个选项和工程一起保存。

在登录前查询通讯参数：只要执行了命令“联机”“登陆”，将首先打开通讯参量对话框，为进入联机模式，必须首先通过 OK 选择这个对话。

工程中不保存通讯参量：通讯参量的设置对话（“联机”“通讯参数”）将不会和工程一起保存。

显示空白区域边缘：在每个编辑窗口，当前设置打印范围的界限用红色虚线标记。它们的尺寸依赖于打印机的特性（纸张大小，方向）和打印版面的“内容”区域的大小。（“菜单”“文本设置”）。

F4 忽略警告：在编译之后，当在一个消息窗口按下 F4 时，就会跳到有错误信息“警告信息被忽略”的行中。

MDI 表示：这个选项默认情况下是激活的，因而能同时打开几个窗口。如果选项是无效的（单文档模式）只有一个窗口能打开并全屏显示。例外情况：一个程序的动作和程序自身能在单文档界面模式下并排显示。

语言：在这里定义菜单和对话框文本中显示的语言。请注意：在 windows98 下不能进行语言选择。

5.2.5 颜色

如果在 选项对话框中选择了这个类别，将会出现如下的对话框：

可以对 OtoStudio 中的默认的颜色设置进行编辑。你可以选择为行号码（默认预设置：浅灰色）、为断点位置（黑灰色）、为设置断点（浅蓝色）、为当前断点（红色）、到达位置（绿色）来改变的颜色设置，或者为布尔变量值的监视（蓝色）而改变颜色设置。如果你已经选择了一个显示的按钮，将会打开选择颜色的对话框。

5.2.6 目录

如果在 选项对话框中选择了这个类别，将会出现如下的对话框：

可以在工程和公共区域为 OtoStudio 输入目录（例如编译文件是图和列表文件，而不是符号文件。后者将存储在工程目录），以便进行库文件和控制器配置文件的查询，或者查找存储编译和源上传文件。如果你点击一个区域后面的...按钮，将打开目录选择对话框，对库和配置文件，可以输入多个路径，彼此之间用分号间隔开。

请注意：通过使用前缀.，库文件路径可以基于工程文件路径上输入。例如输入".\libs"。如果当前工程是在'C:\programs\projects\libs'中，库文件将会从'C:\programs\projects\libs'中查找。可以参看 “插入”、“添加库文件”。

请注意：不要在目录路径中使用空格和除了"_"之外的特殊字符。

工程区域里的信息和工程一起保存，在公共区域的信息写到编程系统的 ini 文件中并应用到所有的工程。

目标区域显示了在目标系统中设置的库文件和配置文件的目录。例如，这些区域是不能编辑的，但是条目能被选择和复制（右鼠标键内容菜单）。

OtoStudio 通常首先在“工程”中输入的目录中查找，然后在“目标系统”（在目标文件中定义），最后在“公共”区域查找。如果找到两个同名文件，将使用在目录中首先找到的文件。

5.2.7 日志

如果在选项对话框中选择了这个类别，将会看到如下的对话框：

在这个对话中，配置工程的日志文件。在联机模式过程中它记录所有用户动作和内部过程。

如果一个现有的工程没有日志文件，将会打开一个对话框，让你注意到一个日志现在正在建立，它将接收在下个登录过程中首次输入。

当保存工程时，日志文件在工程目录中自动存储为二进制文件。如果你喜欢存放在一个不同的目标目录，可以激活选项“工程日志目录”，在编辑区域输入适当的路径。

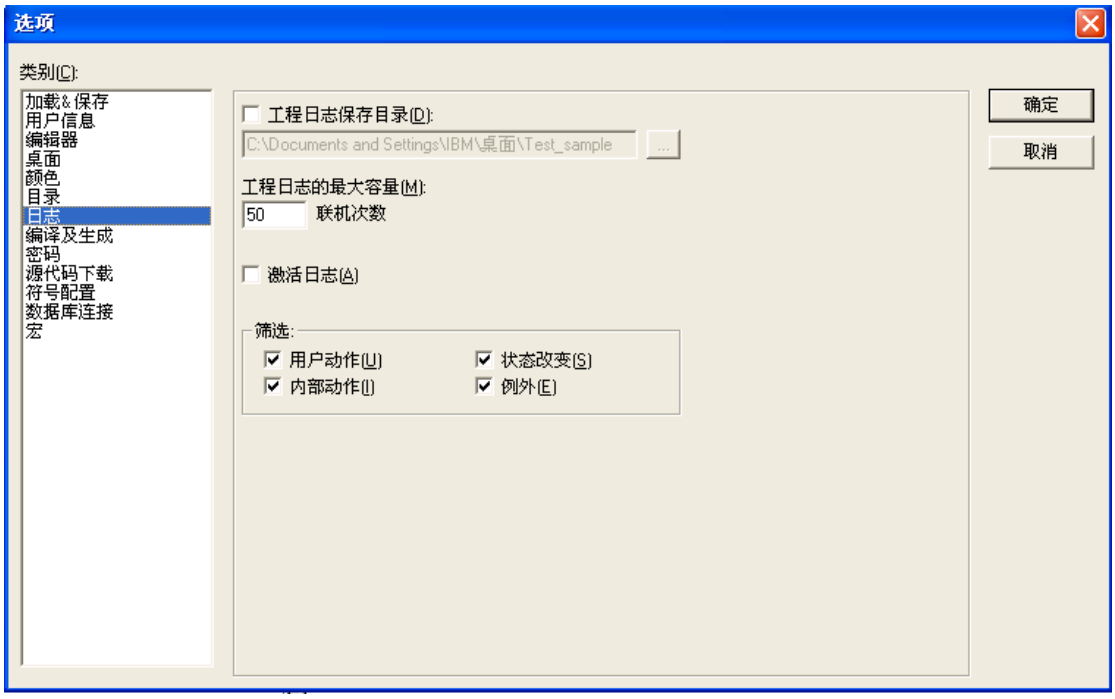
日志文件自动分配工程的名字再加上扩展名‘.log’，联机阶段的最大次数由最大工程日志大小决定。如果在记录过程中超过了这个数字，系统将删除旧的条目为新条目让出空间。

通过选项中的激活日志区域可以控制日志功能的开启或关闭。

可以在 Filter 区域选择记录哪些动作：用户动作、内部动作、状态改变、例外，只有在这里选中的属于目录的动作才会出现在日志窗口并记录到日志文件中（请参照日志来了解关于目录的更多的知识）。

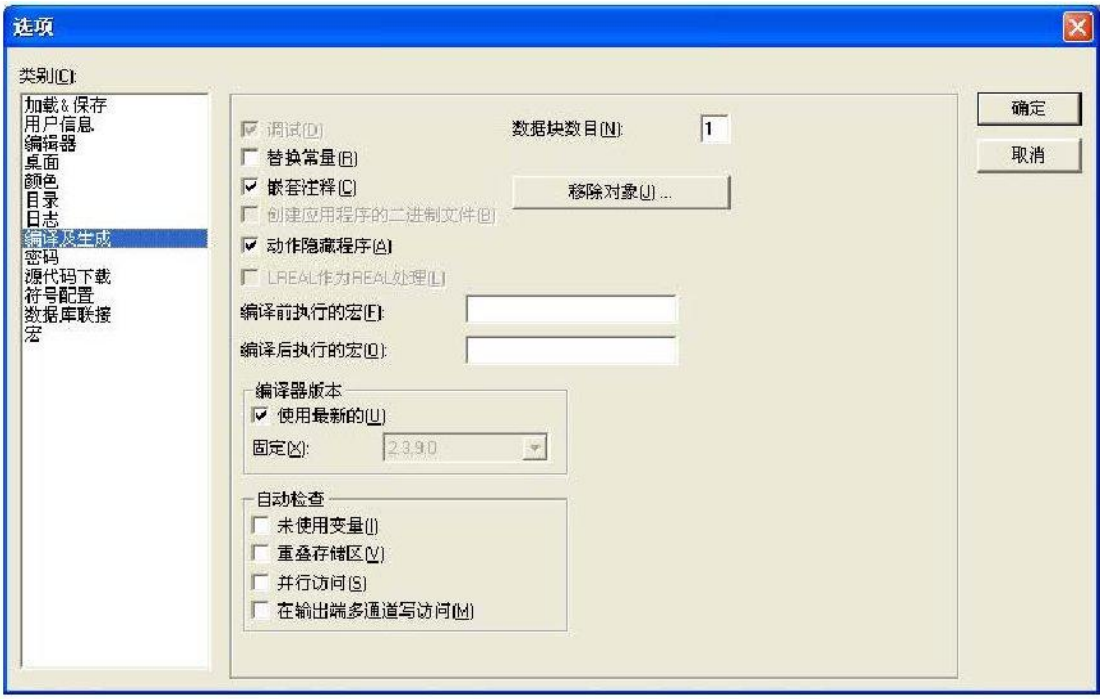
可以通过命令 '窗口''日志'来打开日志窗口。

日志选项对话框：



5.2.8 编译及生成

如果选择 选项中的’生成‘选项，将显示下面的对话框。



调试：取决于用户在目标描述中是否激活此项。如果它被激活，将会创建附加的代码，这些代码相当大。

为了充分使用 OtoStudio 提供的调试功能（例如，断点）需要使用这些代码。当你关闭这个选项，工程处理将加快并且代码的大小减小，这个选项和工程一起保存。

替换常量：每一个常量都直接加载，在联机模式下常量显示为绿色。如果这个选项是未激活的，不能对常量进行 强制, 写 和 监控 。通过变量访问把这个值加载到存储区（这实际上允许写变量值，但意味较长的处理时间）Nested comments（嵌套的注释）：注释可以放在其它的注释里面。例如：

```
(*
a:=inst.out; (* to be checked *)
b:=b+1;
*)
```

这里注释从第一个括号开始，没有在"checked"后的括号结束，而是在最后一个括号结束。

创建应用程序的二进制文件：在编译的过程中在工程目录中创建一个文件名：<project_name>.bin 二进制代码文件（导入工程）。通过命令 '联机' '创建引导工程'在控制器上建立导入工程。

注意：如果一个已有的工程是打开的，这个工程是由先前版本的 OtoStudio 创建的，选项将会失效。但是先前有效的层次（局部变量在全局变量之前在程序之前在局部动作之前）能够保持。

数据块数目：这里可以为 PLC 中的工程数据定义应该分配多少内存段。即使是新变量加入，这个空间也能保证使联机交换成为可能。如果在编译过程中，得到信息“超出全局数

据内存……”，输入一个较高数字，此时这个局部程序变量会当作全局变量来处理。

移除对象：这个按钮打开对话从结构中排除：在工程组件的树形图中选择那些在编译过程中不被考虑的 POU，激活选项排除，在这里排除的 POU 在选择树上将会显示为绿色。如果你只想显示在程序中当前使用的那些 POU，按按钮排除不使用的。

编译器版本：这里说明将要用到的编译器版本，OtoStudio 在 V2.1 后的版本除了包含目前的编译器版本，还包含了先前的版本。如果你想在任何情况用当前的版本编译工程，激活选项最近使用。如果工程需要用一个特殊的版本编译，通过在装置的选择列表中定义它。

自动检查：为了在每个工程的编译时得到语义上的正确性，可以激活下列选项：

未用的变量

重叠存储区

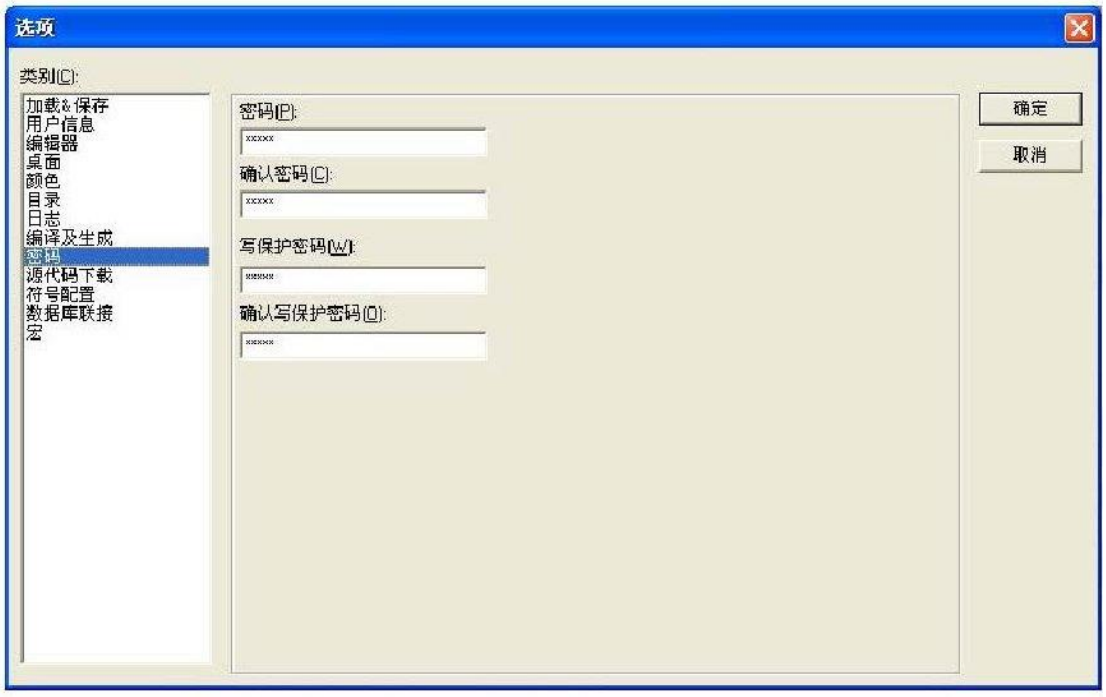
并行访问

在输出端多通道写访问

结果将会在消息窗口显示，这些检查也能通过菜单“工程”的子菜单“检查”的命令激活。

5.2.9 密码

如果在选项对话框中选择了这个类别，将会出现如下的密码选项对话框：



为了保护你的文件不受非授权的访问，OtoStudio 提供了密码选项来保护文件的打开和更改。

在密码区域输入你期望的密码。对每个敲入的字符在区域中出现一个*。在确认密码区域中必须重复输入相同的字符，按 OK 键关闭对话框。如果出现消息“密码和确认密码不相符”，那么在两个条目中的一个中输入有错误,在这种情况下在重复在两个条目输入直到关闭时不出现消息为止。

如果你现在保存文件，然后重新打开它，你会看到一个要求你输入密码的对话框，只有在正确输入密码的情况下才能打开工程，否则，OtoStudio 报告“输入密码不正确”。

随着打开文件，你也可以用密码来保护你的文件不被修改。你必须在写保护密码区域设置一个密码并在下面的区域确认。

当打开一个文件时，如果 OtoStudio 要求输入写保护密码，如果按退出按钮，可以不需要密码打开一个写保护的工程。现在你可以编译工程，加载到 PLC 中，模拟，等等，但是不能改变它。

当然记住两个密码是很重要的，但是，如果忘记了一个密码，可以联系你的 PLC 制造商。

密码和工程一起存储。

为了创建不同访问权限你可以定义用户组和 用户组密码。

另外可以通过加密的方式保护一个工程，例如如果不正确输入密码，此库不可使用。

注意:设置保存在工程中。

5.2.10 源代码下载

如果在选项对话框中选择了这个类别，将会出现如下的对话框：

可以选择定时或者多大范围，把工程文件加载到控制器系统。选项只下载源代码只包含 OtoStudio 文件扩展*.pro 的文件，选项全部文件也包含相关库文件、可视化位图、配置文件等等。

选项根据选择下载允许在使用命令“联机” '下载'时自动加载被选择的文件范围到控制器系统中。选项根据选择文件创建引导工程允许在使用命令“联机” “创建引导工程”时自动加载被选择的文件范围到控制器系统。

选项下载出现提示对话框提供一个对话，当命令“联机” '下载'是给定，将出现“你想写源代码到控制器系统吗？”。按是按钮将会自动加载被选择的文件范围到控制器系统，或者你可以选择 No 来放弃。

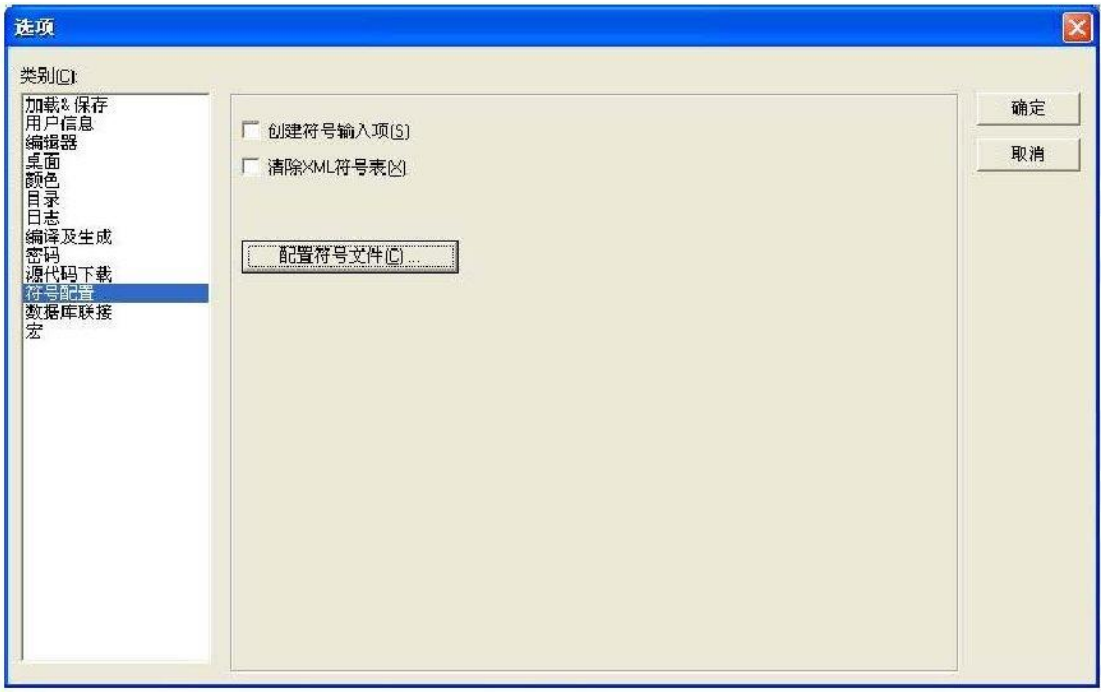
若使用选项需求，则必须通过命令 '联机' '源代码下载'加载被选择的文件范围到控制器系统中。在控制器系统存储的工程可以通过使用命令 '文件' '打开'从 PLC 中重新打开，文件在处理过程中解包。

注意:设置保存在 OtoStudio 中。

5.2.11 符号配置

这里提到的对话框是用于配置符号文件。这会在工程目录中创建文本文件<工程名称>.sym，二进制文件<project name>.sdb（依赖于在用的网关版本），符号文件是为通过符号接口和控制器进行数据交换和用作那个目的，例如，网关 DDE 服务器。

符号配置选项对话框



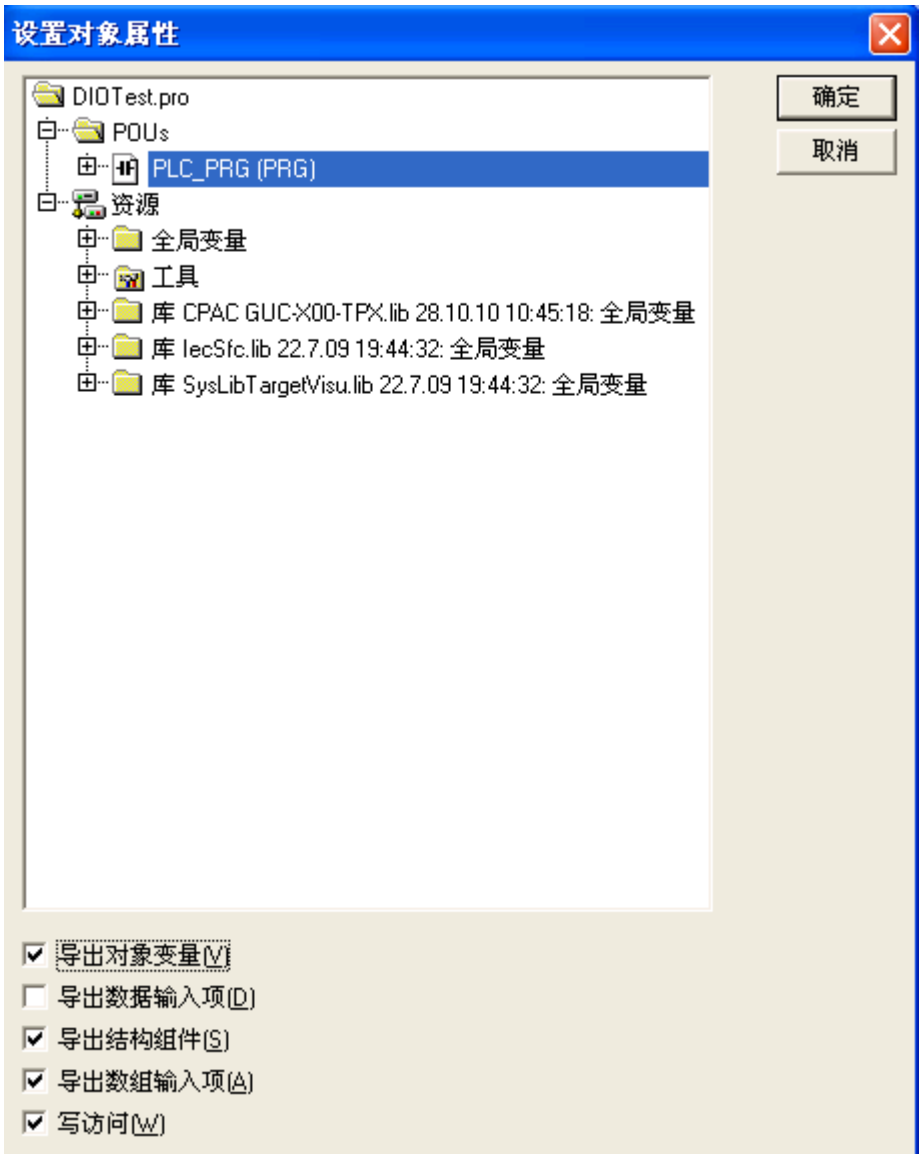
如果激活选项创建符号条目，那么工程变量的符号条目在工程的每个编译时在符号文件中自动创建。

如果激活附加的选项创建 XML 符号表格，那么在工程目录中也创建一个包含符号信息的 XML 文件，它被命名为<project name>.SYM_XML。

当配置符号条目时请注意以下信息：

a.如果在目标设置中激活了选项'INI-file 符号配置'，那么符号配置将会从 OtoStudio.in 中读出或在这里定义的其它的 ini 文件中读出（此时，在 OtoStudio 中不能编辑对话“设置对象属性”）；

b.如果选项“INI-file 符号配置”没有激活，将会产生与“设置对象属性”对话设置一样的符号条目，使用按钮设置符号文件能完成这些：



使用树状结构的选择编辑器来选择工程 POUs 和通过在相应的小框单击鼠标来设置对话框下面的期望的选项，选中的选项是激活的，可以设置下列选项：

导出对象变量：选择对象的变量在符号文件中导出。

下面的选项只有导出对象变量激活才起作用：

导出数据输入项：为对象的结构和数组访问全局变量创建的条目。

导出结构组件： 为每个对象的结构变量组件单独的创建的一个条目。

导出数组输入项： 为每个对象的数组的变量组件单独的创建的一个条目。

写访问：通过 OPC 服务器可能会改变对象的变量。

一旦完成当前选择的 POU 设置，其它的 POU 也能够在关闭对话之前被选中，并且打开一个选项配置，这能完成任意多 POU 的选择，一个接一个，当通过 OK 关闭对话框时，所有的配置从对话框打开开始应用。

注意:设置保存在工程中。

5.3 管理工程

这里讲述菜单项目“文件”和“工程”下包含的命令。

5.3.1 文件->新建

符号： 

通过这个命令可以创建一个名为“Untitled”的空白工程，在保存的时候必须修改这个名字。

5.3.2 文件->从模板中新建

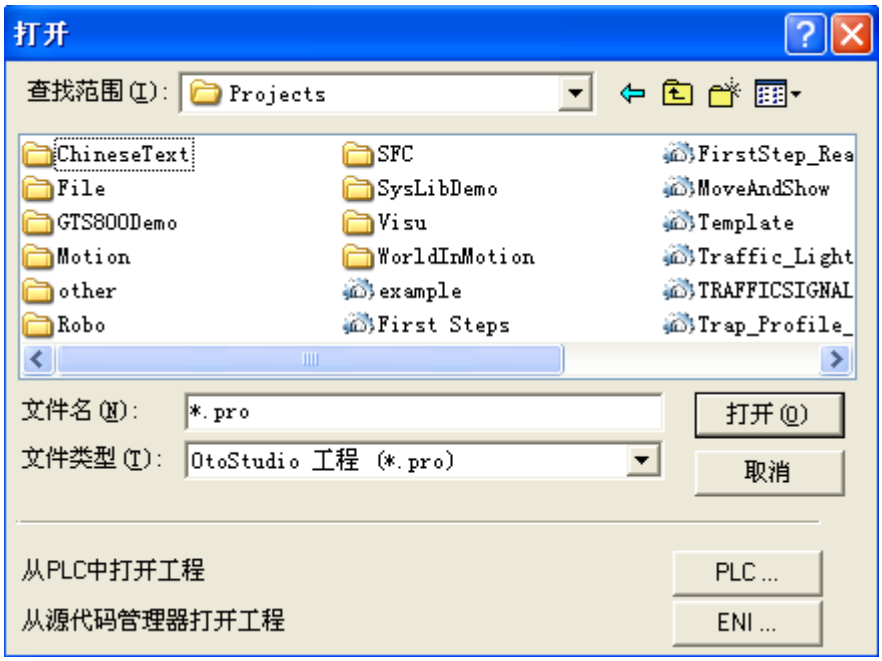
使用这个命令打开一个 OtoStudio 工程作为“模板”工程，打开工程文件对话框弹出，选中的工程以“Unknown”名字打开。

5.3.3 文件->打开

符号： 

使用这个命令可以打开一个现存的工程，如果一个工程已经打开和修改了，那么 OtoStudio 会询问是否要保存这个工程。

打开文件的对话框出现，必须选择一个带扩展名“*.pro”工程文件或一个带扩展名“*.lib”库文件 ，这个文件必须已经存在。通过命令“打开”是不能创建一个工程。



从 PLC 中打开一个工程

在从 PLC 中打开工程文件时，按“PLC”键，就可以从 PLC 中上传一个工程文件。下一步，出现一个通讯变量对话框（请参看“联机”“信息参数”菜单） 用以没有进行 PLC 连接时的传输变量设置。一旦联机成功，系统可以检测到具有相同命名的工程文件存在于你的

计算机硬盘目录中。当从控制器加载工程的对话框出现时，你可以决定是否用控制器使用的工程替代本地文件（这个顺序与“联机”“加载源代码”的顺序是相反的。工程源文件保存在控制器中。不要与“创建引导工程”相混淆）。

注意：在任何情况下必须给工程一个新名字，当你把它从本地目录加载到 PLC 时，否则它是没有名称的。如果目标系统支持，输入到工程信息中的'Title'将会预定义为新工程文件名。在这个情况下从 PLC 中调用工程时，保存文件的对话框将会打开。新文件名是自动添加的并且可以确认或修改。

如果还没有加载工程到 PLC，会得到一个错误信息。

最近打开的文件

最近打开的文件在文件菜单中在命令“文件”“退出”的下边列出。如果你选择它们中的一个，相应的工程将打开。

如果为工程定义了 用户组或 密码，会出现一个要求输入密码的对话框。

5.3.4 文件->关闭

用这个命令可以关闭当前打开的工程，如果工程被修改了，OtoStudio 会询问是否保存这些变化。如果要保存的工程名字为“Untitled”,必须赋予它一个名字（参照 '文件' '另存为'）。

5.3.5 文件->保存

符号： 

快捷方式 <Ctrl>+<S>

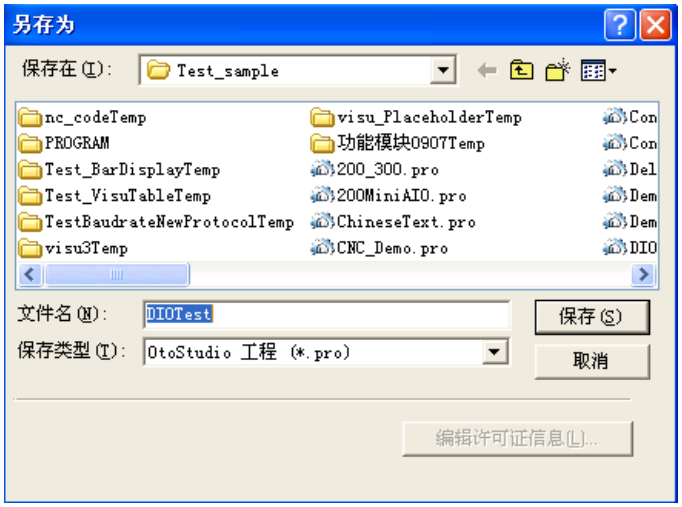
用这个命令可以保存工程中的任何变化,如果要保存的工程名字为“Untitled”， 你必须给它一个名字。

5.3.6 文件->另存为

用这个命令当前工程可以保存为另外的文件或存为一个库文件，这不会改变原工程文件。

在命令选择之后，出现保存对话框，选择一个存在的文件名或输入一个新文件名，并选择合适的文件类型。

“另存为”对话'



如果工程文件以一个新文件名保存，还应选择文件类型 OtoStudio Project (*.pro)。

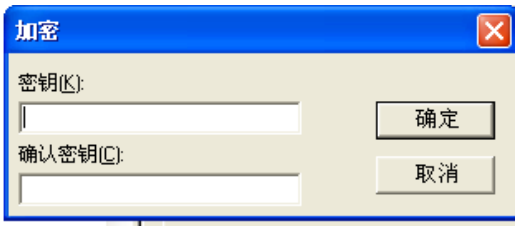
如果你选择了文件类型 Project Version 1.5 (*.pro), 2.0 (*.pro), 2.1 (*.pro) or 2.2 (*.pro)，当前的工程保存为先前版本 1.5, 2.0, 2.1 or 2.2 创建的，版本 2.3 的特殊的数据可能因此丢失，然而，工程可以在版本 1.5, 2.0, 2.1 or2.2 上运行。

为了在其它工程中使用它，也可以把当前工程保存为一个库文件，如果你已经在 OtoStudio 中编写了 POUs，选择文件类型自带库 (*.lib)。

如果你想运行程序，或以其他语言（例如 C 语言）集成 POUs，就可以把文件类型选择为扩展库 (*.lib)。这意味着其它文件也可以使用库文件名保存，但是扩展名必须是“*.h”。这个库文件的构造是一个 C 语言头文件。它保留对所有程序组织单元（POU）、数据类型和全局变量的声明。如果使用外部库，在仿真模式时，可以对 OtoStudio 中程序组织单元（POU）执行写操作。在 C 语言中编写的执行程序将会与硬件一起工作。

如果将工程保存成加密的工程或库文件，请选择加密 OtoStudio 工程(*.pro)、加密内部库(*.lib)或加密外部库(*.lib)，在这种情况下，将弹出‘加密’对话框，你可以定义和确认访问密码。设置密码后如果没有密码，您将无法打开它。

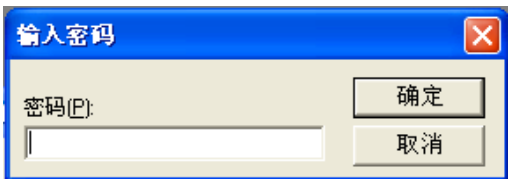
工程加密对话框



加密功能增强了工程的保护性，到目前为止，只能借助写保护密码访问工程，这种方法将一直沿用，注意如果不输入库密码，那么不能将一个库插入到工程中。

定义的密码将一直保存在工程中，如果要修改它，请再次使用“另存为”对话框进行设置。如果打开加密工程或插入加密库到工程中，将显示输入密码对话框。

输入密码对话框



在做完所有的设置后，按是，当前工程将会保存到指定的文件中，如果新文件名已经存

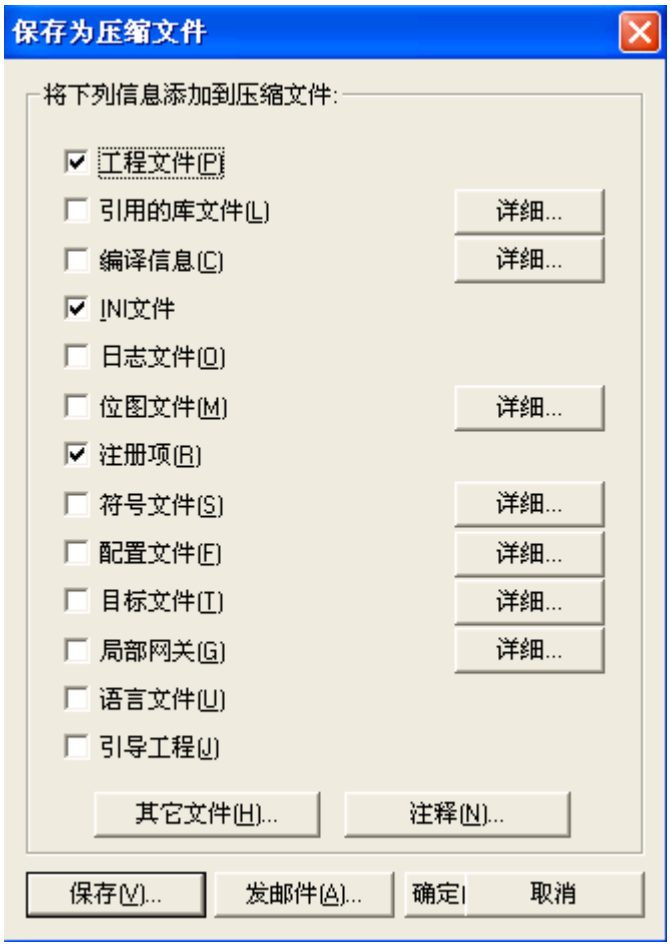
在，那么会询问你是否想覆盖这个文件。

当工程文件存为一个库文件时，要对整个工程文件进行编译。编译中如果出现错误，将会被告知创建库文件的工程必须是正确的，工程不能存为库文件。

5.3.7 文件->保存/发送压缩文件

这个命令用来建立和创建一个工程存档文件，OtoStudio 工程使用的和引用的所有的文件能被压到一个压缩包中，压缩文件能在 email 中存储或直接发送。如果你想传送一套工程的相关文件，这是非常有用的。

当命令执行时，打开保存存档文件对话框：



这里可以定义把那些文件类别加入到压缩档案文件中：通过激活/取消相应的检查框选择或取消选定一个类别，在检查框上单击鼠标或在类别名上双击鼠标。如果一个类别被标记了，这个类别的所有文件将会添加到压缩文件中。按动相应的按钮，可以进行同一个类别中的单个文件的选择。

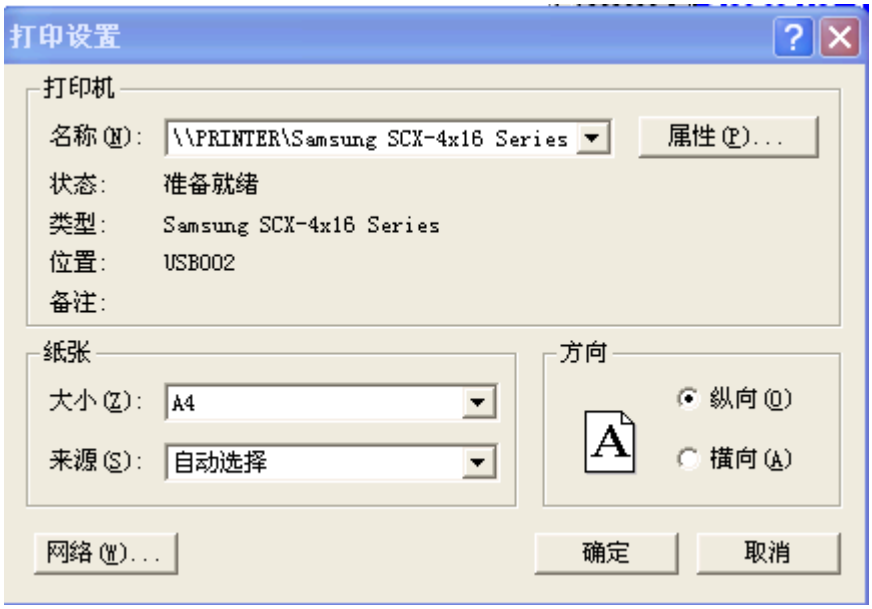
5.3.8 文件->打印

快捷方式：<Ctrl>+<P>

用这个命令打印活动窗口的内容。

在选中命令后，接着出现打印对话框，选择期望的选项或配置打印机然后按 OK，就可以打印活动窗口，可以打印出编辑器中的所有颜色。

打印设置对话框



可以选定文件打印的份数，以及把打印内容输出到一个文件。

还可以通过“属性”按钮，设置打印机。

使用 打印设置，你可以设置打印输出方式。

在打印过程中，对话框显示打印完成的页数，如果关闭此对话框，在下一页打印完成后打印中止。

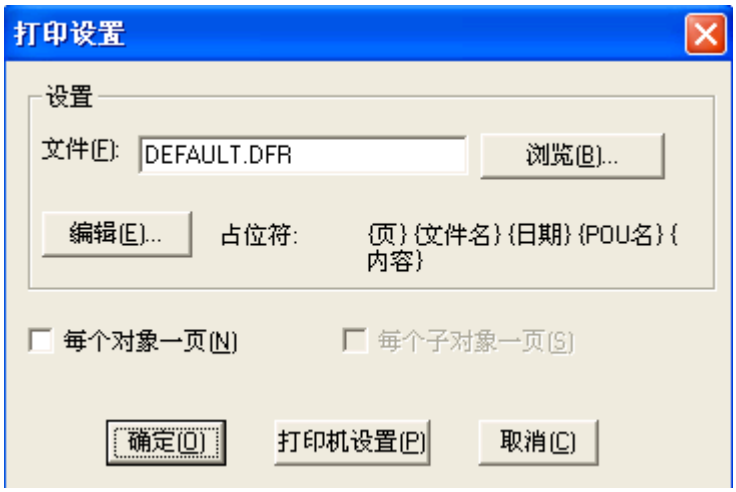
使用命令 '工程''文档' 可以把整个工程文档化。

如果想为工程创建一个文档框架，在这里可以保存关于工程中所用变量的注释，那么打开一个全局变量列表和使用命令“附加”“创建文档框架文件”。

5.3.9 文件->打印设置

用这个命令可以决定打印页的版面，出现下面的对话框：

页面设置对话框



在文件区域，输入要保存的版面中带扩展名".dfr"的文件的名字，默认的目的文件设置是文件 DEFAULT.DFR。

如果你想改变目前的版面，用按钮浏览在目录树中浏览寻找期望的文件，也可以选择一个项目或者一个子项目为一个新页面。使用打印机设置按钮来打开打印机设置页面。

页面占位符设置

用菜单项目“插入”“占位符”和并发的选择在五个占位符(Page、 POU name、 File name、 Date andContent)，通过在版面上按鼠标左键拖动一个矩形插入一个所谓的占位符到版面，在打印输出它们被替换为下面：

命令	占位符	作用
页面	{页面}	当前页码出现在打印输出中
POU 名	{POU 名}	显示当前 POU 名
文件名	{文件名}	显示工程名
日期	{日期}	显示当前日期
内容	{内容}	显示 POU 内容

另外，用“插入”“位图”命令可以在页面中插入一个位图（如公司的标志等），在选择图形之后，在版面上使用鼠标可以在这里画一个矩形，也能插入其它的可视化组件（参照可视化）。

如果模板改变了，在窗口关闭时，OtoStudio 会询问是否保存这些变化。

5.3.10 文件->退出

快捷方式：<Alt>+<F4>

用这个命令你可以从 OtoStudio 中退出。

如果一个工程已经打开，那么它会象 '文件' '保存'中讲述的那样关闭。

5.3.11 工程->编译生成

快捷方式：<F11>

使用“'工程' '编译生成' ”来编译工程，编译过程基本上是补充增加式的，当保存工程时，最后一次编译所需的信息保存在 ci-file 中。只有改变的 POUs 才重新编译，如果首先执行命令“工程” “清空”，能得到一个非增量编译。

对于联机修改的目标系统，所有装入控制器的 POUs，在编译时，在对象管理器被标上一个蓝色的箭头。

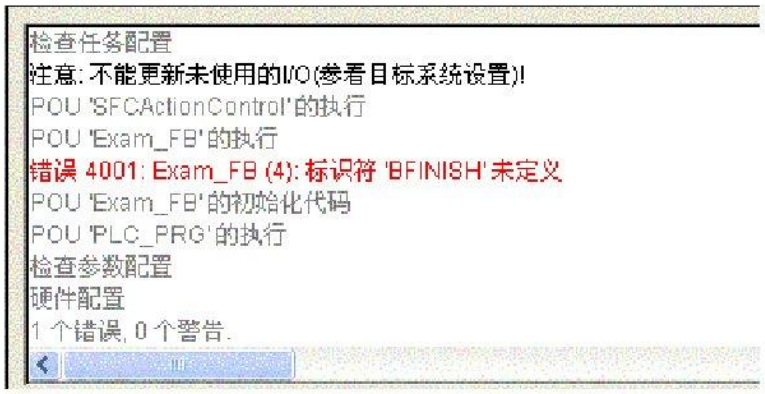
如果控制器通过 '联机' '登录'登录，随着“'工程' '编译生成”编译过程自动执行。

参看此处 图表，它表明了工程－编译生成、工程－下载、在线修改和登录到目标系统之间的关系。

在编译过程中，消息对话框显示了编译的进程、在编译过程中可能发生任何错误和警告和在使用中的 POU 的索引和内存空间（数字和百分比）的信息。错误和警告用数字标记，使用 F1 可以得到当前选择的错误的更多信息。

参照所有变量 出错信息和警告的列表。

在信息窗口中的错误报警和编译信息:



如果在加载 & 保存 类别中的选项对话中选择了选项编辑前保存，工程在编译之前先保存。

注意：交叉引用创建在编译过程中并和编译信息一起保存。为了在菜单“工程”“检查”中能使用命令'显示调入树'、 '显示交叉参考' 和命令 '未使用变量'、 '并行访问'和 '多重输出' ，工程必须在改变后重新建构。

5.3.12 工程->全部重新编译生成

用“'工程' '全部重新编译生成”，不象增量编译('工程' '编译生成')那样，工程将彻底重新

编译。

5.3.13 工程->全部清除

用这个命令，删除来自上次下载和来自上次编译的所有信息。

在命令选择后，出现一个对话框，报告没有新下载的登录将不再有效。这里，命令既可以取消也可以确认。

注意:在执行"全部清除"命令后,，对 PLC 工程进行在线修改只有在下述的情况下可以进行：即在执行全部清除以前已经重新命名了工程目录中的*.ri 文件(保存上次下载信息)或已经将此文件保存到工程目录以外 (参看 '加载下载信息')，在登录前先加载它。

5.3.14 工程->加载下载信息

使用这个命令能卸载工程的下载信息，如果它和工程不在同一个目录中存储，在选择命令后，会打开标准对话框“文件打开”。

下载信息在下载时自动保存，它依赖于目标系统，潜在地在每次离线时在文件中创建一个导入工程，文件命名为<工程名><目标标识符>.ri 并放置到工程目录中。每次工程重新打开时这个文件重新加载，在登录时它用来检查那个 POU 的代码已经改变，在联机交换过程中只有这些 POUs 随后加载到 PLC 中。

注意:在执行"全部清除"命令后,，附属于当前工程的*.ri 文件将从工程目录中自动删除.只有在执行全部清除前重新命名此文件或将文件保存到其它地方,然后在登录前通过此命令加载才能实现在线修改.

5.3.15 工程->文档

这个命令能打印整个工程的文档，一个完整的文档包括以下组件：

程序组织单元（POU）

文档的内容

数据类型

可视化

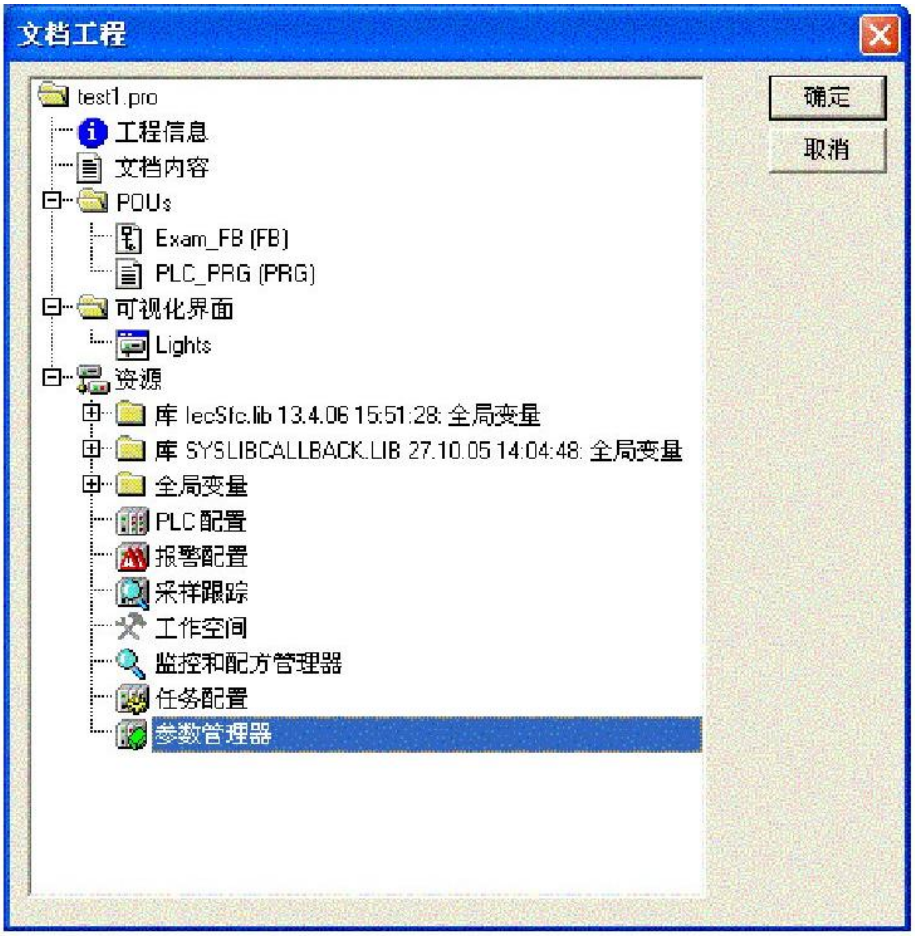
资源、全局变量、变量配置、采样追踪、PLC 配置、任务配置、查看和配方管理器。

程序组织单元和数据类型的 调用树

交叉参考列表

对最后两个条目，工程编译时必须没有错误。

工程文档对话框



只有在对话框中以高亮度蓝色表示的那些区域才能打印输出。

在第一行选择的工程的名字就能选择整个工程。

如果，另一方面，你只想选择一个对象，那么在相应的对象上点击或用方向键移动带点的矩形框到期望的对象上。对象在它们的符号附近有加号的是组合对象包含了其它的对象，在加号上单击对象展开，在减号上点击它有关闭。当你选择一个组合对象，那么所有相关的对象也被选择。通过按<Shift>键你能选择一组的对象，通过按<Ctrl>键你能选择几个不连续的对象。

一旦做好了选择，按 OK。打印对话框出现。你能用 '文件' '打印机设置'决定要打印的页的版面。

5.3.16 工程->导出

工程能导入和导出，这允许在不同的 IEC 编程系统之间交换程序。

在指令表、结构化文本和顺序功能图表（IEC 1131-3 功能组件格式）中的程序组织单元有一个标准化的交换格式，在梯形图和功能模块图中的程序组织单元和其它对象，OtoStudio 有它自己的归档格式选择的对象写到了一个 ASCII 文件。

POUs、数据类型、可视化和资源能被导出。另外，库文件管理器中的条目也就是库的连接信息也能导出（不是库文件）。

重要：如果在图形编辑器注释包含一个引用标记(')，重新导入一个导出的 FBD 或 LD

POU 会导致错误，因为这个被解释为字符串的开始。

在对话框窗口做出了选择（和用 '工程' '文档'有相同的情形），此时能决定导出选择的部分到一个文件还是导入到多个文件中，一个对象一个文件，切换选项 One file for each object 开或关，然后按 OK 按钮。

出现保存文件的对话框，输入一个带扩展名“.exp”的文件名字，为对象导出文件分别存放于目录中，它将以文件名<对象名.exp>保存在这里。

5.3.17 工程->导入

在结果对话框中打开选择期望的 导出文件对话。

数据导入到当前的工程，如果在同一的工程中存在一个与对象相同的名字，那么出现一个对话框询问“你想替换它吗”：如果你回答 Yes，那么在工程中的对象被导入文件中的对象替换。如果你回答 No，那么接收新对象的名字，且补充一个下划线和一个数字("_0", "_1", ..)。

如果导入的信息同一个库链接，库将被加载和附加到库文件管理器中的列表的最后。如果库已经加载到了工程，它不会被重新加载。然而，如果导出文件正在被导入，它显示库中不同的存储时间，库文件名在库文件管理器中用一个"*"标记（例如，standard.lib*30.3.99 11:30:14），和加载一个工程相似，如果不能找到库，那么出现一个信息对话：“不能找到库{<路径>\}<名称> <日期> <时间>”，与当一个工程被加载时一样。

导入的信息在 信息窗口提示。

5.3.18 工程->导入西门子程序及变量

在子菜单“导入西门子工程”中，能找到导入 POUs 和 Siemens-STEP5 文件中的变量的命令。

可用到下面的命令：

·“导入一个 SEQ 符号文件”

·“导入一个 S5 文件”

20）工程->比较

对于工程中的所有单元，不同处用颜色改变来标记。对于当前打开的工程，可以接受比较工程的结果（这种接受是单向的，从比较工程到当前工程）。

为了接受某个单元的改变，请使用‘接受单独改变’指令。

这个命令用来比较两个工程或比较一个工程的目前的版本和最后保存的版本。

定义：

当前工程：当前使用的工程；

引用的工程：与当前工程比较的工程；

比较模式：工程会在“工程->比较”执行后被显示出来；

单元：参与比较的最小单元。可以是一条直线（声明编辑器，ST 编辑器，IL 编辑器），可以是一个网络（FBD 编辑器，LD 编辑器）或是一个元素/POU（CFC 编辑器，SFC 编辑器）。

在比较模式中当前的工程和引用的工程将会出现在一个双向的窗口，发现差别，POUs

的名字被用颜色标记，在编辑器中 POUs 的内容和 POUs 以面对面的方式显示。在比较模式下结果和呈现的方式依赖于：1.为比较运行激发了什么过滤条件，这影响在比较过程中对空白和注释的考虑 2.在行或网络或组件内修改是否被认为插入一个完全新的 POU。

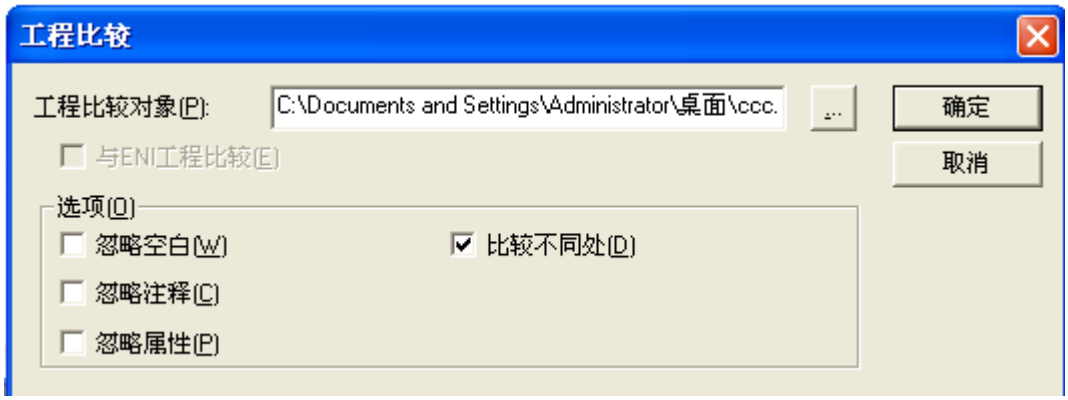
请注意：在比较模式（查看状态栏：COMPARE）工程不能被编辑！

参看：

- 执行比较
- 比较结果的表示方法
- 工作在比较模式下

执行比较

在执行命令“工程->比较”后，打开对话框工程比较：



在工程比较对象处插入 引用工程的路径， 如果你想使用标准对话框来打开一个工程 j 进行比较，按下按钮  。如果你插入当前工程的名字，工程的当前版本将会和它最后保存的版本比较。

如果工程在 ENI 数据库里在源控制下，那么本地版本可以和在数据库中找到的当前版本比较。要这么做需激活选项 ENI 工程比较 。

可以激活比较中的下列选项：

忽视空白:检测到空白的地方视为没有区别，也就是忽略空白。

忽视注释:对注释视为无区别

忽视属性:对对象的属性视为无区别

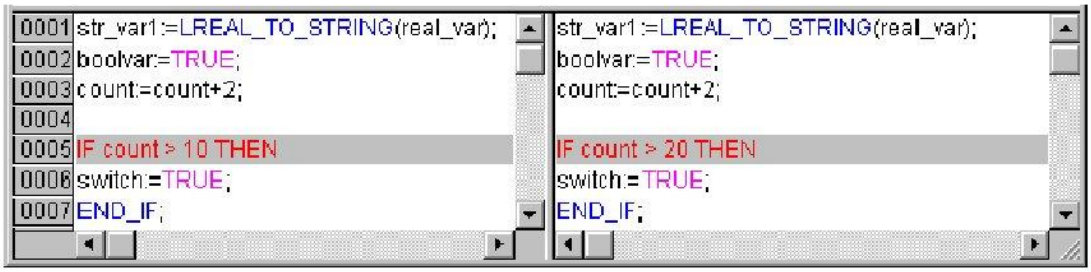
比较不同处:如果在一个 POU 内的一个行、一个网络或一个元件已经被修改，在比较模式中它将在双向的窗口显示并直接与其它工程（标记为红色，查看下面）的版本相对。如果选项是未激活的，相应的行将会作为“删除”显示在引用工程中，作为“插入”显示在当前的工程中（蓝/绿，查看下面）。这意味着它不会显示为直接地与其它工程中的相同的行对立。

例如：在当前工程中（左边）0005 行已经被修改。

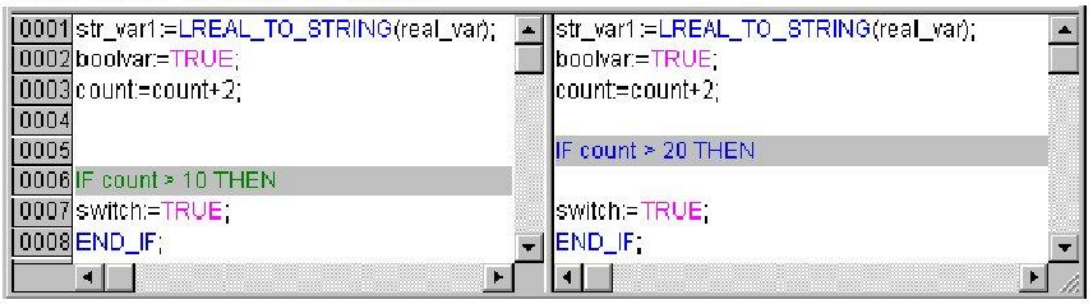
结果图

Example for "Oppose differences"

Option 'Oppose differences' activated:



Option 'Oppose differences' not activated:



当按 OK 关闭对话“工程比较”时，将根据设置执行比较。

参看:

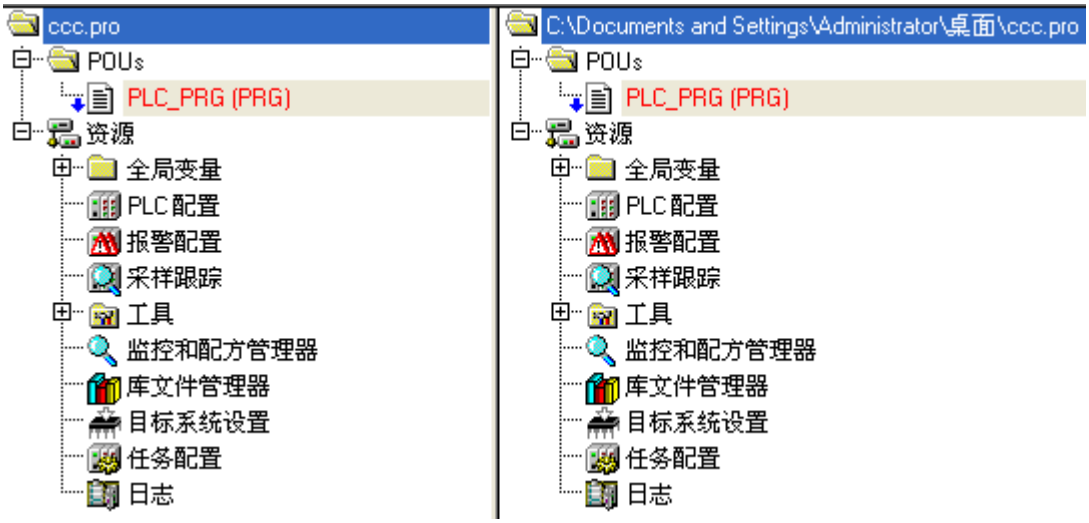
- 比较结果的表示方式
- 在比较模式下工作

比较结果的请求

首先将会打开标题为“工程比较”工程的结构树，来显示比较的结果，这里你能选择特殊的 POUs 来查看详细的区别。

1.比较模式中工程的概览

在工程比较之后，打开一个双向的窗口，它显示了在比较模式下的工程。在标题栏中显示工程的路径：“工程比较<当前工程路径>- <参考工程路径>”。当前的工程是在窗口的左半部分表示，引用的工程在右边表示。每个结构树在最上面显示了工程的名字，它符合对象管理器的结构。



不同的 POUs 在结构树中通过阴影、特殊的颜色和附加的文本来标识：

红:单元已经被修改；它在两部分窗口中用红颜色字母显示

蓝:单元只在引用的工程中可用；在当前工程的预览结构中相应的位置插入一个间隙。

绿:单元只在当前工程中用到；在当前工程的预览结构中相应的位置插入一个间隙。

黑:在没有发现区别的地方用到。

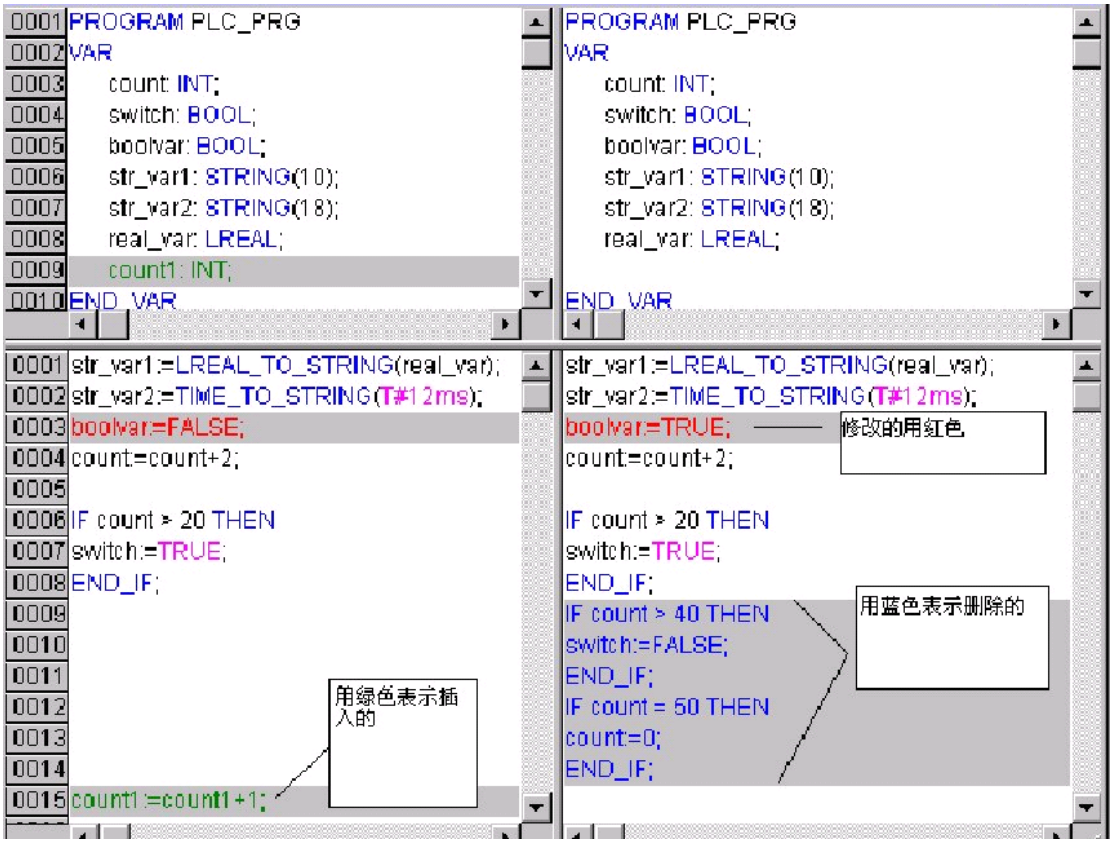
“(属性差别)”：如果检测到了 POU 的属性差别，在工程结构树里这个文本附加到 POU 文件名后。

“(进入正确的变化)”: 如果检测到了 POU 的访问权限的差别，在工程结构树里这个文本附加到 POU 文件名后。

2.比较模式下 POU 的内容

如果它是一个文本或图形编辑器 POU，将会在一个双向窗口中打开。引用工程的内容（右边）与当前工程的内容（左边）是相对应的，在比较过程中最小的单元是一行(declaration editor, ST, IL)、一个网络 (FBD,LD)或一个元素(CFC, SFC)，使用与上面工程概览描述的一样的颜色。

例如，在比较模式下的 POU：



如果它不是编辑器 POU，而是任务配置、目标设置等编辑器，可以在工程结构上各个行上双击鼠标，打开当前工程和引用工程的 POU 版本。此时不显示那些工程的 POU 详细的区别。

参看：

- 在比较模式下工作

在比较模式下工作

把鼠标光标放在双向窗口中的表示不同内容的某一行上，就会出现菜单“附加”。根据工程浏览或者 POU 的不同，菜单可以提供以下指令的选择：

'下一个不同处'

'前一个不同处'
'接受改变'
'接受改变项目'
'接受属性'
'接受访问权限'

附加->下一个不同处

快捷方式：<F7>

这个命令在比较模式下是可用的（查看上面 '工程''比较'）。

光标跳到下一个单元，在这里指明了一个差别（在工程概览中的行，在 POU 的行/网络/元件）。

附加->前一个不同处

快捷方式 ： <Shift><F7>

这个命令在比较模式下是可用的（查看上面 '工程''比较'）。

光标跳到上一个单元，在这里指明了一个差别（在工程概览中的行，在 POU 的行/网络/元件）。

附加->接受改变

快捷方式 ： <Space>

这个命令在比较模式下是可用的（参看上面的 '工程''比较'）。

对于所有的有相同的类别,不同标记的相连接的单元，参考工程的版本被当前的工程接收。相应单元(带颜色)将显示在窗口的左侧。如果它是一个标记为红色的单元， 在当前工程中的接收标记为黄色。

使用'接受改变项目'来接受特定单元的修改.

注意: 对于接受工程之间的不同处或访问权限属性,只能是当前工程接受参考工程的.

附加->接受改变项目

快捷方式 ： <Ctrl> <Spacebar>

这个命令在比较模式下是可用的（查看上面 '工程''比较'）。

只有光标当前位于单个单元（行，网络，元件）上，才接受当前的版本，相应的单元(带颜色的)将显示在窗口的左边.

对于一个 POU,如果在树型结构中标记为红色,则说明它的内容修改了;如果接受改变,则在当前工程中的 POU 标记成黄色.

附加->接受改变属性

这个命令在比较模式下是可用的（查看上面 '工程''比较'）。

执行此命令,光标位置处的 POU 的对象属性将被设置成参考工程的。

附加->接受访问权限

这个命令在比较模式下的工程概览中才有用：（参看上面的 '工程''比较'）。

执行此命令,光标位置处的 POU 的访问权限将被设置成参考工程的。

5.3.19 工程->合并

使用这个命令，可以进行各种对象（POUs、数据类型、可视化和资源）的合并，还可以把其它工程文件导入到你工程文件内。

当执行这个命令时，首先出现打开文件的标准对话框。在这个对话框中选择一个文件后，又出现一个对话框。在这个对话框里，你可以象“'工程' '文档'中讲述的那样选择期望的对象。如果在工程文件中存在与对象相同的名称，则在新对象的名称后附加下划线和数字(如"_1", "_2"等) 。

5.3.20 工程->工程信息

在这个菜单条目下可以存储工程的信息，当给出命令后，打开下面的对话框：
工程信息对话框

显示下列各项信息：

文件名

目录路径

最近改变的时间（改变日期）

另外，你可以添加下列的信息：

工程的标题：请注意，如果目标系统支持，当通过命令“文件” “从 PLC 中打开工程”加载工程时（在这种情况会打开一个保存文件的对话），这个标题自动作为工程文件名。

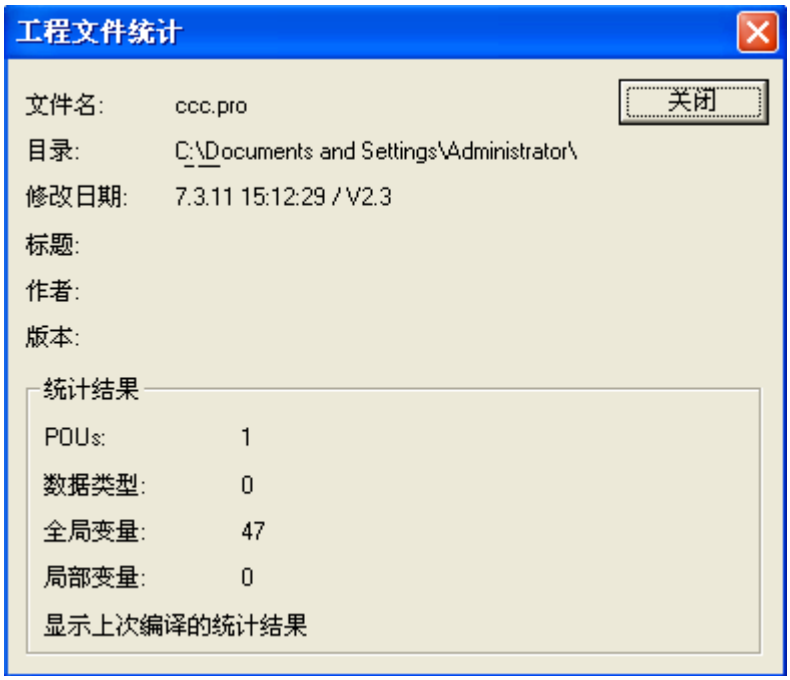
作者的名字，

版本号

·工程的描述。

这个信息是可选的，当你按按钮 Statistics 时你可以接收工程的统计信息。它包含了信息比如与它们在上次编译被追踪一样的 POUs 数量、数据类型和局部和全局变量。

工程统计结果



当通过命令 '文件' '另存为...'处理已经存储的带有许可信息的 OtoStudio 工程时，按钮许可证信息可用。

在这种情况下按钮打开对话“Edit Licensing Information”，在这里你可以修改或删除许可

如果你选择了选项对话框中的类别加载 & 保存中的选项询问工程信息，当保存一个新工程，或在一个新名字下保存一个工程时，自动的调用工程信息对话。

5.3.21 工程->全局搜索

使用这个命令，可以搜索 POUs 中的文本、数据类型或在全局变量的对象里的位置。

当输入命令后，会打开一个对话，在这里，可以象在 '工程' '文档'描述的那样来选择期望的对象。

按 OK 键确认了选择后，会带开搜索的标准对话，当通过使用在菜单栏中的符号 i. 来调用“全局搜索”时会迅速出现这个对话；在工程中的所有搜索范围内自动执行搜索。可以通过搜索区域的组合框来选择最新输入的搜索字符串，如果在一个对象中找到一个文本字符串，被加载到相应的编辑器或加载到库文件管理器和字符串被找到地方的位置将被显示。找到的文本的显示，与命令“编辑”“搜索”的搜索和搜索下一个功能的动作相似。

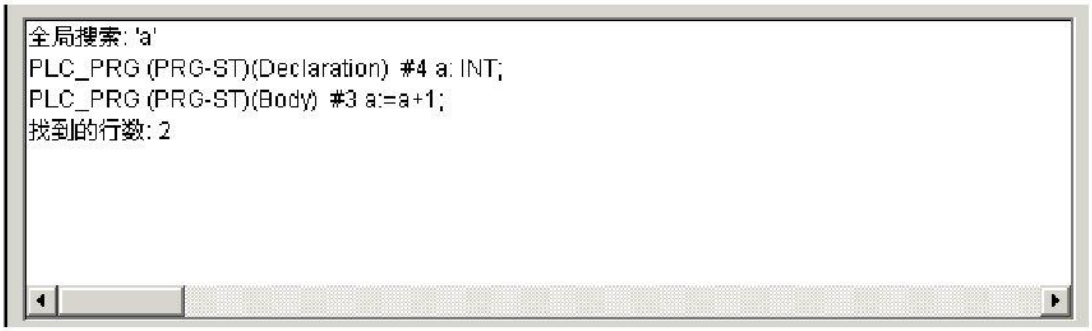
如果你选择了按钮信息窗口，在被选择的对象中的所有一系列的符号被搜索的地方在消息窗口将会一行一行的列出，然后，将会显示找到的位置的数量。

如果消息窗口没有打开，每个找到的位置显示下列各项：

对象名

POU 的声明或执行部分的位置

行和网络编号
文本编辑器中的全部行
在图形编辑器中完整的文本元素
信息窗口中显示搜索结果.



如果在信息窗口中在一个行上双击鼠标或按<Enter>键，打开加载对象的编辑器，对象中的相关行被标记，可以使用功能键<F4> 或 <Shift>+<F4>在显示行之间进行切换。

5.3.22 工程->全局替换

使用这个命令，可以进行 POUs 中文本、数据类型或全局变量的对象位置的搜索，并用其它的文本替换这个文本。执行这个命令和使用命令“'工程' '全局搜索'或 '编辑' '替换'效果相同。库文件不能进行选择，也不能在信息窗口中显示。

搜索结果在信息窗口显示。

5.3.23 工程->检查

这个命令用来检查工程的语义上的正确性，这个命令将考虑最近编译的状态，在测试执行以前，工程必须是没有错误的；否则菜单项目是灰色的。

打开一个列出下列命令的子菜单：

未使用的变量
内存重叠区域
并行访问
在输出端多通道写访问

结果在信息窗口显示。

请注意：在工程选项类别 '编译生成'中，你能定义它们以便在工程编译时自动做这些检查。

未使用变量

在菜单 '工程' '检查'中这个功能用来搜索在程序中已经定义的但还没有使用的变量，它们通过 POU 名和行表示，例如：PLC_PRG (4) – var1。不检查库中的变量。

搜索结果显示在信息窗口中。

内存重叠区域

在菜单 '工程' '检查'中的这个功能可以检查通过"AT"声明的变量在特殊内存区域中的

分配是否有重叠发生。例如，当分配一个变量“var1 AT %QB21: INT”和 “var2 AT %QD5: DWORD”时发生重叠，因为它们都使用 21 字节。检查结果如下：

%QB21 被下列变量引用：

PLC_PRG (3): var1 AT %QB21

PLC_PRG (7): var2 AT %QD5

检查结果显示在信息窗口中。

在输出端多通道写访问

菜单 '工程' '检查'中这个功能用来搜索内存区域中单个工程在多个地方进行写访问。检查结果如下：

%QB24 被写到下列地址：

PLC_PRG (3): %QB24

PLC_PRG.POU1 (8): %QB24

搜索结果显示在信息窗口中。

并行访问

菜单'工程' '检查'中这个功能用来搜索被多个任务引用的 IEC 地址的内存区域.写和读访问是没有区别的。例如：

%MB28 在下列任务中引用：

Task1 – PLC_PRG (6): %MB28 [read-only access]

Task2 – POU1.ACTION (1) %MB28 [write access]

结果显示在信息窗口中。

5.3.24 工程->用户组密码

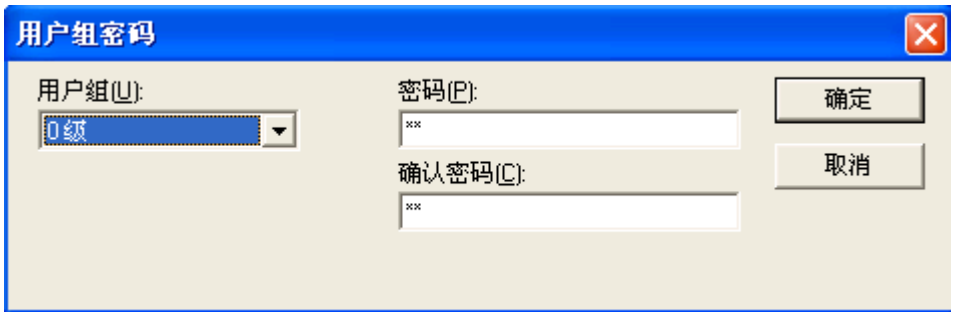
在 OtoStudio 中，能建立八个对 POUs、数据类型、可视化和资源有不同访问权限用户组，能建立访问单个对象或全部的对象权限的用户组，只有特定的用户组才能打开工程文件，可以通过密码来识别这样的用户组。

用户组从 0 到 7 编号，0 组拥有管理员权限，例如，只有 0 组的成员才能决定所有的组和/或对象的密码和访问权限。

当一个新工程开始，所有的密码初始化为空，直到为 0 组设置了密码，输入工程的成员自动被认为是 0 组的成员。

使用这个命令你可以打开对话框为 用户组设置密码，这个命令只能由用户组 0 组的成员执行，当给出命令时，出现下面的对话框：

输入密码对话框.



在左边用户组下拉选项中选择用户组，在密码区域为用户组输入期望的密码，对每个输入的字符在区域中显示一个*号。你必须在确认密码区域中重复输入相同的密码，在密码输

入后按'确定'关闭对话框，如果你得到消息：“密码和确认密码不一致”，那么这两个输入项不一致，在这种情况下重新输入它们。

如果必要，通过再次调用命令为下一个组分配一个密码。

注意：如果任何用户组都没有设置密码，那么任何用户组都可以打开此工程文件。

使用命令 '工程' '访问权限'对单个对象或全部对象来分配权限。

5.4 管理工程中的对象

管理工程中的对象

在以下的章节中，读者可以了解到如何进行对象的管理，以及在对象轨迹（文件夹、调用树、参考表等）上可以获得帮助等信息。

参照：

对象

文件夹

'新建文件夹'

'展开节点' '收缩节点'

'工程' '对象删除'

'工程' '对象添加'

'工程' '对象重命名'

'工程' '对象转换'

'工程' '对象复制'

'工程' '对象打开'

'工程' '对象访问权限'

'工程' '数据库连接'

'工程' '对象属性'

'工程' '添加动作'

'工程' '打开实例'

'工程' '显示调入树'

'工程' '显示交叉参考'

5.4.1 对象

POUs、数据类型、可视化和资源全局变量、变量配置、采样追踪、PLC 配置、任务配置和观察和接收管理器等，都可以被定义为“对象”。其中部分包括用于构造工程的被插入的文件夹。工程的所有对象都处于对象管理器之中。

鼠标光标在对象管理器中的一个 POU（包括程序、功能和功能块）上停留几秒，就会在工具条中出现有关 POU 类型的提示。对于全局变量工具条上将显示关键字 (VAR_GLOBAL, VAR_CONFIG)。

在一个对象类型内，可以通过拖拉进行对象（和文件夹，参看 '文件夹'）的移动。选择对象并按住鼠标左键，可以把对象移动到期望的地点。如果移动中出现导致名称冲突现象，新引进的元素将会通过附加一个连续的数字来进行标识（例如，“Object_1”）。

5.4.2 文件夹

为了进行大工程的管理，应该在文件夹中对 POUs、数据类型、可视化和全局变量进行分组。

可以创建任意多级别的文件夹。如果在一个收缩的文件夹  前面有一个加号，说明这个文件夹中包含对象和附加的文件夹。在加号上点击，就可以在打开文件夹的同时显示子对象。在减号（替代了加号）上点击，文件夹又关闭了。在文本菜单中也能找到相同的功能的命令 "展开节点' '收缩节点"。

通过拖拉，可以移动对象，连同在它们对象类型内部的文件夹。方法是选择对象并按住鼠标左键，把它拖放到期望的位置。

通过命令 '新建文件夹'，可以创建更多的文件夹。

注意：文件夹对程序没有影响，只为了清晰的表达工程。

对象管理器中的文件夹

5.4.3 新建文件夹

使用这个命令，可以以一个结构化对象的方式插入一个新 文件夹。如果已经选择了一个文件夹，就可在其下面创建新文件夹。否则就在同级创建新文件夹。如果选择了一个动作，新文件夹将创建到与 POU 同级的动作 POU 中。

对象管理器的文本菜单包含了这个命令，选择一个对象或对象类型时，按鼠标右键或<Shift>+<F10>出现这个命令。

最新插入的文件夹最初有名称“新建文件夹”，应遵守下面的文件夹命名的约定：

在层次中同级的文件夹必须有唯一性，在不同层次的文件夹可以具有相同的名字。

对象在相同层次时文件夹中不能有相同的名字。

如果在相同的层次上已经存在一个名为“新建文件夹”的文件夹，文件夹名字后面自动附加连续的数字（例如，"新建文件夹 1"），不能重命名为一个正在使用的文件夹名字。

5.4.4. '展开节点' '收缩节点'

用这个命令展开所选对象下包含的子对象,使它们可见。

使用收缩命令从属的对象将不再显示。

通过鼠标双击或按<Enter>键可以打开或关闭 文件夹 。

对象管理器的内容菜单中包含这个命令，当一个对象或对象类型被选中时并且你按了鼠标右键或<Shift>+<F10>时，这个命令出现。

5.4.5 对象删除

快捷方式 ： <Delete>

当前选中的对象（POU、数据类型、可视化或全局变量），或带从属对象的文件夹将从对象管理器中删除从而从工程中删除，对象的删除可以通过命令“编辑” “取消”来恢复。

删除的对象可以通过命令“编辑” “取消”来恢复。

如果对象的编辑器窗口是打开的，它自动关闭。

使用命令 '编辑' '剪切'来删除，对象被放置于剪贴板上。

5.4.6 添加对象

快捷方式：<Insert>

用这个命令可以创建一个新对象，对象（POU、数据类型、可视化或全局变量）的类型依赖于在对象管理器中被选择的选项，“数据类型”、“功能”、 “功能块”或“程序”类型的对象能使用一个 模板来创建。

在出现的对话框中输入新 POU 的名字，对象的名字可以是没有使用过的名字。

注意下面的限制：

POU 的名字中不能包含空格

一个 POU 不能和另一个 POU 或数据类型具有相同的名字。为了避免可视化改变带来的问题,也不应与可视化对象具有相同名字.

数据类型不能与其它数据类型或 POU 有相同的名字。

全局变量列表不能与其它的全局变量列表有相同的名字。

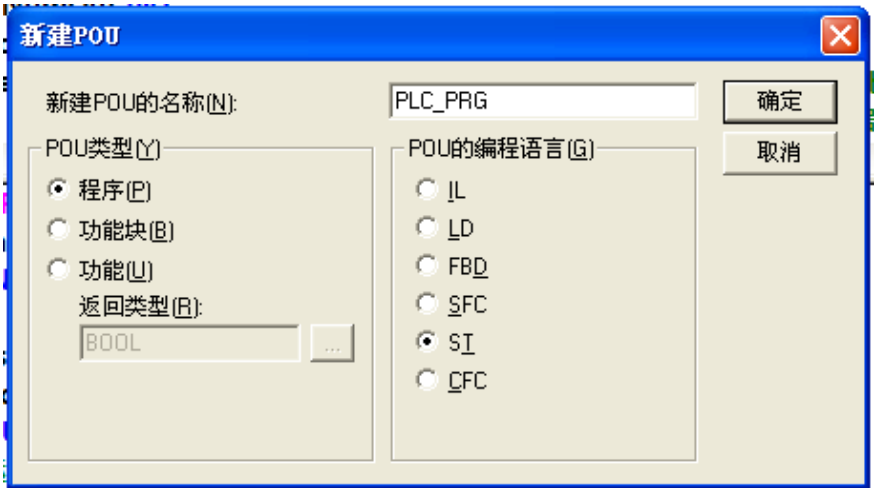
在同一 POU 中动作和其它的动作有相同的名字。

可视化不能与其它的可视化有相同的名字。为了避免可视化改变带来的问题,也不应与 POU 具有相同名字.

在其它情况下，同一命名是允许的，例如动作属于不同的 POUs 可以有相同的名字，可视化可以和 POU 同名。

POU，POU 类型（程序，功能或功能模块）和编程的语言必须被选择，“程序”是 POU 的类型的默认值，POU 的语言的默认值是最近创建的 POU 的语言，如果创建了一个功能类型的 POU，期望的数据类型必须输入到返回类型文本输入区域，这里所有的基本的和定义的数据类型（数组，结构体，枚举类，别名）是允许的，允许使用输入帮助（按<F2>）。

添加对象对话框



只有在按上面的命名规则没有冲突之后才能按 OK，在对象管理器中创建了新对象并且输入窗口出现。

使用命令“编辑”“插入”，不出现对话，在剪贴板上的对象插入。如果插入的对象名字和命名规则冲突（查看上面），附加带一个下划线字符的一系列数字来唯一标识。（例如，"Rightturnsig_1"）。

5.4.7 另存为模板

“数据类型”、“功能”、 “功能块”或“程序”类型的对象能保存为模板，在对象管理器中选择对象并在上下文菜单（鼠标右键）选择命令“另存为模板”。相同类型的 新对象自动得到最近创建的对象模板的声明部分。

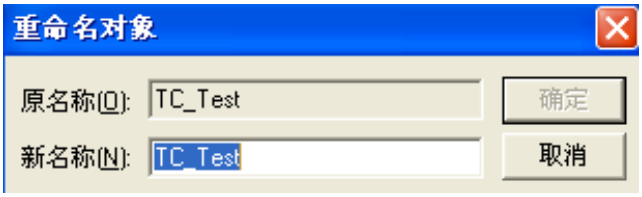
5.4.8 重命名对象

快捷方式 ： <Spacebar>

为当前选择的对象或 文件夹起一个新名字，对象的名字可以是还没有使用过的。

如果对象的编辑窗口打开，当对象名字改变时它的标题也跟着自动改变。

重命名 POU 对话框



5.4.9 对象转换

这个命令只能在 POU 中使用，可以从 SFC、 ST、FBD、 LD 和 IL 语言转换成 IL,

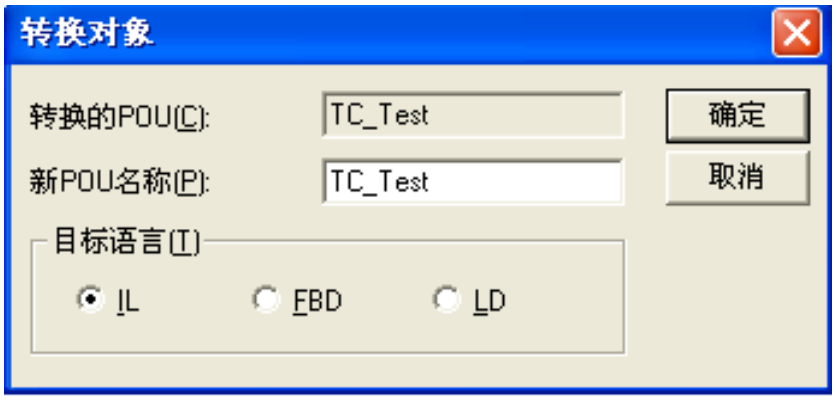
FBD, 和 LD 三种语言中的一种。

工程必须被编译，选择你想转换到的语言并给 POU 起一个新名字，POU 的名字可以是还没有使用过的，然后按 OK，新 POU 添加到了 POU 列表中。

在转换过程中处理的类型与应用于编译的类型相一致。

注意：动作不能转换。

转换 POU 对话框



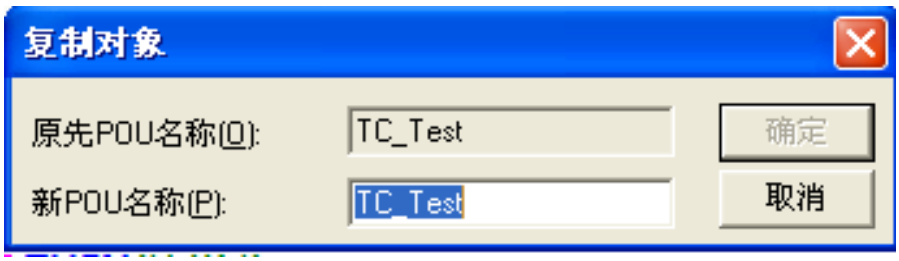
注意下列可能性:在 FBD 编辑器中创建的 POU，能使用命令'附加' '查看' 且不需要任何转换就能在 KOP 编辑器中显示和编辑.

5.4.10 对象复制

使用这个命令可以复制一个对象并存为一个新名字，在结果对话框中输入新对象的名字，对象的名字可以是没有使用过的名字。

另一方面，使用命令“编辑”“复制'，将对象复制到剪贴板上,此时不出现对话框，。

对象复制对话框



5.4.11 打开对象

快捷方式 ： <Enter>

使用命令把在 对象管理器中选择的对象加载到各自的编辑器中，这个对象的窗口已经打开，将会放置在显著的位置并能编辑。

这里有两种打开对象的其它方法：

在期望的对象上双击

如果在对象管理器中敲入对象名字的首字母，则打开一个对话框,在这里显示了以这个

字母为开头的所有对象，动作用符号<POU 名称>.<动作名称>列出。在对象选择框中的对象按字母顺序列出的，POU 的动作经常位于这个 POU 下面，选择期望的对象并按按钮 Open 把对象加载到它的编辑窗口中。在对象上以分级方式放置的所有的文件夹将会展开，资源中的全局变量也支持这个选项.

打开对象对话框

5.4.12 对象属性

在对象管理器中,显示当前选中对象的“属性”对话框 。

执行命令 '工程''对象访问权限'也将打开此对话框。

依据对象和工程的设置,附加的页面可用于定义对象的属性：

全局变量列表：

在全局变量列表中，显示列表中的参数和与网络变量数据交换的参数,可以在此修改它们。如果在对象管理器中，选择"全局变量"中的某一项来创建一个新的全局变量列表并执行"添加对象"，也将打开此对话框。

可视化：

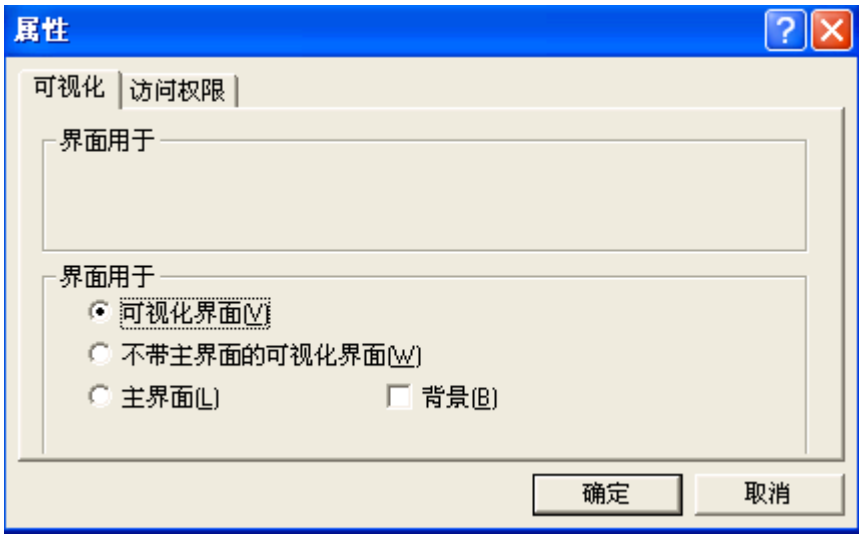
在'可视化界面'下，可以定义可视化对象的属性,下面是涉及‘主界面’应用的设置

可视化界面：此界面作为一般的可视化界面。

可视化界面不带主界面:：如果在工程中定义了主界面,此界面将不使用主界面。

主界面： 此界面将作为主界面.在默认方式下,主界面总是在一个可视化界面的前面，除非将它设置成背景。

可视化界面属性对话框如下：



5.4.13 对象访问权限

打开给不同用户组分配访问权限的对话框：

访问权限对话框

用户组 0 的成员现在为每个用户组分配单个的访问权限，有三个可能的设置：

不可访问：用户组的成员不能打开对象

只读访问：用户组的成员只能查看对象,但不能修改它们

完全访问：用户组的成员能够打开并修改对象

如果选中'应用到全部组件',工程中的所有 POU，数据类型，可视化和资源的设置都将采用在 对象管理器中当前选中对象的设置。

如果给用户组设置了密码，打开工程时将要求输入密码。

请注意:在设置访问权限时,要考虑涉及到可视化元件（可视化，安全性）操作的可能性。

5.4.14 添加动作

在对象管理器中这个命令用来给所选 POU 添加一个动作，在出现的对话框中输入动作的名字和选择编程的语言。

在对象管理器中新动作置于块的下面，在块的前面出现一个加号，在加号上点击出现动作对象并且在块的前面出现一个减号，重新在减号上点击动作消失,加号又出现。这也能在上下文菜单命令 '展开节点' '收缩节点'中完成。

5.4.15 打开实例

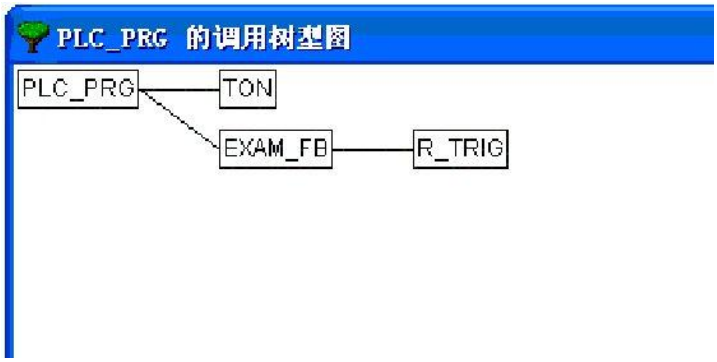
用这个命令打开并显示一个在对象管理器中选中的功能块的实例；另外，在对象管理器中双击功能块上将打开一个选择对话框，在这个对话框中列出了功能块的实例和执行代码，在这里选择期望的实例或执行代码并按确认，所选项将显示在窗口中。

请注意：如果你想查看实例，你首先要登录！（工程经过编译没有错误并通过命令 '联机' '登录'加载到 PLC 中）。

5.4.16 显示调用树

用这个命令你能打开一个显示了在 对象管理器中选中的对象的调用树的窗口，工程必须编译（参看'全部重新编译生成'），调用树包含了 POU 的调用和引用的数据类型。

显示调用树的例子



'工程' '显示交叉参考'

使用这个命令，打开一个对话框，此对话框可以列出所有变量、地址或 POU 的输出点。执行这个命令前，工程必须先编译（请参看 '工程' '编译生成'）。

首先选择类别“变量”,“地址”,或 “POU”,然后输入期望的元素的名字(或 F2)，在名字中输入一个"*"可以得到相应类别的所有元素 。

如果在上次编译后,工程作了修改,那么在对话框的标题中显示'未更新'.在这种情况下，如果不重新编译，那么在交叉参考列表中将不包含修改部分.

按'获得参考'按钮，得到所有应用位置的列表，和 POU 相连的是行号或网络编号、变量名和绑定的地址，范围空间显示了变量是局部的还是全局的；访问列显示了变量在当前位置上是能读还是能写，列的宽度根据条目的长度自动调整。

当选择了交叉引用列表中的一行并按'转到'按钮或在行上双击，包含此行的 POU 在编辑器中显示，通过这种方式你不需要耗时的搜索就能跳转到所有的应用位置。

为是使处理过程简化，使用发送到消息窗口按钮把当前的交叉引用列表移到信息窗口，从信息窗口中切换到相应的 POU 中。

对话框和显示交叉参考的例子

5.5 编辑功能

在所有的编辑器中你能使用下面的命令和在对象管理器中使用部分命令，命令位于菜单“编辑”下面和鼠标右键打开的上下文菜单中。

如果计算机上安装了 IntelliPoint 软件，OtoStudio 支持微软智能鼠标的所有功能。所有的编辑器具有缩放功能：按<Ctrl>键并滚动鼠标轮来放大，想缩小的话，按<Ctrl>键并向后滚动鼠标轮。

参照:

'编辑' '撤消'

'编辑' '重复'

'编辑' '剪切'

'编辑' '复制'

'编辑' '粘贴'

'编辑' '删除"

'编辑' '查找'

'编辑' '查找下一个'
'编辑' '替换'
'编辑' '输入助手'
非结构化显示
结构化显示
'编辑"自动声明
'编辑' '下一个错误'"
'编辑' '前一个错误'"
'编辑' '宏'

5.5.1 撤消

快捷方式 ： <Ctrl>+<Z>

这个命令用来取消在当前打开的编辑器窗口中或在对象管理器中最近执行的动作；重复使用取消动作，所有的动作后退到窗口刚打开时的动作。这个命令适用于编辑器中对 POU，数据类型，可视化和全局变量所做的操作,和在对象管理器中所做的操作。

用 '编辑' '重复'你可以恢复一个已经撤消的动作。

注意：命令撤消 和 重复适用于当前窗口。每个窗口带有自己的动作列表。如果你想在几个窗口中撤消动作，那么必须激活相应的窗口。当在对象管理器中撤消和重复时,焦点必须位于这里。

5.5.2 重复

快捷方式 ：<Ctrl>+<Y>

在当前打开的编辑器窗口或在对象管理器中使用这个命令时，能恢复已经撤消的动作('编辑' '撤消')。

每当你先前执行了命令“撤消”，你可以执行命令“重复”。

注意：命令撤消 和 重复适用于当前窗口。每个窗口带有自己的动作列表。如果你想在几个窗口中撤消动作，那么必须激活相应的窗口。当在对象管理器中撤消和重复时，焦点必须位于这里。

5.5.3 剪切

符号: 

快捷方式: <Ctrl>+<X> or <Shift>+<Delete>

执行这个命令，可以把当前的选择对象从编辑器转移到剪贴板中。即选择的对象从编辑器中移除。

此命令也可以在对象管理器中应用，不是所有的对象都能剪切，如 PLC 配置。

并不是所有的编辑器都支持剪切命令，它在某些编辑器中应用时会受到限制。

选择的形式依赖各自的编辑器：

在文本编辑器 IL, ST 和变量声明中，选择的是一列字符。

在 FBD 和 LD 编辑器中，选择的是由虚线框包围的网络、带输入的框、框和操作数。

在 SFC 编辑器中选择的是虚线框包围的一系列步。

使用命令 '编辑' '粘贴'来粘贴剪贴板的内容，在 SFC 编辑器中你也能使用命令 '附加' '插入并行分支(右侧)"或 '附加' '粘贴在下面'来做这些。

使用命令 '编辑' '复制'复制选择到剪贴板上而不删除它。

为了不改变剪贴板的内容而移除选中的区域，使用命令 '编辑' '删除'来完成。

5.5.4 复制

符号： 

快捷方式: <Ctrl>+<C>

执行这个命令，可以将当前的选择从编辑器复制到剪贴板上，它不改变编辑器窗口的内容。

这个命令同样应用于被选择的对象，不是所有的对象都被复制。例如，PLC 配置。

不是所有的编辑器支持复制，它在某些编辑器中应用时会受到限制。

对于选择的类型与使用命令“编辑” “剪切”具有相同的规则。

选择的形式依赖于各自的编辑器：

在文本编辑器(IL, ST 和变量声明)中选择是一列字符。

在 FBD 和 LD 编辑器中，选择的是由虚线框包围的网络、带输入的框、框和操作数。

在 SFC 编辑器中选择的是虚线框包围的一系列步。

使用命令 '编辑' '粘贴'来粘贴剪贴板的内容，在 SFC 编辑器中你也能使用命令 '附加' '插入并行分支(右侧)'或 '附加' '粘贴在下面'来做这些。

使用命令'编辑' '复制'复制选择到剪贴板上而不删除它。

为了删除一个选中区域同时把它放到剪贴板上，使用命令 '编辑' '剪切'来完成。

5.5.5 粘贴

符号： 

快捷方式：<Ctrl>+<V>

在编辑器窗口中把剪贴板上的内容粘贴到当前位置上，在 图形化编辑器中当插入产生一个正确的结构时命令才执行。

可以使用 对象管理器来从剪贴板上粘贴内容。

不是所有的编辑器都支持粘贴，它在某些编辑器中应用时会受到限制。

根据编辑器的类型不同，当前位置也有不同的定义。

在文本编辑器中（IL, ST 和变量声明）的当前位置是光标闪烁（一个垂直的直线）的地方，可以单击鼠标把光标定位到这里。

在 FBD 和 LD 编辑器 中，当前的位置是在网络编号区域中带虚线框的的第一个网络，剪贴板的内容插入到这个网络的前面，如果复制一部分结构，它将插入到选中元素的前面。

在 SFC 编辑器 中，当前位置是由虚线框包围处决定，依赖于选择和剪贴板的内容，这些内容或者插入到选择的前面或者作为一个分支(平行或可选分支)插入到选择位置的左边。在 SFC 中可以使用命令 '附加' '插入平行分支 (权利)' 或 '附加' '粘贴'来插入剪贴板的内容。

用命令 '编辑' '复制'来复制选择到剪贴板上而不删除它的内容。

用命令 '编辑' '删除'来删除一个选中的区域而不改变剪贴板内容。

5.5.6 删除

快捷方式: <Del>

从 编辑窗口中删除选中的区域，这不改变剪贴板中的内容。

对于选择的类型，与使用命令'编辑' '剪切'的规则相同。

选择的形式依赖于各自的编辑器：

在文本编辑器中（IL, ST 和变量声明）选择的是一列字符。

在 FBD 和 LD 编辑器中，选择的是在网络编号区域内的虚线框包围的许多网络。

在 SFC 编辑器中选择的是虚线框包围的一系列步。

在库文件管理器中选择是当前选中的库名。

使用命令 '编辑' '剪切'来删除一个选中的区域同时把它的内容放到剪贴板上。

5.5.7 查找

符号： 

用这个命令你可以在当前的编辑器窗口中搜索文本段。查找对话框一直打开，直到按取消键关闭。

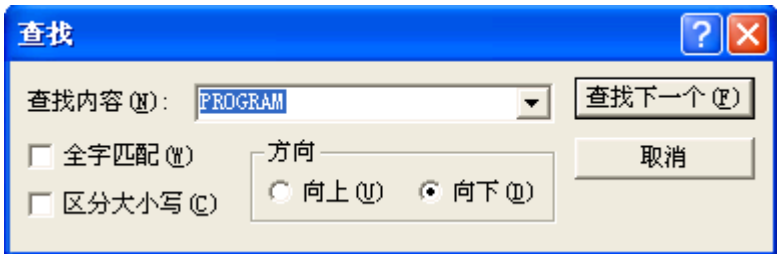
在区域查找内容中输入你要查找的内容。

另外，你能决定要查找的文本是否完全匹配，或区分大小写，和从当前光标位置开始是向上搜索还是向下搜索。

按钮'查找下一个'开始在选中的位置上继续按选择的方向搜索。

如果找到文本信息，它被加亮显示，如果文本没有找到，出现消息说明未发现信息。搜索能连续地被重复执行，直到编辑器窗口中内容的开始或最后。在 CFC 编辑器中将会考虑元素的集合顺序，搜索将从窗口的左上角开始到右上角结束，请注意 FBD 中 POU 是从右到左处理的。

查找对话框



5.5.8 查找下一个

符号 ： 

快捷方式 ： <F3>

用这个命令与最后执行的 '编辑' '查找'使用相同的参数执行搜索。

请注意在 FBD 中，POU 的搜索顺序是从右到左！

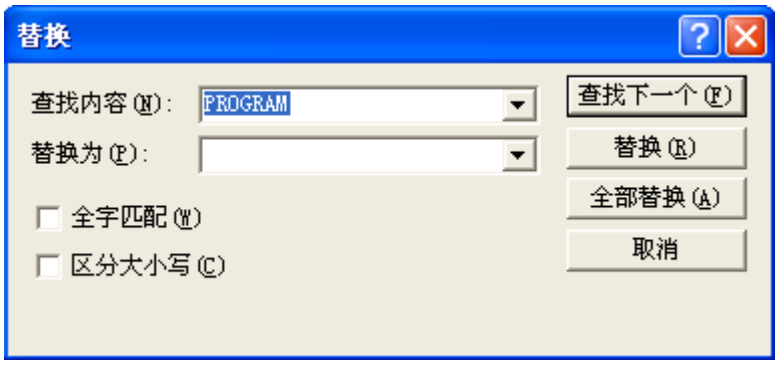
5.5.9 替换

用这个命令与使用命令 '编辑' '查找'一样查找一特定的信息，用其它的信息来替代它，在你选择了命令后查找和替换的对话框出现，直到按取消 或 关闭按钮才关闭这个对话框。

在编辑器中你标记过的字符自动作为查找内容，也可手工输入搜索字符串，按‘替换’将会用在区域替换为给出的字符串替换当前的，用‘查找下一个’找到下一个符合的字符。请注意，在 FBD 中，POU 是从右到左处理的。

按钮全部替换，将找到的字符全部替换，在处理过程最后报告替换了多少处。

查找和替换对话框



5.5.10 输入助手

快捷方式 ： <F2>

在编辑器窗口中使用这个命令将打开一个变量输入对话框，在左边列出了可选的输入类别，在右边列出了此类别下的条目，按‘确认’将选择插入到当前光标位置。

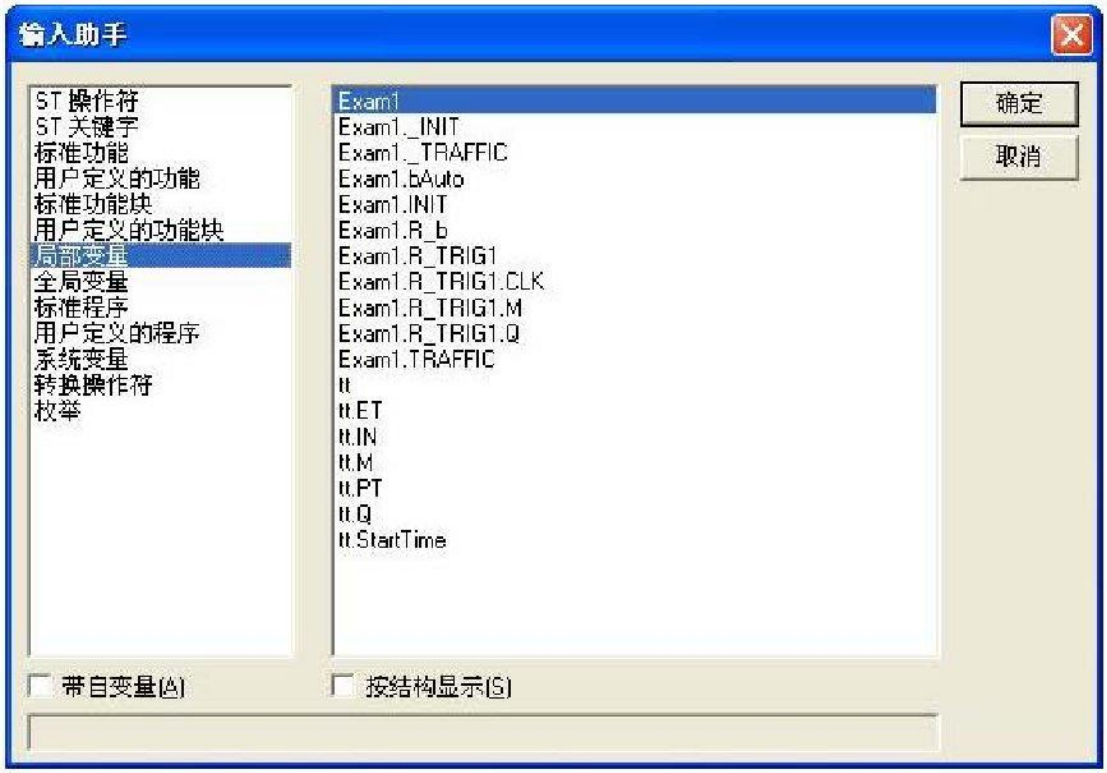
提供的类别依赖于当前光标在编辑器窗口中的位置。例如，那些可以在此处输入的（如，变量、操作数、POU、转换等）。

如果选项'带自变量'激活，当被选择元素插入时，元素所带变量也将插入，例如：功能块 fu1 被选中，它定义了输入变量 var_in ,则在当前位置插入 fu1(var_in:=)；功能 func1 的插入使用 var1 和 var2 作为传递变量 ： func1(var1,var2)。

按结构化还是按 非结构化来显示变量可以通过激活选项'结构化显示'来切换。

注意：也能使用智能功能来插入标识符.

非结构化显示

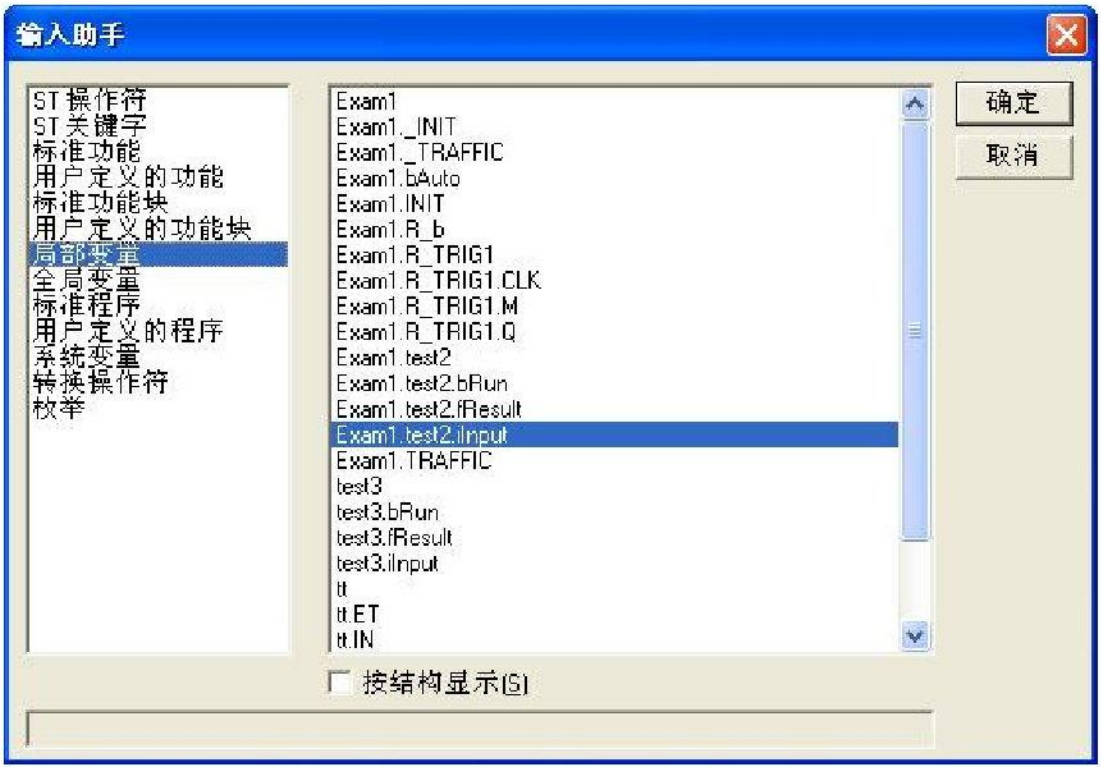


在每个类别中 POU、变量或数据类型只是简单地按字母顺序列出。

在不同的地方（比如，在监控列表中），需要多阶段变量，在那种情况下，输入帮助对话框显示所有 POU 列表和以点方式显示全局变量列表，在每个 POU 名字后有一个点。如果双击或按'确定'选中一个 POU，将打开它所包含的变量列表，如果 POU 中存在实例和数据类型，可以按级打开各自的变量列表，按'确定'选中的变量。

通过激活按结构显示能切换到结构化显示。

结构化显示



如果选中结构化显示，POU、变量或数据类型将按层次排列。对标准程序、标准功能、标准功能模块、用户定义的程序、用户定义的功能、用户定义的功能模块、全局变量、局部变量、定义的数据类型、监控变量也能按层次排列。可视化界面和它的层次显示与对象管理器中的显示相同；如果引用了库中的元素，这些元素按字母表的顺序插入在相应类别的最上面,并且相关层次显示与在库文件管理器中一样。

功能模块中被定义为局部或全局变量的输入输出变量在实例名下的“局部变量” 或 “全局变量” 里列出，选择实例名（例如，Inst_TP）按'确定'。

如果功能模块的实例被选中，可以选中选项'带自变量'。在文本语言 ST, IL 以及任务配置过程中，将插入功能模块的实例名和输入参数。

例如，如果选中 Inst (DeklarationInst: TON;) ，将插入：

Inst(IN:= ,PT:=)

如果选项没有被选中，只有实例名插入。在图形化语言编辑器或监控窗口中，通常只插入实例名。

结构组件的显示与功能模块实例类似。

对于枚举变量，单个的枚举值在枚举类型下面下列出。顺序是：库的枚举变量、数据类型的枚举变量、POU 的局部枚举变量。

通常的规则是包含子对象的行是不可选的（除了实例，参看上面），只能对自己的层次进行扩展显示或合拢，这与多阶段变量名使用一样。

如果在监控和配方管理器中或在采样跟踪配置对话框中选择跟踪变量时使用输入助手，有多种选择方式，按住<Shift>键时，你能选择一个范围的变量；当按住<Ctrl>键时，你能选择许多单个的变量，选中的变量被标记。在范围选择过程中不包含被选中的无效变量行，这些行将不会包含在选择中，当单个选择选好后，这些行不能被标记。

在监控窗口和采样跟踪配置中能够从输入助手对话框中选择结构，数组或实例。双击鼠标将展开和合拢所选元素的层次，在这些情况下只能按'确定'进行选择。

然后，选中的变量按行插入在监控窗口中，也就是说每个选中的变量独占一行。对于采

样跟踪变量，每个变量在采样跟踪变量列表中独占一行。

允许的采样跟踪变量最多 20 个,如果选中的变量超过了 20，会出现一个错误的信息“只允许 20 个变量”， 其余的选中变量不再插入列表中。

通过取消选项'按结构显示'来切换到 非结构化显示。

注意：一些条目只有在编译之后在输入助手对话框中才更新。

5.5.11 变量声明

快捷方式 ： <Shift>+<F2>

这个命令打开一个 变量的声明对话框，使用菜单 '工程' '选项' '编辑' '变量声明'或当在声明编辑器中使用一个新的未定义的变量时，这个对话框会自动打开。

5.5.12 下一个错误

快捷方式 ： <F4>

工程编译出错后，这个命令用于显示下一个错误.执行此命令后显示包含错误的编辑器窗口并且标记出错误的位置，同时在信息窗口显示相应的错误信息。

5.5.13 前一个错误

快捷方式 ：<Shift>+<F4>

工程编译出错后，这个命令用于显示前一个错误.执行此命令后显示包含错误的编辑器窗口并且标记出错误的位置，同时在信息窗口显示相应的错误信息。

5.5.14 宏

此菜单项将列出所有在工程中定义的宏(更多的信息请参看'工程' '选项' '宏'.当选中一个可执行的宏后,将打开对话框“执行宏”，对话框中显示宏的名字和当前执行的命令行，可以按'取消'按钮来取消宏的处理，当前命令的处理过程将无条件终止，在信息窗口中显示一条消息并且在联机操作过程中的日志中记录：

“<Macro(宏)>: 执行被用户终止”。

可以联机或离线执行宏指令,但是只有在相应模式下允许的命令可以执行.

5.6 联机功能

菜单项“联机”下面包含了在联机时用到的命令，某些命令的执行依赖于当前激活的编辑器。

联机命令在登录之后可用，下面的章节将进行详细讲述。

“联机修改”功能能够对控制器上运行的程序做在线修改，参看 '联机' '登录'。

参看此处 图表，它表明了工程－编译生成、工程－下载、在线修改和登录到目标系统之间的关系。

5.6.1 登录

符号： 

快捷方式 ： <Alt>+<F8>

这个命令把编程系统和控制器结合起来（或启动 仿真程序）并使程序进入联机模式。如果当前的工程自从打开或自从上次修改后还没有编译，将先进行编译。如果在编译过程中有错误，OtoStudio 不会进入联机模式。

如果当前的工程在上次下载后在控制器上做了修改，并且上一次下载信息还没有用命令“工程”“全部清除”删除，那么在执行登录命令之后弹出一个对话框：“程序已修改，是否下载新程序吗？”，按'是'按钮后，工程中修改的部分将加载到控制器中(相关信息参看此处图表，它表明了工程－生成、工程－下载、在线修改和登录到目标系统之间的关系。)，若选择'否'，程序不发生任何变化加载到控制器中，按'取消'取消命令，<全部加载>将把整个工程重新加载到控制器中。

扩展的登录对话框

如果选中'工程'-'选项'-'桌面'中的“安全模式下联机”,并且目标系统支持这个功能，按'详细'按钮后会扩展这个对话框：它将显示在 OtoStudio 中当前打开的工程的工程信息和当前加载到控制器中的工程的工程信息。

注意:联机登录对话框打开的样式依赖于哪个按钮设置为默认按钮。

注意:修改任务配置或 PLC 配置，或插入一个库以及执行命令“工程”“全部清除”（参看下面）后将不能进行联机修改。

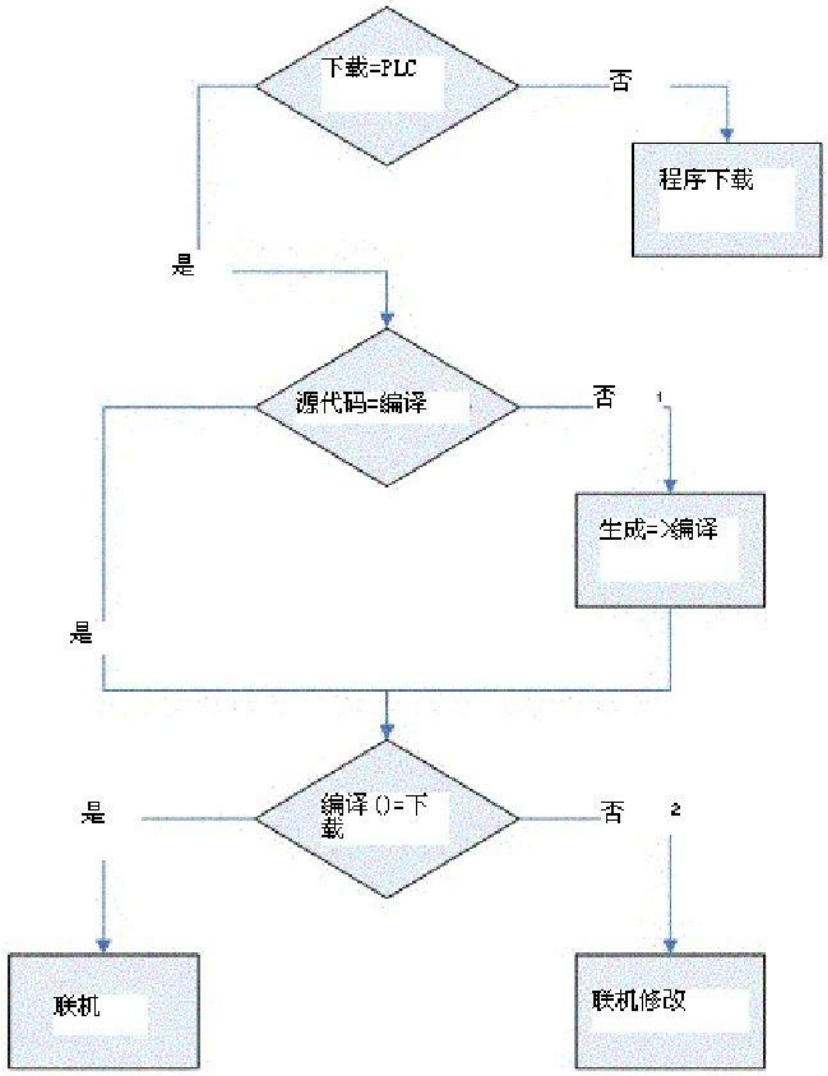
注意:联机修改将不对变量进行重新初始化.当联机修改执行后保留变量一直保持它们的值，当工程重新下载后重新初始化它们(参看"联机""下载")。

在成功登录后，所有的联机功能都能使用（如果在“工程”“选项”中的类别 '编译生成'中相应的设置都已经设置），将可以监控 变量声明中所有变量的当前值。

使用命令 '联机' '退出'来退出联机模式。

登录、生成、下载和在线修改之间的关系

下图表明了登录、生成（编译）、下载和在在线修改之间的关系:



使用的术语如下:

源代码	当前 OtoStudio 工程(*.pro 文件, 本地 PC)
编译	上次生成过程中的编译信息将用于补充编译.(*.ci-文件, 本地 PC)
下载	最后一次加载到 PLC 中的信息(*.ri-文件,本地 PC)
PLC	在 PLC 中可获得的工程(*.prg 文件, 目标系统)

在线修改说明

在修改任务配置或 PLC 配置、插入一个库文件、执行'工程' '全部清除'后，在线修改功能无效(参看下面说明)。

如果删除了上次下载（或在线修改）时创建的工程下载信息文件(文件 <工程名><目标标识号>.ri)，例如：执行命令 '工程' '全部清除', 那么在线修改功能不再起作用, 除非在另外一处保存了 ri-文件或重命名了它，这样你仍然可以得到上次下载的信息或通过'工程' '加载下载信息'获得. 相关内容可参看下面 '工程的在线修改....'.

在线修改不重新初始化变量，因而在线修改不考虑初始值的修改!

执行在线修改后保持变量（retain）保持原值, 工程重新下载也一样(参看下面, '联机' ' 下载').

当一个工程在多个 PLC 上运行时在线修改程序:

如果在两个同样的 PLC1 和 PLC2（相同的目标系统）上运行一个工程 proj.pro， 为

了确保使用在线修改功能使得工程的更新在两个控制器上都能实现，方法如下：

(1) 在 PLC1 中加载并启动工程，保存 PLC1 的下载信息

1. 建立 OtoStudio 工程 proj.pro to 与控制器 PLC1 的连接 (联机/通讯参数)，将工程 proj.pro 加载到 PLC1 (联机/登录,下载). 在工程目录中将创建包含下载信息的文件 proj00000001.ri.

2. 重命名文件 proj00000001.ri, 如为了清楚起见改为 proj00000001_PLC1.ri.改名保存这个文件是非常必要的，因为再下载 proj.pro 后新的下载信息将覆盖原来的信息，这样 PLC1 的下载信息将丢失。

3. 启动 PLC1 上的工程然后退出 ('联机' '运行', '联机' '退出').

(2) 在 PLC2 中加载并启动工程,保存 PLC2 的下载信息:

1. 与 PLC2 建立连接(使用与 PLC1 相同的目标系统，并将 proj.pro 下载到 PLC2. 因此在工程目录中将

再次创建包含下载信息的文件 proj00000001.ri.

2.为了清楚起见将新文件 proj00000001.ri 改名为 proj00000001_PLC2.ri.

3. 启动 PLC2 上的工程然后退出 ('联机' '运行', '联机' '退出').

(3) 在 OtoStudio 中线修改工程:

在 OtoStudio 中修改 proj.pro，然后使用在线修改功能传送到两个 PLC 中.

(4) 对 PLC1 中的程序进行在线修改, 为 PLC1 重新保存下载信息:

1. 在登录时 OtoStudio 查找文件 proj00000001.ri，因此为了对 PLC1 做在线修改，你必须首先恢复在 PLC1 中的下载信息文件，它现在保存在文件 proj00000001_PLC1.ri.

有两种方式:

(a) 将 proj00000001_PLC1.ri 改回 proj00000001.ri. 这样在登录到 PLC1 时，相应的下载信息可自动获得，OtoStudio 将询问你是否做在线修改.

(b)在登录前使用命令'工程' '加载下载信息'加载文件 proj00000001_PLC1.ri。在这种情况下你不需要给 ri-文件改名.

(5) 对 PLC2 中的程序进行在线修改, 为 PLC2 重新保存下载信息:

方法同上。只是下载信息文件在 proj00000001_PLC2.ri 中 .

如果系统报告错误：“选中的控制器配置与目标系统中的不匹配……”

确保在目标系统设置（资源）中输入的目标系统与在命令 '联机' '通讯参数'中输入的变量匹配。

错误：“通讯错误，执行退出”

检查控制器是否工作，检查在“联机”“通讯参数”中输入的变量是否与控制器中的变量匹配。特别是，应该检查是否输入了正确的端口；控制器和编程系统的波特率是否匹配；如果使用了网关服务器，检查通道设置是否正确。

错误：“程序已经被修改！要加载新程序吗？”

在编辑器中打开的工程和在 PLC（或程序正在运行的 仿真模式）中的当前工程不兼容，因而不能进行监控和 调试，你既可以选择‘否’来退出，然后打开所需的工程，也可以选择‘是’，在 PLC 中加载当前的工程。

信息：“程序已经改变，加载改变的部分吗？（联机改变）”。

工程正在控制器上运行，目标系统支持联机修改并且工程在控制器上随着最新下载或最新联机改变而改变，你可以决定这些变化是否要加载或命令是否要取消，也可以选择全部加载按钮加载这个编译过的代码。

5.6.2 退出

符号 ： 

快捷方式 ： <Ctrl>+<F8>

和 PLC 的连接中断或 仿真模式程序终止，转换到离线模式。

使用命令 '联机' '登录'可以转换到联机模式。

5.6.3 下载

这个命令把编译过的工程加载到 PLC 中。

下载的信息存储在文件<工程名>0000000ar.ri 之中，在 联机改变过程中用来比较当前程序和最近加载到控制器中的程序，因而只有改变的程序部分才重新加载，用命令 '工程' '清除'可以删除这个文件。

关于在多个 PLC 上做联机修改，参看: '联机' 登录'. 注意：在联机修改时*.ri-file 也将被更新。

在离线模式下，依据创建引导工程时的目标系统设置，可以重新生成*.ri 文件。

在下载后，只有永久变量保持它们的值（不重新初始化）。

5.6.4 运行

符号 ： 

快捷方式 ： <F5>

这个命令启动在 PLC 中或 仿真模式中的程序。

在命令'联机' '下载'或在 PLC 中的用户程序被命令 '联机' '停止'终止后，或用户程序停在一个断点处，或当执行了命令 '联机' '单循环' 后，这个命令开始执行。

5.6.5 停止

符号 ： 

快捷方式 ： <Shift>+<F8>

在两个循环周期内,终止 PLC 中或 仿真模式下的程序执行。

使用命令 '联机' '运行'来继续程序运行。

5.6.6 复位

这个命令把除了保持变量(VAR RETAIN)之外的所有变量的当前值复位成初始化值，如果变量设置了初始化值，这个命令使变量恢复到初始化值，所有的其它变量赋予系统默认值(如整型变量默认值是 0)。作为一个预防措施，在覆盖所有的变量之前 OtoStudio 会询问你来确认你的操作，在电源故障或关闭了控制器情况下，那么程序运行时将复位系统。

使用命令 '联机' '运行'重新启动程序。

请参看 '联机' '复位 (初始状态)'和“'联机' '复位(冷)'。

5.6.7 冷复位

这个命令使除了永久变量之外的所有变量也包括保持变量 复位到它们的初始化值。这种情况发生在加载到 PLC 中之前已经下载的程序启动时（冷启动），只有永久变量在复位之前保留它们的值。

相关的信息查看 '联机' '复位' 和 '联机' '复位 初始状态' 和变量重新初始化概述 - 保持变量。

5.6.8 复位到初始化状态

这个命令把所有的变量包括（保持变量）复位到它们的初始化值并在控制器上删除用户程序，控制器恢复到它的最初状态。

相关请查看 '联机' '复位' 和 '联机' '冷复位'和变量重新初始化概述 - 保持变量。

5.6.9 设置断点

符号 ： 

快捷方式 ： <F9>

这个命令用来在活动窗口的当前位置设置一个断点，如果在当前位置已经设置了断点，将取消这个断点。

设置断点的位置依赖于在活动窗口中用什么语言编写 POU。

在文本编辑器(IL, ST)，如果要在某行设置断点（可以设置断点的行的号码区是深灰色），断点设置在光标定位的地方，你也可以文本编辑器中的行号码区单击设置或删除一个断点。在 FBD 和 LD 中，断点设置在当前选中的网络上。在 FBD 或 LD 编辑器中也可以在网络号码区单击设置和删除一个断点。

在 SFC 中，断点设置在当前选中的步上，在 SFC 中你也能使用<Shift>加双击来设置或删除一个断点。

如果一个断点已经设置，行号区、网络号码区或步的背景色将会改变成高亮蓝色。

程序运行到断点，会自动暂停，相应的区域显示为红色背景色，为了继续运行程序，使用命令 '联机' '运行'、 '联机' '单步进入'、或 '联机' '单步跳过'。也可以用断点对话框来设置和删除断点。

5.6.10 断点对话框

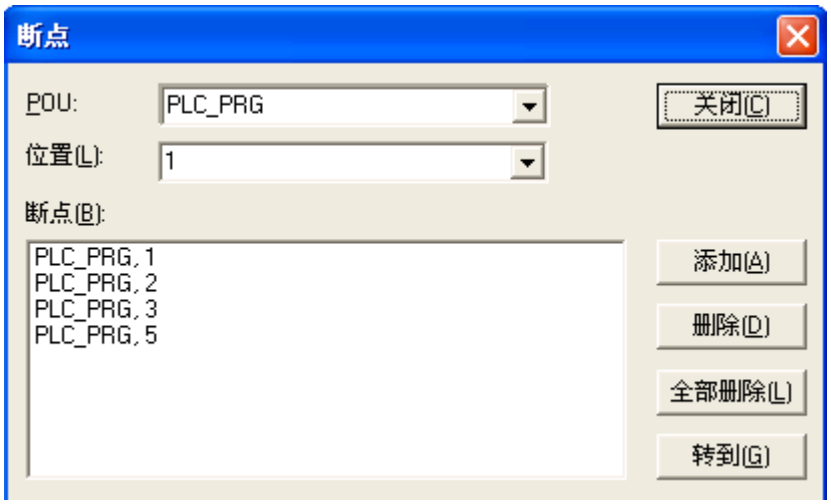
执行这个命令将打开整个工程的 断点编辑对话框，对话框也显示当前设置的所有断点。为了设置一个断点，在 POU 组合框中选择一个 POU，在断点位置组合框中选择相应的行或网络，然后按添加按钮，断点将添加到列表中。

为了删除一个断点，在断点列表中点击选择要删除的断点并按删除按钮。

全部删除按钮用来删除所有的断点。

为了在编辑器中定位到断点设置的位置，在断点列表中点击选择断点并按转到按钮。

断点编辑对话框



可以使用命令 '联机' '设置断点'来设置或删除断点。

5.6.11 单步跳过

符号 ： 

快捷方式 ： <F10>

使用这个命令，可以 单步执行程序，如果程序在调用一个 POU，程序在调用 POU 执行后暂停，在 SFC 中将执行一个完整的 动作。

如果当前的指令是调用一个功能或功能模块，那么功能或功能块将全部执行完。如果要运行到调用的功能或功能块的第一条指令，使用命令 '联机' '单步进入'。如果到达了最后一条指令，程序将会返回到 POU 中的下一条指令继续运行。

5.6.12 单步进入

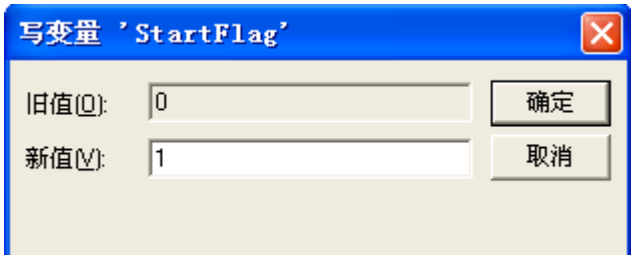
使用这个命令时进行 单步执行，程序暂停在调用的 POU 的第一条指令。
如果必要，可以转变到一个打开的 POU 中。
如果当前的位置是调用一个功能或功能块，那么光标停在调用的 POU 的第一条指令上。
在其它情况下，此命令和 '联机' '单步跳过'功能一样。

5.6.13 单循环

快捷方式 ： <Ctrl>+<F5>
这个命令执行 一个 PLC 循环并在这个循环后停止。此命令可以连续使用。
当执行命令 '联机' '运行'后，单循环结束。

5.6.14 写入新值

快捷方式 ：<Ctrl>+<F7>
用这个命令可以在循环开始时写入用户为变量设置的新值。(参看命令 '联机' '强制新值 '为变量设置永久值)。
只要单元素变量在监控窗口中可见，就可以改变的它们的值。
在命令“写入新值”执行之前。要为变量设置新值。
对于非布尔型变量，双击变量声明的行或选中变量所在行后按<回车>键，弹出对话框“写变量 <x>”，可在此对话框中输入变量的新值。
写变量对话框



对布尔型变量，通过双击变量声明的行来设置新值（它只在 TRUE 和 FALSE 之间切换，而没有其它值）不出现对话框。
要写入的值以青绿色显示在变量旧值后面的一个括号中。例，a=0 <:=34>。
提示：在 FBD 和 LD 编辑器中值在变量名后面以青绿色显示但不带括号。
要写入新值的变量的数目不受限制。
可以用同样的方式纠正和删除要输入的变量新值。
要写入的值在写入、删除或通过命令“强制新值”转移到强制列表之前,先存储在写入列表 (监控列表)中。
有两种方法进入写入新值对话框：
在菜单“联机“”写入新值“

点击对话框“编辑写入列表和强制列表“中的按钮”写入新值“。

当执行“写入新值“命令时，在写入列表中的变量新值将在循环开始时写到控制器中，然后从写入列表中删除。（如果执行了命令”强制新值“，要写入新值的变量也将从写入列表中删除，并转移到强制列表中！）

注意：在顺序功能图语言中，当变换是一个组合表达式时,不能对组合中的单个值用“写入新值“来改变，这是因为监控的是表达式的值，而不是单个变量的值（例如，“a AND b”,如果两个变量都是'真'时显示 TRUE）另外，在 FBD 中，例如如果一个表达式用做一个功能模块的输入，只有表达式的第一个变量才被监控，因而“写入新值”命令只能对这个变量写入新值。

5.6.15 强制新值

快捷方式 ：<F7>

使用这个命令，可以对一个或多个变量进行永久性赋值。(参看命令 '联机' '写入新值'，在循环开始为变量设置一次新值)。

赋值在运行系统中进行，在循环的开始处和循环的结束处新值生效。

一个循环的时序为：1.读入输入量 2. 强制赋值 3. 处理代码, 4. 强制赋值 5.写输出。

此项功能一直保持有效，直到被用户取消（用户使用命令“联机”“解除强制”）或退出编程系统。为了设置新值，首先要创建一个写入表(参看“联机”“写入新值”的描述)。写入表中包含的变量在监控中被标记出来。当执行命令“联机”“强制新值”后，写入表转换为强制写入表。如果当前已经激活了一个强制写入表，那么系统根据要求更新。写入表被清空，同时新值用红色表示（红色表示是强制值）。下次执行“强制新值”时将强制写入表中的修改部分写入到程序中。

注意：强制写入表的修改通过使用命令下一个“强制新值”来传递程序中。

注意：在写入表中包含的变量写入新值之前，当写入表存在的同时创建强制写入表为写入表中的变量强制新值。

为变量强制新值的方法有两种：

1、在菜单“联机“中的命令”强制新值“

2、使用对话框 “编辑写入值列表和强制值列表“中的按钮”强制新值“

注意：在顺序功能图语言中，当变换是一个组合表达式时,不能对组合中的单个值用“强制新值“来改变，这是因为监控的是表达式的值，而不是单个变量的值（例如，“a AND b”,如果两个变量都是'真'时显示 TRUE）另外，在 FBD 中，例如如果一个表达式用做一个功能模块的输入，只有表达式的第一个变量才被监控，因而“强制新值”命令只能对这个变量强制新值。

5.6.16 解除强制

快捷方式 ：<Shift>+<F7>

这个命令用于解除控制器中变量的强迫赋值。变量值采用正常的方式进行更改。

被强制赋值的变量能通过红色显示在监控器中进行识别。可以删除整个强制列表，也可以根据需要选择要解除强制的变量。

删除整个强制列表，对所有的变量进行强制释放，可以选择下面方式中的一个：

菜单“联机”中的命令“解除强制”。

使用对话框 '编辑写入值列表和强制值列表'中的按钮“解除强制”。

在对话框“移动写入/强制值列表”中使用命令“解除强制”来删除整个强制列表，如果你选择命令“解除强制”的同时 写入列表存在将打开这个对话。

对单个变量解除强制，首先用下面的方法来标记这些变量，标记的变量后面将出现提示“<解除强制>”。

在一个非布尔型变量所在行上双击鼠标，打开对话 '写入值 <x>'。选中<解除此变量的强制>，然后按“确定”。

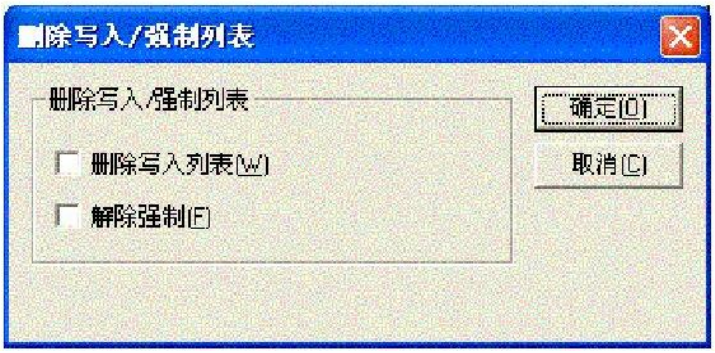
在布尔变量声明的行上双击后在行的末端显示<解除强制>。

使用菜单“联机”"写入/强制对话框"命令打开写入/强制对话框,删除列 '强制值'编辑字段中的值。

设置成"<解除强制>"的变量将在声明窗口显示，执行命令“强制新值”将把修改的强制列表传递到程序中。

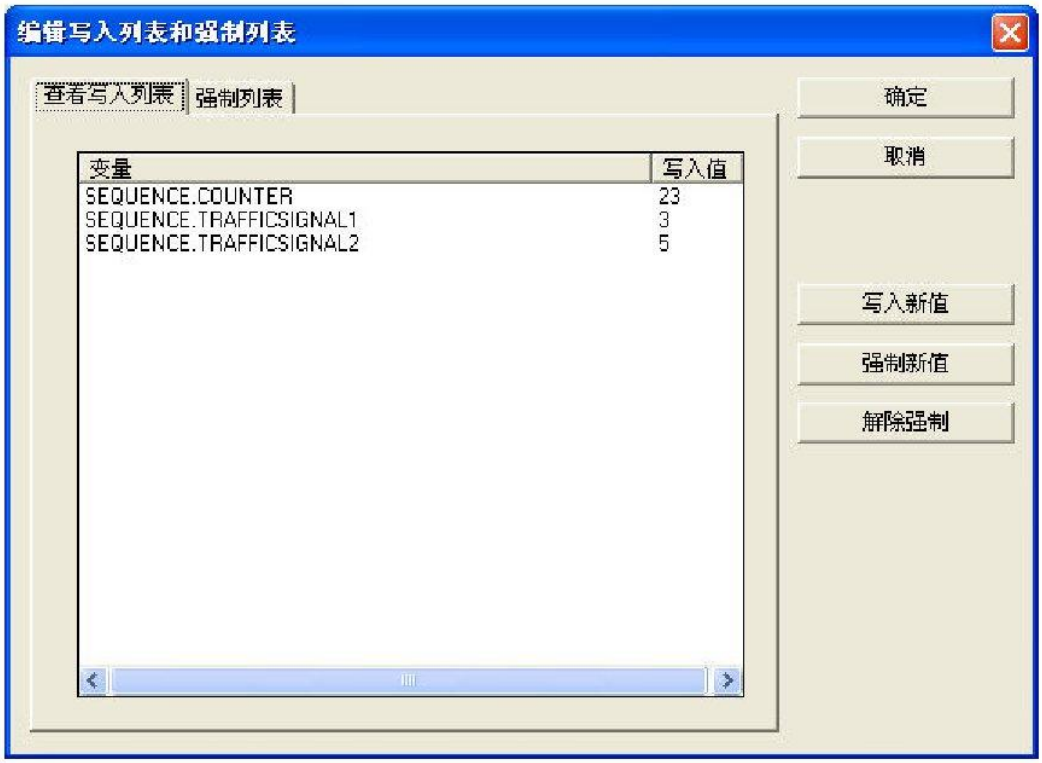
在执行命令“解除强制”时，如果当前写入列表（查看 '联机' '写入新值'）是非空的，对话框“删除写入/强制值列表”将会打开，用户可以决定是释放强制还是想删除写列表或把两个列表都删除。

删除写入/强制值列表对话框



5.6.17 写入/强制对话框

这个命令打开一个对话框，在这个对话中显示了选项卡写入列表和强制列表，在表中列出了写入或强制的变量的名字和数值。



通过命令 '联机' '写入新值'，可以把变量加进写入列表中。通过命令 '联机' '强制新值'，变量被转移到强制列表中。单击某变量的列“新值”或“强制值”来编辑新值。如果输入值的类型与定义的不一致，将显示一个错误消息。如果删除一个数值，条目也将从写入表中删除。或者变量被强制中止，并不需要退出指令而关闭对话框。

下面的命令，与联机菜单中的一样，可以通过按钮来使用它们：

强制新值: 当前写入列表中的全部变量将转入到强制列表，控制器中变量被强制赋值。所有标记<解除强制>的变量都不再被强制。然后关闭对话框。

写入新值：当前写入列表中的变量新值被写入到控制器中。然后关闭对话框。

解除强制: 在强制列表中的所有变量将被删除，或如果写入列表存在，将打开“删除写入/强制列表”对话框，由用户根据需要做选择是解除强制、删除写入列表还是两者都选。单击确定后，两个对话框同时关闭。

5.6.18 显示调用堆栈

当仿真模式在一个 断点停止时，可以使用这个命令，出现一个 POU 调用栈的列表的对话框。

调用堆栈例子

最先的 POU 通常是 PLC_PRG，因为这是执行的开始。

最后的 POU 通常是正在执行的 POU。

在选择了 POU 并按了‘转到’按钮，选中的 POU 加载到编辑器中来显示,显示的是正在处理的行或网络。

5.6.19 显示流程控制

依据目标系统的设置，用户可以激活或不激活流程控制功能。如果激活了它，在菜单项目的前面出现一个对勾。PLC 循环中正在执行的每一行或每一个网络将会被标记。

正在运行的行的行编号区域或网络编号区域将显示为绿色。一个附加的区域将添加到 IL 编辑器中，在这里显示当前累加器中的内容。在功能模块图和梯形图编辑器中，不传递布尔型值的连线上都将插入一个框。

在框中显示变量值。当连线上传输的布尔变量值为 TRUE 时，连接线将显示成蓝色，这可以监控信息/数据的流向。

注意：

1、使用流程控制后将增加程序的运行时间。这可能会造成以时间为周期的程序出现超时错误。

2、在激活的断点处不显示流程控制。

3、如果定义了与任务有关的看门狗，当激活流程控制时将关闭此功能。

5.6.20 仿真模式

如果选择了仿真模式，在菜单的前面出现一个对勾。

在仿真模式中，用户程序运行在 PC 机的 windows 操作系统下，这个模式用来测试工程。PC 和仿真模式的通讯使用的是 Windows 的消息机制。

如果程序不处于仿真模式，程序将运行在 PLC 上，PC 和 PLC 的通讯通常采用串口。

这个标记的状态和工程一起存储。

请注意：外部库的 POU 将不能运行在仿真模式下。

5.6.21 通讯参数

当本地 PC 和实时系统之间的通讯运行在系统中的网关服务器中时（如果使用了 OPC 或 DDE 服务器，在配置中必须输入相同的通讯变量），使用配置通讯参数对话框来设置通讯参数。

参看下面的项目：

网关系统的原理

本地 PC 的通讯参数对话框

设置期望的网关服务器和通道

为本地网关建立新的通道

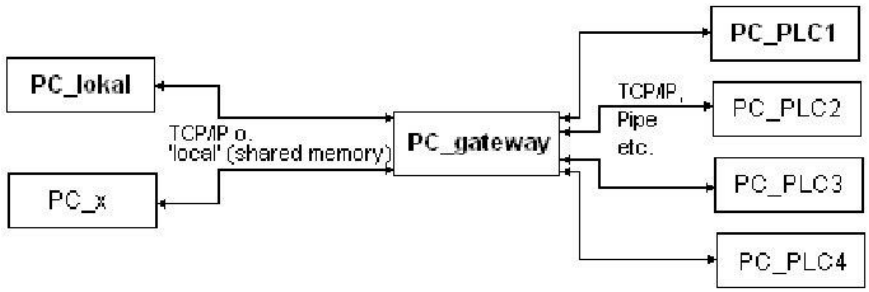
本地 PC 上通讯参数对话框显示的内容

网关系统的原理:

在我们解释对话的操作之前先来解释一下网关系统的原理：

网关服务器可以允许本地 PC 和一个或多个实时系统之间进行通讯，网关服务器可以和实时系统一起运行在本地 PC 上。如果我们要处理在其它计算机上运行的网关服务器时，我们必须保证它已经运行。本地安装的网关服务器，当登录到目标实时系统时，它自动启动。你可以通过在任务栏右下的 OtoStudio 符号的出现来识别它是否启动。在符号上单击鼠标右键，可以得到 Info 和 Finish 菜单项。

下面是一个网关系统图：



PC_local 是本地计算机，PC_x 是另外的计算机，PC_gateway 是网关服务器，PC_PLC1 到 PC_PLC4 是运行实时系统的计算机。图表显示了分离的模块，实际上，网关服务器可以和实时系统一起安装在本地计算机上。

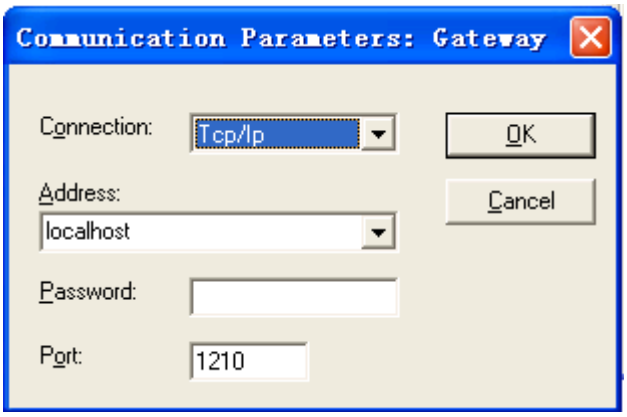
重要：和网关服务器之间的通讯只能建立在 TCP/IP 上，要保证你的计算机的配置的正确性。

从网关到不同的实时系统计算机可以运行在不同的协议上(TCP/IP、Pipe 等等)。

建立期望的网关服务器和通道

1.在通讯变量对话中建立期望的网关服务器和信道。

为了定义与期望的网关服务器之间的连接，按网关按钮来打开对话“网关信息参数”。



在这里你可以输入或编辑下面的部分：

从你计算机到正在运行的网关服务器计算机之间的连接类型，如果网关服务器在本地计算机上运行，连接通过共享内存或通过 TCP/IP 都是可以的；如果需要连接到不同的计算机上，只能使用 TCP/IP 协议。

正在运行的网关服务器的地址：IP 地址或正确的符号名比如 localhost。在初始化，标准的'localhost'作为计算机名字（地址），可以访问本地安装的网关。名字'localhost'与 IP 地址 172.0.0.1 在大部分情况下相同，在某些情况下必须在地址区域中直接输入名字。如果你想访问其它计算机上的网关服务器，你必须用它的名字或 IP 地址来替换'localhost'。

为选中的网关服务器设置密码，如果它在一个远程计算机上。如果它不正确的输入，或没有输入完整，出现一个错误消息。注意这个连接：你可以通过以下步骤为本地安装的网关服务器设置密码：在工具栏右下部分的网关符号上单击鼠标右键并选择“改变密码”。出现一个改变或输入密码的对话。如果你访问本地网关服务器，不要求输入密码。

网关服务器正运行的计算机的端口，作为一个规则已经给出了选中的网关服务器端口。

按 OK 键关闭对话，相应的条目（计算机地址）出现在在“通信参数”对话的顶部信道区域中，在它下面是网关服务器可用的信道。

2.在选中的网关服务器上建立期望的信道

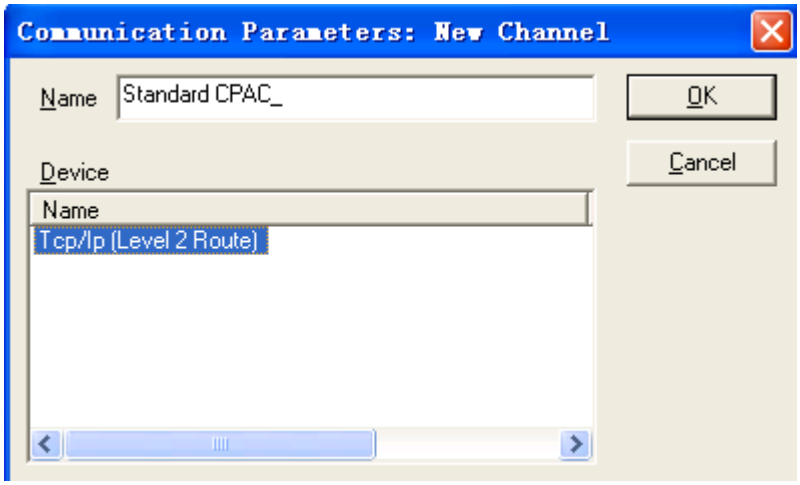
用鼠标在条目上单击选择其中的一个信道，相应的变量随后显示在表中。如果没有能和选中的网关地址建立连接，可能是因为还没有启动它或地址不正确，在地址之后的括号中显示‘没有连接’并出现一个消息：“不能找到网关的设置”。

一旦建立了期望的信道，按 OK 关闭对话。设置和工程一起保存起来。

为本地网关联机新的通道

可以为当前连接的网关服务器设置一个新的信道，这样可以从服务器上建立更多的连接，例如，和一个控制器的连接。可用的选择依赖于计算机上安装的设备驱动程序的数目的选择。

在通讯变量对话中，按新建按钮，打开对话 通信参数: 新通道。



Name 输入区域自动包含最近输入的信道的名字，如果没有定义信道，将会提供当前的网关名字，后跟一个下划线字符，例如，“localhost_”，可以在这里编辑信道的名字，信道的名字不必有一个唯一的名字，但是推荐使用唯一的名字。

在 Device 下面的表中列出了网关计算机上用到的设备驱动程序，在名字列中，用鼠标单击来选择一个可用的驱动程序，相应的注释，如果有的话，将出现在信息列中。

如果按 OK 键关闭了“...新信道”对话，新定义的信道出现在“通信参数”对话中，在信道中作为在减号下面的最下位置一个新条目，它只保存在本地工程中。你可以编辑列（查看下面的提示），按 OK 来确认输入的参数，然后退出“通信参数”对话。

为了使网关服务器 xy 能识别新输入的网关信道和它的变量，同时也为了使其它的计算机能访问这个网关，必须先登录到实时系统中，再打开“联机”“通信参数”对话时，新信道出现在信道树形结构中，不但在它的先前位置，而且网关服务器 xy 的名字或地址下面，这表明了它已经存在于网络中。

除了本地计算机外，也可以在其它计算机上打开通讯变量对话，选择网关 xy 并使用它的新信道。

如果登录时出现通讯错误，可能是接口不能打开（例如，串行连接的端口 COM1），可能是因为端口被其它设备使用了，也可能是控制器没有运行。

网关服务器已经识别的信道的变量在配置对话中将不能编辑，变量区域显示为灰色，当它未激活时，可以删除这个连接。

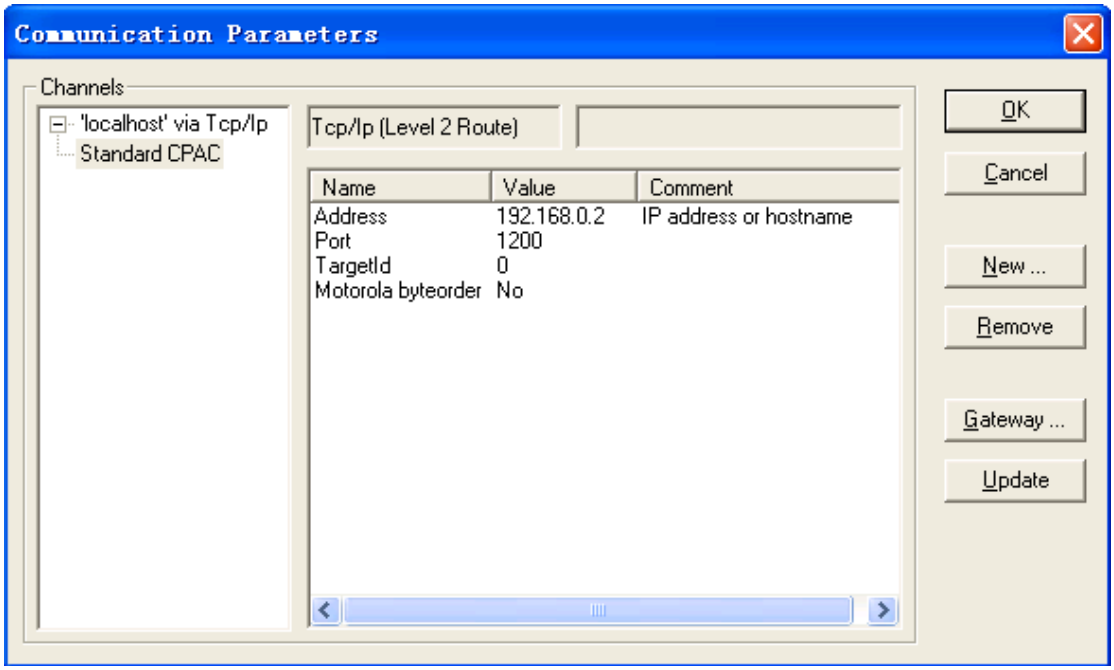
重要：删除一个信道是不可逆的，当按按钮移动时，它就被删除了。

如何在本地 PC 表示通信参数菜单

这个对话用来 选择一个与 PLC 通讯的网关服务器，而且可以为安装在本地计算机上的网关服务器 设置新的信道，这些信道能够被网络上的其它计算机访问。

使用按钮更新可以在任何时候调用当前的设置。

如果根据例子'网关系统原理'已经配置了通讯变量，将出现下面的对话：



标题 Channels 下列出了两个类别的连接：

一方面显示出的安装在当前连接的网关服务器上的所有连接称为“本地主机”。这个网关的地址或名字位于在减号后面的最上位置，在我们的例子中运行在本地计算机上。在通常情况下“本地主机”对应于本地计算机的 IP 地址 127.0.0.1。它的下面，是网关信道设置到的(PC_PLC1 to 3)实时计算机的三个地址，它们可能在本地计算机或其它的没有连接到网关服务器的计算机上设置过。

第二个类别的信道包括所有连接到从本地计算机上设置的网关的连接，它们在减号下面直接创建了分支 PC_PLC1 和 PC_PLC4。这些地址不需要在网关上识别。对于例子中的 PC_PLC4，配置变量存储在本地 工程中但下次登录到实时系统时，首先被网关识别。对于 PC_PLC1 这个已经出现，相关的网关地址已经作为一个子分支出现在了信道树形结构中。

在对话的中间部分可以看到左边选中的信道和 Name、Value 和 Comment 下相关变量的说明。

在通信参数菜单中编辑参数的提示

只能编辑值列下的文本区域。

用鼠标选择一个文本区域，通过双击或按空格键进入编辑模式，文本输入完成时按<Enter>确认。

你可以使用<Tabulator>或<Shift> + <Tabulator>来跳转到下一个或先前切换或编辑中。

为编辑数字值，可以用方向键或 Page Up/Down 键来分别地改变值一个或十个单位。

鼠标双击也能增加一个单位，为数字值安装了输入检查：<Ctrl> + <Home> 或 <Ctrl> + <End>可以为问题中的变量类型输入最低或最高值可能值。

快速检查连接网关的不成功连接

如果连接到选中的网关计算机不成功，应该有一个下面的检查（在信道区域中网关服务器地址之后 通讯变量对话中出现消息“没有连接” ）：

网关服务器已经启动了吗？（工具栏的低部右边部分出现三色符号）

在通讯变量对话中输入的 IP 地址是不是网关运行的计算机的？（使用“ping”来检查）

TCP/IP 连接工作吗？错误可能在于 TCP/IP。

5.6.22 源代码下载

这个命令将工程的源代码加载到控制器系统中，不要与工程编译时产生的代码相混淆！在 '工程''选项''代码源下载 '对话框中设置下载选项（时间，大小）。

5.6.23 创建引导工程

使用这个命令，在控制器上为已编译的工程创建引导工程，当重新启动时可以自动加载它。创建三个文件：default.prg 包含工程代码，default.chk 包含代码检验，default.sts 包含了重新启动之后的控制器状态（启动/停止）。

在离线模式下，也可以对编译没有错误的工程使用命令“联机”“创建引导工程”，在这种情况下，在工程目录中创建下面的文件：保存引导工程代码的<工程名>.prg 和包含检验的<工程名>.chk，这些文件可以重命名，然后复制到 PLC 中。

依据目标系统的设置，在离线模式下创建引导工程时，可能会创建一个新的 *.ri-文件 ，如果这个文件已经存在，依据目标系统设置会出现一个对话框。

注意：如果激活了工程-选项-源代码下载- 使用选择文件创建引导工程 ，在使用命令“联机”“创建引导工程”时，选中的资源自动加载到控制器中。

5.6.24 文件写入 PLC

这个命令可以把任何期望的文件加载到控制器中，在打开的对话框“在控制器中写入文件”中可以选择期望的文件。

在使用“打开”按钮关闭对话框后，文件加载到了控制器中并用同一名字保存，在加载过程中有一个进度提示。

使用命令 '联机''从 PLC 中读取文件'可以取回先前加载到控制器中的文件。

5.6.25 从 PLC 中读取文件

用这个命令将打开对话框“从控制器中读取文件”，用于读取回先前使用命令 '联机' '文件写入 PLC'加载到控制器中的文件。在文件名下面，提供了期望文件的名字，在选择窗口中输入它加载的目录，按“保存”按钮关闭对话。

5.7 设置窗口

窗口设置

在“窗口”菜单下,能找到管理窗口的所有命令，有自动设置窗口命令、有打开 库文件管理器的命令、有切换打开窗口的命令，在菜单的最后你会找到一个按它们打开时的顺序打开

的所有窗口的一个列表，在相关的条目上单击鼠标能切换到期望的窗口，在活动窗口的附近出现一个√。

参照：

'窗口' '水平平铺'

'窗口' '垂直平铺'

'窗口' '层叠窗口'

'窗口' '最小化排列'

'窗口' '关闭所有'

'窗口' '信息'

'窗口' '水平平铺'

用这个命令你能在工作区水平的排列所有的窗口使它们不重叠并且占据整个工作区间。

'窗口' '垂直平铺'

用这个命令你能在工作区垂直的排列所有的窗口使它们不重叠并且占据整个工作区间。

'窗口' '层叠窗口'

使用这个命令你能在工作区间中以层叠的方式排列所有的窗口，一个窗口跟着一个窗口。

'窗口' '最小化排列'

使用这个命令你能在工作区间排列所有的最小化窗口使它们在工作区间的底端排成一列。

'窗口' '关闭所有窗口'

使用这个命令能关闭工作区间的所有窗口。

'窗口' '信息'

快捷方式 ： <Shift>+<Esc>

用这个命令你能打开和关闭带有来自最近编译，校核，或比较过程的消息的 信息窗口 如果消息窗口是打开的，在命令的附近会出现一个√。

5.8 帮助

'帮助' '内容' 和 '搜索'

使用‘帮助’菜单下的‘内容’或‘搜索’，打开帮助主题窗口。此窗口中的文件也可以借助 HTML 帮助查看器（Internet Explorer V4.1 或更高版本）。

内容选项卡显示帮助手册的目录。通过双击或点击‘+’/‘-’可以打开或关闭书。在窗口右侧显示所选目录的帮助信息。具有不同颜色和下划线的文本表明此处有超级链接，即查看更多相关内容。在这些文字上单击将打开链接或图片。例如，点击在这页下面的‘帮助主题窗口’，将显示帮助主题窗。点击‘上下文帮助’将打开相应的帮助信息页。

在索引选项卡上，可以寻找特定词的帮助信息。在搜索选项卡上，在所有帮助信息中查找搜索词。

参看: 上下文关联帮助.

帮助主题窗口

上下文关联帮助

快捷方式 ： <F1>

为了打开 在线帮助，在活动窗口中、在一个对话框中，或菜单命令上按<F1>键。

你也可以选中文本（例如：关键字或一个标准功能），然后按‘F1’,则显示相关的帮助。

第六章 OtoStudio 中的编辑器

6.1 关于所有的编辑器

编辑器的组件

POU 的所有编辑器由声明部分和主体部分组成，主体可以由其它的文本或图形编辑器组成；声明部分通常是文本编辑器。主体和声明部分通过能拖动的屏幕分割器来分开，通过用鼠标点击它并朝上或朝下移动它到希望的位置。

打印范围

如果在工程选项中的对话“工作区域”中的选项“显示打印范围”被选中，编辑器内容打印时用到的水平和垂直的页边空白用红色虚线显示。打印机的属性和打印版面的尺寸在菜单 '文件' '打印机设置'中选择。如果没有打印机设置或打印版面输入，应用默认设置 (Default.DFR 和默认打印机)。

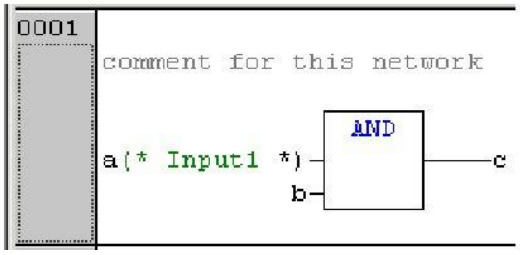
注意：当放大因数选择 100%时，打印页边空白原样显示。

注释

用户注释必须封套入特殊的符号序列中“(*”和“*)”。例如，(*这是一个注释*)。

在所有的文本编辑器中，在任何期望的位置、在所有的声明、IL 和 ST 语言和在自定义数据类型中都允许使用注释。如果工程使用一个模板来打印输出，在变量声明过程中输入的注释出现在每个变量后的基于文本编程组件中。

在 FBD 和 LD 图形化编辑器中，可以为每个网络输入注释。



搜索你想添加注释的网络并激活“插入”“信息”，如果在菜单 '附加' '选项'中相应的选项激活，在梯形图编辑器中可以为每个特殊的触点和线圈添加附加的 注释，在 CFC 中有特殊注释的 POU 可以按自己的意愿放置。

在 SFC 中，你能在编辑步属性对话中输入关于步的注释。

在 '工程' '选项' '建立选项'中的合适的选项激活，允许使用嵌套注释。

在联机模式，如果鼠标在变量上停留，在内容提示中将显示变量和地址的注释。

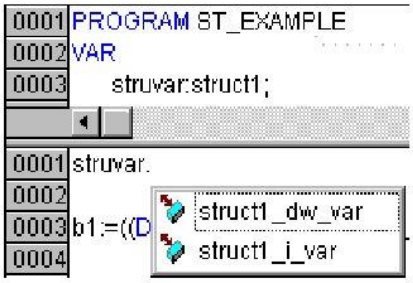
切换到 POU

快捷方式 ： <Alt>+<Enter>

用这个命令选中的 POU 加载到它的编辑器中，如果光标放置在文本编辑器中的 POU 名字上或如果 POU 框在图形编辑器中被选中，在内容菜单(<F2>)或在“附加”菜单中可用到这个命令。

如果你从库中处理一个 POU，那么库文件管理器被调用，相应的 POU 显示。

'附加' '打开实例'
这个命令与"'工程' '打开实例' 相同。
如果光标放置在文本编辑器中的功能模块的名字上或如果功能模块框在图形编辑器中被选中，在内容菜单(<F2>)或在'附加'菜单中可用到这个命令。
智能功能
如果选项列 表组件在工程选项对话中为类别"编辑器"激活，智能功能将在所有的编辑器中、在监控和配方管理器中、在可视化和采样跟踪中可用：
如果插入一个圆点"."来代替一个标识符，出现一个选择框，列出了工程中的所有本地和全局变量，在这里可以选择这些元素中的一个并按"返回"来插入到圆点之后，你也能通过双击在列表条目上来插入元素。
如果在圆点后输入了一个 功能模块实例或一个结构化变量，选择框列出了相应功能模块的所有输入和输出变量 或列出结构化组件，在这里可以能选择期望的元素并按"Return"或双击来输入它。



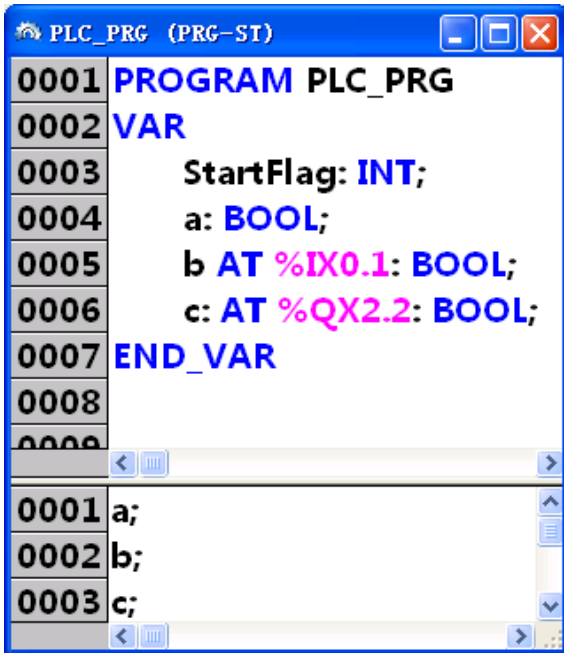
脱机状态下的标识符
在脱机状态且所有的编辑器被如下申请：如果指针被放置在可编辑的标识符上，变量被按数据类型划分
（如 VAR_GLOBAL），变量特征（如 RETAIN)，地址和注释将被显示。

6.2 声明编辑器

声明编辑器用来声明 POU 变量和全局变量、声明数据类型，它能使用通常的窗口功能，如果安装相应的驱动程序还能使用智能鼠标的功能。
在改写模式下，"OV"在状态栏上显示为黑色，通过<Ins>键可以在插入和改写模式之间切换。
句式颜色支持变量的声明。
在内容菜单（鼠标右键或<Ctrl>+<F10>）中有最重要的命令。

声明部分

只有在这个POU中的所有将要使用的变量才在POU的声明部分中声明，这些变量包括：输入变量、输出变量、输入/输出变量、 局部变量、添加的变量和常量。声明格式是基于 IEC61131-3 标准。关于使用模板创建全局变量、数据类型、功能、功能模块或程序类型的对象的可能性。下面是在 OtoStudio 编辑器中正确声明变量的例子



在变量声明(变量名称), 用户定义的 数据类型 和创建 POUs (功能,功能块,程序)和 可视化界面时定义标识符.为了保证标识符的唯一性,你可以使用下面的标识符命名方法..

(1) 变量名称

在应用程序和库文件通常采用匈牙利符号法:

对于每个变量,主名称的命名应该直观,简短.每个词的第一个字母应该大写,其它的小写.(例如: FileSize. 如果需要,可以创建其它语言的翻译文件.

对应于变量的数据类型,可以在主名称前增加前缀,以表明变量的类型,最好用小写字母.为了区别于 BYTE 和易于 IEC 编程人员的理解(参看赋地址%IX0.0),强烈推荐在 BOOL 变量前使用 x 作为前缀.

例如:

```
bySubIndex: BYTE;
sFileName: STRING;
udiCounter: UDINT;
```

在嵌套声明中,按声明顺序增加前缀:

```
pabyTelegramData: POINTER TO ARRAY [0..7] OF BYTE;
```

功能块实例和用户定义的数据类型的变量的前缀可以分别使用 FB 和数据类型名(例如: sdo).

例如:

```
cansdoReceivedTelegram: CAN_SDOTelegram;
TYPE CAN_SDOTelegram :
  STRUCT
    wIndex:WORD;
    bySubIndex:BYTE;
    byLen:BYTE;
    aby: ARRAY [0..3] OF BYTE;
  END_STRUCT
END_TYPE
```

对于局部常量(c) 开头以 c 为前缀, 后跟下划线(_),然后是类型前缀和变量名.
例如:

```
VAR CONSTANT
    c_uiSyncID: UINT := 16#80;
END_VAR
```

对于全局变量(g)和全局常量(gc) 附加的前缀+'_'被附加到库库前缀:
例如:

```
VAR_GLOBAL
    CAN_g_iTest: INT;
END_VAR
VAR_GLOBAL CONSTANT
    CAN_gc_dwExample: DWORD;
END_VAR
```

(2) 用户定义的数据类型 (DUT)
每个结构数据类型的名称组成方式是: 库名前缀(如: CAN), 下划线'_'和结构的恰当的描述(如: SDOTelegram). 对于这个结构所使用的变量,相关的前缀应该直接放在冒号后.
例如:

```
TYPE CAN_SDOTelegram : (*  前缀: sdo *)
  STRUCT
    wIndex:WORD;
    bySubIndex:BYTE;
    byLen:BYTE;
    abyData: ARRAY [0..3] OF BYTE;
  END_STRUCT
END_TYPE
```

枚举 以库名前缀开始(如: CAL), 后跟下划线'_'和大写字母的标识符.
注意:在 OtoStudio 老版本中,枚举值大于 16#7FFF 会发生错误,因为它们不能自动转换成 INT 型数值. 由于这个原因枚举(ENUM)应该使用正确的 INT 型数值定义.
例如:

```
TYPE CAL_Day :(
  CAL_MONDAY,
  CAL_TUESDAY,
  CAL_WEDNESDAY,
  CAL_THIRSDAY,
  CAL_FRIDAY,
  CAL_SATURDAY,
  CAL_SUNDAY);
```

声明:

```
eToday: CAL_Day;
```

(3) 功能, 功能块,程序(POU)
功能, 功能块和程序的名称组成是库前缀(Example: CAN), 下划线'_'和 POU 的恰当描述(如:SendTelegram). 与变量声明一样,POU 名的第一个词的第一个字母应该大写,其它是小写.建议 POU 名由动词和名词组成.
例如:

FUNCTION_BLOCK CAN_SendTelegram (* 前缀: canst *)

在声明部分,为 POU 注释一个简短的描述. 同时为所有的输入和输出加注释. 在功能块下,对于创建的实例,相关的前缀应该直接放在名称后面.

动作可以没有前缀; 因为他只是内部调用, 即由 POU 本身调用, 以 prv_开头.

由于要与 OtoStudio 老版本兼容,每一个功能至少要有一个参数. 外部功能不能使用结构作为返回值.

(4) 可视化界面的名称

注意: 你必须避免可视化界面的名称不能与工程中的 POU 具有相同的名字. 否则在可视化界面之间转换时会出现错误.

输入变量

在关键字 VAR_INPUT 和 END_VAR 之间，所有定义的变量作为 POU 的输入变量，在调用位置，可以随着调用赋予变量值。

例如：

```
VAR_INPUT
    in1 : INT; (* 1. Inputvariable*)
END_VAR
```

输出变量

在关键字 VAR_OUTPUT 和 END_VAR 之间，定义的所有变量作为 POU 的输出变量，也就是说调用后值返回

到 POU。

例如：

```
VAR_OUTPUT
    out1 : INT; (* 1. Outputvariable*)
END_VAR
```

输入输出变量

在关键字 VAR_IN_OUT 和 END_VAR 之间，定义的所有变量作为 POU 的输入和输出变量。

使用这个变量，传递变量的值被改变。那意味着变量的输入值不能为常量。

正是这个原因，所以不能通过<functionblockinstance><in/outputvariable>来从外部直接地读或写功能模块的输入和输出变量。

例如：

```
VAR_IN_OUT
    inout1:INT; (* 1. 输入输出变量 *)
END_VAR
```

局部变量

在关键字 VAR 和 END_VAR 之间定义了 POU 的所有局部变量，它们没有外部的连接，换句话说，它们不能从外部写入。

例如：

```
VAR
    loc1:INT; (* 1. 局部变量*)
END_VAR
```

保持变量

Remanent 变量能在程序运行期间始终保持它们的值，这些变量包括保留变量。

保留变量用关键字 RETAIN 来识别，这些变量保持它们的值即使是在控制器的非正常关闭时和正常的关闭和其中的一个控制器或在命令 '联机' '复位'时。当程序重新运行时，存储的值将进行进一步的处理。一个具体的例子是生产线上的饼形计数器在电源故障后重新开始计数。

所有其它的变量从新初始化，不是用它们的初始化值或标准初始化的值。

例如：

```
VAR RETAIN
    rem1:INT; (* 1. 保持变量*)
END_VAR
```

注意：

1.如果一个局部变量定义为保留变量，变量将保存在保留区（象一个全局保留变量）

2.如果在功能模块中的一个局部变量定义为保留变量，功能模块的整个实例将会保存在保留区（POU 的所有数据），因而只有定义的保留变量才处理为保留变量。

3.如果在功能中的局部变量定义为保留变量，这不起任何作用，变量将不保存在保留区内！

常量

常量用关键字 CONSTANT 来识别，它们能定义为局部或全局变量。

句式：

VAR CONSTANT

<Identifier>:<Type> := <initialization>;

END_VAR

例如：

```
VAR CONSTANT
con1:INT:=12; (* 1. 常量*)
END_VAR
```

外部变量

要导入到 POU 的全局变量用关键字 EXTERNAL 来指定，它们也出现在联机模式中声明部分的观察窗口中。

如果外部变量声明与全局变量声明在各方面不匹配，出现下面的消息：“变量的声明和全局变量的声明不匹配”。

如果全局变量不存在，出现下面的错误消息“不能识别的全局变量”

例如：

```
VAR EXTERNAL
    var_ext1:INT:=12; (* 外部变量 *)
END_VAR
```

关键字

关键字在所有编辑器中书写为大写字母，关键字不能用作变量。例如关键字：VAR、VAR_CONSTANT、 IF、 NOT、

INT。查看 OtoStudio 中的有效关键字目录.

声明变量

变量的声明采用下面的句式：

<Identifier> {AT<Address>}:<Type> {:=<initialization>};

{}中的部分是可选择的。

关于标识符，是一个变量的名字，需要注意的是它不能包含空格或变元音字符，它不能重复定义并且不能与关键字相同。不区分大小写，VAR1、 Var1 and var1 是相同的变量。在标识符中的下划线是有意义的，例如，A_BCD 和 AB_CD 在解释的时候认为是不同的变量。不允许在标识符前面或在标识符中有多个连续的下划线。标识符的长度是不限制的。

变量的所有声明和数据类型元素能包含初始化值。它们通过“：=”来操作。对于基本类型的变量它们的初始化值是常量，所有的声明的默认的初始化值是 0。

例如：

var1:INT:=12; (* 整型变量,初始值是 12*)

如果你想让变量直接使用明确的地址，那么必须用关键字 AT 来定义变量。

为了快速输入声明，使用快捷方式模式。

在功能块你能指定变量不完整的地址描述，为了在本地实例中使用这样的变量，在变量配置中必须使用它的条目。也可以自动声明。

AT 声明

如果你想让变量直接使用明确的地址，那么必须用关键字 AT 来定义变量。

这样做的好处是你能给地址赋予一个恰当的名字，输入和输出信号的任何改变只要在一个地方修改就可以了。

要求输入的变量不能被写操作访问。

例如：

counter_heat7 AT %QX0.0: BOOL;

lightcabinetimpulse AT %IX7.2: BOOL;

download AT %MX2.2: BOOL;

注意：如果布尔变量被赋予了字节，字，或 DWORD 地址，它们只占一个字节而不是在偏移后的第一个位。

‘插入' ‘声明关键字'

你能用这个命令来打开一个在 POU 的声明部分用到的关键字的列表，在选中 关键字后并且确认了选择，关键字插入在当前光标位置。

当你打开 输入助手(<F2>)并选择了'声明'类别时，你也能查看关键字列表。

‘插入' ‘类型'

用这个命令你能接收一个为变量的声明的可能类型，当你访问输入帮助(<F2>)后你也能接收列表。

类型分为下列类别：

标准类型 布尔，字节，等等。

结构体，枚举类型，等等。

实例声明的标准功能模块

实例声明的已定义的功能模块

OtoStudio 支持 IEC1131-3 的所有 标准类型：

语句颜色

在所有的编辑器中你能在变量的应用和声明中使用可视化的帮助。可以避免错误，或很快查出错误，因为文本用颜色显示。

注释解释指令，迅速显示；关键字不会偶然拼错，等等。

将会用到下面强调的颜色：

蓝色　　关键字

绿色　　文本编辑器中的注释

粉红色　　特殊常量（例如，TRUE/FALSE, T#3s, %IX0.0）
红色　　输入错误（例如，无效时间常量，关键字，小写……）
黑色　　变量，常量

快捷模式

OtoStudio 的声明编辑器允许你使用快捷模式。
当你使用<Ctrl><Enter>结束一行时这个模式激活。
支持下面的捷径
所有的标识符一直到最近的标识符将成为变量声明标识符
声明的类型由行的最近的标识符决定。在本书中，使用到以下快捷：

B or BOOL　声明为　BOOL

I or INT　声明为　INT

R or REAL　声明为　REAL

S or string　声明为　STRING

如果通过这些规则还没有建立类型，类型是 BOOL 并且最近的标识符将不会用作一个类型。（例 1.）
每个常量，依赖声明时的类型，将会变成一个初值或字符串。（例 2 和 3）
在分号后面的文本成为一个注释（例 4.）
所有在行中的其它字符被忽略（例如，例 5.的感叹号）

例如：	声明
A	A: BOOL;
A B I 2	A, B: INT := 2;
ST S 2; A string	ST:STRING(2); (* 字符串 *)
X %MD12 R 5 Real Number X AT %MD12: REAL := 5.0;(* 实数 *)	
B !	B: BOOL;

自动声明

如果在选项对话框的编辑器类别中选中 自动声明选项，在没有定义的变量输入后在所有编辑器中出现一个对话框，在对话框帮助下，现在可以声明变量。
变量声明对话框

在 CLASS 组合框包含有局部变量(VAR)、输入变量(VAR_INPUT)、输出变量(VAR_OUTPUT)、输入/输出变量 (VAR_INOUT)或全局变量(VAR_GLOBAL)，选择你要处理的类型。

通过常量、保持变量，你能决定要处理常量还是保持变量。在编辑器中输入的变量名添加到了'名称'区域，BOOL 已位于'类型'区域。用来打开输入助手对话框,你可以选择所需的数据类型。

数组的声明:

如果选择了数组作为变量的类型，输入数组边界的对话框出现：

通过用鼠标在相应的区域单击打开编辑区域可以在'起始'和'结束'下输入数组边界，数组的数据类型输入在'类型'区域。为了做这些，用按钮来调用输入助手对话框。

按 OK 按钮来退出数组边界对话框，在 IEC 格式中的变量声明建立在对话中的 Type 区域中的条目。例如，

整型数组 ARRAY [1..5, 1..3]，在区域'初始值'中，可以输入声明变量的初始化值。如果这是一个数组或结构体，通过按钮...或打开其它类型变量的输入帮助对话，你将可以打开一个特殊的初始化对话框。

在数组初始化对话中，出现一列数组元素；在“：=”前的空白处单击打开一个编辑区域来输入元素的初始化值。

为结构体和单个元素初始化的对话显示在一个树状结构中，类型和默认的初始化值出现在变量后面的括号中；每个后面跟着“：=”。在“：=”后面的区域单击打开一个编辑区域，在这里可以输入期望的初始化值。如果元素是数组，数组中单个区域的显示可以通过鼠标在加号上单击来扩展开并且可以用初始化值来编辑这些区域。

按'确定'键退出初始化对话框后，数组或结构体的初始化值出现在在 IEC 格式中的声明对话框中的'初始值'区域中。

例如：x:=5,field:=2,3,struct2:=(a:=2,b:=3)

在'地址'区域中，可以绑定声明到 IEC 地址的变量。

如果需要，可以输入一个注释，注释可以通过使用组合键<Ctrl> + <Enter>来形成换行格式。

按'确定'键，声明对话关闭并且根据 IEC 的句法变量输入到相应声明编辑器中。

注意：变量声明的对话框，也可以通过命令 '编辑' '声明变量'来得到。在联机模式下，如果光标停留在一个变量上，用<Shift><F2>可以打开自动声明的窗口，在这个窗口中，当前变量的相关设置显示在这里。

声明编辑器中行号

在脱机模式下，在一个特定的行号码上单击将标记整个文本行。

在联机模式下，若此行显示的是结构变量，那么在行号上单击将会展开或隐藏结构变量

中的变量。

按表格形式声明

在选项对话框如果声明表格选项激活，声明编辑器象一个表格，就象一个卡片索引框，你能选择各自变量类型的选择卡并编辑变量。每个变量有下面的条目区域。

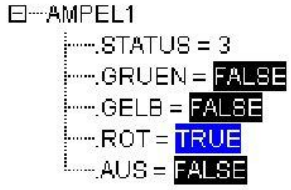
名字：变量的输入标识符。
地址：如果必须，变量的输入地址（AT 声明）
类型：变量的输入类型。（当例示为功能模块时，输入功能模块）
初始化：输入变量的值（与赋值符号":= "相同）
注释：在这里输入一个注释。
声明编辑器的两个显示类型之间可以切换，在联机模式下，显示没有区别。
选择命令 '插入' '新声明'来编辑一个新变量。
声明的表格形式

'插入' '新声明'

使用这个命令,将在声明编辑器的 声明表格中加入一个新变量，如果当前的光标位置在表格中区域，新变量将会粘贴在行的前面；否则，新变量粘贴在表格的最后。而且，通过使用右箭头或在表格最后的区域中使用 TAB 键，也能在表格的最后粘贴一个新声明。
你会得到一个变量名字位于'名称'区域、"Bool"值位于'类型'区域，作为它的默认设置。可以改变这些值为自己期望的变量完整定义所需的值、名字和类型。

联机模式中的声明编辑器

在联机模式中，声明编辑器变为一个监视窗口。在每个行中变量后面跟着等号(=)和它的值，如果在这个点上的变量是未定义的，会出现三个问号？？？。对于功能模块，只有打开的实例（命令："工程""打开实例"）才显示变量的值。
在每个多元素变量附近有一个加号，按<Enter>键或在变量上双击之后，打开变量。如例子交通信号结构将打开。



当打开一个变量，它的所有变量在它后面列出，减号出现在变量附近。如果你双击或按<Enter>键，变量将会关闭，加号将重新出现。
在单变量上按<Enter>键或双击将打开一个对话框来写变量，这里可以改变变量的当前

值，在布尔变量的情况下，不出现对话框；这些变量被触发。
新值显示在变量之后，在突出的括号中颜色为青绿色并保持不变，如果执行了命令 '联机' '写入新值'，所有的变量被放于选中的列表中并显示为黑色。
如果使用了命令 '联机' '强制新值'，所有的变量将被设置为选中的值，直到执行了 '强制赋值'命令。在这种情况下，强制值的颜色改变为红色。

6.3 声明编辑器中的预处理 pragma 指令

编程指令影响编译和预编译过程中变量的属性，在声明编辑器的编程行中或在它自己的行中可以使用辅助的文本。

编程指令写在{}中，不区分大小写，如{ <Instruction text> }。

如果编译器不能解释指令文本，这个程序作为一个注释处理并读取一次，出现一个警告：“编译器忽略指令‘<Instruction text>’！ ”。

依赖于程序的类型和内容，Pragma 在它所在的行上操作,或在 pragma 结束前的所有行上操作，或执行了相同的带有不同参数的 pragma 前的所有行上操作,或到达文件末尾。这里的文件是声明部分,执行部分,全局变量列表和类型声明.

左括号应跟在变量名后面,左和右括号应在同一行中。

在 OtoStudio 中可用到下面的 Pragma：

Pragma{flag}用来初始化，监控，创建符号

Pragma{bitaccess...}用来访问位

Pragma{parameter..}, {template...}, {instance...}用来创建变量管理器中的条目

用于库声明部分显示控制的 Pragmas.

初始化，追踪，符号创建，位存取的预处理 Pragma 指令

Pragma {flag [<flags>] [off|on]}

<flags>可以是下面标记的组合：

noinit:	变量将不初始化
nowatch:	将不再监视变量
noread:	变量导出到一个非读权限的符号文件中
nowrite:	变量导出到一个非读权限的符号文件
noread, nowrite:	变量将不导出到符号文件中

{flag on}开始 Pragma 对所有随后的变量声明操作直到{flag off}结束，或直到被其它{flag <flags> on}pragma 覆盖。

没有{flag on}和{flag off}时，Pragma 只对当前变量声明起作用（变量声明被下一个分号关闭）。

使用 pragma {flag}的例子：

变量的初始化和监视：

变量 a 将不初始化和被监视，变量 b 将不初始化：

```
VAR
  a : INT {flag noinit, nowatch};
  b : INT {flag noinit };
END_VAR
VAR
```

```
  {flag noinit, nowatch on}
  a : INT;
  {flag noinit on}
  b : INT;
  {flag off}
END_VAR
```

两个变量都不初始化：

```
{flag noinit on}
VAR
  a : INT;
  b : INT;
END_VAR
{flag off}
VAR
  {flag noinit on}
  a : INT;
  b : INT;
  {flag off}
END_VAR
```

把变量添加到符号文件：

使用“noread” 和“nowrite”，在 POU 中有读和/或写权限，来为选中变量提供严格的访问权限。变量的默认的访问权限和变量定义的 POU 中的设置一样。如果一个变量没有读写权限，它不会导出到符号文件。

例如：

```
VAR
  a : INT {flag noread};
  b : INT {flag noread, nowrite};
END_VAR
VAR
  { flag noread on}
  a : INT;
  { flag noread, nowrite on}
  b : INT;
  {flag off}
END_VAR
```

变量 a 和 b 都不导出到符号文件：

```
{ flag noread, nowrite on }
VAR
  a : INT;
  b : INT;
END_VAR
{flag off}
VAR
  { flag noread, nowrite on}
```

```
  a : INT;
  b : INT;
  {flag off}
END_VAR
```

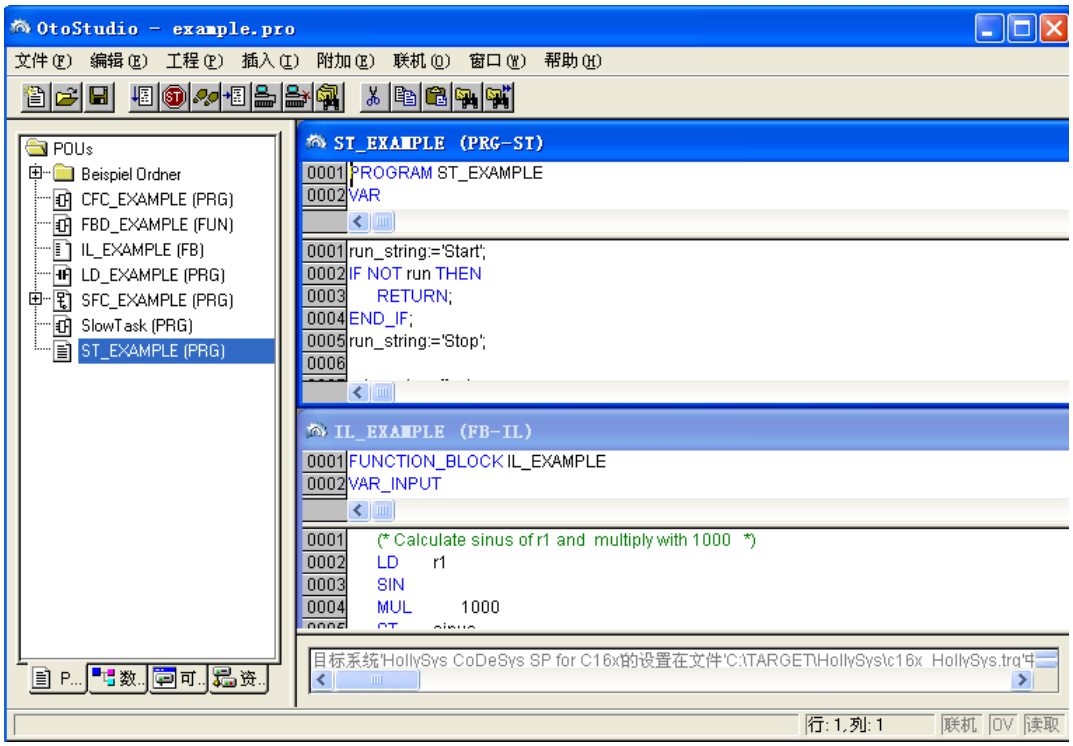
pragma 指令对后面的变量声明做附加操作..
例如：（所有使用的 POU 将会导出读和写权限）

```
a : afb;
...
FUNCTION_BLOCK afB
VAR
  b : bfb {flag nowrite};
  c : INT;
END_VAR
...
FUNCTION_BLOCK bfB
VAR
  d : INT {flag noread};
  e : INT {flag nowrite};
END_VAR
```

“a.b.d”: 将不会被导出
“a.b.e”: 导出后只具有只读权限
“a.c”: 导出后具有读写权限
Pragma {bitaccess...} 用于位访问
这个 pragma 可以用来正确显示，在全局常量、输入帮助、智能功能变量在声明窗口监视的帮助之下进行位访问操作的变量。当在特殊的 POU 的声明窗口中监视这个变量时，使用的全局常量在各自结构体变量下面。
注意: 必须激活工程->选项->生成-> '替换常量' !
Pragma 必须插入到结构体的声明中的一个单独的行。这个行不能用分号终止。
句式：
{bitaccess <Global Constant> <Bitnumber> '<注释>'}
<Global Constant>： 全局常量的名称, 它必须在全局变量列表中定义.
<Bitnumber>： 全局常量数值,在全局变量列表中定义.
例如,在变量中给位赋地址
Pragma {link} 用于在代码产生过程中联接 POU
通常情况下,在工程中不被调用的 POU(程序,功能,功能块)或数据类型定义(DUT),在代码产生中不被联接.
但是如果一个通过库包含在工程中的功能,即使不被应用程序直接使用(例如,检查操作),那么在下载到运行系统后也应该可以使用.由于这个原因,为了与 POU 联接,你可以在 POU 或 DUT 声明部分的任一位置添加{link} pragma 指令.

6.4 文本编辑器

文本编辑器为 OtoStudio 的执行部分提供通用的窗口文本编辑器功能。
句式颜色支持文本编辑器中的执行部分。



在改写模式下 状态栏 显示一个黑色的 OV，你可以通过<Ins>键来在改写模式和插入模式之间切换
在内容菜单（鼠标右键或<Ctrl>+<F10>）中包含有重要的命令。
文本编辑器使用下面的菜单命令：
参照:

'插入' '操作符'
'插入' '操作数'
'插入' '功能'
'插入' '功能块'
调用带有输出参数的 POUs
联机模式下的文本编辑器
'扩展' '监视选项'
断点位置
如何设置断点
删除断点
断点处的状态
在文本编辑器中行号码
指令表编辑器
结构化文本编辑器

'插入' '操作符'在文本编辑器中

使用这个命令，当前语言中用到的所有运算符将显示在对话框中。如果选中一个运算符按 OK 列表关闭，然后高亮度显示的运算符将插入到当前光标位置。(象在输入帮助中一样操作)。

'插入' '操作数'在文本编辑器中

使用这个命令将显示在对话框中的所有变量，你可以从全局变量、局部变量、系统变量中选择要显示那个变量的列表。

如果选中了其中的一个操作数，按'确定'后对话框关闭，高亮度显示的操作数将会插入到当前的指针位置（和输入助手的操作一样）。

'插入' '功能' 在文本编辑器中

使用这个命令在对话框中将显示所有的 功能，你选择是显示用户定义的列表显示还是标准功能列表显示。

如果选中了一个功能，按 OK 键关闭对话框，高亮显示的功能将插入到当前指针位置。如果在对话框中选中了'带形参'选项，必须的输入和输出变量也会插入进来。

'插入' '功能块' 在文本编辑器中

使用这个命令在对话框中显示所有的 功能块，你可以选择显示用户定义的列表还是标准功能模块的列表。

如果选中一个功能模块，按'确定'将关闭对话框，高亮度显示的功能模块将插入到当前指针位置。

如果在对话框中选择了'带形参'选项，功能模块的必要的输入和输出变量也将插入进来，但是你不能强制给这些变量赋值。

在文本编辑器中调用带有输出参数的 POUs

在文本语言 IL 和 ST 中，可以直接为被调用 POU 的输出变量赋值 。

例如：

Afbinst 的输出变量 out1 赋予变量 a .

IL: CAL afbinst(in1:=1, out1=>a)

ST: afbinst(in1:=1, out1=>a);

如果通过输入帮助(<F2>)用选项“With arguments”在 ST 或 IL POU 的执行窗口中插入一个 POU，它将自动和所有的变量以这个句法格式显示出来，但不能强制这些变量赋值。

联机模式下的文本编辑器

在编辑器中的联机功能是用来设置断点和单个步处理（步），和监视功能结合起来，用户就拥有了标准语言调试器的调试功能。

在联机模式，文本编辑器窗口垂直分为两部分，在窗口左边的部分是标准程序文本，在右边是显示的变量，它的值在各自的行上被改变。

显示与声明部分的显示一样。当 PLC 运行时，将会显示各自变量的当前值。当监视表达式或位地址变量时要注意以下：在监视表达式时，通常显示整个表达式的值。例如，a 和 b 显示为蓝色或如果 a 和 b 的值是 TRUE 显示“：=TRUE”。对于位地址变量，位值通常被监视（例如，a.3 显示为蓝色或如果 a 有值 4）

如果你用鼠标在变量上停留一会，变量的类型、地址、注释将会在工具提示中显示出来。

'附加' '监控选项'

用这个命令可以配置监视窗口。在文本编辑器中，在监视过程中窗口被分成两部分，程序加载在左边部分，在右边部分中，相应程序行中的变量被监视。

可以指定监视窗口的宽度和在一行中两个变量之间的距离，声明距离为 1 对应选中一行字体的高度。

在文本编辑器中断点位置

在 OtoStudio 中多个 IL 行组合为一个 C 代码行，因此断点不能在每一行中设置。断点的位置包括在程序中变量的值改变的地方或程序流程图中断的地方的所有位置。（例外：功能调用，如果必须，功能中的断点必须设置在这里。）

在 IL 中提供了下面的断点位置：

在 POU 开始的地方

在每个 LD、LDN 处

在 JMP、JMPC、JMPCN 处

在每个标签处

在每个 CAL、CALC、CALCN 处

在每个 RET、RETC、RETCN 处

在 POU 结束的地方

结构化文本提供下面的断点位置：

在每个赋值处

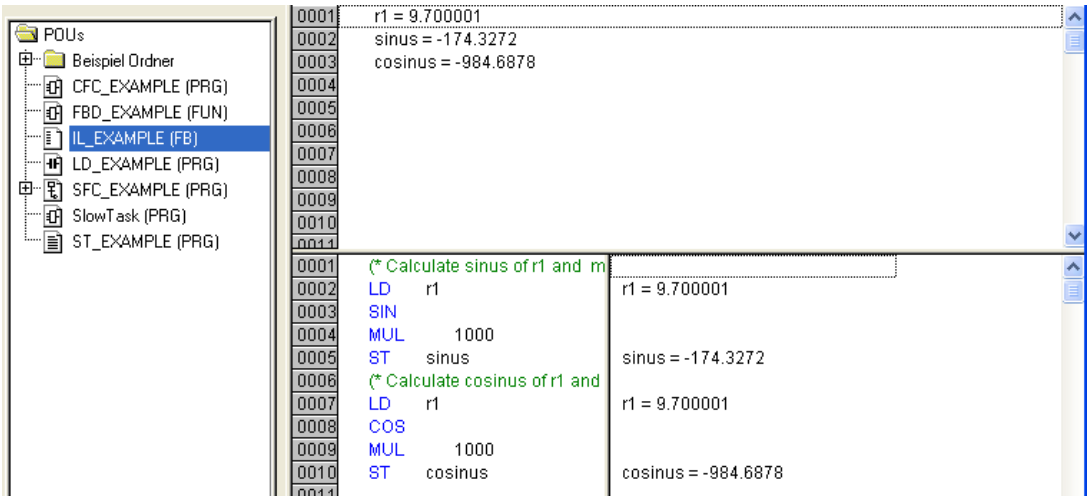
在每个 RETURN 和 EXIT 指令处

在计算条件的行处(WHILE, IF, REPEAT)

在 POU 结束的地方

在行号码区域处，用在工程选项中设置的颜色来标记断点位置。

可能断点位置的 IL 编辑器 (暗色数字部分)



如何设置断点

为设置一个断点，在你要设置断点的行的行号码区域单击，如果选中的区域是断点的位置，行号码区域的颜色将从黑灰色转变为淡蓝色并且断点将在 PLC 中激活。

删除断点

相应地，为删除一个断点，在有断点的行的行号码区域单击，断点将删除。

设置和删除断点也可以通过菜单命令('联机' '删除断点')、通过功能键<F9>或通过在工具栏中的符号来选择。

断点处的状态

在 PLC 中到达断点时，屏幕上会显示断点和相应的行，行的行号码区域将显示为红色，用户程序在 PLC 中停止。

如果程序到达断点，处理过程可以通过 '联机' '运行'命令来继续。另外，用 '联机' '跳过'或 '进入'命令，你可以使程序在下一个断点位置运行。如果你所在处的指令是 CAL 或如果在行中直到下一个断点位置处有一个功能调用，你可以使用命令“跳过”越过功能调用。用“进入”，将进入打开的 POU 分支。

在文本编辑器中的行号码

文本编辑器的行号码给出了 POU 的执行部分的每个文本行的号码。

在脱机模式下，在一个特定的行号码上单击将会标记整个文本行。

在联机模式下，行号码的背景颜色指示了每个行的断点的状态。

颜色的标准设置有：

黑灰色：这个行是断点的可能的位置。

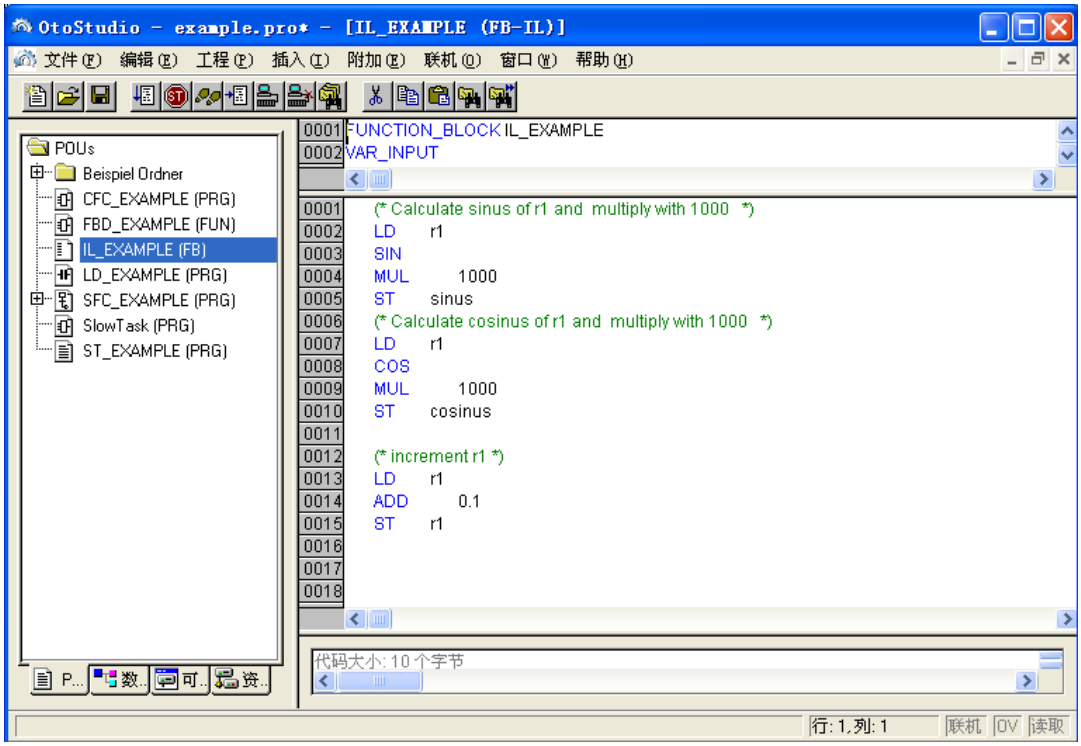
淡蓝色：断点已经在这个行中设置。

红色： 程序到达了这个断点。

在联机模式下，单击鼠标将会改变这个行中的断点状态。

6.4.1 指令表编辑器

下面是在 OtoStudio 中相应的编辑器 IL 中编写 POU 的窗口。



POU 的所有编辑器有声明和主体部分，它们被屏幕分割器分割开。

指令列表编辑器是具有文本编辑器窗口功能的文本编辑器，在内容菜单（鼠标右键或<Ctrl>+<F10>）中可以找到最重要的命令，多行 POU 调用也是可以的：

例如：

```
CAL CTU_inst(
```

```
CU:=%IX10,
PV:=(
LD A
ADD 5
)
);
```

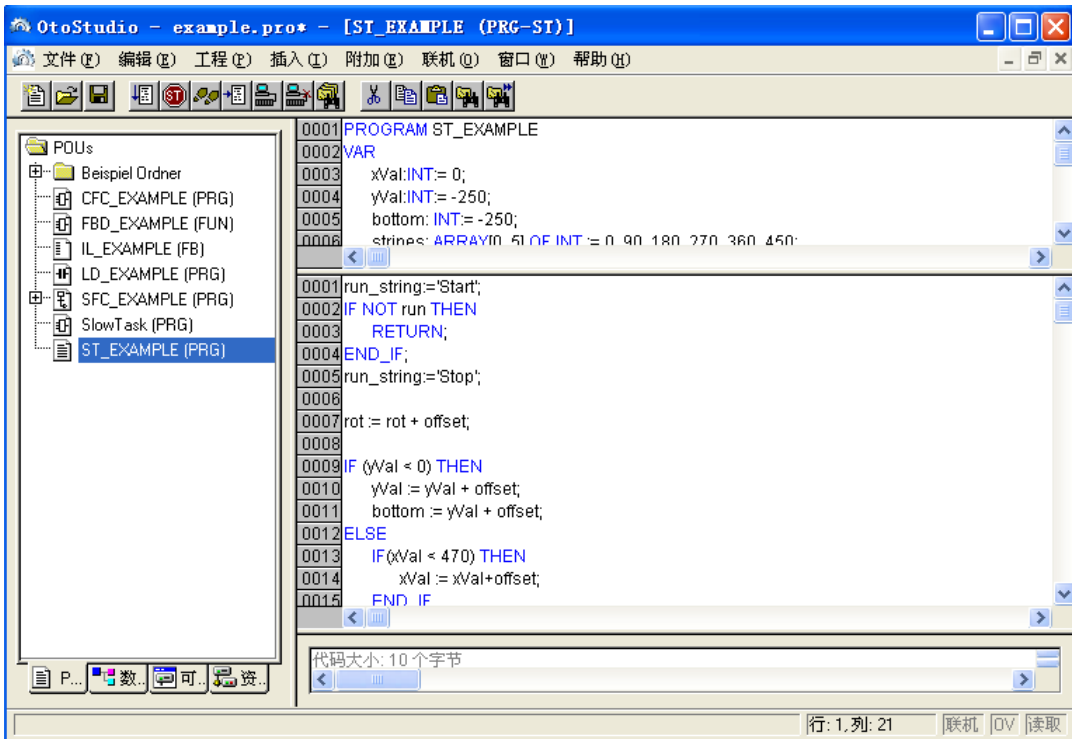
联机模式下的 IL

用命令 '联机' '溢出控制'，附加的域插入到 IL 编辑器中左边部分的每一个行中。

在联机模式下关于 IL 的更多信息，参照 '联机模式下的文本编辑器'

6.4.2 结构化文本编辑器

下面是在 OtoStudio 中相应的编辑器 ST 中编写 POU 的窗口。



POU 的所有编辑器有声明和主体部分。它们被屏幕分割器分割开。

结构化文本编辑器是具有文本编辑器窗口功能的文本编辑器，在内容菜单（鼠标右键或<Ctrl>+<F10>）中可以找到最重要的命令。

6.5 图形化编辑器

图形化导向的语言编辑器、顺序功能图 SFC、梯形图和功能模块图和自由图形化功能模块图有许多共同点。在下面的段中将总结这些特点；关于 LD、FBD、CFC 和

顺序功能图表语言 SFC 在分开的部分中讲述，句式色彩支持图形化编辑器的执行部分。

参考:

缩放

网络

标志

网络注释,'扩展' '选项'

'插入' '网络（后面）' 或者'插入' "网络 (前面)"

联机模式下的网络编辑器

功能块图编辑器

梯形图编辑器

顺序功能图编辑器

连续功能图编辑器

缩放

在语言 SFC、LD、FBD、CFC 和在可视化中的对象如 POU、动作、转换等可是通过缩放功能来放大或缩小尺寸，执行部分的窗口内容中的所有元素都将受到影响，声明部分保持不变。

在标准形式中，每个对象以缩放级别 100%显示，缩放级别在工程中存储为对象的属性。工程文档的打印预览通常显示为 100%。

缩放级别可以在工具栏中通过选择列表来设置，它的值可以从 25% 到 400%之间选择，在 10%和 500%之间的值可以通过手工输入。

如果光标停留在在图形化语言创建的对象上或停留在一个可视化对象上，就可以选择缩放级别。

在联机模式，每个对象根据设置的缩放级别来显示，联机功能没有任何限制。

当使用智能鼠标时，通过按<CTRL>键并同时向前或向后旋转鼠标轮能够放大/缩小对象。

网络

在 LD 和 FBD 编辑器中，程序由一系列的网络组成，每个网络都由左边的网络编号指示，并且还包括逻辑或算术表达式的结构、程序、功能或功能模块调用、跳转或返回指令。

标签

每个网络都有一个标签，标签可以为空，通过单击网络编号旁的网络的第一行来编辑标签，然后在冒号后面输入一个标签。

注释

每个网络都可以添加一个多行的注释，在通过命令“附加”“选项”打开的对话“功能块和梯形图选项”中，可以设置关于注释的设置。

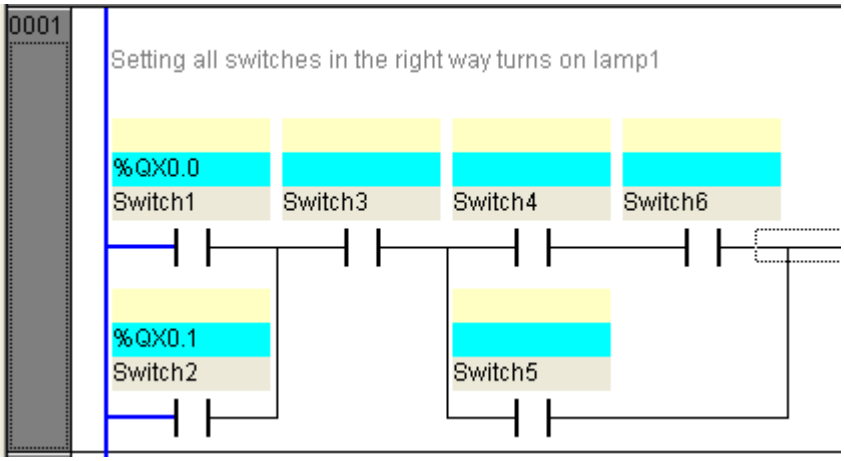
功能块和梯形图选项

在最大注释尺寸区域中，可以输入注释行的最大的数字（默认值为 4），在最小注释尺寸区域中可以输入通常要保留的行数字。如果，输入了 2，那么，在每个网络的开始处在标签行后面都有两个空行，默认的值为 0，这样做的优点是使更多的行显示在屏幕区域。

如果最小注释数字比 0 大，为了输入一个注释只需在注释行行单击然后输入注释，否则，就需要首先选择要输入注释的网络，并使用命令“插入”“注释”来插入注释行。和程序文本相比，注释显示为灰色。

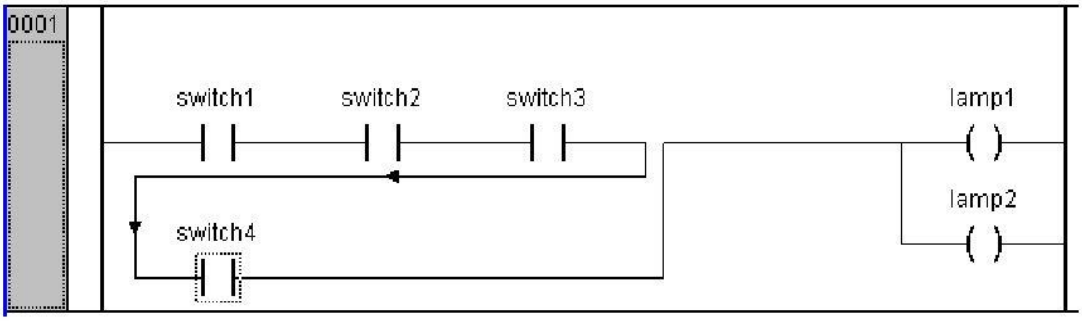
在梯形图编辑器中可以为每个接点和线圈添加注释，只要激活选项注释每个接点并插入到编辑区域变量注释行输入保留用来显示注释的行的编号，当完成这个设置时，注释区域将显示在编辑器中每个触点和线圈的上面。

如果激活选项注释每个接点，在梯形图编辑器中可以定义哪个行编号用于接点或线圈的变量名，这用来显示由多行组成的长名字。



在梯形图编辑器中，通过激活选项网络自动换行，在网络的长度超出给定的窗口尺寸并且一些元素看不见的情况下可以自动换行。

自动换行例子

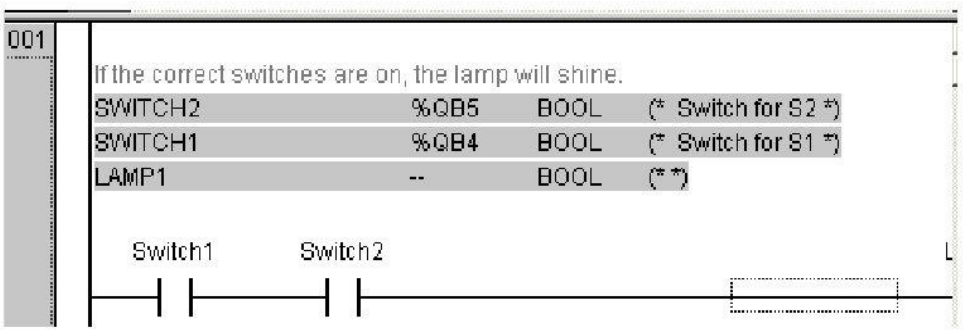


输入地址后用符号替换: (只适用于梯形图编辑器): 如果此项激活,那么在接点或线圈处输入地址后,如果地址已经分配给了一个变量,地址将被变量名替换,否则保留地址.

接点注释设置成变量注释: 如果此项激活, 将在接点或线圈注释区内显示变量声明时编写的注释.注释可以再编辑(参看上面的图片) ,为达到这个目的,必须激活'每个接点的注释'选项.注意:即使变量在声明区没有注释,原有注释也会自动被变量声明区中的注释替换!

显示符号地址: (只适用于梯形图编辑器): 如果此项激活同时接点或线圈的变量与一个地址对应,那么将显示在变量名上面.(参看上面的图片).

打印输出时按行显示变量注释: 如果此项激活,每个网络中用到的变量各占一行,被显示在网络上面,显示的内容:名称,地址,数据类型和注释.这和变量声明一样. 此项功能在工程打印输出时很有用. 例如:



应用选项:

确定: 按此按钮将当前设置用于当前 POU 并关闭对话框.

应用: 按此按钮将设置用于整个工程. 按此按钮后将打开一个提示框来确认是否修改设置.

'插入' '网络（后面）' 或者'插入' "网络 (前面)

快捷方式：<Shift>+<T> (网络后面)

为了在 FBD 或梯形图编辑器中插入一个新网络，选择命令“插入”“网络（后面）”或'插入' "网络 (前面)"，取决于你想在当前网络之前或之后插入一个新网络。当前的网络可以通过在网络编号上单击来改变，可以通过在编号下面的点划矩形框来识别它，使用<Shift key>和单击鼠标可以选中从当前到单击的网络的整个区域。

联机模式下的网络编辑器

在 FBD 和 LD 编辑器中只能为网络设置断点，设置断点的网络的编号区域显示为蓝色。处理过程在断点设置的网络附近停止，在这种情况下，网络编号区域显示为红色，通过单步处理可以在网络之间跳转。

在进入和退出网络 POU（程序组织单元）时，所有的值都被监视。

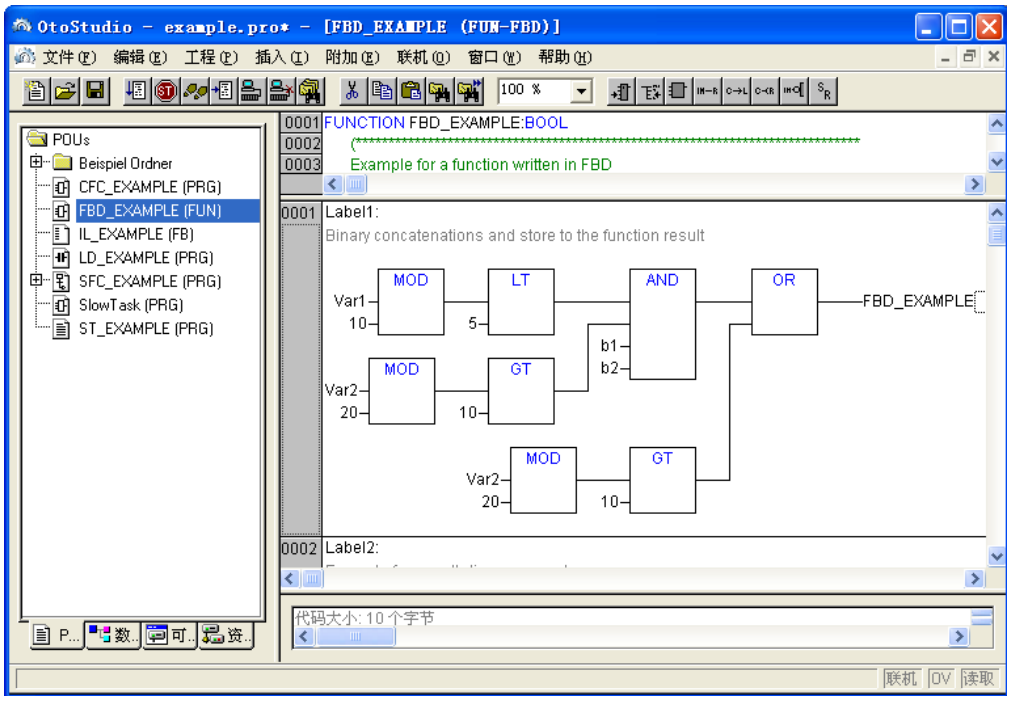
当监视表达式或位地址变量时，应注意以下方面：在表达式中，例如，a AND b，作为

变换条件或功能模块输入，整个表达式的值通常显示（如果 a AND b 的值为 TRUE 时， a AND b 显示为蓝色或：=TRUE）。对于位地址变量，寻址的位值通常要被监视（例如，当 a 有值 4 时，a.3 显示为蓝色或带:=TRUE）'联机' '流程控制'命令启动流程控制，使用流程控制，可以查看当前网络连接线上传递的值。如果连接线没有传递布尔型的值，值将显示在一个特定的输入区域。未使用的变量的监视区域（例如，在功能 SEL 中）显示灰色的阴影，如果连接线传递布尔型的值，当它们传递 TRUE 时，将显示为蓝色。所以，在 PLC 运行时可以观察流程的信息。

如果用指针在变量上停留一会，在工具提示中将显示变量的类型、地址和注释。

6.5.1 功能模块图编辑器

下面是在 OtoStudio 中相应的编辑器 FBD 中编写一个 POU 的窗口



功能模块图编辑器是一个图形化的编辑器，每个网络包含了显示、逻辑或算术表达式、功能模块的调用、功能、跳转、返回指令。在内容菜单（按鼠标右键或<Ctrl>+<F10>）中可以找最重要的命令。

可以在 FUP-和 KOP 之间切换 FUP-POU 的显示，离线模式和联机模式相同。

参照:

FBD 的当前位置

如何设置 FBD 的当前位置

'插入' '赋值'

'插入' '跳转'

'插入' '返回'

'插入' '框'

'插入' '输入'

'插入' '输出'

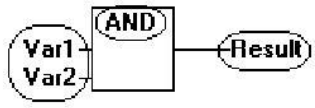
'扩展' '否定'

'扩展' '设置/复位'
'扩展' '预览'
'扩展' "快速'
'扩展' '打开实例'
'扩展' '选项'
剪切，复制，粘贴和删除
联机模式下的功能块图

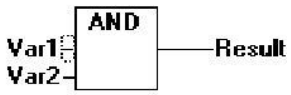
FBD 的当前位置

每个文本是一个可能的光标位置，选中的文本以蓝色背景表示并且当前可以被修改。通过点矩形框也能识别当前光标位置，下面是光标所有可能位置的例子：

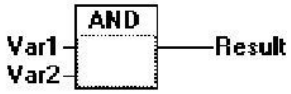
1）每个文本区域（加黑色框的可能指针位置）：



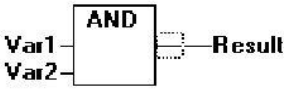
2）每个输入：



3）每个操作数、功能或功能模块：



4）输出，如果随后有赋值或跳转：



5）行在赋值、跳转、返回指令上发生交叉：

6）在最外面对象的网络图的右边（“最后的指针位置”，与选择网络的指针位置相同）：

7）在赋值附近的线性交叉：

如何设置 FBD 的当前位置

通过单击鼠标或使用键盘可以在特定的位置放置光标，在任何时候，使用箭头键可以在选择的方向上跳到最近的光标位置。通过这种方式可以访问所有的光标位置包括文本区域。如果最近的光标位置被选中， 就可以使用<up>或 <down>方向键来选择先前的网络或随后

的网络的最近光标位置。
空的网络只包含三个问号“？？？”，在它们之后单击，最近的光标位置被选中。

'插入' '赋值'

符号 ： 

快捷方式：<Ctrl>+<A>

这个命令插入一个赋值。

插入赋值依赖于选中的位置（参照 'FBD 的当前位置'），在选中的输入端附近（光标位置 2）、在选中的输出端（光标位置 4）之后或在网络的末端（光标位置 6）可以直接插入赋值。

对于一个插入的赋值随着选中输入的文本“？？？”被选中，赋值可以被将要赋值的变量替代，也可以使用输入帮助。

使用命令“插入”“输出”给当前存在的赋值插入一个附加的赋值。

'插入' '跳转'

这个命令插入一个跳转。

插入依赖于选中的位置（参照 'FBD 的当前位置'），在选中的输入端（光标位置 2）附近、在选中的输出端（光标位置 4）后或在网络的末端（光标位置 6）可以直接插入跳转。对于插入的跳转，随着选中输入的文本“？？？”选中跳转，它可以被要赋予它的标签替代。

'插入' '返回'

这个命令插入一个返回指令。

插入返回指令依赖于选中的位置（参照 'FBD 的当前位置'），在选中的输入端附近（光标位置 2）、在选中的输出端（光标位置 4）之后或在网络的末端（光标位置 6）可以直接插入返回指令。

'插入' '框'

用这个命令能插入运算符，功能，功能模块和程序。首先，它通常插入一个“AND”运算符，这可以通过选择和类型文本的覆盖来改变到其它操作符，功能，每个功能模块和每个程序中。

通过使用输入帮助(<F2>)可以选择期望的 POU。如果新选择块中有其它输入的最小数字，这些将绑定。

如果新块中有输入的较小数字，最近的输入将被删除。

在功能和功能模块中，显示输入和输出的正式名字。

在功能模块中的方框上存在一个可编辑的实例区域。通过改变类型文本来调用一个没名的功能模块，将显示带有两个输入和给定的类型的运算符方框。如果实例区域被选中，可以通过<F2>和变量选择的类别获得输入帮助。

最新的 POU 插入到选中的位置：

如果选中一个输入端（光标位置 2），POU 插入到这个输入端的附近。这个 POU 的第一个输入端连接到选中输入端左边的分支。新 POU 的输出连接到了选中的输入端。

如果选中了一个输出端（光标位置 4），那么 POU 插入到输出端之后。POU 的第一个输出端与选中的输出端连接。新 POU 的输出端与选中的输出端连接的分支相连接。

如果 POU、功能或功能模块被选中（光标位置 3），旧的 POU 将被新的 POU 替代。反之将会与它们被替代之前的相同方式相连接。

如果旧元素的输入端比新元素多，不能连接的分支将被删除，相同的分支为输出端保持真值。

如果选中了跳转或返回指令，POU 将会插入到这个跳转或返回指令之前。POU 的第一

个输入端与选中元素的左边的分支相连接。POU 的输出端与选中的元素的右边的分支相连接。

如果选中了网络的最近光标位置（光标位置 6），POU 将会插入到最后元素的后面。POU 的第一个输入端与选中位置的左边的分支相连接。

不能连接的 POU 的所有输入端出现文本“？？？”。这个文本必须改变为期望的常量或变量。

插入的 POU 的右边分支，分支将会分配给第一个 POU 输出端。否则输出端保持不赋值。

'插入' '输入' 

这个命令插入一个输入端运算符。随着运算符，输入端的数目可以不同（例如，ADD 可以后 2 个或多个输入端。）

为了通过输入来扩展这样一个运算符号，你必须选择输入端，在它的附近插入附加输入端（光标位置 1）或者你必须选择运算符本身（光标位置 3），如果要插入一个低输入端（参照 'FBD 的当前位置'）。插入的输入端分配了文本“？？？”。这必须改变为期望的常量或变量。你也可以使用输入帮助来做这些。

'插入' '输出'

符号 ： 

这个命令用来添加一个附加的赋值到已存在的赋值中。

'附加' '取反'

符号 ： 

快捷方式：<Ctrl>+<N>

用这个命令可以对输入、输出、跳转或返回指令进行否定操作，否定的符号是在连接处的一个小圆圈。

如果选中了一个输入（光标位置 2）（查看 'FBD 的当前位置'），随后这个输入将被否定。

如果选中了一个输出端（光标位置 4），那么这个输出端将被否定。

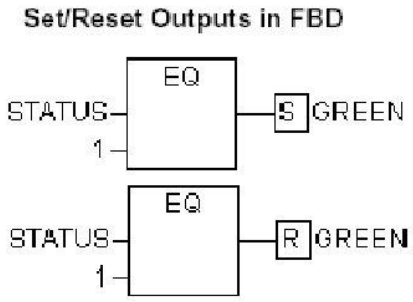
如果一个跳转或返回被标记，那么跳转或返回将被否定。

否定可以通过重新否定来取消。

'附加' '置位/复位'

符号 ： 

用这个命令可以象设置或复位那样来定义输出，设置的输出用[S]表示，复位输出端用[R]表示。



如果属于它的栅格返回了 TRUE，输出端设置为 TRUE，输出将保持这个值，即使栅格跳回了 FALSE。

如果属于它的栅格返回了 FALSE，输出端设置为 FALSE。输出现在保持这个值，即使栅格跳回了 FALSE。

随着重复执行这个命令，输出将在设置、复位和正常输出之间改变。

'附加'预览'

这个命令为在 FBD 编辑器中创建的 POU 选择显示在梯形图或者 FBD 编辑器中，离线模式和联机模式功能相同。

'附加' '打开实例'

这个命令与“'工程' '打开实例' 相同。

如果光标放置在文本编辑器中的功能模块的名字上或如果功能模块框在图形编辑器中被选中，在内容菜单(<F2>)或在'附加'菜单中可用到这个命令。

剪切，复制，粘贴和删除在菜单项目“编辑”下可以用到“剪切”,“复制”“粘贴”和“删除”如果选中了一个交叉线（光标位置 5），那么位于交叉行下面的赋值、跳转或返回将会被剪切、删除或复制。

如果选中了一个 POU（光标位置 3），选中的对象自身将被剪切、删除或复制，同时也包括在输入端的所有独立的分支和第一种情况外的分支。

否则，在光标位置附近的所有分支将会被剪切、删除或复制。

在复制或剪切之后，删除或剪切的部分位于剪贴板上，可以随意粘贴它。

首先选中粘贴点，有效的粘贴点包括输入和输出端。

如果 POU 被加载到了剪贴板上(这种情况下所有连接的分支除了第一条外都一块位于剪贴板上)，第一个输入连接到粘贴点之前的分支上。

否则，位于粘贴点附近的所有分支将被剪贴板中的内容替代。

最后粘贴的元素将连接到粘贴点附近的分支上。

注意：下面的问题可以通过剪切和粘贴来解决：一个新的操作数插入到网络的中间，分支位于操作数的右边，现在将和第一个输入相连接，但应该和第二个输入相连接。你可以选择第一个输入端并执行命令“编辑”“剪切”，这样，你可以选择第二个输入端并执行命令“编辑”“粘贴”，分支就连接到了第二个输入端。

联机模式下的功能块图

在功能模块图，断点只能设置到网络上。如果已经在网络上设置了断点，网络编号区域将显示为蓝色，处理过程将停止在设置断点的附近。在这种情况下，网络编号区域将显示为红色。使用 单步，你可以从一个网络跳到另一个网络。

显示每个变量的当前的值，例外：如果功能模块的输入是一个表达式，只有表达式中第一个变量才被监视。

在变量上双击将打开写变量的对话框，在这里可以改变当前变量的值。如果是布尔变量的情况，不出现对话框，这些变量被选中。

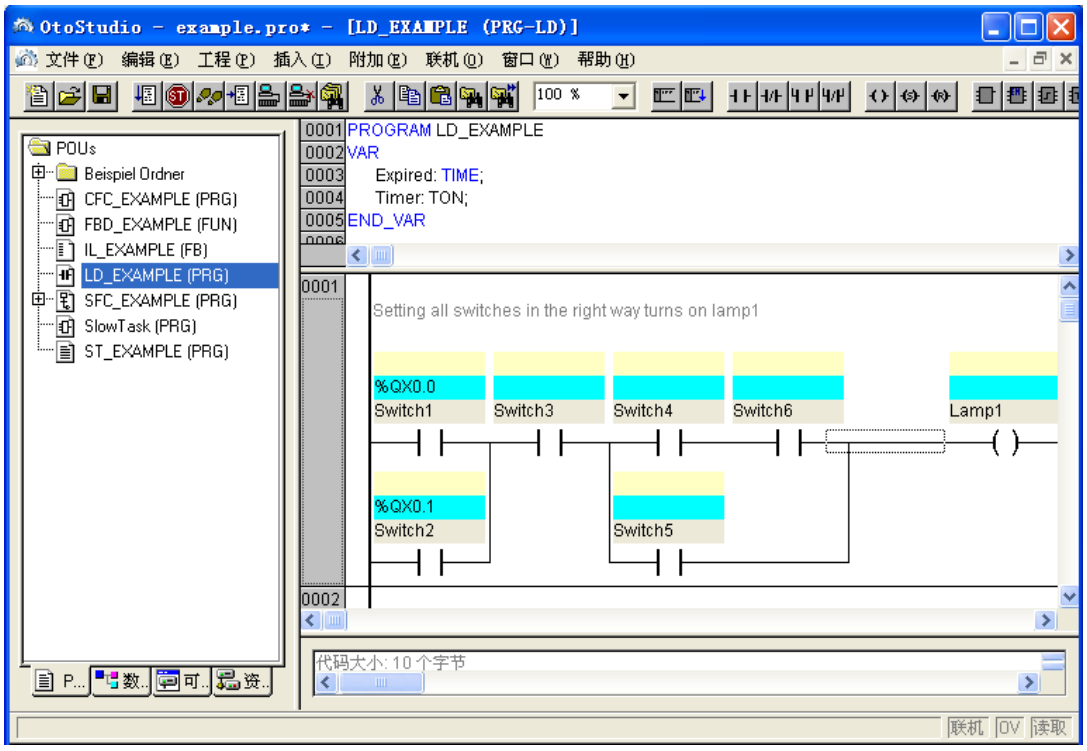
如果使用了命令“'联机' '写入新值'，新值将变为红色并保持不变。所有的变量将放入选择列表中，然后重新显示为黑色。

用命令 '联机' '流程控制'来开始流向控制，你可以查看当前网络上正在传递的值。如果连接线上没有传递布尔值，值将显示在一个特殊的插入区域。如果线上传递的是布尔型的值，当它们传递 TRUE 值，以暗蓝色显示。通过这个方法，你可以在 PLC 运行时知道流程的信息。

如果鼠标在变量上停留一会，变量的类型、地址和注释将显示在工具提示中。

6.5.2 梯形图

下面是在 OtoStudio 编辑器 LD 中编写 POU 的窗口



POU 的所有编辑器包括声明部分和主体部分，它们被屏幕分割器隔开。

梯形图编辑器是一个图形化的编辑器，在内容菜单（按鼠标右键或<Ctrl>+<F10>）中包含有最重要的命令。

关于元素的信息，参照梯形图。

参照:

LD 编辑器的当前位置

元件的移动

'插入' '网络(前)'

'插入' '网络 (后)'

'插入' '连接'

'插入' '并联'

'插入' 'LD 的功能块'

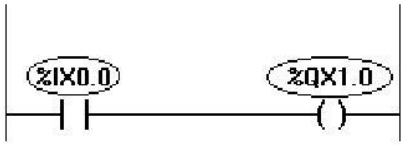
'插入"线圈'

以 EN 输入的 POUs
'插入' ' LD 中 EN 的框'
'插入' '在 LD 中插入块’
'插入' '跳转'
'插入' '返回'
'扩展' '向后粘贴'
'扩展' '向下粘贴'
'扩展' '向上粘贴'
'扩展' '否定'
'扩展' '设置/复位'
'扩展' "快速'
'扩展' '打开实例'
联机模式下的梯形图

LD 编辑器的当前位置

下面的位置是可能的光标位置，在功能模块和程序访问中可以作为触点的位置、具有 EN 输入的 POU 和和在功能模块图一样处理的其它的 POU 。编辑这个网络部分可以在 6.4.2 章节找到详细的信息，FBD 编辑器。

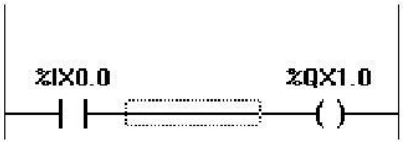
1. 每个文本区域（在黑色框中是光标位置）



2. 触点或功能模块

3.线圈

4.触点和线圈之间的连接线

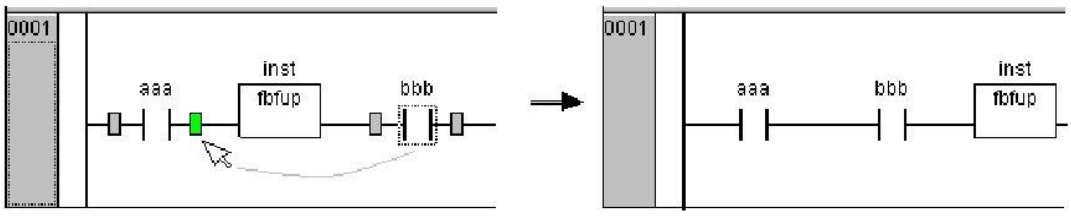


梯形图以特定方式使用下面菜单命令：

元件的移动

在 LD 的 POU 中可以通过拖放来移动元素到一个不同的位置。

为了移动元素，选择期望的元素（触点、线圈、功能模块）并按住鼠标键拖动它离开当前的位置，在 POU 的所有将要移动到的所有可能位置的网络内的元素，将会通过灰色填充的矩形来指示，移动元素到这些位置中的一个并释放鼠标键，元素将插入到新位置。



'插入' '网络(前)'

符号:

这个命令在梯形图中插入一个 网络。如果已经存在网络，新插入的将在当前网络前面。

'插入' '网络 (后)'

符号:

这个命令在梯形图中插入一个 网络。如果已经存在网络，新插入的将在当前网络后面。

'插入' '常开接点', 注释

快捷方式：<Ctrl>+<O>

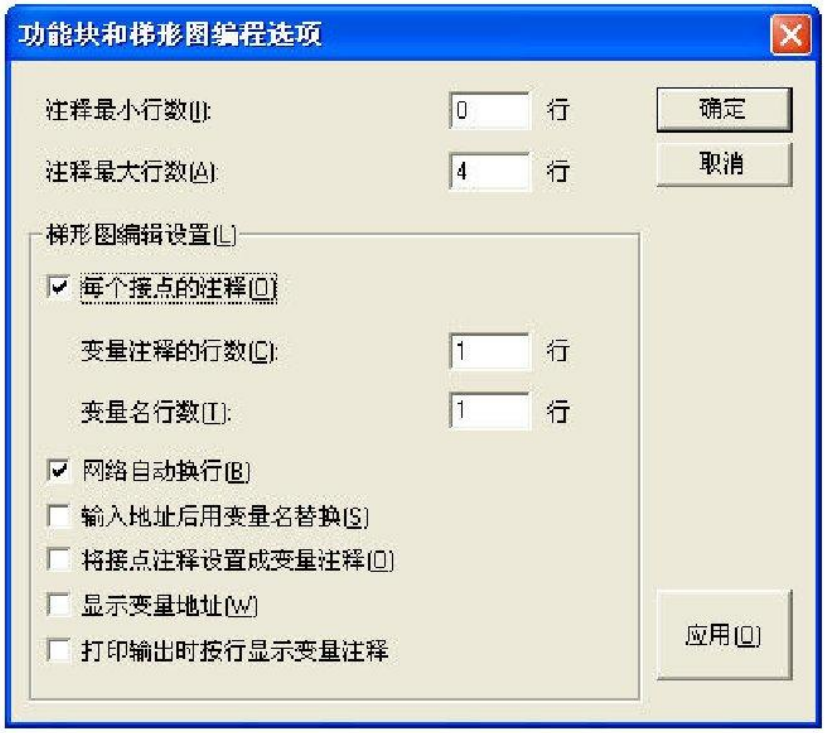
在 LD 编辑器中使用这个命令来在网络图中标记的位置插入一个 接点。

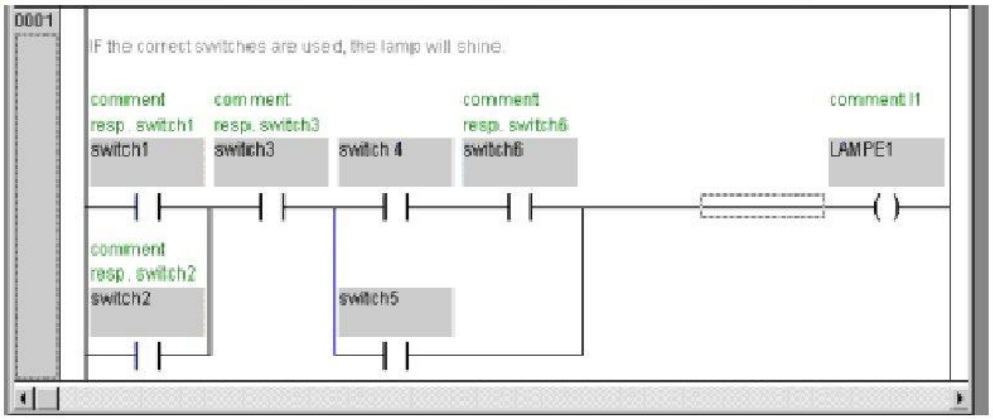
如果 标记的位置是一个 线圈或在触点和线圈之间连接线，新触点将会连续地连接到先前的触点。

触点预设文本为“？？？”，你可以在这个文本上单击并改变它为期望的变量或期望的常量，你也可以使用输入助手，也能激活对话 '功能块和梯形图选项'('附加' '选项')中的选项注释每个连接并在变量注释下画线来为变量名保留一定数量的行，如果使用长变量名，这可能对保持网络简略是有用的。

可以在梯形图选项中激活选项'网络自动换行'。

下面是在 FBD 或梯形图网络中的选项对话和结果显示的例子：





'插入' '常闭接点'

符号:

快捷键: <Ctrl> + <G>

这个命令插入一个 常开接点。像 ’ 插入’’ 接点’ 和 ’ 附加’’ 取反’ 一样，也可插入一个常闭接点

'插入' '常开并联接点'

符号 ：

快捷方式 ： <Ctrl>+<R>

在 LD 编辑器中使用这个命令来在网络中标记位置处插入一个平行的 接点。

如果 标记的地方是一个 线圈或接头和线圈之间的连接，新的接头将与整个先前接头连接平行进行连接。

接点预设为文本“？？？”，可以单击文本并改变它为期望的变量或期望的常量，你也可以使用输入助手来做这些。

'插入' '常闭并联接点'

快捷键: <Ctrl> + <O>

这个命令插入一个 常闭并联接点。像 ’ 插入’’ 常开并联接点’ 和 ’ 附加’’ 取反’ 一样，也可插入一个常闭并联接点

'插入''线圈'

符号 ： 

快捷方式 ： <Ctrl>+<L>

在编辑器 LD 中使用这个命令来插入一个与先前线圈平行的 线圈。

如果 标记的位置是线圈和 接点之间的连接，新线圈将插入到最后，如果标记的位置是线圈，新线圈将直接插入到它的上面。

线圈的默认文本是“？？？”。可以在文本上单击并改变它为期望的变量。也可以使用输入帮助来做这些。

也可以为线圈添加一个单独的注释，详细的讲述参照“插入” “接点”。

'插入' ''置位线圈'

符号: 

快捷方式: <Ctrl> + <I>

这个命令插入一个 置位线圈。像 '插入' '线圈'和 ' 附加' '置位/复位'一样，也可插入一个置位线圈.

'插入' '复位线圈'

符号: 

这个命令插入一个 复位线圈。像 '插入' '复位'和 ' 附加' '设置/复位'一样，也可插入一个复位线圈.

'插入' '功能块'

符号： 

快捷方式：<Ctrl>+<B>

用这个命令来插入一个运算符、功能模块、功能或作为 POU 的程序。在接点和线圈之间的连接，或触头与触头之间的连接，必须标记出来。新 POU 在起先有名称 AND，可以改变它为其它的名称，也能使用输入帮助来做这些。标准的和自定义的 POU 都是可用的。POU 的第一个输入放置在输入连接上，第一个输出在输出连接上，这些变量必须定义为 BOOL 类型。POU 的所有其它的输入和输出被文本“？？？”填充，这些优先的条目能改变为其它的常量、变量或地址。也可以使用输入助手来做这些。

带 EN 输入的 POUs

如果你想使用你的 LD 网络作为一个 PLC 来调用其它的 POU，必须结合一个带 EN 输入的 POU，这样的 POU 与 线圈平行连接，在这个 POU 之后你可以展开网络，就象在功能模块图中那样。在菜单项目“插入”“插入块”下，在 EN POU 的地方能找到插入命令。
操作符、功能模块、程序或带 EN 输入的功能与在功能模块图中的相应 POU 以相同的方式执行操作，除了它们的执行在 EN 输入上控制。这个输入附加到线圈和触点之间连接的地方，如果这个连接携带信息"On"，将会计算 POU 的值。

如果创建的 POU 带有 EN 输入，这个 POU 可以用来创建网络。这意味着来自通常操作符、功能和功能模块的数据能流进 EN POU 并且 EN POU 也能传送数据到这样的通

用的 POU 中。如果，你想在 LD 编辑器中编写一个网络，就象在 FBD 中那样，你只需在新网络中首先插入一个 EN 操作符。
随后，从这个 POU 开始，能象在 FBD 编辑器中那样继续从你的网络创建。

'插入' ' LD 中 EN 的框'

符号： 

用这个命令可以把一个功能模块、操作符、功能或带 EN 输入的程序插入到 LD 网络中。

标记的位置是触点和线圈（光标位置 4）或线圈（光标位置 3）之间的连接。新 POU 平行插入到在线圈下面，它包含最初的名称“AND”。你也可以改变这个名称为其它的名字，也可以使用输入帮助来做这些。

'插入' '在 LD 中插入块’

使用这个命令你可以插入附加的元素到已经插入元素的 POU（也包括带 EN 输入的 POU）中，在这个菜单项目下的命令就象在功能模块图中相应命令一样可以在同一光标位置执行。

用输入可以添加一个新的输入到 POU 中。

用输出可以添加一个新输出到 POU 中。

用 POU，插入一个新 POU，过程与“插入“ ”POU“讲述的一样。

用分配你能给变量插入一个值。首先，显示为三个问号“？？？”，可以用期望的变量来编辑和替换，也可以使用输入帮助来做这些。

'插入' '上升沿触发器'

符号: 

这个命令时插入一个在引入信号中检测上升沿(FALSE -> TRUE)的 R_TRIG 功能块。像 '插入' '功能块'一样，它也能用于插入任何一个可用功能块。

插入\下降沿检测触触发器

符号: 

这个命令时插入一个在引入信号中检测下降沿(FALSE -> TRUE)的 F_TRIG 功能块。像 '插入' '功能块' 一样，它也能用于插入任何一个可用功能块。

'插入' '计时器 (TON)'

符号: 

这个命令是插入一个 TON 计时器功能块。它负责接通延迟（延迟引入信号的通过），像 '插入' '功能块'一样，它也能被用作插入 TON 模块。

'插入' '跳转'

用这个命令能插入一个平行的跳转在选中的 LD 编辑器中，在平行分支中，在先前 线圈的末端。如果引入行传递值"On"，跳转将跳转到指示标签处执行，标记的位置必须是在触点和线圈之间的连接。跳转用文本“？？？”表示，可以单击这个文本并改为期望的标签。

'插入' '返回'

在 LD 编辑器中，可以使用命令来在先前线圈的末端的平行部分插入一个返回指令。如果输入线上传递值“On”，在这个网络中 POU 的处理过程中断。
标记的位置必须是在触点和线圈之间的连接部分。

'附加' '粘贴在后面'

在编辑器中使用这个命令来粘贴剪贴板中的内容作为在标记的位置下面的连续接点，这个命令只有在剪贴板中的内容和标记的位置是接点的网络组成时才起作用。

'附加' '粘贴在下面'

快捷方式 ： <Ctrl>+<U>

在 LD 编辑器中使用这个命令来插入剪贴板的内容作为在标记位置之下的行接点，这个命令只有在剪贴板的内容和标记位置是接点的网络组成时才有效。

'附加' '粘贴在上面'

在编辑器中使用这个命令来插入剪贴板的内容作为标记位置下面的平行接点，这个命令只有在剪贴板的内容和标记的位置是接点的网络组成时才有效。

'附加' '取反'

符号 ： 

快捷方式 ： <Strg>+<N>

使用这个命令来对一个触点、线圈、跳转或返回指令，或在当前 光标位置的 EN POU 的输入或输出进行否定操作。

在 线圈的圆括号或 接点的直线之间，出现一个斜线((/) or |/|)。如果这些是跳转，返回指令，或 EN POUs 的输入或输出，在连接处出现一个小圆圈，和在 FBD 编辑器中的一样。线圈现在在各自的布尔型变量中写入输入连接的否定值，如果各自的布尔型变量传递值 FALSE，否定触头切换输入的状态为输出。

如果标记了一个跳转或一个返回，这个跳转的输入或返回将被否定。

可以对否定进行否定来取消否定。

'附加' '置位/复位'

符合： 

如果在 线圈上执行这个命令，会接收一个 设定线圈。这个线圈在各自布尔变量从不覆盖值 TRUE。这意味着一旦为变量设置了值 TRUE，它将一直保持 TRUE。设置线圈在线圈符号中用“S”表示。如果你再次执行这个命令，会得到一个复位线圈。这个线圈在各自的布尔变量中从不覆盖值 FALSE。这就是说，一旦为这个变量设置了值 FALSE，它将一直保持 FALSE，复位线圈在线圈符号中用"R"表示。

如果你重复执行这个命令，线圈将在设置和复位和常态之间改变。

联机模式下的梯形图

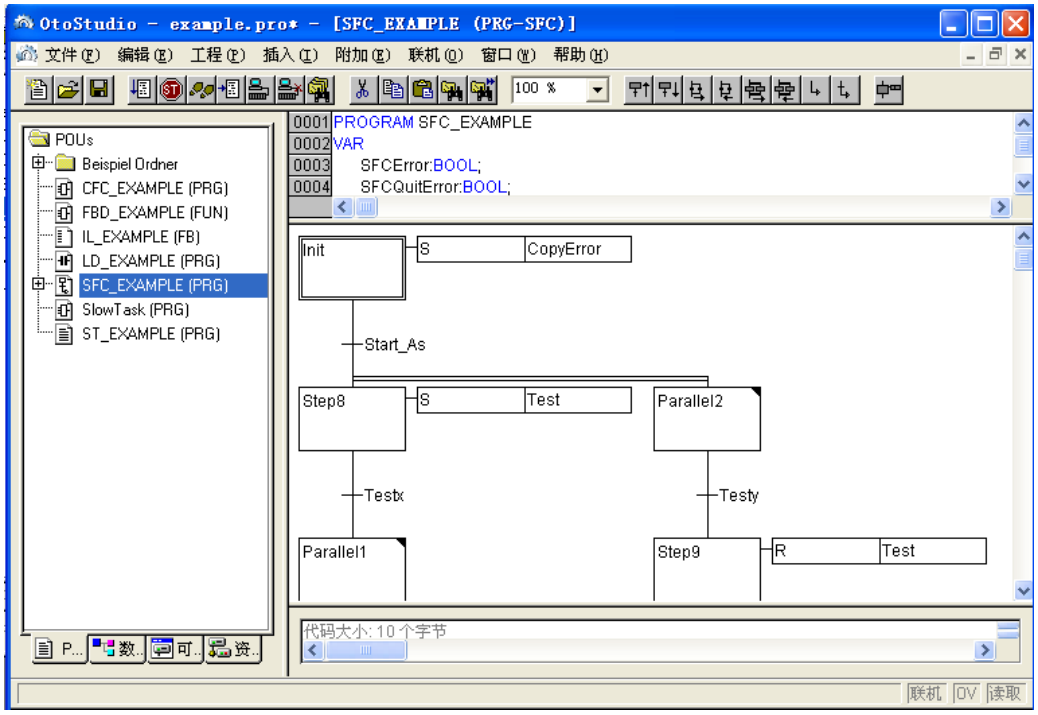
在联机模式下，在梯形图中的在“On”状态的触点和线圈以蓝色表示，同样地，所有

传递“On”的线也涂上蓝色，在 功能模块的输入和输出处，相应变量的值显示出来。
断点只能在网络上设置，通过使用步，可以从网络跳到另一个网络。

如果你把鼠标在变量上停留一会。变量的类型、地址、注释将在工具提示中显示出来。

6.5.3 顺序功能图表编辑器

下面是在 OtoStudio 中相应的编辑器 SFC 中编写 POU 的窗口：



POU 的所有编辑器包含一个声明部分和主体部分，它们被屏幕分割器隔开。

顺序功能图表编辑器是一个图形化的编辑器，在内容菜单（按鼠标右键或<Ctrl><F10>）中能找到最重要的命令。工具提示显示缩放模式中全名或步的表达式、转换、跳转、跳转标签、标识符或相关的动作。离线模式和联机模式的显示一样。

顺序功能图表编辑器必须与 SFC 的特殊编辑器相一致，下面的菜单项目将会常用到。

参照:

在 SFC 中标记块

'插入' '转换的步 (前)'

'插入' '转换的步 (后)'

'插入' '选择分支(右)'

'插入' '选择分支(左)'

'插入' "平行分支(右)"

'插入' '并行分支(左)'

'插入' 'J 跳转'

'插入' '转换跳转'

'插入' '添加进入动作'

'插入' '添加退出动作'

'扩展' '粘贴并行分支(右)'

'扩展' '增加并行分支标志'

删除标志
'扩展' '向后粘贴'
'扩展' '快速移动/转换'
'扩展' "清除动作/转换"
'扩展' '步的特性'
'扩展' '时间总览'
'扩展' '选项'
'扩展' '关联动作'
'扩展' '使用 IEC-Steps'
联机模式下的顺序功能图

在 SFC 中标记块

标记的块是用虚线框包围的一组 SFC 元素。

通过移动鼠标到元素上并按鼠标左键，或使用方向键，可以选择一个元素（步、转换或跳转）。为了标记一组中多个元素，在标记的块上按<Shift>，并选择组的左下角或右下角的元素，这样就选中了包括这些元素的最小的组。

请注意，步只能和它之前或随后的转换一起 删除（delete）!

'插入' '步-转换 (前)'

符号： 

快捷方式 ： <Ctrl>+<T>

这个命令在 SFC 编辑器中的标记块的附近插入一个后跟转换的步。

'插入' '步-转换(后)'

符号 ： 

快捷方式 ： <Ctrl>+<E>

这个命令用来在 SFC 编辑器中的标记块附近第一个转换后的转换后插入一个删除步和转换条件步只能和前面或随后的转换一起被删除，移动选择框到步和转换周围并选择命令“编辑”“删除”或按<Del>键。

'插入' '选择分支(右)'

符号 ： 

快捷方式 ： <Ctrl>+<A>

这个命令插入一个可选分支在 SFC 编辑器中作为标记块的右分支。

标记块必须开始和结束于一个变换，新的分支组成一个转换。

'插入' '选择分支(左)'

符号 ： 

这个命令插入一个可选分支在 SFC 编辑器中作为标记块的左分支。

这个标记块必须开始和结束于转换，新分支组成一个转换。

'插入' "平行分支(右)"

符号 ： 

快捷方式 ： <Ctrl>+<L>

这个命令在 SFC 编辑器中插入一个平行分支作为标记块的右分支。

标记块必须开始和结束于步，新分支作成一个步。要使已创建的平行分支允许跳转，必须提供一个跳转标签。

'插入' '并行分支(左)'

符号 ： 

这个命令在 SFC 编辑器中插入一个平行分支作为标记块的左分支。

标记块必须开始和结束于步，新分支作成一个步，要使已创建的平行分支允许跳转，必须提供一个跳转标签。参照 '附加' '增加并行分支标志'

'插入' '跳转'

符号 ： 

在 SFC 编辑器中用这个命令在标记块所属的分支的末端插入一个跳转，这个分支必须是可选分支。

在插入的跳转中文本字符串“Step”可以被选中和被步名字或要跳转到平行分支的跳转标签替代。

'插入' '转换跳转'

符号 ： 

在 SFC 编辑器中使用这个命令来插入一个转换，在选择分支的末端后跟一个跳转，这个分支必须是选择分支。

在插入的跳转中插入的文本字符串“Step”能够被选中并且能被步的名字或要跳转到的平行分支的跳转标签替代。

'插入' '添加进入动作'

使用这个命令可以为步添加一个 进入动作。在步 激活之后，进入动作只执行一次，进入动作可以以你选择的语言执行。

带进入动作的步在左下角用"E"表示。

'插入' '增加退出动作'

用这个命令可以为 步添加一个 退出动作，在步失效之前，退出动作只执行一次，退出动作可以以选择的语言执行。

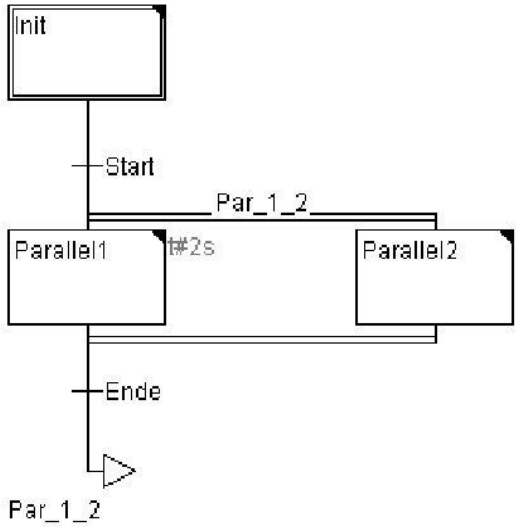
带退出动作的步在右下角用"X"表示。

'附加' '粘贴并行分支(右)'

这个命令粘贴剪贴板的内容作为标记块的右 分支，标记块必须开始和结束于一个步，剪贴板的内容同样也必须是开始和结束于步的 SFC 块。

'附加' '增加并行分支标志'

为了提供一个新的带跳转标签的插入平行分支，跳转发生在平行分支必须被标记并且命令“增加并行分支标志”被执行。在那个点，平行分支将会给定一个标准名字包含“并行”和一个附加序列号，它可以按照标识符名字的规则来编辑。



删除标志

通过删除标签名可以删除标签。

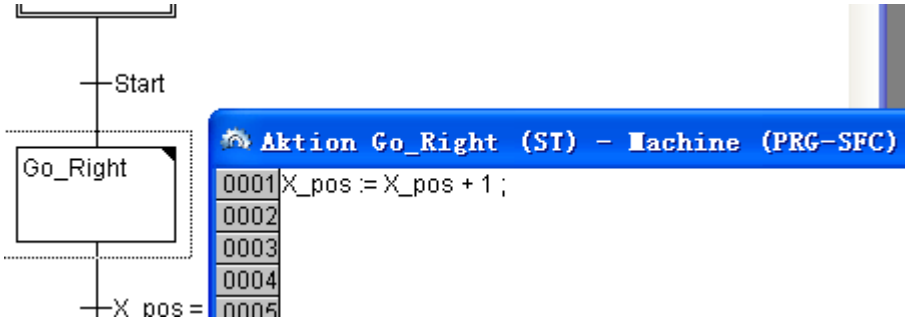
'附加' '粘贴在后面'

这个命令粘贴剪贴板上在第一个步之后或标记块的第一个转换后 SFC 块，（正常在标记块附近复制粘贴它。）根据语言标准，如果 SFC 结构是正确的，这个将开始执行。

'附加' '添加移动/转换'

快捷方式 ： <Alt>+<Enter>

标记块的第一个步的 动作或标记块的第一个转换的转换主体被加载到各自书写它们语言的编辑器中，如果动作或转换主体是空的，那么书写它们的语言必须被选中。



'附加' "清除动作/转换"

使用这个命令你能删除标记块的第一个步的 动作或第一个转换的转换主体的动作。

如果在一个步中，只执行了动作、进入动作或只执行退出动作，那么相同的将会通过命令被删除，否则出现一个对话框，你可以选择那个或那些动作要删除。

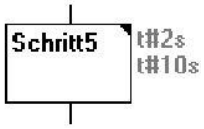
如果光标位于 IEC 步的动作上，那么只有这个连接被删除，如果 IEC 步和相关的 动作被选中，相关连接被删除，有多个动作的 IEC 步，将出现一个选择对话框。

'附加' '步的特性'

用这个命令能打开一个对话框，在对话框中能编辑标记步的属性。

可以利用在步属性对话框中的三个不同条目，在最小时间下，你可以输入这个步的处理过程占用的最小时间长度。在最大时间，你可以输入这个步的处理过程占用的最大时间长度。条目是时间类型，因此应该使用时间常量（例如，T#3s）或时间类型的变量。

在注释下，你可以为步插入一个注释，使用命令“附加” “选项”打开“顺序功能图选项”对话，在 SFC 编辑器中能决定显示步的注释还是步的时间设置。在右边，步的旁边，注释或时间设置将出现。如果超过了最大时间，用户可以查询设置的 SFC 标记。



例子中显示了一个在联机模式下至少执行 2 秒最多执行 10 秒的步，除了这来两个时间外，还显示了步已经激活了多长时间。

'附加' '时间总览'

用这个命令你能打开一个窗口，在这个窗口中能编辑 SFC 步的时间设置：

SFC POU 的所有步都显示出来，如果已经为步 输入一个时间边界，时间界限显示在步的右边（首先是时间下限，然后是时间上限）。你可以编辑这些时间界限。在期望的步上单击，步的名字显示在窗口的下面，在最小时间或最大时间区域里输入期望的时间界限，如果按 OK 关闭对话，所有的改动将被保存。

在例子中，步 2 和 6 有时间边界，Shift1 持续最少 2 秒，最多 10 秒。Shift2 持续最少 7 秒，最多八秒。

'附加' '选项'

用这个命令打开一个对话框，在对话框中能为 SFC POU 设置不同的选项。

在 SFC 选项对话框中能设置五个条目，在步高下，输入在 SFC 中一个 SFC 步占多少个行高，在这里是标准设置是 4。在步宽下，可以输入一个步占多少个列，标准设置是 6。也可以预设显示步。可以有三种选择的可能性：你可以选择不显示，或显示注释，或显示时间限制。最后两个条目与在 '扩展' '不的特性'显示的方式相同。

'附加' '关联动作'

用这个命令 动作和布尔变量可以连接到 IEC 步。

在 IEC 步右边连接一个附加的分开框，对于动作的连接，在左边的区域预设有限定词“N”并且在右边区域有“Action”的名字，可以通过 输入助手来改变它们。
一个 IEC 步最多能连接 9 个步。使用命令“工程”“添加动作”在对象管理器中创建 IEC 步的新动作，

'附加' '使用 IEC 步'

如果激活这个命令（在菜单项目的附近有一个对号表示），IEC 步将替代简单的步插入到步 转换和 平行分支上。

如果这个选项打开，当你创建一个新 SFC POU 时初始化步设置为 IEC 步。
这个选项保存在文件“OtoStudio.ini”中并且当 OtoStudio 重新启动时恢复。

联机模式的顺序功能图表

使用联机模式下的顺序功能图编辑器，当前 激活的步显示为蓝色。如果在命令下 '扩展' '选项'已经设置了它，那么步右边描述时间管理。在下和上边界已经设置的第三个时间指示器将出现，从这里能读出步已经激活了多长时间。

在上面的图片中描述的步已经激活了 8 秒和 410 毫秒，步在退出之前至少还要激活 7 分钟。

用命令“联机”“托拽断点”，将会在步上设置一个断点，或在一个动作的位置处用使用的语言来设置，处理过程在这个步之前停止或在程序中动作位置之前停止，断点设置所在步或程序的位置标记为亮蓝色。

如果在平行分支的数个步都激活，激活的步显示为红色，它的动作将处理下一个。

如果已经使用了 IEC 步，在联机模式所有激活动作将显示为蓝色。

使用命令“联机”“跳过”，它将跳到动作正在执行的下个步中。

如果当前位置是：

•在 POU 的线性处理中的步或在 POU 的最右边分支中的步，从 SFC POU 的执行将返回到调用处，如果 POU 是主程序，下个循环开始。

•在平行分支中除了最右边的步，执行跳转到下一个平行分支的活动步。

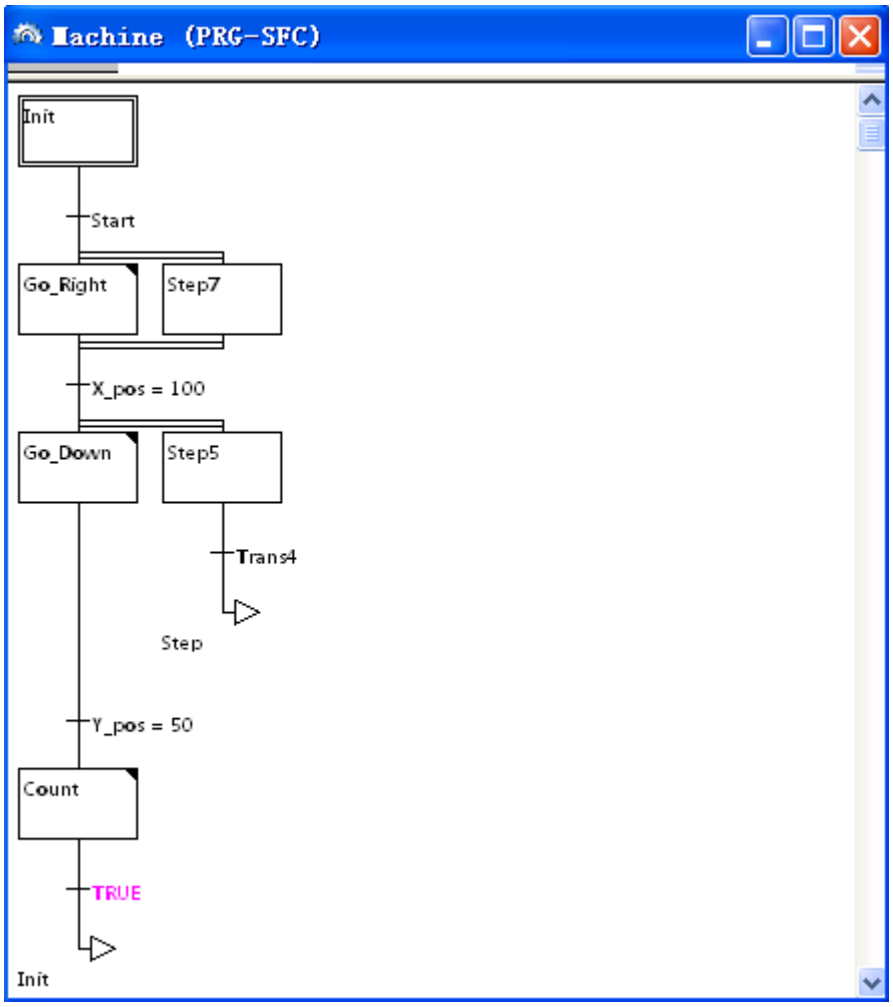
•最后的断点位于 3S 动作之内，执行跳转到 SFC 的调用处。

•最后的断点位于 IEC 动作内，执行跳转到 SFC 的调用处

•最后的断点位于输入或输出动作之内，执行跳转到下一个活动的步中。

使用“联机” “进入”，包括能进入动作中，如果输入、输出或 IEC 动作要被跳入，必须在这里设置一个断点。在动作内，相应编辑器的所有调试功能对用户都可用。

如果光标在声明编辑器中的变量上停留一会，变量的类型、地址和变量的注释将显示在工具提示中。



请注意：如果你重命名了一个步并在步是活动的时候执行了一个联机改变，那么程序将会停止在未定义状态

在序列中的元素的处理顺序:

1.首先，在序列中使用的 IEC 动作中的所有动作控制块标示符将复位（然而，IEC 动作的标示符在动作内部调用）

2.所有步按顺序测试（从上到下和从左到右）

3.对于所有的步，以下部分按序列中设定的顺序来完成：

-如果可用，逝去的时间复制到相应的步的变量中。

-如果可用，任何超时都被检测并且 SFC 错误标示符按要求提供。

对非 IEC 步，相应动作开始执行。

4.在序列中用到的 IEC 动作按字母顺序执行，通过动作列表有两种途径来完成这个动作，第一种是，所有的在当前循环中未激活的 IEC 动作开始执行。第二种途径是，在当前循环中所有激活的 IEC 动作执行。

5.转变开始进行：如果在当前循环中的步是激活的并且下面的转变条件返回 TRUE（如果最小激活时间已经消逝），下面的步是激活的。

关于动作的执行部分应该注意以下部分：

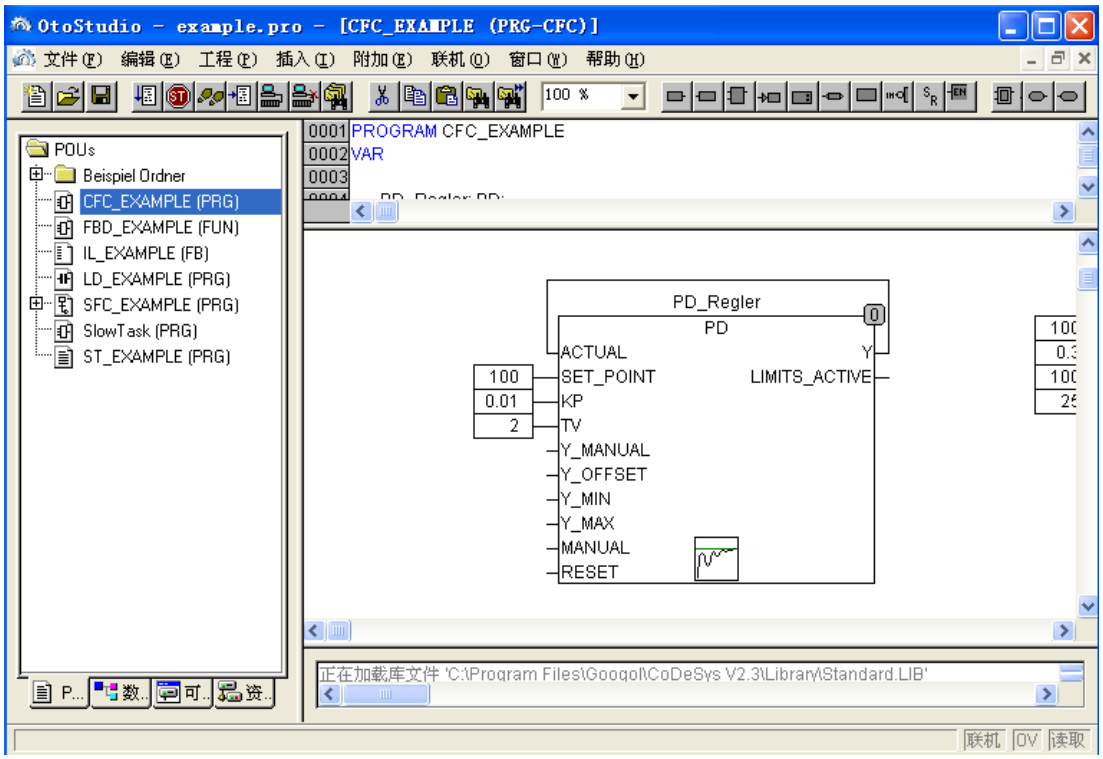
动作可以在一个循环中执行多次因为它关联了重复序列。（例如，一个 SFC 可以有两个 IEC 动作 A 和 B，它们都在 SFC 中执行，并且都可以调用 IEC 动作 C，那么 IEC 动作 A 和 B 都可以在相同的循环中激活，而且在这两个动作中 IEC 动作 C 也能激活，C 将被调用两次）。如果相同的 IEC 动作在 SFC 中不同的级别中同时使

用，由于上述处理序列的原因将导致不可预料的结果。因而，在这种情况下将回出现出错信息。它也可能出现在其它旧版本的 OtoStudio 工程创建过程中。

注意：在转换条件中监视表达式（例如，A AND B）时，显示的是整个表达式的值

6.5.4 连续功能图表编辑器

它看起来象使用 连续功能图表编辑器 CFC 产生的块：



连续功能图表编辑器中没有使用捕捉栅格，因此元素可以任意放置。连续处理列表的元素包括框、输入、输出、跳转、标签、返回和注释。这些元素的输入和输出可以通过用鼠标拖动连接来连接起来，连接线自动画出。最短的可能的连接线要考虑到现有的连接，当元素移动时连接线自动调整，如果连接线因为缺乏空间不能画出，在输入和相关的输出之间出现一个红线，这个红线只有当空间充足时才转化为连接线。

与通常的 功能模块图编辑器相反的连续功能图表的一个优点是反馈路径可以直接插入。

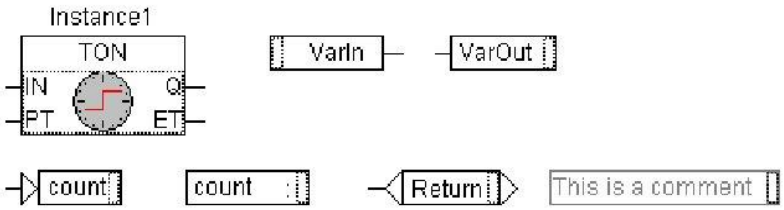
在内容菜单中可以找到最重要的命令。

CFC 的当前位置

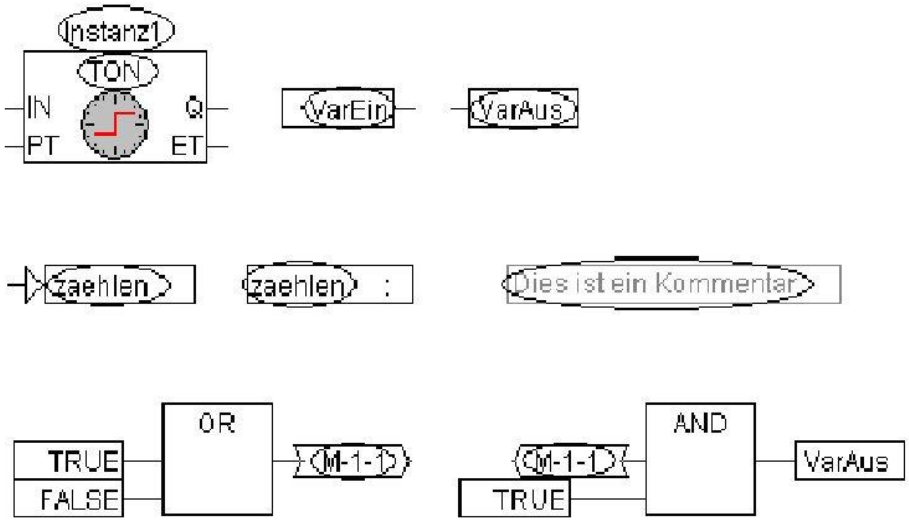
每个文本都是光标可能的位置，选中的文本渐变为蓝色并且可以被修改。

在其它的情况下当前鼠标的位置通过虚线矩形框来显示，下面是光标可能位置的例子：

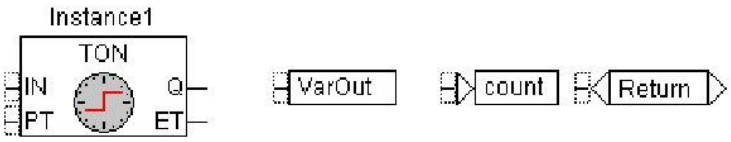
1.框、输入、输出、跳转、标签、返回和注释这些元素的中继线。



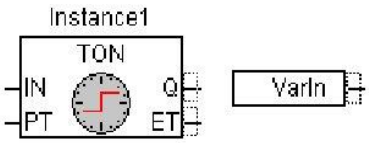
2.与连接标记的文本区域一样框、输入、输出、跳转、标签、返回和注释这些元素的文本区域。



3. 框、输入、输出、跳转、标签、返回这些元素的输入端



4. 元素框和输入的输出端：



'插入' '框'

符号 ： 

快捷方式 ： <Ctrl>+<B>

通常插入一个“AND”操作符，可以通过选中操作符并用其它其它的操作符、功能、功能模块和程序覆盖文本来修改它，输入帮助提供从支持块的列表中选出期望的块的帮助。如果新块，有输入的其它的最小数字，这些将附加上。如果新块有输入的较小最高号码，最后的输入将被删除。

'插入' '输入'

符号 ： 

快捷方式 ： <Ctrl> + <E>

这个命令用来插入一个输入，出现的文本“???”可以被选中并被变量或常量替代，输入帮助也可以在这里使用。

'插入' '输出'

符号 ： 

快捷方式 ： <Ctrl>+<A>

这个命令用来插入一个输出，可以选中出现的文本"???"并且可以用变量替代，在这里也可以使用输入帮助，与这个输出的输入相关的值分配给这个变量。

'插入' '跳转'

符号 ： 

快捷方式 ： <Ctrl>+<J>

这个命令用来插入一个跳转，可以选中出现的文本“???”并用程序要跳转的跳转标签来替代它。使用命令“插入”“标志”插入跳转标签。

'插入' '标签'

符号 ： 

快捷方式 ： <Ctrl>+<L>

这个命令用来插入一个标签，可以选中出现的文本“???”并用跳转标签来替代它，在联机模式下自动插入一个标记 POU 的末端的跳转标签。

用命令“插入”“跳转”可以插入跳转。

'插入' '返回'

符号 ： 

快捷方式 ： <Ctrl> + <R>

这个命令插入一个返回命令，注意在联机模式下带 RETURN 名字的跳转标签自动插入到第一列和编辑器最后元素的后面，在分支中，它自动跳转到执行将离开 POU 之前的地方。

'插入' '注释'

符号 ： 

快捷方式 ： <Ctrl> + <K>

这个命令用来插入注释。

用<Ctrl> + <Enter>在注释内获得一个新行。

'插入' '输入框'

快捷方式 ： <Ctrl> + <U>

这个命令用来在方框处插入一个输入。不同的操作符有不同的输入端数（例如，ADD 可以有两个或多个输入端）。

必须选中方框才能为这个操作符增加输入端的数目，每次添加一个输入端。

'插入' '输入针'

'插入' '输出针'

符号 ： 

为了编辑而打开宏时，这些命令是可用的，它们用来插入宏的输入或输出或输入和输出。通过它们的显示和有没有位置索引来区分 POU 的输入和输出端。

'附加' '取反'

符号 ： 

快捷方式 ： <Ctrl> + <N>

这个命令用来对输入、输出、跳转、返回命令取反，在连接上用一个十子交叉表示否定的符号。

元素块、输出、跳转或返回当它们被选中时，它们的输入否定。

元素块的输出或输入当它被选中时被否定（光标位置 4）。

通过再次否定可以取消原来的否定。

'附加' '置位/复位'

符号 ： 

快捷方式 ： <Ctrl> + <T>

这个命令只能用来选择输出元素的输入端。

设置的符号是 S，复位的符号是 R。

如果 VarIn1 传递 TRUE，VarOut1 设置为 TRUE，VarOut1 保持这个值，即使当 VarIn1 跳回到 FALSE。

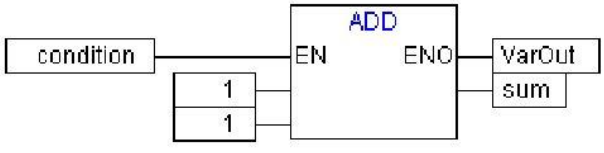
如果 VarIn2 传递 TRUE，VarOut2 设置为 FALSE。VarOut2 保持这个值，即使当 VarIn1 跳回到 FALSE。

重复激活这个命令可以使输出在设置、复位和正常状态之间转换。

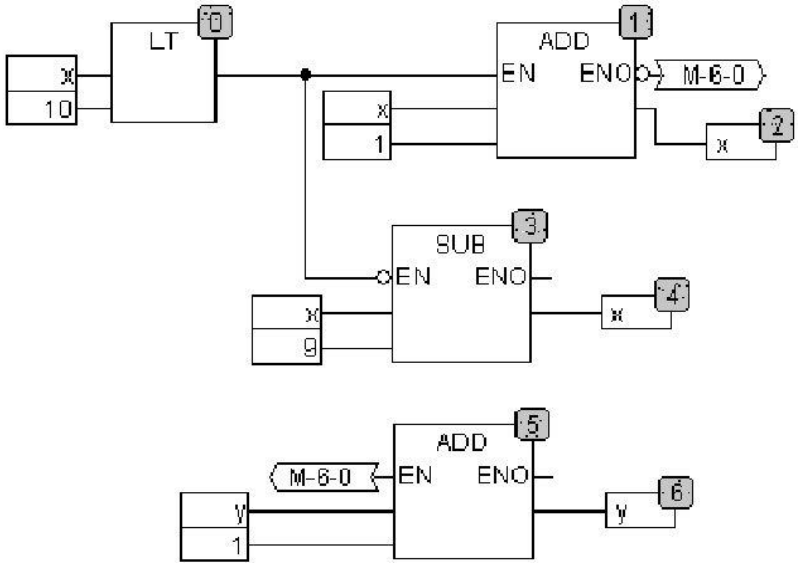
'附加' 'EN/ENO'

快捷方式 ： <Ctrl> + <0>

这个命令用来给选中的块添加一个附加的布尔型使能输入 EN 端（使能输入端）和一个附加的布尔型使能输出 ENO（使能输出端）。



在这个例子中，当布尔变量“Bedingung”（状态）为 TRUE 时，ADD 才执行，在 ADD 执行后 VarOut 设置为 TRUE。当“Bedingung”（状态）为 FALSE 并且 VarOut 保持 FALSE 值时 ADD 将不会执行。下面这个例子显示了 ENO 值是怎样为随后的块工作的：



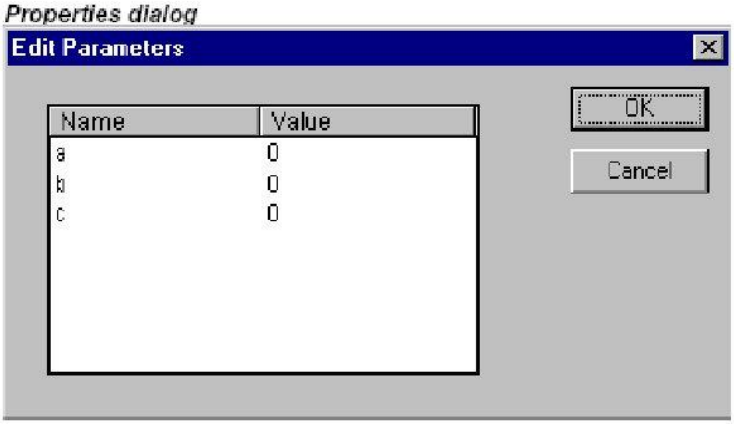
x 应该初始化为 1 和 y 初始化为 0，在块的右角上的数字表明了执行的顺序。

x 将会每次增加 1 直到值为 10，块的输出 LT（0）传递值 FALSE 并且 SUB（3）和 ADD（5）将会被执行，x 的值设置回 1y 的值每次增加 1，只要 x 的值比 10 小，LT（0）就重复执行，y 记录 x 在值 1 到 10 之间通过的次数。

'附加' '属性 '

在连续功能编辑器中不显示 功能和 功能模块中的常量输入参数(VAR_INPUT CONSTANT)，当你选中块的主体并且使用命令“附加”“属性”或者在主干上双击时“特性编辑器”对话框打开，在这里显示和修改它们的值：

Properties dialog



常量输入参数(VAR_INPUT CONSTANT)的值是可以改变的，在这里你必须在列值中标记参数值。单击鼠标或按空格键可以编辑参数值，按<Enter>键确认或者按<Escape>键取消。按'确认'存储所有的改变。

选择元件

在元素的主体上单击来选中它。

为标记多个元素，按<Shift>键并一个接一个单击需要选择的元素，或者按鼠标左键并拖动鼠标到要标记的元素上。

命令“附加”“选择所有”一次能标记所有的元素。

移动原理

一个或多个选中的元素可以通过方向键来移动就如按住<Shift>键一样，也可以通过按住鼠标左键拖动来移动元素。通过释放鼠标左键放置这些元素直到它们不覆盖其它的元素或者超出编辑器的边界。否则，标记的元素跳回到它的初始位置并出现一个警告声音。

复制原理

一个或多个选中的元素可以使用命令“编辑”“复制”来复制并且可以用命令“编辑”“粘贴”来插入。

创建连接

元素的输入端可以精确的连接到另一个元素的输出端，一个元素的输出端可以连接到其它元素的输入端。

元素 E2 的输入端和元素 E1 的输出端的连接有多种可能性。

把鼠标置于元素 E1 的输出端，单击鼠标左键，按下鼠标左键并拖动鼠标指针到 E2 的输入端，然后释放鼠标键，在用鼠标拖动的过程中产生从元素 E1 的输出端到鼠标指针的连接，放置鼠标到元素 E2 的输入端，单击鼠标左键，按下鼠标左键并拖动鼠标指针到元素 E1 的输出端并释放鼠标左键。

创建连接

在元素 E1 的输出端和元素 E2 的输入端之间的连接可以很容易的改变为元素 E1 的输出端和元素 E3 的输入端之间的连接，在 E2 的输入端上单击，按下鼠标左键，移动鼠标指针到 E3 的输入端并释放。

产出连接

删除 E1 的输出端和元素 E2 的输入端之间的连接有多种方法：

选中元素 E1 的输出端并按<Delete>键或执行命令“编辑” “删除”，如果 E1 的输出端连接到了多个输入端，那么将会同时删除多个连接。

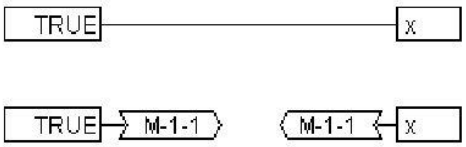
选中元素 E2 的输入端并按<Delete>键或执行命令“编辑”“删除”。

用鼠标选中元素 E2 的输入端，按下鼠标左键并拖动连接从 E2 的输入端离开，当鼠标左键在屏幕的自由区域中释放时，连接删除。

'附加' '连接标记'

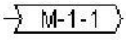
连接也可以用一个连接器（连接标记）表示来替代连接线，在这里唯一的名字连接器添加到输出和相关的输入端。

在两个元素之间已经存在的连接现在用连接器表示，连接线的输出端被标记并且菜单中的“扩展”“连接标记”被选中，下面图表显示一个连接在菜单点选择前后的连接情况。

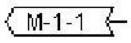


程序给出一个唯一的标准化的名字，开始于 M，连接器的名字存储为一个输出变量，它可以被更改，既可以在输入端更改也可以在输出端更改。

连接器的名字是和连接的输出端的属性相关的并且和它一起保存。

1.在输出端编辑连接器： 

如果在连接器中的文本被替代，在输入端的所有相关的连接器都会采用新的连接器名字。一种情况不会采用，选择的名字已经属于另外的连接器，因为违背了连接器名字的唯一性。

2.在输入端编辑连接器： 

如果连接器中的文本被替换，在其它的 POU 中相应的连接器也会被替换。在标记连接的输出端（光标位置 4）并且选择菜单点“扩展” “连接标记”后，连接器表示中的连接可以转换为正常的连接。

改变连接

元素 E1 的输出和元素 E2 的输入之间的连接可以被方便的改为元素 E1 的输出和元素 E3 的输入。把鼠标选中 E2 的输入，然后按住左键移动鼠标到 E3 的输出位置松开左键。

删除连接

有很多方法来移除元素 E1 的输出和元素 E2 的输入：

选择元素 E1 的输出然后按 <Delete> 键或者执行命令'编辑' '删除'。如果 E1 输出连接多个输入，删除情况相同。

选择元素 E2 的输入然后按<Delete> 键或者执命令'编辑' '删除'。

用鼠标选择 E2 的输入，按住鼠标左键从 E2 处拖出，在空白处松开鼠标 E2 就被移除

插入 输入/输出 "不工作状态"

如果选中一个元素的输入或输出端，相应的输入或输出可以直接插入，编辑器区域中充满了在键盘上输入的字符串。

执行次序

元素块、输出、跳转、返回和标签每个都具有一个编号表明了它们执行的顺序，在这个连续的顺序中，每个元素在运行时被计算。

当元素的编号根据拓扑顺序自动给出（从左到右和从上到下），如果顺序已经改变了，新元素将接收它的拓扑继承者的编号，并且所有较高的编号增加 1，当它移动时元素的编号保持不变。

顺序影响结果，在一定情况下必须改变顺序。

如果显示了顺序，相应连续执行编号在元素的右上角显示。

参照:

'附加' '次序' '显示'

'附加' '次序' '次序拓扑'

'附加' '次序' '向上移动一格'

'附加' '次序' '向下移动一个'

'附加' '次序' '移动到首端'

'附加' '次序' '移动到末端'

'附加' '次序' '依照数据溢出的所有次序'

'附加' '显示排序'

这个命令用来切换执行顺序显示的开和关闭。默认的设置是显示（在菜单点的附近用一个 ü 识别）。

在块、输出、跳转、返回和标签这些元素执行编号的相关顺序出现在元素的右上角。

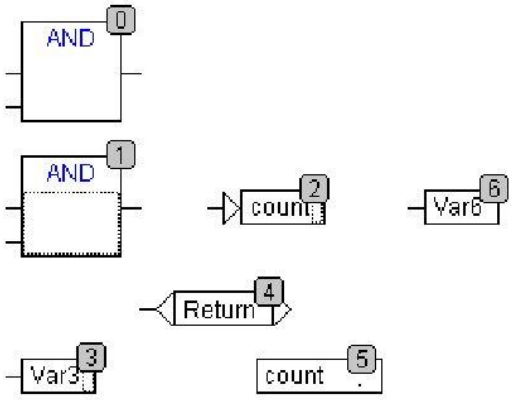
'附加' '排序' '拓扑排序'

当执行从左到右并且从上到下执行时，元素按拓扑顺序处理。编号从左到右增加并且元素从上到下拓扑安排。连接是不相关的，只是元素的位置是重要的。

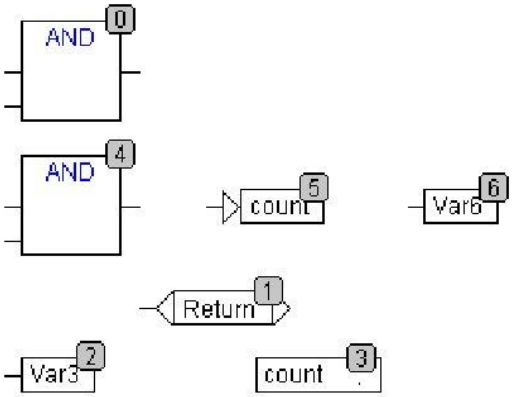
当使用命令“附加”“排序” “拓扑排序”时，所有选中的元素以拓扑结构安排。在选择中的所有元素

在这个处理过程中从连续处理列表中去掉。元素然后各自地从右下到左上输入到保留的连续处理过程列表中。

每个标记的元素在它的拓扑继承者之前输入到列表中，例如，当编辑器中的所有元素都按拓扑顺序进行了排序，在拓扑顺序中在元素之前插入的将在元素之后执行。可以用下面这个例子阐明。



编号为 1、2、3 的元素被选中。如果命令“Order topologically”被选中元素将首先从连续处理列表中除去。Var3，跳转和 AND 操作符然后一个接一个插入，Var3 放置在标签之前并且接收编号 2，跳转然后被排序并首先接收编号 4 但是随后在 AND 插入之后变为 5，新的执行顺序出现：



当引入一个新产生的块时，它将在顺序处理列表中默认放置它的拓扑继承者的附近。

'附加' '排序' '向上移动一步'

除了在连续处理列表中开始处的元素外，用这个命令选中的所有元素将在处理列表中向前移动一个位置编号。

'附加' '排序' '向下移动一步'

除了在连续处理列表中末端处的元素外，用这个命令选中的所有元素将在处理列表中向后移动一个位置编号。

'附加' '排序' '移动到首前端'

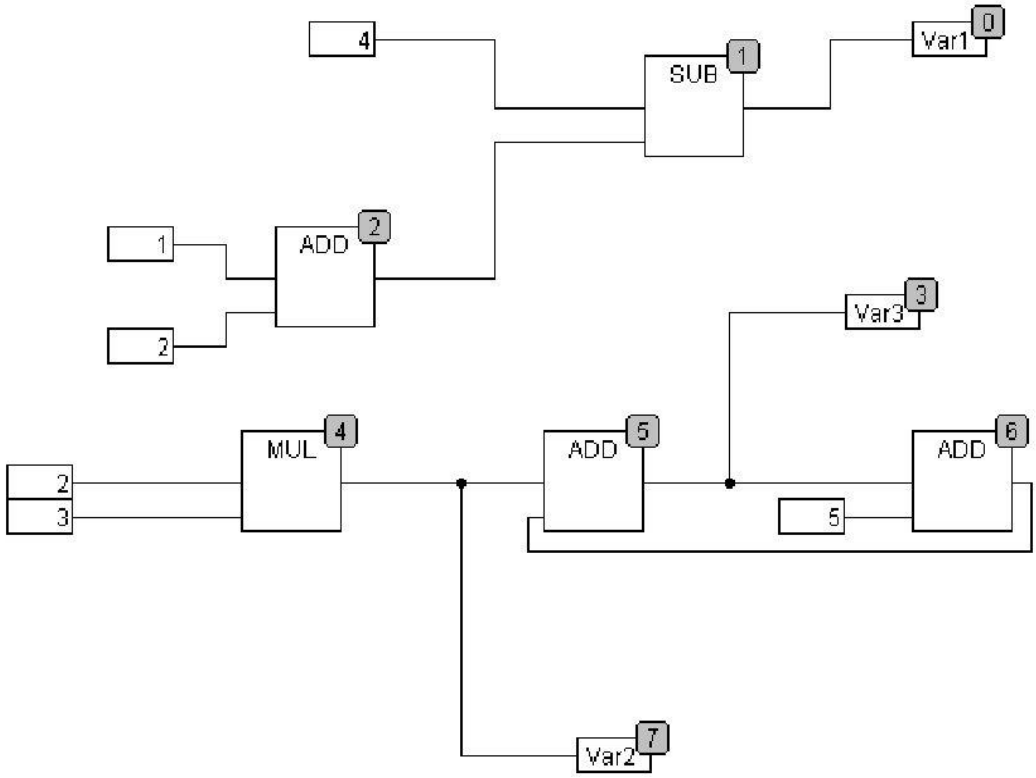
使用这个命令所有选中的元素将会移动到连续处理列表的附近，在选中元素的组内顺序保持不变，未选择元素的组内的顺序也保持相同。

'附加' '排序' '移动到末端'

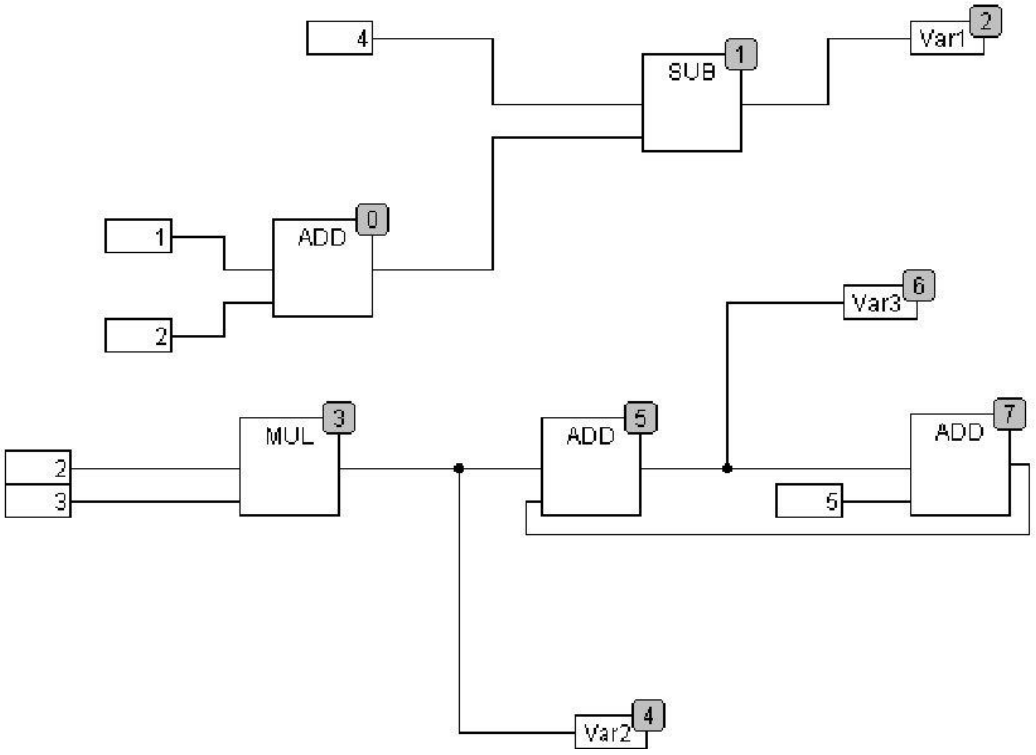
使用这个命令所有选中元素将会移动到连续处理列表的末端，选中元素的组内顺序保持不变。未选中元素的组内的顺序也保持不变。

'附加' '排序' '依照数据流排序'

这个命令影响所有的元素。执行顺序由元素的数据流向决定而不是它们的位置决定。下面的图表显示已经按拓扑结构排列的元素。



下面是命令选择后的元素排列：



当选中这个命令后，元素将首先按拓扑顺序排列。创建一个新的处理列表。基于已经知

道的输入值，计算机将计算哪个未编号的元素下一步处理。在上面“network”块 AND，例如，当在它的输入端（1 和 2）的值都已经确定下迅速处理。在块 ADD 的结果处理后，块 SUB 才执行。反馈路径最后插入。

'附加' '创建宏'

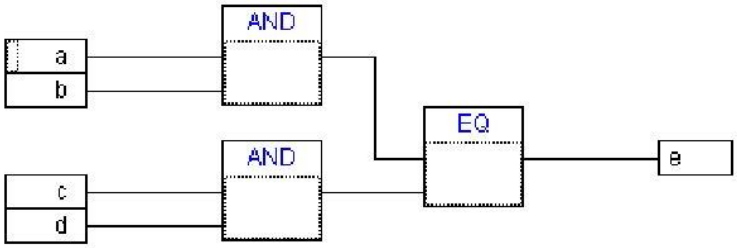
符号： 

用这个命令，同时选中的多个 POU 可以组合到一个块中，这个块可以称为一个宏。宏只能通过复制/粘贴来复制，每个复制的宏成为一个名字可以独立选择的单独的宏。宏不能引用。被宏的创建删除的所有连接在宏上产生输入或输出针脚。输入端的连接产生输入针脚。默认的名字出现在针的旁边形式为 In<n>。输出端的连接，出现 Out<n>。影响有连接标记先于宏的创建的连接，保留连接标记在宏的针脚上。

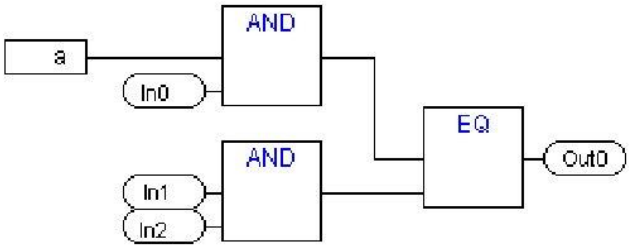
首先，宏有默认的名字“MACRO”，可以在宏使用的名字区域中来改变它。如果宏被编辑，宏的名字跟在 POU 名字后面将显示在编辑窗口的标题栏中。

例如：

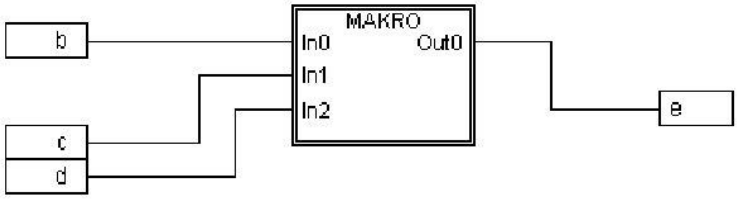
选择



宏：



在编辑器中：



'附加' '编辑宏'

符号： 

通过这个命令，或在 宏的主干上双击，在相关的 POU 中的编辑窗口中宏被打开，宏的名字跟在 POU 名字的后面显示在标题栏中。

为宏的输入和输出产生的针脚方框在创建过程中可以作为普通 POU 的输入和输出端

处理。它们也能移动，删除，添加等等，它们的不同之处在于它们怎样显示和有没有位置索引。对于添加你可以使用按钮（输入）或按钮（输出），这些按钮在菜单栏上是可用的，针框有圆角，在针框中的文本匹配在宏中显示的针的名字。

在宏方框中的针脚的顺序遵循宏中的元素的执行顺序，低顺序索引在高顺序的前面，高针脚在低针脚的前面。

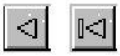
在宏内的处理顺序是封闭的，换句话说宏在主 POU 中的宏的位置处作为一个块来处理，管理执行顺序的命令只在宏能起作用。

'附加' '扩展宏'

使用这个命令，选中的 宏重新展开并且它里面包含的元素在宏的位置处插入到 POU 中。与宏的针脚的连接重新显示就象连接到元素的输入或输出端一样。如果在宏方框的位置处因为缺乏空间而不能展开宏，宏将放置到右下直到空间可用。

注意：如果工程在版本 2.0 下保存，宏将都同样地展开。所有的宏在转化为其它的语言前也都展开。

'附加' '后退一级宏', '附加' '后退所有宏'

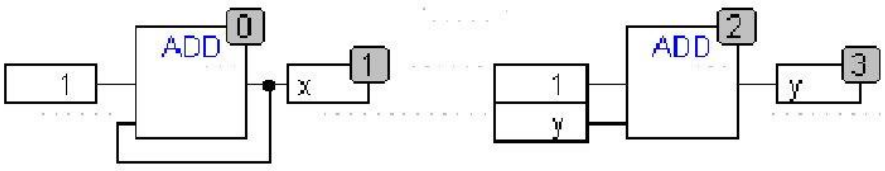
符号： 

只要宏打开，这些命令在工具栏中也可用。如果 宏彼此互相嵌入，可以转换到下一个高的或最高的显示级别。

在 CFC 中的反馈路径

反馈路径只能直接显示在连续功能图编辑器中，不能在通常的功能模块图编辑器中显示。应该注意块的输出经常携带一个内部中间变量。中间变量结果的数据类型，对于操作数，从输入的最大数据类型。常量的数据类型从最小可能数据类型，常量'1'采用数据类型 SINT。如果有一个带加号的反馈和常量'1'被执行，第一个输入给出数据类型 SINT 和因为反馈第二个是未定义的。因而内部中间变量只分配给输出变量。

下面的图表显示了一个带反馈的加法和带变量的加法。变量 x 和 y 在这里是 INT 型。



两种加法的区别是：

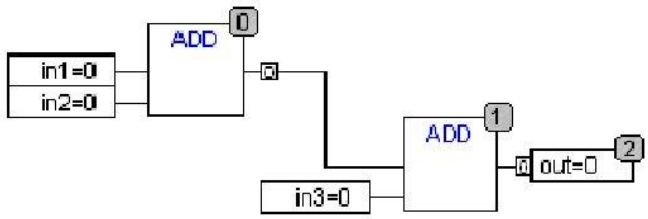
变量 y 可以初始化为一个不等于零的值但这不是左边加法的中间变量的情况。左边加法的中间变量有数据类型 SINT，在右边的中间变量有数据类型 INT。在调用 129th 后，变量 x 和 y 有不同的值，变量 x，尽管是 INT 型，包含值 127 因为中间变量已经溢出。另一方面，变量 y 包含值 129。

联机模式下的 CFC

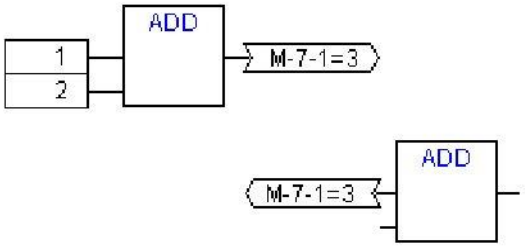
监视：

输入和输出的值显示在输入或输出方框中。常量不被监视。对于非布尔型变量，方框展开来适应值的显示。对于布尔型连接如果值为 TRUE，变量名和连接显示为蓝色，否则它们保持黑色。

在 TRUE 状态时，内部布尔型连接在联机模式下也显示为蓝色。内部的非布尔型连接的值在一个带圆角的小框中在连接的输出针脚上显示。



宏的针脚和输入或输出方框的监视一样：



带连接标记的非布尔型连接在连接标记内显示它们的值。对于布尔型连接，如果线中传递值 TRUE，线和标记名显示为蓝色，否则显示为黑色。

流程控制：

当流程控制为开时，横越的连接标记为工程选项中选择的颜色。

断点：

断点在所有的有处理顺序索引的元素上设置。程序的处理各自元素的执行之前停止并且

返回标签：

在联机模式下，名字为“RETURN”的跳转标签在第一列中和编辑器中最后元素之后自动产生，这个标签标记 POU 的末端，当执行前的分级离开 POU 时跳转到 POU ，在宏中没有 RETURN 标记插入。

分级：

当使用 '跳过'时，将通常跳转到下一个较高级别索引的元素处，如果当前的元素是一个宏或 POU，那么它的执行分支当 '跳入'有效时，如果一个“跳过”从这里执行，将跳转到宏的顺序索引跟随的元素处。

切换到 POU

快捷方式 ：<Alt>+<Enter>

用这个命令选中的 POU 加载到它的编辑器中，如果光标放置在文本编辑器中的 POU 名字上或如果 POU 框在图形编辑器中被选中，在内容菜单(<F2>)或在“附加”菜单中可用到这个命令。如果你从库中处理一个 POU，那么库文件管理器被调用，相应的 POU 显示。

第七章 资源

你需要用资源来配置和组织你的工程文件和追踪变量的值：

- 工程文件或网络中使用的**全局变量**。
- 添加库文件到工程文件中的**库文件管理器**
- 记录在线期间工作的**日志文件**
- 在工程中为报警处理进行**报警配置**
- 配置可编程控制器的 **PLC 配置**
- 通过任务来引导创建程序的**任务配置**
- 显示变量值和添加默认变量值的**监控和配方管理器**
- 选择 目标设置和必要时的确定的**目标系统设置**
- 作为工程选项的**工作空间**

根据在 OtoStudio 中作出的目标系统和目标设置，在你的工程中也要用的到下列资源：

- 用于变量图形显示的**采样追踪**
- 用于在同一个网络中与其它控制器交换数据的**变量管理器**
- 作为控制监视的 **PLC 浏览器**
- 用于在 OtoStudio 内部调用它外部的工具程序的**工具箱**

7.1 全局变量、变量配置、文件框架

'在全局变量' 中的对象

在对象管理器的资源登记卡中，在全局变量文件夹内有二类对象（对象默认值的名字在圆括号里）。

•全局变量列表

•变量配置

在这些对象中定义的所有变量在整个工程中被公认。

如果全局变量文件夹未打开（文件夹前面有个加号），可以在该行双击或<回车>来打开它。

选择相应的对象，用'对象添加'指令先打开以前定义的全局变量的窗口。这个编辑器跟声明编辑器有同样的工作方式。

多个变量列表

必须在不同的对象中定义 全局变量，全局网络变量(VAR_GLOBAL), 全局网络变量(VAR_GLOBAL，目标平台专用)和 变量配置 (VAR_CONFIG)。

如果你已经声明了许多全局变量并且想更好地构造全局变量列表，那么你可以创建更多的变量列表。

在对象管理器中，选 Global Variables 文件夹或一个已有的全局变量的对象。然后执行'工程' '对象添加'指令。给出现在对话框中的对象一个相应的名字。有了这个名字，将会用关键字 VAR_GLOBAL 来创建一个新的对象。如果你希望对某个对象进行变量配置，你可以把相应的关键字变成 VAR_CONFIG。

7.1.1 全局变量

VAR_GLOBAL

贯穿整个工程众所周知的“标准的”变量，常数或剩余的变量可以被声明为全局变量，也可以是其他的网络用户交换数据的网络变量。

注意：在一个工程中你可以定义一个跟全局变量重名的局部变量。在这种情况下，在 POU 的范围内使用局部变量。

不允许两个全局变量有同样的名字。例如，如果你在 PLC 配置中定义了一个变量“var1”，同时这个变量也存在于全局变量列表中，那么你将得到编译错误。

参考:

- 网络变量
- 创建全局变量列表
- 编辑全局变量和网络变量列表
- 编辑剩余的全局变量列表
- 全局常量

网络变量

注意：使用网络变量必须得到目标系统的支持，必须在目标平台设定（网络功能类）中激活它。

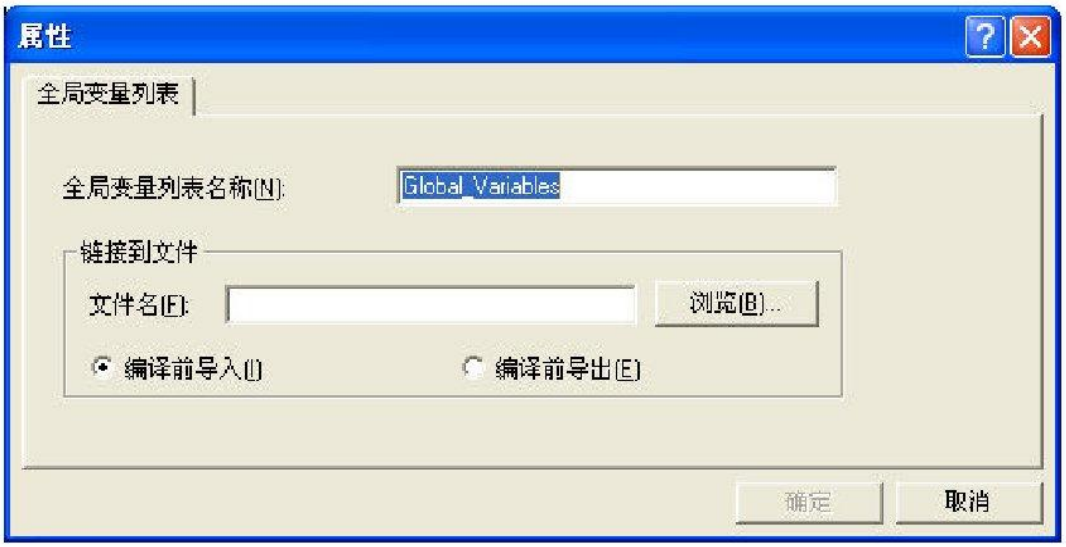
通过自动的数据交换（与通过参数管理器的非自动的数据交换比较），在 OtoStudio 相互兼容的控制器网络内的几个控制器系统上，可以更新网络变量的值。这不需要控制器特定的功能，但是，网络用户须为工程中的网络变量使用完全一样的声明列表和相称的转移配置。为了使前项操作成为可能，建议在创建列表时对每个控制器应用不要用手工输入变量的声明，而是用一个独立的文件来加载（见'创建一个全局变量列表'）。

创建全局变量列表

为了创建一个全局变量列表，打开对象管理器中的'资源' ，选择条目'全局变量'，或者选一个已经存在的列表。接着，选择指令 '工程' '对象' '添加'来打开全局变量列表对话框。

如果在对象管理器中标明了一个存在的全局变量列表，那么也可以用指令 '工程' '对象' ' 配置'打开它的对话框。这个列表的配置显示如下：

创建一个新的全局变量列表的对话框



全局变量列表名称：插入列表名称

链接文件：

文件名：如果有一个输出文件(*.exp)或者是一个 DCF 文件，这些文件包含了想要的变量，那么可以对这个文件建立链接。为了做到这些，要在文件名域中插入文件的路径。点击浏览按钮将得到标准的对话框'选择文本文件'。当读入它们的时候，DCF 文件被转换为 ICE 格式。

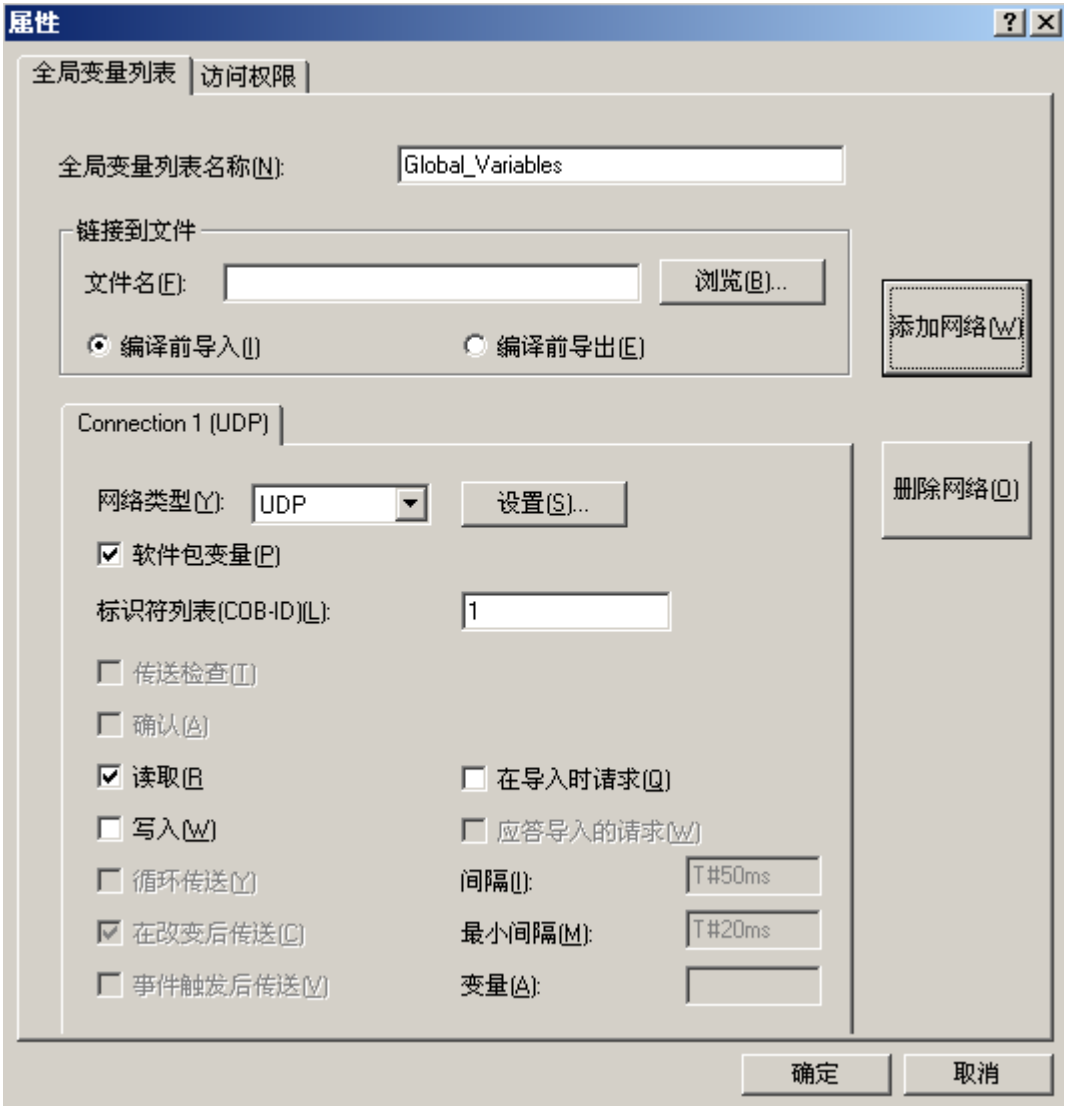
如果希望在每一个工程编译之前从外部文件读取变量列表，那么就激活编译前输入选项。如果希望在每一个工程编译之前把变量列表写到外部文件，那么就激活编译前输出选项。

如果点击 OK 来关闭'全局变量列表'对话框，那么就创建了这个新对象。全局变量列表在对象管理器中通来标识。你可以用'工程' '对象' '道具'指令重新打开'全局变量列表'配置对话框。

网络变量的配置

如果在目标平台设定中激活选项'支持网络变量'，那么就可采用按钮<添加网络>。点击这个按钮，对话框将会加长如上所示。如果没有激活这个选项，那么按钮<添加网络>不起

作用。



连接 <n> (<网络类型>):在对话框的下部，可以建立多达四个网络连接的配置，一个配置设置为网络内的特定变量列表定义数据交换的参数。为了在网络内按要求交换数据，必使相同变量列表有一致的配置以便跟其他的网络用户相匹配。

如果尚未配置，点击’添加网络’按钮，就可得到一张在 UDP 网络的情况下，题为'连接 1 (UDP)'的表单。每按一次'添加网络'按钮，便可得到一张’连接 n’表单（n 是表单序号）。

网络类型：从列表中选择希望的网络类型。以上列表是由目标系统条目定义的。例如，"UDP" 是"UDP"传输系统的缩写。

设置：这个按钮用下列配置参数打开<网络类型>的设置对话框。

UDP:

使用标准： 如果点击这个按钮，那么端口(port)1202 定义为和其他的网络用户进行数

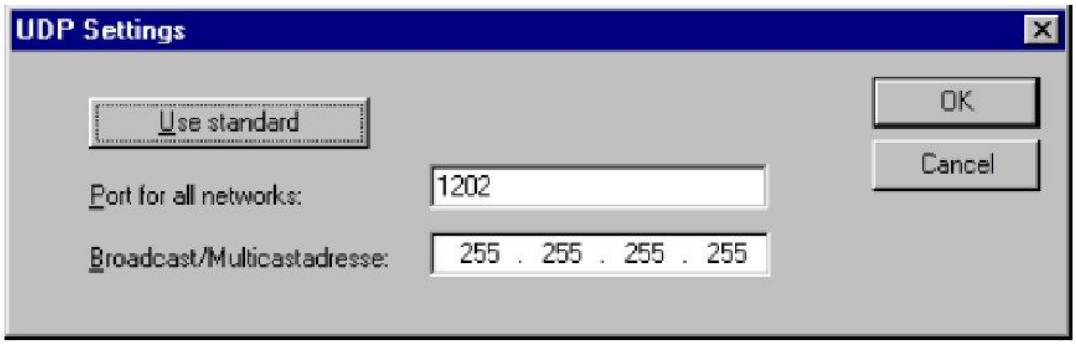
据交换的端口，广播/多路传输地址被设置到"255 . 255 . 255 . 255" 上，这意味着，数据交换将在所有的网络用户之间进行。

端口：写入要求的端口号（按照标准）。确保网络里的其他的节点定义相同的端口！如果在工程中定义了一个以上的 UDP 连接，那么根据在这里的输入将自动修改所有的配置集中的端口号。

广播/多路传输地址:如果要用标准设置，在这里写入代表子网的地址范围的地址，（例如，如果你想与 IP 地址 197 . 200 . 100 . x 的所有的节点通讯，那么其他地址应为”197.200.100.255”）。

对于 Win32 系统，广播/多路传送地址必须和 PC 配置的 TCP/IP 的子网掩码相匹配。

‘UDP 设置’对话框



变量包：以网络许可的大小大包传输（报文），需要组装变量。如该选项无效，那么就为每个变量打包。

变量电报号码: 第一个信息包的标识号，默认值为 1，后续信息包以升序编号。

包含检验: 对每个被发送的信息包都要加上校验和接受检查校验。以便确认发送器和接收器的变量定义是同一个。不会接收校验和不一致的信息包，并将给出否定应答。

使用公认传输:接收器将应答每一个信息。发送器在一个周期内不能得到至少一个应答，马上就产生错误信息。

读取：读取列表中的变量；如果这个选项无效，通过网络发送的变量将被忽略。

启动时的请求：如果本地节点是个“读”节点（激活选项“读取”），那么一旦重新启动本地节点，就马上向所有的“写”节点请求实际的变量值然后“写”节点发送这些值，它不同于通常触发通信的其它条件（时间，事件）前提条件：在写节点配置中，必须激活选项“回复启动请求”!（参看下述内容）。

写入：写变量，应用下列选项：

回复启动请求：如果本地节点是“写”节点（“写”选项已激活）那么要回答一个“读”

节点在启动时的每一个请求（选项导入请求，见上述内容）发送的读入节点要求得到答复，那意味着，此时，即使任何一个其他定义的传输引起物（时间，或者事件）不能被传送，实际变量的值也将被传送。

全局传送：在一个间隔后的指定的间隔时间内写变量。（时间表示法 T#70ms）。

变换传送:只有当变量的值变化时才写变量。在最小域内可以设置在两次传输之间的最小时间间隔。

事件传送：一旦插入在可执行变量编程 TRUE 之后，马上就写列表中变量。

在对象管理器中，全局网络变量列表通过标志来标识。

注意：如果一个网络变量用于多个任务，对传送的时间有下列要求：当访问每个任务时，要测试决定哪个参数要用于变量值的传输（在'全局变量列表'对话框里配置）。变量值的传送与否依赖于规定的时间间隔是否已过去。在每个传送开始时，变量的时间间隔计数器将复位到零。

发送总是由起作用的控制器的运行系统承担。于是不必为数据交换提供专门的控制功能。

编辑全局变量和网络变量列表

全局变量的编辑器与声明编辑器类似。但是，注意，你不能在这个编辑器中编辑连接外部变量列表映象的列表！外部变量列表仅仅能在外部被编辑，并且，他们将在每个次打开和编译工程时被读取。

语句：

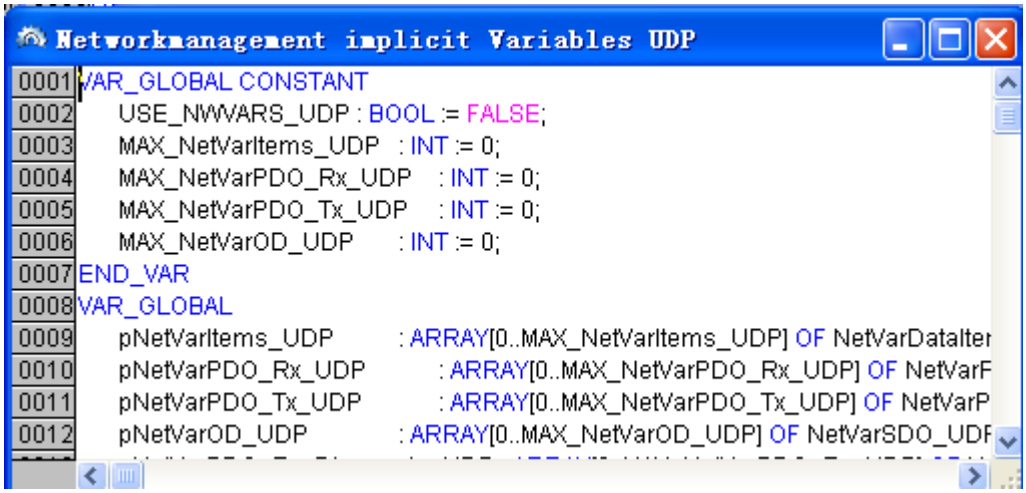
VAR_GLOBAL

(* Variables declarations *)

END_VAR

如果目标系统允许，可以只使用网络变量。也可以在这语句中定义网络变量。

例子：用链接一个输出文件*.exp 创建的一个网络变量列表，命名为 NETWORKVARIABLES_UDP 用加载一个输出文件*.exp 创建的一个网络变量列表，命名为 Network_Vars_UDP 的网络变量列表的例子：



编辑剩余的全局变量列表

如果运行系统支持剩余全局变量列表，那么就可以处理剩余变量。剩余全局变量有两种类型：

保存变量：在 OtoStudio 中，运行时间系统(off/on)或者'联机' '复位'不受控制的停机后，保存变量保持不变。运行系统(停止，开始)或者'联机' '冷启动'或者下载受控制的停机后，变量保持不变。剩余的变量被赋给关键字 RETAIN。

剩余变量的语句定义。

语句：

VAR_GLOBAL RETAIN

(* Variables declarations *)

END_VAR

VAR_GLOBAL PERSISTENT

(* Variables declarations *)

END_VAR

Network variables (target specific) are also defined using this syntax.

用这个语句也可以定义 网络变量（指定目标）。

全局常量

全局常量的关键字是 CONSTANT。

语句:

VAR_GLOBAL CONSTANT

(* Variables declarations *)

END_VAR

7.1.2 变量配置

VAR_CONFIG

如果把变量定义放在关键字 VAR 和 END_VAR 之间，在功能块中，为不完全定义的输入输出指定地址是可能的。不完全定义的地址用星号识别。

例如:

FUNCTION_BLOCK locio

VAR

loci AT %I*: BOOL := TRUE;

loco AT %Q*: BOOL;

END_VAR

这里定义了两个局部 I/O 变量，一个是局部输入 (%I*)另一个是局部输出 (%Q*)。

如果你想在对象管理器资源登录卡中为变量配置设定局部 I/Os，通常可以应用对象 Variable_Configuration。对象可以重新命名，并且，为变量配置可以创建其它对象。

变量配置编辑器就象声明编辑器那样工作。

局部 I/O 配置变量必须位于关键字 VAR_CONFIG 和 END_VAR 之间。

这样变量的名字由完整的实例路径构成，在路径中用句点把各个 POUs 和实例名互相分开。在功能块中，声明必须包括一个这样的地址，它的类（输入/输出）对应于不完全指定地址(%I*, %Q*)。数据类型也必须和功能块中的声明一致。

因为实例不存在而实例路径无效的配置变量被指示出错。另一方面，如果实例变量无配置也要报告出错。为了收到全部要配置的变量的列表，可以使用'插入'菜单里的子菜单 "所有实例路径"。

变量配置的例子

假设在程序中给出功能块有下列定义：

PROGRAM PLC_PRG

VAR

Hugo: locio;

Otto: locio;

END_VAR

那么正确的变量配置如下：

VAR_CONFIG

PLC_PRG. Hugo.loci AT %IX1.0 : BOOL;

PLC_PRG. Hugo.loco AT %QX0.0 : BOOL;

PLC_PRG. Otto.loci AT %IX1.0 : BOOL;

PLC_PRG.Otto.loco AT %QX0.3 : BOOL;

END_VAR

'插入' '所有实例路径'

用这个指令，可以产生 VAR_CONFIG END_VAR 块，这个块包含了该工程中的全部有效的实例路径。为了包含已经存在的地址，不需要再入现有的声明。如果工程已被编译('工程' '重建所有')，可以在变量配置的窗口中找到子菜单。

7.2 报警配置

报警配置综述

集成在 OtoStudio 中的报警系统可以检测关键的过程状态，记录下他们，并且借助可视元件的帮助，用户可以看见他们。在 OtoStudio 中或者 PLC 中可以做报警处理。对于在 PLC 上的报警处理，请看目标平台可视化（'Visualization'）设置类目。在资源中，可以利用条目'报警配置'来配置报警系统。

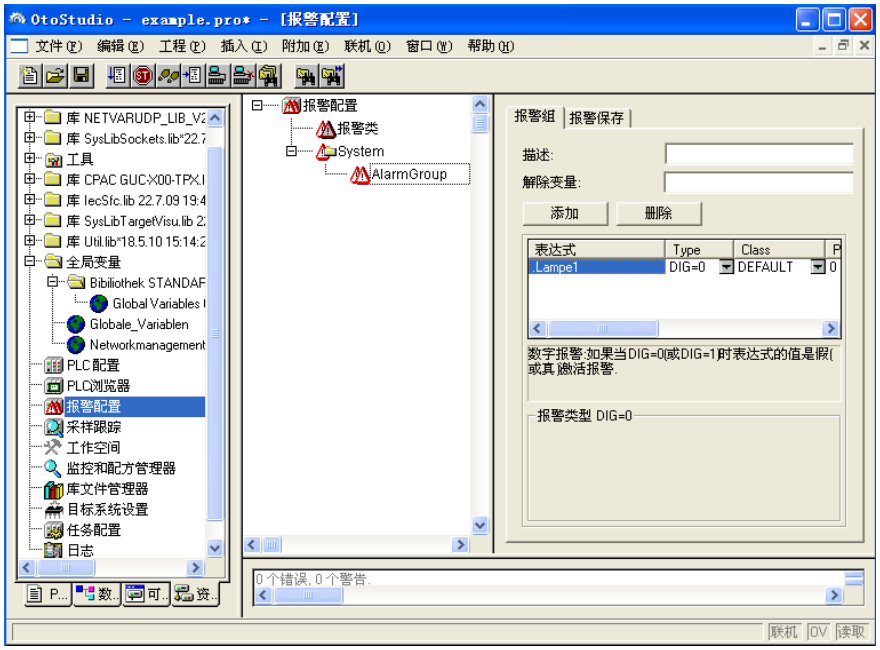
在这里定义报警类（Alarm classes）和报警组（Alarm groups）。报警类用于对警报的分类，这表示，它把某参数分配给报警。报警组用于具体配置一个或几个报警（对它们分配某个类和更多的参数）。类对于构造有效的报警是很有用的。在配置树中，用户通过在报头‘'System'下边插入适当的条目来定义不同的报警组。

在 OtoStudio 可视化中，为了报警的可视化，可以用报警表”Alarm table”。使用报警表，用户可以监视和应答警报。

如果报警事件（Alarm Events）的历史记录，要写到日志文件上，那么必须定义此文件，并且对每个警报组必须定义 存储特性。

当你在资源表中打开报警配置（'Alarm configuration'）的时候，打开带有双分窗口的对

话框'Alarm configuration'，它相关的操作模式跟 PLC 配置或任务配置相似。在左边窗口显示配置树，在右边窗口打开相应的配置对话框。



通过用鼠标单击条目 '报警配置'前的加号，打开当前可以利用的配置树。如果你打算创建新配置，这树将仅仅显示条目 '报警分类'和 '系统'。

参考:

- 警报的一般信息，术语
- 报警类
- 报警组
- 存储报警
- '附加' '设置' '文本框架'
- 警报的一般信息，术语

在 OtoStudio 里，警报系统的使用遵守下列有关报警的通用描述和定义：

•报警：通常报警被当作特殊的状态（表达值）。

•优先级：报警优先级，也称为“严重性”，描写报警状态的重要程度。最高的优先级是“0”，最低的有效的优先级是“255”。

•报警状态： 为报警控制配置的表达式/变量能有下列状态：NORM（没有报警），INTO（警报刚来到），ACK （警报已经来了，用户已经应答了），OUTOF（报警状态已经终止了，警报“已经过去”，但是还没被应答！）

•子状态：报警状况有极限值(Lo, Hi)和超极限值（LoLo，HiHi）。 例子：一个表达式的

值上升，首先通过 Hi-limit，这样引起 Hi-alarm 的到来。在警报得到用户应答前，如果值继续上升并且超 HiHi-limit， 那么 Hi-alarm 将自动被应答，并且，只有 HiHi-alarm 留在报警列表中（那是应用警报管理器的内部列表）。在这种情况下，Histate 被称为子状态。

报警应答：报警的主要的目的是把报警的状态通知用户。在这样做的过程中，必须确认用户已经注意到这个信息（参看在报警类配置中分配给报警的可能的动作）。为了把报警从警报列表中除掉，用户必须应答报警。

•报警事件： 警报事件一定不要和警报状况混淆。警报状况可以在更长的时间内有效，报警事件仅描述瞬时出现的变化，例如，从正常状态变为报警状态。在 OtoStudio 中事件的三种类型的报警配置和相应的报警状态用相同的名字(INTO, ACK, OUTOF)。

在 OtoStudio 中支持下列特征：

•对单一报警和报警组撤销报警的产生

•通过定义警报组和优先级来选择要显示的报警

•存储在警报表中所有警报事件

•在 OtoStudio 可视化中的报警表（'Alarm table'）元素的可视化

参考:

警报的一般信息，术语

报警类

报警组

存储报警

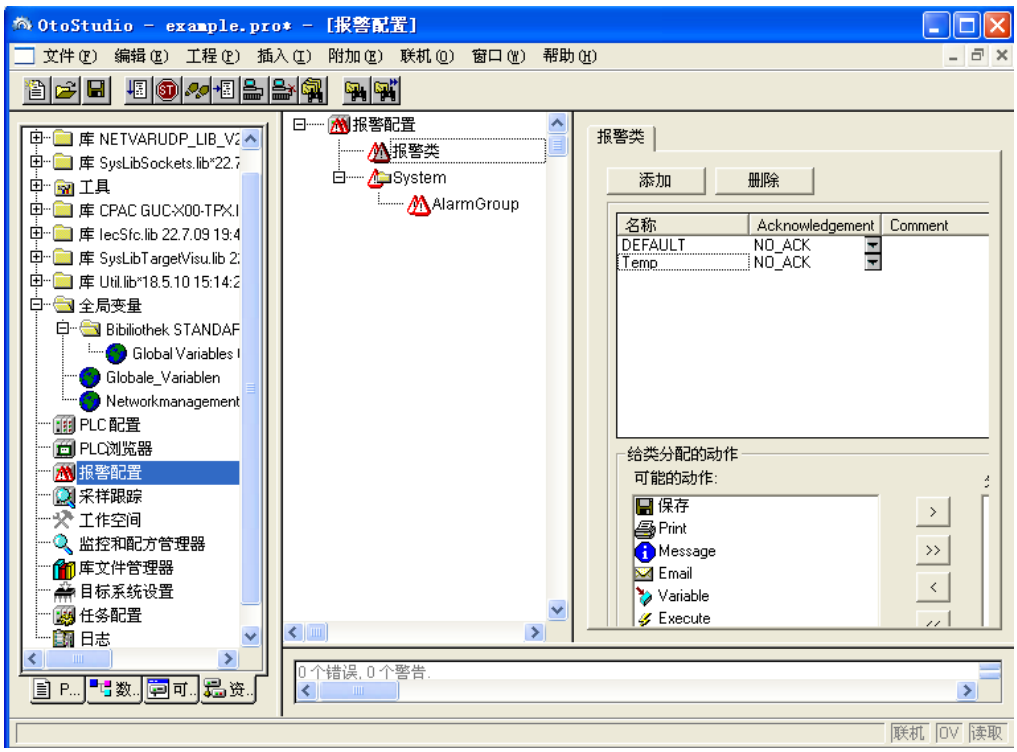
'扩展' '设置' '文本框架'

报警类

报警类用于某警报准则的一般描述，例如如何处理应答（用户确认警报）。一旦某种 报警状态被察觉之后，应该马上自动决定执行哪些动作，报警表的可视化要使用什么样的颜色和位图。在 报警配置全局性地定义报警类，在配置 报警组时，报警类用作基础配置。

报警类的配置：

在警报配置树中选择条目'报警类'。打开配置对话框'报警类'：



点击按钮 Add 创建新的警报类。随即在上边的窗口插入一行，原来在'Acknowledgement' 栏里只有条目“NOACK” （不应答）。在 Name 栏相应的域中，为警报类定义一个名字（通过鼠标点击域来打开编辑框），如果必要，在 Acknowledgement 栏中修改应答类型。

可采用下列应答：

NO_ACK：不要求用户应答报警

ACK_INTO： （状态"INTO"，报警发生）必须由用户确认出现的报警状况。

ACK_OUT OF：（状态“OUTOF”，报警终止）必须由用户确认终止报警。

ACK_ALL：必须由用户 确认报警的出现和终止。

另外，你可以输入注释（Comment）。追加的报警类条目将被添加在列表的末尾。

用 Delete 按钮从列表中删除当前选择的条目。

对类分配动作:

在上边窗口定义的每个报警类可以分配给一列动作，报警事件发生后，应该马上执行这些动作。

在可能的动作（Possible actions）列表中选择一个动作，点击按钮">"把它分配给动作（Assigned actions）域。这个域将最终包括分配给此警报类的动作。通过按钮“>>”你可以一下子加进全部动作。通过按钮"<" 和 "<<"可以从已经完成的现有的选择中除去一个或者所有的动作。对于在'Assigned actions' 列表中标明的动作，通过"..."可以打开相应的对话框，

以便定义想要的 e-mail 设置，打印设置，过程变量可执行程序及信息正文。

支持下列动作类型(可能的动作)：

动作	描述	在相应的对话框中要做的设置
存储	报警事件将保存在内部以便分发到 日志文件中。请注意：在这种情况下必须在 警报组的配置中定义记录文件	在报警存储对话框内报警组定义中进行设置
打印	把信息正文发送给打印机	Printer：选择一个在本地系统上定义的打印机；Outputtext ：应该打印的信息正文（见下述内容）
消息	在当前的报警 可视化中，打开消息窗，显示定义的文本	Message：在消息窗中要显示的消息正文
E-Mail	发送包含规定消息的电子邮件	From：发送者的电子邮件地址；To：接收者的电子邮件地址；Subject：任何主题；Mesage：消息正文（见下述内容）；Server：电子邮件服务器的名称
变量	OtoStudio 程序的变量将得到警报状态和消息正文串	Variable：变量名称：通过输入帮助(<F2>)，可以选择工程变量：布尔变量将指示报警状态 NORM=0 和 INTO=1，一个整型变量指示不报警状态 NORM =0, INTO =1, ACK =2, OUTOF =4；一个字符串变量得到在域中定义的消息正文；Mesage（见下述内容）
执行	一旦报警事件发生后，马上就开始执行可执行文件	Executable ： 要执行的

		文件名称（例如notepad.exe），可以使用“...”按钮得到标准的 对话框来选择文件；Parameter：调用执行文件所需的参数

消息正文的定义：

对于动作类型'信息'， '打印'， 'Email'或'变量'，可以定义信息正文，在报警事件发生时要输出信息正文。

可以用<Ctrl>+<Enter>在'信息'Email'或'变量'正文定义中插入行分隔符。

在定义报警消息的时候，可以使用下列占位符：

MESSAGE	使用在 报警组的配置中为特别的警报定义的消息正文
DATE	日警报状态到达(INTO)的日期
TIME	警报进入的时间
EXPRESSION	引起警报的表达式（在警报组中定义）
PRIORITY	警报的优先级（对警报组定义）
VALUE	表达式的当前值（见上述内容）
TYPE	报警类型（在报警组中定义）
CLASS	报警类（在警报组中定义）
TARGETVALUE	报警类型目标值 DEV+ 和 DEV-（在警报组中定义）
DEADBAND	报警容差（在警报组中定义）
ALLDEFAULT	输出警报的任何信息，如同在日志文件（历史记录）中行条目描述信息

定义警报消息的例子：

此外，当在报警组中定义报警时，在'Message' 栏中输入"Temperature critical !"

注意： 如果消息正文包括在*.vis-file 或 翻译文件*.tlt 内，在工程语言改变的情况下，消息正文也将受到影响。但是：在这种情况下一象提交到可视化的正文一样，正文必须放在两个"#"之间（例如，在上面的例子中："#Temperature critical !#"和"TIME /EXPRESSION: MESSAGE #current#: VALUE"，以便将此正文输入到象 ALARMTEXT_ITEMs 那样的翻译文件）

在 报警组的配置中要定义'Save'动作的日志文件（见 6.3.4 节）。

对动作的警报事件：对每一个动作，要定义在哪个 报警事件下启动它。

激活要求的事件：

INTO

OUTOF

产生报警。Status = INTO。

用户应答已经完成。 Status = ACK。

报警状态结束。 Status = OUT OF。

类的颜色 / 位图<类名>每个警报类都可以得到自己的颜色和位图，在可视化元件 警报表中，他们将被用于区别报警。为可能的事件 INTO, ACK 和 OUTOF（见上述内容）选择前景颜色和背景颜色。在颜色符号上单击鼠标，马上就打开选择颜色的标准对话框。为了选择位图，在灰色矩形框上的点击鼠标将打开选择文件的标准对话框。

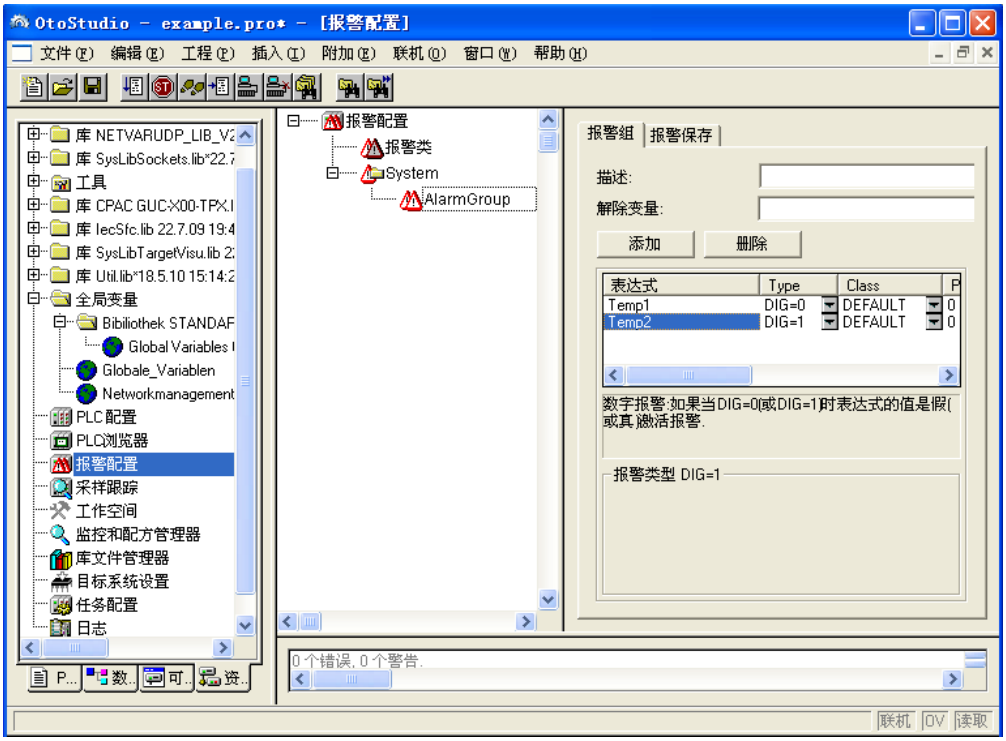
报警组

报警组用于组织可用的警报。每个报警明确的被分配到一个恰当的报警组并且由这个警报组来管理。组里的所有报警逗能被分配一个公共的去激励变量（Deactivation variable）和若干报警存储用的公共参数。

注意，即使是单个报警，也必须在报警组内配置。

通过 fFolder 元素定义报警组的层次结构。在配置树中选择警报组时，将自动显示对话

框 Alarm group：



在 Description 域中，你可以输入警报组名称。

可以定义一个布尔工程变量作为去激励变量（Deactivation variable）。在这个变量的上升沿处，关闭组内的所有警报，在下降沿处它重新激活报警。

通过按钮 Add，可以把报警添加到报警组。在表格窗口插入新行，并且设置下列参数：

Expression：输入工程变量或者报警设计的表达式(例如， "a + b")。为了正确输入，推荐使用输入帮助<F2> 和智能功能

Type: 可以使用如下所列的报警类型。对每种类型注意在该表外作出的相应注释和定义。

DIG=0 数字警报，一旦表达式得到 FALSE 后马上就报警。

DIG=1 数字警报，一旦表达式得到 TRUE 后马上就报警。

LOLO 模拟警报，表达式的值降到报警类型 LOLO 定义的值以下后马上就报警。你可以定义一个容差死区

（Deadband）。只要表达值在容差的范围内，即使 LOLO 值降到界限以下，报警也不被激活。LO 与 LOLO 相应模拟警报，表达式的值超过报警类型定义的值后马上报警。你可以定义容差死区（Deadband）。

只要表达值在容差的范围内，即使 HIHI 值超过界限，报警也不被激活。HI 与 HIHI 相应目标值的偏差；一旦表达式的值降到报警类型 DEV-定义的值加百分比偏差以下就立即报

警。百分比偏差＝目标值*（用％表示的偏差）／100。

目标值的偏差；一旦表达式的值超过警报类型 DEV+加百分比偏差定义的值。百分比偏差＝目标值*（用％表示的偏差）／100。

单位时间变动率；表达式的值严重偏离前面的值后马上报警。激活警报的极限值是通过值的每秒变化（变动率），每分钟变化，或每小时变化来定义。

分类：选择要求的 警报类。选择列表将提供所有的类，这些类是在工程的最后保存之前在报警类配置中定义的。

优先级：可以定义优先级的范围 0-152。0 是最高的优先级。优先级将影响报警表范围内的报警排序。

消息：在报警的情况下，为消息框定义正文。消息框由用户用 OK 来确认，但是，这 OK 不会自动应答警报！为了确认（应答）报警，必须访问报警表。通过可视化元素报警表或者通过进入该表的报警日期，确认（应答）报警是可行的。此日期可以从选择性创建的日志文件里读取。

钝化：输入一个工程变量，在该变量的上升沿，关闭所有创建的警报。

保存报警在 警报类配置对话框中，如果把一个存储动作分配给此类，对每个警报组，都可以定义存储报警事件的文件。

在配置树中选择警报组，打开对话框表‘'Alarm saving'：

可以作下列定义：

文件路径：文件所在的目录路径，这个文件是在文件名称中定义的；通过按钮"..."，你可以得到选择目

录的标准对话框。

文件名：保存警报事件的文件名称（例如，"alarmlog"）。文件是自动创建的，它在这里

得到定义的名称加附属的数字及扩展名".alm".。数字表示日志文件的版本号。第一个文件的数子是"0"；往后的文件根据定义的文件变动事件创建的文件－被编号为 1，2 等。（如，"alarmlog0.alm", "alarmlog1.alm）文件改变事件： 在这里定义引起创建新的报警存储文件的事件。可能的事件有：

Never（无） Hour（一小时后），Week(一周后)，Month（一月后），TriggerVariable（在 TriggerVariable 域中定义的变量的上升沿），Number of record（超过了在 Number of records 中定义的文件记录数）

Triggervariable 和 Number of records：文件变动事件见上述内容。

删除旧文件：文件创建后的天数，过了这几天后除了最新的日志文件，删除所有的报警日志文件。

日志文件 (历史) 包含下列条目：

(参看栏目类型及两个报警的典型输入)

Date/Time in DWORD	Date	Time	Event	Expression	Alarm Type	Limit	Tolerance	Current value	Class	Prior
1046963332	6.6.03	16:08:52	INTO	PLC_PRG.b	LO	-30	5	-31	Alarm-high	0
1043963333	6.3.03	16:08:53	ACK	PLC_PRG.n	HIHI	50			Wraning	9

在日志文件中可以看到的例子：

1046963332,6.3.03 16:08:52,INTO,PLC_PRG.ivar5,HIHI,,,, 9.00,a_class2,0,

1046963333,6.3.03 16:08:53,INTO,PLC_PRG.ivar4,ROC,2,,, 6.00,a_class2,2,

1046963333,6.3.03 16:08:53,INTO,PLC_PRG.ivar3,DEV-,,,,-6.00,a_class2,5,

1046963334,6.3.03 16:08:54,INTO,PLC_PRG.ivar2,LOLO,-35,,3, -47.00,warning,10,warning: low temperature !

1046963334,6.3.03 16:08:54,INTO,PLC_PRG.ivar1,HI,20,,5,47.00,a_class1,2,temperature to high ! Acknowledge !

'附加'菜单：设置

在报警配置中，用指令'扩展' '设置'打开对话框报警配置设置：

Categorie Date/Time：

在这里设置 日志文件中报警表达格式。根据下列语句定义格式。破折号和冒号要放在引号内。

日期： dd'-'MM'-'yyyy -> 如， "12.Jan-1993"

时间： hh':'mm':'ss -> 如， "11:10:34"

Language：

在 OtoStudio 中选择当改变 语言时应该使用的语言文件。注意，为了整个目的，语言文件必须包括报警配

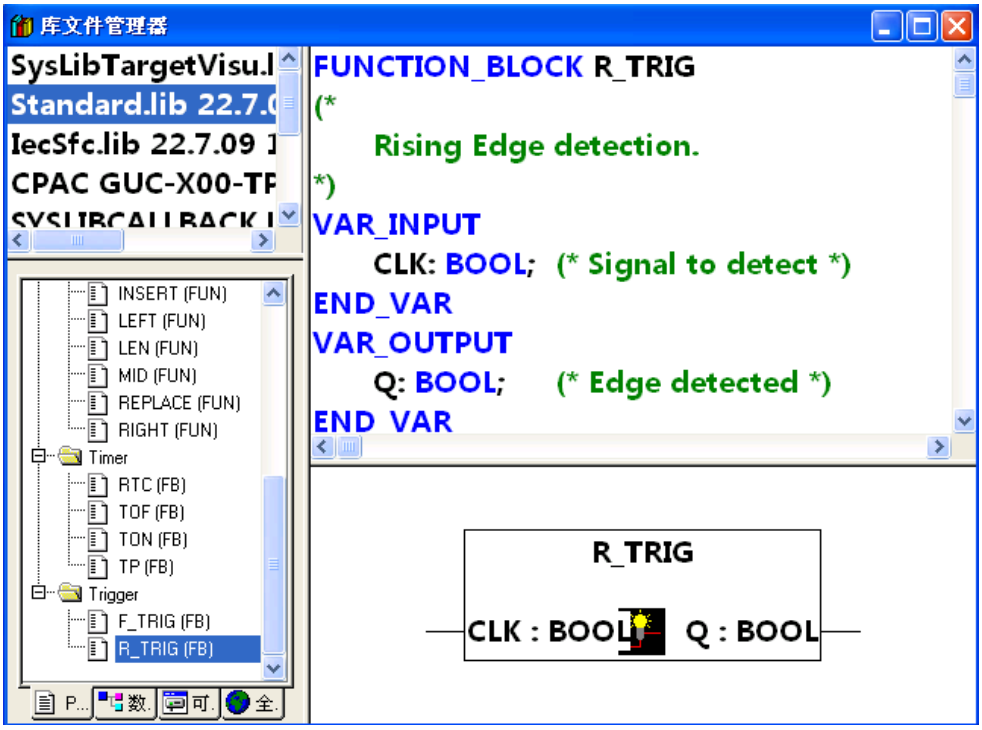
置的文字串的翻译。联系上下文，见下列描述：

- 可视化，设置它的语言，见 OtoStudio 可视化用户手册
- 把工程翻译成其它语言，见 4.3 节

7.3 库文件管理器

库文件管理器显示连接当前工程的所有库。库的 POUs 数据类型和全局变量像用户定义的 POUs 数据类型和全局变量那样用相同的方式被使用。

用指令'窗口' '库文件管理器'打开库文件管理器。在工程中贮存了它包含的各库的信息，并且在‘'扩展库信息'对话框中可看到。打开对话框，选择在库文件管理器中相应的库的名称，并且执行指令'附加' '特性'。



参照:

- 使用库文件管理器
- 标准库
- 自定义的库
- '插入' '扩展库'
- 删除库

使用库文件管理器

屏幕划分器把库文件管理器窗口分为 3—4 部分。附属于工程的库列在区域的左上方。

在这个区域的下部，根据选择的登录卡，列出了上部区域选定库的 POUs，数据类型，可视化或者全局变量。

双击行或者按<Enter>键来打开和关闭文件夹。在关闭的文件夹前有一个加号，在打开的文件夹前有一个减号。

如果点击鼠标或者用箭头键来选择 POU，那么 POU 的声明将在出现库文件管理器的右上方；在右下方是图形显示器，以黑箱的形式来输入和输出。

数据类型和全局变量的声明显示在库文件管理器的右边。

标准库

带有"standard.lib"的库总是可以使用的。它包含了 IEC61131-3 要求的全部功能和功能块作为 IEC 编程系统的标准的 POUs，标准和操作符之间的区别是操作符通过编程系统来

隐含识别，而标准的 POUs 必须连接到工程中(standard.lib)。

这些 POUs 的代码作为 C-库而存在，是 OtoStudio 的一个元件。

自定义的库

如果一个工程用它的实体编译并且没有错误，那么可以用'文件'菜单里的指令'另存为'把它保存在库中。工程本身保持不变。将产生一个附加的文件，这文件有默认的扩展名".lib"。然后，这个库就像标准库一样可以被使用和访问。

为了达到在其它工程中使用该工程的 POUs 的目的，把该工程存储为内部库 *.lib。然后，使用库文件管理器可以把这个库插入到其它工程中。

如果你用其它的 编程语言（如 C 语言）来实现 POUs，并且想让他们进入库，那么用数据类型外部库 *.lib 来存储此工程。你可以得到库文件和另一个带有扩展名"*.h"的文件。这个文件的结构象 C 头文件一样并且包含所有 POUs，数据类型和全局变量的声明，这些声明被库采用。如果在工程中使用外部库，那么在仿真方式

下将执行 POUs 程序，这执行是用 OtoStudio 写的；但是在目标方式下，将被处理用 C 写的程序。

如果你想给库添加许可证信息，那么点击按钮编辑许可证信息...并且在对话框'编辑许可证信息'里插入相应的设置。参见在在 OtoStudio 中 '文件' '另存为...'和 许可证管理的相应的描述'插入' '添加库'用这个指令，你可以把附加的库插入到工程里。

当执行这个指令时，出现打开文件的对话框。选择你想要的带扩展名"*.lib"的库，点击 OK 关闭对话框。

现在该库列出在库文件管理器，你可以象使用用户定义的对象那样使用这些库对象。

一旦包含需要许可证的库，但是没有发现有效的许可证，马上就得到这样的消息：库仅仅在仿真模式中可用或当前的设置目标得不到许可证。那时你可以忽略这消息或者启动关于许可证的相应动作。在编译(工程' '建立').期间，无效的许可证将产生错误。在这种情况下双击错误消息或按<F4>键将打开对话框 ‘'许可证信息'，在这里你可以启动相应的动作（见关于'许可证管理器'相关文档）。

移除库

用指令 '编辑' '删除'可以从工程和库文件管理器中删去库。

'附加' '特性'这个指令将打开对话框'自带库（扩展库）信息'。对内部库你可以发现所有的数据，当库在 OtoStudio 中已经创建完成时，这些数据已经插入到 工程信息中了（包括可使用的许可证信息）。对外部库，将显示库的名称和路径。

7.4 日志记录

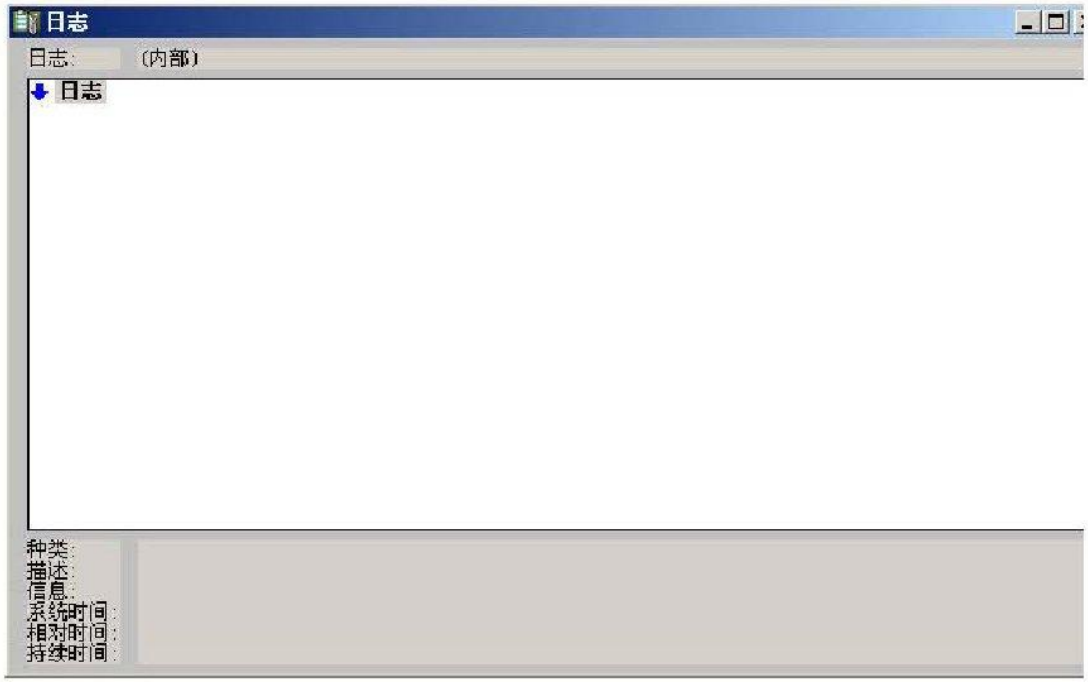
日志按时间顺序保存在线期间发生的动作。为达到这个目的，建立了一个二进制日志文件(*.log)。然后，在外部日志中，用户可以贮存来自相应工程日志记录的摘录。

日志窗口在脱机或在线的模式中打开，于是可以用作直接在线监视器。

参照:

- '窗口''日志'
- 菜单日志
- 贮存文件日志记录
- '窗口''日志'

为了打开日志，选择子菜单'窗口''日志'或者在资源表中选择条目'日志'。



在日志窗口中，当前显示的日志文件名出现在 log 后；如果这是当前工程的日志，将显示单词"(Internal)"。

在日志记录窗口中显示记录的条目。最新条目总是出现在底端。只显示这样的动作：属于在菜单 '工程"选项''日志'的'过滤'域中被激活的类别的动作。

关于当前选择条目的可用信息显示在日志窗口的下面：

种类：特定的日志条目属于的类别。可能是下列四种类别：

•用户动作：用户已经完成了一个在线动作（一般的来自在线菜单）。

•内部动作：在在线层中已经执行了一个内部动作（例如，删除缓冲器或开始调试） 。

•状态变化：运行系统的状态已经变化（例如，如果达到断点那么就从运行变为中断）。

•异常：发生一个异常，例如通信出错。

描述：动作类型。用户动作跟他们相应的菜单指令有相同的名称；其它的动作是用英语描述并且跟相应的 OnlineXXX()功能有相同的名称。

信息: 这个域包含一个在动作期间产生的错误描述。如果没有错误产生，这个域是空的。

系统时间: 动作开始的系统时间，到最近的秒。

连接时间: 从在线活动开始测量的时间，到最近的毫秒。

持续时间: 用毫秒表示的动作延续时间。

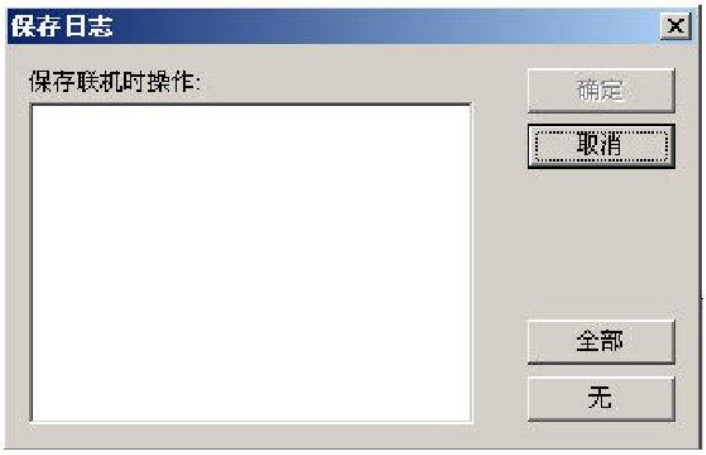
菜单日志当日志窗口有输入活动时，在菜单条中出现菜单选项 Log 而不是'扩展' 和 '选项'里。

菜单包括下列子菜单：

Load…用标准的文件对话框可以加载和显示外部日志文件*.log。此指令不会改写存在于工程里的日志。

如果日志窗口关闭并且稍后再打开，或者启动新的在线活动，那么这个载入的版本将再一次被工程日志取代。

Save…如果工程日志记录正在显示，那么只能用这个子菜单。它允许工程日志记录的摘录贮存在外部文件里。因此，将显示下列对话框，在这里可以选择要贮存的在线活动：



成功的选择后，打开贮存文件的标准对话框('保存日志')。

显示工程日志 如果当前显示外部日志，那么只能选择这个指令。它把显示切换到工程日志。

贮存文件日志记录

不管日志是否贮存在外部文件（见上述内容），工程日志自动贮存在名为 <projectname>.log 的二进制文件。如果在'工程' '选项' '日志'对话框里没有明确的给出不同路径，那么文件保存在存储工程的目录里。

可以在 '工程' '选项' '日志'对话框里输入在线活动的最大数。如果在记录期间超过了这个数，那么就删除最早的那个活动以便为最新的腾出空间。

7.5 任务配置

使用任务管理除了声明特定的 PLC_PRG 程序外，还可以控制工程的处理。

一个任务是一个在 IEC 程序处理中的时间单位。用名称，优先级和确定触发任务启动的条件的类型来 定义它。这条件可以通过时间（周期的，随机的）或者通过内部或外部的触发任务的事件来定义，例如，全局工程变量的上升沿或者控制器的中断事件。

对每个任务，可以设定一串由任务启动的程序。如果在当前周期内执行此任务，那么这些程序会在一个周期的长度内受到处理。

优先权和条件的结合将决定任务执行的时序。

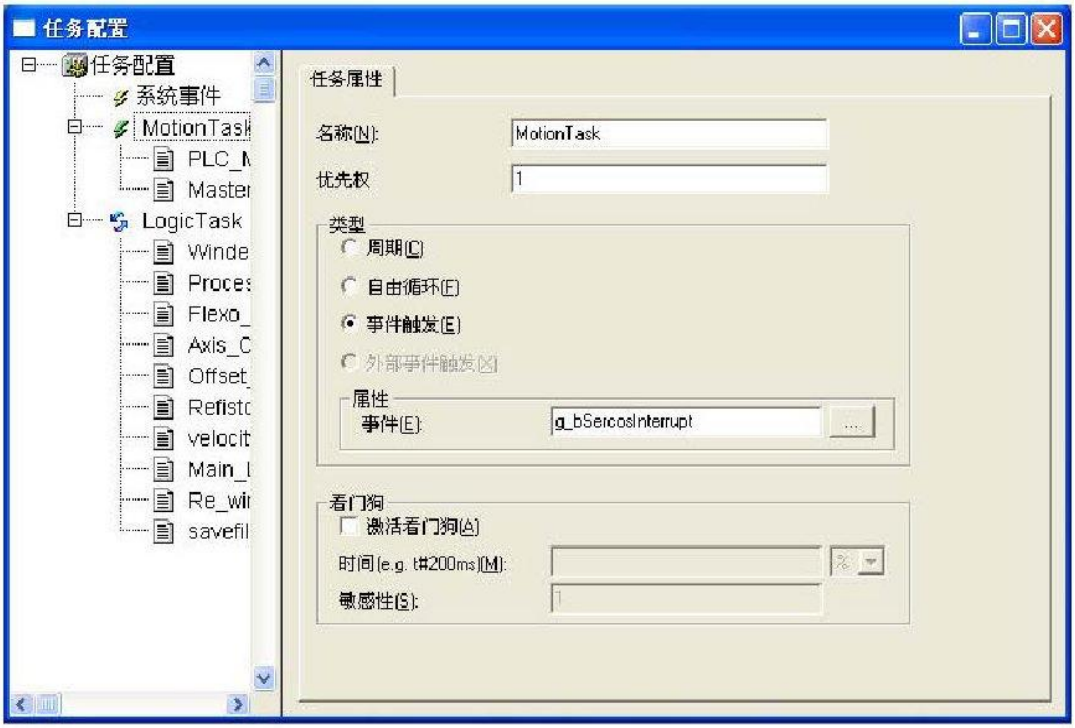
每个任务可以直接 启用或停用。

对每个任务，可以配置一个 监视器（时间控制）。

在在线模式中，可以用图表监视任务的处理。

另外，可以把系统条件（例如，开始，停止，重启）直接与工程 POU 的执行连接起来。

在对象管理器的资源表中可以找到任务配置对象，  。任务编辑器打开在两分窗口中。



在窗口的左边，用配置树描述各任务。在最顶端，你会发现'任务配置'条目。下面有'系统事件'条目和各个用户任务名描述的条目，在每个任务条目下，插入指定的程序调用。每一行前有一个图标。

在窗口的右边，显示一个属于当前标明的配置树条目的对话框，你可以配置任务属性(Task properties)，程序调用(Program call) ，定义系统事件(System events)的连接。

注意: 在几个任务中不要用相同的字符串功能(见 standard.lib)，因为这可能由重入引起的程序错误。

任务配置的工作状态

•在上下文菜单中可找出最重要的指令（鼠标右键）。

•任务配置标题是"任务配置"，如果在标题前有加号，那么顺序列表是关闭的。通过在列表上双击鼠标或者点击<Enter>，可以打开列表。出现减号。通过再次双击，可以再次关闭列表。对每个任务，有一个附属的调用程序列表。用同样的方式可打开和关闭这个列表。

•用 '插入' '插入任务' 指令，可以插入一个任务。

•用 '插入' '附加任务' 指令，可以在配置树底端插入一个任务。

•用 '插入' '添加调用程序'，可以把一个程序调用分配给在配置树中选择的实际任务。

•进而，对于配置树中的每个表项，相应的配置对话框出现在窗口的右边。选项可被激活／关闭。对编辑域可以输入。取决于配置树中选择的表项，有定义任务属性'任务属性' (见 '插入任务')的对话框，定义程序调用'程序调用' (见 '添加调用程序') 或者系统事件 系统事件

表格的对话框。 一旦把此窗口区设置给配置树，配置树立即接受在对话框中作出的设置。

•任务名称或程序名称也可以在配置树中编辑。为此，在名称上点击鼠标或者选择此表项按<Space> 键来打开编辑框。

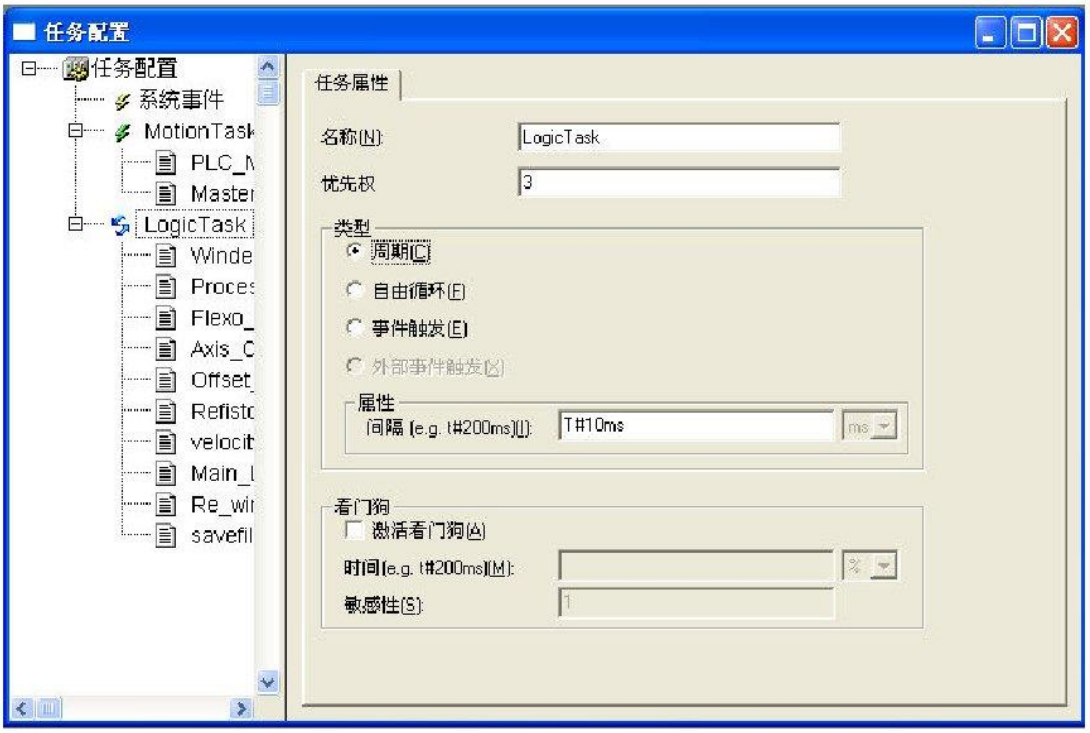
•可以用箭头键选择配置树中的上一个或下一个表项。

•通过点击任务名或程序名，或者通过按<Space bar>，可以在名称周围设置一个编辑控制框。然后可在任务编辑器中直接改变名称。

'插入' '插入任务' 或 '插入' '扩展任务'用这个指令，可以把新任务插入到任务配置。每个表项由符号和任务名称构成。

如果选择了一个任务或表项'系统事件'，那么你可以用指令'插入任务'，将一个新的任务插到被选的任务之后。如果选择输入'任务配置'，那么可以用'扩展任务'，把一个新的任务插入到现有的列表的表项底端。

目标系统定义任务的最大数量为 32。在插入一个任务时，将打开设置任务属性的对话框：



插入要求的属性：

名称: 任务名称，在配置树中用这个名称描述任务；在选择表项上单击鼠标或按<Space>键后，可以编辑名称。

优先权(0-31): (0 和 31 之间的数字；0 是最高优先权， 31 是最低优先权)。

类型：

循环（）：根据在域'Interval'中的给出的时间定义，周期处理此任务（见下述内容）。

自动循环 () ：程序启动后，马上就处理此任务，并且在一次运行结束时，自动重启动一个连续循环。没有定义循环时间。

触发事件 () ：一旦在 Event 域中定义的变量达到上升沿马上启动此任务。

触发附加事件 () ：一旦在 Event 域中定义的系统事件发生后，马上启动此任务。在选择列表中支持和提供哪些事件依赖于目标系统。（只针对扩展 IO 模块支持该类型任务）。

属性：

间隔(用于 '循环' 或'自动循环'类型)：任务重启动的时间周期。你只要输入一个数字，然后在编辑域后的选择框中选择想要的单位（毫秒[ms]或者微秒[µs]）。重画窗口后，就会用 TIME 格式中马上显示输入（例如"t#200ms"）但是你也可以直接用 TIME 格式输入值。用[ms]的输入总是显示为纯粹的数字(例如，"300")。

事件(用于 '触发事件' 或 '触发扩展事件'类型)：是一个其上升沿被察觉后，马上触发任务启动的全局变量。用按钮...或者输入帮助键<F2>得到一个包含所有可用的全局变量的列表。

如果在上述两个属性处中均没有输入，那么任务间隔将取决于运行系统；例如，在这种情况下，对 OtoStudio V2.1 和更高版本使用 10 ms 间隔。

监视器：

激活监视器： 当激活这个选项时()，一旦处理所用时间超过在'时间'域中定义的时间，任务马上以出错状态停止。

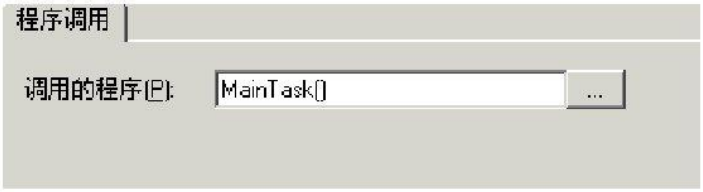
时间 (e.g.: t#200ms)：监视时间；在这一段时间期满后，将激活 watchdog，除非任务尚未中止。

灵敏度：允许不发错误的监视时间超时次数。

'插入' '添加调用程序'或 '插入' '扩展程序调入'

在任务配置中，用这些指令可以打开向任务输出程序调用的对话框。在任务配置树中每个表项都由符号（）和程序名称组成。

用'插入程序调入'，在被选择程序调用前，插入一个新的程序调用，用'扩展程序调入'，将这个程序调用 l 添加到现有的列表或程序调用的底端，一个任务可以插入多个程序调用。



在域'程序调入'中指定该工程中的有效程序名称或用 Select 按钮打开输入帮助（导入向导） 以选择一个有效的程序名。程序名称也可以在配置树中修改。为此选择表项输入按<Space>键或者点击鼠标打开编辑域。

如果选择的程序需要输入变量，那么用通常的形式和被声明的类型输入这些变量（例如，prg(invar:=17)）。

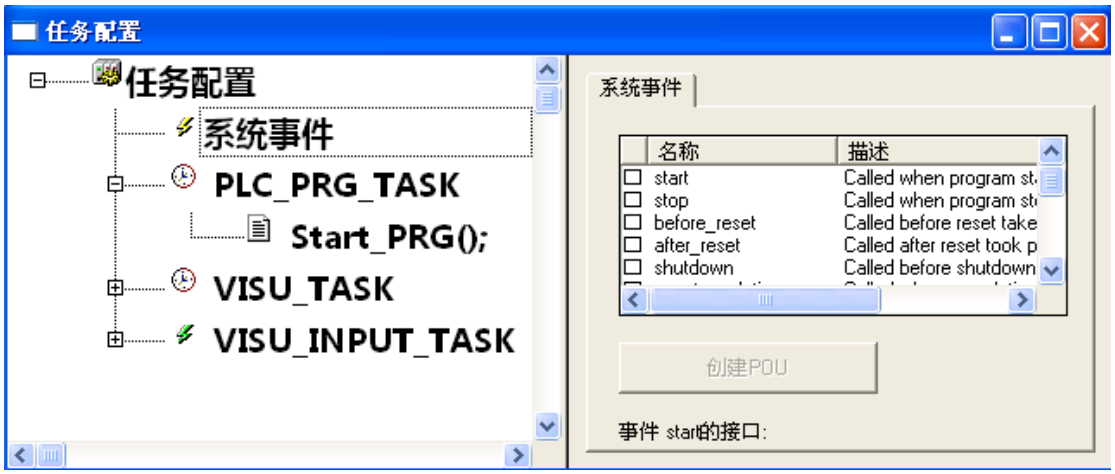
在线模式中，程序调用的处理将根据任务编辑器中的次序（从上到下）进行。

注意: 在几个任务中不要使用相同的字符串功能（见标准库元素），因为，在这种情况下任务处理期间可能对变量产生冲击。

系统事件

用系统事件而不是任务也能调用工程的 POU。可用的系统事件是目标系统特定的。例如，可能的事件有：停止，开始，复位。

在任务配置编辑器中把系统事件分配给 POUs。用对话框'事件'，在任务配置树中表项马上就打开此对话框：



"系统事件"后，对每个事件的描述：名称 和 描述按目标文件中的定义显示，在 called POU 栏中，可以输入，事件产生后马上要调用的工程 POU 名。

为此，使用帮助键(<F2>)或者手工输入已经存在的 POU 的名称（例如，"PLC_PRG" 或 "PRG.ACT1"），或者插入尚未存在的 POU 的名称。为了创建这个 POU，按 Create POU 按钮。于是，在对象管理器中插入此 POU。

事件需要的输入输出参数自动定义在 POU 声明中。在分配表下，用图形显示当前选择的事件和要求的参数。

如果你实际上希望用事件调用 POU，在分配表()激活此表项。用鼠标点击控制框完成激活/关闭 。

正在处理哪个任务

•任务的执行应用下列规则

•执行满足条件的任务，即，如果其指定的时间到期或在其条件（事件）变量成县上升沿。

•如果几个任务都具备有效的条件，那么将执行最高优先权的任务。

•如果几个任务都具备有效的条件和同等的优先权，那么首先执行等待时间最长的任务。

•系统调入的处理将根据任务编辑器中的顺序（从上到下）完成程序调用处理。

联机模式下的任务配置

在线模式中，每个任务的状态和通过的周期数将在配置树中显示。在图表中监视时间流动，其前提是：

在工程中包含库 SysTaskInfo.lib 和 SysTime.lib，以提供内部计算任务次数的功能。一旦支持任务监视的目标设置后，就立即自动包含了这些库。

在结构树中的显示任务状态：

在线模式中，任务的当前状态和已经通过处理的周期数显示在配置树中任务表项末端的括号里。这个更新的间隔跟通常 PLC 值的监视一样。可能的状态有：

Idle:自最后一次更新以来，还没有启动过；尤其是用于事件任务

Running:从最后一次更新以来至少启动过一次

Stop 停止

Stop on BP 因为达到任务中的断点而停止

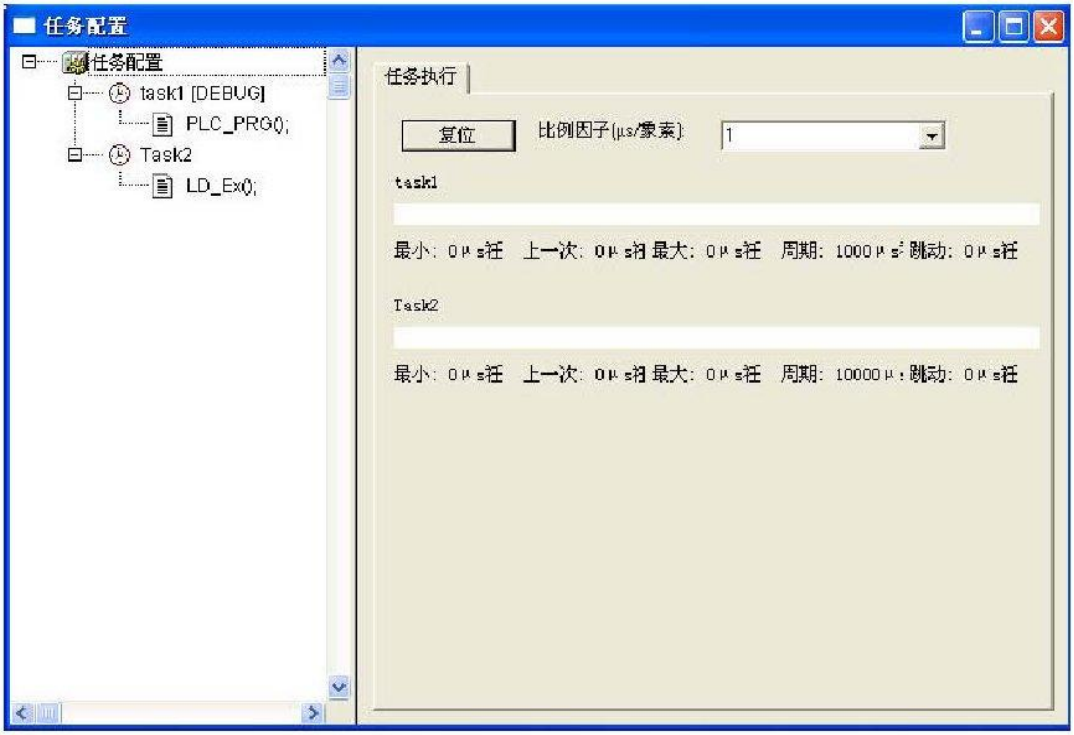
Stop on Error 错误，例如，被零除，页面错误等 Stop

Watchdog 已经超过周期时间了

在'遇错停止' 或 '停止监控'状态下，任务项将显示红色。

任务的时间流显示

如果在配置树中选择'任务配置'表项，那么将在窗口右边用条形图显示任务的使用情况：



对每个任务，都显示一个条形图。条的长度表示了周期的长度。在条的下面，下列测量值用用条的相应的标记来图解。

Min: 测得的最小运行时间 μs

Akt: 上一次测得的运行时间 μs

Max: 测得的最大运行时间 μs

Cycle: 周期的总长 μs

Jitter: 测得的最大抖动量精确到 μs

复位:按钮可以用来把 Min., Max. 和 Jitter 的值归零。

图表的比例（微秒／像素）可以通过在 Scaling [μs/Pixel]的选择列表的帮助来调节。

在上下文菜单和'Extras'菜单中的其它在线功能：

'附加' '设置调试任务'

用这个指令，在 任务配置中，在线模式下设置一个调试任务。出现文本[DEBUG]设置任务后。

然而，仅仅对于这个任务具有调试能力。换言之，只有由此设置的任务运行的程序，才能停到断点。

'附加' '启用/禁用任务'用这个指令，可以使在任务配置中当前标出的任务启用或停用。在程序处理期间，不考虑停用的程序。

在配置树中，它用灰色的表项表示。

'附加' '调用堆栈' 在 任务配置中这个指令可以在扩展菜单中采用。如果在调试期间程序停在断点，它可以用来显示相应的 POU 的调用栈（调入堆栈）。为了这个目的，必须在任务配置树中选择 调试任务。打开窗口'调入任务中的堆栈 <task name>'。得到 POU 的名称和断点位置(例如， "prog_x (2)" for line 2 of POU prog_x)。下面以向后的顺序显示全部的调用栈。如果按按钮'转到'，视点将跳到在调用栈中当前标出的 POU 中那个位置。

7.6 监控和配方管理器

在监控和配方管理器的帮助下，可以观察所选变量的值。监控和配方管理器还可以用确定的值预置变量并且以组的形式把他们传送给 PLC ('写入配方')。用同样的方式，当前 PLC 的值可以读入和储存在监控和配方管理器里('读取配方')。这些功能对于设置和输入控制参数是有用的。

在监控和配方管理器的左栏显示所有 建立的监控表。点击鼠标或箭头键来选择这些列表。在监控和配方管理器的右边，显示在任何给定的时间都可应用的变量。

为了使用监控和配方管理器工作，打开对象管理器中的"资源"选项卡中打开监控和配方管理器。

脱机方式下的监视和收据管理器

在脱机方式下，用 '插入' '新建监控列表'在监视和收据管理器中创建几个监视列表。

为了输入被监视的变量，可以用输入帮助召唤一个包含所有变量的列表，或者根据下列表示法用键盘输入变量：

<POUName>.<Variable Name>

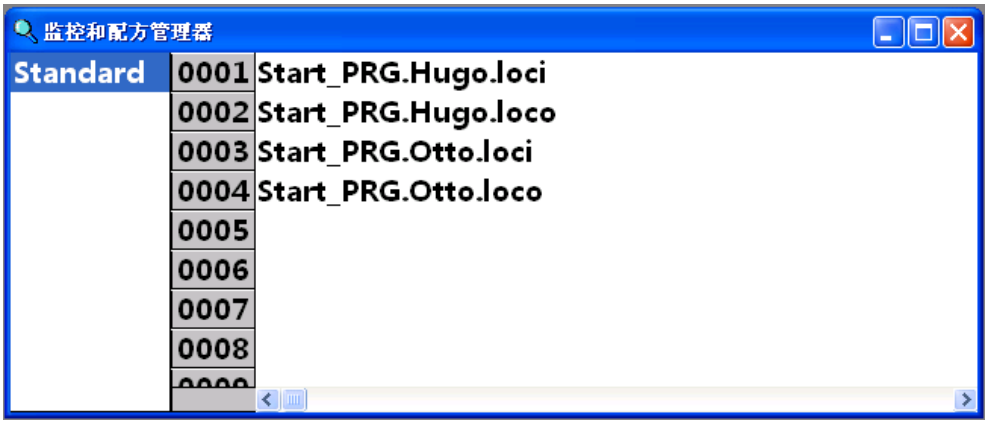
采用全局变量时则可省去 POU Name。从点开始。再次，变量名称可包含多层。可直接输入地址。

多层变量的例子：

PLC_PRG.Instance1.Instance2.Structure.Componentname

全局变量的例子：

.global1.component1



在监视列表中的变量可以用常数值预置。那意味着在线模式中可以用 '附加' '写入配方' 指令把这些值写入变量。必须用:=来分配变量的常数值：

例如：

PLC_PRG.TIMER:=50

在例子中，PLC_PRG.COUNTER 变量用值 6 预置'插入' '新建监控列表'

在脱机方式中用这个指令，可以把新的监视列表插入到监控和配方管理器。在出现的对话框中输入要求的监视列表名。

'附加' '删除监控列表'

用这个指令可以在监控和配方管理器中改变监控列表名称。在出现的对话框中输入监视列表新名称。

'附加' '保存监控列表'

用这个指令可以保存 监控列表。它打开保存文件的对话框。文件名用监控列表名来事先调整并且得到扩展名"*.wtc"。

用 '附加' '读取监控列表'可再次加载已保存的监控列表。

'附加' '读取监控列表'用这个指令可以重新加载已保存的监控列表。它打开一个为打开文件的对话框。用扩展名"*.wtc"选择想要的文件。在出现的对话框中，可以给监控列表一个新的名称。预制不带扩展名的文件名。

用 '附加' '保存监控列表'，可以保存监控列表。

在线模式下的监控和配方管理器

在线模式中，显示输入变量的值。

结构体的值（数组，结构或者功能块实例）用加号在标识符前面作记号。用鼠标点击加号或者按<Enter>，打开或关闭变量。如果在监控列表中选择一个功能块变量，那么就把相关的上下文菜单扩展到包含'缩放' 和'打开实例'。

为了输入新的变量，用 '附加' '激活监控'指令关闭显示。输入变量后，可以用相同的指令重新激活值的显示。

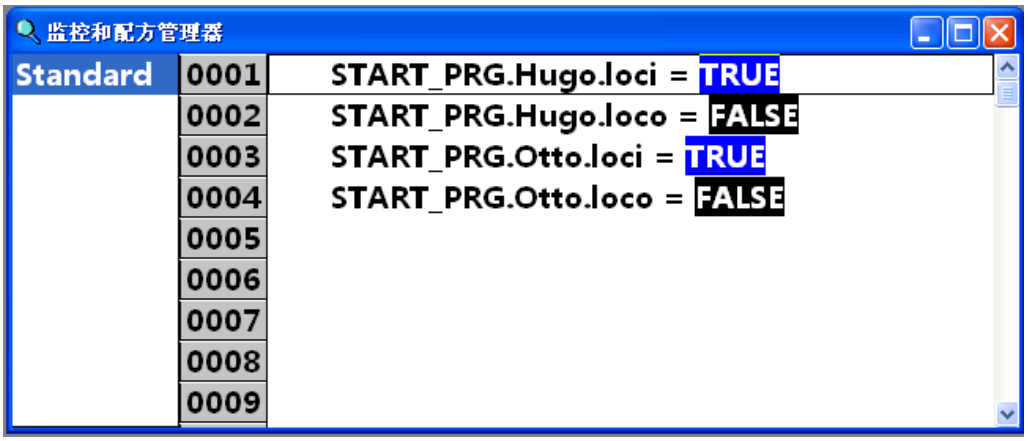
在脱机方式下，可以用常数值预置变量（通过在变量后输入:= <value>）。在线方式下，用 '附加' '写入配方'指令可以把这些值写入变量。

用 '附加' '读取配方'指令，可以用变量现有的值替换变量的预设值。

注意: 只能加载那些在监控和配方管理器中选择的监控列表值!

'附加' '监控激活'在 监控和配方管理器在线方式下， 用这个指令打开或者关闭显示。如果显示激活，记号(?)将出现在此菜单项前。

为了输入新的变量或预置一个值（见脱机方式），通过此指令关掉显示。输入变量后，可以用相同的指令激活值的显示。



'附加' '写入配方'

在监控和配方管理器的在线方式下，用这个指令可以把预设值写入变量（见脱机方式）。

注意: 只能加载那些在监控和配方管理器中选择的监控列表值!

'附加' '读取配方'在监控和配方管理器的在线方式下，用这个指令可以用变量现有值替换变量的预设值（见 脱机方式）。

例如：

PLC_PRG.Counter [:= <present value>] = <present value>

注意: 只能加载那些在监视和收据管理器中选择的监视列表值!

强制值：在监视和收据管理器还可以'强制值' and '写入值'。如果点击某个变量值，那么就打开对话框，在这里可以输入新的变量值。改变的变量在监视和收据管理器中以红色显示。

7.7 工作空间

资源表项中的这个对象提供当前设置的工程选项的一个映象（见 4.2 节，工程选项）。如果打开它，可以得到已知的类别的选项'对话框。

7.8 对象系统设置

"对象设置"是一个资源对象。在这里要定义工程要使用什么样的对象系统和怎么配置它。如果用'工程"新建'开始一个新工程，那么打开一个目标设置对话框，意味着为目标预先定义一个配置。

可用的目标列表依赖于哪个目标支撑软件包 TSP（对象支持包)安装在电脑中。这些描述平台的特定的基本配置和定义，在目标设置对话框中用户可以修改。

注意: 如果无有效的 TSP，那么在目标系统选择框只能设置为'None'。这将自动切换到仿真模式不可能设定配置。

参照:

- 对象支撑软件包
- 对象设置对话框

对象设置对话框

当创建新的工程时，对话框对象设置自动打开。用选择对象管理器资源登陆中选择子菜单'对象设置'也能打开此对话框。

选择在配置中提供的目标配置中的一个。

如果没安装目标之称软件包,只能选择'None'，这意味着在仿真模式下工作。如果选择了一个已安装了的配置，那么它依赖于目标文件中的条目，在 OtoStudio 对话框中可以定制它的配置。如果选择了一个在计算机上没有有效的许可证的配置，OtoStudio 要求选择另一个目标。

如果选择了一个在相应的目标文件中有条目"隐藏设置"的配置，那么只能看见配置的名称。否则，有五个可用的对话框来修改给出的配置：

- 目标平台（Target Platform）
- 内存安排（Memory Layout）
- 常规配置（General）

- 网络功能（Networkfunctionality）
- 可视化（Visualization）

注意 ：要知道，对预先确定的目标配置的每次修改可以引起目标性能和特性的重大变化！如果你希望把目标设置恢复导目标文件给出的标准配置，请按<Default>。

7.9 PLC 配置

在对象管理器中，  PLC 配置是登录卡 Resources 中的一个对象。你必须用 PLC 配置编辑器描述建立工程的硬件（包括运动控制轴，IO 模块和人机界面），输入输出的数量和位置对于程序的实现尤其重要。用这个描述，OtoStudio 检验程序中使用的 IEC 地址是否真正存在于硬件内。具体的操作请参见《OtoStudio 总线 IO 扩展模块使用手册》,《OtoStudio 运动控制库编程手册》。

7.10 采样追踪

7.10.1 综述和配置

在 OtoStudio 资源中可以采用  Sample tracing 对象。采样追踪可以用来追踪已被跟踪了一段时间的变量的的值的行踪。这些值被写入了环形缓冲器(trace buffer)。如果储存器已满，那么就会重写最早的值。可以同时追踪多达 20 个变量。每个变量可以最多追踪 500 个值。

因为在 PLC 内追踪缓冲器的规模有一定值，在很多或很广的变量 DWORD 的事件中，只能追踪少于 500 个值。

例如：如果追踪 10 WORD 变量并且如果在 PLC 中储存器有 5000 字节长，那么对每个变量，可以追踪 250 个值。

为了能够执行一个追踪，在对象管理器， 在资源登录卡中为一个采样跟踪打开此对象。建立和加载相应的追踪配置，定义被追踪的变量。(见'附加' '跟踪配置')。

在 PLC 中创建了配置和启动了追踪后，那么就会追踪变量值。用'读取跟踪'，最后追踪的值将被读出并显示除图形曲线。

在工程格式(*.trc) 或在 XML 格式 (*.mon) 中，可以保存和重新载入 Trace（变量值和配置）。通过*.tcf-file，仅仅储存和重新载入配置。

在工程中可以采用各种追踪。他们被列在窗口的右上角选择列表('Trace')中。可以选择其中的一个用于当前的追踪配置。

注意: 如果任务配置用于控制程序，那么追踪功能归属于调试任务。

选择要显示的变量

窗口右边的组合框定义追踪配置中的最终变量，窗口的右边显示追踪曲线。如果从列表选择一个变量，那么在读除追踪缓冲器后，变量将用相应的颜色显示(Var 0 绿色，等)。在曲线已经被显示时，便可选择变量。

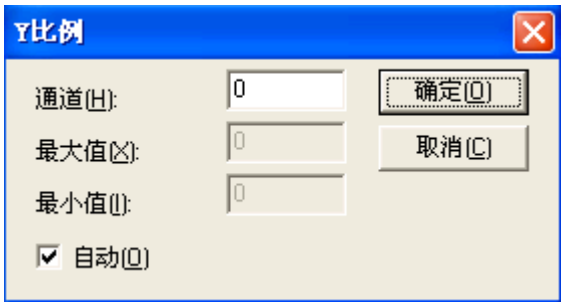
在追踪窗口中最多可以同时观察八个变量。

'附加' 'Y 向比例'

用这个指令在追踪显示中可以改变曲线预设的 Y 比例。双击曲线，将得到对话框'Y 向比例'。

只要激活选项'自动读取跟踪'，便将使用默认的比例，这比例取决于使用的变量的类型。为了改变比例，关闭选项'自动读取跟踪'并且输入相关曲线的通道号(Channel)和 y 轴上新的最大值和最小值。

最早的通道号和比例位时预制的。



'附加' '启动跟踪'

符号  :用这个指令，追踪配置传送到 PLC，并且，追踪采样在 PLC 中启动。

'附加' '读取跟踪'

符号  :用这个指令，从 PLC 读当前的追踪缓冲器，并且，显示被选变量的值。

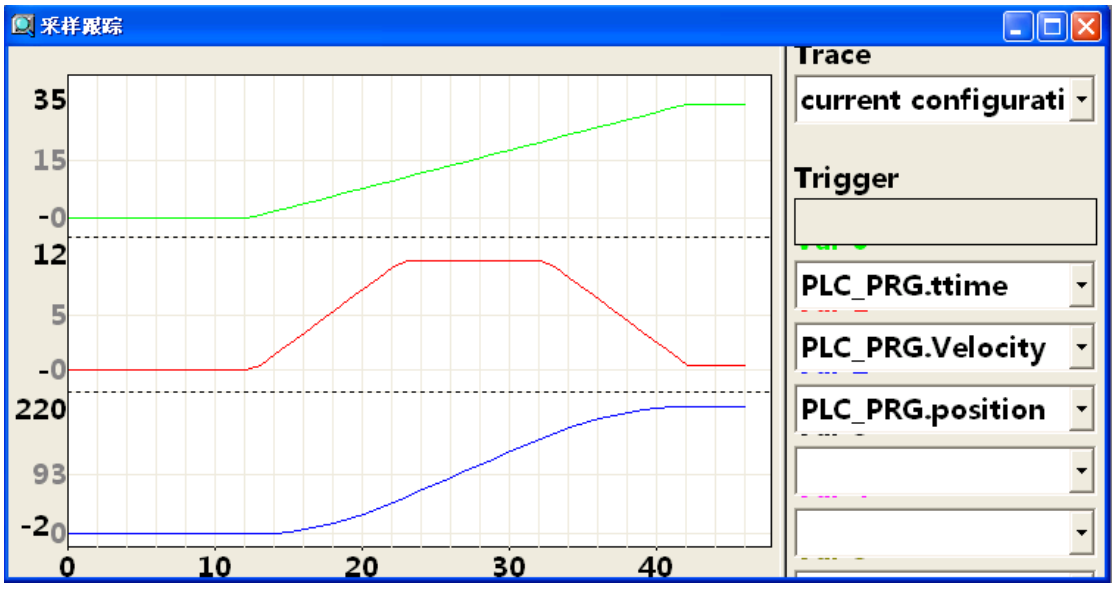
用这个菜单的一些指令，把追踪配置和追踪值存入文件或从文件加载追踪配置和追踪值，从控制器把追踪加载到工程，或者某个追踪设置成工程要用的追踪。

注意: 考虑用菜单指令'附加"保存跟踪'来保存和重载跟踪的另一种方式

'附加' '停止跟踪'

符号  :这个指令在 PLC 中停止采样追踪。

7.10.2 采样追踪的显示



如果加载了一个追踪缓冲器，那么将读出和显示所有要显示的变量的值。如果没有设置扫描频率，那么将在 X 轴上写上追踪值的连续数字。追踪窗口的状态指示器（第一行）显示追踪缓冲器是否装满，追踪什么时候完成。

如果指定了扫描频率的值，那么 x 轴将规定追踪值的时间。把这个时间分配给“最早的”追踪值。例如，显示最后 25 秒的值。

把值以相应的数据类型写在 Y 轴上。一句使最低和最高位都能安插在显示区的方法设置比例。例如，Var0 的最低值是 6，最高值是 100，其比例设置在左边。

如果满足触发条件，那么在触发条件出现前后的分界面处显示一条垂直的点线。

在改变工程或者离开系统前一致保存已被读出的存储器。

'附加' '光标模式'

在监视区设置光标的最简单的方法就是用鼠标左键点击那里。那样就会出现光标并且可用鼠标移动它。

在监视窗口顶部，显示当前光标的 x 位置。在相邻的'Var 0', 'Var 1', ..., 'Var n'域中，显示各自变量的值。

另一种方式是指令'附加' '光标模式'。用这个指令两条垂直的直线将出现在采样追踪中。

开始时，它们重迭在一起。用箭头键可以左右移动其中的一条直线。按<Ctrl>+<left> 或<Ctrl>+<right>，可以使移动速度增加 10 倍。

如果再按<Shift>键，可以移动第二条直线，显示与第一条线间的差。

'保存到文件'

用这个指令，在文件中以 XML 格式保存追踪配置和追踪值。为了这个目的，打开保存文件的标准对话框。将自动使用文件扩展名*.mon.

用指令'从文件中加载'可以重载*.mon-文件。

'从文件中加载'

用这个指令可以把 XML 格式文件（*.mon）中的追踪配置和追踪值加载到工程中。它打开一个打开文件的对话框你可以浏览带扩展名*.mon 的文件。加载的追踪将被显示并且被添加到配置对话框域'Trace’中的选择列表。如果要把它设置为当前工程追踪配置，那么使用指令'应用于工程配置'。

用指令'保存到文件'可以建立一个*.mon-文件。

注意: 考虑通过使用菜单'附加' '保存跟踪值'的指令保存追踪的一种替代方法。

'附加' '读取跟踪'

符号  :用这个指令从 PLC 读现存的追踪缓冲器，并且显示被选变量的值。

'应用于工程配置'

用这个指令，可以把有效的追踪列表（追踪窗口中域'Trace'）中当前选择的追踪配置设置为此工程内的现行配置。

选择列表除了当前使用的（在顶端）外还提供了通过指令'从文件中加载'从*.mon-文件载入工程的所有追踪。

7.10.3 保存采样追踪

用这个菜单的指令把追踪配置和追踪值保存到文件中，从文件把它们重载到工程中。此外，可以把追踪保存到 ASCII 形式的文件中。

注意: 考虑通过使用菜单'附加' '外部跟踪配置'的指令储存和重载追踪的一种替代方法(XML 格式, *.mon-文件)！

'保存值'

用这个指令可以保存采样追踪的值和配置数据。打开保存文件的对话框。文件名接受扩

展名"*.trc"。

注意：在这里保存追踪的值和追踪配置，然而，在配置对话框中'保存值'仅仅涉及配置数据。

保存的采样追踪可以用'附加' '加载值'再次载入。

'加载值'

用这个指令，可以重新载入保存的采样追踪的值和配置数据）。打开一个打开文件的对话框。用"*.trc"扩展名选择想要的文件。

用'附加' '保存值'

可保存采样追踪。

'附加' '拉伸'

符号  :

用这个指令伸展开显示的采样追踪的值。用水平的画面调整条设置开始的位置。随着一次次的重复伸展，显示在窗口中的追踪段将不断放大。

这个指令与'附加' '压缩'相对。

'附加' '显示网格':

用这个指令你可以在绘图窗口切换栅格的开和关。当打开栅格时，记号(　)将出现在相邻的菜单项里面。

'附加' '压缩'

符号  :用这个指令压缩显示采样追踪的值；即，使用这个指令后可以在更大的时间框架内看追踪变量的变化。可以多次执行此指令。

这指令与'Extras' 'Stretch'相对。

'Extras' 'Cursor Mode'在监视区设置光标的最简单的方法就是用鼠标左键点击那里。随之出现光标并且可用鼠标移动它。在监视窗口顶部，显示当前光标的 x 位置。在相邻的'Var 0', 'Var 1', ..., 'Var n'域中，显示各自变量的值。

另一种方法是指令'Extras' 'Cursor mode'。用这指令两条垂直的直线将出现在采样追踪中。开始时它们重迭在一起。用箭头键可以左右移动其中的一条直线。按<Ctrl>+<left> 或<Ctrl>+<right>，可以使移动的速度增加 10 倍。

如果再按<Shift>键，可以移动第二条直线，显示与第一条线之间的差。

'附加' '多通道'

用这指令可以在采样追踪的单一通道和多重通道的采样追踪显示之间改变。多通道显示时在菜单项前有记号( )。

已经事先设置了多通道显示。显示窗口被分为八个显示曲线。对每条曲线，最大值和最小值显示在边缘上。

在单一通道显示，所有的曲线用相同的比例因子显示并且迭加在一起。当显示曲线紊乱时，它很有用。

'值以 ASCII 码形式保存'

用这指令，可以把采样追踪保存在 ASCII-文件中。它打开一个打开文件的对话框。文件名接受扩展名"*.txt"。值按照下列模式存放在文件里：

BODAS Trace

D:\BODAS\PROJECTS\TRAFFICSIGNAL.PRO

Cycle PLC_PRG.COUNTER PLC_PRG.LIGHT1

0 2 1

1 2 1

2 2 1

.....

如果在追踪配置中没有设置扫描频率，那么第一栏是周期；那意味着每一周期在任一给定时间只记录一个值。另一方面，这一项是以毫秒数为单位存储变量值的时间点。

后面多栏保存追踪变量的相应的值。在任一给定时间，用空格把值彼此分开。

依据顺序，在第三行显示一个接一个的附属变量名，其顺序是(PLC_PRG.COUNTER, PLC_PRG.LIGHT1)。

7.11 PLC 浏览器

PLC-浏览器是一个基于文本的控制监视器（终端）。以一个输入项输入来自于控制器的特定的信息请求命令，字符串发送给控制器。在浏览器的一个结果窗口显示返回的响应串。这功能为诊断和调试按钮服务。

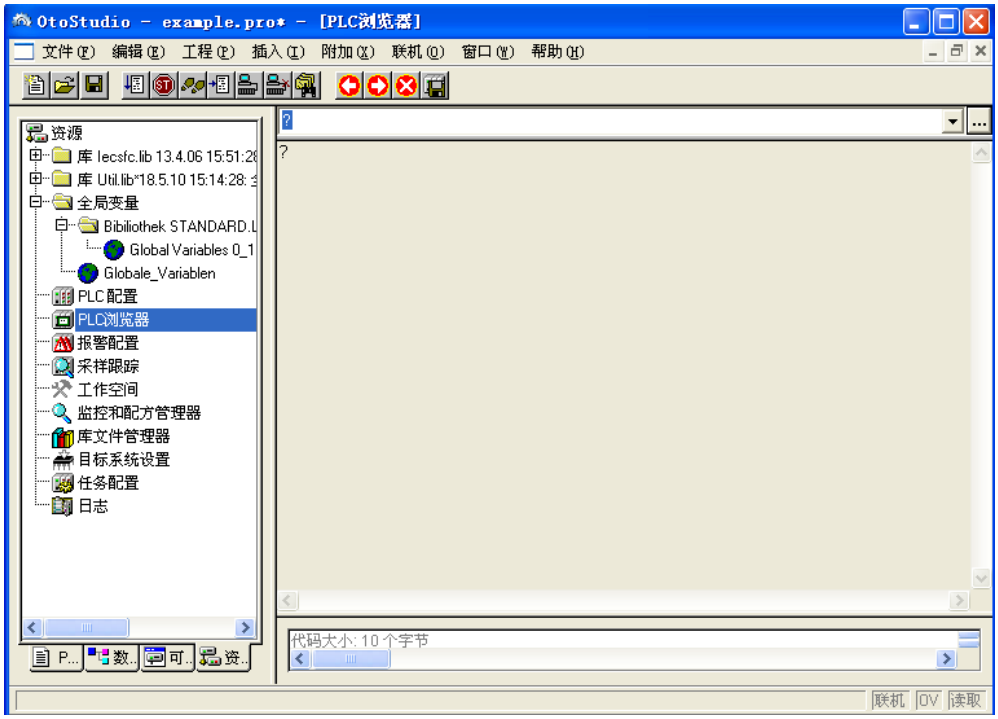
设置号的目标系统可用指令由 OtoStudio 的标准集加控制器生产商可能的扩展集组成。他们在 ini 文件中被管理，从而在运行系统中被执行。

请参考：

- 关于 PLC 浏览器操作的一般讨论
- 在 PLC 浏览器中的命令输入
- 在 PLC 浏览器输入命令时使用宏指令
- PLC 浏览器的更多选项

关于 PLC 浏览器操作的一般讨论

在资源中选择 PLC-浏览器。只要在当前目标设置(网络功能类目中)激活它，就可以采用它。



浏览器由指令输入行和结果 / 显示窗口组成。

在选择框中，输入行显示自工程启动以来所有输入的指令列表输入历史记录。它们是可以再次选择直到关闭工程。只有那些不同于现有的指令才被添加到此列表中。

用<Enter>把输入的指令发送到控制器。如果不是在线连接，那么在结果窗口用与发送到控制器的同样的方式显示指令，否则，在那里显示来自于控制器的响应。如果把新的指令发送到控制器，那么会删除结果窗口的内容。

可以以指令串 的形式输入指令，也可能使用宏指令。

在 PLC 浏览器中的命令输入

基本上 PLC-浏览器可以采用在标准的调试诊断命令。这些标准指令可以进一步的补充专门指令，例如，自我诊断功能或控制应用的其它状态信息。一般指令语句：

<KEYWORD><LEER><KEYWORD-DEPENDEND PARAMETERS>

Keyword 是指令。可扩招那的指令参数在输入帮助窗口相关的工具提示中描述。基本命令，可通过？获取

在输出数据窗口重复已经被发送的指令，控制器的响应出现在它下面。

例如，用指令“pid”向控制器请求工程的 Id。

在命令行输入：

pid.....

在结果窗口输出：

pid

Project-ID: 16#0025CFDA

为每个标准指令用“？” <BLANK><KEYWORD>提供帮助文本。在 ini 文件中给出了类似的定义。

下列指令完全集成在运行系统中，并且其相应的条目包含在 ini 文件中以便提供输入帮助，工具提示和帮助：

指令	描述
?	运行系统提供可用的指令列表。此列表与目标系统描述文件的状态无关。
mem	内存区域的十六进制映像 语句 1：mem <start address> <end address> 语句 2：mem <start address>-<end address> 可以用十进制，十六进制或作为一个宏来输入地址。
Memc	与控制器中代码的起始地址相关的十六进制影响 ；象 mem 一样，数据被添加到代码区。
memd	与控制器中数据库地址相关的十六进制映像；象 mem，数据被添加到代码区。
reflect	为试验目的，反映当前命令行
dpt	读数据指针表
ppt	读 POU 表格
pid	读工程 Id
pinf	读工程信息

tsk	显示包含任务信息 IEC-task 列表
startprg	启动 PLC 程序
stopprg	停止 PLC 程序
resetprg	复位 PLC 程序。仅仅初始化非保留的数据
resetprgcold	冷复位 PLC 程序。也要初始化保留的数据
resetprgorg	复位最初的 PLC 程序。删除当前应用程序及所有数据（包括保留数据和持久数据）
reload	重新加载引导工程
getprgprop	程序特性
getprgstat	程序状态
filedir	文件指令"dir"
filecopy	复制文件[从] [到]
filerename	重命名文件[旧名称] [新名称]
filedelete	删除文件[文件名]
saveretain	保存保留变量
restoreretain	加载保留变量
setpwd	在控制器上设置密码 语句： setpwd <password> 其中 level＝0（默认值）在从偏程系统登陆时才有效 level＝1 对所有应用均有效
delpwd	删除控制器密码

注意: 输入的指令次序的第一个单词被认为是关键字，如果在关键词后有“<SPACE>?”(例如, “mem ?”),

那么就会搜索 ini 文件以查找对此关键字的帮助部分。如果找到了，不向控制器发送什么，只是在输出数据窗口中显示。

如果控制器没有识别出指令输入的第一个词（关键字），那么没有找到关键词的响应，在结果窗口中显示未找到关键词。

在 PLC 浏览器输入命令时使用宏指令

如果在命令行输入与宏结合的指令，在指令发送到控制器前就把宏展开。在结果窗口的响应以同样的展开形式显示。

输入语句：<KEYWORD><macro>

<KEYWORD> is 指令。

<macro>是宏，宏有：

%P<NAME>	如果 NAME 是 POU-名，那么把此表达式展开程到<POU-Index>，否则没有变更
%V<NAME>	如果 NAME 是变量名，那么把此表达式展开成#<INDEX>:<OFFSET>，否则没有变更（由控制器解释把这种表示法#<INDEX>:<OFFSET>解释成内存地址）
%T<NAME>	如果 NAME 是变量名，那么把此表达式展开成<VARIABLENTYP>，否则没有变更
%S<NAME>	如果 NAME 是变量名，那么把此表达式展开成<SIZEOF(VAR)>，否则没有变更

如果换码符\（反斜杠）放在前方，那么符号%就被忽视。如果换码符写成\\，那么只是传送此符号。

例如：

在命令行输入：（variable .testit 的内存映像？）

mem %V.testit

在结果窗口输出：

mem #4:52

03BAAA24 00 00 00 00 CD CD CD CD .... ÍÍÍÍ

PLC 浏览器的更多选项

在'附加' 菜单或在 PLC 浏览器工具栏中，有下列指令用来处理命令输入或历史记录列表：

用'向前一个历史记录'和 '向后一个历史记录'，你可以向后和向前滚动来查询已完成的结果。

历史记录一直继续到你离开工程。

用'取消命令',可以中断已经开始的查询。

用'保存历史记录'，可以把已完成的查询结果保存到外部文本文件中。出现对话框'另存

为'，在这里你可以输入带有扩展名".bhl"的文件名（浏览器历史记录列表）。指令'打印上一次命令'(Print lastcommand)打开打印用标准对话框。可以打印当前查询和在信息窗内的输出数据。

针对运动控制及 IO 的诊断命令详细介绍请参看《CPAC 远程诊断功能 2.2_Ch》。

第八章 CPAC 控制器的加密机制

CPAC 支持两类加密形式：

1、软件加密：即应用程序开发人员在 OtoStudio 环境下开发应用程序过程中，设置密码保护程序。具体分为两种：

[1] 保护程序代码：在 OtoStudio->选项->密码：设置密码和保护密码。

[2] 保护程序运行：例如，在 OtoStudio 程序开发中，在程序中给出一个密码比较语句，作为是否运行程序的条件。

2、硬件加密：固高科技可以提供两种硬件加密方式：

[1] 在运行 GoogolRuntime 时候，自动检查硬件版本号，如果版本号不正确，则不能正常启动。（版本号由固高科技提供，同一系列的 CPAC 控制器，版本号是相同的）

[2] 提供绑定网卡地址的函数 GetMacAddress()，应用程序开发人员可通过读取网卡地址来加密应用程序。（每套硬件平台都有唯一的网卡地址，且不可更改）

回款加密：固高科技可以提供如下方案

在应用程序中设置定时器，读取 CPU 的时钟，例如，希望客户在 3 个月内付款，否则应用程序无法工作。

附 录

附录 1 关于数据类型

编程时可以使用标准和自定义的数据类型。每个数据类型有一个标示符，它指定占用的存储空间和数据值类型。

标准数据类型

BOOL

类型	最小值	最大值	存储空间
BOOL	FALSE/0	TRUE/1	8 Bit

BYTE

类型	最小值	最大值	存储空间
BYTE	0	255	8 Bit

WORD

类型	最小值	最大值	存储空间
WORD	0	65535	16 Bit

DWORD

类型	最小值	最大值	存储空间
DWORD	0	4294967295	32 Bit

SINT

类型	最小值	最大值	存储空间
SINT	-128	127	8 Bit

USINT

类型	最小值	最大值	存储空间
USINT	0	255	8 Bit

INT

类型	最小值	最大值	存储空间
INT,	-32768	32767	16 Bit

UINT

类型	最小值	最大值	存储空间
UINT	0	65535	16 Bit

DINT

类型	最小值	最大值	存储空间
DINT	-2147483648	2147483647	32 Bit

UDINT

类型	最小值	最大值	存储空间
UDINT	0	4294967295	32 Bit

当范围较大的类型转换为较小的类型时，可能丢失信息。

REAL

REAL 是浮点数。用他们显示有理数。REAL 是 32 位

LREAL

LREAL 是浮点数。用他们显示有理数。LREAL 是 64 位。

STRING

STRING 类型可包含任意多个字符。声明部分在圆括号里指定了字符数量。如果不说明大小，缺省大小是 80。

如：35 个字符的字符串声明：

Str:STRING(35):='This is a string';

TIME

数据类型 TIME,TIME_OF_DAY(TOD),DATA 和 DATA_AND_TIME(DT)象 DWORD 一样处理。在 TIME 和 TOD 以毫秒给出时间，TOD 时间从中午 12：00 开始。DATA 和 DT 类型从 1970 年 1 月 1 日开始，以秒给出。

自定义数据类型

ARRAY

一维、二维和三维数组作为基本数据类型。在程序的声明部分和全局变量表都可以定义数组。语法：<Field_Name>:ARRAY[<ll1>...<ul1>,<ll2>...<ul2>]OF<elem.Type>.

Ll1,ll2,ll3 标识字段范围的最小值；ul1,ul2 和 ul3 标识最大值。范围必须是整形的。

比如：Card_game:ARRAY[1...13,1...4]OF INT;

初始化数组：

可以初始化中所有元素或者不初始化

例如：arr1:ARRAY[1...5]OF INT:=1,2,3,4,5;

arr2:ARRAY[1..2,2..3,3..4]OF INT:=2(0),4(4),2,3;(*即 0,0,4,4,4,4,2,3*)

访问两维数组的数组元素：

<Field_Name>[Index1,Index2]

比如:Card_game[9,2]

注释：使用 CheckBounds 名在工程中定义函数，可以使用它检查范围溢出。

POINTER

程序运行时，变和功能量地址块地址以指针保存。

指针声明语法格式：

<Identifier>:POINTER TO<Datatype/函数 block>;指针可以指向数据类型或者功能块和用户自定义类型。地址运算符 ADR 用于把变量或者功能块的地址赋给指针。在指针后面加取内容运算符“^”解除指针参考。

例子：

pt:POINTER TO INT;

Var_int1:INT:=5;

Var_int2:INT;

Pt:=ADR(var_int1);

Var_int2:=pt^;(*Var_int2 是 5*)

ENUM

枚举是自定义数据类型。它由许多字符常量组成。把这些常量称作枚举值，即使枚举值是在程序的内部局部定义的，整个工程中都可以识别它，最好在对象管理器  数据类型下创建枚举对象。它以关键字 TYPE 开始 END_TYPE 结束。

语法：

TYPE<Identifier>:(<Enum_0>,<Enum_1>,...,<Enum_n>);

END_TYPE

变量可以使用枚举值之一并且初始化为第一个值。这些值是数字量的，可以象使用 INT 类型一样使用枚举值。可以赋给变量数值 X。如果枚举值没有初始化，从 0 开始记数。当初始化时，保证初始化值是不断增加的。运行时可以查看数值有效性。

比如：

```
TYPE TRAFFIC_SIGNAL:(Red,Yellow,Green:=10);(*Red 初始值 0， Yellow 1,Green 10*)
END_TYPE
TRAFFIC_SIGNAL1:TRAFFIC_SIGNAL;
TRAFFIC_SIGNAL1:=0；(*交通信号为 Red*)
FOR i:=Red TO Green DO
i:=i+1;
END_FOR;
```

不能多次使用相同的枚举值，如：

```
TYPE TRAFFIC_SIGNAL:(Red,Yellow,Green）;
COLOR:(Blue,White,Red);
```

错误：red may not be used for both TYPE TRAFFIC_SIGNAL and COLOR.

STRUCT

结构位于对象管理器  数据类型下面。如下：

TYPE<Structurename>:

STRUCT

<Declaration of Variables 1>

……

……

……

<Declaration of Variables n>

END_STRUCT

结构名可以在整个工程中识别，可作为标准数据类型使用。

可以使用互锁结构。唯一的限制是变量不能放在地址上（不允许 AT 声明）

例子：

TYPE Polyg 在线

STRUCT

Start:ARRAY[1..2]OF INT;

Point1:ARRAY[1..2]OF INT;

Point2:ARRAY[1..2]OF INT;

Point3:ARRAY[1..2]OF INT;

Point4:ARRAY[1..2]OF INT;

END:ARRAY[1..2]OF INT;

END_STRUCT

END_TYPE

结构初始化

Poly_1:polyg 在线:=(Start:=3,3,Point1:=5,2,Point2:=7,3,Point3:=8,5,Point4:=5,7,End:=3,5);

访问结构元素的语法：

<Structure_Name>.<Componentname>

比如：Week 是结构，Monday 是它的一个元素。可以这样调用：

Week.Monday

控制器寄存器地址

地址类型：

%I

Input

大小：

X：bit　　W：word　　B：byte　　D：dword

%IX0.0...IX0.15=%IW0，

%IW0=%IB1+%IB2，

%Q

Output

X：bit　　W：word　　B：byte　　D：dword

%QX0.0...QX0.15=%QW0，

%QW0=%QB1+%QB2，

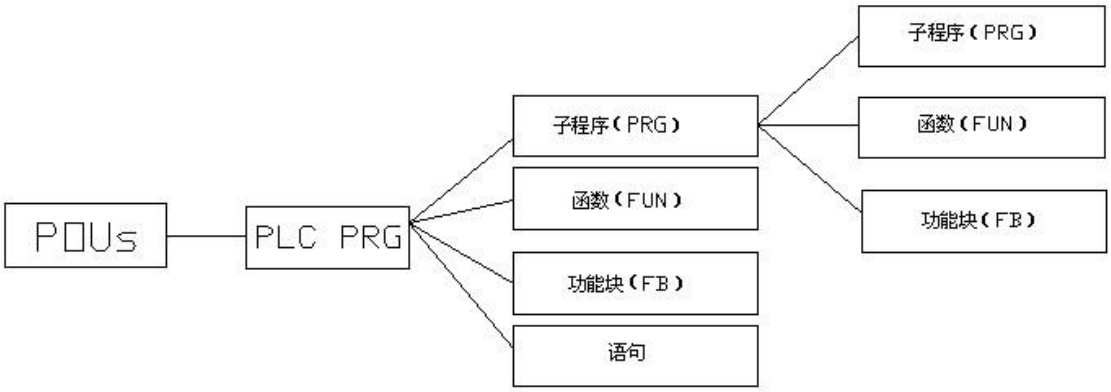
%M

Marker

%MW0-%MW247为248个应用参数。

程序结构

一个工程文件包含PLC程序里的所有对象：POUS(program organization units)、数据类型、可视化界面、资源。POUs包括主程序（PRG）、子程序（PRG）、功能块（FB）、函数（FUN）及语句。

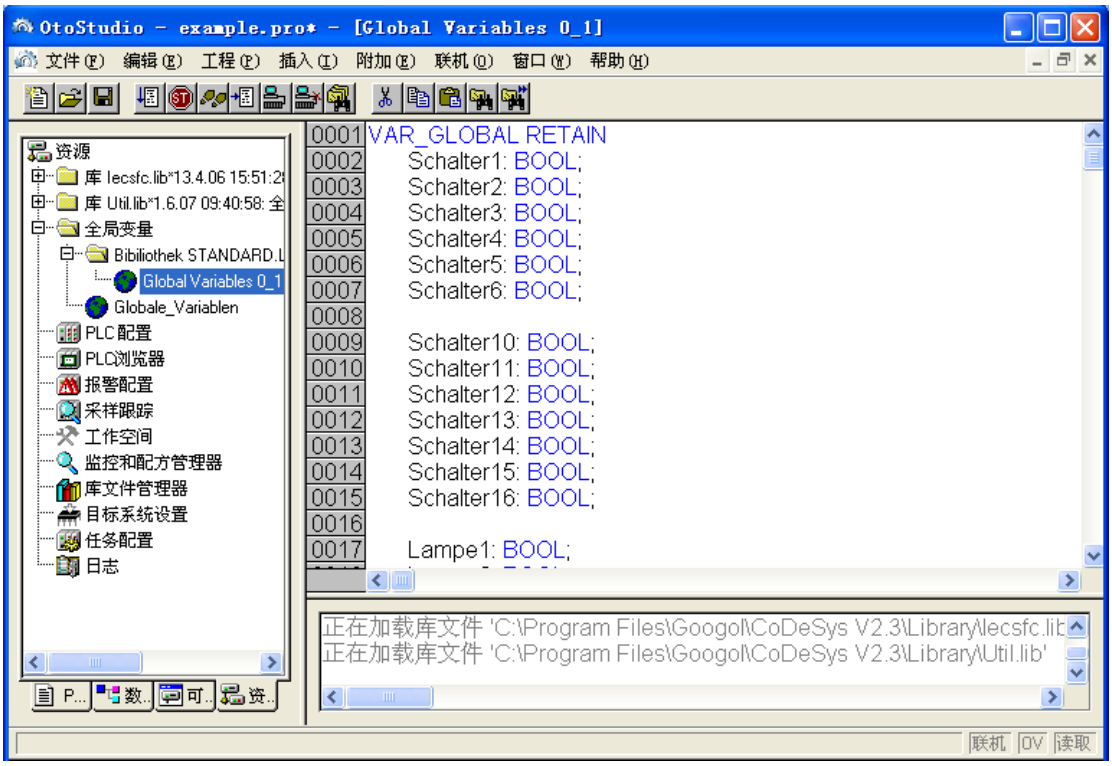


※主程序必须命名为 PLC_PRG。

※子程序可调用函数和功能块，但函数，功能块不能调用子程序，且子程序中的中间变量值是可视的，但函数，功能块里的中间变量值是不可视的，且函数没有返回值。

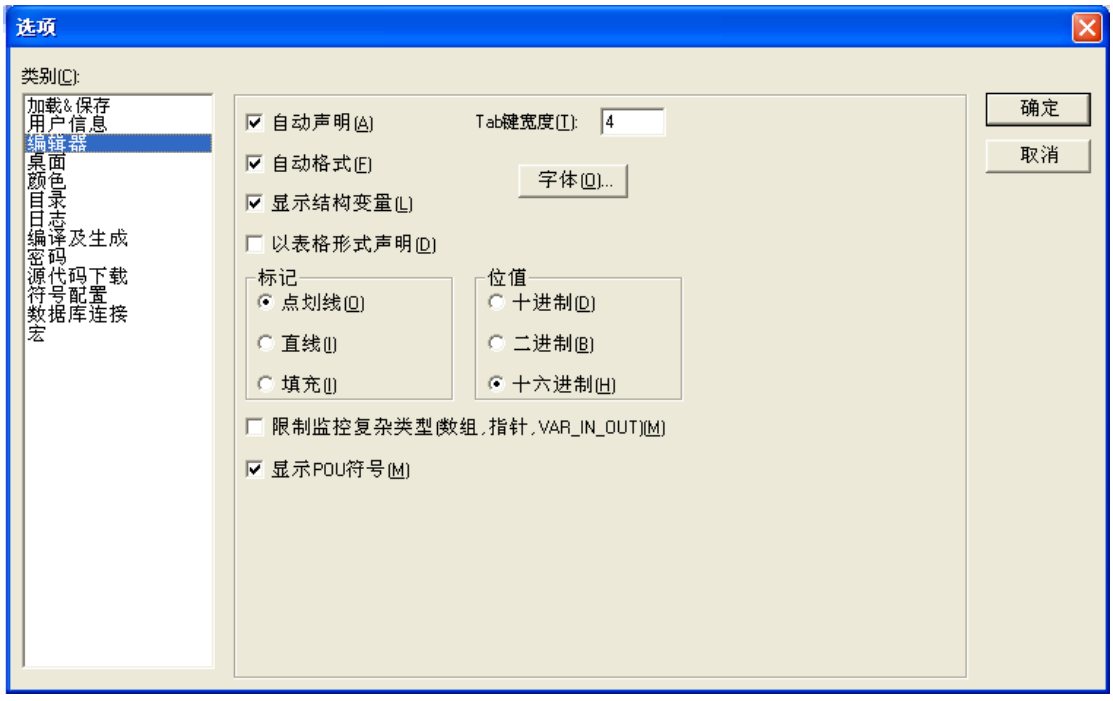
变量说明

在工程文件中，按适用范围有两种类型的变量，全局变量(Global)、局部变量(local)。全局变量存在于程序的任何模区域，而局部变量只存在于子程序，函数和功能块中。全局变量的说明在“资源”的 “全局变量”里：



局部变量的说明在“程序体”上部的局部变量说明区。

变量说明有两种方式：一种是在变量区进行说明；另一种是自动说明。自动说明是在主菜单里选择“工程”，“选项”，“编辑器”，出现以下对话框：



选中“自动声明”。这样，当编写程序，写到新的变量时，自动弹出对话框：

输入要定义的变量类型、地址、初始值。局部变量不用指定地址。

附录 2 键盘快捷键的使用

一般功能

在 POU 的声明部分和命令部分切换	<F6>
在对象管理器，对象和消息窗口间切换	<Alt>+<F6>
上下文菜单	<Shift>+<F10>
声明的快捷方式	<Ctrl>+<Enter>
从消息窗口的一个消息切回到编辑区的最初位置	<Enter>
切换到下一个打开的编辑窗口	<Ctrl>+<F6>
切换到先前打开的编辑窗口	<Ctrl>+<Shift>+<F6>
打开或关闭多层的变量	<Enter>
打开或关闭文件夹	<Enter>
在对话框中移到下一个区域	<Tab>
关于上下文的帮助	<F1>

一般命令

‘文件’‘保存’	<Ctrl>+<S>
‘文件’‘打印’	<Ctrl>+<P>
‘文件’‘退出’	<Alt>+<F4>
‘工程’‘检查’	<Ctrl>+<F11>
‘工程’‘生成’	<Shift>+<F11>
‘工程’‘全部重新生成’	<F11>
‘工程’‘删除对象’	<Del>
‘工程’‘增加对象’	<Ins>
‘工程’‘重命名对象’	<Spacebar>
‘工程’‘打开对象’	<Enter>
‘编辑’‘撤销’	<Ctrl>+<Z>
‘编辑’‘重复’	<Ctrl>+<Y>
‘编辑’‘剪切’	<Shift>+<Del>
‘编辑’‘复制’	<Ctrl>+<C>
‘编辑’‘粘贴’	<Ctrl>+<V>
‘编辑’‘删除’	<Del>
‘编辑’‘查找下一个’	<F3>
‘编辑’‘输入助手’	<F2>
‘编辑’‘自动声明’	<Shift>+<F2>
‘编辑’‘下一个错误’	<F4>
‘编辑’‘前一个错误’	<Shift>+<F4>
‘联机’‘登录’	<Alt>+<F8>
‘联机’‘退出’	<Ctrl>+<F8>

‘联机’‘运行’	<F5>
‘联机’‘设置断点’	<F9>
‘联机’‘单步跳过’	<F10>
‘联机’‘单步进入’	<F8>
‘联机’‘单个循环’	<Ctrl>+<F5>
‘联机’‘设置新值’	<Ctrl>+<F7>
‘联机’‘强制新值’	<F7>
‘联机’‘解除强制’	<Shift>+<F7>
‘联机’‘设置/强制对话框’	<Shift>+<F7>
‘窗口’‘信息’	<Shift>+<Esc>

FBD 编辑器命令

‘插入’‘网络（插入在当前行后）’	<Shift>+<T>
‘插入’‘赋值’	<Ctrl>+<A>
‘插入’‘跳转’	<Ctrl>+<L>
‘插入’‘返回’	<Ctrl>+<R>
‘插入’‘操作符’	<Ctrl>+<O>
‘插入’‘功能’	<Ctrl>+<F>
‘插入’‘功能块’	<Ctrl>+<B>
‘插入’‘输入’	<Ctrl>+<I>
‘附加’‘取反’	<Ctrl>+<N>
‘附加’‘转换’	<Alt>+<Enter>

CFC 编辑器命令

‘插入’‘POU’	<Ctrl>+<B>
‘插入’‘输入’	<Ctrl>+<E>
‘插入’‘输出’	<Ctrl>+<A>
‘插入’‘跳转’	<Ctrl>+<G>
‘插入’‘标号’	<Ctrl>+<L>
‘插入’‘返回’	<Ctrl>+<R>
‘插入’‘注释’	<Ctrl>+<K>
‘插入’‘POU 输入’	<Ctrl>+<U>
‘附加’‘取反’	<Ctrl>+<N>
‘附加’‘置位/复位’	<Ctrl>+<T>
‘附加’‘联接标记’	<Ctrl>+<M>
‘附加’‘EN/ENO’	<Ctrl>+<E>
‘附加’‘转换’	<Alt>+<Enter>

LD 编辑器命令

‘插入’‘网络（插入在当前行后）’	<Shift>+<T>
‘插入’‘常开接点’	<Ctrl>+<K>
‘插入’‘关联常开接点’	<Ctrl>+<R>
‘插入’‘功能块’	<Ctrl>+<B>
‘插入’‘线圈’	<Ctrl>+<L>
‘附加’‘粘贴在下面’	<Ctrl>+<U>
‘附加’‘取反’	<Ctrl>+<N>
‘附加’‘转换’	<Alt>+<Enter>

SFC 编程器命令

‘插入’‘步—转换（插入在当前行前）’	<Ctrl>+<T>
‘插入’‘步—转换（插入在当前行后）’	<Ctrl>+<E>
‘插入’‘选择分支（插入在右边）’	<Ctrl>+<A>
‘插入’‘选择分支（插入在左边）’	<Ctrl>+<L>
‘插入’‘跳转’	<Ctrl>+<U>
‘附加’‘缩放动作/转换’	<Alt>+<Enter>
从 SFC 总图返回编辑器	<Enter>

关于 PLC 任务配置的操作

打开和关闭结构成员	<Enter>
在名称附加放置一个编辑控制框	<Spacebar>
‘附加’‘登录编辑’	<Enter>

关于参数管理编辑器的操作

连接导航窗口和列表编辑器	<F6>
在列表编辑框下删除一行	<Ctrl>+<Del>
	<Shift>+<Del>
在列表编辑框下删除一个区域	<Del>

附录 3 SFC 标志符

SFC 程序组织单元标志符用来控制操作，它在工程运行期间隐含的创建，为了能读这些标志符，你必须定义合适的全局变量或局部变量。例如，如果在一个 SFC 程序组织单元中一个步激活的时间超过了它定义的属性，那么就会设置一个标志符，通过用一个“SFCError”变量可以访问到这个标志符（此时 SFCError 得到真值）。
可以定义下列标志符变量：

SFCEnableLimit

这个变量的类型是布尔型，当它的值为 TRUE 时，这一步的超时将会注册进 SFCError，其它的超时将被忽略。

SFCInit

当这个布尔变量值为 TRUE 时，顺序功能图复位到初始状态，其它的 SFC 标志符也会被复位。初
始步保持激活，直到变量值为 TRUE 时，才开始执行。只有当 SFCInit 被重新设置为 FALSE 时，模块才能正常
工作。

SFCReset

这个布尔变量，与 SFCInit 很相似，不同之处在于，进一步的处理发生在步的初始化之后，因
而，例如 SFCReset 标志符可以在初始化步中被复位到 FALSE。
注意:从 2.3.7.0 版编译器开始，SFCReset 可以用于复位与 IEC 步相关联的布尔型动作。

SFCQuitError

当这个布尔变量得到 TRUE 时，SFC 的执行将会停止，因此，在变量 SFCError 中一个可能超时将复位，当这个变量呈现 FALSE 时，激活步中的所有时间都会复位，先决条件是在 SFC 中已经定义过登记任何超时设定的标志符 SFCError。

SFCPause

当这个布尔变量值为 TRUE 时，SFC 图表的执行就会停止。

SFCError

当在 SFC 图表中有超时时这个变量得到 TRUE。如果在这个之后还有其它的超时发生，除非是这个变量已经复位，否则，这些状态将不会登记。如果你想使用其它的时间控制标志符(SFCErrorStep,SFCErrorPOU, SFCQuitError,SFCErrorAnalyzation)的前提条件是定义

SFCErrorSFCTrans

当一个转换被驱动时，这个布尔变量得到真值。

SFCErrorStep

这个变量是一个字符变量，如果 SFCError 中登记了一个超时，这个变量将存储这个超时步的名字。前提条件是在 SFC 中已定义了登记任何超时的标志符 SFCError。

SFCErrorPOU

这个字符变量包含了发生超时的模块名字。前提条件是在 SFC 中已定义了登记任何超时的标志符 SFCError。

SFCCurrentStep

这个字符变量存储了那个被激活步的名字，它与的时间监控无关。在仿真的情况下，此步存储在外部适当的分支种。如果一个超时发生其它的将不再登记，而且 SFCError 也不会复位。

SFCErrorAnalyzationTable

这是数组型变量，它提供一个转换表达式的分析结果，表达式中对转换中的
FALSE 和上一步起时有影响的每个一元素。要把下列信息写进 ExpressionResult 结构中写进：名称，地址，注释，当前值。
最大可以容纳 16 个元素，因此，数组的范围从（0-15）
ExpressionResult 结构和隐含使用的分析模块都是由 AnalyzationNew.lib 库文件提供的，分析模块也能够被其它的不用 SFC 编写的程序组织单元显式使用。
上一步的超时登记是分析一个转换表达式的先决条件，因此在这里必须有一个时间监视的应用，而且，SFCError 必须在声明窗口被定义。

SFCTip

这个布尔变量允许 SFC 的渐进模式。当用在 SFCTipMode=TRUE 来切换它时。如果 SFCTip 设置值为 TRUE 时它只可能跳到下一个步，只要 SFCTipMode 是设置为 FALSE 时，它可能跳过转换。
注意:对于扫描的状况和分步运动时间 隐藏变量还是可用的。

SFCTipMode

见 SFCTip 说明

附录 4

IEC 数据结构与 C 语言类型对比
BYTE unsigned char
WORD unsigned short
DWORD unsigned long
BOOL char
INT short
UINT unsigned short
DINT long
REAL float
LREAL double

CPAC-OtoStudio 可视化界面开发手册

第一章 概述

1.1 可视化编辑器概述

可视化是一种图形化表达法，它允许工程项目变量通过鼠标和键盘的方式输入在线运行的 PLC 程序中。OtoStudio 的可视化编辑器，是 OtoStudio 编程系统的一部分，它提供了一种图形化的元素，这种元素可以和工程项目中的变量链接并能按照需要进行动作。因此在运行模式下，图形化元素会根据变量值的改变而变化。简单的例子：为表现液位值在 PLC 程序中计算出的结果，可以绘制条状图并将其链接到相应的变量上，这样条状图形的长度和颜色将会显示出液位的当前值。增加文本区域，就能以字符串形式显示当前值，利用按钮可以控制程序的开始和停止。

单个可视化元素和整个的可视化对象的属性都可以在组态对话框和属性对话框中进行定义，其中还能通过激活选项的方式设置基本的参数以及通过输入对象变量的方式定义动态的参数。

此外，可以在可编程的元素属性中通过结构变量来给出特殊的功能配置。如果要多次使用同一个可视化对象来显示程序运行中的不同结果，利用组态对话框中的占位符（占位符）功能可以实现这样的功能。在程序联机运行的状态下，编程环境中创建的可视化界面可以作为唯一可用的用户接口，来控制及观察相关的 PLC 程序的运行，基于这种目的，程序中的变量只能通过激活可视化界面中的元素来输入。要实现上述功能，可以在可视化属性配置中选择所需的的输入方法，并且对于每个特殊的可视化变量可以选择特殊的热键进行定义。

OtoStudio 中创建的可是化界面可以在下述情况下使用：

1、 OtoStudio-可视化，这是在内嵌于 OtoStudio 编程环境的可视化，在离线模式下可编辑，在联机模式下可操作。

2、 Web-可视化，基于网页的可视化，可以通过互联网操作和调用（常用于远程维护）。

3、 目标可视化，目标可视化，可以直接在 PLC 中运行。

务必将此手册交给用户

- 非常感谢您选购 CPAC 控制器
- 在您使用之前，请仔细阅读此手册，确保正确使用。
- 请将此手册妥善保存，以备随时查阅。

1.2 创建一个新的可视化

可视化对象可以在对象管理器中的‘可视化界面’中进行管理，它包含可视化元件的管理并且对不同的对象可以根据个人需要进行管理。一个 OtoStudio 工程文件中可以包含一个或多个可视化对象，并且相互之间可以连接。

要想在对象管理器中创建一个可视化对象，必须先从程序界面左下角点击  进入“可视化界面”区域，在下拉菜单中按以下路径选择‘工程\对象\添加’命令，或者直接在“可视化界面”中点击鼠标右键选择‘添加对象’，都能创建一个新的可视化界面，并且可以在弹出的‘新建可视化界面’对话框中为创建的界面命名。当输入的名称中不含非法字符并且无重名时，即可按‘确认’按钮关闭对话框，此时一个新的界面窗口将被打开，在其中即可编辑新的可视化对象。在创建的可视化对象中可以通过选择下拉菜单上的‘工程’‘对象’‘属性’选项，可以打开‘属性’对话框，在那里可以设置对象的用法以及是否作为主界面。

在定义可视化界面对象的名称时，请注意以下几点：

1.如果没有明确的可视化界面被配置，自动默认命名为'**PLC_VISU**'的可视化界面作为目标可视化，Web 可视化或 OtoStudio 可视化的启动首界面。

2。在工程中可视化界面的名称不能与其它对象重名，因为当切换可视化界面时可能会出现错误。

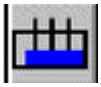
 注意	**如果要使用系统变量 CurrentVisu(字符串类型)来直接选择当前打开的可视化对象，必须始终使用大写字母来命名界面（如 PLC_VISU）**

第二章 OtoStudio 中的可视化编辑器

2.1 可视化元件的调用

插入可视化元件

可视化元件是一种图形化元素，用来构建可视化对象。所有可用的元素都可以在 OtoStudio 的菜单栏中找到，每种都有独立的属性配置文件。用户可以在新建的可视化对象中随意插入各种几何图形、位图、图元文件、按钮，以及已有的可视化组件。几何图形包括矩形、圆角矩形、椭圆/圆、多边形等。

可以在下拉菜单的‘插入’选项中自由选择以下命令：矩形  、圆角矩形  、椭圆/圆  、多边形  ，多边线  ，曲线  ，饼图  ，位图  ，按钮  ，表格  ，ActiveX 元素  ，仪表  ，条状图  ，折线图  ，报警列表  ，趋势图  ，图元文件  等。其功能如上表所示。编辑视图的快捷工具按钮位于视图编辑区域顶端标题栏的工具栏中，如图所示，

在编辑视图时，将鼠标移到视图编辑工具按钮处，会看到相关工具的自动提示。选中命令后，如果将鼠标置于编辑窗口，鼠标指针将会出现相应的图标（如选择  ，将出现  ），在编辑窗口选择所绘制图形的开始点的区域进行点击，并且按住鼠标左键拖动直至图形达到所要的尺寸。

插入长方形

命令图标：  ，用此图标在视图中插入长方形。

插入圆角长方形

命令图标：  ，用此图标在视图中插入圆角长方形。

插入椭圆形

命令图标：  ，用此图标在视图中插入椭圆形。

插入多边形

命令图标：  ，用此图标在视图中插入多边形。

插入多线段

命令图标：  ，用此图标在视图中插入多线段。

插入曲线

命令图标：  ，用此图标在视图中插入贝塞尔曲线。

插入饼图

命令图标：  ，用此命令在视图中插入饼图。

点击选中该命令后，按下鼠标左键并拖动可以生成所需尺寸的椭圆区域，区域中的位置包含一个显示半径的线段，只要按住左键通过拖曳鼠标就可以改变所绘图形的位置。

以下操作可以定义饼图的起始角度：在绘制好的椭圆区域上点击鼠标左键，将会在椭圆区域内出现三个黑点，圆心位置是一个带有白色十字的圆点，圆周和半径的交汇处是一个矩形黑点，指明构成角度的两边，两条构成 0° 的半径能够分别选中并绕中心拖动；在图形边上还有一个矩形黑点，指示出所绘制椭圆虚拟的外接矩形的一个角。如果想给定具体数值来定义起始角度，可以通过双击鼠标或者点击右键从菜单中选择“配置对话框”弹出自定义窗口，在‘角度’目录中输入想定义的变量名。通过拖动图形中心点的位置还能调整图形的尺寸和形状（鼠标左键置于图形中心后将出现两个斜交叉的箭头，此时按住鼠标左键并拖动，即可改变），同样，选中并移动图形边上的黑点也可以改变图形的尺寸和形状。如果想移动整个图形，则应将鼠标放在椭圆区域，当出现两个垂直交叉的箭头时，按下左键并拖动鼠标即可移动图形，抬起鼠标左键即完成图形的移动。

插入位图文件

命令图标：  ，用此命令在视图中插入位图（BMP，JPG。）

点击选中该命令后，按鼠标左键并拖动，可以形成一个带对角线的矩形区域，同时弹出对话框提示选择要填充的文件，当选择了要打开的文件后，图片将被填充在矩形区域内。通过双击鼠标或者点击右键从菜单中选择“配置”将会弹出自定义窗口，对图片进行多项处理。

插入已创建的其他可视化界面

命令图标：  ，用此命令可以将已创建的视图作为一个元素插入当前视图。

点击选中该命令后，按鼠标左键并拖动，形成一个带对角线的矩形区域，同时弹出对话框提示选择打开已存在的视图文件，选中后视图文件将插入已定义的区域。

插入按钮

命令图标： 

，用此命令在视图中插入按钮。

点击选中该命令后，在界面所要放置的区域按下鼠标左键并拖动即可生成所需尺寸的按钮。如果定义的是自锁的按钮，则按下按钮和弹起按钮将会以的不同显示表示出所定义变量的状态。如果是不自锁的按钮，则只有按着按钮不放才能锁定变量。

插入图元文件 WMF

命令图标： 

，用此命令在视图中插入图元文件。

点击选中该命令后，按鼠标左键并拖动，形成一个带对角线的矩形区域，同时弹出对话框提示选择图元文件（扩展名*。wmf）， 选择文件后将插入已定义的区域，需要注意的是没有可连接的文件插入后和插入位图的效果是一样的。

插入表格

命令图标：  ，用此命令在视图中插入表格，该命令常用于显示一组变量的当前值。

点击选中该命令后，按鼠标左键并拖动可以形成想要的尺寸区域，同时弹出对话框“配置表”，在其中可以定义表格的安全性、提示信息，以及行、列等属性。

插入仪表

命令图标：  ，用此命令在用户定义的视图中插入表控件。

该命令提供的仪表中带有能定义刻度的圆弧，以及能同变量关联的元素点，当按按下鼠标左键并拖动后，将形成一个选定大小的区域，在显示仪表前先会弹出一个属性对话框“配置仪表”，在其中可以定义表中要显示的变量参数并预览完成定义后的表格。

插入条状图

命令图标：  ，用此命令在用户定义的视图中插入条状图。

该命令能显示指定变量的值，并以柱（条）状图来表明其值的变化。当按按下鼠标左键并拖动后，将形成一个选定大小的区域，图形显示前先会弹出一个属性对话框'配置棒图显

示'，在其中可以定义表中要显示的变量参数，并在确认插入前预览完成定义后的图形。

插入柱状图

命令图标：  ，用此命令在视图中插入柱状图。它利用柱状图的长度表明变量的数值。

当按按下鼠标左键并拖动后，将形成一个选定大小的区域，图形显示前先会弹出一个属性对话框“配置柱状图”，在其中可以定义表中要显示的变量参数，并在确认插入前预览完成定义后的图形。

插入报警列表

命令图标：  ，用此命令在视图中插入报警列表。

使用该命令时，当按下鼠标左键并拖动后，将形成一个选定大小的区域，图形显示前先会弹出一个属性对话框“'配置报警表”，在其中有包括“报警列表”、“报警种类设置”、“列(栏)”等显示项目在内的多个选项，还有提示信息、安全级别设置等一般选项，利用这些选项可以对报警列表的显示内容、显示格式等进行定义。

 注意	**该处的报警列表仅仅对用于对报警的显示，具体的设置及定义在“工程”菜单的“'配置报警”选项中。**

插入趋势图

命令图标：  ，用此命令在视图中插入趋势图。

使用该命令时，当按下鼠标左键并拖动后，将形成一个选定大小的区域，带有“定义坐标轴”、“变量”、“历史记录”等项目的属性对话框“趋势”将会自动弹出。趋势图，又称示波器，通常用来观察变量在一个特定时间段的变化情况，它将数据存储在用户定义的文件中并以图形显示出来，当变量的值发生改变时，就会有新的数据输入存储的文件并在数据/时间轴上显示更新，通过设定用户需要的背景（图片，颜色等），可以使要观察的变量更加清晰、分明。

2.2 布置视图

选择可视对象

“选择方式”是鼠标默认的指令，用鼠标左键点击视图对象即可将其选中。

可以按下<Tab>键，选择视图对象中的第一个视图对象，再次按下<Tab>键则选择下一个视图对象。如果同时按下<Tab>和<Shift>键，可以按照视图对象中相反的顺序选择。在选择一个视图对象后按下<Shift>键，同时点击相应的视图对象，可以选择多个视图对象。也可以按住鼠标左键不放，在要选择的视图对象上拉一个窗口，同样可选择多个对象。

选择方式

如果“附加”菜单中的“选择方式”菜单前有“√”，或者快捷工具栏中表示鼠标状态的快捷工具按钮被按下时，表示此时处于“选择”状态，可以选择视图对象。否则处于绘图状态。

全选:使用“附加”菜单中的“全选”菜单可以将当前视图中的所有视图对象全部选中。

如果打开了元件列表(‘附加’‘元件列表’)，你可以选择其中某一行以便选择在可视化界面中的相应元件。

改变选择和插入模式

插入视图元件后，鼠标将自动改回选择模式，要想继续进行下一个视图元件的插入，可以从下拉插入菜单中选取命令，或者点选工具条上的“  ”图标，退出选择模式。在按下“Ctrl”键不放时，通过点击鼠标右键可以在选择模式和插入模式间进行切换。

‘附加’‘选择’

从下拉插入菜单中选取命令，或者点选工具条上的“  ”图标，退出选择模式。在按下“Ctrl”键不放时，通过点击鼠标右键可以在选择模式和插入模式间进行切换。

‘附加’‘全选’

使用这个命令可以在当前的可视化对象中全部选择可视化元件。

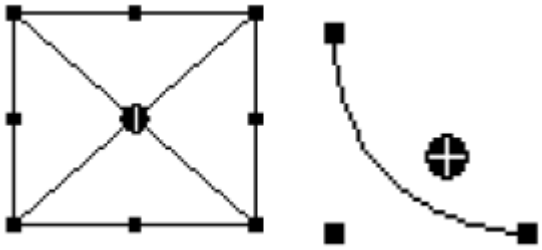
复制可视化元件

使用“编辑”/“复制”命令，或者<Ctrl>+<C>组合键，可以复制所选择的一个或多个视图对象。复制视图的另外一种方法是选择要复制的视图对象，按下<Ctrl>键的同时点击该视图对象，则会在原来的视图对象上产生复制的视图对象。

修改可视化元件

用鼠标点击对象或者按下<Tab>键，可以选择一个视图对象。在所选择视图对象的周围有一些小的黑色矩形。通过点击这些小的黑色矩形，按住鼠标左键，可以改变视图对象的大小，控制视图对象的轮廓。选择一个视图对象后，同时显示旋转点。旋转点是一个中间带有

白色十字的黑色圆圈（ ），可以按一定的角度使对象绕这点旋转。可以按下鼠标左键拖动旋转点。对于多边形，使用同样的方法拖动每个顶角。鼠标拖动时，按下<Ctrl>键，可以插入一个顶角。按下<Shift>+<Ctrl>键，可以删除一个顶角。



拖动可视化元件

点击鼠标左键选择视图对象。在所选的视图对象上按下鼠标左键，或者按下方向键，可以拖动一个或多个视图对象。

组合可视化元件

选择多个视图对象，点击“附加”菜单中的“组合”菜单，可以将多个视图对象组合成一个视图对象。组合后视图对象的行为与一个视图对象的行为相同。

取消组合：选择一个组合的视图对象，点击“附加”菜单中的“取消组合”菜单，可以将该组合的视图对象分解为多个单独的视图对象。

‘附加’‘置于前面’

用这个命令可将可视化元件放到最前面显示。

‘附加’‘置于后面’

用这个命令可将可视化元件放到最后面显示。

‘附加’‘对齐’

使用这个命令可以排列已选择可视化元件。

以下的排列选择是可用的：

左：最左边对齐方式排列元件

右/上/下，与上面相同

水平中心：按元件水平中心对齐

垂直中心：按元件垂直中心对齐

‘附加’‘元件列表’

点击标题栏“其它”菜单中的“对象列表”菜单，打开视图对象列表对话框，如下图所示，其中包括视图对象的号码、类型和位置等信息。点击该对话框右侧的工具按钮，可以对其进行编辑。在绘图区点击鼠标右键，选中“对象列表”，也会弹出此对象列表。

“确认”按钮：当对对象列表内容操作完毕后，点击“确认”按钮，关闭对话框。

“置于最前面”按钮：把选择的视图对象放在最上层，此时元件号最大。

“置于最后面”按钮：把选择的对象放在最底层，此时元件号最小。

“向前一个”按钮：把选择的视图对象向上移动一层，此时元件号增一。

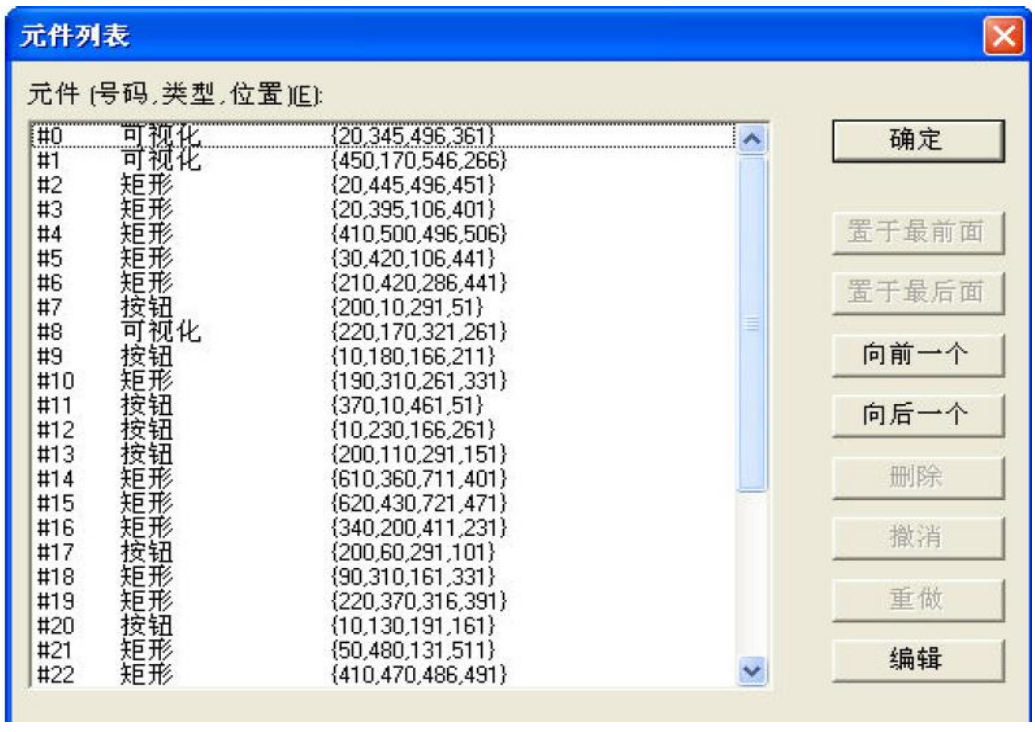
“向后一个”按钮：把选择的视图对象向下移动一层，此时元件号减一。

“删除”按钮：删除选择的对象

“取消”按钮：取消上一次操作

“重做”按钮：恢复上一次操作。

“编辑”按钮：对视图对象进行编辑操作。



可视化状态条

如果一个可视化有中心，相对于图像的左上角，鼠标指针在状态条上显示当前的 X 和 Y 坐标。如果鼠标锁定了一个元件，或者元件被处理，那么显示元件的编号。如果选择插入一个元件，然后这个元件也将出现（例如，长方形）。

2.3 配置一个可视化概述

对于可视化元件或是可视化对象的全部配置可以通过“附加”菜单的对话框来配置，可视化的附加设置可以在属性对话框中进行。

在工程选项中，有单独的目录用于保存可视化文件。

2.4 配置可视化元素

配置可视化元件

通过‘附加’‘配置’命令打开配置对话框。在该对话框中用激活选项或动态插入工程变量来设置一个元件或对象的属性。此外，可以通过定义各可视化元件结构变量的组件来编写属性。

有关在线模式中遵循的解释程序：

——通过工程变量动态给出的值重写同一属性设定的参数。

——如果一个元件属性通过“标准”工程变量以及通过一个结构变量的组件来定义时，那么在在线模式中，首先关注工程变量的值。

如果在 OtoStudio 中，要将可视化用作目标系统可视化或 web 可视化，即将可视化用作 PLC 程序的用户接口，那么 “占位符” 和“专用输入” 是很有用的。

对于含有配置可视化元件 颜色和字体的对话框，随着当时选定的目标系统的不同而有不同的外观，它可能不是标准的对话框，而是一个与项目相适应的选项对话框，它用于指定的目标可视化。

在可视化中的占位符

在配置对话框中，每个输入变量或文字的地方，都能放入一个占位符来替代相关的变量和文字。如果创建的一个可视化对象不直接用在程序中，而是作为一个“实例”插入到其它可视化对象中，那么占位符是有意义的。在配置这样的一个实例时，可以用变量名或文字来替代此占位符。(参看配置一个插入的可视化界面， 在那部分帮助中有占位符使用的例子)。

任何用两个“$”括起来的字符串都是一个有效的占位符（例$Variable$，Variable$X$ ）。在‘占位符列表’对话框（从‘附加’‘占位符列表’调用）中，对每个占位符，都可以把一个“值组”定义为一个输入明细。在配置一个可视化对象的实例时，你可以用这些值中的一个替代这个占位符。在实例中采用一个占位符列表来做这种替代。

占位符应用举例:

在相同可视化界面的实例帮助下，功能块实例很容易被显示。例如，在配置用于显示功能块变量的可视化界面中，每个变量以FUB开头(如FUB.a)，如果使用可视化界面实例(在另一个可视化界面中插入一个可视化界面或使用'转到'调用)，那么在这个实例的配置中，占位符FUB可以用功能块的实例名替代，以便显示变量值。

如下所示:

在工程中定义了包含下列声明:

```
FUNCTION_BLOCK fu
VAR_INPUT
 changecol : BOOL; (*  在可视化界面中改变颜色  *`)
```

END_VAR

在 PLC_PRG 定义了两个'fu'实例:

inst1_fu : fu;
inst2_fu : fu;

创建可视化对象'visu'，插入一个元件并打开配置对话框，打开'变量'选项，在字段‘改变颜色’中输入"FUB.changecol"。打开‘输入’选项，在字段‘触发但不保持变量值’中输入"FUB.changecol"。打开'文字'选项，输入"FUB - change color "

创建另一个可视化对象'visu1'。在'visu1'中插入两个'visu'(两次参考'visu')

选择'visu'的第一个参考，打开配置对话框中的'可视化'选项，点击'占位符'按钮后将显示占位符列表，用'PLC_PRG.inst_1'替换 'FUB'。

选择第二个'visu'实例，按上面的描述，用'PLC_PRG.inst_2'替换 'FUB'。

在联机模式下，用于配置'fu'实例的变量值将在相应的'visu'实例中显示结果。

当然，占位符FUB可以在'visu'配置中的所有可输入变量或文本的位置使用。

‘附加’‘占位符列表’

占位符列表如图所示。

占位符：列出用于配置所有视图对象的占位符。

元件号：显示包含占位符的对象号。

替换：可以输入一些字符串。例如文本、变量或表达式等。

在 OtoStudio 中这个列表用于管理和配置占位符:

在配置以后用于可视化对象实例的可视化界面时，使用此列表。由于这个原因，可以使用占位符替换配置对话框中的变量或字符串。可以通过菜单'附加‘’占位符列表'或上下文菜单

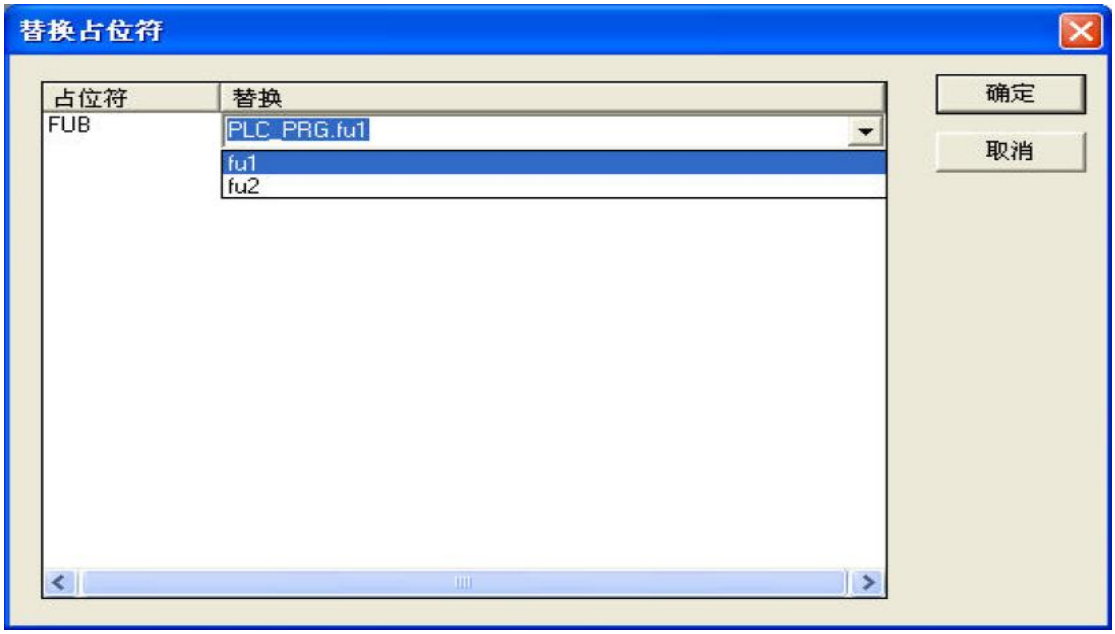
打开占位符对话框，在列表中有三栏。

占位符列表例子

占位符列出了所有在当前可视化界面配置中使用的占位符。元件号包含占位符的元件。在替换栏中，可以输入一个或多个字符串(文字，变量，表达式)，以便在配置可视化界面实例中替换占位符时选择它们。输入的可选元件必须用逗号分开。 如果没有设置或设置了一个不可能的替换字符串，在配置可视化界面实例时可以用期望的文字替换。

当配置上面提到的可视化界面实例时，可以使用占位符列表，即通过命令‘插入’‘可视化’在另一个可视化界面中插入这个对象后。为了实现这个，作如下操作，打开对话框，选择插入的可视化界面，执行命令‘附加’‘配置’然后按‘可视化’选项中按钮'占位符，在这种情况下，对话框只包含两栏：

在可视化实例中用于替换占位符的占位符列表



如上图所示，占位符栏中列出了所有在主可视化对象中定义的占位符。如果可能的替换元件已定义，可在替换栏中选择它们，选择一个替换当前实例中的占位符。如果没有预先定义，手动输入一个表达式或变量，也可以在替换栏的字段处双击打开编辑字段。

'附加' '配置'

用这个命令，可以打开‘配置元件’对话框，用来配置将选择的对话框，(参看选择可视化元件)，当你在这个元件上双击时，便弹出此对话框。在对话框的左边区域内选择一个类目（可用的类目取决于元素类型），并在右边区域内填写所需的信息。它通过激活选项插入定义元素属性值的有效变量名来完成。

注意	**还有供元件组使用的配置对话框。该设置仅对元素组有效。如果你想配置组内的某元件，你必须分解这个组。** **如果你已通过“静态”静态设定且通过一个动态变量来定义一个元件的属性，那么该变量在在线模式中改写静态值。（例如可以在类别“颜色”中静态定义“内部报警颜色”，又在类别“颜色变量”中用一个变量给一动态定义）。如果用一个“标准”项目变量和一个结构变量，同时进行设定，那么“标准”项目变量将改写结构变量值。** **对于仪表图、条形显示和柱形图，必须事先重组。**

在元件配置中操作变量的部位可以作下列输入：

•变量名，可采用输入助手来输入它。

•表达式，它是由分量存取，有常数索引的字段存取，变量和直接地址装配而成的。

•操作字和常数，需要时可与上述的表达式组合。

•用变量名或正文串替换占位符。

允许的表达式举例：

x + y

100*PLC_PRG.a

TRUE

NOT PLC_PRG.b

9*sin(x + 100)+cos(y+100)

不能调用函数。无效的表达式会在登陆时产生一个错误信息。无效表达式的例子：fun(88)，a:=9，RETURN。

在配置对话框中，写全局变量有两种可能的方法：“.globvar”和“globvar”是等同的。然而，有一个点的表达式不允许用于装配表达式中。

角度

角度 Angle 属性用来定义**饼图**对象的角度。双击饼图对象，弹出饼图对象属性配置对话框，如下图所示。点击角度 Angle 属性，在“开始角度”和“结束角度”文本框中分别输入饼图对象的开始角度和结束角度，则将会以顺时针方向画出所需要的饼图。选中“只显示弧段”，则饼图对象不显示夹角，只显示弧段。

举例：

声明变量

PROGRAM PLC_PRG

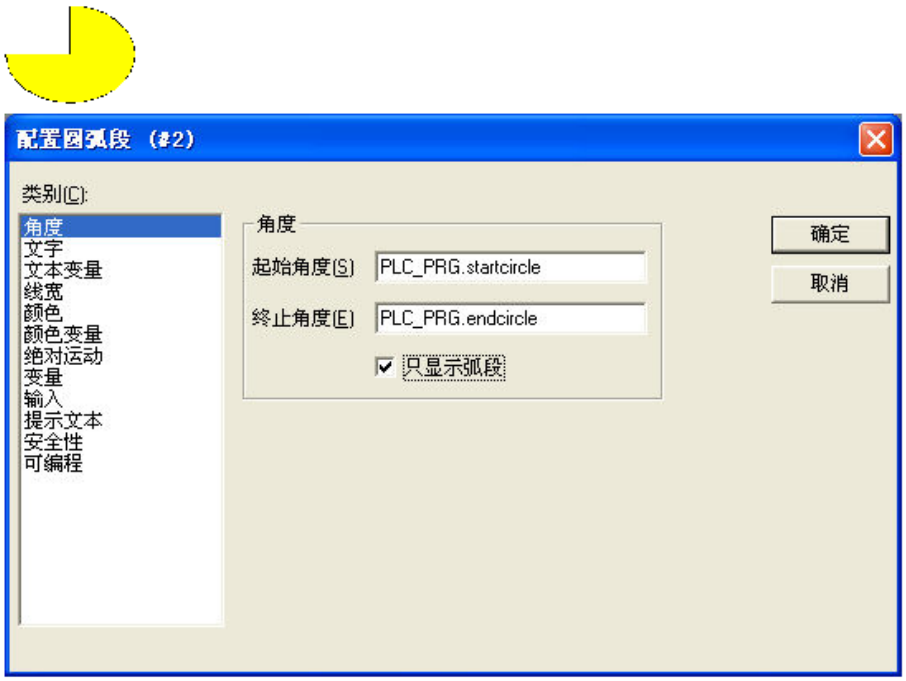
VAR

angle_start: REAL := 90;

angle_end: REAL := 180;

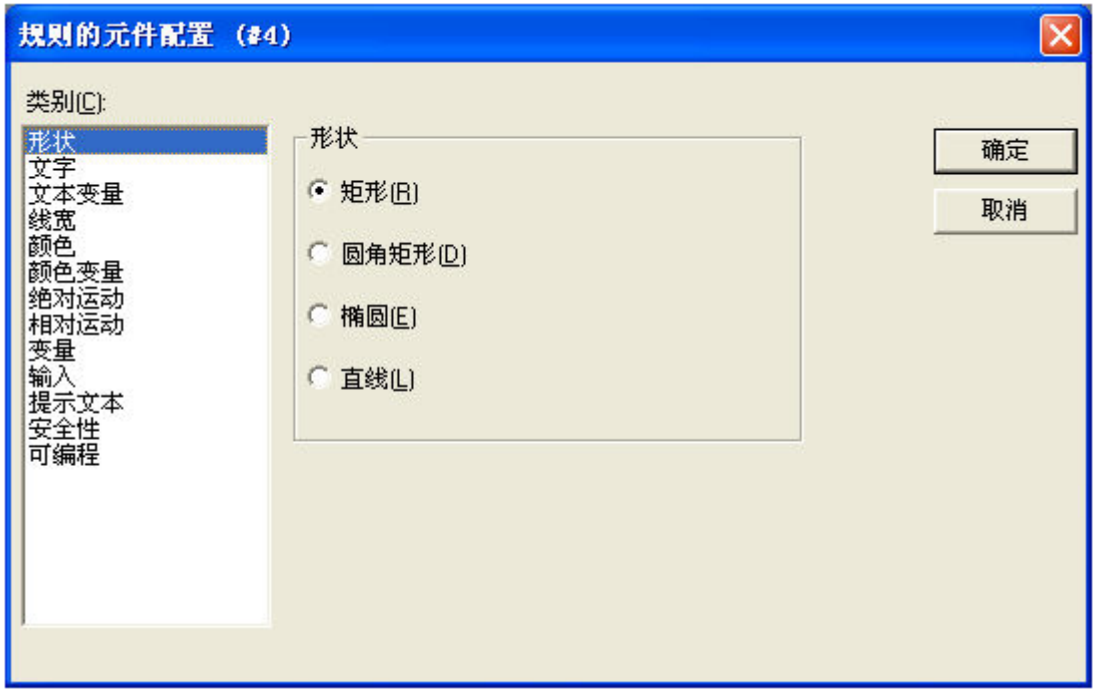
END_VAR

视图运行结果如图所示：



形状

形状 Shape 属性用来定义视图对象的形状。对于规则视图，形状 Shape 属性可以选择矩形、圆角矩形、椭圆或直线等。对于不规则视图，形状 Shape 属性可以选择多边形、折线或曲线等。形状 Shape 属性的改变只在所确定的范围内进行。



文字

在视图对象中可以添加文本，用文字属性来设置，输入变量时可以使用 F2功能键弹出快捷选项框。如图所示。

内容：在“内容”文本框中输入文本，按<Ctrl>+<Enter>组合键换行。

水平：在“水平”选项中设置文本在视图对象中的左、中、右位置。

垂直：在“垂直”选项中设置文本在视图对象中的上、中、下位置。

点击“字体”或“标准字体”按钮，可以设置文本的字体。

注：

◆ 如果在文字中包含“%s”，那么文字中的这个位置在程序运行后，将被变量的特定形式的值所代替，具体的含义参见下表

字符	含义
d,i	整型（十进制）
o	无符号的八进制数（0不能作首位）
x	无符号的十六进制数（0不能作首位）
u	无符号的十进制数
c	单字符
s	字符串
f	浮点型格式，在“f”前面以x.y的形式书写数字，前面的x表示整数部分显示的最少位数，后面的y表示小数部分的精确度（默认数字为6，即6位小数）。例如3.4f% 表示最大整数位是百位，精确到小数点后第四位。

2) 还可以在标准的 C 程序库中，使用特定的字符串，其含义如下表所示。

%a	星期缩写
%A	星期全写
%b	月缩写
%B	月全称
%c	日期时间
%d	一月中的天数（01-31）
%H	24 小时格式（00-23）
%I	12 小时格式（01-12）
%j	一年中的天数（001-366）
%m	月（01-12）
%M	分钟（00-59）
%p	12 小时A.M/P.M 格式
%S	秒（00-59）
%U	一年的第几个星期(00-53)，周日为星期第一天
%w	星期（0-6，周日是0）
%W	一年的第几个星期(00-53)，周一为星期第一天
%x	日期
%X	时间
%y	不含世纪的年（00-99）
%Y	含世纪的年
%z, %Z	时区名
%%	百分号

举例

1、 如果在文本内容中填入%2.5f mm ，则在程序运行时显示：32.8889 mm 。

2、 若填入“%t%a %b %d.%m.%y %H:%M:%S”，则运行时显示：Wed Aug 28.01.07 16:32:45。

3、 %Today is %d.%m.%y，则运行时显示： Today is 28.01.07

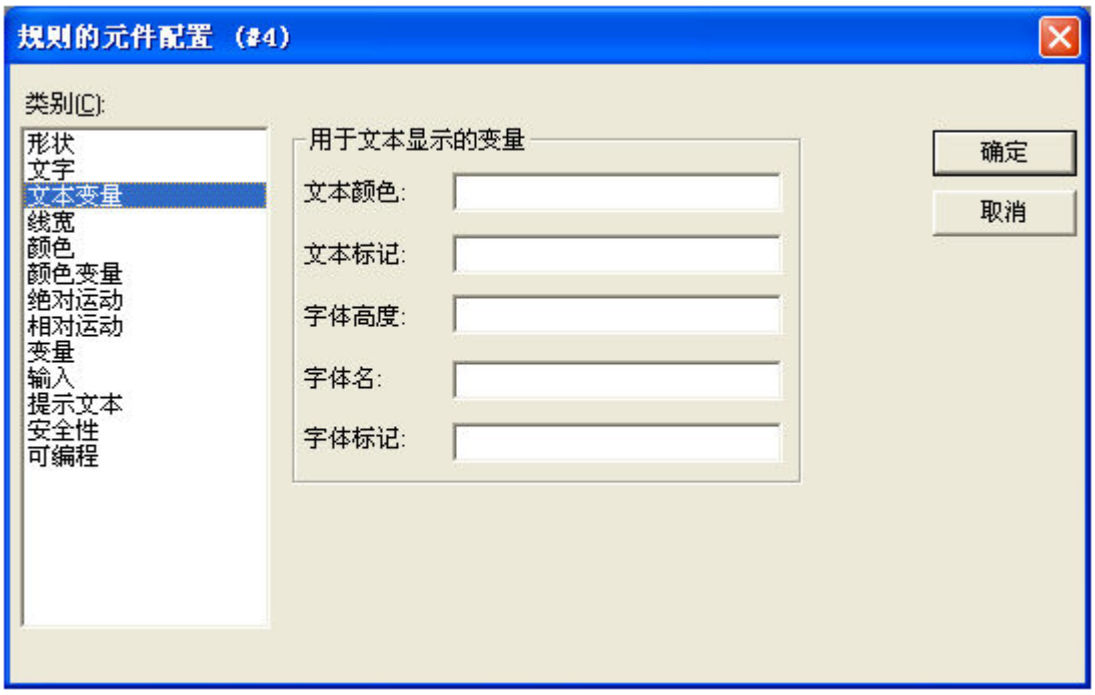
文本变量

在配置可视化界面对话框的‘**文本变量’**选项中可以定义一个变量，用于设置‘文本’选项中的文本的颜色和字体。最好借助输入助手(F2)输入变量名。

也可以使用结构 VisualObjectType 中的变量来设置文本属性。参看‘可编程性’选项部分的描述;在那儿可以找到特定结构变量的允许值和它们的作用。

注意	**如果在'文本'选项中有相应的静态设置，它们将被动态文本替换。**

在一个元件属性有多个定义的情况下，依据联机模式下数值可能被替换，要考虑优先的顺序关系。



对话框参数:

参数:	含义:	工程变量输入:	程序中变量的使用:	对应到结构VisualObjectType中的变量:
文字颜色:	文字颜色	"plc_prg.var_textcolor"	var_textcolor=16#FF00FF 颜色	dwTextColor
文本标记:	对齐 (右, 左, 中心...)	"plc_prg.textpos"	textpos:=2 字体右对齐	dwTextFlags
字体高度:	字体高度,以象素为单位	".fonth"	fonth:=16; 字体高度16pt	ntFontHeight
字体名:	字体名	"vis1.fontn"	fontn:=arial; 使用arial字体	stFontName
字体标记:	字体显示 (粗体, 下划线,斜体...)	"plc_prg.fontchar"	fontchar:=2 文字按粗体显示	dwFontFlags

线宽

线宽属性用来定义视图对象线条的宽度。下图所示为线宽设置对话框，共有5种线宽可以直接选择。选中“其它”选项可以输入线宽。在“线宽变量”中可以输入控制线宽的工程变量，使线宽产生动画效果，输入工程变量时可以使用 F2功能键。联机模式下，动态线宽属性会替换静态线宽属性。

线宽设置对话框

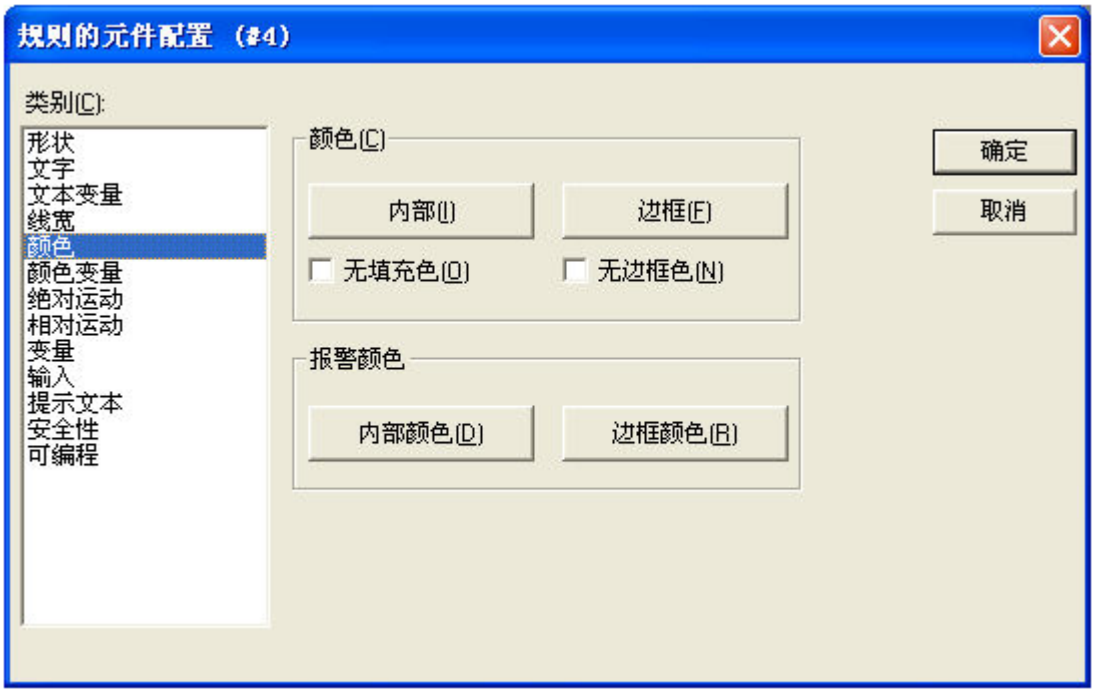
颜色

颜色属性用来定义视图元件的颜色和报警的颜色。选择'无填充色'和'无边框色'可以创建透明元件。

注意	**如果有相应的静态设置，它们将被动态的替换。**

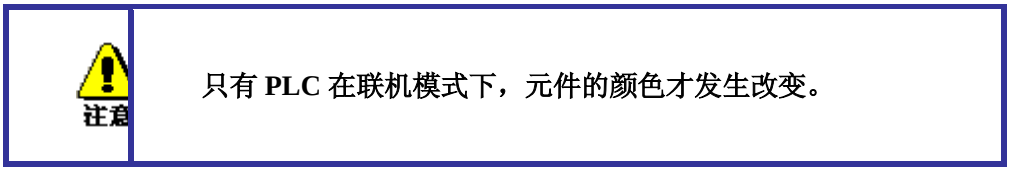
在一个元件属性有多个定义的情况下，依据联机模式下数值可能被替换，要考虑优先的顺序关系。

颜色设置对话框



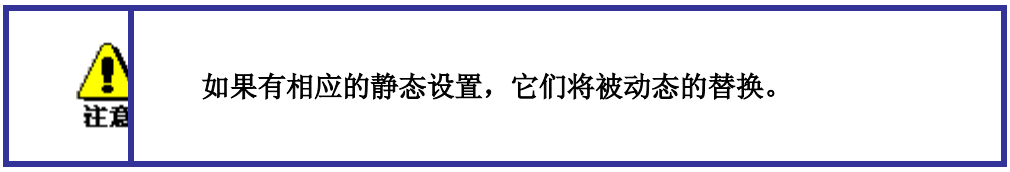
如果在'变量'选项的'改变颜色'字段输入一个布尔型变量，如果变量是'False'，元件按'颜色'设置中的颜色显示，如果是'True'，按'报警颜色'设置中的颜色显示。

按'边框'或'内部'按钮打开颜色选择对话框来设置颜色。

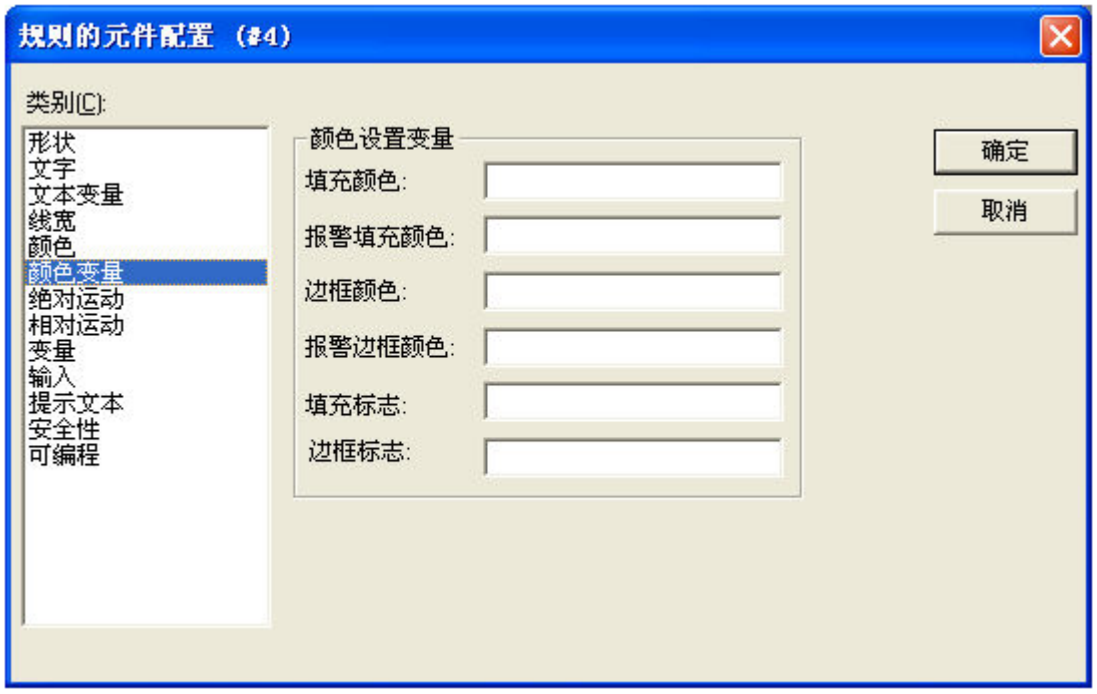


颜色变量

在这个对话框中输入工程变量(如:PLC_PRG.color_inside)，它可以确定在联机模式下元件的颜色属性。也可以使用结构 VisualObjectType 中的变量来设置颜色属性。参看 '可编程性'选项部分的描述; 在那儿可以找到特定结构变量的允许值和它们的作用。

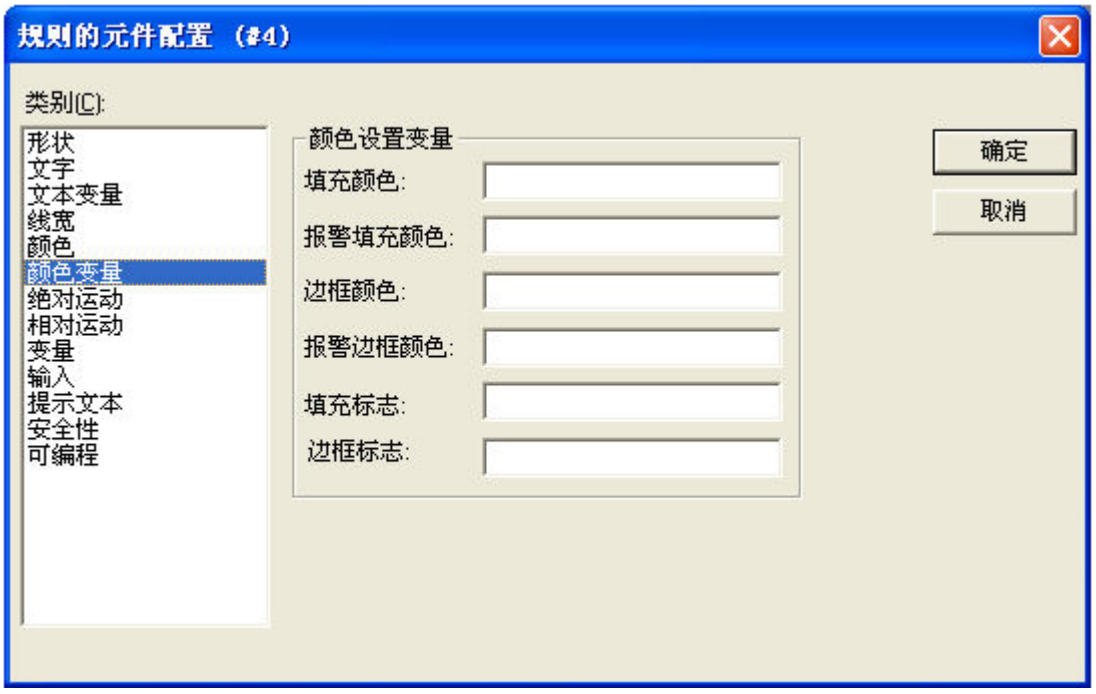


在一个元件属性有多个定义的情况下，依据联机模式下数值可能被替换，要考虑优先的顺序关系。



对话框参数:

参数:	含义:	工程变量输入:	程序中变量的使用:	对应到结构VisualObjectType中的变量:
填充颜色:	填充颜色	"plc_prg.var_fillcol"	var_fillcol=16#FF00FF 填充颜色是粉红色	dwFillColor
报警填充颜色:	当改变颜色变量为True时填充的颜色	"plc_prg.var_fillcol_a"	var_fillcol_a=16#FF00FF 报警填充颜色是粉红色	dwFillColorAlarm
边框颜色:	边框颜色	"plc_prg.var_frameco1"	var_framecol=16#FF00FF 边框颜色是粉红色	dwFrameColor
报警边框颜色:	当改变颜色变量为True时边框的颜色	"plc_prg.var_frameco1_a"	var_framecol_a=16#FF00FF 报警边框颜色是粉红色	dwFrameColorAlarm
填充标志:	可以启用(False)或 禁用(True)填充颜色设置	"plc_prg.var_col_off"	var_col_off:=1 不改变填充颜色,边框保持有效	dwFillFlags
边框标志	边框的显示(实线,点划线等)	"plc_prg.var_linetype"	var_linetype=2; 边框用点划线显示	dwFrameFlags



绝对运动

在可视化元件配置对话框中的‘绝对运动’选项里，可以在‘X 偏移量’或‘Y 偏移量’输入变量。根据相应的变量值，元件可以在 X 或 Y 方向上偏移。在‘缩放比例’中的变量将使元件根据当前值进行放大或缩小。它的数值作为比例因子使用，在起作用时要除以1000，因此要缩小元件不需要使用实数类型变量。缩放涉及到参考点。

‘角度’字段中的变量将使元件按照旋转点旋转一个角度(正值为顺时针旋转)，单位是度。对于多边形来说每个点都旋转，即整个多边形旋转，对于其它可视化元件，在旋转时总是保持上边在顶部。

在元件上点击将出现一个中间是白十字线的黑圈，它是元件的旋转中心。可以按住鼠标左键进行拖拽。

在联机模式下，在'绝对运动'对话框中设置的变量值可以被用于定义同一属性的结构变量的值替换。参看‘可编程性’。

在一个元件属性有多个定义的情况下，依据联机模式下数值可能被替换，要考虑优先的顺序关系。

绝对运动设置对话框

相对运动

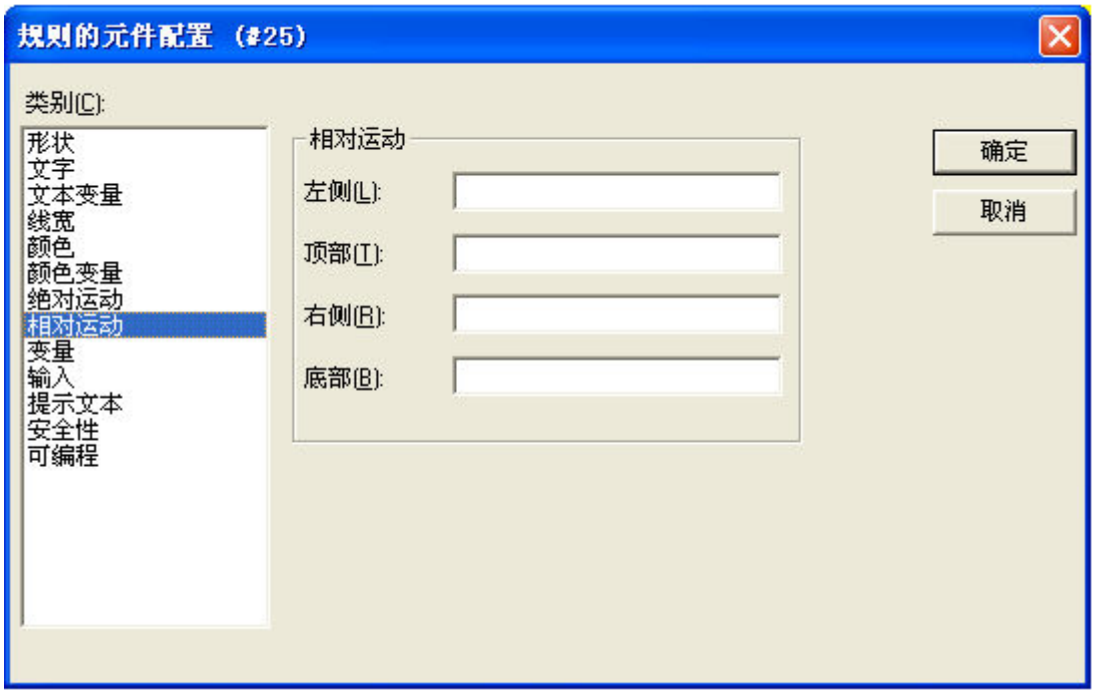
在可视化元件配置对话框中的'相对运动'选项里，可以给各个元件的边界分配一个变量，根据变量的数值，相应的元件边界移动一个距离。输入变量的最简单的方法是使用输入助手(F2)。

四个输入项对应于元件的四个边。角的基准坐标是0。变量输入新值后，对应的边以象素为单位移动这个距离，因此变量应该使用整型变量。

注意	**在联机模式下，在'相对运动'对话框中设置的变量值可以被用于定义同一属性的结构变量的值替换。参看 '可编程性'。**

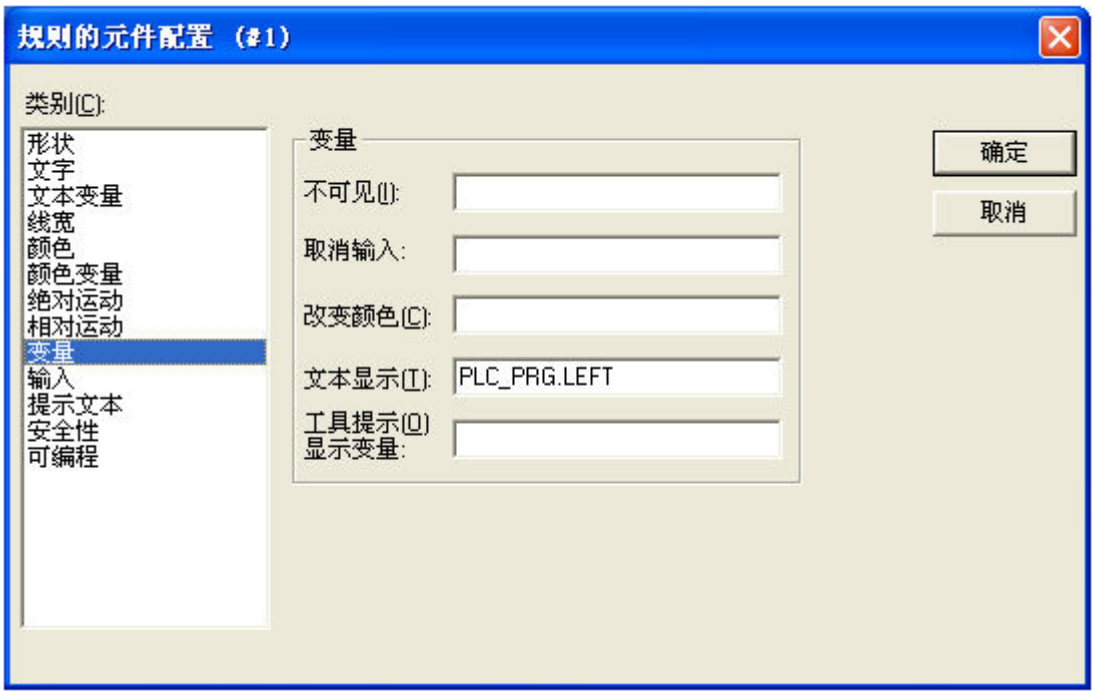
在一个元件属性有多个定义的情况下，依据联机模式下数值可能被替换，要考虑优先的顺序关系。

相对运动设置对话框



变量

变量设置对话框



在可视化元件配置对话框中的'变量'选项里，可以输入描述可视化元件状态的变量。输入变量的最简单的方法是使用输入助手(F2)。

注意	**在联机模式下，在'相对运动'对话框中设置的变量值可以被用于定义同一属性的结构变量的值替换。参看 '可编程性'。**

在一个元件属性有多个定义的情况下，依据联机模式下数值可能被替换，要考虑优先的顺序关系。

在'不可见'和'改变颜色'字段中输入布尔型变量，由变量值来决定动作。如果'不可见'字段中的变量是 False，则可视化元件可见，否则不可见。

取消输入：如果此处变量是 True，则忽略'输入'选项中的设置。

改变颜色：如果此处变量是 False，可视化元件按默认颜色显示，否则按报警颜色显示。

文本显示：

如果已经在“文本”的“内容”中输入了“%s”，则联机模式下，“文本显示”中定义的变量值将会替换“%s” 显示在文本区域。

如果已经在“文本”的“内容”中输入了“%<PREFIX>”(PREFIX 必须是字符串)，那么变量所对应的数值应作为 ID 号输入到'文本显示'中，ID 和 PREFIX 组合在一起对应 XML 文件中描述的文本。在联机模式下它们对应的文本将会替换"%<PREFIX>"。因此可以联机修改文本显示。参看'设置‘’语言'选项。

如果希望在联机模式下使用键盘编辑变量值，可以使用'输入'选项下的'文本显示变量的输入字符'来设置。

工具提示：此处使用的变量是字符串型变量，在联机模式下当鼠标位于此元件区域内时显示变量值(字符串)。

输入

'输入'设置对话框

触发并保持变量值：如果激活此选项，在联机模式时，每次点击可视化元件都将触发变量。可以通过<F2>输入变量，鼠标每点击一次，逻辑变量的值都会从 TRUE 变成 FALSE，再点击从 FALSE 变成 TRUE。

触发但不保持变量值：如果激活此选项，在联机模式时，字段中的布尔变量值将在 TRUE 和 FALSE 之间切换。把鼠标光标放在对象上，按下鼠标键不要松开。如果激活'触发成 False'选项，当按下鼠标时，变量值被设置成 FALSE，否则设置成 TRUE。当释放鼠标时，变量的值变回初始值。

转到：如果激活此选项，在字段中输入同一工程中的可视化界面名，它是在联机模式时用鼠标点击元件后期望打开的界面。

允许的输入是:

当前工程中可视化界面名(参看对象管理器)

如果要转到包含'占位符'的界面中，在调用时占位符可以直接被变量名或文本替换。但必须遵守下列语法:

<Visuname>(<Placeholder1>:=<Text1>， <Placeholder2>:=<Text2>，...， <Placeholder n>:=<Textn>).

在编译过程中将检查文本是否占位符列表中定义的文本相匹配，如果不匹配将产生警告。

例如:

调用界面 visu1，在界面 visu1中使用的占位符var_ref1和var_ref2将分别被变量 PLC_PRG.var1和 PROG.var1替换:

visu1(var_ref1:=PLC_PRG.var1， var_ref2:=PROG.var1)

如果在此处输入了类型为 STRING 的程序变量(如 PLC_PRG.xxx)而不是可视化对象，那么这个变量要定义成可视化名(如 ，visu1') 则当点击鼠标时，程序变量被赋值(如. xxx:= ，

visu1)。

如果输入命令 "ZOOMTOCALLER"， 在联机模式下点击元件将回到上次打开的界面。

注意: 隐含变量 CurrentVisu (类型为 STRING， 作为隐含(系统)变量使用，参看'隐含变量') 保存当前打开的可视化界面名称。 例如在应用中可以控制当前要打开哪个界面。但要注意，当编译器的版本小于 V2.3.7.0时，如果在工程中没有包含库 SysLibStr.lib，那么可视化界面的名称必须使用大写字母(参看 '创建一个可视化对象')。 例如: CurrentVisu:='PLC_VISU';

执行程序： 如果激活此选项， 可以在字段中输入'ASSIGN'或'INTERN'指令，在联机模式下，当用鼠标点击元件时可以执行这个指令。按'...'可以打开配置对话框，在对话框中可以选择命令(添加)和设置指令的执行顺序(上移，下移)。

注意:当可视化界面是唯一的操作接口时，这个功能非常有用。参看'用于操作的可能的特殊输入'。

文本显示变量的输入字符：如果激活此选项，在联机模式下，可以将文本输入到可视化元件的编辑字段。按<Enter>键后输入值写入到'变量‘’文本显示'中定义的变量中。

在滚动条中选择联机模式下的输入方式：

文本:打开编辑字段，输入数据。

Numpad 和 Keypad:它们分别打开数字键盘和字母键盘，然后输入所需数据。如果使用的是触摸屏来操作界面,这个功能非常有用。可以在最小值和最大值中定义输入数值的范围。

注意:在使用目标可视化时，由于可以在鼠标点击后通过特殊接口功能获得用户输入信息，所以要考虑使用的可能性。

在一个元件属性有多个定义的情况下，依据联机模式下数值可能被替换，要考虑优先的顺序关系。

工具提示文本

工具提示文本提供了一个文本输入字段，当在联机模式下光标移动到元件上时显示输入的内容。在“内容”文本框中输入文本，按<Ctrl>+<Enter>组合键可换行。

安全属性

安全属性（Security）用来定义不同的用户组的操作方式和可视化界面的显示。可以通过给不同的可视化元件定义相应的访问权限来实现这个。OtoStudio 有8个用户组(参看‘工程’‘对象’‘属性’或‘project’‘工程’‘用户组密码’)。在访问权限对话框中通过激活相应的选项来分配元件的权限。

访问权限对话框

在联机模式下可视化元件的权限:

无权限	元件不可见
只读权限	元件可见，但不可操作，即无输入权限。
完全访问	元件可见，也可操作

如果给某个元件分配的权限也适合于所有其它元件，激活选项‘适用于所有可视化元件’。

可编程性

一个可视化元素的属性不仅可以用一个静态设定值或用一个“标准”工程变量来定义，而且可以用一个结构变量的分量来定义。结构变量是专门用于编写可视化元件的。

为了达到这个目的，必须使用库文件 SysLibVisu.lib 中的结构 VisualObjectType。它的结构分量可以用于定义元件的大部分属性。

注意	**在一个元件属性有多重定义的情况下，“标准”工程变量的值将改写结构变量的值，同时两者都会改写一个静态定义。**

为了使用一个结构变量来配置元件属性，需做下列工作：

打开配置对话框，选择‘可编程性’选项，在字段“对象名”内输入一个新的、唯一的名字! 要输入对象名，必须在复选框中用鼠标激活此选项。此变量被自动声明为 Visual ObjectType 类型，它是一个包含在库 SysLibVisu.lib 内的结构。该声明是隐含作出的，对用户来说是不可见的。要确保该库文件已包含在库管理器内。

在下一次编译后，在工程内就可采用新分配的结构变量。（提示：激活‘工程’‘选项’‘编辑器’中的智能功能‘显示结构变量’，从而在变量名字后输入一个点后，就可得到结构分量的选择列表）

例子：如果你已为一个可视化对象定义了一个对象名‘Visul-line’，那么就可为这个元件编写线宽，例如“Visul-line.nLineWidth:=4”。

结构 Visual/ObjectType ：

下表表示此结构的现有分量及在配置对话框的不同类目中对相应条目的引用。

可编程性对话框

VisualObjectType 的结构：

下表列出了结构中所有变量，这些变量对应于不同的配置对话框：

在变量名前表明变量的类型：

n INT

dw DWORD

b BOOL

st STRING

变量 (+类型)	功能	举例 (定义的对象名是"vis1")	对应的配置对话框中的输入项:
nXOffset : INT;	X-方向上移动元件	vis1.nXOffset:=val2; (元件在 X 方向上移动到 val2)	- 绝对运动: X 偏移量
nYOffset : INT;	Y-方向上移动元件	vis1.nYOffset:=22; (元件在 Y 方向上移动到 val2)	- 绝对运动: Y 偏移量
nScale : INT;	改变大小	vis1.nScale:=plc_prg.scale_var; (元件大小按 plc_prg.scale_var 值发生变化)	- 绝对运动: 缩放比例
nAngle : INT;	绕元件中心旋转	vis1.anglevar:=15; (元件顺时针旋转 15 度)	- 绝对运动: 角度
bInvisible : BOOL;	元件可见/不可见	vis1.visible:=TRUE; (元件不可见)	-颜色: 无填充色 +无边框色 - 颜色变量:填充颜色 +报警填充颜色
stTextDisplay : STRING;	元件中显示文本	vis1.TextDisplay:='ON / OFF'; 显示文本 ON / OFF	- 文本: '内容'
bToggleColor : BOOL;	当变量在 TRUE 和 FALSE 之间变化时，改变颜色	vis1.bToggleColor:=alarm_var; (只要 alarm_var 变为 TRUE， 元件使用'颜色	- 输入: 触发并保持变量值 + - 变量:改变颜色

		变量'或'颜色'中设置按 dwFillColorAlarm 和 dwFrameColorAlarm 定义的颜色显示.	
bInputDisabled: BOOL;	如果是 FALSE: 忽略'输入'选项中的设置。	vis1.bInputDisabled:=FALSE; (不允许输入)	- 变量: '取消输入'
stTooltipDisplay:STRING;	工具提示文本	vis1.stTooltipDisplay:='Switch for .....';	- 工具提示: '内容'
dwTextFlags: DWORD;	文本位置: 1 左对齐 2 右对齐: 4 水平中心 8 顶部 10 底部 20 垂直中心 注意: 总设置在水平和垂直中心(两值相加)!	vis1.dwTextFlags:=24; (文本放置在元件的中心 (4 + 20)	- 文字:水平和垂直选项 - 文本变量: 文本标记
dwTextColor : DWORD;	文本颜色(颜色定义参看表下面的说明)	vis1.dwTextColor := 16#00FF0000; (文本是蓝色)	- 文字: 字体 | 颜色 - 文本变量: 文本颜色
nFontHeight : INT;	字体高度，以象素为单位。 范围是 10-96	vis1.nFontHeight:=16; (文字高度 16 pt)	- 文字: 字体 | 大小 - 文本变量: 字高
dwFontFlags : DWORD;	字体样式: 1 斜体 2 fett 4 下划线 8 删除线 + 组合值	vis1.dwFontFlags:=10; (文本显示成蓝色删除线)	- 文字: 字体 | 样式 - 文本变量: 字体标记
stFontName : STRING;	改变字体	vis1.stFontName:=Arial (使用 Arial 字体)	- 文字: 字体 | 名称 - 文本变量:字体名
nLineWidth : INT;	边框的线宽(象素为单位)	vis1.nLWidth:=3; (边框线的宽度是 3 个象素)	- 线宽
dwFillColor : DWORD;	填充颜色 (颜色定义参看表下面的说明)	vis1.dwFillColor":=16#00FF0000; (Element ist im "Normalzustand" blau)	- 颜色: 颜色 | 内部 - 颜色变量: 填充颜色
dwFillColorAlarm : DWORD;	只要 bToggleColor 为	vis1.dwFillColorAlarm:=16#00808080;	- 颜色: 报警颜色 | 内部颜色

	TRUE，就填充颜色(参看上面) (颜色定义参看表下面的说明)	(只 要 toggleva 为 TRUE，元件将显示灰色)	- 颜色变量: 报警填充颜色
dwFrameColor: DWORD;	边框颜色 (颜色定义参看表下面的说明)	vis1.dwFrameColor:=16#00FF0000; (边框为蓝色)	- 颜色: 颜色 |边框 - 颜色变量: 边框颜色
dwFrameColorAlarm: DWORD;	只要 bFrameColor 为 TRUE，就填充颜色(参看上面) (颜色定义参看表下面的说明)	vis1.dwFrameColorAlarm:=16#00808080; (只要变量 vis1.bToggleColor 为 TRUE，边框显示灰色)	-颜色: 报警颜色 |报警边框颜色 - 颜色变量: 报警边框颜色
dwFillFlags: DWORD;	显示或不显示由颜色变量定义的颜色。 0 = 显示颜色 >0 = 忽略设置	vis1.dwFillFlags:=1; (不显示颜色)	- 颜色: 无填充色 +无边框色 - 颜色变量: 填充标志
dwFrameFlags: DWORD;	边框的线型: 0 实线 1 虚线 (---) 2 点线 ( ) 3 点 划 线 (_.-._) 4 双 点 划 线 (_..-_..) 8 无变量	vis1.FrameFlags:=1; (边框的线型是虚线)	- 颜色变量: 边框标志

定义颜色值:

例如: e1.dwFillColor := 16#00FF00FF;

颜色按十六进制输入，它由蓝/绿/红 (RGB)组成。 "16#"后面必须先添加两个0以便满足 DWORD 型。 每个颜色都可以得到256个数值(0-255)。

FF 蓝色

00 绿色

FF 红色

闪烁可视化元件：

定义全局变量 'blink1'，类型是 VisualObjectType，将它设置到一个矩形元件。在程序中修改结构变量的值。

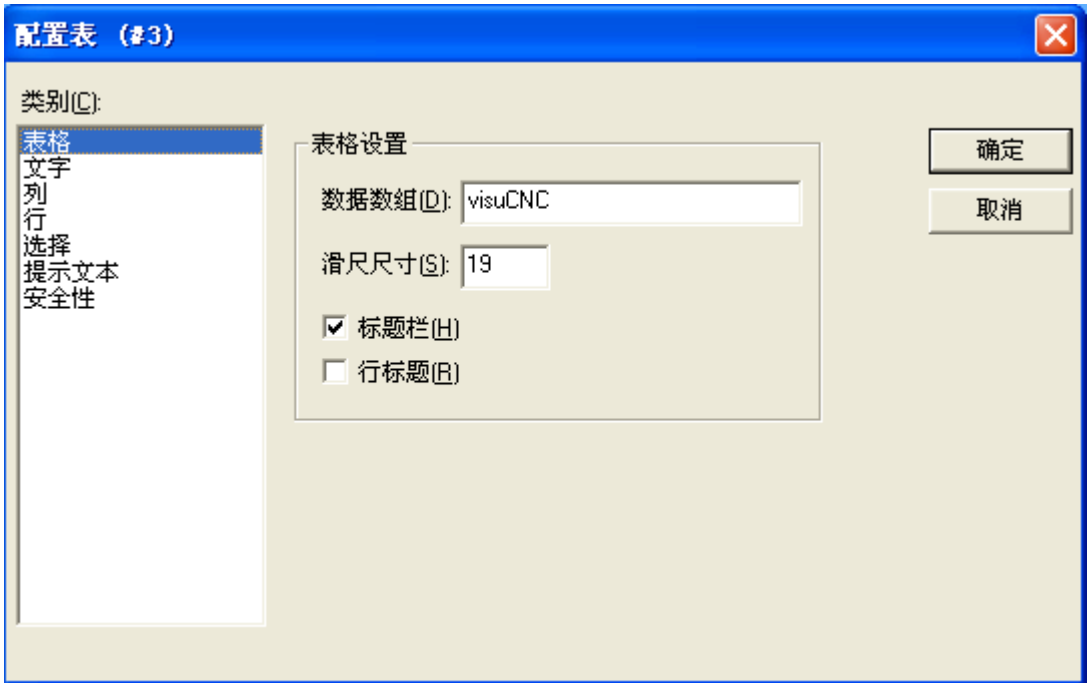
```
PROGRAM PLC_PRG

VAR

  n:INT:=0;

  bMod:BOOL:=TRUE;

END_VAR

(* Blinking element *)

n:=n+1;

bMod:= (n MOD 20) > 10;

IF bMod THEN

  blinker.nFillColor := 16#00808080;    (* Grau *)

ELSE

  blinker.nFillColor := 16#00FF0000;    (* Blau *)

END_IF
```

表格

为了在可视化界面显示一个数组，可以插入一个表格，插入表格后将打开**表格配置**对话

框。 除了'工具提示'和'安全属性'可以用于这个可视化元件外，下面的选项也可用于配置表格的内容显示：

表格配置对话框



表格设置如下:

数组(Data array):输入在表格中将要显示的数组名称，建议使用输入助手<F2>。

滑尺尺寸(Slider size):当数组的大小超过表格的高度时，显示滑尺以便显示看不见的部分。

标题栏(Column header)和行标题(Line header):如果期望显示它们，激活相应的选项。行标题是数组的索引号(表格的第一列)，在选项'列(Column)'中定义列标题。

‘列（Column）’设置

在列设置对话框中设置表格元素，在左侧的窗口中列出了可选的所有元素，它们都是按照数组的索引号来处理。如果是结构数组，它们将是一个结构组件。

使用'>'按钮可将选择的元素移动到右侧窗口，在右侧窗口的元素将在表格中显示。点击'>>'按钮可将所有元素移动到右侧窗口。同样的，点击'<'和'<<'可删除右侧窗口中的元素。如果要修改表格中每个元素的显示样式，可以双击元素或按'...'按钮，这将打开列设置对话框。

配置'表格‘’列‘’列属性'对话框

编辑列标题和列宽:

通常，在列标题(Column header)将自动创建一个标题(如果是结构数组，标题为"PLC_PRG.arr1[INDEX].iNo"，'iNo'表示的是结构中的变量)。你可以修改这个标题。此外可以设置列宽(字符数)。

配置列中的元素:

在默认方式下，表格单元只显示为一个矩形且不可编辑。但是如果在选择列后激活'编辑模板'，可以修改此列中的字段参数设置，如线宽，文本输入等。模板对此列中所有字段都起作用，模板也可以借助可视化元件配置对话框进行编辑。

如果期望配置列中的几个特定字段，可以使用占位符来确定行和列：$ROWCONST$，$COLCONST$， INDEX. (INDEX 与$ROWCONST$具有相同的作用)。

占位符使用的例子 ：

占位符输入可以使用„AND“ 或“OR“指令： 例如： "$ROWCONST$=1 OR $ROWCONST$=3" 使得单元格不同的设置。

是否使用配置模板可以激活或取消选项'**使用模板**'。

行（Row）设置

行高： 输入期望的高度值，单位是象素。

选择(selection)设置

设置与选择单元格有关的参数：

选择颜色： 按此按钮定义单元格选中时的颜色。它将打开标准的颜色对话框以便选择所需颜色。

选择类型： 在联机模式下，当用鼠标点击表格中的单元格后，将选择表格中的哪些部分：

选择单元格： 只有点击的单元格被选中。

只选择行： 单元格所在行被选中。

只选择列： 单元格所在列被选中。

选择行和列： 单元格所在行和列都被选中。

所选单元格周围显示边框： 选中单元格用一个框包围。

选择 X(行)的变量， 选择 Y(列)的变量： 在此处输入一个工程变量，它们分别表示所选单元格的 X 和 Y 索引号。

创建一个表格元件来显示一个结构数组

声明下面的结构：

TYPE strucTab ：

STRUCT

iNo： INT：

bDigi ： BOOL：

sText：STRING：

byDummy： BYTE：

END_STRUCT

END_TYPE

在 PLC_PRG 中声明下面的结构数组：

arr1：ARRAY [1..5] OF strucTab：

and the following variables：

selX：INT：

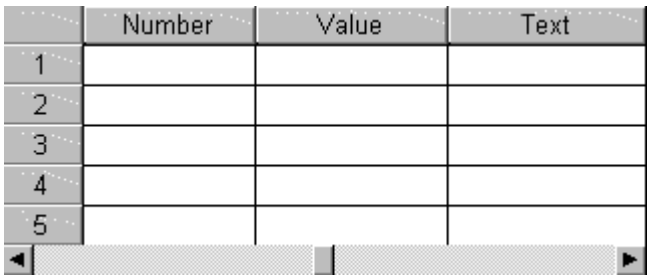
selY：INT：

创建一个可视化界面并插入一个表格元件。配置如下：

表格： 数据数组： "PLC_PRG.arr1"

列： 将结构中的变量 iNo， bDigi， sText 移动到右侧窗口中，在右侧窗口中鼠标双击第一项(PLC_PRG.arr1[INDEX].iNo)，则打开对话框，用"Number"替换默认标题。 按'确认'，然后重复定义另外两列的标题(如："Value" 和 "Text")。

选择： 在 '选择 X(行)的变量'中输入： "PLC_PRG.selX" 在'选择 Y(列)的变量'中输入：PLC_PRG.selY"。 激活选项'所选单元格周围显示边框'。按'选择颜色'后选择'黄色'。 关闭配置对话框。 表格元件显示如下：



左侧边界显示了数组索引号，在上面显示了所选结构变量的标题。当将光标位于两列的

分界线，会出现一个水平双箭头，移动鼠标可以修改列宽。

在联机模式，在表格单元格里显示数组变量值。当用鼠标点击单元格时，单元格变为黄色且边框加粗显示。如：

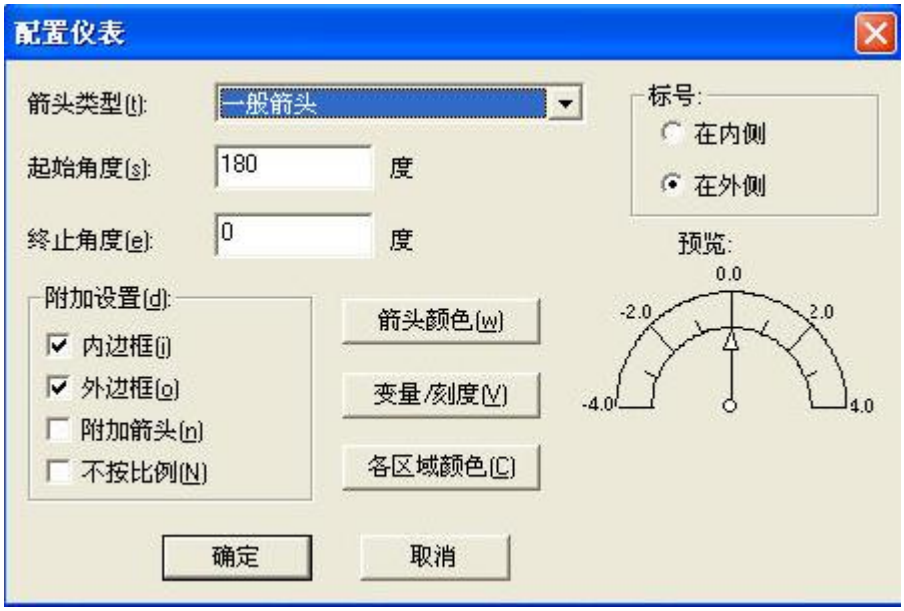
	Number	Value	Text
1	0	TRUE	text1
2	33	TRUE	text2
3	55	FALSE	abc
4	0	FALSE	
5	0	TRUE	

第一个按钮是设置选择后表格颜色的变化，在选择类型（selection type）选项中可以定义点击时选择的区域，分别是单元格（cells）、整行（only rows），整列（only column）和全部（rows and columns）。还可以定义选择区域的框架。

在表格中选择单元格后显示的表格形式如图例所示。

仪表

仪表元件设置对话框



在一个可视化对象中插入一个仪表元件将自动打开一个对话框。 在对话框的右下角有

预览，可以显示设置完下列参数后的结果：

箭头类型：定义仪表指针的箭头类型。可选的类型： 一般箭头， 窄箭头，宽箭头，窄针箭头。

起始角度， 终止角度：定义仪表圆弧起始和终止角度，以度为单位。 (如： 起始角度是180度，终止角度是0度将显示上半圆)。

箭头颜色：按此按钮将打开标准颜色选择对话框或特定目标的颜色选择列表来定义指针的颜色。

变量/刻度：按此按钮打开配置刻度和变量 对话框。

各区域颜色：按此按钮打开 配置颜色区域对话框。为刻度的不同区域定义不同的颜色。

标号：依据选中的是‘**在内侧**’还是'**在外侧**'，刻度的标号将显示在圆弧内侧或外侧。

其它设置：

内边框， 外边框： 是否显示仪表元件圆弧的内外边框。

附加箭头：除了主指针外，一个小箭头指向刻度的当前值。

不按比例： 选中此项后，仪表的大小不随元件大小的改变而改变。

条状图

条状图设置对话框

在一个可视化对象中插入条状图元件将自动打开一个对话框。 在对话框的右下角有预览，可以显示设置完下列参数后的结果：

图表类型：可选类型：'刻度在条状图外面'， '刻度在条状图里面' 和 '条状图在刻度里面'。

方向：条状图的方向是水平方式还是垂直方向。

运行方向：条状图的伸长方向是从左到右还是从右到左。

条状图颜色：按此按钮打开标准对话框来选择颜色。定义条状图在一般状态下的颜色(非报警颜色)。 如果激活'使用区域颜色'(参看下面)，此项将被禁用。

报警颜色：按此按钮打开配置报警表对话框，在此处定义显示报警颜色的数值和报警颜色：在编辑字段输入限制值并选中报警条件是大于设置值还是小于设置值， 以便当值大于或小于限制值后显示报警颜色 按报警颜色按钮打开标准对话框或或特定目标的颜色选择列表选择报警颜色。按'确认'按钮确认选择的颜色并返回到主配置对话框。如果激活'使用区域颜色'(参看下面)，此项将被禁用。

变量/刻度：按此按钮打开配置刻度和变量对话框，此对话框的使用与仪表元件中的一样。

元件边框：选中此项后条状图被一个边框包围。

带背景色：选中此项后条状图的整个显示范围以黑色作为背景，否则只按当前值显示条状图的长度。

使用区域颜色：选中此项后，在'条状图颜色'和'报警颜色'中的设置将无效。在这种情况下将使用定义的区域颜色，在'各区域颜色'中设置区域颜色。按'各区域颜色'按钮打开区域颜色设置对话框。(参看下面)为刻度的不同区域定义不同的颜色 。

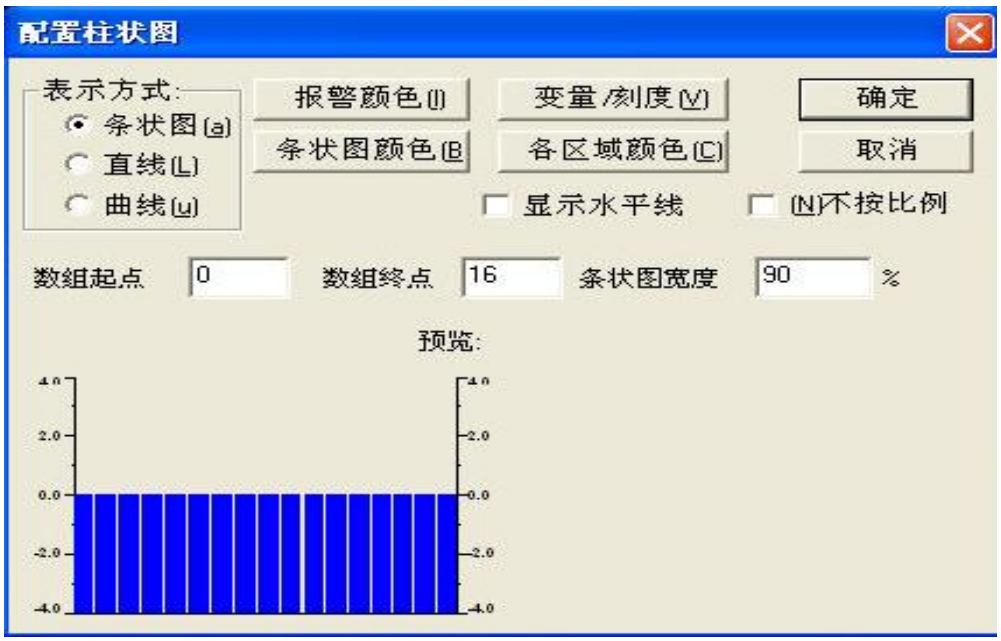
各区域颜色：按此按钮打开打开'配置区域颜色'对话框，在此处为刻度的不同区域定义不同的颜色。这些定义只有在选中'使用区域颜色'才生效。此对话框的使用与仪表元件中的一样。

不按比例：选中此项后，条状图的大小不随元件大小的改变而改变。

柱状图

为了在可视化界面显示一个数组，数组元素的值通过并排的条状图或线来表示，它们的高度表明元素的当前值。

柱状图配置对话框：



在一个可视化对象中插入柱状图元件将自动打开一个对话框。 在对话框的左下角有**预览**，可以显示设置完下列参数后的结果：

表示方式： 选择柱状图的显示方式：条状图，直线，和曲线。

显示水平线： 选中此项，每个刻度上将增加一条横贯柱状图的水平线。

不按比例： 选中此项后，柱状图的大小不随元件大小的改变而改变。

报警颜色： 按此按钮打开配置报警表对话框，在此处定义显示报警颜色的数值和报警颜色：在编辑字段输入限制值并选中报警**条件**是**大于**设置值还是**小于**设置值， 以便当值大于或小于限制值后显示报警颜色 按报警颜色按钮打开标准对话框或或特定目标的颜色选择列表选择报警颜色。按'确认'按钮确认选择的颜色并返回到主配置对话框。

变量/刻度： 按此按钮打开配置刻度和变量对话框，此对话框的使用与仪表元件中的一样。

各区域颜色： 按此按钮打开打开'配置区域颜色'对话框，在此处为刻度的不同区域定义不同的颜色。此对话框的使用与仪表元件中的一样。

条状图颜色： 按此按钮打开标准对话框来选择颜色。定义条状图在一般状态下的颜色(非报警颜色)。

定义数组显示的范围：

数组起点： 显示的第一个数组元件(索引号)。

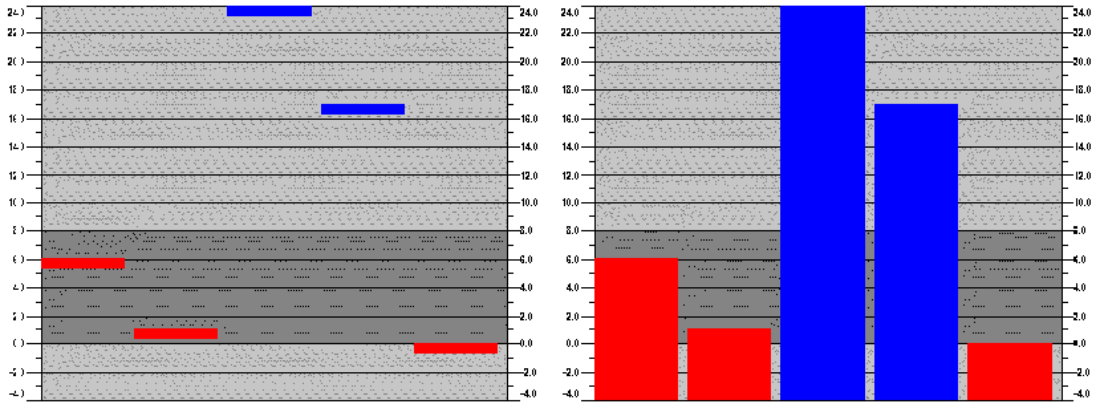
数组终点： 显示的最后一个数组元件(索引号)。

条状图宽度：定义条状图的显示宽度，以百分比为单位。

例如：

下图是柱状图(条状图和直线)在联机模式下的显示，它们表示整型数组 arr1 [0..4]。数组起点是"0"，终点是 "4"， 刻度起点是"-4"，终点是 "24"， 主刻度是"2"， 子刻度是"1"，

刻度范围0到8的颜色是深灰。当某个数组元素的值超过8时，条状图显示蓝色。在图中数组元素 arr1[2]和 arr1[3] 处于报警状态：



报警表

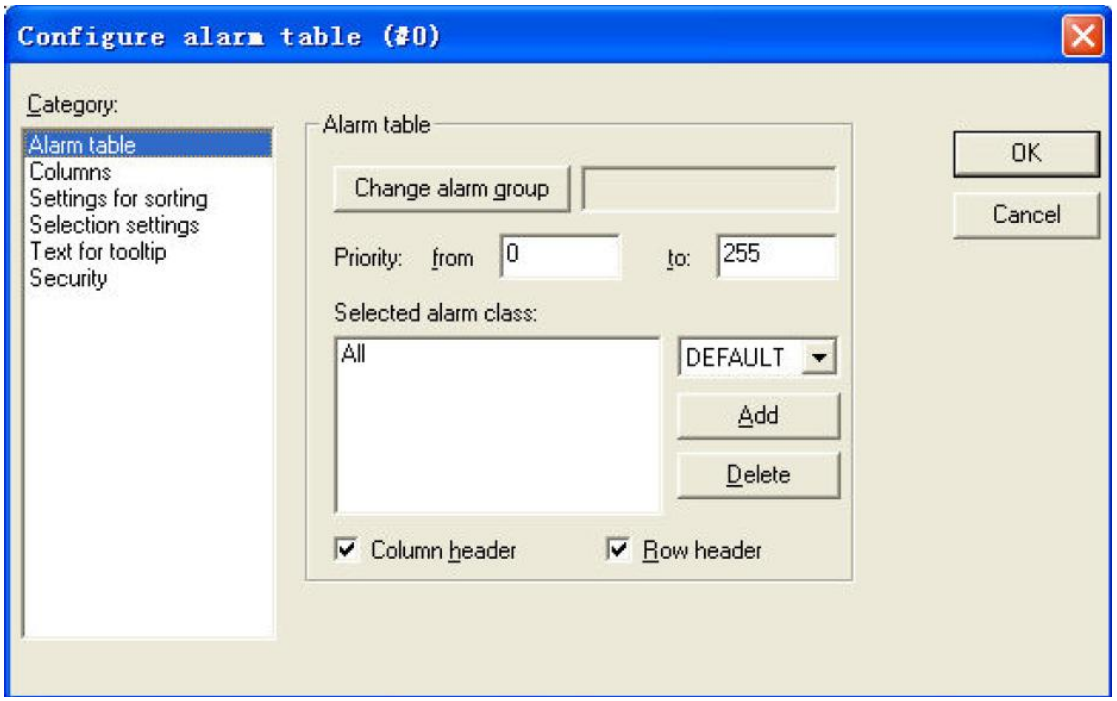
元件'报警表'用于查看联机模式下的报警，这些报警要首先要在 OtoStudio 的报警配置中配置。

注意	**Depending 依据不同的目标系统，要考虑在 PLC 中处理报警的可能性。**

插入报警表后将打开**报警表配置**对话框。 除了'工具提示'和'安全属性'可以用于这个可视化元件外，下面的选项也可用于配置报警表的显示：

报警表选项：

报警表选项对话框



定义在报警表中显示的内容：

改变报警组： 按此按钮将打开已在报警配置中定义的报警组列表，选择期望显示的报警组(即使报警组只包含一个报警)。

权限： 定义显示所有报警的权限。允许值从0 到255。

报警类： 选中期望显示的报警类然后按'**添加**'按钮将它添加到'已选报警类'窗口中。重复做此步以选择所有期望显示的报警类。如果想删除某个已选报警类，按'**删除**'按钮。

激活选项**列标题**和**行标题**，可以在报警表中显示标题。

设置分类选项：

定义报警表分类的方式：

分类列： 根据权限，报警类，日期/时间或报警状态分类

分类排序： 升序或降序： 例如：根据权限升序排序，从0开始，后面的数越来越大。

列选项：

定义在报警表中显示哪些报警参数： 可以定义的参数(除了报警发生的日期和时间以及在报警组中配置的报警状态以外)是： 位图， 日期， 时间， 表达式， 值， 信息， 权限，类型， 类别， 状态， 目标值(用于报警类型 DEV+和 DEV-)， 死区。

使用按钮">", ">>"可以将左侧窗口中的一个或所有参数添加到右侧窗口。在右侧窗口中定义的选项将在报警表中显示。使用按钮"<" 和"<<" 可以删除已选的选项。

在右侧窗口中的某个选项上双击鼠标可以打开'列配置'对话框。在对话框中可以定义**列标题**和**列宽**。

报警表的选择设置选项：

设置表格字段的显示方式：

选择颜色： 按此按钮打开标准颜色对话框或特定目标的颜色选择列表来选择颜色。定义当选中字段后显示哪种颜色。

行高： 表格中每行的高度，单位：象素。

滑尺尺寸： 滑尺高度，单位：象素。

选中行带边框： 选中此项后，所选的行带边框显示。

显示状态行： 选中此项后， 在报警表下面显示一个状态行，在联机模式下它提供如下的按钮用于操作：

确认： 在报警表中标记的所有报警将被确认。

全部确认： 在报警表中列出的所有报警将被确认。

历史： 按此按钮，在报警表中显示的是发生的所有事件(所有转换)的列表，而

不是报警的当前状态。在列表中不能进行报警确认!新事件自动添加。

如果定义了记录文件，在激活'保存'后，在文件中记录发生的所有报警类事件。

启动： 取消停止 (参看下面)

停止： 在重新按'启动'按钮以前，停止记录发生的事件。

确认变量： 此选项只有在不选择'显示状态行'的情况下才能使用(参看上面描述)。如果选中此项，在状态行中的按钮操作将由变量来控制。要定义变量，首先从列表中选择功能，然后在编辑字段中输入工程变量。因此，例如，在联机模式下，确认所有报警可以通过一个上升沿变量来触发。

趋势图

趋势图配置对话框

在联机模式下，趋势图元件用于记录变量值随时间的变化。可以与跟踪功能相比较。 在联机下采用图表的方式表示，如果记录到文本文件，每个值独占一行。

在趋势图配置对话框的'趋势'选项中可作如下配置：

曲线类型： X/t， 水平轴是时间轴，垂直轴是数值

方位： 从左到右或从右到左： 新记录的值是显示在左边还是右边：

轴：

水平轴：

趋势图元件中的水平轴配置对话框

分割线：如果显示拉长的刻度线，激活选项'可见'。 在这种情况下可以定义'刻度线设置': 设置的数值确定在水平轴上分割线的间隔。可以用鼠标点击对应的矩形分别来设置分割线的类型 (直线 ___， 虚线 _ _ _， 点线 .....， 点划线 _ . _ .) 和颜色。

刻度： 在'**持续时间**'中定义刻度的显示范围。如果定义为"T#20s0ms"，刻度范围将是20秒的周期。 采用相同的语法，可以定义显示的主刻度分割线和子刻度分割线， 它们分别用长线段和短线段来标记。

精度： 此处定义显示变量当前值的间隔(采用日期的标准格式，如 T#5ms)。

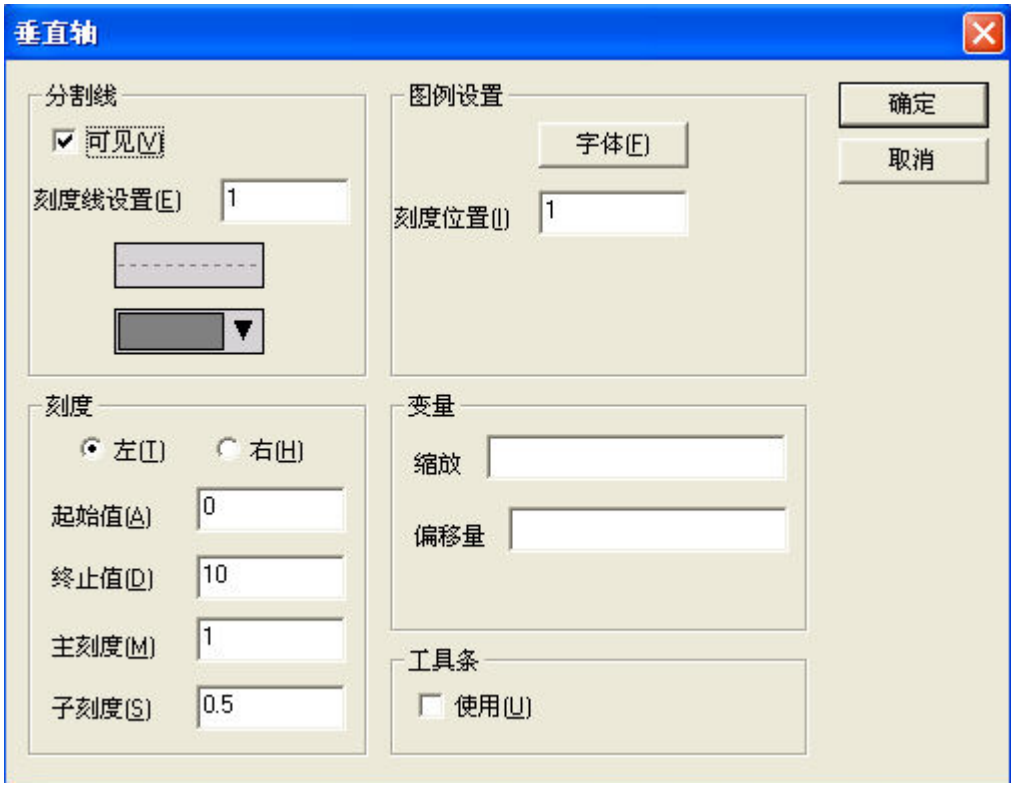
图例设置： 此处设置图例。 按字体按钮打开标准或特定目标的字体对话框选择字体。在**刻度位置**处输入显示刻度的间隔距离(如： T#4ms，那么每隔4ms 显示一个刻度值)。根据激活的选项，刻度中可以包含时间和日期。可以在'格式'字段中定义期望的格式。

变量： 这些变量用于水平轴的缩放和偏移。例如偏移量变量的值是10，那么水平轴显示范围的偏移量设置成10。

工具条： 选中此项将在水平轴下面增加一个工具条，在联机模式下，使用工具条中的按钮可以滚动或缩放趋势图。单箭头按钮将显示范围沿时间轴移动一步，双箭头将显示范围偏移到记录的开始或结尾处。缩放按钮实现水平轴刻度的缩放。为了恢复成缩放和偏移量的初始设置，在垂直轴中使用工具条将可得到'恢复'按钮。

垂直轴：

趋势图元件中的垂直轴配置对话框



分割线： 参看水平轴描述。

刻度： 设置趋势图的刻度是显示在趋势图的左侧还是右侧。 设置刻度的'起始值和'终止值'以及'主刻度'和'子刻度'分割线。

图例设置： 字体和分割线：参看水平轴描述

变量： 参看水平轴描述

工具条： 参看水平轴描述， 附加的'恢复' 按钮用于将缩放和便移量恢复到轴的其始设置。

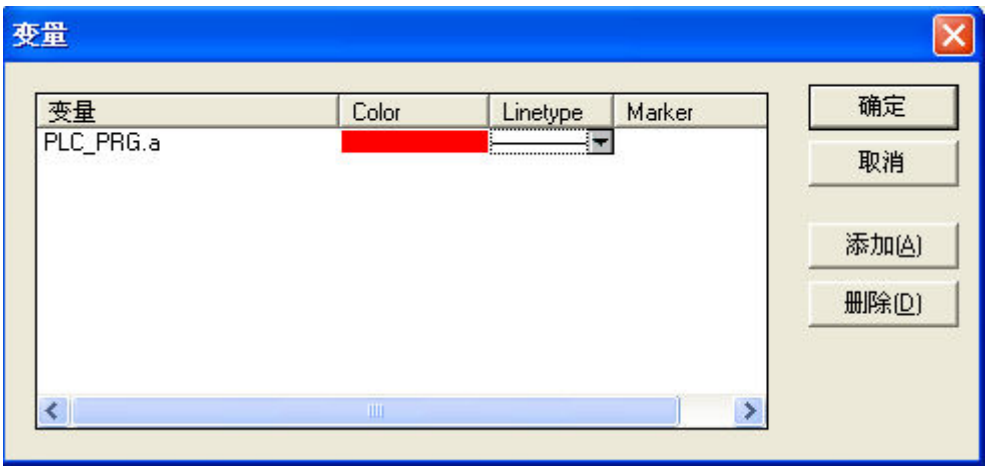
记录： 选择趋势图记录方式：'联机时'或'历史'记录方式。'联机时'就是根据设置的刻度范围按时间将变量值显示在图表上，'历史'就是将记录存储到文件中，按'配置'按钮'打开对话框进行配置，这个对话框和报警日志文件的一样。

注意	**如果在目标系统中激活选项'趋势数据保存到 PLC 中'， 将忽略定义的路径，日志文件将存储在 PLC 的下载目录中。**

在日志文件中，每次测量结果都单独占一行，每行都包含所有的变量名称和数值。每一行以唯一的标识符按 DWORD 格式开头，它由测量的日期构成。

选择变量： 按此按钮打开变量配置对话框， 在对话框中配置跟踪记录的变量和它们的显示方式：

趋势图元件中的变量配置



在'**变量**'栏中输入工程变量，可以在输入字段处点击鼠标进行输入。最好使用输入助手 <F2>或智能功能进行输入。

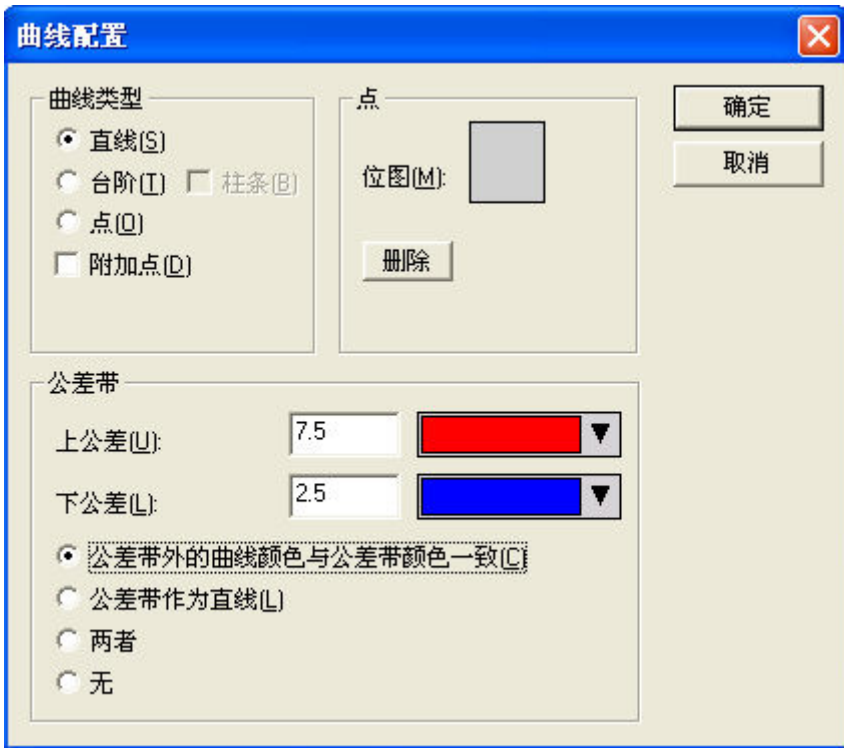
颜色和线型：用于设置变量在记录中的样式。在'颜色'栏中相应的字段处点击鼠标打开标准或特定目标的颜色选择列表，在'线型'栏中的相应字段选择线型(直线 ___， 虚线 _ _ _， 点线 .....， 点划线 _ . _ .)。

在'**标记**'栏中定义一个变量用于在联机模式下使用标记功能显示当前的记录值。标记在图表的左上角显示成一个小灰三角形。如果点中它并保持鼠标按下，可以沿时间轴平移这个标记。作为标记的变量将从记录曲线中读出相应的变量值。

设置所期望的所有记录。使用**添加**键在列表的末尾处增加一行，用**删除**键删除当前行。

曲线配置： 按此按钮打开曲线配置对话框设置趋势曲线：

趋势图元件中的曲线配置



曲线类型： 选择直线，台阶和点中的一种。 对于前两项可以使用显示**附加点**。在显示

附加点时可以使用位图，否则附加点显示成一个填充的矩形(颜色与曲线的一致)符号。点击位图旁边的矩形将打开选择位图文件对话框选择文件。使用**删除**键删除配置的位图。

公差带： 可以定义垂直轴的上下极限值作为公差带来显示。每个公差带可以定义一个颜色(按矩形将打开标准或特定目标的颜色选择对话框)。 如果在联机模式下显示公差带，激活选项**公差带作为直线**。 当数值超过公差值时期望曲线显示的颜色与公差带的一致，则激活**公差带外的曲线颜色与公差带的颜色一致**选项。如果想同时使用上述两种方式或都不使用，激活选项**两者**或**无。**

举例： 联机模式下的趋势图元件：

在程序 PLC_PRG 中的声明：

```
VAR
n： INT：
rSinus：REAL：
rValue：REAL：
rSlider1：REAL： (*用于标记功能*)
rSlider2：REAL： (*用于标记功能*)
END_VAR
```

PLC_PRG 程序：

```
n：=n+1：
rValue ：= rValue + 0.01：
rSinus：=SIN(rValue)*50 + 50：
IF n>100 THEN
```

n：=0：

END_IF

在可视化界面中配置趋势图元件：

方位：右-左： 历史记录方式

水平轴： 分割线： T#2s， 持续时间： T#10s， 主刻度： T#1s， 子刻度： T#500ms，精度： T#200ms， 图例设置： 时间格式 ('hh'：'mm'：'ss')， 刻度位置 T#2s。 使用工具条。

垂直轴： 分割线可见， 刻度线设置： 10， 点线， 灰色： 刻度：左， 起始值： 0，终止值： 100， 主刻度： 10， 子刻度： 5： 图例设置： 10： 使用工具条。

变量：

1. 变量 PLC_PRG.rsinus， 蓝线， 标记变量： PLC_PRG_TRD.rSlider1：

2. 变量 PLC_PRG.n， 红线， 标记变量： PLC_PRG_TRD.rSlider2

曲线配置： 直线，无公差带

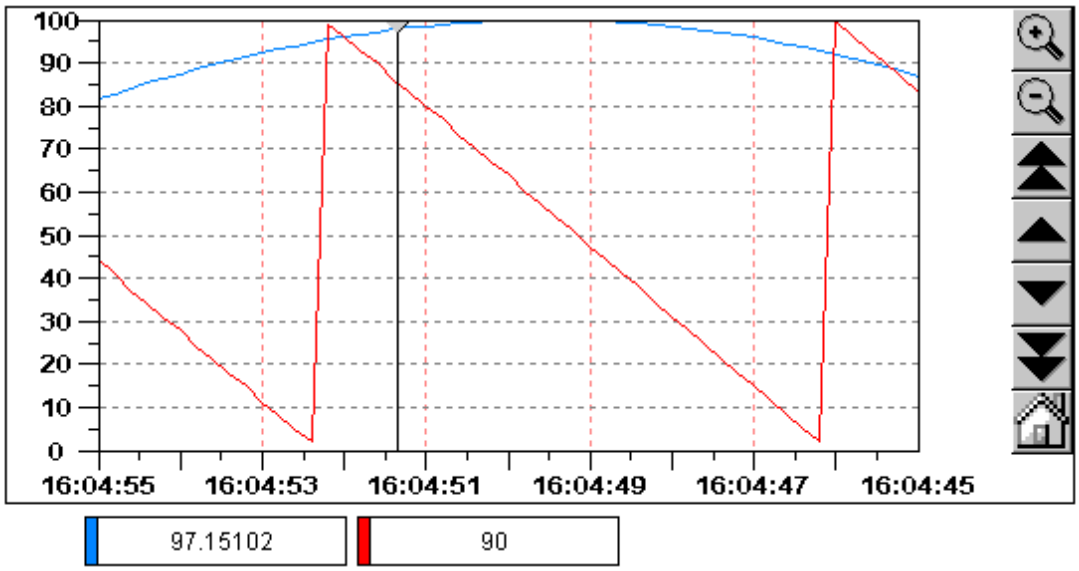
配置由标记变量提供的当前记录值的显示：

矩形元件1： '文字'选项： 输入"%s"到内容字段： '变量'选项： 在文本显示中输入：PLC_PRG.rSlider1

矩形元件2： '文字'选项： 输入"%s"到内容字段： '变量'选项： 在文本显示中输入：PLC_PRG.rSlider2

(在矩形元件1和2的左边分别放置一个矩形元件，矩形的填充颜色与记录变量的曲线颜色一致。)

程序在联机模式下的运行结果：



记录曲线从左向右移动显示： 最新值显示在最左边： 每隔200ms 增加一个新值。如果沿时轴移动标号(左上角的灰三角)，在矩形元件中显示记录变量在曲线当前位置的数值。

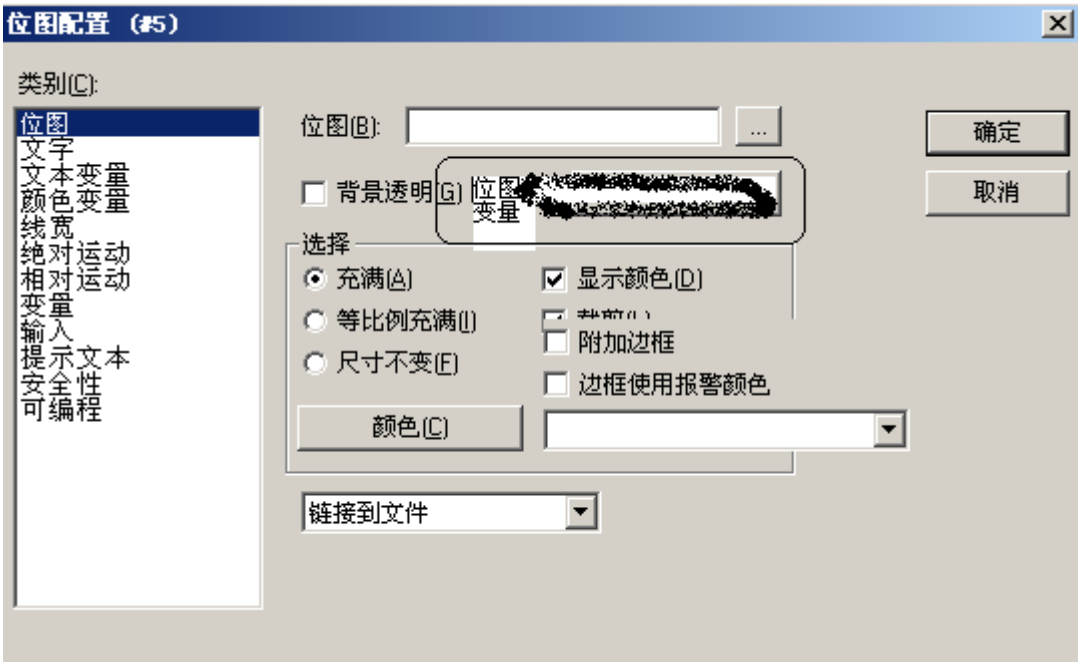
位图

在可视化元件配置对话框的‘位图’选项中，可以作如下的设置：

指定要使用哪个图象文件，其定义可以是静态或动态的。如果在‘位图’字段和‘位图变量’字段中都有输入，那么将忽略‘位图’字段中的输入：

位图：静态定义：输入本地文件系统中的有效图象文件路径。通过按钮'...'，打开浏览文件对话框来选择所需文件。

位图变量：动态定义：输入 STRING 类型的工程变量，它包含当前要使用的图像文件名。这可以实现在联机模式下动态改变图像，但是只能使用工程全局“位图列表”中的“位图文件”（见‘附加’‘位图列表’）。即使在位图列表中指明了全路径，这个字符串变量也必须指定文件名。



注意上图中的位图变量选项自动隐藏了，需要用鼠标点击黑色区域，编辑框会自动弹出。

请看下面的例子，'stBitmap'为字符串变量，它从全局位图列表中获得不同的图象文件。

```
CASE nId OF

0：  stBitmap ：= 'background.bmp'：

1：  stBitmap ：= 'deutest.bmp'：

2：  stBitmap ：= 'alarm.bmp'：

END_CASE
```

如果在 PLC_PRG 对象中声明 stBitmap，那么，可以在配置对话框‘位图变量’字段中输入：“PLC_PRG.stBitmap”。

下列设置项将影响位图的显示。

选择“充满”“等比例充满”或“尺寸不变”来指明当边框尺寸发生变化时，位图的尺寸将如何改变。"充满"就是位图的尺寸与边框的一样大，位图的高与宽分别调整以便适合于边框。"等比例充满"就是即使位图的整个尺寸发生变化，但位图依旧保持着高

与宽之间的比例。"尺寸不变"就是位图保持其原始尺寸，不受边框尺寸变化的影响。

如果同时选择了"尺寸不变"和"裁剪"选项，那么只显示在边框中包含的那部分位图。

如果选择"显示颜色"选项，那么边框按选择的颜色显示，颜色是在颜色对话框(标准或特定目标)中的"颜色"和"报警颜色"选项中进行设置。当"变量""改变颜色"中的变量为“TRUE”时，才显示报警颜色。

在对话框下部的选择列表中，可以定义将位图插入到工程内（**嵌入的**）还是**链接**到一个外部位图文件中（其路径在上面的‘位图’字段中输入）。将位图文件保存在工程目录下是合理的，因为此时可以输入一个相对路径，否则必须输入一个绝对路径，特别是当把工程导入到另一个工作环境时会产生一些麻烦。

位图元件配置对话框

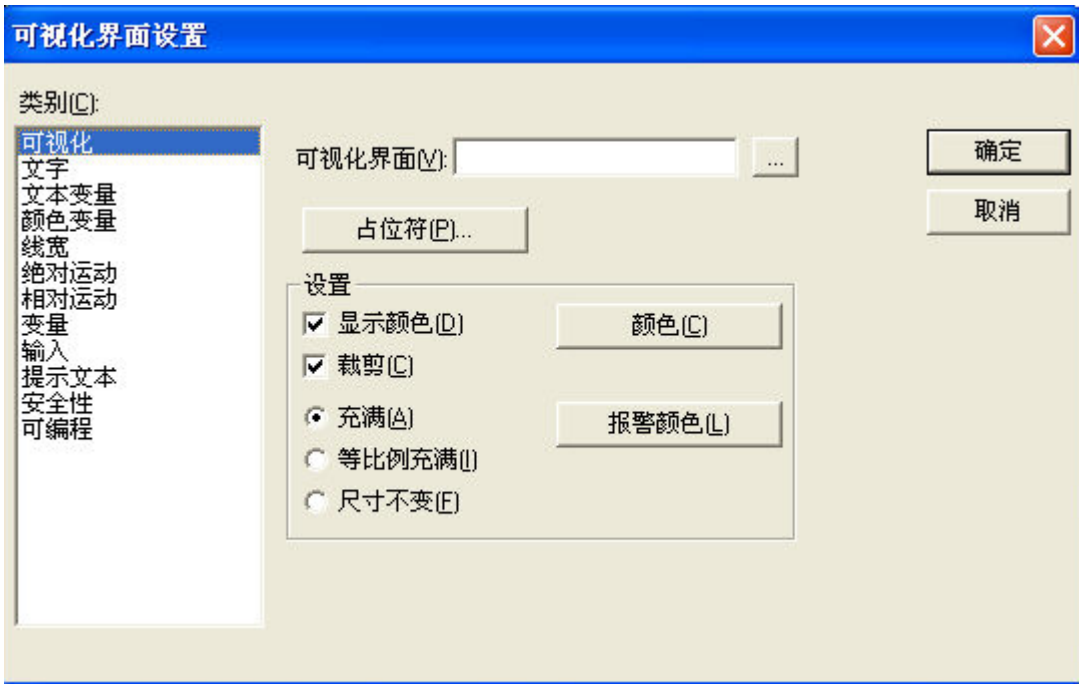
可视化界面元件

当将一个可视化界面插入到其他可视化界面中时，意味着正在建立一个可视化的“**实例**”。

可以在可视化元件配置对话框中的“**可视化**”选项对这个实例作配置。

在“可视化”字段内输入可视化对象的名字。使用'...'按钮打开一个包含工程中可使用的可视化界面列表对话框。除了本身的可视化界面外，可插入任何一个可视化界面。

可视化界面元件配置对话框



有关可视化界面元件**边框**的设置，可以参看'位图'中的描述。

点击'占位符'按钮可以打开'占位符列表'对话框，在"占位符"栏中列出了所有可以插入到"主"可视化配置对话框中的占位符。在"替换"栏中可以用一个确定值来替代当前实例的占位符。在给定情况下是否能够替代取决于在"主"可视化界面中的‘附加’‘占位符列表’对话框中预先定义的数值组。如果是这种情况，就会显示在一个选择组合框。如果未预先定义，则在'替换'栏的相应字段上双击，打开一个编辑区段，填入所需的设置值。

另一种替换实例中的占位符的方法是在'**输入**‘’**转到**'配置对话框中设置可视化界面的调用。

当作用配置对话框‘输入’类目中的‘Zoom to vis’区段来定义可视化调用时，可以直接替代实力占位符。

一个可视化实例的在线表现：并且它含有的可视化元素作出与原始可视化元素一样的响应。

不可能有按时序替代的控制！因此，不应该用含有占位符的正文来替代占位符！

当使用占位符时，它已不再有可能在工程编译时立即检查可视化元件配置中的无效输入。因此，相应的错误信息首先在联机模式中显示。(...无效的查看表达式..).

占位符的应用例子

使用相同的可视化界面可以很容易的显示功能块实例。例如，配置一个可视化界面 visu 用于监控功能块中的变量，每个变量以占位符 FUB开头 (如 FUB.a)。 如果使用 visu 的实例(在另一个可视化界面中插入 visu 或使用'转到'来调用)，那么在这个实例的配置中，占位符FUB 必须用功能块的实例名替换，这样才能监控变量。

如下所示：

在工程中定义了包含下列声明：

```
FUNCTION_BLOCK fu
VAR_INPUT
 changecol ： BOOL： (* 在可视化界面中改变颜色 *`)
END_VAR
```

在 PLC_PRG 定义了两个'fu'实例：

```
inst1_fu ： fu：
inst2_fu ： fu：
```

创建可视化对象'visu'，插入一个元件并打开配置对话框，打开'变量'选项，在字段'改变颜色'中输入 "FUB.changecol"。打开'输入'选项，在字段'触发但不保持变量值'中输入

"FUB.changecol"。打开'文字'选项，输入"FUB - change color "

创建另一个可视化对象'visu1'。在'visu1'中插入两个'visu'(两次参考'visu')

选择'visu'的第一个参考，打开配置对话框中的'可视化'选项，点击'占位符'按钮后将显示占位符列表，用'PLC_PRG.inst_1'替换 'FUB'。

选择第二个'visu'实例，按上面的描述，用'PLC_PRG.inst_2'替换 'FUB'。

在联机模式下，用于配置'fu'实例的变量值将在相应的'visu'实例中显示结果。

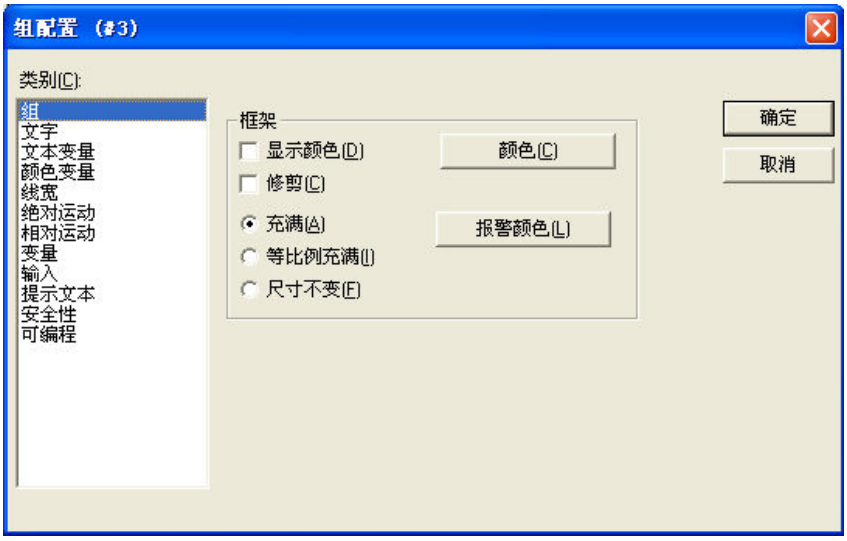
当然，占位符FUB可以在'visu'配置中的所有可输入变量或文本的位置使用。

联机模式下的可视化界面实例：如果插入一个可视化元件，然后选择和配置这个实例，那么它被视作单个对象，且在联机模式中根据它的配置，对输入作出响应，相反： 如果不配置实例，那么在联机模式下它的显示与原始的一样。

组

用于成组的可视化元件对话框提供的设置与位图的'边框'选项中的一样，如充满，等比例充满，尺寸不变，显示颜色，修剪，颜色和报警颜色。

但要考虑在组边框拉伸或缩小时，如何保持组中可视化元件的大小。



用于操作的可能的特殊输入

OtoStudio 可视化可以作为一个纯粹的操作接口专用于 OtoStudio HMI 或 target 可视化，然而对于用户来说没有可用的菜单、状态条和工具条，也就不可能修改代码。

于是，在用 OtoStudio 创建一个可视化界面用作操作时，必须把基本的操作和监视功能赋予可视化元件，于是在联机模式下通过鼠标和键盘来访问它们。

对于配置用于 OtoStudio HMI 的可视化元件可以有下列某些特殊的输入，它们可以在可视化元件配置对话框中设置：

在“输入”选项“执行程序”字段中按照下列语法输入内部命令（可以使用“配置程序”对话框）。

INTERN <COMMAND(命令)> [PARAMETER(参数)]*

下表描述了可以采用的内部命令。某些命令要求接受几个参数，参数间用空格分隔。方括号中的参数是可选参数。对于那些要求指定监控列表的命令可以使用占位符来替代它们的名字。如果一个元件中要输入多个命令，那么这些命令用逗号分隔。

命令	OtoStudio 编程系统中的命令	描述
ASSIGN <变量名>：=<表达式>	赋值	将一个变量或一个表达式赋给另一个变量。 如： INTERN ASSIGN PLC_PRG.ivar1：=PROG1.ivar+12：
PROGRAM <可执行的程序路径> [打开文件的路径] 2)	程序调用	执行一个程序。 如： INTERN PROGRAM C：\programms\notepad.exe text.txt
LANGUAGEDIALOG2)	可视化界面设置	打开包含'语言'选项的可视化界面对话框。
LANGUAGE <在语言*.vis，*。tlt 或 *.txt 文件中设置的标识符 > 注意： For 对于可视化界面	可视化界面设置， 语言	在不打开可视化界面设置对话框的情况下，设置期望的语言。

建议使用 *.vis 语言文件 ！		
LANGUAGE DEFAULT <在语言文件中设置的标识符>	可视化界面设置， 语言	对于动态文本，使用在 xml-文件 中定义的语言。
DEFINERECEIPT <监控列表名>	选择监控列表	命令执行时从配方管理器中选择一个监控列表。 在监控列表中的变量将被监控和显示。
READRECEIPT <监控列表名>	'读取配方'	在定义的监控列表中用当前值替换变量的预先定义值。 注意： 在使用 DEFINERECEIPT 以前必须定义监控列表，并添加 500 ms 延时 (参看上面的 DELAY 命令)！
WRITERECEIPT <监控列表名>	'写入配方'	需要输入配方管理器中监控列表的名称。写入监控列表的配方。不需要先执行 DEFINERECEIPT。
SAVEWATCH	'保存监控列表'	保存在文件中的配方将读入到监控列表中。 注意： 为定义当前的配方而调用一个先前的 DEFINERECEIPT，应延时 500ms (参看上面的 DELAY 命令)！
LOADWATCH	'加载监控列表'+ '写入配方'	打开标准的'文件打开'窗口，从选择保存的配方。选择后，这个配方中的数据立刻写入到控制系统中。
CHANGEUSERLEVEL	-	打开设置用户组对话框。OtoStudio 中有 8 个 用户级别可供选择。
CHANGEPASSWORD	'工程' '用户组密码...'	打开修改用户组密码对话框。

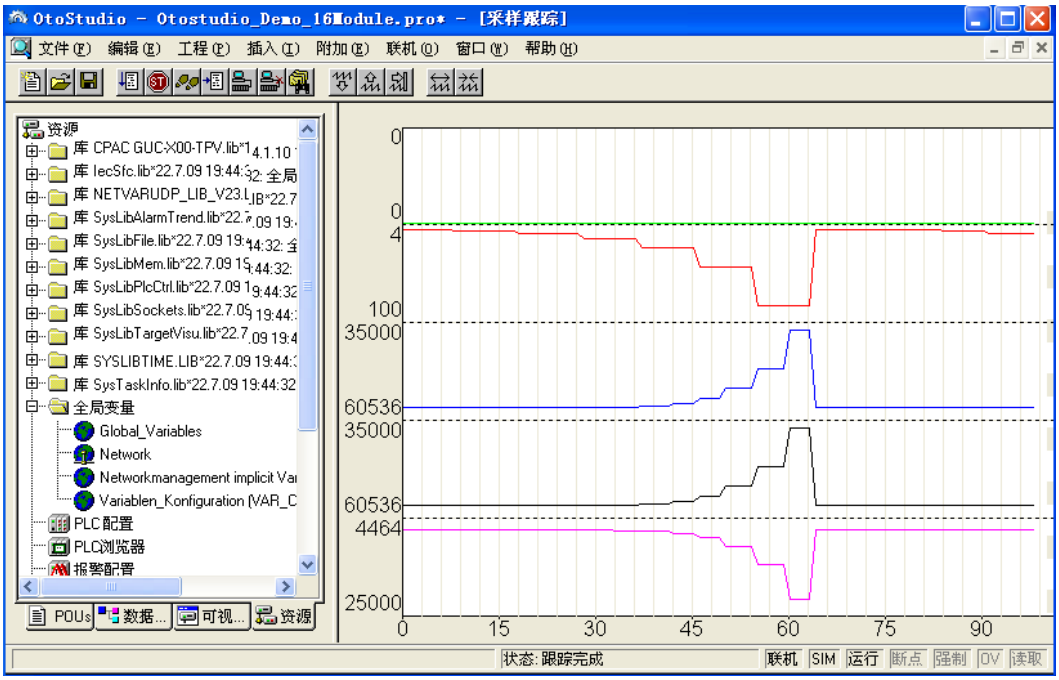
2) 在 Web 可视化下不支持

下面的命令只在 Web 可视化中使用：

INTERN LINK <URL>	Web 可视化切换到定义的 URL 浏览器(Unified resource location：在 Internet 的 WWW 服务程序上用于指定信息位置的表示方法)：
INTERN LINK <HTTP file path>	打开定义的文件： 如 "INTERN LINK http：//localhost：8080/test.pdf"
INTERN LINK mailto：<EMail-Address>	打开发送邮件对话框，邮件地址要先定义好： 如. " INTERN LINK mailto：s.sdfjksk@companyxy.com"
INTERN CONNECT_TO <PLC name>|<Start-Visu>	改变目标 PLC：先决条件： WebServer 必须配置连接目标系统的参数，要有与 PLC-Handler 相匹配的 ini 文件。 PLC 名称：目标 PLC 名称，它在 PLC-Handler 的 ini 文件中定义。 启动的可视化界面： 首先打开的可视化界面名称。 WebServer 将自动实现与各自的 PLC 的连接。

	如： "INTERN CONNECT_TO PLC1|PLC_VISU"

操作下的跟踪记录对话框



2.5 可视化元件，配置

除了配置各个可视化元件外，也可以配置整个可视化对象。这可能涉及到框架、语言、背景、占位符等以及专门的热键定义分配（键盘使用）。这些对一个可视化对象来说都应该是完全有效的。

这些设置是在可视化编辑器中通过‘**附加**’菜单来完成。

除了编辑器外，也可以在可视化对象的**属性**对话框中作某些设置。它涉及到目标可视化，web 可视化的用法及总体布局。

在工程选项中，可以用一个独立的目录来保存可视化文件。

'附加' '设置'

使用“附加”“设置”命令，打开一个对话框，可作出一些影响可视化界面显示和语言以及检查可视化变量的设置。

显示、框架和语言也可以在联机模式中编辑。

1.“显示”选项：将一个在10到500%之间的缩放因子输入到“缩放”区段内，以便放大或缩小可视化界面显示的尺寸。

2.“框架”选项：若选择'**自动滚动**'，那么在拖拉或移动可视化元件并达到图像边缘时，可视化窗口的可见部分将自动移动。若选择'**在联机模式下适合屏幕**'，那么在联机模式下整个可视化界面将全部显示在窗口内，而不考虑窗口的尺寸。当选择'**包含背景位图**'，那么背景位图要适合于窗口，否则只考虑元件的显示。

3.“网格”选项：在这里定义网格点在离线模式中是否可见。可视网格点之间的间隔至少为10，即使输入尺寸小于10也取10。在这种情况下，网格点按输入间隔的倍数显示。若设置

为“激活”，那么在元件被拖移和移动时，元件将被放置在捕捉到的网格点上。在“**间距**”字段中设置网格点的间隔。

4.“编译”选项：在默认情况下，在工程进入联机模式前不检查它们的有效性。如果你想在编译生成工程时检查它们，那么要在编译时激活选项“在编译时检查可视化变量”。在信息窗口中用警告来显示无效的变量。

5.“语言”选项：这里定义用何种语言来显示元件在'**文本**'和'**工具提示**'选项中的文本。此外，用'动态文本'选项还可以动态改变显示的文本。

'附加‘’选择背景位图'

使用这个命令，打开文件选择对话框，选择扩展名为'*.bmp'的文件，选择的文件将作为背景显示在可视化界面中。

可以使用命令'附加‘’取消背景位图'来删除背景位图。

取消背景位图

在背景位图设置好后，如果背景位图不满足需求，选择“取消背景位图”可对其进行删除。

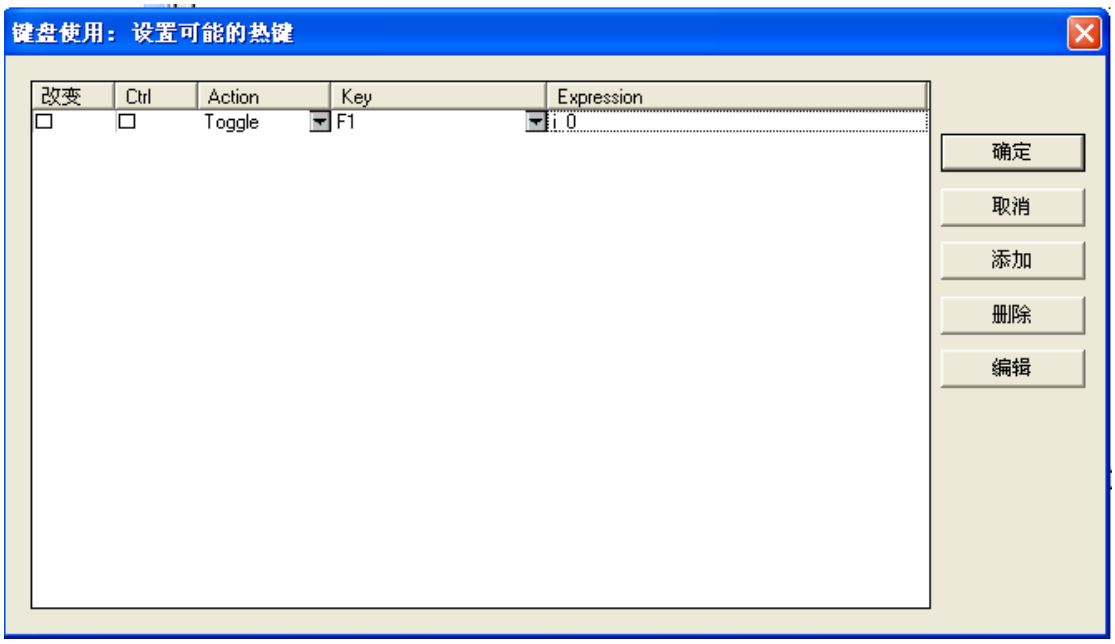
'附加' '使用键盘'

热键可以优化可视化界面中的纯键盘操作。

在可视化界面的配置中可以定义热键，它们可以产生与可视化元件一样的动作。例如，可以定义在联机模式下当可视化界面'xy'激活时，按<ctrl><F2>将停止程序，此动作也可以通过点击可视化界面'xy'中的'z'元件来执行。

总之，在联机模式下，可以使用默认键<Tabulator>， <Space>和 <Enter>来选择或激活可视化界面中的每个元件。

执行'附加‘’使用键盘'或上下文菜单打开热键设置对话框：



在'键(Key)'栏中提供了如下可选键列表：

VK_TAB	Tab-键
VK_RETURN	Enter-键
VK_SPACE	Space 键
VK_ESCAPE	Esc 键
VK_INSERT	Insert 键
VK_DELETE	Delete 键
VK_HOME	Home 键
VK_END	End 键
VK_PRIOR	Bild (↑)键
VK_NEXT	Bild (↓)键
VK_LEFT	左箭头键 (←)
VK_RIGHT	右箭头键 (→)
VK_UP	上箭头键 (↑)
VK_DOWN	下箭头键(↓)
VK_F1-VK_F12	功能键 F1 ～ F12

0-9	数字键 0 ～ 9
A-Z	字母键 A ～ Z
VK_NUMPAD0-VK_NUMPAD9	数字操作板上的键0 ～9
VK_MULTIPLY	数字操作板上的键 *
VK_ADD	数字操作板上的键 +
VK_SUBTRACT	数字操作板上的键 -
VK_DIVIDE	数字操作板上的键 /

在栏'Shift'和'Ctrl'，可以选择<Shift>和/或<Ctrl>键与已选择键组成组合键。

在栏'动作(Action)'中定义当键(组合键)按下时，产生什么动作。从列表中选择所要执行的动作和插入适当的表达式。 参看下面的变量动作和有效的表达式，它们与'输入'选项中的设置相对应：

动作	含义	表达式
Toggle	触发变量	变量， 如"plc_prg.tvar"
Tap true	触发变量但不保持(设置为 TRUE)	程序变量， 如 "plc_prg.svar"
Tap false	触发变量但不保持(设置为 FALSE)	程序变量， 如 "plc_prg.xvar"
Zoom	转到	转到的可视化变量名，如"Visu1"
Exec	执行程序	执行程序名，如 "notepad C：\help.txt" (启动 Notepad 并打开文件 help.txt)
Text	变量'文本显示'的输入文本	配置用于文本输入的元件号，如 "#2" (在'附加‘’设置'中打开显示的元件号： 参看 '元件列表...')

在'表达式(Expression)'栏中，根据动作类型必须变量名，INTERN 命令，可视化界面名称。这要和在'输入'选项配置可视化元件严格一致。

请参考例程：Projects\visu\EnhancedVisuFeatures.pro

使用“**添加**”键可以在表格的末端添加一个空行。使用“**删除**”键可以删除光标所在行。'**确认**'和'**取消**'将保存或不保存设置并关闭对话框。

可以为每一个可视化对象单独设置键盘的使用方法。所以，相同的按键（或者组合键）可以在不同的可视化对象中启动不同的动作。

用于动态文本的 XML 文件

文件必须使用 XML 文件格式（即文件名.XML）。在这种文件中，文本被赋予一个标识符（这个标识符是一个前缀和一个 ID 号的组合）。为了在联机模式时可以显示各自的文本，这些组合将输入到一个可视化元件的配置内。即前缀输入到'文本'选项的'内容'字段，ID 输入到'变量'选项的'文本显示'字段。

在文件的开头部分，定义了默认的语言和字体。可参看本页末尾说明文件中的举例。

在 XML 文件中描述部分必须放在<dynamic-text>和<\dynamic-text>之间，即以<dynamic-text>开头，以<\dynamic-text>结束。

动态文本的语言文件（从 OtoStudio V2.0开始）可以由 Unicode (UTF-16) 或 ANSI (ISO-8859-1)代码构成。可以通过位于 xml 文件开头部分的编码进行定义。

请注意：

xml 文件主格式不能用于<dynamic-text>\<\dynamic-text>和文件开头，但以后会支持的.

目标可视化提供用于扫描动态文本输入的接口。因而可以直接用于程序。

文件结构：

文件的开头部分以<header>开始，以<\header>结束。如果你希望定义一个默认的语言文字，可以使用输入项<default-language>，可以使用输入项<default-font>为语言文字定义默认的字体。这些输入项都是可选的。如果某个输入项丢失，可以根据可视化配置文件进行显示。

<header>	
<default-language><language></default-language>	默认语言文字： 如果没有用于当前语言文字的文本输入项，那么使用此输入项的默认语言文字.如果默认语言文字也没找到，将显示"<PREFIX> < ID>. 如果使用多个 XML 文件，从而提供了多个开头部分定义，那么只使用最后一个.所以建议使用一个开头部分定义!语言符号必须符合文本输入项中使用的(参看下面). 注意： 在联机模式下，默认语言文字可以借助可视化元件设置，此可视化元件使用 '输入'，执行程序配置了命令 INTERN LANGUAGE DEFAULT.
<default-font>	用于<language>的默认字体：设置的字体(如 "Arial") 将自动用于所有元件，用<language>定义的语言文字显示动态文本. 语言符号必须符合文本输入项中使用的(参看下面).
<language><language></language>	
<font-name><font></font-name>	
</default-font>	
<default-font>	用于其它语言文字的默认字体.
<language>.....	
....	
</default-font>	
</header>	

PREFIX-ID 组合输入的文本列表必须以<text list>开始，以</text list>结束.特殊的文本输

入可以单独以<text prefix>开始以<\text>结束。请参考 Projects\ChineseText\ ChineseText.pro

主界面

在可视化中“主界面”的用处就是提供一个可用于不同可视化界面的对话框，而不需要为每一个可视化界面都插入这个对话框。通过在主界面配置中定义变量来控制在联机模式下对话框何时显示或是否显示。

作为主界面

如果一个可视化界面被定义为“主界面”，这个界面就被自动地插入到其他的所有可视化界面(那些没有被排除在外的)中，并且在联机模式下，可以使用其所有的功能。这个界面总是处于前面。如果你希望其处于后面，就要激活可视化元件属性对话框中的“背景”选项（参看以下的说明）。主界面不能在所插入的可视化界面中进行编辑。只有在主界面上对主界面进行修改。

通过属性对话框中的'工程' '对象' '属性'选项可以定义一个可视化界面作为主界面使用，即激活'主界面'选项。如果事先已经有另一个可视化界面被定义为主界面，那么那个可视化界面就自动重新定义为“标准”的可视化界面（在属性对话框中的'**可视化界面**'选项）。

不带主界面的可视化界面

在属性对话框中，也可以将一个可视化界面定义为一个**不带主界面的可视化界面**。

作为 web 可视化或目标可视化

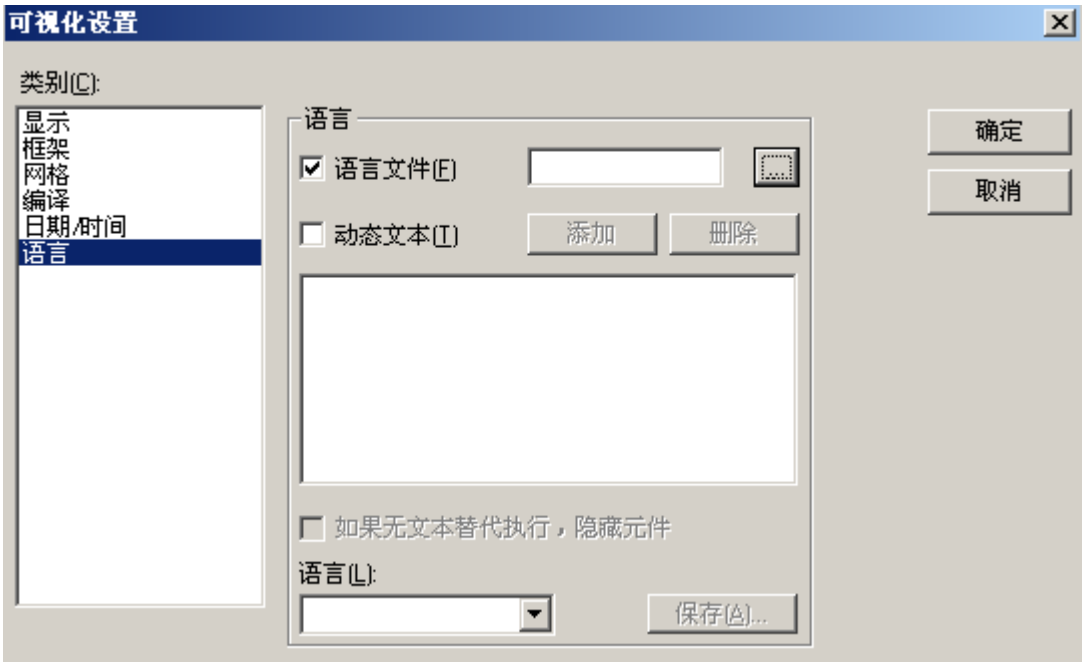
如果一个工程将用于 web 或目标可视化，那么每一个可视化都可以定义是否作为 web 或目标可视化使用。

在项目管理中可以选择可视化项目，也可以打开属性对话框（即'工程' 对象' '属性' ）。如果在目标设置中选中 web 或目标可视化，那么在属性对话框中与 web 可视化或目标可视化相应的选项也被激活.

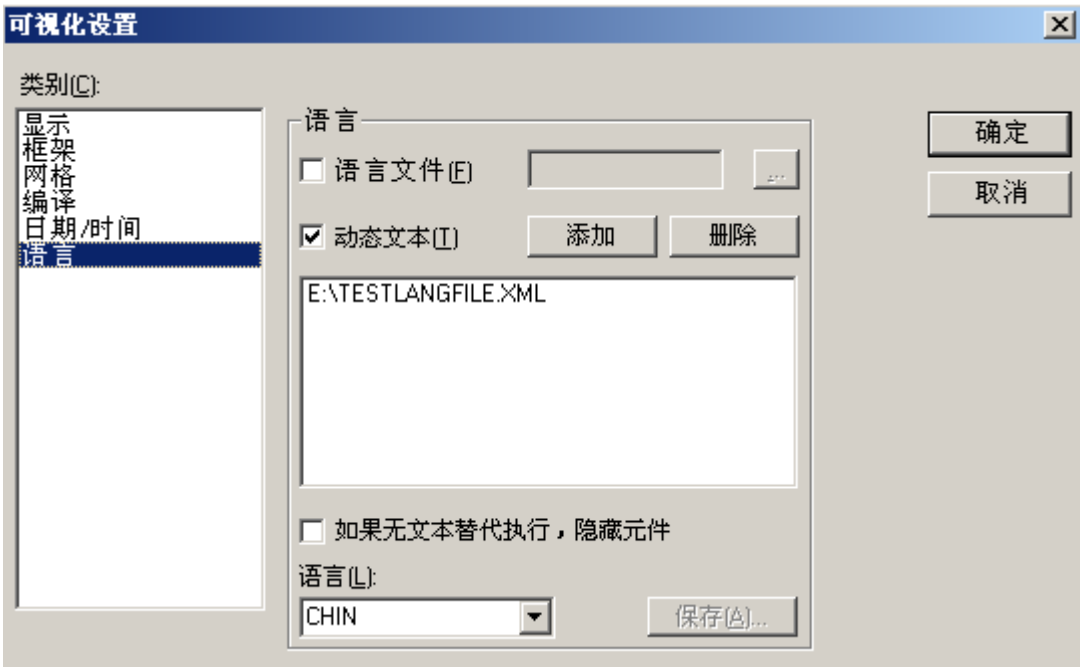
第三章 语言转换

3.1 语言设置

当开启选择功能时（选中'语言文件'，即小窗口打√），通过选择右边按钮，可以从程序的安装目录中添加相应的显示语言。然后在下方的下拉菜单中进行选择。其中的动态文本选项，用于在显示语言中插入用户定义的 XML 格式的动态文本。



注意：XML 文件的路径必须是英文的，否则会提示无法加载。请见谅



可以借助提供的**静态**或者**动态**文本文件实现可视化界面中文本语言的转换。但 Unicode 格式只能进行动态转换。

如何进行语言转换：

在可视化对话框中下部的'语言'选项中选择在语言文件中定义的一种语言作为开始的界面语言。例如为 chinese（中文）或 english（英语）。

借助可视化元件可以在线模式下进行语言转换。可以使用内部命令 “INTERN LANGUAGE <language>" and “INTERN LANGUAGEDIALOG”进行（可参看“用于操作的特殊输入方法”中的内容）。输入方式可以使用配置对话框中的‘输入’选项。

例如：

你可以在可视化界面中插入一个按钮元件，用于转换到德语。为了实现这个目的，用'chin'标记元件，在配置的‘Input’选项中激活'执行程序’选项，并定义指令“INTERN LANGUAGE <language>”指令。'language'将被在语言文件中定义的语言替换。如果在联机模式下按这个按钮，可视化字符串将根据语言文件中应用的中文进行显示。

3.2 静态语言转换

静态语言转换需要使用使用一个语言文件（格式为*.vis， *.tlt， *.txt），如何生成一个语言文件可参看以下内容。与动态语言转换不同的是在实时过程中不能使用项目变量进行语言定义。

对于可视化进程，建议使用*.vis 格式的语言文件，因为*.tlt 文件和*.txt 格式的转换文件只能用于 OtoStudio 平台和 OtoStudio HMI，而不能在'仪表'，'条状图'和'柱状图'元件中使用。

在“可视化设置”对话框中，你可以对项目所使用的语言文件进行配置。为了选择转换文件（*.tlt 或*.txt 文件），或者一个纯的包含了各种远字符的可视化语言文件（*.vis），需要激活语言文件选项，并且在输入域内输入相应的文件路径（可以是中文路径）。使用按钮就可以得到标准的对话框并打开文件。

用于一个可视化界面语言选择的对话框

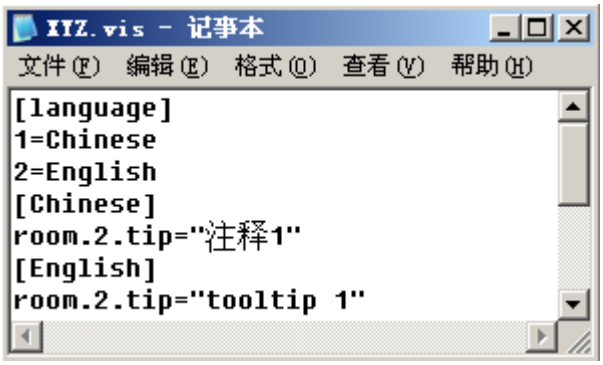
生成一个可视化进程专用的语言文件（*.vis 格式文件），应执行以下步骤：

为了可视化进程而建立一个新的*.vis 语言文件，要：

打开可视化设置对话框中的语言选项；

选择语言文件。在相关的输入域内输入你需要存储的文件名，扩展名应为.vis。

在输入域中填写可视化界面中需要的语言名（例如 chinese 或者 C），然后按动存储按钮。这样一个以.vis 为后缀的文件就产生了。这个文件可以使用标准的文件编辑器进行编辑。例如你可以使用 Windows 中的记事本工具打开一个语言文件。请看这里的例子。



可以为当前正在使用的可视化界面建立字符变量表。这个字符变量表中包括了表头的参考数字，例如“1=chinese”可以作为表头[chinese]的参考数字。可以采用拷贝整行的方法来输入这个变量表，然后再用英文表头取代德文表头，并生成一个新的表头 [english]。于是，除了1=chinese 这一行内容外，又添加了2=english 的新行。为了以一个新的语言中观察可视化界面，需要再次打开语言对话框。现在在选择域中就可以选择中文或者英文了。

3.3 动态语言转换

动态语言可以借助可视化元件在不同语言文本版本之间进行转换（包括文字、工具条中的文字和报警表中的警告信息）。与静态转换不同的是可以借助应用程序中的变量进行文本选择。

在可视化元件的配置中，输入一个带前缀的 ID 组合码，这个组合码与 XML 文件中的文本相对应（下面命名为“textlist”）。这个 ID 码可以由工程变量进行定义。

应用实例：ID 表示一个错误号，例如使用“Error”作为前缀。借助前缀 ID 组合码，语言文件提供相应的错误信息，可以依据当前所选择的语言进行显示。

- 用于动态字符的语言文件可以有 Unicode (UTF-16)或者 ANSI (ISO-8859-1)两种格式。即<?xml version='1.0' encoding='UTF-16'?>。

- 对于目标可视化，开始时使用的语言，xml 文件的路径以及 xml 文件表等都可以在目标系统中进行定义。这说明不需要生成一个新的引导工程就可以修改这些参数。所以，可以有一个容易的方法去修改已经存在的工具条，添加新的语言。如果目标系统可以提供这样的配置，在 OtoStudio 的可视化进程中定义的工具条就可以不论是否在在线模式都可以使

用。如果没有为语言转换专门使用的目标配置，对 OtoStudio 中定义的工具条的修改之后，就必须下载一个项目文件。

3.4 对动态语言转换的配置

在联机模式下，用户可以利用前缀 ID 组合码动态控制在可视化元件中显示的文本，因为前缀 ID 组合码指向 XML 文件中定义的文本.

为了实现这个目的，在可视化配置中，应将用于描述文本赋值的 XML 文件与工程连接在一起。这个 XML 文件具有确定的格式。语言代码添加到特定的字符串内，因而随后用户不仅可以在不同的文本内容之间进行转换，而且也支持语言的转换。

在可视化元件的配置中，对于哪些文本显示可以实现动态转换，可以输入前缀和 ID 号（参看以下的内容），ID 号可以由工程变量提供。通过 INTERN 指令 (LANGUAGE DEFAULT).就可以定义默认的语言。

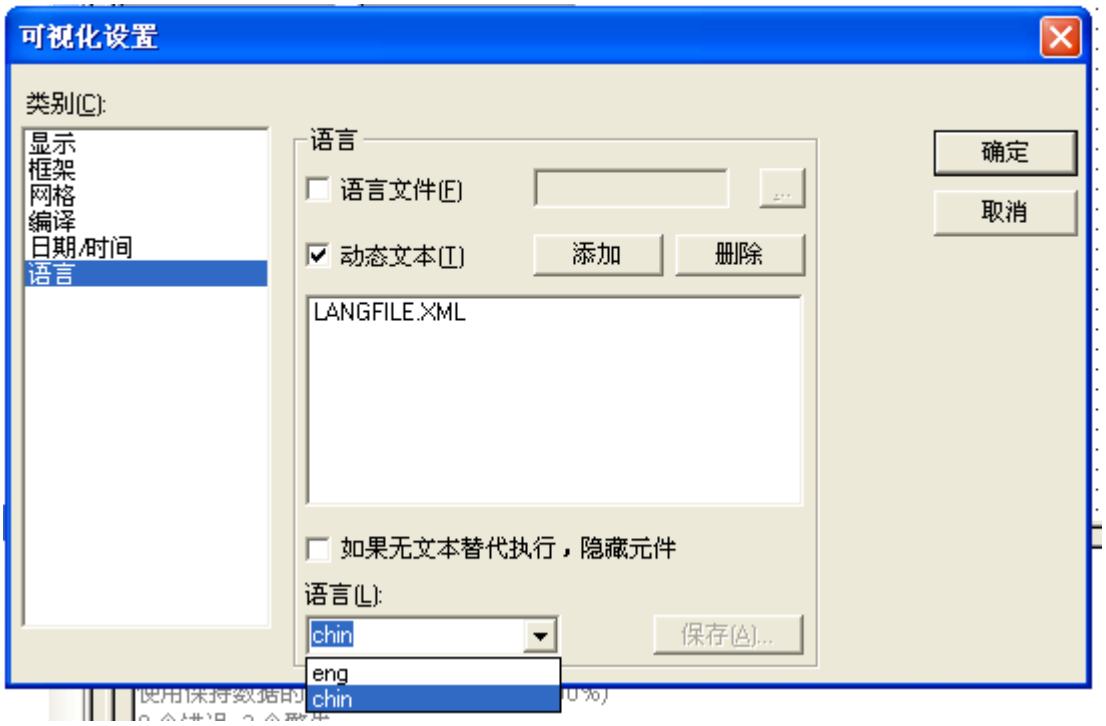
因此，应在一个可视化界面的不同配置对话框内做如下设置：

1. 连接 XML 文件，选择开始时的语言：“设置”对话框中的“语言”选项：

为了将一个或一些可以应用于你的系统的 XML 文件连接到工程上，需要激活“**动态文本**”选项，并按“**添加**”按钮。可选文件将列表显示在按钮下方的窗口内。如果要从列表中删除一个选定的文件，可以按“删除”按钮。如果你只希望显示经过动态变换的可视化元件，那么可以激活“**如果无文本替代执行，隐藏元件**”选项。

选择一个在“语言”字段中列出的语言识别符，就可以显示这些文本的版本（即相应的前缀 ID 组合），这个版本是由 XML 文件中的语言标识符标记的。

用于配置动态文本的“设置”对话框，“语言”选项



2. 在选项“变量”的字段“文本显示”中定义 ID 码（在 XML 文件中使用的）：

输入一个数字，它对应定义了文本 ID 的工程变量（在 XML 文件中使用的）。

一个报警表中的信息字符，其 ID 码应与表中相应的行数相一致（参看此处的图形）。

3. 在选项“文本”中定义文本格式：

在内容字段处，插入一个占位符“%<PREFIX>”，用于在联机模式下显示动态文本。为了替换这个“PREFIX”，你可以插入任何一个与 XML 文件中 PREFIX 定义相匹配的字符串。请参看有关配置对话框“文本”的说明。

对于每一个可以在 XML 文件找到的前缀 ID 组合码，所赋予的文本都在联机模式下，被显示在可视化元件中。如果没有找到相应的输入项，就不做替换。

第四章 在线模式的可视化

4.1 在线模式的可视化

应注意与在线模式可视化有关的以下内容：

- 评估的顺序：

动态定义的元件属性通过变量将对在配置对话框中由“选项”定义的基本设置（即静态设置）进行改写。

如果一个元件属性由一个“标准的”工程变量进行定义，同时又由一个结构变量组进行定义，那么，工程变量的数值将首先采用。

可视化在线配置的途径可以是单独通过键盘进行输入。对于使用 OtoStudio 的可视化作为目标。或者作为 web 可视化的时候，键盘输入是一个特别重要的的功能。

- 在下载工程之后，应关注当前目标系统设置中'禁止当前可视化界面文件下载'选项的当前设置情况。这个设置与当前可视化进程中使用的所有文件都有关系。所有用于目标或 web 可视化的可视化文件都要下载，包括位图文件、语言文件和用于网站可视化的 XML 说明文件。

- “显示， 框架和语言”也可以在在线模式下进行编辑。

- 只要一个可视化界面的“实例(引用)”没有进行详细的配置，在在线模式中这个“实例(引用)”的特定元件将会起作用，就象它们在原有的可视化界面中一样。

- 当进行语言转换时（即选用'附加' '设置'时），这些选择只对在线模式的显示有效。

- 可以在在线模式下，进行可视化界面的打印。

- 如果一个可视化界面被作为目标可视化使用，用户通过鼠标点击输入的信息就可以作为专用接口的帮助程序，在工程中使用。

用户可以在一些在线使用的可视化元素中找到各种相关说明，可以作为相应元素配置的帮助页面。

4.2 在线模式下的键盘操作

为了分别从鼠标或者触摸屏获取信息，单纯使用键盘进行可视化的配置是很有用的：

在线模式下，每次按动以下按键（或者组合键）就可以进行以下的操作：

• 按动<Tabulator>键，可以选择经过配置的输入元件表中的第一个元件。每按动一次这个按键，都可以下移到元件表中的下一个元件上。在元件表中可以跳转到下一个表字段。在按住<Shift>键的同时按动这个按键可以选择前边的元件。根据目标的不同，也可以选择简单的输入处理。

• 箭头键可以用于改变所选定的元件，即根据箭头方向移动到当前选定元件旁边的元件上。

• <Space bar>键可以用于激活所选定的可视化元件。如果这个元件带有输出变量，或者是一个表字段，就可以打开一个文字输入框，在框中显示当前变量的文字内容。按<Enter>键确认写入的数值。

在配置对话框“使用键盘”中可以定义用于联机操作的附加键（即组合键）。除了上边说过 的标准按键，也可以将<Tab>、<Space>和<Enter>等按键赋予其他功能。

在线模式中引用的各个元件的动作都可以与被引用的可视化元件相对应。所以这些元件对于鼠标或键盘的输入或操作具有相同的反应。在引用中提示工具条的显示内容依赖于元件。在处理元件表时，例如在使用<tab>键，从一个输入元件移动到另一个输入元件时，在移动到列表中的下一个元件前，参考实例中所有元件的处理从列表中的参考位置继续进行。

如果在 web 可视化或者目标可视化中使用 OtoStudio HMI 那么在线模式下使用键盘具有重要的意义。在 web 可视化进程中，如果当前打开了一个输入框，在 web 中进行设置，使用键盘可以非常有效。

4.3 在线模式下的'文件' '打印'

在线模式下，'文件' '打印'指令可以用于打印可视化窗口的内容。但在可视化窗口中有移

动元件时，可视化窗口边界的变化可能产生前后不一致的情况。

第五章 库文件的可视化

库文件也可以存储可视化进程，因而也可以使用库 POU 中的工程。库文件可以如同引用那样进行插入，使用“转到”指令进行调用。

 注意	**在一个工程中使用的可视化进程应具有单独的名称。如果出现从库文件中调用的可视化进程在一个工程中具有相同的名称，就可能出问题。因为在处理程序的引入或者调用的可视化进程时，只处理第一个可视化进程。**

第六章 可视化中的系统变量

隐含变量

以下由系统生成的隐含变量可以在程序可视化中使用：

隐含变量名称	数据类型	功能	可用的位置			
			HMI	仿真	目标可视化	Web可视化
CurrentVisu	String[40]	当前打开的可视化界面. 如果名称被改变,对应的另一个可视化界面也将改变!. 请注意名称字符串必须用大写字母. 根据目标系统设置,这个变量可在目标系统设置中的可视化选项激活或不激活.	X	X	X	X
CurrentCaller	String[40]	先前打开的可视化界面的名称.被用于ZOOMTOCALLER功能. 只能在目标可视化中设置和修改.	-	-	X	-
CurrentLanguage	String[40]	当前设置的可在语言文件中获得的语言.语言必须用大写字母. 只能在目标可视化中设置和修改.	-	-	X	-
CurrentUserLevel *	INT	当前设置的用户级别 0..7.	X	X	X	X
CurrentPasswords[0 .. 7] *	ARRAY [0..7] of String[20]	在用户组密码中定义的所有密码	X	X	X	X

作为保持变量使用的隐含变量：

一个目标可视化进程中的隐含变量可以声明为保持变量：

为了这个目的，变量应先声明为全局变量。此项声明必须资源选项中最上面的'全局变量'文件夹中进行（这个变量表是字母排序的）。如果声明在其他的全局变量表中进行，就会出现编译错误。

例如：

VAR_GLOBAL RETAIN

VisuDoExecuteUserlevelInit ： BOOL ： = TRUE：

CurrentUserLevel ： INT ： = 0：

CurrentPasswords ： ARRAY[0..7] OF STRING[20] ： = 'a'，'b'，'c'，'d'，'e'，'f'，'g'，'h'：

END_VAR

* 请注意变量 CurrentUserLevel 和 CurrentPasswords。这些变量必须具有相同的类型（如 normal、RETAIN、PERSISTENT 等）！如果他们被定义为保持变量，另一个类型为 BOOL 的变量 VisuDoExecuteUserlevelInit（其初始值应为 TRUE）就必须在全局变量表中公告为 RETAIN 型的变量。

CPAC-**OtoStudio** 用户编程实例手册

示例：机械臂运动

说明：一个机器操作工正在监视一台正在运行的机器，正确的运行必须是在规定的时间间隔内完成的，如果超出运行时间就会产生一个警告，一会儿电机停止运行。

机器的动作：手臂沿着一个矩形路径运动，每完成一周计数器加一。

1 启动 OtoStudio 编程系统

启动 OtoStudio 编程系统：(安装目录下)

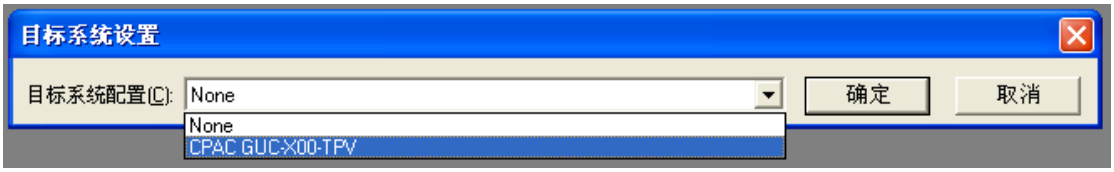
开始-> 所有程序-> Googol -> CPAC Platform ->OtoStudio

2 新建一个项目

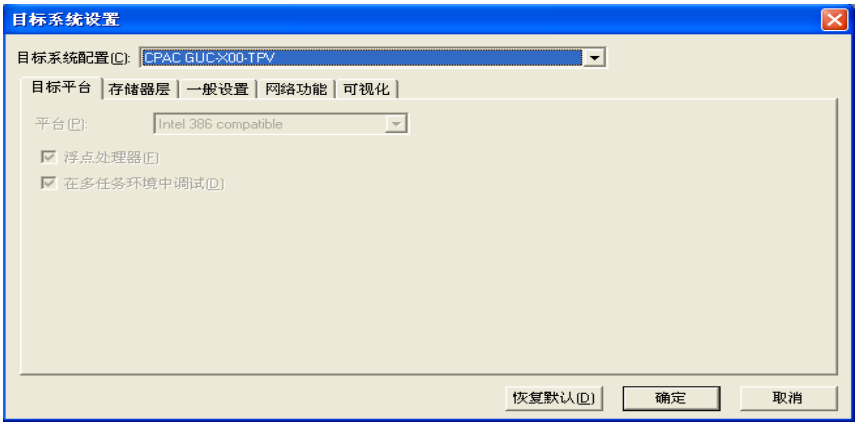
点击**文件->新建（或者点击**  **图标）**

3 目标系统设置

在新建了一个项目之后会自动弹出目标系统设置对话框。该对话框用于选择相应的控制平台。

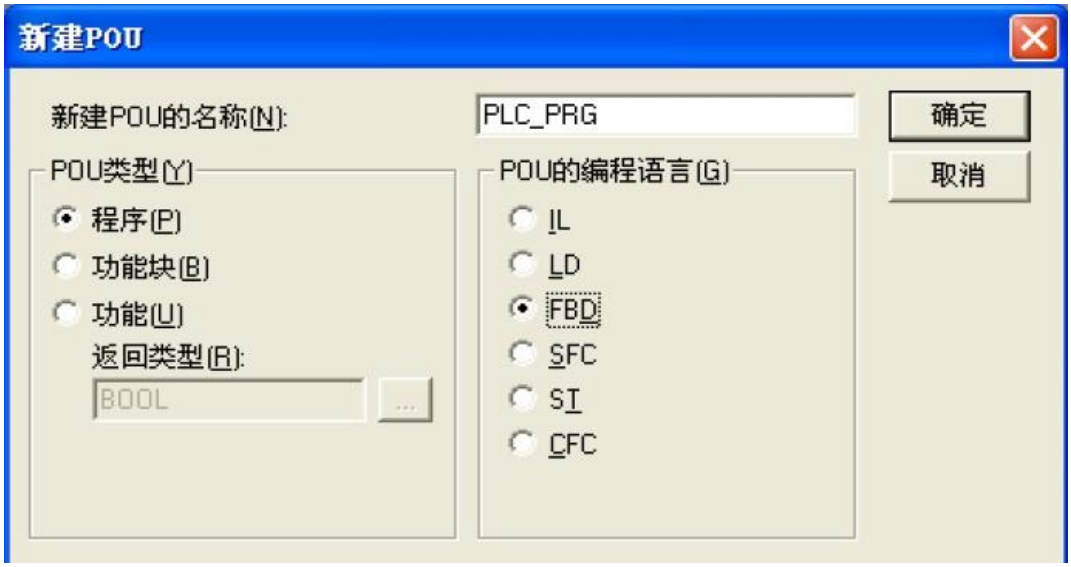


我们选择固高的控制平台 CPAC GUC-X00-TPV 控制器，同时进行相应的设置（无特殊要求”确定”默认设置即可）。



4 PLC_PRG POU

在新建对话框中有两部分的选择：1、POU 的类型的选择（程序、功能块、功能）；2、编程语言的选择（IL、LD、FBD、SFC、ST、CFC）。这里我们选程序类型为程序，编程语言为 FBD，POU 名为 PLC_PRG(类似于 C 语言的主程序，整个程序将从此入口开始运行，其他的 POU 都在这里被调用)，如下图



5 程序的编写

声明确认开关

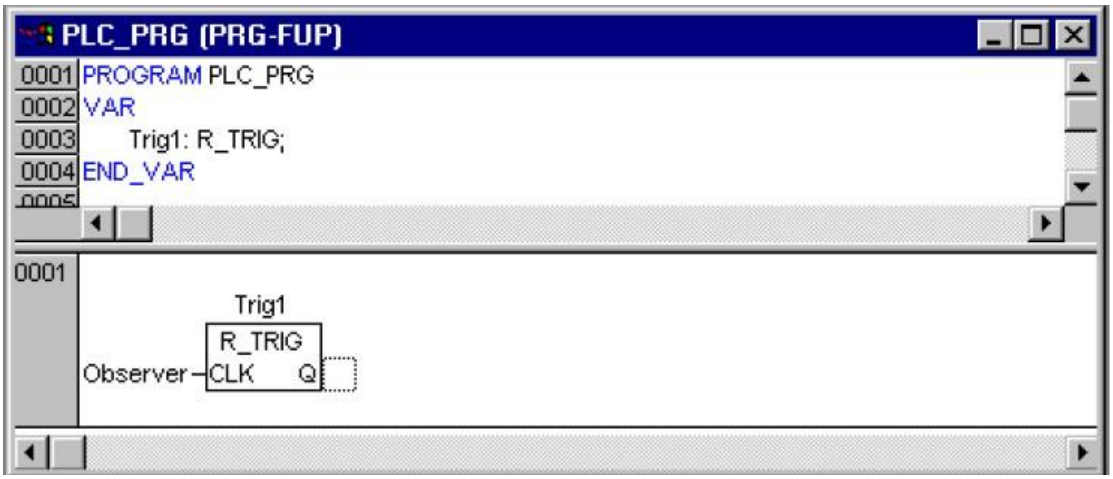
我们从确认开关开始。可以看到第一个网络中有三个???，输入开关的名称，例如 observer，鼠标右击或按回车键将自动弹出声明变量对话框。

将**类别**改成 VAR_GLOBAL(定义成全局变量)。点击**确定**，在**资源->全局变量->VAR_GLOBAL** 中将可以看到下列定义：

```
VAR_GLOBAL
    Observer: BOOL;
END_VAR
```

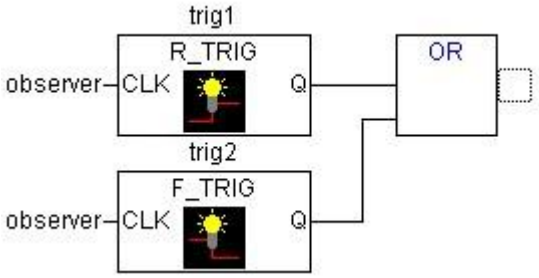
确认开关的上升沿

在 Observer 变量后点击鼠标左键，将会出现小正方形。通过鼠标右键执行框命令，将插入一个带 AND 操作符的框，点击选中 AND 后，按 F2（输入助手）打开一个包含可选操作符的对话框，首先选择“标准功能块”项，然后选择 standard.lib 中的 R_TRIG(上升沿触发器)。此时一个 R_TRIG 实例被创建，然后把出现在 R_TRIG 框上面的???改一个名字，例如 Trig1。之后回车会弹出声明变量对话框，在类别，名字，类型中已经分别输入 VAR（局部变量）、Trig1 和 R_TRIG。按确认后变量被写到此 POU 的声明部分。



确认开关的下降沿

在功能块后点击出现小正方形，通过快捷菜单执行框命令，将 AND 改成 OR（逻辑或）；点击 OR 框的第二个输入插入 F_TRIG（下降沿触发器）框，声明实例名为 Trig2。点击 Trig2 功能块前的???，按 F2 键打开输入助手对话框，在全局变量选项中选择 Observer。



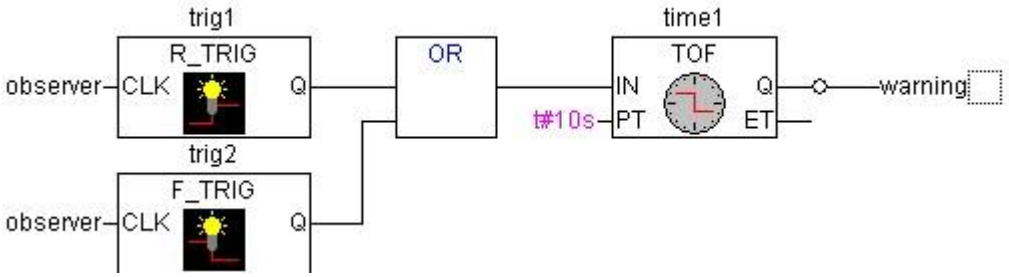
第一个时间控制

在 OR 功能块后插入 TOF（延时闭合）功能块，命名为 timer1。在 PT 输入端将三个???替换成 T#10S（延时 10 秒）。

发出 Warning 信号

在 timer1 功能块的 Q 后面鼠标右击插入赋值。将???改为 Warning。在变量声明中将它设置成类别 VAR_GLOBAL 和 BOOL 类型。

为了使 Warning 正确执行，使用快捷菜单在 Warning 前插入取反命令，它使布尔型变量的输出取反，取反用小圆圈表示。

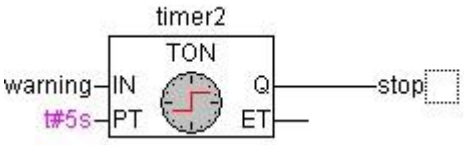


在超出第二个时间限制后设置停止信号

用菜单命令在当前行后插入一个新网络。

在这个网络中添加类型为 TON（延时打开功能块）的框，声明实例名为 timer2。

使用 F2 键将变量 Warning 分配给 TON 的 IN 输入端，然后将时间常量 T#5S 分配给 PT 输入端。Timer2 功能块后面使用赋值命令，将 TON 的 Q 输出赋值到变量 Stop(类别 VAR_GLOBAL，类型 BOOL)。



新建名为 Machine 的 POU

在对象管理器中的 POUS 选项页面下，点击鼠标右键执行添加对象命令新建一个 POU，命名为 Machine，类型为程序，编程语言选 SFC（顺序功能图）。

新建的 SFC 由步“Init”,转换“Trans0”和跳转回“Init”组成。

定义机器的运动顺序

机器操作的每个阶段都需要一步。

点击转换 Trans0 后 Trans0 四周出现一个矩形框，借助快捷菜单执行命令步-转换（插入在当前行后）。此命令执行五次。如果直接点击在步或转换的名称上，它们将用蓝色标记，可以改变它们的名称。在 Init 后面的步骤依次命名为 go_right,go_down,go_left,go_up 和 count。

编写 go_right 步中的程序

双击 go_right 步后弹出选择编程语言对话框，选择 ST（结构化文本）编程语言，按“确定”后弹出一个程序编辑窗口。

机械臂沿 X 方向，程序如下：x_pos:=x_pos+1;

输入完成后按回车键，声明变量 x_pos 的类型为 INT 型。

步的右上角出现的小三角表示此步中已经写有程序。

编写后续步

重复上述步骤，声明变量 y_pos 和 counter 的类型为 INT。

在 go_down 步中程序：y_pos:=y_pos+1;

在 go_left 步中程序：x_pos:=x_pos-1;

在 go_up 步中程序：y_pos:=y_pos-1;

在 count 步中程序：counter:=counter+1;

编写转换条件

转换条件是程序从一个阶段转到下一个阶段运行的条件。将 Init 后面的转换条件 Tran0

改为 Start,类别为 VAR_GLOBAL，类型是 BOOL。当 start 开关按下时机器开始运行。

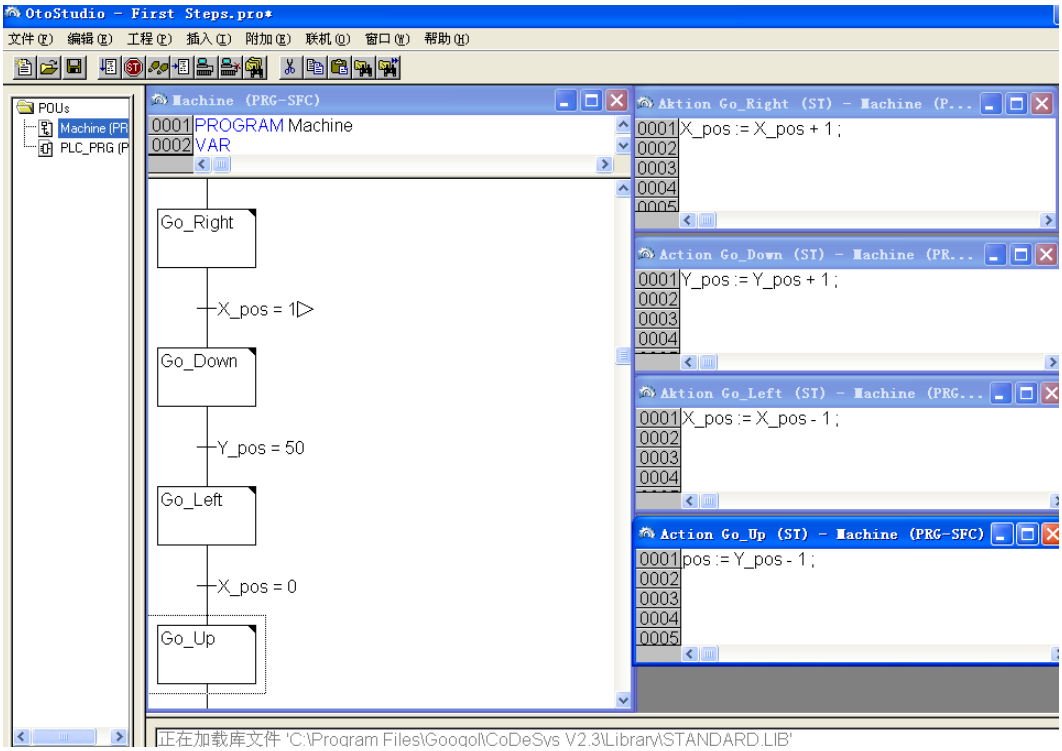
第二个转换条件是 x_pos=100，即当 X 位置到达 100 时转到下一个阶段运行；

第三个转换条件是 y_pos=50；

第四个转换条件是 x_pos=0；

第五个转换条件是 y_pos=0；

第六个转换条件是 TRUE（一次循环结束后继续运行，表示程序循环运行）。



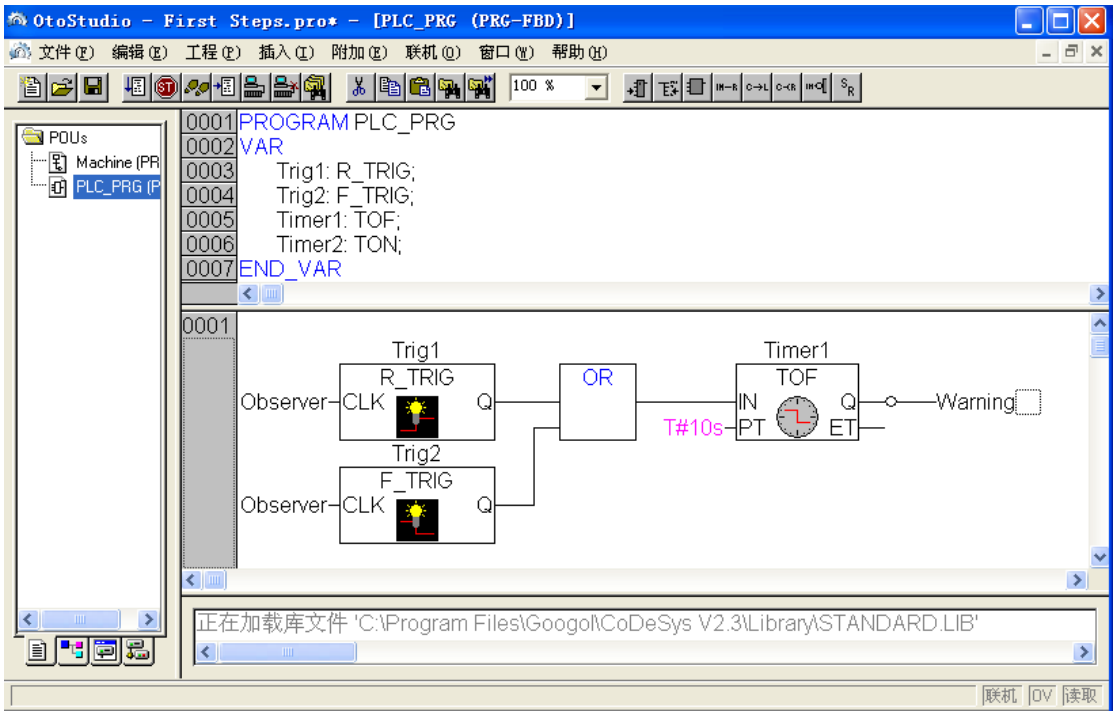
在停止时的处理

返回到 PLC_PRG POU，插入第三个网络。

用变量 stop 替换???，通过快捷菜单插入放回命令。当 stop 为 TURE 时执行返回命令将退出 PLC_PRG POU。

调用 machine POU

添加一个新网络，使用快捷菜单插入一个框，按 F2 键打开输入助手对话框，在用户定义程序选项中选择 machine POU。完整的程序如下：



编译生成工程

使用菜单工程->全部重新编译生成或 F11 功能键编译工程。编译生成后在信息窗口的右下角显示“0 错误 0 警告”。如果有错误，根据错误提示修改错误。

6 可视化界面的编写

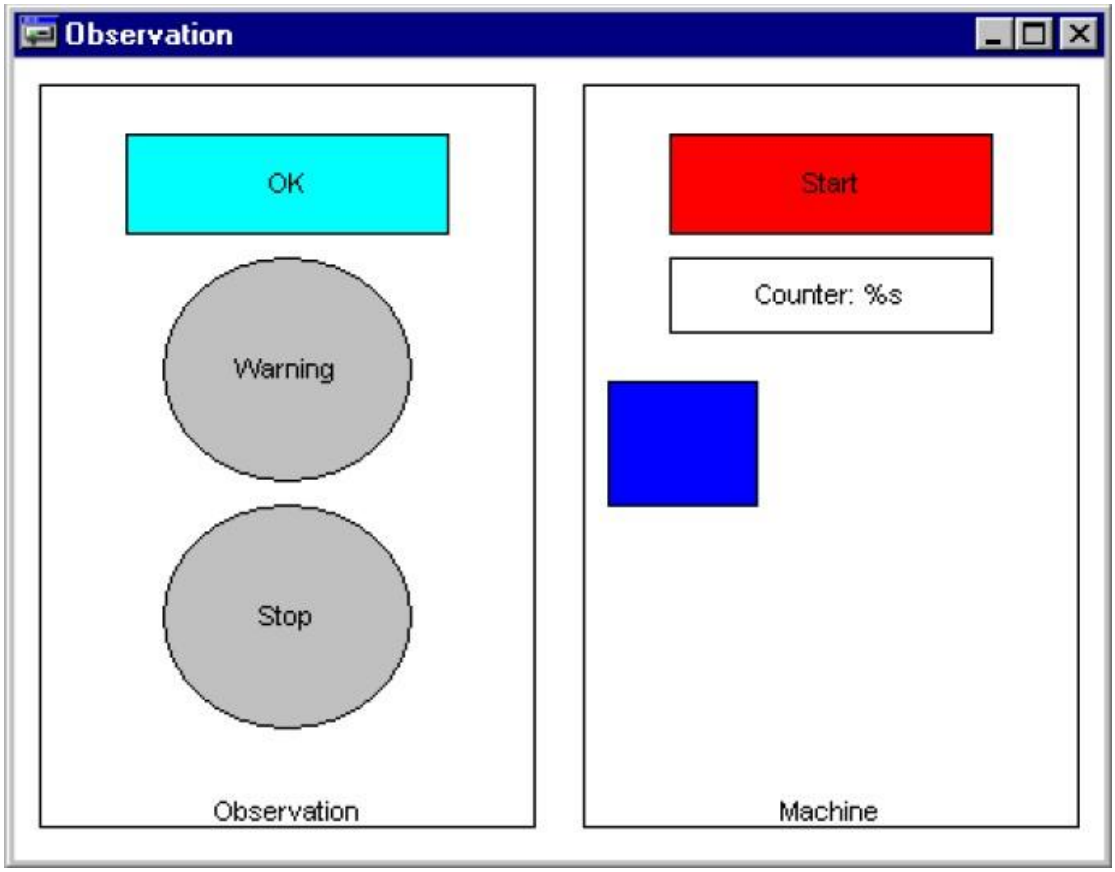
创建可视化见面

在选择对象管理器中左下角选择“可视化界面”

使用对象管理器中的快捷菜单命令添加对象。

给可视化对象命名，如 Observation。

完整的可视化界面如下所示：



添加可视化界面中的元件

首先在工具栏中选择矩形元件。

在可视化编辑器中按住鼠标左键拖拽一个矩形。

配置第一个可视化元件

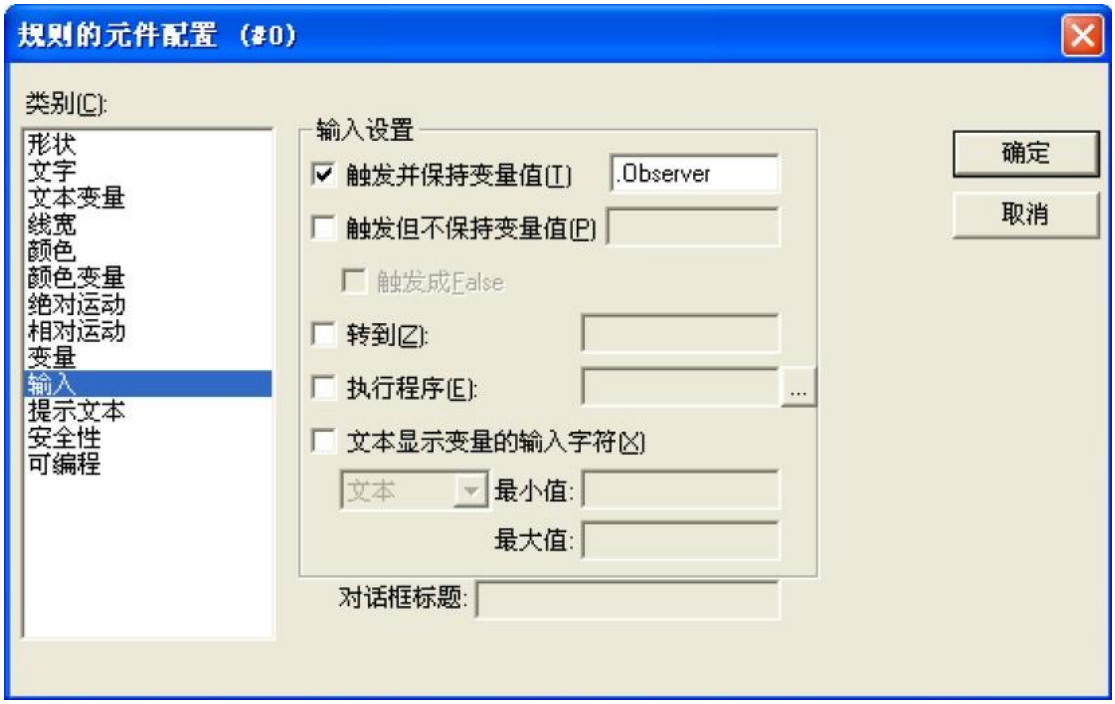
在矩形上双击鼠标打开配置对话框。

在文字选项内容字段中输入 OK。

选择变量选项，在改变颜色字段中点击鼠标，然后按 F2 键打开输入助手对话框，在对话框中的 GLOBAL_Variables 上双击，将列出所有全局变量，选择 Observer 变量，则在字段中显示**.Observer**。

选择颜色选项，点击颜色下的内部按钮选择一种颜色浅兰色，点击报警颜色下的内部颜色选择一种颜色蓝色。

在输入选项中，选中“触发并保持变量值”，使用 F2 功能键在后面的输入项中输入变量**.Observer**。



经过上述设置，在程序运行过程中当 Observer 变量为 FALSE 时矩形的颜色是浅蓝色，当 Observer 变量是 TURE 时，矩形的颜色为蓝色。点击一下矩形，Observer 变量从 TURE 变为 FALSE，再点击一次 Observer 变量从 FALSE 变为 TURE。

添加其他可视化元件

画一个圈，作如下配置：

文字选项，内容字段中输入 **Warning**。

变量选项，改变颜色字段中输入**.Warning**。

颜色选项，“**颜色**”“**内部**”设置成灰色，“**报警颜色**”“**内部颜色**”为红色。

复制并粘贴一个新圆，修改下面的配置：

文字选项，内容字段中输入 **Stop**。

变量选项，改变颜色字段中输入**.Stop**。

画一个矩形，用于机器启动，并作如下配置：

文字选项，内容字段中输入 **Start**。

变量选项，改变颜色字段中输入**.Start**。

在输入选项中，选中“触发并保持变量值”，使用 F2 功能键在后面的输入项中输入变量**.Start。**

“报警颜色”“内部颜色”为绿色**。**

画一个矩形，用于计数器，并作如下配置：

文字选项，内容字段中输入：**%s(表示变量值的占位符)**

变量选项，文本显示字段中输入 **Maching.Counter**

画一个矩形，用于表示机器运动，并作如下配置：

绝对运动选项，X-偏移量字段中输入 **Maching.X_pos。**

绝对运动选项，Y-偏移量字段中输入 **Maching.Y_pos。**

颜色选项**，“颜色”“内部”**设置成蓝色**。**

也可以画两个大矩形框，在文字选项，内容字段中分别输入 **Observation** 和 **Machine。**同时选中这两个矩形，使用快捷菜单中的命令**“对齐”“底部”**以底部为基准对齐它们，并执行命令**“置于后面”**将它们放在其他元件后面显示。

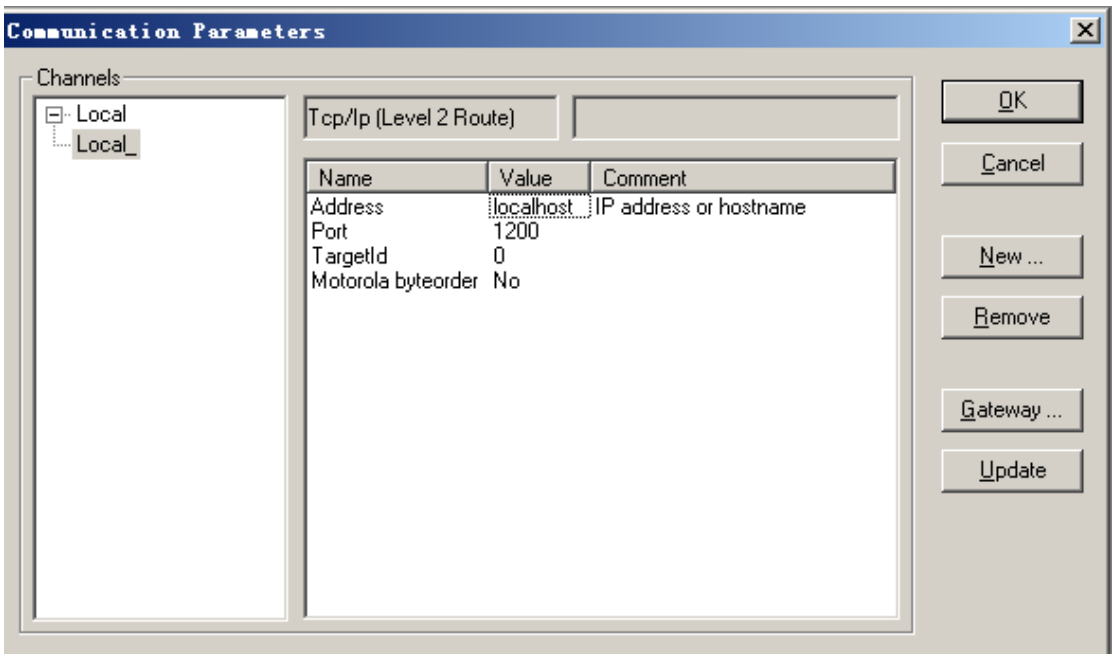
下面 7、8、9 步只有计算机上安装了运行系统时才被运行。运行系统要与 OtoStudio 中**“目标系统设置”**中的一致。否则程序只能运行于仿真模式，见第 9 步登录和运行工程。

7 启动目标系统

启动目标运行系统，执行。

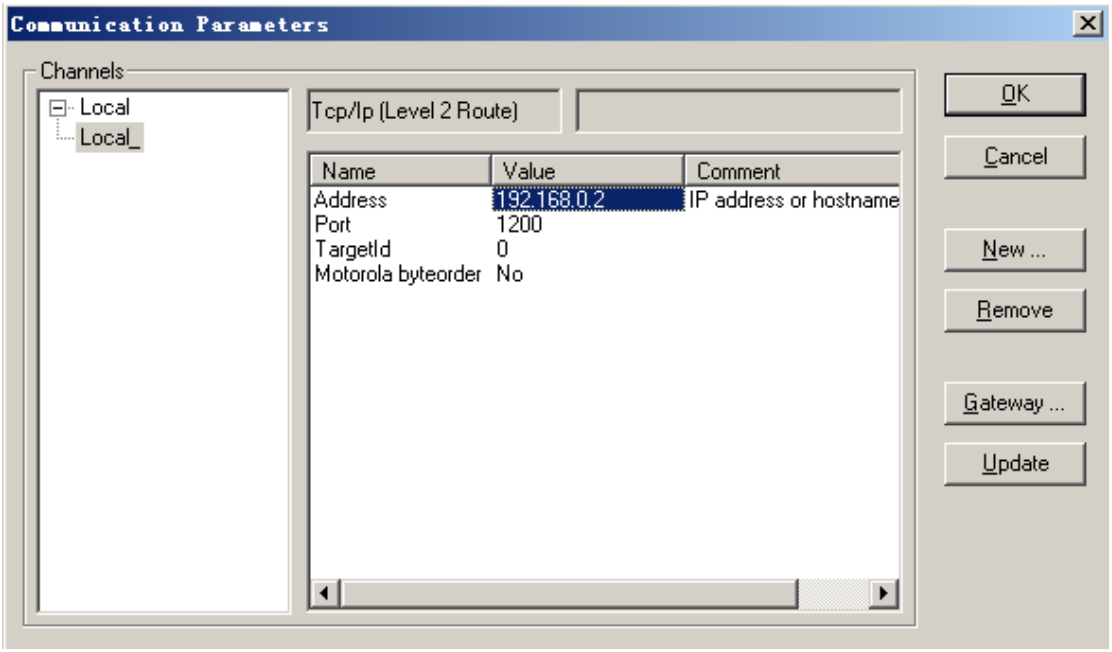
8 进行连接设置

执行命令**联机->通讯参数，**弹出下列对话框：



点击新建按钮配置与目标系统的连接。在新的对话框中选择一种连接方式并输入名字。默认是 Local_ 连接方式是 TCP/IP。

如果是在本计算机上仿真，则不需要设置通讯方式，只要选择**联机->仿真模式**，即可进入下一步。如果需要将程序下载到目标控制器上，则需要在上图中，双击“localhost”,修改为目标控制器制定的 IP 地址，如：192.168.0.2 （默认），如下图所示：



注意：修改完 IP 地址，需要点回车或点对话框空白处以确认修改，然后再点 OK 关闭对话框。

9 运行工程

通过命令**联机->登录**建立 OtoStudio 开发环境与运行系统（目标控制器）的连接；

执行**联机->运行，**程序将在运行系统（目标控制器上运行）。**（如果在仿真模式下运行，激活联机->仿真模式选项）**

也可以利用可视化界面启动机器并操作确认开关。

更多信息请参阅在线帮助和《CPAC 系统安装指导手册》

CPAC-OtoStudio 运动控制编程手册

第一章 OtoStudio 中运动函数库的使用

1.1 OtoStudio 软件库的使用

在 CPAC 软件平台下使用运动控制器，直接安装 Setup，运动控制器指令函数库将存放在默认路径下。GUC-X00-TPX 控制器的库文件名为 CPAC GUC_X00_TPX.lib。

1 OtoStudio 平台中库的使用

1. 启动OtoStudio.exe，新建一个工程；
2. 选择目标平台：CPAC GUC-X00-TPX 或者CPAC GUC-X00-TPX Mini
3. 系统自动添加CPAC GUC-X00-TPX.lib

至此，用户就可以在 OtoStudio 中调用函数库中的任何函数，开始编写应用程序。

第二章 命令返回值及其意义

2.1 指令返回值

CPAC 控制器按照主机发送的指令工作。CPAC 控制器指令封装在动态链接库中。用户在编写应用程序时，通过调用 CPAC 控制器 中运动控制库 GUC-X00-TPX.lib 指令来操纵 CPAC 控制器的运动控制。

CPAC 控制器在接收到主机发送的指令时，将执行结果反馈到主机，指示当前指令是否正确执行。指令返回值的定义如下。

运动控制器指令返回值定义

返回值	意义	处理方法
0	指令执行成功	
1	指令执行错误	1. 检查当前指令的执行条件是否满足
7	指令参数错误	1. 检查当前指令输入参数的取值
-1	主机和运动控制器通讯失败	1. 是否正确安装运动控制器驱动程序 2. 检查运动控制器是否接插牢靠 3. 更换主机 4. 更换控制器
-6	打开控制器失败	1. 是否正确安装运动控制器驱动程序 2. 是否调用了 2 次 GT_Open 指令 3. 其他程序是否已经打开运动控制器
-7	运动控制器没有响应	1. 更换运动控制器

	建议在用户程序中，检测每条指令的返回值，以判断指令的执行状态。并建立必要的错误处理机制，保证程序安全可靠地运行。

第三章 系统配置

在使用 CPAC 控制器进行运动控制操作之前，需要对运动控制进行配置，使控制器的状态和各种工作模式能够满足客户的要求。这个过程，叫做系统配置。在 OtoStudio 软件的 PLC 配置中包括一个系统配置的组件，用户可以利用该组件来对运动控制器进行配置，配置完成之后，生成相应的配置文件*.cfg，用户在编程时，调用相关的指令，将配置信息传递给运动控制器，即可完成整个运动控制器的配置工作。

系统配置基本概念

运动控制器内部包含了各种软硬件资源，各种软硬件资源之间相互组合，即可实现运动控制器的各种应用。

第四章 运动模式

硬件资源

数字量输出资源(do)：包括伺服使能数字量输出、伺服报警清除数字量输出、通用数字量输出。

数字量输入资源(di)：包括正限位数字量输入、负限位数字量输入、驱动报警数字量输入、原点信号数字量输入、通用数字量输入。

编码器计数资源(encoder)：用来对外部编码器的脉冲输出进行计数。

脉冲输出资源(step)：脉冲输出通道，可以输出“脉冲+方向”或者“CW\CCW”控制脉冲。

电压输出资源(dac)：电压输出通道，输出-10V~+10V 的控制电压。

软件资源

规划器资源(profile)：根据运动模式和运动参数实时计算规划位置和规划速度，生成所需的速度曲线，实时地输出规划位置。

伺服控制器资源(control)：根据伺服控制算法、控制参数、跟随误差实时地计算控制量。

轴资源(axis)：将软件资源、硬件资源进行组合，作为整体进行操作。其中包括驱动报警信号、正限位信号、负限位信号、平滑停止信号、紧急停止信号的管理；规划器输出的规划位置的当量变换；编码器计数位置的当量变换等功能。

资源组合

系统配置就是将上述的硬件资源和软件资源相互组合，并对各个资源的基本属性进行配置的过程。下面的两个例子描述了资源组合的基本概念。

步进控制方式的基本配置如图 3-1-1 所示。

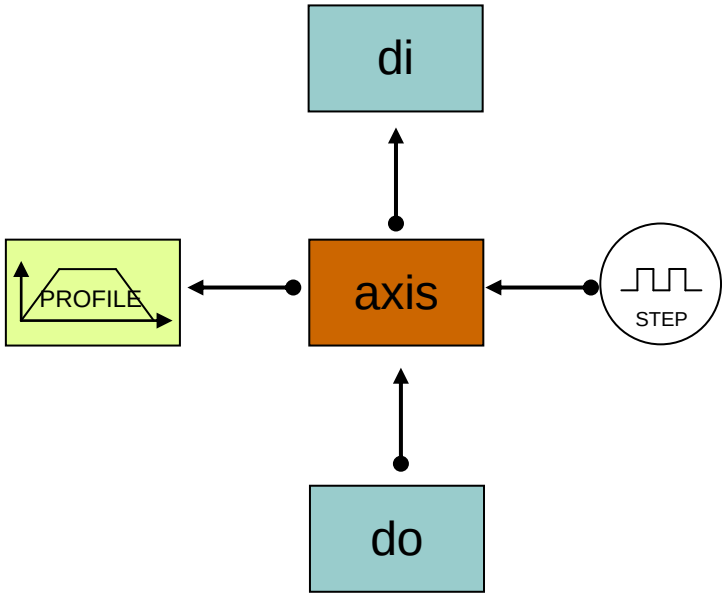


图 3-1-1 步进控制

该实例中，profile 输出的规划位置进入 axis 中，在 axis 中进行当量变换的处理后，输出到 step，由 step 产生控制脉冲，驱动电机运动。axis 需要驱动报警、正向限位信号、负向限位信号、平滑停止信号、紧急停止信号等一些数字量输入信号来对运动进行管理；同时，axis 需要输出伺服使能信息给数字量输出，来使电机使能。

伺服控制方式的基本配置如图 3-1-2 所示。

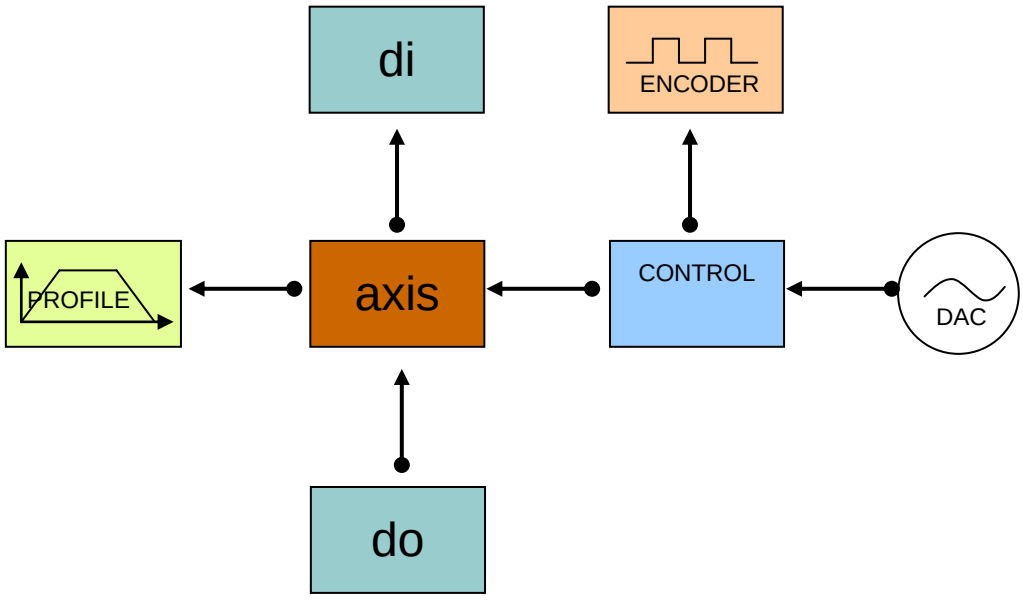


图 3-1-2 伺服控制

该实例中，profile 输出的规划位置进入 axis 中，在 axis 中进行当量变换的处理后，输出到伺服控制器中，伺服控制器将规划位置与 encoder 的计数位置进行比较，获得跟随误差，并通过一定的伺服控制算法，得到实时的控制量，将控制量传递给 dac，由 dac 转换成控制电压来控制电机的运动。axis 需要驱动报警、正向限位信号、负向限位信号、平滑停止信号、

紧急停止信号等一些数字量输入信号来对运动进行管理；同时，axis 需要输出伺服使能信息给数字量输出，来使电机使能。

系统配置工具

使用固高科技提供的 OtoStudio 软件能够方便地对系统进行配置，启动以后显示如下界面。

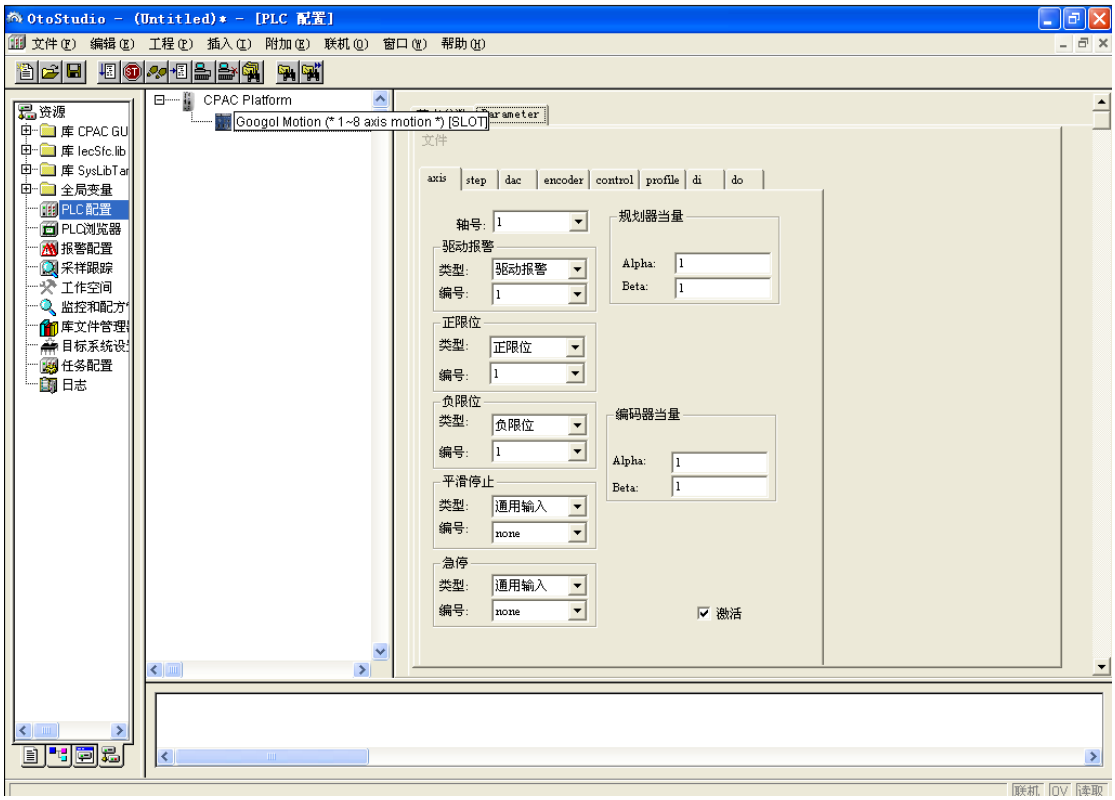


图 3-2-1 OtoStudio 运动控制配置组件

选择“parameter”菜单,打开运动控制器配置面板就可以对系统进行配置。

配置 axis

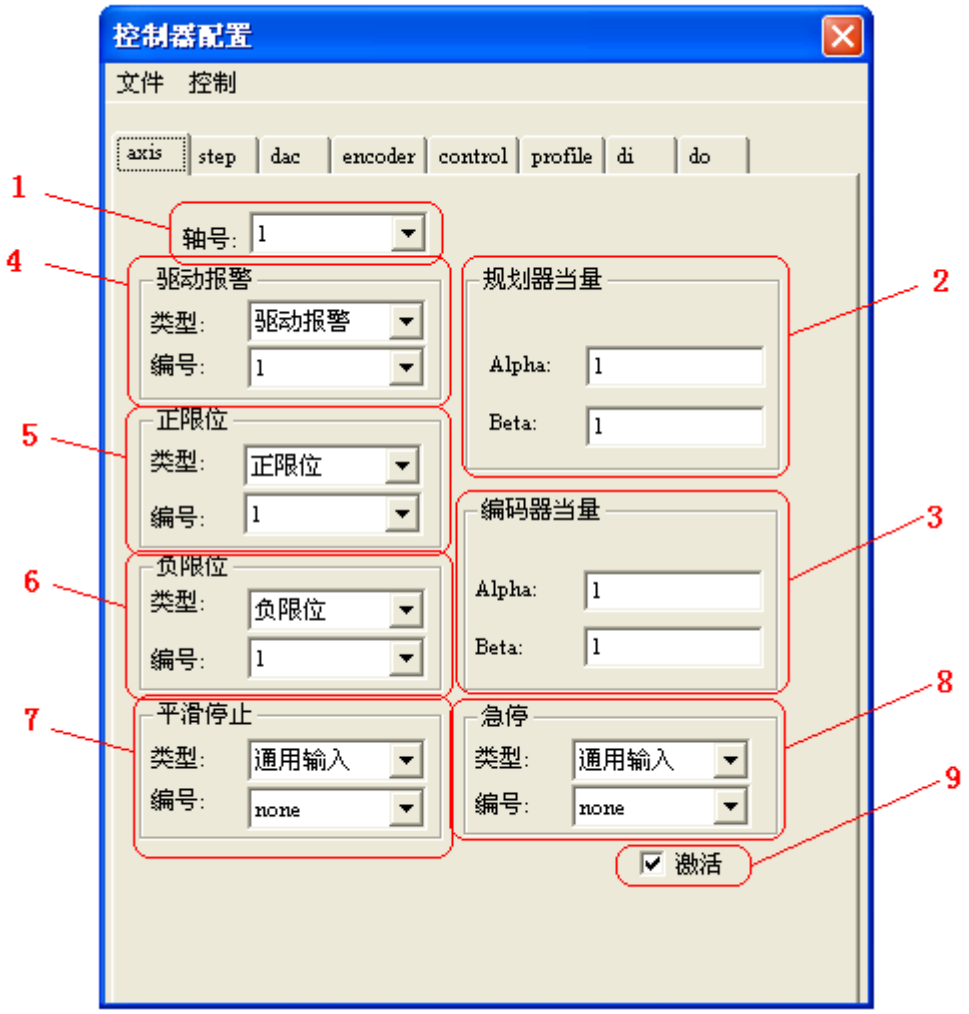


图 3-2-2 axis 配置界面

1. axis 编号选择：选择需要进行配置的 axis 的编号。

2. 规划器当量变换参数：如果需要在 axis 中对规划器输出的规划位置进行当量变换，则可以对该项的参数进行设置，当量变换的关系如下：

$$\frac{\Delta P_{profile}}{\Delta P_{axis}} = \frac{Alpha}{Beta}$$

其中：

$\Delta P_{profile}$ ——规划器输出的规划位置的变化量

ΔP_{axis} ——axis 输出的规划位置的变化量

系统默认的 Alpha 和 Beta 都为 1，所以，规划器输出的规划位置在经过 axis 之后没有经过任何变化。Alpha 的取值范围：(-32767,0)和(0,32767)；Beta 的取值范围：(-32767,0)和(0,32767)。

3．编码器当量变换参数：如果需要在 axis 中对编码器计数的位置值进行当量变换，则可以对该项的参数进行设置，当量变换的关系如下：

$$\frac{\Delta E_{enc}}{\Delta E_{axis}} = \frac{Alpha}{Beta}$$

其中：

ΔE_{enc}——编码器计数的位置值的变化量

ΔE_{axis}——axis 输出的编码器位置值的变化量

系统默认的 Alpha 和 Beta 都为 1，所以，编码器计数的位置值在经过 axis 之后没有经过任何变化。Alpha 的取值范围：(-32767,0)和(0,32767)；Beta 的取值范围：(-32767,0)和(0,32767)。

4．驱动报警信号数字量输入选择：选择驱动报警信号的数字量输入的来源，运动控制器支持将任何数字量输入信号配置为驱动报警信号，增加用户进行硬件接线的自由性。该项的第一个下拉列表选择数字量输入的类型，默认为选择驱动报警数字量输入；第二个下拉列表选择数字量输入的编号，在第二个下拉列表中如果选择“none”，则表示该 axis 的驱动报警信号无效。

5．正限位信号数字量输入选择：选择正限位信号的数字量输入的来源，运动控制器支持将任何数字量输入信号配置为正限位信号，增加用户进行硬件接线的自由性。该项的第一个下拉列表选择数字量输入的类型，默认为选择正限位数字量输入；第二个下拉列表选择数字量输入的编号，在第二个下拉列表中如果选择“none”，则表示该 axis 的正限位信号无效。

6．负限位信号数字量输入选择：选择负限位信号的数字量输入的来源，运动控制器支持将任何数字量输入信号配置为负限位信号，增加用户进行硬件接线的自由性。该项的第一个下拉列表选择数字量输入的类型，默认为选择负限位数字量输入；第二个下拉列表选择数字量输入的编号，在第二个下拉列表中如果选择“none”，则表示该 axis 的负限位信号无效。

7．平滑停止信号数字量输入选择：选择平滑停止信号的数字量输入的来源，运动控制器支持将任何数字量输入信号配置为平滑停止信号，增加用户进行硬件接线的自由性。该项的第一个下拉列表选择数字量输入的类型，默认为没有平滑停止信号；第二个下拉列表选择数字量输入的编号，在第二个下拉列表中如果选择“none”，则表示该 axis 没有平滑停止信号。

8．紧急停止信号数字量输入选择：选择紧急停止信号的数字量输入的来源，运动控制器支持将任何数字量输入信号配置为紧急停止信号，增加用户进行硬件接线的自由性。该项的第一个下拉列表选择数字量输入的类型，默认为没有紧急停止信号；第二个下拉列表选择数字量输入的编号，在第二个下拉列表中如果选择“none”，则表示该 axis 没有紧急停止信号。

9．axis 激活选项：如果 axis 不被激活，则与该 axis 相关的所有计算和管理任务将会无效。默认 axis 都是激活的。但是如果没有用到某个 axis 的相关功能，则可以不把该 axis 激活，这样可以节约运动控制器的处理资源。

配置 step

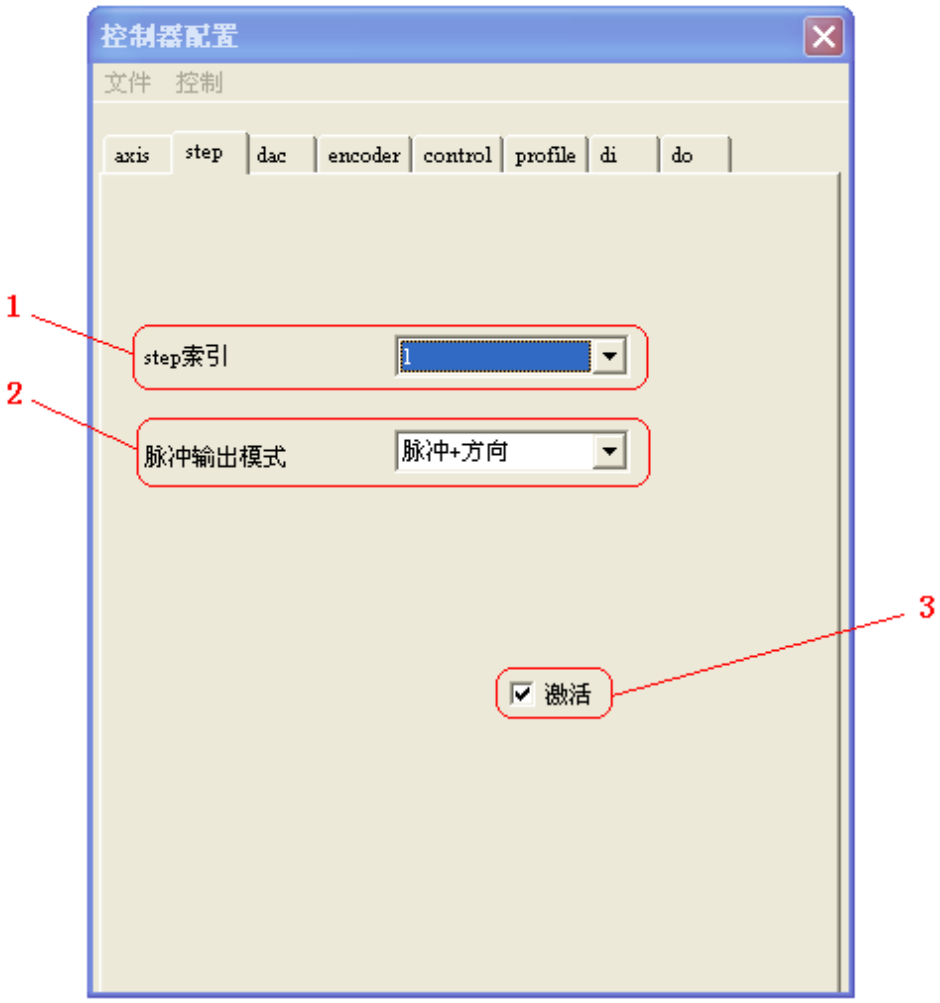


图 3-2-3 step 配置界面

1. step 编号选择：选择需要进行配置的 step 的编号。
2. step 输出脉冲信号模式选择：可以选择 step 脉冲输出通道的脉冲输出模式，可以为“脉冲+方向”或者“CW/CCW”，默认为“脉冲+方向”。
3. step 激活选项：如果 step 不被激活，则该脉冲输出通道将不可用，不会输出脉冲。默认 step 都是激活的。但是如果没有用到某个 step，则可以不把该 step 激活，这样可以节约运动控制器的处理资源。

配置 dac

图 3-2-4 dac 配置界面

1. dac 编号选择：选择需要进行配置的 dac 的编号。
2. dac 输出电压反转：选择是否需要将 dac 的输出电压取反，如果为“正常”，则向 dac 中写入正值时，dac 输出正电压，向 dac 中写入负值时，dac 输出负电压；如果为“取反”，则反之。
3. dac 的零漂补偿值：如果需要对 dac 进行零漂补偿时，在这里设置具体的零漂补偿值。
4. dac 输出电压饱和极限：该项设置 dac 能够输出的最大电压绝对值。如果设置为 32767，则允许输出的电压范围为：-10V~+10V，设置为 16384，则允许输出的电压范围为：-5V~+5V。如果 control 输出的控制量绝对值，或者用户使用 GT_SetDac()指令设置的电压值的绝对值超过设定值时，将会按照该项设置的参数被限制在指定电压范围之内。
5. dac 激活选项：如果 dac 不被激活，则该电压输出通道将不可用，不会输出电压值。默认 dac 都是激活的。但是如果没有用到某个 dac，则可以不把该 dac 激活，这样可以节约运动控制器的处理资源。

配置 encoder

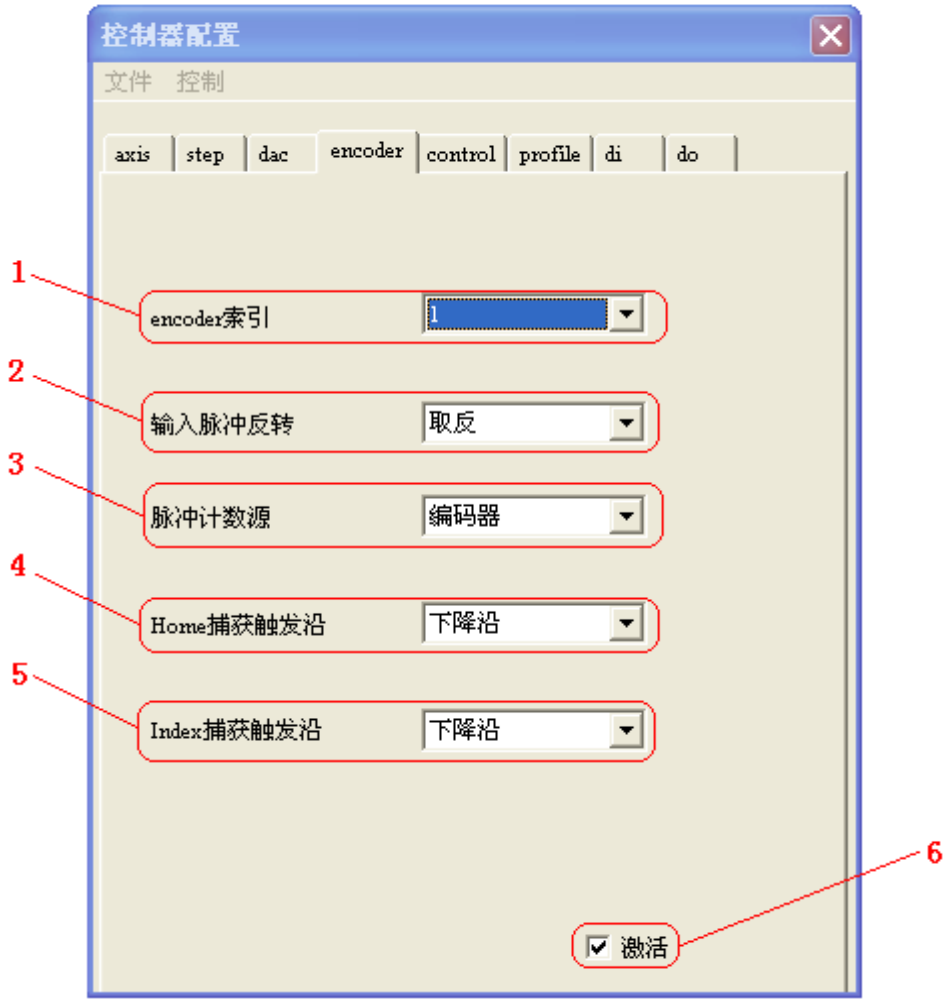


图 3-2-5 encoder 配置界面

1. encoder 编号选择：选择需要进行配置的 encoder 的编号。
2. encoder 输入脉冲反转：运动控制器可以接收正交编码器信号，该项选项与反馈脉冲方向以及编码器计数方向的关系如下表所示：

	正常		取反	
A 相				
B 相				
编码器	计数增加	计数减少	计数减少	计数增加

3. 脉冲计数源选择：表示编码器计数来源，默认情况下是外部编码器计数。如果没有外接编码器，则可以将其设置为脉冲计数器，encoder 将会对 step 输出的脉冲个数进行计数。

4. Home 捕获触发沿：用来设置 Home 捕获的触发沿，默认为下降沿触发。如果选择了常闭开关，可以将捕获沿设置为上升沿触发。

5. Index 捕获触发沿：用来设置 Index 捕获的触发沿，默认为下降沿触发。

6. encoder 激活选项：如果 encoder 不被激活，则将不会对输入脉冲进行计数。默认 encoder 都是激活的。但是如果没有用到某个 encoder，则可以不把该 encoder 激活，这样可以节约运动控制器的处理资源。

配置 control

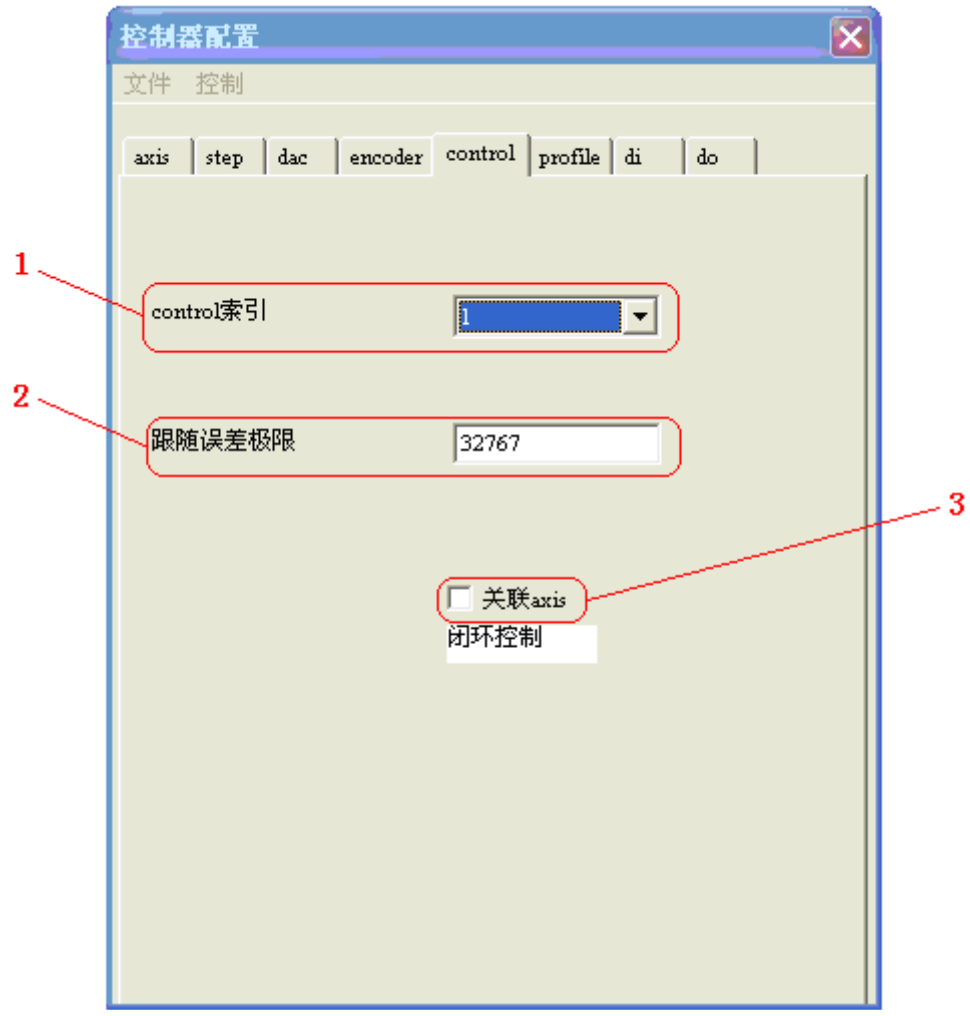


图 3-2-6 control 配置界面

1. control 编号选择：选择需要进行配置的 control 的编号。

2. 跟随误差极限：表示当规划位置和实际位置的误差的极限。当跟随误差超过设定的极限时，自动关闭 axis 的驱动器使能信号。默认为：32767，单位：pulse。

3. control 关联选项：如果需要伺服闭环控制，应该使 control 与 axis 关联。默认为：不关联，及开环脉冲控制方式。关联之后，运动控制器会将相应编号的 encoder、dac、axis、control 关联在一起，如图 3-1-2 所示，此时，dac 的值将不能独立设置，如果调用 GT_SetDac()指令将会无效。

配置 profile

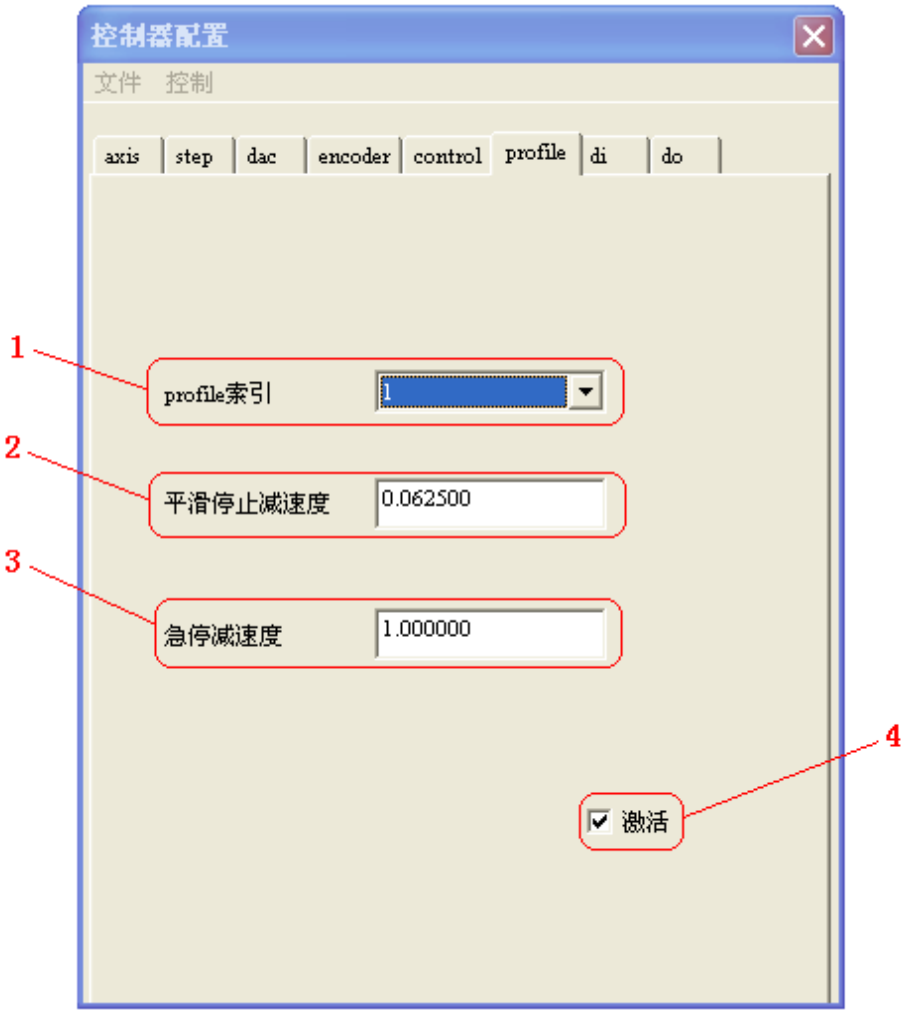


图 3-2-7 profile 配置界面

1. profile 编号选择：选择需要进行配置的 profile 的编号。
2. 平滑停止减速度：表示调用 GT_Stop 指令平滑停止时所使用的减速度，默认为 0.0625 脉冲/毫秒 ^{2}。
3. 紧急停止减速度：表示调用 GT_Stop 指令急停时所使用的减速度，默认为 1 脉冲/毫秒 ^{2}。
4. profile 激活选项：如果 profile 不被激活，则将不会进行运动规划。默认 profile 都是激活的。但是如果没有用到某个 profile，则可以不把该 profile 激活，这样可以节约运动控制器的处理资源。

配置 di

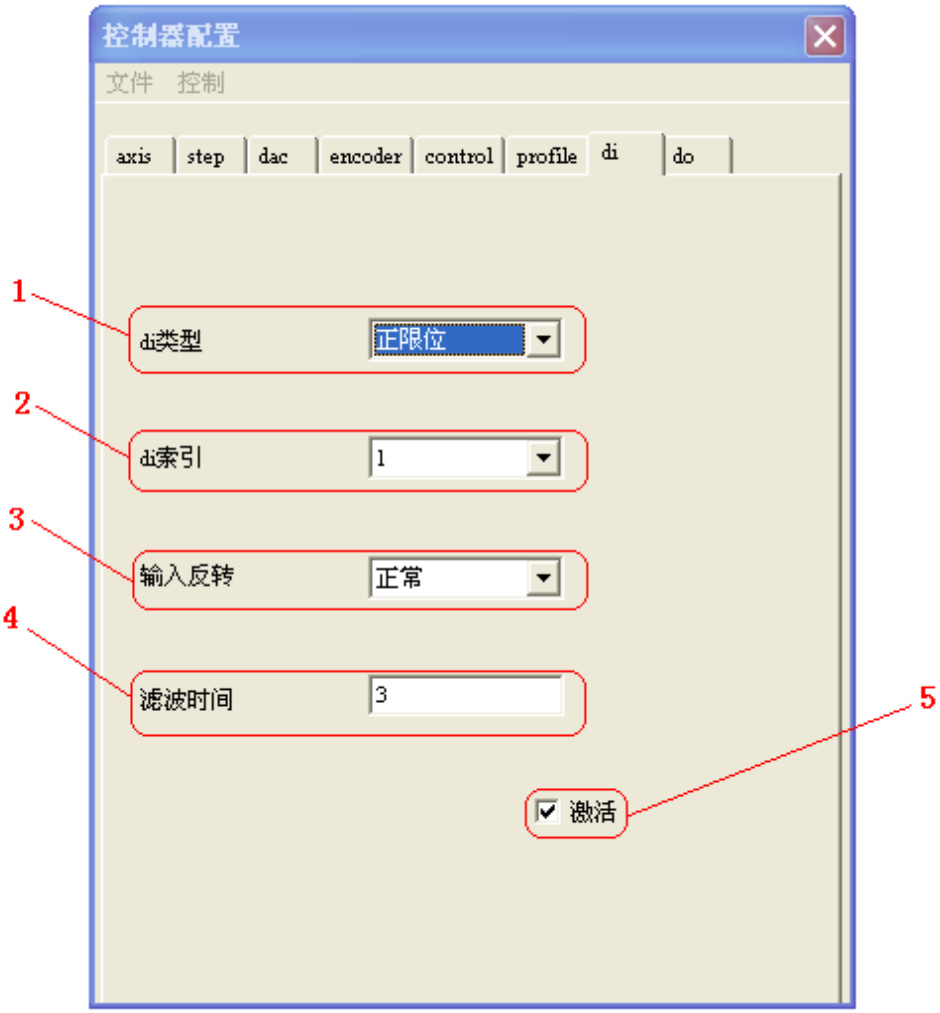


图 3-2-8 di 配置界面

1. di 类型选择：选择需要配置的 di 的类型，包括：驱动报警、正限位、负限位、原点、通用输入。
2. di 编号选择：选择需要配置的 di 的编号。
3. 输入反转：表示 di 的输入逻辑取反。默认情况下，0 表示输入低电平，1 表示输入高电平；输入反转后，0 表示输入高电平，1 表示输入低电平。
4. 滤波时间：表示 di 输入信号维持设定时间才有效，默认滤波时间为 3，单位为：250 微秒。
5. di 激活选项：如果 di 不被激活，则该数字量输入将不起作用。默认 di 都是激活的。但是如果没有用到某个 di，则可以不把该 di 激活，这样可以节约运动控制器的处理资源。

配置 do

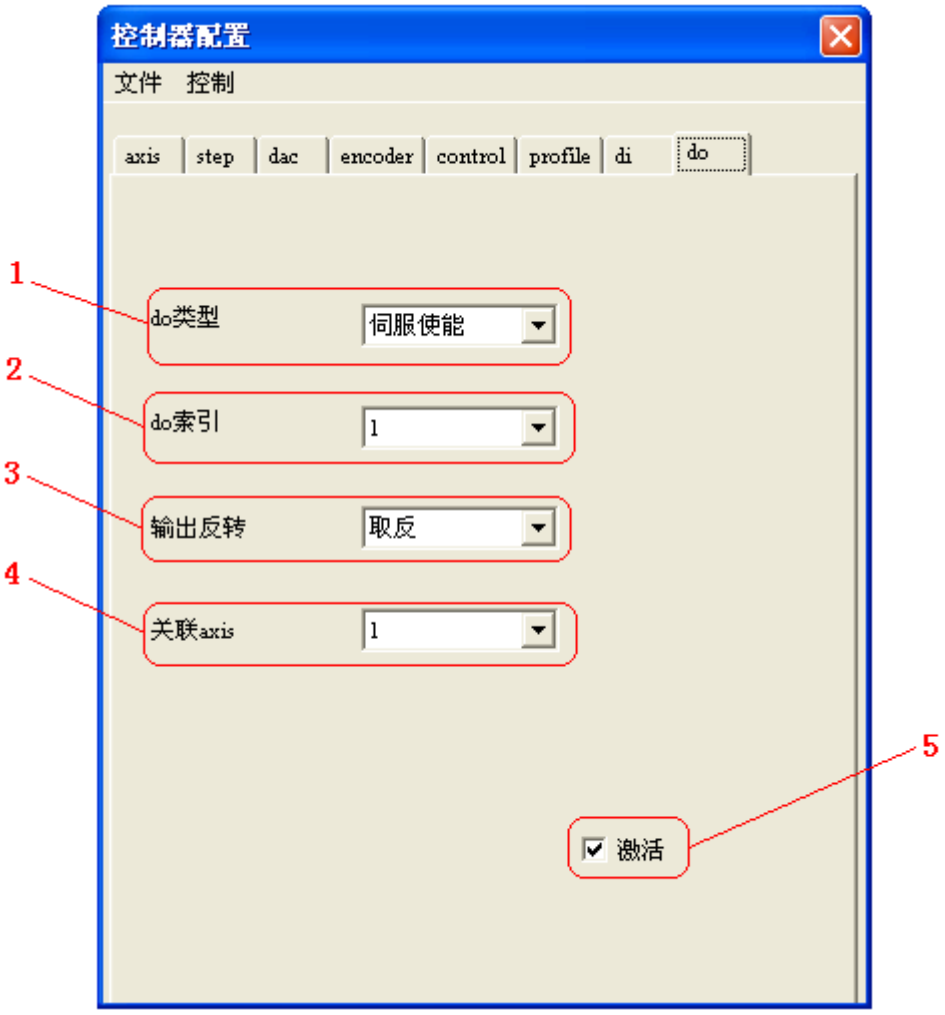


图 3-2-9 do 配置界面

1. do 类型选择：选择需要配置的 do 的类型，包括：伺服使能、清除报警、通用输出。
2. do 编号选择：选择需要配置的 do 的编号。
3. 输出反转：表示 do 的输出逻辑取反。默认为：正常，0 表示输出低电平，1 表示输出高电平。输出反转后，0 表示输出高电平，1 表示输出低电平。
4. 关联 axis：表示该 do 关联到指定 axis 的驱动器使能。默认情况下，各轴都具有独立的驱动器使能 do 作为输出，当调用 GT_AxisOn()指令时，该 do 置 1。如果驱动器是低电平使能，必须把“输出反转”设置为取反。do 一旦和 axis 关联，就不能调用 GT_SetDo 或 GT_SetDoBit 指令直接设置其输出电平。如果不需要驱动器使能信号，可以取消和驱动器使能 do 的关联。取消关联以后，驱动器使能 do 就可以作为普通 do 来使用，可以调用 GT_SetDo 或 GT_SetDoBit 直接设置其输出电平。
5. do 激活选项：如果 do 不被激活，则该数字量输出将不起作用。默认 do 都是激活的。但是如果没有用到某个 do，则可以不把该 do 激活，这样可以节约运动控制器的处理资源。

配置文件生成和下载

按照 3.2 中所描述进行运动控制器的配置之后，如图 3-3-1 所示，在“控制器配置”界面的“文件”菜单，点击“将 MC 配置加载到当前工程”，即将配置信息保存到当前工程所在目录下以备下载；点击“将 MC 配置写到文件”，即可对配置信息进行保存备份。也可通过“从文件中读取 MC 配置”来导入已有的配置文件信息。

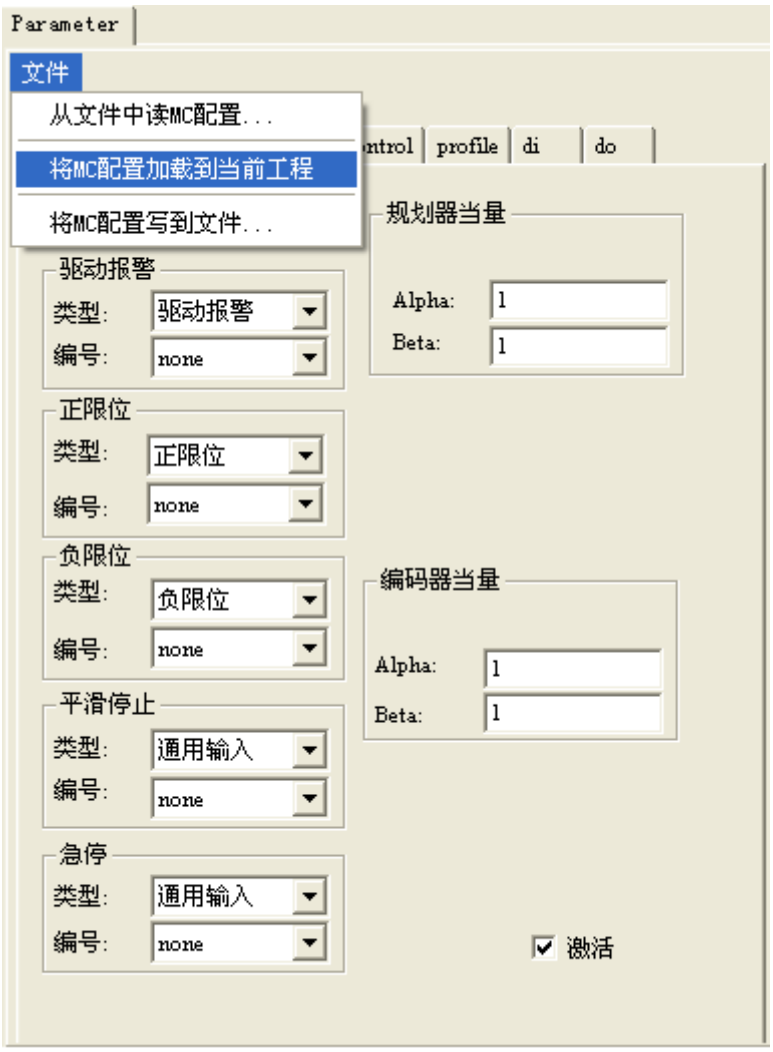


图 3-3-1 生成配置文件界面

当下载程序时候，系统会自动将加载到当前工程的配置信息加载到下位机中。
注：每个新建的工程系统都会分配一个默认配置文件到工程目录下。当打开已有的工程时，在配置对话框中，系统会自动打开上一次加载到当前工程的配置信息（如果无，则加载默认标准配置信息）。

第四章 运动模式

CPAC-OtoBox-UCT2-4PV/4PG/8PV/8PG 控制器每个轴都可以独立工作在点位、Jog、PT、电子齿轮或 Follow 运动模式下。

设置运动模式指令列表

指令	说明
GT_PrfTrap	设置指定轴为点位模式
GT_PrfJog	设置指定轴为 Jog 模式
GT_PrfPt	设置指定轴为 PT 模式
GT_PrfGear	设置指定轴为电子齿轮模式
GT_PrfFollow	设置指定轴为 Follow 模式
GT_GetPrfMode	查询指定轴的运动模式

只能在规划静止状态下切换轴运动模式。调用 GT_Stop 指令可以停止一个或多个轴的运动。

4.1 点位运动

4.1.1 指令列表

设置点位运动指令列表

指令	说明
GT_PrfTrap	设置指定轴为点位运动模式
GT_SetTrapPrm	设置点位模式运动参数
GT_GetTrapPrm	读取点位模式运动参数
GT_SetPos	设置目标位置
GT_GetPos	读取目标位置
GT_SetVel	设置目标速度
GT_GetVel	读取目标速度
GT_Update	启动点位运动

GT_PrfTrap

设置指定轴为点位运动模式

GT_PrfTrap(profile)	
Profile:INT	轴号

GT_SetTrapPrm

设置点位模式运动参数

GT_SetTrapPrm(profile,pPrm)	
Profile:INT	轴号

PPrm:POINTER TO TTrapPrm	设置点位模式运动参数
	typedef struct TrapPrm
	STRUCT TTrapPrm
	Acc:LREAL; // 加速度，单位“脉冲/毫秒 ^{2}”
	Dec:LREAL; // 减速度，单位“脉冲/毫秒 ^{2}”
	VelStart:LREAL; // 起跳速度，单位“脉冲/毫秒”
	SmoothTime:INT; // 平滑时间，单位“毫秒”

GT_GetTrapPrm

读取点位模式运动参数

GT_GetTrapPrm(profile,pPrm)	
Profile:INT	轴号
PPrm: POINTER TO TTrapPrm	读取点位模式运动参数

GT_GetPos

读取目标位置

GT_GetPos(profile, pPos)	
Profile:INT	轴号
PPos: POINTER TO DINT	读取目标位置，单位是脉冲

GT_SetPos

设置目标位置

GT_SetPos(profile,pos)	
Profile:INT	轴号
Pos:DINT	设置目标位置，单位是脉冲

GT_GetVel

读取目标速度

GT_GetVel(profile,pVel)	
Profile:INT	轴号
PVel: POINTER TO LREAL	读取目标位置，单位是“脉冲/毫秒”

GT_SetVel

设置目标速度

GT_SetVel(profile, vel)	
Profile:INT	轴号
Vel:LREAL	设置目标速度，单位是“脉冲/毫秒”

GT_Update

启动点位运动

GT_Update(mask)	
Mask:DINT	按位指示需要启动点位运动的轴号 bit0 表示 1 轴，bit1 表示 2 轴，… 当 bit 位为 1 时表示启动对应的轴

TTrapPrm

设置点位模式运动参数

```
STRUCT TTrapPrm
   Acc:LREAL;          // 加速度，单位“脉冲/毫秒 2”
   Dec:LREAL;          // 减速度，单位“脉冲/毫秒 2”
   VelStart:LREAL;     // 起跳速度，单位“脉冲/毫秒”
   SmoothTime:INT;     // 平滑时间，单位“毫秒”
```

4.1.2 重点说明

在点位运动模式下，各轴可以独立设置目标位置、目标速度、加速度、减速段、起跳速度、平滑时间等运动参数，能够独立运动或停止。

调用 GT_Update 指令启动点位运动以后，控制器根据设定的运动参数自动生成相应的梯形曲线速度规划，并且在运动过程中可以随时修改目标位置和目标速度。

设定平滑时间能够得到平滑的速度曲线，从而使加减速过程更加平稳，如图所示。

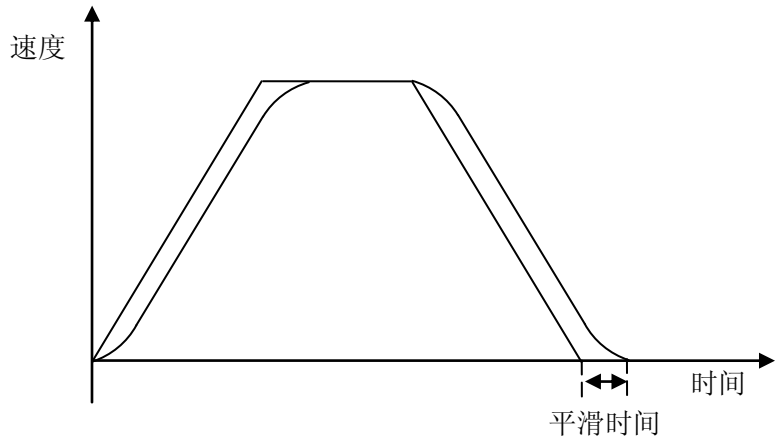


图 4-1-1 点位运动速度曲线

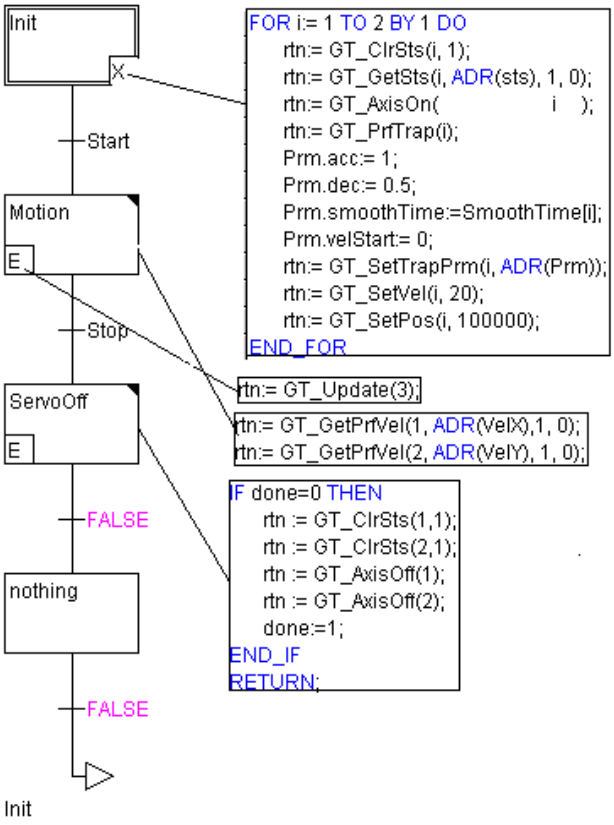
平滑时间是指加速度的变化时间，单位是毫秒，取值范围是[0,50]。

4.1.3 例程

该例程生成一段梯形曲线速度规划，如图所示：

```
0001 PROGRAM PLC_PRG
0002 VAR
0003     rtn: INT;
0004     sts: DWORD;
0005     i: SINT;
0006     Prm: TTrapPrm;
0007     SmoothTime: ARRAY[1..2] OF INT:= 0,100;
0008
0009     Start: BOOL := FALSE;
0010     VelX: LREAL;
0011     VelY: LREAL;
0012     Stop: BOOL := FALSE;
0013     Reset: BOOL := FALSE;
0014     done: BOOL:= FALSE;
0015
0016     startFlag: BOOL := FALSE;
0017     temp: LREAL;
0018     tempPos: DINT;
0019 END_VAR
```

第四章 运动模式


Init
X
Start
Motion
E
Stop
ServoOff
E
FALSE
nothing
FALSE
Init
FOR i:= 1 TO 2 BY 1 DO
rtn:= GT_ClrSts(i, 1);
rtn:= GT_GetSts(i, ADR(sts), 1, 0);
rtn:= GT_AxisOn(i);
rtn:= GT_PrfTrap(i);
Prm.acc:= 1;
Prm.dec:= 0.5;
Prm.smoothTime:=SmoothTime[i];
Prm.velStart:= 0;
rtn:= GT_SetTrapPrm(i, ADR(Prm));
rtn:= GT_SetVel(i, 20);
rtn:= GT_SetPos(i, 100000);
END_FOR
rtn:= GT_Update(3);
rtn:= GT_GetPrfVel(1, ADR(VelX),1, 0);
rtn:= GT_GetPrfVel(2, ADR(VelY), 1, 0);
IF done=0 THEN
rtn := GT_ClrSts(1,1);
rtn := GT_ClrSts(2,1);
rtn := GT_AxisOff(1);
rtn := GT_AxisOff(2);
done:=1;
END_IF
RETURN;

4.2 Jog 模式

4.2.1 指令列表

设置 Jog 模式指令列表

指令	说明
GT_PrfJog	设置指定轴为 Jog 模式
GT_SetJogPrm	设置 Jog 运动参数
GT_GetJogPrm	读取 Jog 运动参数
GT_SetVel	设置目标速度，单位是“脉冲/毫秒”
GT_GetVel	读取目标速度，单位是“脉冲/毫秒”
GT_Update	启动 Jog 模式运动

GT_PrfJog

设置指定轴为 Jog 模式

Jog 模式指令参数说明

GT_PrfJog(profile)	
Profile:INT	轴号

GT_SetJogPrm

设置 Jog 运动参数

GT_SetJogPrm(profile,TJogPrm *pPrm)	
Profile:INT	轴号

PPrm: POINTER TO TJogPrm	设置 Jog 模式运动参数
	STRUCT TJogPrm
	Acc:LREAL; // 加速度，单位“脉冲/毫秒
	Dec:LREAL; // 减速度，单位“脉冲/毫秒
	Smooth:LREAL; // 平滑系数，取值范围[0,1)

GT_GetJogPrm

读取 Jog 运动参数

GT_GetJogPrm(profile, pPrm)	
Profile:INT	轴号
pPrm: POINTER TO TJogPrm	读取 Jog 模式指令运动参数

TJogPrm

设置 Jog 模式运动参数
STRUCT TJogPrm
Acc:LREAL; // 加速度，单位“脉冲/毫秒
Dec:LREAL; // 减速度，单位“脉冲/毫秒
Smooth:LREAL; // 平滑系数，取值范围[0,1)

4.2.2 重点说明

在 Jog 运动模式下，各轴可以独立设置目标速度、加速度、减速段、平滑系数等运动参数，能够独立运动或停止。

调用 GT_Update 指令启动 Jog 运动以后，按照设定的加速度加速到目标速度后保持匀速运动，在运动过程中可以随时修改目标速度，如图所示：

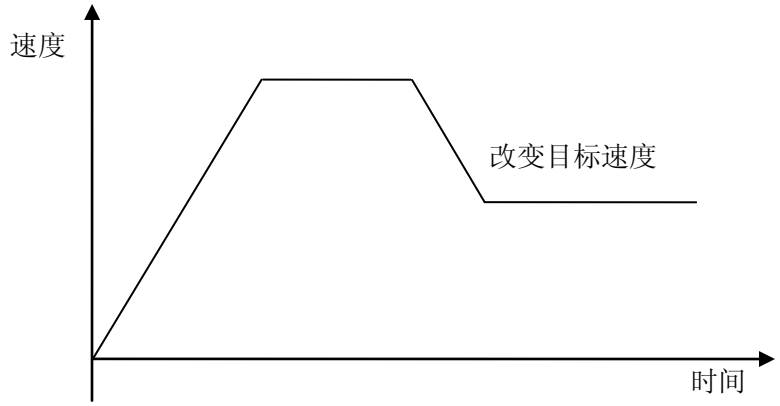


图 4-2-1 Jog 模式速度曲线

设定平滑系数能够得到平滑的速度曲线，从而使加减速过程更加平稳。平滑系数的取值范围是[0，1)，越接近 1，加速度变化越平稳。

4.2.3 例程

该例程在 jog 模式下动态改变目标速度，如图所示：

第四章 运动模式

```
0001 PROGRAM PLC_PRG
0002 VAR
0003     Enable: BOOL := TRUE;
0004     rtn: INT;
0005     AXIS_X: INT:= 1;
0006     Sts: DWORD;
0007     JogPrm: TJogPrm;
0008     PrfVel: LREAL;
0009     PrfPos: LREAL;
0010     trans: BOOL;
0011     STOP: BOOL := FALSE;
0012     Done: BOOL := TRUE;
0013 END_VAR
```

```
IF Enable THEN
    (*上伺服*)
    rtn:= GT_ClrSts(AXIS_X, 1);
    rtn:= GT_GetSts(AXIS_X, ADR(Sts), 1, 0);
    rtn:= GT_AxisOn(AXIS_X);
    (*将X轴设置为JOG模式*)
    rtn:= GT_PrfJog(AXIS_X);
    JogPrm.acc:= 0.5;
    JogPrm.dec:= 0.5;
    JogPrm.smooth:= 0.8;
    rtn:= GT_SetJogPrm(AXIS_X, ADR(JogPrm));
    (*启动Jog运动*)
    rtn:= GT_SetVel(AXIS_X, 50);
    rtn:= GT_Update(SHL(DWORD#1, AXIS_X-1));
    Enable:= FALSE;
    trans:= TRUE;
END_IF
rtn:= GT_GetSts(AXIS_X, ADR(Sts), 1, 0);
rtn:= GT_GetPrfVel(AXIS_X, ADR(PrfVel), 1, 0);
rtn:= GT_GetPrfPos(AXIS_X, ADR(PrfPos), 1, 0);
IF PrfPos>=100000 AND trans=TRUE THEN
    (*修改目标速度*)
    rtn:= GT_SetVel(AXIS_X, 25);
    rtn:= GT_Update(SHL(DWORD#1, AXIS_X-1));
    trans:= FALSE;
END_IF

IF STOP=TRUE AND Done THEN
    rtn:= GT_ClrSts(AXIS_X, 1);
    rtn:= GT_GetSts(AXIS_X, ADR(Sts), 1, 0);
    rtn:= GT_AxisOff(AXIS_X);
    Done:= FALSE;
END_IF
```

4.3 PT 模式

4.3.1 指令列表

设置 PT 模式指令列表

指令	说明
GT_PrfPt	设置指定轴为 PT 模式
GT_PtSpace	查询 PT 指定 FIFO 的剩余空间
GT_PtData	向 PT 指定 FIFO 增加数据
GT_PtClear	清除 PT 指定 FIFO 中的数据 运动状态下该指令无效 动态模式下该指令无效
GT_SetPtLoop	设置 PT 模式循环执行的次数 动态模式下该指令无效
GT_GetPtLoop	查询 PT 模式循环执行的次数 动态模式下该指令无效
GT_PtStart	启动 PT 模式运动
GT_SetPtMemory	设置 PT 运动模式的缓存区大小
GT_GetPtMemory	读取 PT 运动模式的缓存区大小

GT_PrfPt

设置指定轴为 PT 模式

GT_PrfPt(profile,mode)	
Profile:INT	轴号
Mode:INT	指定 FIFO 使用模式 PT_MODE_STATIC 静态模式，默认 PT_MODE_DYNAMIC 动态模式

GT_PtSpace

查询 PT 指定 FIFO 的剩余空间

GT_PtSpace(profile,pSpace,fifo)	
Profile:INT	轴号
PSpace:POINTER TO INT	读取 PT 指定 FIFO 的剩余空间
Fifo:INT	指定所要查询的 FIFO，默认为 0 动态模式下该参数无效

GT_PtData

向 PT 指定 FIFO 增加数据

GT_PtData(profile,pos,time,type,fifo)	
Profile:INT	轴号
Pos:LREAL	段末位置，单位脉冲
Ttime:DINT	段末时间，单位毫秒
Type:INT	数据段类型 PT_SEGMENT_NORMAL 普通段，默认 PT_SEGMENT_EVEN 匀速段 PT_SEGMENT_STOP 减速到 0 段
Fifo:INT	指定存放运动数据的 FIFO，默认为 0 动态模式下该参数无效

GT_PtClear

清除 PT 指定 FIFO 中的数据
运动状态下该指令无效
动态模式下该指令无效

GT_PtClear(profile,fifo)	
Profile:INT	轴号
Fifo:INT	指定所要清空的 FIFO，默认为 0 动态模式下该参数无效

GT_SetPtLoop

设置 PT 模式循环执行的次数
动态模式下该指令无效

GT_SetPtLoop(profile,loop)	
Profile:INT	轴号
Loop:DINT	指定 PT 模式循环执行的次数 动态模式下该参数无效

GT_GetPtLoop

查询 PT 模式循环执行的次数
动态模式下该指令无效

GT_GetPtLoop(profile,pLoop)	
Profile:INT	轴号
PLoop:POINTER TO DINT	查询 PT 模式循环执行的次数

	动态模式下该参数无效

GT_PtStart

启动 PT 模式运动

GT_PtStart(mask,option)	
Mask:DINT	按位指示需要启动 PT 运动的轴号 bit0 表示 1 轴，bit1 表示 2 轴，… 当 bit 位为 1 时表示启动对应的轴
Option:DINT	按位指示所使用的 FIFO，默认为 0 bit0 表示 1 轴，bit1 表示 2 轴，… 当 bit 位为 0 时表示对应的轴使用 FIFO1 当 bit 位为 1 时表示对应的轴使用 FIFO2 动态模式下该参数无效

GT_SetPtMemory

设置 PT 运动模式缓冲区大小

GT_SetPtMemory(profile, memory)	
Profile:INT	规划轴号
Memory:INT	PT 运动缓存区大小标志： 0：每个 PT 运动缓存区有 32 段空间。 1：每个 PT 运动缓存区有 1024 段空间。

GT_GetPtMemory

读取 PT 运动模式缓冲区大小

GT_GetPtMemory(profile,pMemory)	
Profile:INT	规划轴号
PMemory:INT	读取 PT 运动缓存区大小标志

4.3.2 重点说明

PT 模式非常灵活，能够实现任意速度规划。用户通过直接输入位置和时间参数描述运动规律。

PT 模式使用一系列“位置、时间”数据点描述速度规划，用户需要将速度曲线分割成若干段，如下图所示。

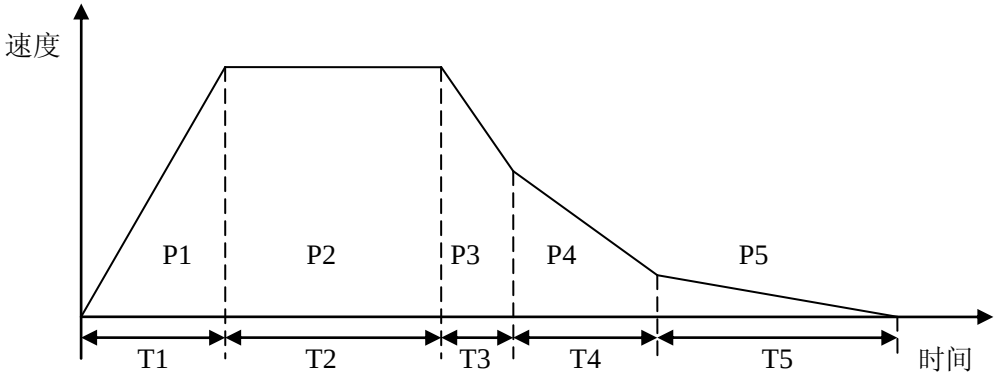


图 4-3-1 PT 运动速度曲线

整个速度曲线被分割成 5 段，第 1 段起点速度为 0，经过时间 T1 运动位移 P1，因此第 1 段的终点速度为 $v1 = \frac{2P1}{T1}$；第 2 段起点速度为 v1，经过时间 T2 运动位移 P2，因此第 2 段的终点速度为 $v2 = \frac{2P2}{T2} - v1$；第 3、4、5 段依此类推。

用户只需要给出每段所需时间和位移，运动控制器会计算段内各点的速度和位置，生成一条连续的速度曲线。为了得到光滑的速度曲线，可以增加速度曲线的分割段数。

在描述一次完整的 PT 运动时，第 1 段的起点位置和时间被假定为 0，各段的终点位置和时间都是相对于第 1 段的起点的绝对值。位置的单位脉冲，时间单位是毫秒。

4.3.2.1 数据段类型

PT 模式的数据段有 3 种类型。

PT_SEGMENT_NORMAL 表示普通段，FIFO 中第 1 段的起点速度为 0，从第 2 段起每段的起点速度等于上一段的终点速度。

PT_SEGMENT_EVEN 表示匀速段，FIFO 中各段的段内速度保持不变，段内速度=段内位移/段内时间。如图所示：

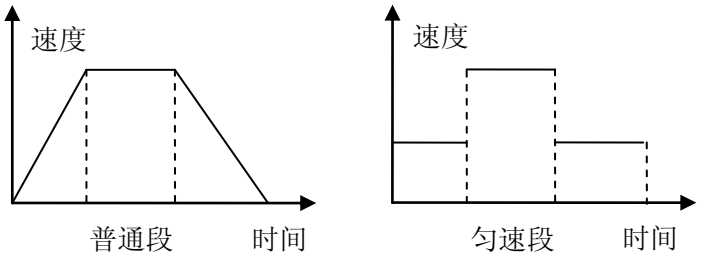


图 4-3-2 PT 模式匀速段类型

PT_SEGMENT_STOP 表示停止段，该段的终点速度为 0，起点速度根据段内位移和段内时间计算得到，和上一段的终点速度无关。如图所示：

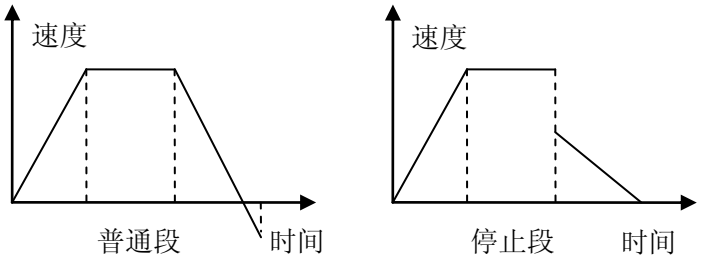


图 4-3-3 PT 模式停止段类型

4.3.2.2 PT 运动模式

PT 具有 2 种 FIFO 使用模式：静态模式和动态模式。

PT 具有 2 个 FIFO 用来存放数据。

静态模式下，可以选择启动其中一个 FIFO，运动完成以后规划停止。控制器不会清除 FIFO 中的数据，用户可以重复使用 FIFO 中的数据。静止状态下调用 GT_PtClear 指令可以清空指定 FIFO。在运动状态下不能清空正在使用的 FIFO，但可以清除未使用的 FIFO。

动态模式下，当一个 FIFO 中的数据用完以后会自动清空，同时切换到另一个 FIFO，此时可以向控制器发送新的 PT 数据。当 2 个 FIFO 中的数据都用完以后规划停止。为了避免异常停止，必须在 2 个 FIFO 中的数据都用完之前及时发送新的数据。调用 GT_PtSpace 指令可以查询剩余多少数据空间。

在切换到 PT 模式的同时设置 FIFO 为“静态模式”或“动态模式”。进入 PT 模式以后就不能修改 FIFO 的使用模式。

4.3.3 例程

4.3.3.1 PT 静态 FIFO

该例程生成一段梯形曲线速度规划，如图所示：

```
PROGRAM PLC_PRG
VAR
  Start: BOOL := FALSE;
  Enable: BOOL := TRUE;
  rtn: INT;
  AXIS_X: INT := 1;
  space: DINT;
  pos: LREAL;
  gtime: DINT;
  Sts: DWORD;
```

```
  PrfVel: LREAL;
  PrfPos: LREAL;
  Stop: BOOL := FALSE;
  Done: BOOL := TRUE;
END_VAR
-----------------------------------------------------
IF Start AND Enable THEN
  (*上伺服*)
  rtn:= GT_ClrSts(AXIS_X, 1);
  rtn:= GT_AxisOn(AXIS_X);
  (*设置为PT模式*)
  rtn:= GT_PrfPt(AXIS_X, PT_MODE_STATIC);
  rtn:= GT_PtClear(AXIS_X, 0);
  (*向缓冲区压入数据*)
  rtn:= GT_PtSpace(AXIS_X, ADR(space), 0);
  pos:=10000;
  gtime:= 2000;
  IF space>0 THEN
      rtn:= GT_PtData(AXIS_X, pos, gtime, PT_SEGMENT_NORMAL, 0);
  END_IF

  rtn:= GT_PtSpace(AXIS_X, ADR(space), 0);
  pos:=pos+20000;
  gtime:= gtime+2000;
  IF space>0 THEN
      rtn:= GT_PtData(AXIS_X, pos, gtime, PT_SEGMENT_NORMAL, 0);
  END_IF

  rtn:= GT_PtSpace(AXIS_X, ADR(space), 0);
  pos:=pos+10000;
  gtime:= gtime+2000;
  IF space>0 THEN
      rtn:= GT_PtData(AXIS_X, pos, gtime, PT_SEGMENT_NORMAL, 0);
  END_IF
  (*启动PT模式运动*)
  rtn:= GT_PtStart(SHL(INT#1, AXIS_X-1), 16#00);

  Enable:= FALSE;
END_IF
(*查询状态，规划速度， 规划位置*)
rtn:= GT_GetSts(AXIS_X, ADR(Sts), 1, 0);
rtn:= GT_GetPrfVel(AXIS_X, ADR(PrfVel), 1, 0);
rtn:= GT_GetPrfPos(AXIS_X, ADR(PrfPos), 1, 0);
```

```
    IF Stop AND Done THEN
      (*关闭伺服*)
      rtn:= GT_ClrSts(AXIS_X, 1);
      rtn:= GT_AxisOff(AXIS_X);
      Done:= FALSE;
    END_IF
```

4.3.3.2 PT 动态 FIFO

该例程生成一段梯形正弦曲线速度规划，如图所示：、

```
PROGRAM PLC_PRG
VAR CONSTANT
  T: DINT := 2000;
  DeltaT: DINT:= 2;
  PI: LREAL:= 3.1415926;
END_VAR
VAR
  InitDone: BOOL := FALSE;
  Start: BOOL := FALSE;
  rtn: INT;
  AXIS_X: INT := 4;
  PrfPos: LREAL;
  PrfVel: LREAL;
  Sts: DWORD;
  Pos: LREAL := 0;
  gtime: DINT := 0;
  Vel: LREAL;
  PreVel: LREAL;
  Space: DINT;
  i: INT;
  PTStart: BOOL := FALSE;
  Stop: BOOL := FALSE;
  StopDone: BOOL := FALSE;
  EncPos: LREAL;
END_VAR
-----------------------------------------------------
IF NOT InitDone AND Start THEN
  (*上伺服*)
  rtn:= GT_ClrSts(AXIS_X, 1);
  rtn:= GT_AxisOn(AXIS_X);
  (*设置运动模式为动态PT模式*)
  rtn:= GT_PrfPt(AXIS_X, PT_MODE_DYNAMIC);
```

```
    rtn:= GT_PtClear(AXIS_X, 0);

    Pos:= 0;
    Vel:= 0;
    gtime:= 0;
    PreVel:= 0;
    InitDone:= TRUE;
  END_IF

  (*查询状态，规划速度， 规划位置*)
  rtn:= GT_GetSts(AXIS_X, ADR(Sts), 1, 0);
  rtn:= GT_GetPrfVel(AXIS_X, ADR(PrfVel), 1, 0);
  rtn:= GT_GetPrfPos(AXIS_X, ADR(PrfPos), 1, 0);

  (*向缓冲区压数据*)
  FOR i:=1 TO 5 BY 1 DO
    rtn:= GT_PtSpace(AXIS_X, ADR(Space), 0);
    IF Space>0 THEN
        gtime:= gtime+DeltaT;
        Vel:= 30*SIN(2*PI*gtime/T);
        Pos:= Pos+ (Vel+PreVel)*DeltaT/2;
        PreVel:= Vel;
        rtn:= GT_PtData(AXIS_X, Pos, gtime, PT_SEGMENT_NORMAL, 0);
    END_IF
  END_FOR

  IF NOT PTStart AND InitDone THEN
    rtn:= GT_PtStart(SHL(INT#1, AXIS_X-1), 0);
    PTStart:= TRUE;
  END_IF

  IF Stop  AND NOT StopDone THEN
    rtn:= GT_Stop(SHL(DINT#1, AXIS_X-1), 0);
    rtn:= GT_ClrSts(AXIS_X, 1);
    rtn:= GT_AxisOff(AXIS_X);
    StopDone:= TRUE;
  END_IF
```

4.4 电子齿轮

4.4.1 指令列表

设置电子齿轮指令列表

指令	说明
GT_PrfGear	设置指定轴为电子齿轮模式
GT_SetGearMaster	设置跟随主轴
GT_GetGearMaster	读取跟随主轴
GT_SetGearRatio	设置电子齿轮比
GT_GetGearRatio	读取电子齿轮比
GT_GearStart	启动电子齿轮

GT_PrfGear

设置指定轴为电子齿轮模式

GT_PrfGear (profile ,dir)	
Profile:INT	轴号
Dir:INT	设置跟随方式 0 表示双向跟随，1 表示正向跟随，-1 表示负向跟随

GT_SetGearMaster

设置跟随主轴相关参数

GT_SetGearMastert(profile , masterIndex, masterType,masterItem)	
Profile:INT	轴号
MasterIndex:INT	主轴索引
MasterType:INT	主轴类型 GEAR_MASTER_ENCODER 表示跟随编码器 GEAR_MASTER_PROFILE 表示跟随规划轴 GEAR_MASTER_AXIS 表示跟随合成轴
MasterItem:INT	合成轴类型 0 表示合成轴规划位置 1 表示合成轴编码器位置

GT_GetGearMaster

读取跟随主轴相关参数

GT_GetGearMaster(profile, pMasterIndex, pMasterType,pMasterItem)	
Profile:INT	轴号
PMasterIndex:POINTER TO INT	主轴索引
PMasterType: POINTER TO INT	主轴类型
PMasterItem: POINTER TO INT	合成轴类型

GT_SetGearRatio

设置电子齿轮比

GT_SetGearRatio(profile,masterEven,slaveEven,masterSlope)	
Profile:INT	轴号
MasterEven:DINT	传动比，主轴位移
SlaveEven:DINT	传动比，从轴位移
MasterSlope:DINT	主轴离合区位移

GT_GetGearRatio

读取电子齿轮比

GT_GetGearRatio(profile,pMasterEven,pSlaveEven,pMasterSlope)	
Profile:INT	轴号
PMasterEven: POINTER TO DINT	主轴位移
PSlaveEven: POINTER TO DINT	主轴位移
PMasterSlope: POINTER TO DINT	主轴离合区位移

GT_GearStart

启动电子齿轮运动

GT_GearStart(mask)	
Mask:DINT	按位指示需要启动 Gear 运动的轴号 bit0 表示 1 轴，bit1 表示 2 轴，… 当 bit 位为 1 时表示启动对应的轴

4.4.2 重点说明

电子齿轮模式能够将 2 轴或多轴联系起来，实现精确的同步运动，从而替代传统的机械齿轮连接。电子齿轮模式能够灵活的设置传动比，节省机械系统的安装时间。

电子齿轮模式下，1 个主轴能够驱动多个从轴，从轴可以跟随主轴的规划位置、编码器位置。

为了减少跟随滞后，从轴的轴号应当大于主轴的轴号。

电子齿轮模式能够在线修改传动比，当改变传动比时，可以设置离合区间，实现平滑变速，如图所示。

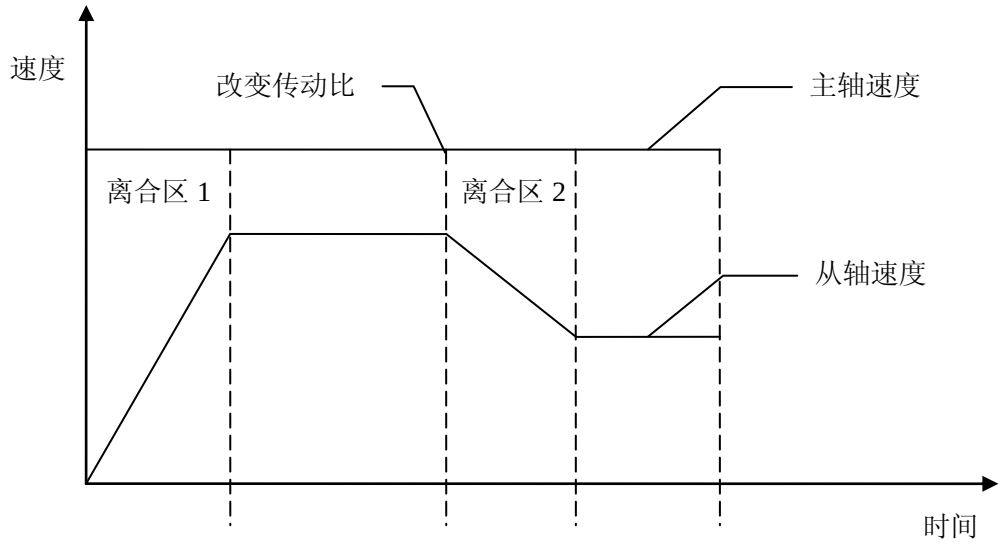


图 4-4-1 电子齿轮模式速度曲线

主轴匀速运动，从轴为电子齿轮模式，在离合区 1 从轴的传动比从 0 逐渐增大到设定传动比。当改变传动比时，在离合区 2 从轴的传动比逐渐变化到新的传动比。离合区越大，从轴传动比的变化过程越平稳。

如果从轴轴号为 slave，当主轴位移 alpha 时从轴位移 beta，主轴运动 slope 位移后从轴到达设定传动比，应当调用以下指令：

GT_SetGearRatio(slave,alpha，beta，slope)；

4.4.3 例程

该例程主轴为 Jog 模式，从轴为电子齿轮模式，传动比为 2：1，主轴运动离合区位移后，从轴达到设定的传动比，如图所示：

```
PROGRAM PLC_PRG
VAR
  Start: BOOL := FALSE;
  Enable: BOOL := TRUE;
  rtn: INT;
  MASTER: INT := 3;
  SLAVE: INT := 4;
  JogPrm:TJogPrm;
  MasterPrfVel: LREAL;
  SlavePrfVel: LREAL;
  MasterPrfPos: LREAL;
  SlavePrfPos: LREAL;
```

```
  Stop: BOOL := FALSE;
  StopDone: BOOL := FALSE;
  ModifyGearRation: BOOL := FALSE;
  ModifyGearRationDone: BOOL := FALSE;
END_VAR
-----------------------------------------------------
IF Start AND Enable THEN
  rtn:= GT_ClrSts(MASTER, 1);
  rtn:= GT_ClrSts(SLAVE, 1);
  rtn:= GT_AxisOn(MASTER);
  rtn:= GT_AxisOn(SLAVE);
  (*将主轴设为JOG模式*)
  rtn:= GT_PrfJog(MASTER);
  JogPrm.acc:= 0.5;
  JogPrm.dec:= 0.5;
  JogPrm.smooth:= 0.8;
  rtn:= GT_SetJogPrm(MASTER, ADR(JogPrm));
  rtn:= GT_SetVel(MASTER, 40);
  rtn:= GT_Update(SHL(DINT#1, MASTER-1));
  (*从轴设为Gear模式*)
  rtn:= GT_PrfGear(SLAVE, 0);
  rtn:= GT_SetGearMaster(SLAVE, MASTER, GEAR_MASTER_PROFILE, 0);
  rtn:= GT_SetGearRatio(SLAVE, 2, 1, 8000);
  rtn:= GT_GearStart(SHL(DINT#1, SLAVE-1));

  Enable:= FALSE;
END_IF

rtn:= GT_GetPrfVel(MASTER, ADR(MasterPrfVel), 1, 0);
rtn:= GT_GetPrfVel(SLAVE, ADR(SlavePrfVel), 1, 0);
rtn:= GT_GetPrfPos(MASTER, ADR(MasterPrfPos), 1, 0);
rtn:= GT_GetPrfPos(SLAVE, ADR(SlavePrfPos), 1, 0);

IF ModifyGearRation AND (NOT ModifyGearRationDone) THEN
  rtn:= GT_SetGearRatio(SLAVE, 4, 1, 10000);
  ModifyGearRationDone:= TRUE;
END_IF

IF Stop AND (NOT StopDone) THEN
  rtn:= GT_Stop(SHL(DINT#1, SLAVE-1), 0);
  rtn:= GT_Stop(SHL(DINT#1, MASTER-1), 0);
  rtn:= GT_ClrSts(MASTER, 1);
  rtn:= GT_ClrSts(SLAVE, 1);
  rtn:= GT_AxisOff(MASTER);
```

```
    rtn:= GT_AxisOff(SLAVE);
    StopDone:= TRUE;
END_IF
```

4.5 Follow 模式

4.5.1 指令列表

设置 Follow 指令列表

指令	说明
GT_PrfFollow	设置指定轴为 Follow 模式
GT_SetFollowMaster	设置跟随主轴
GT_GetFollowMaster	读取跟随主轴
GT_SetFollowLoop	设置循环次数
GT_GetFollowLoop	读取循环次数
GT_SetFollowEvent	设置 Follow 模式启动跟随条件
GT_GetFollowEvent	读取 Follow 模式启动跟随条件
GT_FollowSpace	查询 Follow 指定 FIFO 的剩余空间
GT_FollowData	向 Follow 指定 FIFO 增加数据
GT_FollowClear	清除 Follow 指定 FIFO 中的数据 运动状态下该指令无效
GT_FollowStart	启动 Follow 模式运动
GT_FollowSwitch	切换 Follow 所使用的 FIFO
GT_SetFollowMemory	设置 Follow 运动模式的缓存区大小
GT_GetFollowMemory	读取 Follow 运动模式的缓存区大小

GT_PrfFollow

设置指定轴为 Follow 模式

GT_PrfFollow (profile ,dir)	
Profile:INT	轴号
Dir:INT	设置跟随方式 0 表示双向跟随，1 表示正向跟随，-1 表示负向跟随

GT_SetFollowMaster

设置跟随主轴相关参数

GT_SetFollowMaster(profile, masterIndex, masterType,masterItem)	
Profile:INT	轴号
MasterIndex:INT	主轴索引
MasterType:INT	主轴类型 FOLLOW_MASTER_ENCODER 表示跟随编码器 FOLLOW_MASTER_PROFILE 表示跟随规划轴

	FOLLOW_MASTER_AXIS 表示跟随合成轴
MasterItem:INT	合成轴类型 0 表示合成轴规划位置 1 表示合成轴编码器位置

GT_GetFollowMaster

读取跟随主轴相关参数

GT_GetFollowMaster(profile, pMasterIndex, pMasterType, pMasterItem)	
Profile:INT	轴号
PMasterIndex: POINTER TO INT	主轴索引
PMasterType: POINTER TO INT	主轴类型
PMasterItem: POINTER TO INT	合成轴类型

GT_SetFollowLoop

设置循环次数

GT_SetFollowLoop(profile,loop)	
Profile:INT	轴号
Loop:INT	指定 Follow 模式循环执行的次数

GT_GetFollowLoop

读取循环次数

GT_GetFollowLoop(profile,pLoop)	
Profile:INT	轴号
PLoop: POINTER TO DINT	读取 Follow 模式循环执行的次数

GT_SetFollowEvent

设置 Follow 模式启动跟随条件

GT_SetFollowEvent(profile,event,masterDir,pos)	
Profile:INT	轴号
Event:INT	启动跟随条件 FOLLOW_EVENT_START 表示调用 GT_FollowStart 以后立即启动 FOLLOW_EVENT_PASS 表示主轴穿越设定位置以后启动跟随
MasterDir:INT	主轴运动方向 1 主轴正向运动，-1 主轴负向运动

Pos:DINT	穿越位置，当 event 为 FOLLOW_EVENT_PASS 时有效

GT_GetFollowEvent

读取 Follow 模式启动跟随条件

GT_GetFollowEvent(profile,pEvent,pMasterDir,pPos)	
Profile:INT	轴号
PEvent: POINTER TO INT	启动跟随条件
PMasterDir: POINTER TO INT	主轴运动方向
PPos: POINTER TO DINT	穿越位置，当 event 为 FOLLOW_EVENT_PASS 时有效

GT_FollowSpace

查询 Follow 指定 FIFO 的剩余空间

GT_FollowSpace(profile,pSpace,fifo)	
Profile:INT	轴号
PSpace: POINTER TO INT	读取 FIFO 的剩余空间
Fifo:INT	指定所要查询的 FIFO，默认为 0

GT_FollowData

向 Follow 指定 FIFO 增加数据

GT_FollowData(profile,masterSegment,slaveSegment,type,fifo)	
Profile:INT	轴号
MasterSegment:DINT	主轴位移
SlaveSegment:DINT	从轴位移
Type:INT	数据段类型 FOLLOW_SEGMENT_NORMAL 普通段，默认 FOLLOW_SEGMENT_EVEN 匀速段 FOLLOW_SEGMENT_STOP 减速到 0 段 FOLLOW_SEGMENT_CONTINUE 保持 FIFO 之间速度连续
Fifo:INT	指定存放数据的 FIFO，默认为 0

GT_FollowClear

清除 Follow 指定 FIFO 中的数据。运动状态下该指令无效

GT_FollowClear(profile, fifo)	
Profile:INT	轴号
Fifo:INT	指定需要清除的 FIFO，默认为 0

第四章 运动模式

GT_FollowStart

启动 Follow 模式运动

GT_FollowStart(mask, option)	
Mask:DINT	按位指示需要启动 Follow 运动的轴号 bit0 表示 1 轴，bit1 表示 2 轴，… 当 bit 位为 1 时表示启动对应的轴
Option:DINT	按位指示所使用的 FIFO，默认为 0 bit0 表示 1 轴，bit1 表示 2 轴，… 当 bit 位为 0 时表示对应的轴使用 FIFO1 当 bit 位为 1 时表示对应的轴使用 FIFO2 动态模式下该参数无效

GT_FollowSwich

切换 Follow 所使用的 FIFO

GT_FollowSwitch(mask)	
Mask:DINT	按位指示需要切换 Follow 工作 FIFO 的轴号 bit0 表示 1 轴，bit1 表示 2 轴，… 当 bit 位为 1 时表示切换对应的轴的 FIFO

GT_SetFollowMemory

设置 Follow 运动模式的缓存区大小

GT_SetFollowMemory(profile, memory)	
Profile：INT	规划轴号
Memory:INT	Follow 运动缓存区大小标志： 0：每个 Follow 运动缓存区有 16 段空间。 1：每个 Follow 运动缓存区有 512 段空间。

GT_GetFollowMemory

读取 Follow 运动模式的缓存区大小

GT_GetFollowMemory(profile, pMemory)	
Profile:INT	规划轴号
PMemory:POINTER TO INT	读取 Follow 运动缓存区大小标志

4.5.2 重点说明

在很多应用中，2 轴或多轴之间需要保证位置同步和速度同步。如图所示：

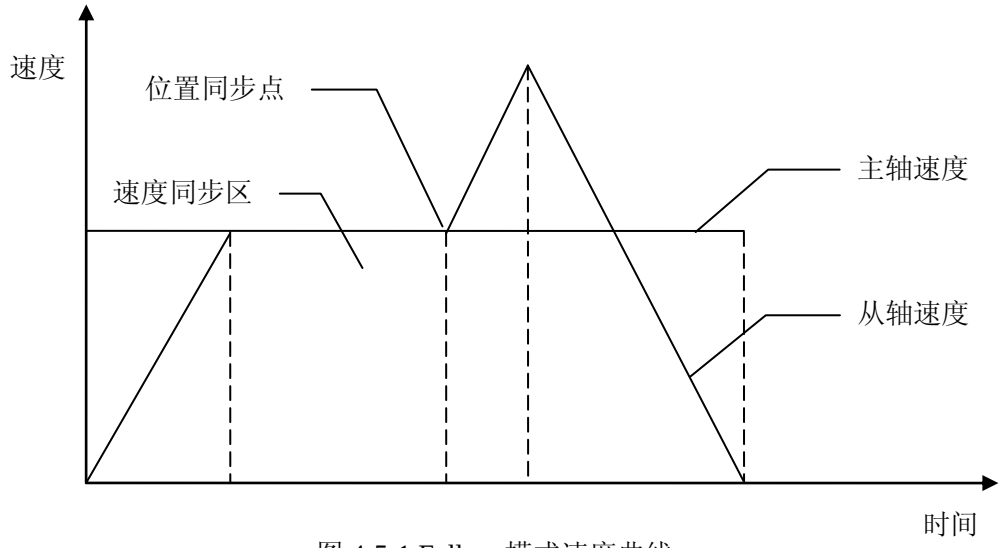


图 4-5-1 Follow 模式速度曲线

位置同步点表示主轴和从轴必须同时到达各自指定位置。

速度同步区表示主轴和从轴之间必须保持准确的速度比。

第 1 段是加速区，从轴逐渐加速，直至到达同步速度。

第 2 段是速度同步区，从轴和主轴保持设定的速度比，速度同步区结束时，主轴和从轴同时到达位置同步点。

第 3 段是加速区，从轴穿越位置同步点以后迅速加速，脱离速度同步区。

第 4 段是减速区，从轴逐渐减速到 0。

为了减少跟随滞后，从轴的轴号应当大于主轴的轴号。

4.5.2.1 数据段类型

Follow 模式的数据段有 4 种类型。

FOLLOW_SEGMENT_NORMAL 表示普通段，FIFO 中第 1 段的起点速度为 0，从第 2 段起每段的起点速度等于上一段的终点速度。

FOLLOW_SEGMENT_EVEN 表示匀速段，FIFO 中各段的段内速度保持不变。

FOLLOW_SEGMENT_STOP 表示停止段，该段的终点速度为 0，起点速度根据段内位移和段内时间计算得到，和上一段的终点速度无关。

FOLLOW_SEGMENT_CONTINUE 表示连续段，FIFO 中第一段的起点速度等于上个 FIFO 的终点速度，从第 2 段起每段的起点速度等于上一段的终点速度。

4.5.2.2 切换 FIFO

Follow 模式下有 2 个独立的 FIFO 用来保存数据。2 个 FIFO 之间可以在运动状态下进行切换。

如图 4-5-2 所示，从轴的运动规律需要从“速度曲线 1”变化到“速度曲线 3”，为了实现从轴速度的平滑过渡，增加了一个“速度曲线 2”的过渡状态。 “速度曲线 2”的起始速度和“速度曲线 1”相等，“速度曲线 2”的终点速度和“速度曲线 3”相等。

为了实现 2 个 FIFO 之间的速度连续，在调用 GT_FollowData 指令时应当将数据类型设置为 FOLLOW_SEGMENT_CONTINUE。

首先将“速度曲线 2”传递到 FIFO2 中，然后调用 GT_FollowSwitch 指令切换工作 FIFO。

当 FIFO1 中的数据全部执行完毕以后才会切换到 FIFO2，并自动清空 FIFO1 中的数据。

当切换到 FIFO2 以后，立即将“速度曲线 3”传递到 FIFO1，然后调用 GT_FollowSwitch 指令切换工作 FIFO。

当 FIFO2 中的数据全部执行完毕以后才会切换到 FIFO1，并自动清空 FIFO2 中的数据。

至此从轴的运动规律从“速度曲线 1”经“速度曲线 2”过渡到“速度曲线 3”。

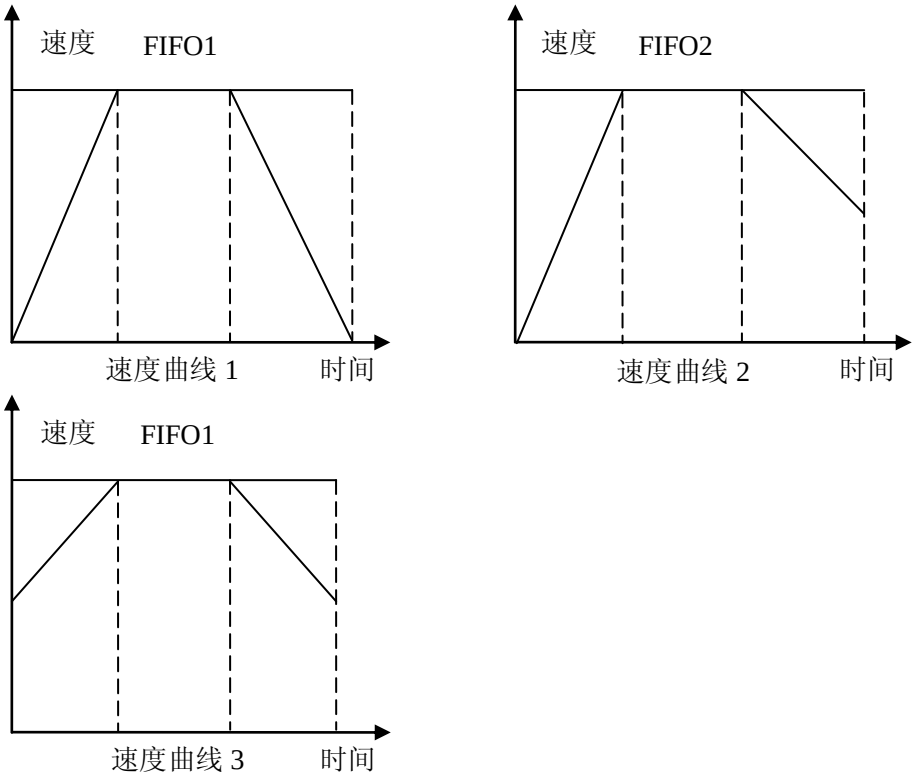


图 4-5-2 Follow 模式切换 FIFO

4.5.3 例程

4.5.3.1 Follow 单 FIFO

该例程主轴为 Jog 模式，从轴为 Follow 模式。从轴的运动规律由 3 段组成，加速段跟随，匀速跟随，减速跟随。如图所示：

```
PROGRAM PLC_PRG
VAR
   Start: BOOL := FALSE;
   Enable: BOOL := TRUE;
   rtn: INT;
   MASTER: INT := 1;
   SLAVE: INT := 2;
   JogPrm: TJogPrm;
   MasterPos: DINT;
   SlavePos: DINT;
   MasterPrfVel: LREAL;
   SlavePrfPos: LREAL;
   SlavePrfVel: LREAL;
   MasterPrfPos: LREAL;
   Stop: BOOL := FALSE;
   StopDone: BOOL := FALSE;
END_VAR
------------------------------------------------------------
IF Start AND Enable THEN
   (*伺服使能*)
   rtn:= GT_ClrSts(MASTER, 1);
   rtn:= GT_ClrSts(SLAVE, 1);
   rtn:= GT_AxisOn(MASTER);
   rtn:= GT_AxisOn(SLAVE);

   (*将主轴设为 JOG 模式*)
   rtn:= GT_PrfJog(MASTER);
   JogPrm.acc:= 0.2;
   JogPrm.dec:= 0.2;
   JogPrm.smooth:= 0.8;
   rtn:= GT_SetJogPrm(MASTER, ADR(JogPrm));
   rtn:= GT_SetVel(MASTER, 40);
   rtn:= GT_Update(SHL(DINT#1, MASTER-1));

   (*从轴设为 Follow 模式*)
   rtn:= GT_PrfFollow(SLAVE, 0);
```

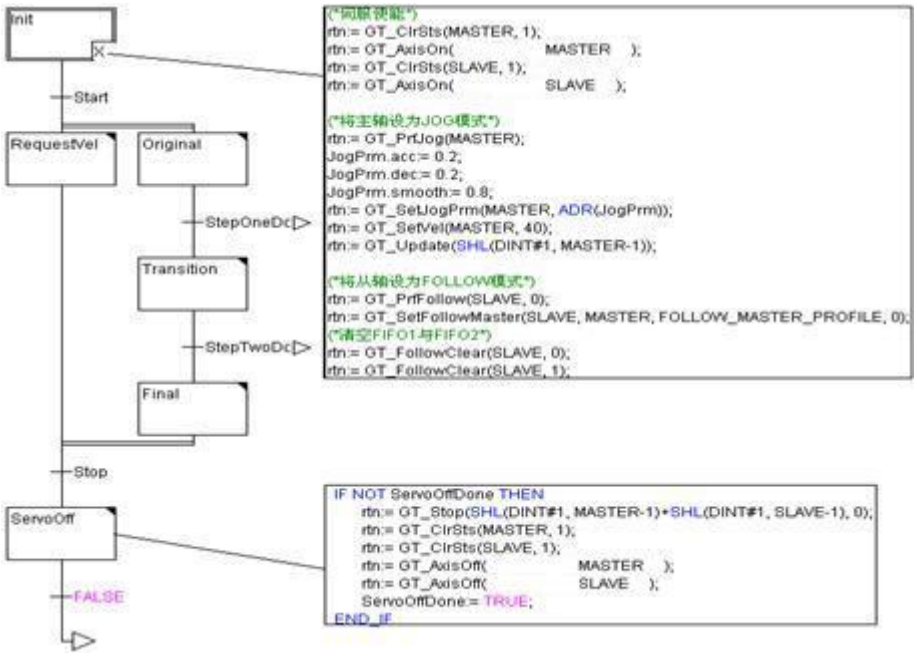
```
    rtn:= GT_SetFollowMaster(SLAVE, MASTER, FOLLOW_MASTER_PROFILE, 0);
    rtn:= GT_FollowClear(SLAVE, 0);
    (*向缓冲区写入数据*)
    MasterPos:= 40000;          (*加速段*)
    SlavePos:=10000;
    rtn:=       GT_FollowData(SLAVE,       MasterPos,       SlavePos,
FOLLOW_SEGMENT_NORMAL, 0);
    MasterPos:= MasterPos+80000;        (*恒速段*)
    SlavePos:= SlavePos+40000;
    rtn:=       GT_FollowData(SLAVE,       MasterPos,       SlavePos,
FOLLOW_SEGMENT_NORMAL, 0);
    MasterPos:=  MasterPos+40000;       (*减速段*)
    SlavePos:= SlavePos+10000;
    rtn:=       GT_FollowData(SLAVE,       MasterPos,       SlavePos,
FOLLOW_SEGMENT_NORMAL, 0);

    (*设置为无限循环，穿越 40000pulse 时启跟踪*)
    rtn:= GT_SetFollowLoop(SLAVE, 0);
    rtn:= GT_SetFollowEvent(SLAVE, FOLLOW_EVENT_PASS, 1, 40000);
    rtn:= GT_FollowStart(SHL(DINT#1, SLAVE-1), 0);

    Enable:= FALSE;
  END_IF
  (*查询规划速度和位置*)
  rtn:= GT_GetPrfVel(MASTER, ADR(MasterPrfVel), 1, 0);
  rtn:= GT_GetPrfVel(SLAVE, ADR(SlavePrfVel), 1, 0);
  rtn:= GT_GetPrfPos(MASTER, ADR(MasterPrfPos), 1, 0);
  rtn:= GT_GetPrfPos(SLAVE, ADR(SlavePrfPos), 1, 0);

  IF Stop AND (NOT StopDone) THEN
    (*关伺服*)
    rtn:= GT_Stop(SHL(DINT#1, SLAVE-1), 0);
    rtn:= GT_Stop(SHL(DINT#1, MASTER-1), 0);
    rtn:= GT_ClrSts(MASTER, 1);
    rtn:= GT_ClrSts(SLAVE, 1);
    rtn:= GT_AxisOff(MASTER);
    rtn:= GT_AxisOff(SLAVE);
    StopDone:= TRUE;
  END_IF
```

4.5.3.2 Follow 双 FIFO 切换

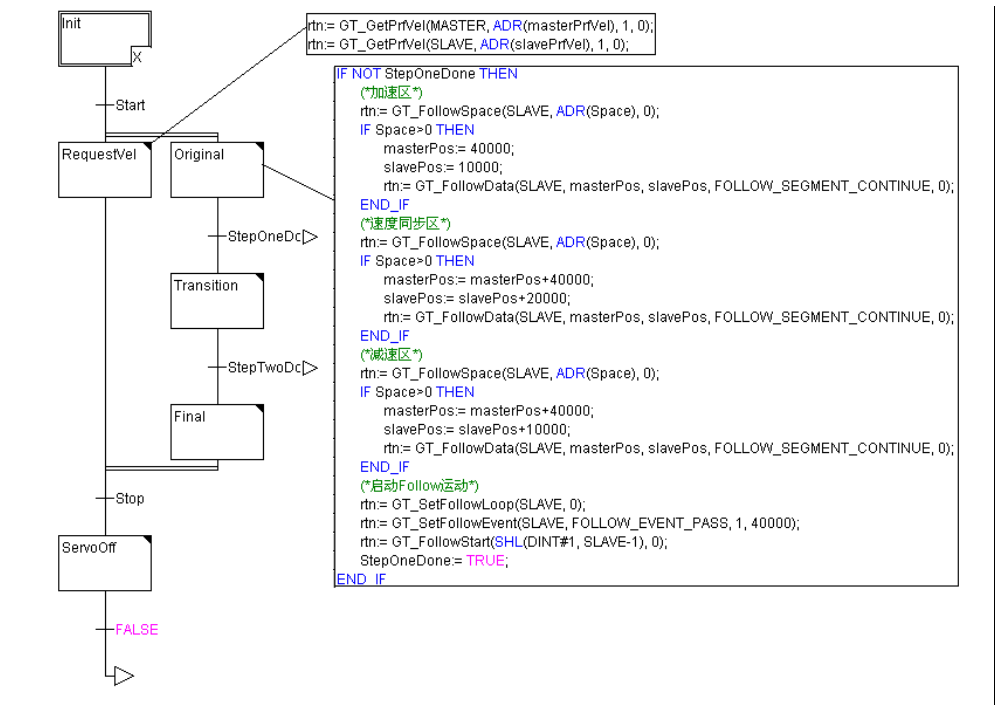
该例程主轴为 Jog 模式，从轴为 Follow 模式，从轴在运动时更换跟随策略。如图所示：

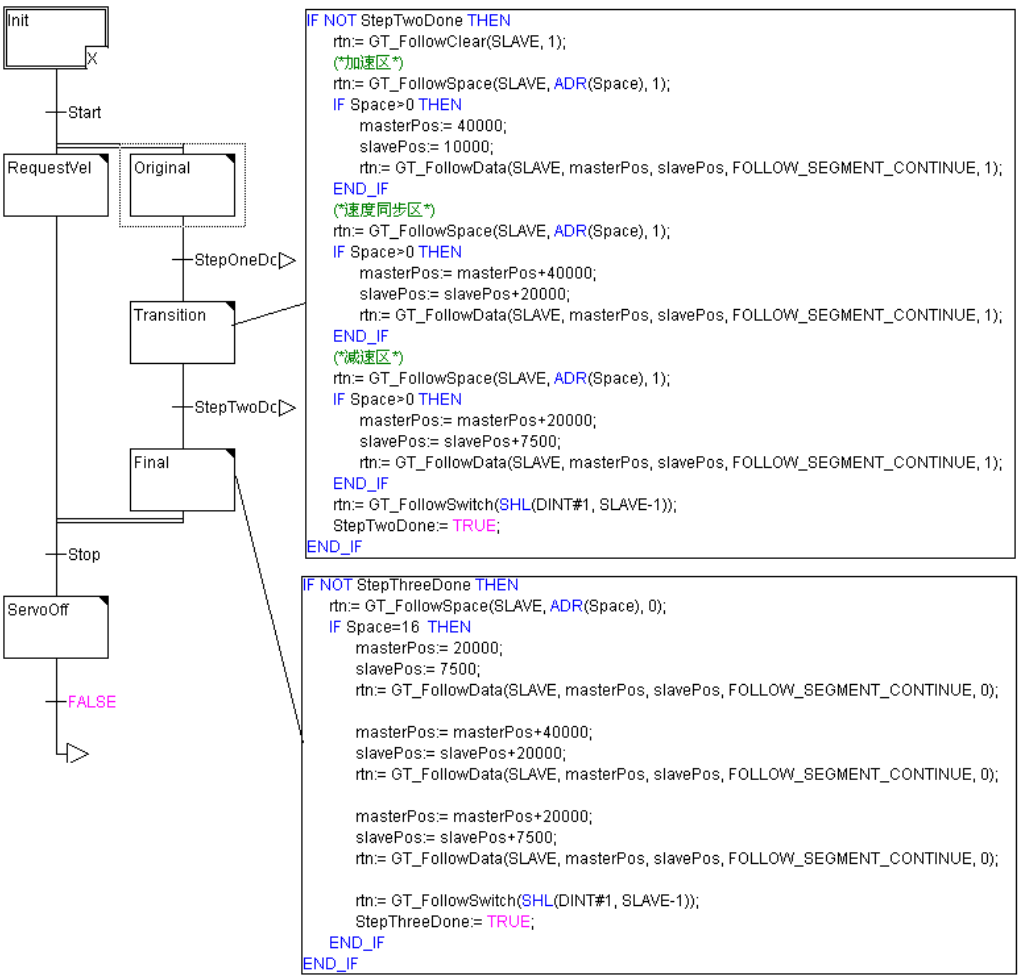
第四章 运动模式

```
0001 PROGRAM PLC_PRG
0002 VAR
0003     rtn: INT;
0004     MASTER: INT := 2;
0005     SLAVE: INT := 4;
0006     JogPrm: TJogPrm;
0007     Start: BOOL := FALSE;
0008     masterPrfVel: LREAL;
0009     slavePrfVel: LREAL;
0010     Stop: BOOL := FALSE;
0011     StepOneDone: BOOL := FALSE;
0012     Space: INT;
0013     masterPos: DINT;
0014     slavePos: DINT;
0015     StepTwoDone: BOOL := FALSE;
0016     StepThreeDone: BOOL := FALSE;
0017     ServoOffDone: BOOL := FALSE;
0018     Switch: BOOL := FALSE;
0019 END_VAR
```



第四章 运动模式





注意：以上几种运动模式的指令中有部分指令的调用比较消耗时间（约 1ms），请在编写程

序时不要反复调用。他们是 GT_Update,GT_PTStart,GT_GearStart,GT_FollowStart.

第五章 访问硬件资源

5.1 访问数字 IO

5.1.1 指令列表

访问数字 IO 指令列表

指令	说明
GT_GetDi	读取数字 IO 输入状态
GT_SetDo	设置数字 IO 输出状态
GT_SetDoBit	按位设置数字 IO 输出状态
GT_GetDo	读取数字 IO 输出状态
GT_SetDiReverseCount	设置数字量输入信号的变化次数的初值
GT_GetDiReverseCount	读取数字量输入信号的变化次数
GT_GetDiRaw	读取数字 IO 输入状态的原始值
GT_SetDoBitReverse	使数字量输出信号输出定时脉冲信号

GT_GetDi

读取数字 IO 输入状态

GT_GetDi(diType,pValue)	
DiType:INT	指定数字 IO 类型 MC_LIMIT_POSITIVE 正限位 MC_LIMIT_NEGATIVE 负限位 MC_ALARM 驱动报警 MC_HOME 原点开关 MC_GPI 通用输入
PValue:POINTER TO DINT	数字 IO 输入状态，按位指示 IO 输入电平 默认情况下，1 表示高电平，0 表示低电平

GT_SetDo

设置数字 IO 输出状态

GT_SetDo(doType,value)	
DoType:INT	指定数字 IO 类型 MC_ENABLE 驱动器使能 MC_CLEAR 报警清除 MC_GPO 通用输出

Value:DINT	按位指示数字 IO 输出电平 默认情况下，1 表示高电平，0 表示低电平

GT_SetDoBit

按位设置数字 IO 输出状态

GT_SetDoBit(doType,doIndex,value)	
DoType:INT	指定数字 IO 类型 MC_ENABLE 驱动器使能 MC_CLEAR 报警清除 MC_GPO 通用输出
DoIndex:INT	输出 IO 的索引
Value:INT	设置数字 IO 输出电平 默认情况下，1 表示高电平，0 表示低电平

GT_GetDo

读取数字 IO 输出状态

GT_GetDo(doType,pValue)	
DoType:INT	指定数字 IO 类型 MC_ENABLE 驱动器使能 MC_CLEAR 报警清除 MC_GPO 通用输出
PValue:POINTER TO DINT	数字 IO 输出状态，按位指示 IO 输出电平 默认情况下，1 表示高电平，0 表示低电平

GT_GetDiReverseCount

读取数字量输入信号的变化次数

GT_GetDiReverseCount(diType,diIndex,pReverseCount,count)	
DiType:INT	指定数字 IO 类型 MC_LIMIT_POSITIVE 正限位 MC_LIMIT_NEGATIVE 负限位 MC_ALARM 驱动报警 MC_HOME 原点开关 MC_GPI 通用输入
diIndex:DINT	数字量输入的索引 取值范围： diType= MC_LIMIT_POSITIVE 时：[1,8] diType= MC_LIMIT_NEGATIVE 时：[1,8] diType= MC_ALARM 时：[1,8]

	diType= MC_HOME 时：[1,8] diType= MC_GPI 时：[1,16]
pReverseCount:POINTER TO DINT	读取的数字量输入的变化次数
Count:INT	读取变化次数的数字量输入的个数，默认为 1 1 次最多可以读取 4 个数字量输入的变化次数。

GT_SetDiReverseCount

设置数字量输入信号的变化次数的初值

GT_SetDiReverseCount(diType,diIndex,pReverseCount,count)	
DiType:INT	指定数字 IO 类型 MC_LIMIT_POSITIVE 正限位 MC_LIMIT_NEGATIVE 负限位 MC_ALARM 驱动报警 MC_HOME 原点开关 MC_GPI 通用输入
diIndex:DINT	数字量输入的索引 取值范围： diType= MC_LIMIT_POSITIVE 时：[1,8] diType= MC_LIMIT_NEGATIVE 时：[1,8] diType= MC_ALARM 时：[1,8] diType= MC_HOME 时：[1,8] diType= MC_GPI 时：[1,16]
pReverseCount:POINTER TO DINT	读取的数字量输入的变化次数
Count:INT	读取变化次数的数字量输入的个数，默认为 1 1 次最多可以读取 4 个数字量输入的变化次数。

GT_GetDiRaw

读取数字 IO 输入状态的原始值

GT_GetDiRaw(diType,pValue)	
DiType:INT	指定数字 IO 类型 MC_LIMIT_POSITIVE(该宏定义为 0) 正限位 MC_LIMIT_NEGATIVE(该宏定义为 1) 负限位 MC_ALARM(该宏定义为 2) 驱动报警 MC_HOME(该宏定义为 3) 原点开关 MC_GPI(该宏定义为 4) 通用输入 MC_ARRIVE(该宏定义为 5) 电机到位信号(仅适用于 CPAC-OtoBox-UC*-4**控制器)
PValue:POINTER TO DINT	数字 IO 输入状态的原始值，按位指示 IO 输入电平 1 表示高电平，0 表示低电平

GT_SetDoBitReverse

使数字量输出信号输出定时脉冲信号

GT_SetDoBitReverse(doType, doIndex, value, reverseTime)	
doType:INT	指定数字 IO 类型 MC_ENABLE(该宏定义为 10)　驱动器使能 MC_CLEAR(该宏定义为 11)　报警清除 MC_GPO(该宏定义为 12)　通用输出
DoIndex:INT	输出 IO 的索引 取值范围： doType=MC_ENABLE 时：[1,8] doType=MC_CLEAR 时：[1,8] doType=MC_GPO 时：[1,16]
Value:INT	设置数字 IO 输出电平 默认情况下，1 表示高电平，0 表示低电平
ReverseTime:INT	维持 value 所设置电平的时间，取值范围：[0,32767]，单位：250μs

5.1.2 重点说明

GT_GetDi 指令可以读取限位、驱动报警、原点、通用输入的输入电平状态。

GT_SetDo 指令可以设置驱动器使能、报警清除、通用输出的输出电平状态。默认情况下由于驱动器使能和轴的关联，不能直接设置驱动器使能的输出电平状态。关于如何设置或取消 do 和轴的关联，请参见“系统配置”。

5.1.3 例程

```
        Alarm, LimitPositive,limitNegative,home,gpi:WORD;
-----------------------------------------
        (*读取驱动器报警电平*)
        GT_GetDi(MC_ALARM,ADR(alarm));
        (*读取正限位开关电平*)
        GT_GetDi(MC_LIMIT_POSITIVE,ADR(limitPositive));
        (*读取负限位开关电平*)
        GT_GetDi(MC_LIMIT_NEGATIVE,ADR(limitNegative));
        (*读取原点开关电平*)
        GT_GetDi(MC_HOME,ADR(home));
        (*读取通用输入电平*)
        GT_GetDi(MC_GPI,ADR(gpi));
```

5.2 访问编码器

5.2.1 指令列表

访问编码器指令列表

指令	说明
GT_GetEncPos	读取编码器位置
GT_GetEncVel	读取编码器速度
GT_SetEncPos	修改编码器位置

GT_GetEncPos

读取编码器位置

GT_GetEncPos(encoder,pValue,count, pClock)	
Encoder:INT	编码器起始轴号
PValue:POINTER TO LREAL	编码器位置
Count:INT	读取的轴数，默认为 1 1 次最多可以读取 8 个编码器轴
PClock:POINTER TO DWORD	读取控制器时钟

GT_GetEncVel

读取编码器速度

GT_GetEncVel(encoder, pValue, count, pClock)	
Encoder:INT	编码器起始轴号
PValue:POINTER TO LREAL	编码器速度
Count:INT	读取的轴数，默认为 1 1 次最多可以读取 8 个编码器轴
PClock: POINTER TO DWORD	读取控制器时钟

GT_SetEncPos

修改编码器位置

GT_SetEncPos(encoder, encPos)	
Encoder:INT	编码器轴号
EncPos:DINT	编码器位置

5.2.2 例程

```
(*读取 8 个编码器轴的位置*)
Enc,vel:array[0..7] of LREAL;
-----------------------------------------------
GT_GetEncPos(1,ADR(enc[0]),8,0);
GT_GetEncVel(1,ADR(vel[0]),8,0);
GT_SetEncPos(1, enc[0]);
```

5.3 访问 DAC

5.3.1 指令列表

访问 DAC 指令列表

指令	说明
GT_SetDac	设置 dac 输出电压
GT_GetDac	读取 dac 输出电压

GT_SetDac

设置 dac 输出电压

GT_SetDac(dac,pValue, count)	
Dac:INT	dac 起始轴号
PValue: POINTER TO INT	输出电压 -32768 对应-10V；32767 对应+10V
Count:INT	设置的轴数，默认为 1 1 次最多可以设置 8 路 dac 输出

GT_GetDac

读取 dac 输出电压

GT_GetDac(dac, pValue, count, pClock)	
dac:INT	dac 起始轴号
PValue: POINTER TO INT	输出电压
Count:INT	读取的轴数，默认为 1 1 次最多可以读取 8 个 dac 轴
PClock: POINTER TO DOWRD	读取控制器时钟

5.4 访问模拟量输入(仅适用于 CPAC-OtoBox-UC*-4**)

5.4.1 指令列表

访问模拟量输入指令列表

指令	说明
GT_GetAdc	读取模拟量输入的电压值
GT_GetAdcValue	读取模拟量输入的数字转换值

GT_GetAdc

读取模拟量输入的电压值

GT_GetAdc(adc, pValue,count, pClock)	
Adc:INT	adc 起始通道号
PValue:POINTER TO LREAL	读取的输入电压值，单位：伏特
Count:INT	读取的通道数，默认为 1 1 次最多可以读取 8 路 adc 输入电压值
PClock:POINTER TO DWORD	读取控制器时钟，默认为：0，即不用读取控制器时钟

GT_GetAdcValue

读取模拟量输入的数字转换值

GT_GetAdcValue(adc,pValue,count, pClock)	
adc:INT	adc 起始通道号
pValue:POINTER TO INT	读取的输入电压值，单位：bit，范围：[-32768,32767]对应的电压值为[-12.5,12.5]伏特
Count:INT	读取的通道数，默认为 1 1 次最多可以读取 8 路 adc 输入电压值
PClock:POINTER TO DWORD	读取控制器时钟，默认为：NULL，即不用读取控制器时钟

第六章 高速硬件捕获

6.1 Home/Index 硬件捕获

6.1.1 指令列表

Home/Index 硬件捕获指令列表

指令	说明
GT_SetCaptureMode	设置编码器捕获捕获方式，并启动捕获
GT_GetCaptureMode	读取编码器捕获方式
GT_GetCaptureStatus	读取编码器捕获状态
GT_SetCaptureSense	设置捕获电平
GT_ClearCaptureStatus	清楚捕获状态

GT_SetCaptureMode

设置编码器捕获捕获方式，并启动捕获

GT_SetCaptureMode(encoder,mode)		
Encoder:INT	编码器轴号	
Mode:INT	编码器捕获模式 CAPTURE_HOME CAPTURE_INDEX	 Home 捕获 Index 捕获

GT_GetCaptureMode

读取编码器捕获方式

GT_GetCaptureMode(encoder, pMode, count, pClock)	
Encoder:INT	编码器起始轴号
PMode: POINTER TO INT	编码器捕获模式
Count:INT	读取的轴数，默认为 1 1 次最多可以读取 8 个编码器轴
PClock: POINTER TO DWORD	读取控制器时钟

GT_GetCaptureStatus

读取编码器捕获状态

GT_GetCaptureStatus(encoder, pStatus, pValue, count, pClock)

EncoderINT	编码器起始轴号
PStatus:POINTER TO INT	读取编码器捕获状态 为 1 时表示对应轴捕获触发
PValue: POINTER TO DINT	读取编码器捕获值 当 Home 捕获或者 Index 捕获触发时，编码器捕获值会自动更新
Count:INT	读取的轴数，默认为 1 1 次最多可以读取 8 个编码器轴
PClock: POINTER TO DWORD	读取控制器时钟

GT_ClearCaptureStatus

清除捕获状态

GT_ClearCaptureStatus(encoder)	
Encoder:INT	编码器轴号

GT_SetCaptureSense

设置捕获电平

GT_SetCaptureSense(encoder,mode,sense)	
Encoder:INT	编码器轴号
Mode:INT	捕获模式 CAPTURE_HOME Home 捕获 CAPTURE_INDEX Index 捕获 CAPTURE_PROBE 探针捕获
Sense:INT	捕获电平，可设置 0 或者 1 0：下降沿触发 1：上升沿触发

6.1.2 重点说明

Home 捕获和 Index 捕获默认都是下降沿触发。

Home 捕获模式下，当出现 Home 下降沿时，FPGA 立刻锁存 Home 开关所对应的编码器位置，同时将该编码器轴的捕获状态标志位置 1，然后退出 Home 捕获模式。

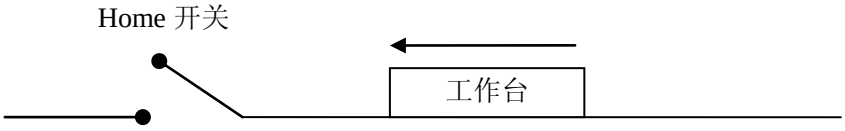
Index 捕获模式下，当出现 Index（编码器 C 相）下降沿时，FPGA 立刻锁存该编码器位置，同时将该编码器轴的捕获触发标志位置 1，然后退出 Index 捕获模式。

当 Home 或 Index 捕获触发以后，重新启动 Home 或 Index 捕获时，会自动清除对应轴的捕获触发标志位。

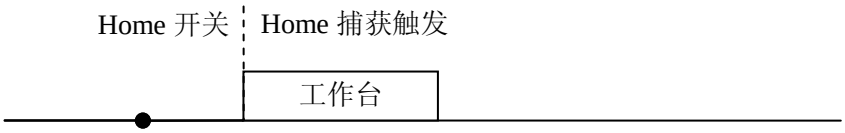
6.2 Home 回原点

6.2.1 重点说明

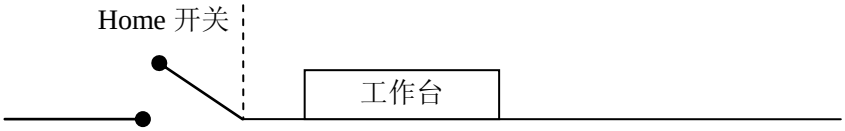
1. 工作台向原点（Home）开关方向运动，启动 Home 捕获。



2. 当 Home 信号产生时，读取 Home 信号触发时工作台的实际位置，并将目标位置修改为“Home 捕获位置+偏移量”。

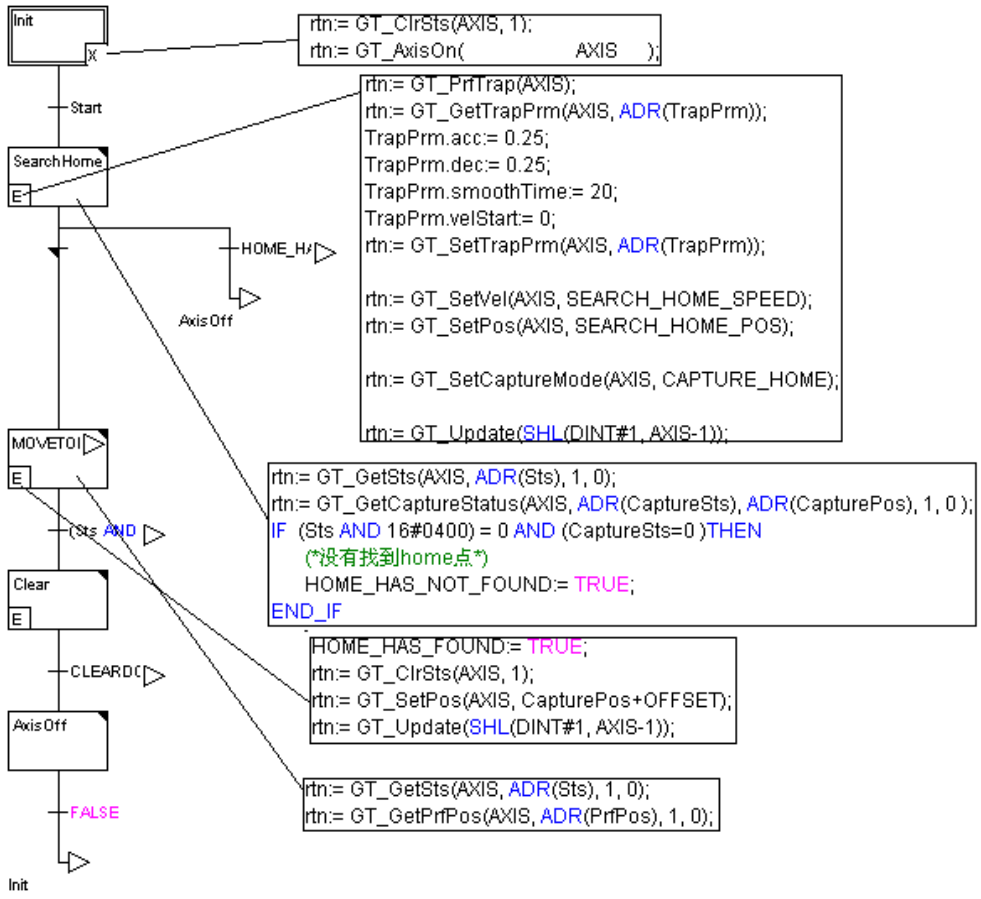


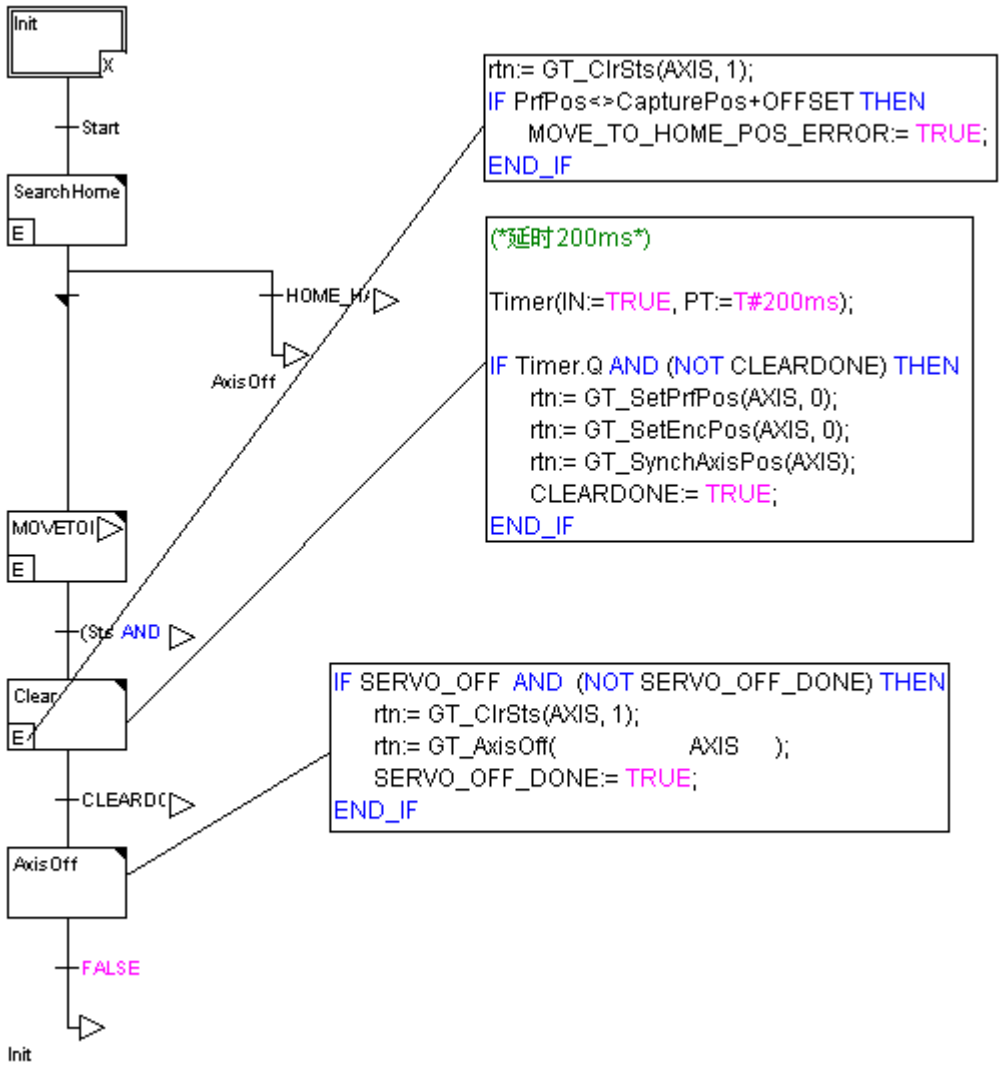
3. 等工作台停稳以后，调用 GT_SetPrfPos 设置机械原点。



6.2.2 例程

```
0001 PROGRAM PLC_PRG
0002 VAR
0003     rtn: INT;
0004     AXIS: INT := 1;
0005     Start: BOOL := FALSE;
0006     TrapPrm: TTrapPrm;
0007     SEARCH_HOME_SPEED: LREAL := 20;
0008     SEARCH_HOME_POS: DINT := -400000;
0009     Sts: DWORD;
0010     CaptureSts: INT;
0011     CapturePos: DINT;
0012     HOME_HAS_NOT_FOUND: BOOL := FALSE;
0013     OFFSET: DINT:= 1000;
0014     PrfPos: LREAL;
0015     MOVE_TO_HOME_POS_ERROR: BOOL := FALSE;
0016     Timer: TON;
0017     DELAY: BOOL := FALSE;
0018     CLEARDONE: BOOL := FALSE;
0019     HOME_HAS_FOUND: BOOL := FALSE;
0020
0021     SERVO_OFF_DONE: BOOL := FALSE;
0022     SERVO_OFF: BOOL := FALSE;
0023 END_VAR
```

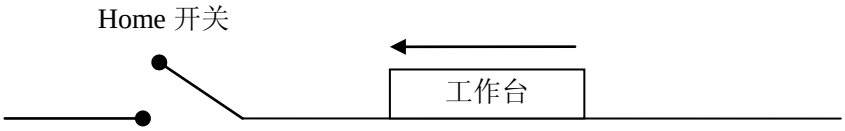




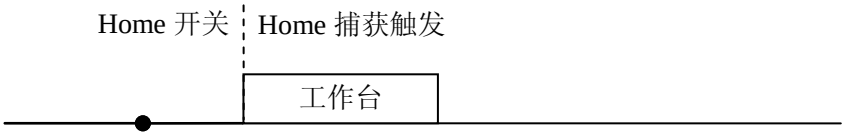
6.3 Home+Index 回原点

6.3.1 重点说明

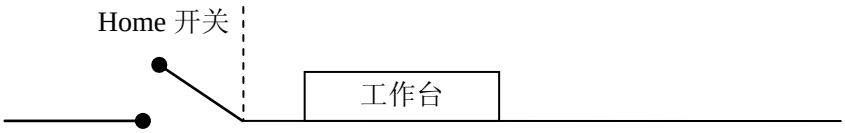
1. 工作台向原点（Home）开关方向运动，启动 Home 捕获。



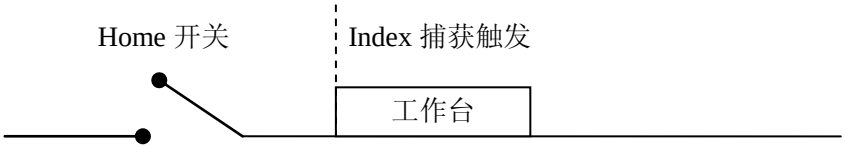
2. 当 Home 信号产生时，读取 Home 信号触发时工作台的实际位置，并将目标位置修改为“Home 捕获位置+偏移量”。



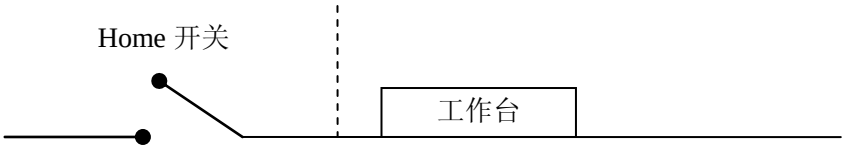
3. 等工作台停稳以后，启动 Index 捕获。



4. 工作台继续运动，寻找 Index 信号。当 Index 信号产生时，读取 Index 信号触发时工作台的实际位置，并将目标位置修改为“Index 捕获位置+偏移量”。



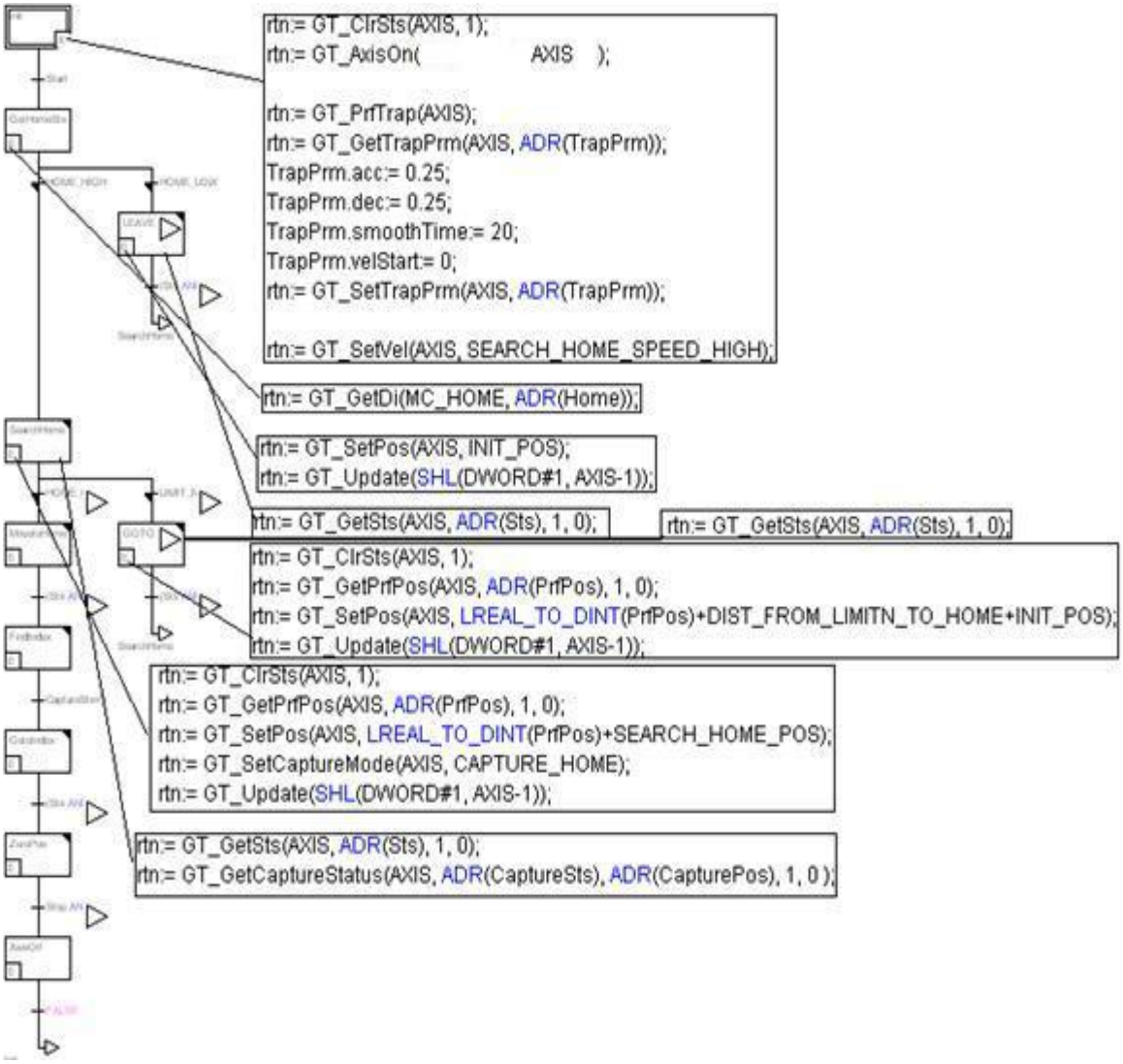
5. 等工作台停稳以后，调用 GT_SetPrfPos 设置机械原点。



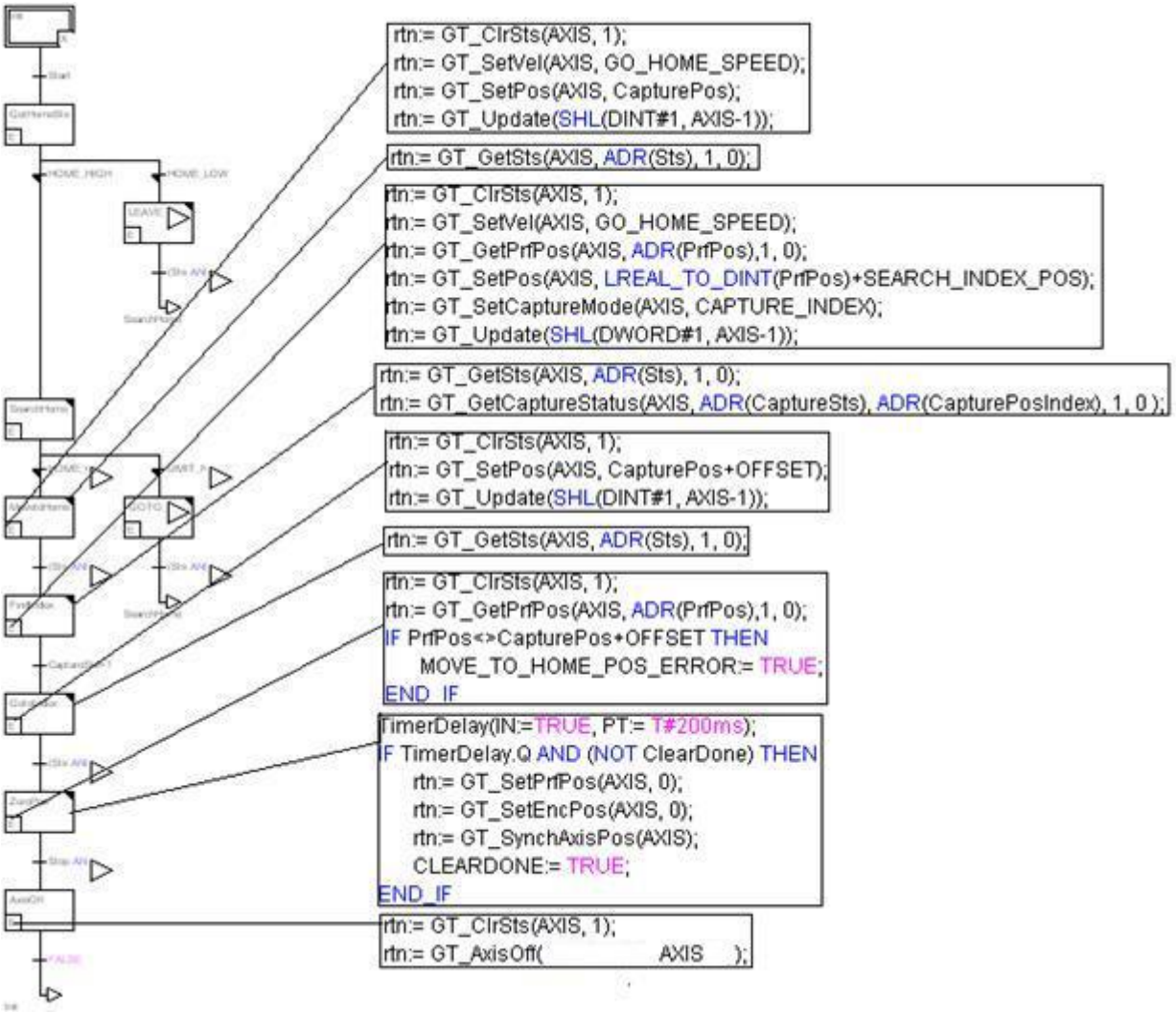
6.3.2 例程

```
0001 PROGRAM PLC_PRG
0002 VAR CONSTANT
0003     AXIS: INT := 1;                (* 轴号 *)
0004     SEARCH_HOME_SPEED_HIGH: LREAL := 20;
0005     SEARCH_HOME_POS: DINT := -800000;
0006     SEARCH_INDEX_POS: DINT := 11000;
0007     INIT_POS: DINT := 20000;
0008     DIST_FROM_LIMITN_TO_HOME: DINT := 20000;
0009     OFFSET: DINT := 2000;
0010     GO_HOME_SPEED: LREAL := 5;
0011 END_VAR
0012 VAR
0013     rtn: INT;
0014     Start: BOOL := FALSE;      (* 启动信号 *)
0015     Home: DWORD := 0;
0016     TrapPrm: TTrapPrm;
0017     Sts: DWORD;
0018     CaptureSts: INT;
0019     CapturePos: DINT;
0020     CapturePosIndex: DINT;
0021     TimerDelay:TON;
0022     PrfPos: LREAL;
0023     MOVE_TO_HOME_POS_ERROR: BOOL := FALSE;
0024     ClearDone: BOOL := FALSE;
0025     Stop: BOOL := FALSE;
0026     EncPos: LREAL;
0027 END_VAR
```

第六章 高速硬件捕获


rtn:= GT_ClrSts(AXIS, 1);
rtn:= GT_AxisOn(AXIS);
rtn:= GT_PrfTrap(AXIS);
rtn:= GT_GetTrapPrm(AXIS, ADR(TrapPrm));
TrapPrm.acc:= 0.25;
TrapPrm.dec:= 0.25;
TrapPrm.smoothTime:= 20;
TrapPrm.velStart:= 0;
rtn:= GT_SetTrapPrm(AXIS, ADR(TrapPrm));
rtn:= GT_SetVel(AXIS, SEARCH_HOME_SPEED_HIGH);
rtn:= GT_GetDi(MC_HOME, ADR(Home));
rtn:= GT_SetPos(AXIS, INIT_POS);
rtn:= GT_Update(SHL(DWORD#1, AXIS-1));
rtn:= GT_GetSts(AXIS, ADR(Sts), 1, 0);
rtn:= GT_GetSts(AXIS, ADR(Sts), 1, 0);
rtn:= GT_ClrSts(AXIS, 1);
rtn:= GT_GetPrfPos(AXIS, ADR(PrfPos), 1, 0);
rtn:= GT_SetPos(AXIS, LREAL_TO_DINT(PrfPos)+DIST_FROM_LIMITN_TO_HOME+INIT_POS);
rtn:= GT_Update(SHL(DWORD#1, AXIS-1));
rtn:= GT_ClrSts(AXIS, 1);
rtn:= GT_GetPrfPos(AXIS, ADR(PrfPos), 1, 0);
rtn:= GT_SetPos(AXIS, LREAL_TO_DINT(PrfPos)+SEARCH_HOME_POS);
rtn:= GT_SetCaptureMode(AXIS, CAPTURE_HOME);
rtn:= GT_Update(SHL(DWORD#1, AXIS-1));
rtn:= GT_GetSts(AXIS, ADR(Sts), 1, 0);
rtn:= GT_GetCaptureStatus(AXIS, ADR(CaptureSts), ADR(CapturePos), 1, 0);

第六章 高速硬件捕获


rtn:= GT_ClrSts(AXIS, 1);
rtn:= GT_SetVel(AXIS, GO_HOME_SPEED);
rtn:= GT_SetPos(AXIS, CapturePos);
rtn:= GT_Update(SHL(DINT#1, AXIS-1));
rtn:= GT_GetSts(AXIS, ADR(Sts), 1, 0);
rtn:= GT_ClrSts(AXIS, 1);
rtn:= GT_SetVel(AXIS, GO_HOME_SPEED);
rtn:= GT_GetPrfPos(AXIS, ADR(PrfPos),1, 0);
rtn:= GT_SetPos(AXIS, LREAL_TO_DINT(PrfPos)+SEARCH_INDEX_POS);
rtn:= GT_SetCaptureMode(AXIS, CAPTURE_INDEX);
rtn:= GT_Update(SHL(DWORD#1, AXIS-1));
rtn:= GT_GetSts(AXIS, ADR(Sts), 1, 0);
rtn:= GT_GetCaptureStatus(AXIS, ADR(CaptureSts), ADR(CapturePosIndex), 1, 0);
rtn:= GT_ClrSts(AXIS, 1);
rtn:= GT_SetPos(AXIS, CapturePos+OFFSET);
rtn:= GT_Update(SHL(DINT#1, AXIS-1));
rtn:= GT_GetSts(AXIS, ADR(Sts), 1, 0);
rtn:= GT_ClrSts(AXIS, 1);
rtn:= GT_GetPrfPos(AXIS, ADR(PrfPos),1, 0);
IF PrfPos<>CapturePos+OFFSET THEN
MOVE_TO_HOME_POS_ERROR:= TRUE;
END_IF
TimerDelay(IN:=TRUE, PT:= T#200ms);
IF TimerDelay.Q AND (NOT ClearDone) THEN
rtn:= GT_SetPrfPos(AXIS, 0);
rtn:= GT_SetEncPos(AXIS, 0);
rtn:= GT_SynchAxisPos(AXIS);
CLEARDONE:= TRUE;
END_IF
rtn:= GT_ClrSts(AXIS, 1);
rtn:= GT_AxisOff(AXIS);

第七章 安全机制

7.1 限位

运动控制器能够通过安装限位开关或者设置软限位来限制各轴的运动范围，如图所示：

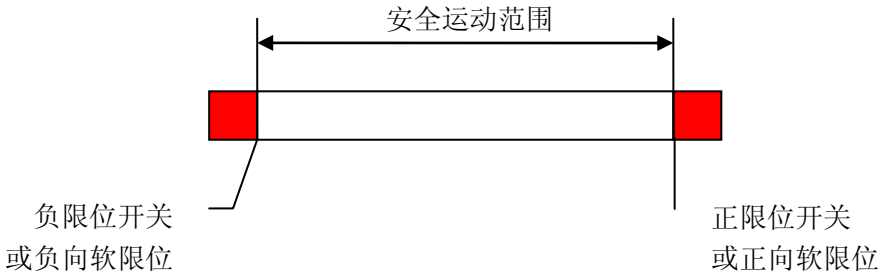


图 7-1-1 轴运动范围

工作台碰到限位开关或者规划位置超越软限位时，运动控制器紧急停止工作台的运动。限位触发以后，运动控制器禁止触发限位方向上运动，同时该轴的限位触发状态置 1。离开限位回到安全运动范围以后，需要调用指令 GT_ClrSts 清除限位触发状态，才能使控制轴回到正常运动状态。

7.1.1 指令列表

软限位指令列表

指令	说明
GT_SetSoftLimit	设置轴正向软限位和负向软限位
GT_GetSoftLimit	读取轴正向软限位和负向软限位

GT_SetSoftLimit

设置轴正向软限位和负向软限位

GT_SetSoftLimit(axis,positive,negative)	
Axis:INT	轴号
Positive:DINT	正向软限位，当规划位置大于该值时，正限位触发 默认值为 0x7fffffff，表示正向软限位无效
Negative:DINT	负向软限位，当规划位置小于该值时，负限位触发 默认值为 0x80000000，表示负向软限位无效

GT_GetSoftLimit

读取轴正向软限位和负向软限位

GT_GetSoftLimit(axis,pPositive,pNegative)	
Axis:INT	轴号
PPositive: POINTER TO DINT	读取正向软限位
PNegative: POINTER TO DINT	读取负向软限位

7.1.2 重点说明

应当在回原点以后再设置软限位。正向软限位必须大于负向软限位。软限位和限位开关可以同时使用，当软限位触发时也会置起限位触发标志。

限位触发以后使用急停加速度紧急停止。默认急停加速度为 1 脉冲/毫秒 ^{2}，如何设置急停加速度请参见“8.8 配置 profile”。

7.1.3 例程

```
PROGRAM PLC_PRG
VAR
    Start : BOOL;
    AXIS:INT;
    Trap: TTrapPrm;
    Sts : WORD;
    prfPos : LREAL;
END_VAR
------------------------------------------------------------
IF Start THEN
   rtn = GT_ClrSts(1,8);
  rtn = GT_SetSoftLimit(AXIS,20000,-20000);
  rtn = GT_PrfTrap(AXIS);
  rtn = GT_GetTrapPrm(AXIS,ADR(trap));
  trap.acc = 0.125;
  trap.dec = 0.125;
  rtn = GT_SetTrapPrm(AXIS,ADR(trap));
  rtn = GT_SetVel(AXIS,50);

  rtn = GT_SetPos(AXIS,1000000);

  rtn = GT_Update(SHL(1,(AXIS-1)));

  Start:=0;
```

```
END_IF

rtn = GT_GetSts(AXIS,ADR(sts),1,0);
rtn = GT_GetPrfPos(AXIS,ADR(prfPos),1,0);
```

7.2 报警

运动控制器提供专用的驱动报警信号输入接口。当检测到驱动器报警信号以后，运动控制器将关闭该轴的伺服使能，急停运动规划，同时该轴报警触发标志置 1。

驱动器报警信号产生以后，应当执行以下操作：

1. 确定引起驱动器报警的原因，并加以改正
2. 复位驱动器
3. 调用GT_ClrSts清除报警，重新回机床原点

7.3 平滑停止和急停

运动控制器的每个轴都可以定义平滑停止 IO 和急停 IO。

当平滑停止 IO 输入为触发电平时（触发电平可以设置），运动控制器自动平滑停止所关联的控制轴，并将轴状态字（bit7）置 1。

当急停 IO 输入为触发电平时（触发电平可以设置），运动控制器自动紧急停止所关联的控制轴，并将轴状态字（bit8）置 1。

IO 平滑停止或者 IO 急停完成以后，必须调用 GT_ClrSts 指令清除停止标志位（bit7 和 bit8），才能继续运动。

7.4 跟随误差极限

对于伺服控制系统而言，在某些异常情况下，电机的实际位置可能与规划位置差距很大。这时通常存在一些危险情况，例如电机故障、编码器 A、B 相信号接反或断线、机械摩擦太大或者机械故障造成电机堵转等。为了及时检测这些情况，增强系统的安全性并延长设备使用寿命，运动控制器提供跟随误差超限自动停止的安全保护机制。

运动控制器在每个控制采样周期内都检查控制轴的实际位置与规划位置的误差是否超越所设定的跟随误差极限。如果位置误差超越所设定的跟随误差极限，运动控制器自动紧急停止该轴的运动，同时该轴跟随误差越限标志置 1。

第八章 运动状态检测

8.1 指令列表

运动状态检测指令列表

指令	说明
GT_GetSts	读取轴状态
GT_ClrSts	清除驱动器报警标志、跟随误差越限标志、限位触发标志 1. 只有当驱动器没有报警时才能清除轴状态字的报警标志 2. 只有当跟随误差正常以后，才能清除跟随误差越限标志 3. 只有当离开限位开关，或者规划位置在软限位行程以内时才能清除轴状态字的限位触发标志
GT_GetPrfMode	读取轴运动模式
GT_GetPrfPos	读取规划位置
GT_GetPrfVel	读取规划速度
GT_GetPrfAcc	读取规划加速度
GT_GetAxisPrfPos	读取轴(axis)的规划位置值
GT_GetAxisPrfVel	读取轴(axis)的规划速度值
GT_GetAxisPrfAcc	读取轴(axis)的规划加速度值
GT_GetAxisEncPos	读取轴(axis)的编码器位置值
GT_GetAxisEncVel	读取轴(axis)的编码器速度值
GT_GetAxisEncAcc	读取轴(axis)的编码器加速度值
GT_GetAxisError	读取轴(axis)的规划位置值和编码器位置值的差值
GT_Stop	停止一个或多个轴的规划运动

运动状态检测指令参数说明

GT_GetSts

读取轴状态

GT_GetSts(axis, pSts,count,pClock)	
Axis:INT	起始轴号
PSts:POINTER TO DINT	32 位轴状态字，详细定义参见重点说明
Count:INT	读取的轴数，默认为 1 1 次最多可以读取 8 个轴的状态
PClock: POINTER TO DWORD	读取控制器时钟

轴状态字的意义如下：

轴状态定义

位	定义
0	保留

1	驱动器报警标志 控制轴连接的驱动器报警时置 1
2	保留
3	保留
4	跟随误差越限标志 控制轴规划位置和实际位置的误差大于设定极限时置 1
5	正限位触发标志 正限位开关电平状态为限位触发电平时置 1 规划位置大于正向软限位时置 1
6	负限位触发标志 负限位开关电平状态为限位触发电平时置 1 规划位置小于负向软限位时置 1
7	IO 平滑停止触发标志 如果轴设置了平滑停止 IO，当其输入为触发电平时置 1，并自动平滑停止该轴
8	IO 急停触发标志 如果轴设置了急停 IO，当其输入为触发电平时置 1，并自动急停该轴
9	电机使能标志 电机使能时置 1
10	规划运动标志 规划器运动时置 1
11	电机到位标志 规划器静止，规划位置和实际位置的误差小于设定误差带，并且在误差带内保持设定时间后，置起到位标志
12~31	保留

GT_ClrSts

清除驱动器报警标志、跟随误差越限标志、限位触发标志

4. 只有当驱动器没有报警时才能清除轴状态字的报警标志
5. 只有当跟随误差正常以后，才能清除跟随误差越限标志

只有当离开限位开关，或者规划位置在软限位行程以内时才能清除轴状态字的限位触发标志

GT_ClrSts(axis,count)	
Axis:INT	轴号
Count:INT	清除的轴数，默认为 1 1 次最多可以清除 8 个轴的异常状态

GT_GetPrfMode

读取轴运动模式

GT_GetPrfMode(profile,pValue,count, pClock)	
Profile:INT	起始规划轴号
PValue: POINTER TO DINT	轴运动模式

	0：梯形曲线，默认 1：Jog 模式 2：PT 模式 3：电子齿轮模式 4：Follow 模式
Count:INT	读取的轴数，默认为 1 1 次最多可以读取 8 个轴的运动模式
PClock: POINTER TO DWORD	读取控制器时钟

GT_GetPrfPos

读取规划位置

GT_GetPrfPos(profile, pValue, count, pClock)	
Profile:INT	起始规划轴号
PValue: POINTER TO LREAL	规划位置
Count:INT	读取的轴数，默认为 1 1 次最多可以读取 8 个轴的规划位置
PClock: POINTER TO DWORD	读取控制器时钟

GT_GetPrfVel

读取规划速度

GT_GetPrfVel(profile, pValue, count, pClock)	
Profile:INT	起始规划轴号
PValue: POINTER TO LREAL	规划速度
Count:INT	读取的轴数，默认为 1 1 次最多可以读取 8 个轴的规划速度
PClock: POINTER TO DWORD	读取控制器时钟

GT_GetPrfAcc

读取规划加速度

GT_GetPrfAcc(profile, pValue, count, pClock)	
Profile:INT	起始规划轴号
PValue: POINTER TO LREAL	规划加速度
Count:INT	读取的轴数，默认为 1 1 次最多可以读取 8 个轴的规划加速度
PClock: POINTER TO DWORD	读取控制器时钟

GT_GetAxisPrfPos

读取轴(axis)的规划位置值

GT_GetAxisPrfPos(axis, pValue, count, pClock)	
Axis:INT	起始轴号
PValue: POINTER TO LREAL	规划位置
Count:INT	读取的轴数，默认为 1 1 次最多可以读取 8 个轴的规划位置
PClock: POINTER TO DWORD	读取控制器时钟

GT_GetAxisPrfVel

读取轴(axis)的规划速度

GT_GetAxisPrfVel(axis, pValue, count, pClock)	
Axis:INT	起始轴号
PValue: POINTER TO LREAL	规划速度
Count:INT	读取的轴数，默认为 1 1 次最多可以读取 8 个轴的规划速度
PClock: POINTER TO DWORD	读取控制器时钟

GT_GetAxisPrfAcc

读取轴(axis)的规划加速度

GT_GetAxisPrfAcc(axis, pValue, count, pClock)	
Axis:INT	起始轴号
PValue: POINTER TO LREAL	规划加速度
Count:INT	读取的轴数，默认为 1 1 次最多可以读取 8 个轴的规划加速度
PClock: POINTER TO DWORD	读取控制器时钟

GT_GetAxisEncPos

读取轴(axis)的编码器位置值

GT_GetAxisEncPos(axis, pValue, count, pClock)	
Axis:INT	起始轴号
PValue: POINTER TO LREAL	编码器位置
Count:INT	读取的轴数，默认为 1 1 次最多可以读取 8 个轴的规划位置
PClock: POINTER TO DWORD	读取控制器时钟

GT_GetAxisEncVel

读取轴(axis)的编码器速度

GT_GetAxisEncVel(axis, pValue, count, pClock)	
Axis:INT	起始轴号
PValue: POINTER TO LREAL	编码器速度
Count:INT	读取的轴数，默认为 1 1 次最多可以读取 8 个轴的规划速度
PClock: POINTER TO DWORD	读取控制器时钟

GT_GetAxisEncAcc

读取轴(axis)的编码器加速度

GT_GetAxisEncAcc(axis, pValue, count, pClock)	
Axis:INT	起始轴号
PValue: POINTER TO LREAL	编码器加速度
Count:INT	读取的轴数，默认为 1 1 次最多可以读取 8 个轴的规划加速度
PClock: POINTER TO DWORD	读取控制器时钟

GT_GetAxisError

读取轴(axis)的规划位置值和编码器位置值的差值

GT_GetAxisError(axis, pValue, count, pClock)	
Axis:INT	起始轴号
PValue: POINTER TO LREAL	轴的规划位置与编码器位置的差值
Count:INT	读取的轴数，默认为 1 1 次最多可以读取 8 个轴的规划位置
PClock: POINTER TO DWORD	读取控制器时钟

GT_Stop

停止一个或多个轴的规划运动

GT_Stop(mask, option)	
Mask:DINT	按位指示需要停止运动的轴号 bit0 表示 1 轴，bit1 表示 2 轴，… 当 bit 位为 1 时表示停止对应的轴
Option:DINT	按位指示停止方式 bit0 表示 1 轴，bit1 表示 2 轴，… 当 bit 位为 0 时表示平滑停止对应的轴

	当 bit 位为 1 时表示急停对应的轴

8.2 重点说明

轴状态定义

驱动器报警标志、限位触发标志、IO 停止、跟随误差越限标志触发以后，不会自动清 0。只有当产生异常的原因已经消除以后，调用 GT_ClrSts 指令才能清除相应的异常标志。

规划运动状态(bit10)只表示理论上的运动状态。置 1 表示处于规划运动状态，清 0 表示处于规划静止状态。由于电机跟随滞后、机械系统震荡等原因，一般在规划静止一段时间以后，机械系统才能完全停止。

电机到位标志(bit11)表示实际到位状态。置 1 表示已经处于规划静止状态(bit10=0)，并且规划位置和编码器位置的误差在设定的误差带内保持了设定时间。当规划运动或者规划位置和编码器位置的误差超出误差带时立即清 0。检测电机到位标志可以保证系统的定位精度，应当根据机械系统的实际情况设置合适的到位误差带和误差带保持时间。如果到位误差带设置的太小，或者误差带保持时间太长，都会使到位时间增长，影响加工效率。

使用电机到位标志(bit11)必须注意以下几点：

1. Axis 正确关联编码器，并且编码器方向和规划运动方向必须一致
2. 正确设置到位误差带，默认情况下到位误差带无效
3. 调用 GT_SetPrfPos 或者 GT_SetEncPos 指令之后应当调用 GT_SynchAxisPos 指令

轴(axis)状态读取的相关指令

在 3.2.1 中关于轴(axis)的配置部分曾经提到，规划器(profile)和编码器(encoder)的输出值可以通过 axis 进行当量变换之后，再输出。上述与 axis 相关的状态读取指令主要用来读取 profile 和 encoder 的输出值经过当量变换之后的值。

其中，GT_GetAxisPrfPos、GT_GetAxisPrfVel 和 GT_GetAxisPrfAcc 用来读取 profile 输出值经过当量变换之后的规划位置、规划速度和规划加速度。GT_GetAxisEncPos、GT_GetAxisEncVel 和 GT_GetAxisEncAcc 用来读取 encoder 输出值经过当量变换之后的编码器位置值。GT_GetAxisError 指令用来读取 profile 经过当量变换之后的规划位置与 encoder 经过当量变换之后的编码器位置的差值。

例程

该例程示例到位标志的使用方法。一个轴运动到位以后，启动另一个轴的运动。

第九章 运动程序

```
PROGRAM Main
VAR CONSTANT
    AXIS_X:INT:=1;
    AXIS_Y:INT:=2;
END_VAR
VAR
    Enable:BOOL:=TRUE;
    Rtn:INT;
    Pid:Tpid;
    Trap: TTrapPrm;
    Sts:DINT;
    posX,posY:DINT;
    prfPos,prfVel:LREAL;
    Current_axis:INT:=1;
    X_DONE,Y_DONE:BOOL;
END_VAR
    ----------------------------------------------------------------------------------------
IF Enable THEN
    (*读取X轴PID参数*)
    Rtn:= GT_GetPid(AXIS_X,1,ADR(pid));
    pid.kp:= 10;
    (*更新X轴PID参数*)
    Rtn:= GT_SetPid(AXIS_X,1,ADR(pid));

    (*读取Y轴PID参数*)
    Rtn:= GT_GetPid(AXIS_Y,1,ADR(pid));
    pid.kp:= 10;
    (*更新Y轴PID参数*)
    Rtn:= GT_SetPid(AXIS_Y,1,ADR(pid));

    (*X轴伺服使能*)
    Rtn:= GT_AxisOn(AXIS_X);

    (*Y轴伺服使能*)
    Rtn:= GT_AxisOn(AXIS_Y);

    (*X轴规划位置清零*)
    Rtn:= GT_SetPrfPos(AXIS_X,0);

    (*X轴编码器位置清零*)
    Rtn:= GT_SetEncPos(AXIS_X,0);

    (*根据Profile的规划位置重新计算Axis的规划位置*)
    (*根据Encoder的实际位置重新计算Axis的实际位置*)
```

```
Rtn:= GT_SynchAxisPos(SHL(DINT#1, AXIS_X-1));

(*设置X轴到位误差带*)
Rtn:= GT_SetAxisBand(AXIS_X,20,5);

(*Y轴规划位置清零*)
Rtn:= GT_SetPrfPos(AXIS_Y,0);

(*Y轴编码器位置清零*)
Rtn:= GT_SetEncPos(AXIS_Y,0);

(*根据Profile的规划位置重新计算Axis的规划位置*)
(*根据Encoder的实际位置重新计算Axis的实际位置*)
Rtn:= GT_SynchAxisPos(SHL(1,AXIS_Y-1));

(*设置Y轴到位误差带*)
Rtn:= GT_SetAxisBand(AXIS_Y,20,5);

(*X轴设为点位模式*)
Rtn:= GT_PrfTrap(AXIS_X);

(*读取X轴点位运动参数*)
Rtn:= GT_GetTrapPrm(AXIS_X,ADR(trap));
trap.acc:= 1;
trap.dec:= 0.5;
(*设置X轴点位运动参数*)
Rtn:= GT_SetTrapPrm(AXIS_X,ADR(trap));

(*设置X轴的目标速度*)
Rtn:= GT_SetVel(AXIS_X,10);

(*Y轴设为点位模式*)
Rtn:= GT_PrfTrap(AXIS_Y);
(*读取Y轴点位运动参数*)
Rtn:= GT_GetTrapPrm(AXIS_Y,ADR(trap));
trap.acc:= 1;
trap.dec:= 0.5;
(*设置Y轴点位运动参数*)
Rtn:= GT_SetTrapPrm(AXIS_Y,ADR(trap));
(*设置Y轴的目标速度*)
Rtn:= GT_SetVel(AXIS_Y,10);
posX:= 10000;
posY:= 20000;
(*设置X轴目标位置*)
```

```
        Rtn:= GT_SetPos(AXIS_X,posX);
        (*启动X轴的运动*)
        Rtn:= GT_Update(SHL(DINT#1, AXIS_X-1));
        Enable:=FALSE;
END_IF
(*等待X轴进入误差带*)
GT_GetSts(Current_axis,ADR(sts),1,0);
GT_GetPrfPos(Current_axis,ADR(prfPos)1,0);
GT_GetPrfVel(Current_axis,ADR(prfVel)1,0);
IF 16#800 = ( sts AND 16#800 ) ) THEN
        IF NOT(X_DONE) THEN
                (*设置Y轴目标位置*)
                Rtn:= GT_SetPos(AXIS_Y,posY);
                Rtn:= GT_Update(2);
                Current_axis:=AXIS_Y;
                X_Done:=TRUE;
        ELSE
                Y_Done:=TRUE;
        END_IF
END_IF
```

第九章 运动程序

详见 OtoStudio 高级运动控制编程手册第五章

第十章 其它指令

10.1 复位运动控制器

GT_Reset

打开/关闭运动控制器指令列表

指令	说明
GT_Reset	复位运动控制器

在使用运动控制器之前，首先需要使用 GT_Open()指令打开运动控制器，和运动控制器建立通讯；在使用运动控制器结束之后，退出应用程序时，应当调用 GT_Close()指令关闭运动控制器。

调用 GT_Reset()指令将使运动控制器的所有寄存器恢复到默认状态，一般在打开运动控制器之后调用该指令。

GT_SetCardNo()用于切换当前运动控制器卡号，一台计算机上使用多个运动控制器时，该指令用于指定当前运动控制器。当指令执行成功后，之后的所有指令只操作当前运动控制器。在多运动控制器系统中，每个运动控制器在操作系统启动时被分配一个卡号(0~15)，用于区别不同控制卡。卡号分配原则遵循 PNP 规则，第一个被系统识别的运动控制器卡号为 0，所以在硬件配置没有改变的情况下，系统每次分配的卡号是相同的。

10.2 读取固件版本号

GT_GetVersion

函数说明

GT_GetVersion(pVersion)	
pVersion:POINTER TO POINTER TO BYTE	读取的运动控制器的固件版本号字符串

为了方便用户核对运动控制器固件版本，提供 GT_GetVersion()指令来读取运动控制器固件版本号，版本号是一个含有 18 个字符的字符串：aaa bbbbbb ccc dddddd。具体的定义如下表：

aaa	固件 1 的版本号，如 100，即表示版本号为：1.00
bbbbbb	固件 1 的版本号的生成时间，如 090908，即表示该版本生成于：2009 年 9 月 8 日
ccc	固件 2 的版本号

dddddd	固件 2 的版本号的生成时间

例程如下：

```
PROGRAM PLC_PRG
VAR
    version:LREAL;
    firmwareVersion:STRING(18);
    pVersion:POINTER TO STRING(18);
    rtn:INT;
    i:INT;
END_VAR
-----------------------------------------------------
rtn:=GetVersion(ADR(version));
rtn:=GT_GetVersion(ADR(pVersion));
firmwareVersion:=pVersion^;
```

10.3 读取系统时钟

GT_GetClock

函数说明

GT_GetClock(pClock, pLoop)	
pClock:POINTER TO DWORD	读取的运动控制器的时钟，单位：毫秒
Ploop:POINTER TO DWORD	内部使用，默认为：0，即不读取该值

运动控制器上电初始化之后，内部计数时钟从 0 开始计数，每 1 毫秒增加 1，通过 GT_GetClock()指令可以读取该计数时钟的值。调用 GT_Reset()指令将会使计数时钟值清零。

GT_GetClockHighPrecision

函数说明

GT_GetClockHighPrecision(pClock, pLoop)	
pClock:POINTER TO DWORD	读取的运动控制器的时钟，单位：250 微妙

运动控制器上电初始化之后，内部计数时钟从 0 开始计数，每 250 微秒增加 1，通过 GT_GetClockHighPrecision()指令可以读取该计数时钟的值。调用 GT_Reset()指令将会使计数时钟值清零。

10.4 打开/关闭电机使能信号

GT_AxisOn

GT_AxisOn(axis)	
Axis:INT	打开伺服使能的轴的编号

调用 GT_AxisOn()指令将打开指定控制轴所连电机的伺服使能信号，使指定控制轴进入控制状态。

GT_AxisOff

GT_AxisOff(axis)	
Axis:INT	关闭伺服使能的轴的编号

10.5 维护位置值

函数列表

指令	说明
GT_SetPrfPos	修改指定轴的规划位置
GT_SynchAxisPos	axis 合成规划位置和所关联的 profile 同步 axis 合成编码器位置和所关联的 encoder 同步
GT_ZeroPos	清零规划位置和实际位置，并进行零漂补偿

GT_SetPrfPos

修改指定轴的规划位置

GT_SetPrfPos(profile, prfPos)	
Profile:INT	规划轴编号
PrfPos:DINT	设置的规划位置的值

GT_SynchAxisPos

axis 合成规划位置和所关联的 profile 同步
axis 合成编码器位置和所关联的 encoder 同步

GT_SynchAxisPos(mask)	
Mask:DINT	按位标识需要进行位置同步的轴号 Bit0 对应 1 轴，bit1 对应 2 轴，… 0：表示不需要进行位置同步

	1：需要进行位置同步

GT_ZeroPos

清零规划位置和实际位置，并进行零漂补偿

GT_ZeroPos(axis, count)	
Axis:INT	需要位置清零的起始轴号
Count:INT	需要位置清零的轴数

第三章系统配置里提到，axis 含有 profile 和 encoder 的当量变换的功能，如果调用了 GT_SetPrfPos()指令或者 GT_SetEncPos()指令之后，profile 的输出值或者 encoder 的输出发生了变化，如果需要将 axis 当量变换之后的值与 profile 或者 encoder 的值同步，需要调用 GT_SynchAxisPos()指令。

10.6 电机到位检测

指令	说明
GT_SetAxisBand	设置轴到位误差带 规划器静止，规划位置和实际位置的误差小于设定误差带，并且在误差带内保持设定时间后，置起到位标志
GT_GetAxisBand	读取轴到位误差带

GT_SetAxisBand

设置轴到位误差带
规划器静止，规划位置和实际位置的误差小于设定误差带，并且在误差带内保持设定时间后，置起到位标志

GT_SetAxisBand(axis, band, time)	
Axis:INT	轴号
Band:DINT	误差带大小，单位：脉冲
Time:DINT	误差带保持时间，单位：250 微秒

GT_GetAxisBand

读取轴到位误差带

GT_GetAxisBand(axis, pBand, pTime)	
Axis:INT	轴号
PBand:POINTER TO DINT	读取误差带大小
PTime:POINTER TO DINT	读取误差带保持时间

用户使用伺服电机时，由于伺服电机在运动的过程中可能会存在运动滞后，会出现规划停止，而实际位置并没有到位的情况。用户可以使用运动控制器的运动到位检测功能来判断电机是否实际到位。运动控制器默认该功能是无效的，当调用 GT_SetAxisBand()指令设置了相应的误差带和保持时间之后，该功能生效。

该功能生效后，当规划器处于静止状态，即轴状态寄存器 bit10 为 0(参见 4.2.1)，并且规划位置和编码器位置的误差在设定的误差带内保持了设定时间，轴状态寄存器 bit11 将被置 1(参见 4.2.1)。当规划器运动，或者规划位置和编码器位置的误差超出误差带时立即清 0。检测电机到位标志可以保证系统的定位精度，应当根据机械系统的实际情况设置合适的到位误差带和误差带保持时间。如果到位误差带设置的太小，或者误差带保持时间太长，都会使到位时间增长。

使用电机到位检测功能必须注意以下几点：

1. axis 正确关联编码器，并且编码器方向和规划运动方向必须一致。
2. 正确设置到位误差带，默认情况下到位误差带无效
3. 调用 GT_SetPrfPos 或者 GT_SetEncPos 指令之后应当调用 GT_SynchAxisPos 指令

10.7 设置 PID 参数

函数列表

指令	说明
GT_SetControlFilter	设定 PID 索引，支持 3 组 PID 参数
GT_GetControlFilter	读取当前 PID 索引
GT_SetPid	设置 PID 参数
GT_GetPid	读取 PID 参数

GT_SetControlFilter

设定 PID 索引，支持 3 组 PID 参数

GT_SetControlFilter(control,index)	
Control:INT	伺服控制器编号
Index:INT	伺服控制参数的索引号，取值范围：[1,3]

GT_GetControlFilter

读取当前 PID 索引

GT_GetControlFilter(control,pIndex)	
Control:INT	伺服控制器编号
PIndex:POINTER TO INT	读取的伺服控制参数的索引号

GT_SetPid

设置 PID 参数

GT_SetPid(control,index,pPid)	
Control:INT	伺服控制器编号
Index:INT	伺服控制参数的索引号，取值范围：[1,3]
PPid:POINTER TO TPid	设置 PID 参数 TYPE TPid : STRUCT kp:LREAL; (* 比例增益*) ki:LREAL; (* 积分增益*) kd:LREAL; (* 微分增益*) kvff:LREAL; (* 速度前馈*) kaff:LREAL; (* 加速度前馈*) integralLimit:DINT; (* 积分项饱和极限*) derivativeLimit:DINT; (* 微分项饱和极限*) limits:INT; (* 输出电压饱和极限*) END_STRUCT END_TYPE

GT_GetPid

读取 PID 参数

GT_GetPid(control,index,pPid)	
Control:INT	伺服控制器编号
Index:INT	伺服控制参数的索引号，取值范围：[1,3]
PPid:POINTER TO TPid	读取 PID 参数

运动控制器支持设置 3 组 PID 参数，并且支持运动时在各组 PID 参数之间进行切换，通过调用 GT_SetControlFilter()指令来切换 PID 参数。

TPid

设置 PID 参数
TYPE TPid :
STRUCT
kp:LREAL; (* 比例增益*)
ki:LREAL; (* 积分增益*)
kd:LREAL; (* 微分增益*)
kvff:LREAL; (* 速度前馈*)
kaff:LREAL; (* 加速度前馈*)

```
        integralLimit:DINT;                    (*  积分项饱和极限*)
        derivativeLimit:DINT;                  (*  微分项饱和极限*)
        limits:INT;                        (*  输出电压饱和极限*)
END_STRUCT
END_TYPE
```

10.8 反向间隙补偿

该功能请使用 Backlash.lib

函数列表

指令	说明
GT_SetBacklash	设置反向间隙补偿的相关参数
GT_GetBacklash	读取反向间隙补偿的相关参数

GT_SetBacklash

设置反向间隙补偿的相关参数

GT_SetBacklash(axis, compValue, compChangeValue, compDir)	
Axis:INT	需要进行反向间隙补偿的轴的编号，取值范围：[1,8]
CompValue:DINT	反向间隙补偿值，当为 0 时表示没有使能反向间隙补偿功能，取值范围：[0, 1073741824]，单位：脉冲
CompChangeValue:LREAL	反向间隙补偿的变化量，取值范围：[0, 1073741824]，单位：脉冲/毫秒 当该参数的值为 0 或者大于等于 compValue 时，则反向间隙的补偿量将瞬间叠加在规划位置上，没有渐变的过程
CompDir:DINT	反向间隙补偿方向 0：只补偿负方向，当电机向负方向运动时，将施加补偿量，当电机向正方向运动时，不施加补偿量 1：只补偿正方向，当电机向正方向运动时，将施加补偿量，当电机向负方向运动时，不施加补偿量

GT_GetBacklash

读取反向间隙补偿的相关参数

GT_GetBacklash(axis,pCompValue,pCompChangeValue,pCompDir)

Axis:INT	查询的轴号，取值范围：[1,8]
PCompValue:POINTER TO DINT	读取的反向间隙补偿值
PCompChangeValue:POINTER TO LREAL	读取的反向间隙补偿值的变化量
PCompDir:DINT	读取的反向间隙补偿的补偿方向

反向间隙误差是指由于传动链中机械间隙的存在，执行部件在运动过程中，从正向运动变为负向运动时，或者从负向运动变为正向运动时，执行部件的运动量与理论量存在误差，最后将反映为叠加至工件上的加工精度的误差。为了消除反向间隙误差，提高机器的加工精度和定位精度，该控制器提供了反向间隙误差补偿功能。用户只要在初始化的时候调用相应的反向间隙误差补偿功能指令 GT_SetBacklash()设置了相应的参数，反向间隙误差补偿功能将会生效；也可以通过指令 GT_SetBacklash()来关闭反向间隙误差补偿功能。

用户可以设置反向间隙误差补偿量的叠加速度，可以瞬间(一个控制周期内)叠加到输出量上，也可以选择以一定的速度叠加到输出量上。通过设置指令 GT_SetBacklash()的 compChangeValue 参数来实现，当 compChangeValue 的值为 0 或者大于等于 compValue 的值时，则表示误差补偿量将瞬间叠加到输出量上，当为其他值时，表示误差补偿量的叠加速度，单位是：pulse/ms。

反向间隙误差补偿方向指的是，反向间隙误差补偿是沿正方向补偿还是沿负方向补偿。如果指令 GT_SetBacklash()的参数 compDir 参数设置为 0 时，则只有电机从正方向转为负方向运动时，反向间隙补偿量生效，当电机向正方向运动时，反向间隙补偿量为 0。如果用户设置了补偿量的变化速度，则从正方向转为负方向时，补偿量以 compChangeValue 的速度叠加到 compValue 的值，当从负方向转为正方向时，补偿量从 compValue 以 compChangeValue 的速度减小为 0。这种情况下，用户应该在回零之后，让工作台向正方向运动一定的距离，以保证正方向运动没有间隙存在。

当指令 GT_SetBacklash()的参数 compDir 参数设置为 1 时，则只有电机从负方向转为正方向运动时，反向间隙补偿量生效，当电机向负方向运动时，反向间隙补偿量为 0。如果用户设置了补偿量的变化速度，则从负方向转为正方向时，补偿量以 compChangeValue 的速度叠加到 compValue 的值，当从正方向转为负方向时，补偿量从 compValue 以 compChangeValue 的速度减小为 0。这种情况下，用户应该在回零之后，让工作台向负方向运动一定的距离，以保证负方向运动没有间隙存在。

反向间隙补偿量会直接叠加到运动控制器的输出量上，当用户读取规划位置时，不会读到反向间隙补偿量。但是用户如果读取电机编码器的值，将会读到反向间隙的补偿量。

10.9 自动回原点功能

该功能请使用 Home.lib

函数列表

指令	说明
GT_HomeInit	初始化自动回原点功能
GT_Home	启动自动回原点功能

GT_Index	设置自动回原点功能为 home+index 模式
GT_HomeStop	启动原点停止功能
GT_HomeSts	查询自动回原点的运行状态

GT_ HomeInit

初始化自动回原点功能

GT_ Home

启动自动回原点功能

GT_Home(short axis,long pos,double vel,double acc,long offset)	
axis	需要进行自动回原点操作的轴号，取值范围：[1,8]
pos	搜索距离，以当前位置为起点，搜索距离为正时向正方向搜索，搜索距离为负时向负方向搜索，单位：脉冲
vel	搜索速度，单位：脉冲/毫秒
acc	搜索加速度，单位：脉冲/(毫秒*毫秒)
offset	原点偏移量，当原点信号触发时，将当前轴目标位置自动更新为“原点位置＋原点偏移”。如果原点偏移量为 0，当原点信号触发时，首先平滑停止减速到 0，然后返回原点。

GT_ Index

设置自动回原点功能为 home+index 模式

GT_Index(short axis,long pos,long offset)	
axis	需要进行自动 home+index 回原点操作的轴号，取值范围：[1,8]
pos	Index 信号的搜索距离。Home 信号触发时，以 Home 位置为起点搜索 Index，搜索距离为正时向正方向搜索，搜索距离为负时向负方向搜索。Index 搜索速度是 Home 搜索速度的一半，Index 搜索加速度和 Home 搜索加速度相同。
offset	Index 信号偏移量，当 index 信号触发时，将当前轴目标位置自动更新为“index 位置＋index 偏移”。如果 index 偏移量为 0，当 index 信号触发时，首先平滑停止减速到 0，然后返回 index 位置。

GT_ HomeStop

启动原点停止功能

GT_HomeStop(short axis,long pos,double vel,double acc)	
axis	需要进行原点急停操作的轴号，取值范围：[1,8]

pos	搜索距离，以当前位置为起点，搜索距离为正时向正方向搜索，搜索距离为负时向负方向搜索，单位：脉冲
vel	搜索速度，单位：脉冲/毫秒
acc	搜索加速度，单位：脉冲/(毫秒*毫秒)

GT_ HomeSts

查询自动回原点的运行状态

GT_HomeSts(short axis,unsigned short *pStatus)	
axis	需要查询自动回原点状态的轴号，取值范围：[1,8]
pStatus	查询到的状态值 0：自动回原点操作正在执行 1：自动回原点操作成功执行完毕

重点说明

1 使用前的注意事项

1．在实际应用中，使用者在进行多轴同时回零操作时，需确保不会因为同时回零而造成多轴之间干涉。

2．在使用本指令时，请确保没有同时使用**运动程序**(参见第九章)的功能。自动回原点功能与控制器的运动程序功能共用了运动控制器内的一些资源，所以这两个功能不能同时使用，如果在运动程序的使用过程中使用了自动回原点功能，则再次使用运动程序时，需要重新下载运动程序代码以及相关的初始化操作。如果在使用自动回原点功能的使用过程中使用了运动程序功能，则再次使用自动回原点功能前需要再次调用 GT_HomeInit()来进行自动回原点功能的初始化。

3．在使用过程中，需保证电机规划位置与编码器位置同向，即当规划位置增大时，编码器位置也在增大。

4．在自动回原点过程执行完毕后，进行零点清除前，需要设置一个延时操作，防止由于电机未到位而产生误差，延时操作一般要大于 100ms。

2 使用方法

1．自动回原点指令函数共有五个函数，GT_HomeInit()，GT_Home()，GT_Index()，GT_HomeStop()，GT_HomeSts()。其作用分别是 Home 回零模式初始化，Home 模式回零，Home+Index 模式回零，原点急停指令，以及查询回原点状态指令。

2．自动回原点功能的初始化

// 在进行所有轴的回原点操作之前，都需要进行自动回原点功能的初始化

rtn := GT_Reset();

rtn := GT_HomeInit()

// 自动回原点功能初始化操作在每次打开卡时只需要调用一次，最好不要频繁调用

3．Home 信号回原点使用方法

rtn := GT_AxisOn(axis);

rtn := GT_Home(axis,pos,vel,acc,offset);

4．Home+Index 信号回原点使用方法

rtn := GT_AxisOn(axis);

rtn := GT_Index(axis,pos,offset);

rtn := GT_Home(axis,pos,vel,acc,offset);

5．HomeStop 急停操作

rtn := GT_AxisOn(axis);

rtn := GT_HomeStop(axis,pos,vel,acc);

6．在指令执行过程中，可以随时调用 GT_HomeSts 来查询当前清零操作的执行结果。当状态值为 0 时表示当前正在运行状态，当状态值为 1 时表示操作完成。

3 例程

```
PROGRAM PLC_PRG
VAR
    First:BOOL:=TRUE;
    Rtn:INT;
    sts1,sts2:INT;
END_VAR
```

```
If first THEN
    rtn := GT_Reset();              (*复位运动控制器*)
    rtn := GT_ClrSts(1,8);          (*清除状态*)
    rtn := GT_HomeInit();           // 初始化自动回原点功能
    rtn := GT_AxisOn(1);            //使能轴 1
```

```
        rtn := GT_AxisOn(2);                    //使能轴 2
        rtn := GT_Index(1,20000,2000);          //轴 1 为 Home+Index 回零模式
        rtn := GT_Home(1,200000,50,0.5,2000);
        rtn := GT_Home(2,200000,50,0.5,3000);   //轴 2 为 Home 回零模式
        first:=FALSE;
END_IF
            rtn := GT_HomeSts(1,ADR(sts1));     //查询返回状态
            rtn := GT_HomeSts(2,ADR(sts2));
```

第十一章 指令列表

GUC-800-TPV 指令列表

系统初始化	
GT_Reset	复位运动控制器
GT_GetVersion	读取运动控制器固件的版本号
访问硬件资源	
GT_SetDo	设置数字 IO 输出电平状态
GT_SetDoBit	按位设置数字 IO 输出电平状态
GT_GetDo	读取数字 IO 输出电平状态
GT_GetDi	读取数字 IO 输入电平状态
GT_SetDac	设置 DAC 输出电压
GT_GetDac	读取 DAC 输出电压
GT_SetEncPos	修改编码器位置
GT_GetEncPos	读取编码器位置
GT_GetEncVel	读取编码器速度
硬件捕获	
GT_SetCaptureMode	设置编码器捕获模式
GT_GetCaptureMode	读取编码器捕获模式
GT_GetCaptureStatus	读取编码器捕获状态
GT_SetCaptureSense	设置捕获电平
运动逻辑管理	
GT_ClrSts	清除驱动报警、限位、IO 停止、跟随误差越限等异常标志
GT_AxisOn	打开驱动器使能
GT_AxisOff	关闭驱动器使能
GT_Stop	平滑停止或急停指定轴
GT_SetPrfPos	修改指定轴的规划位置
GT_SynchAxisPos	axis 合成规划位置和所关联的 profile 同步 axis 合成编码器位置和所关联的 encoder 同步
GT_SetSoftLimit	设置软限位
GT_GetSoftLimit	读取软限位
GT_SetAxisBand	设置到位误差带
GT_GetAxisBand	读取到位误差带
运动状态检测	
GT_GetSts	读取指定轴状态
GT_GetPrfPos	读取指定轴规划位置
GT_GetPrfVel	读取指定轴规划速度
GT_GetPrfAcc	读取指定轴规划加速度
GT_GetPrfMode	读取指定轴运动模式
GT_GetAxisPrfPos	读取 axis 合成规划位置
GT_GetAxisPrfVel	读取 axis 合成规划速度

第十章 指令列表

GT_GetAxisPrfAcc	读取 axis 合成规划加速度
GT_GetAxisEncPos	读取 axis 合成编码器位置
GT_GetAxisEncVel	读取 axis 合成编码器速度
GT_GetAxisEncAcc	读取 axis 合成编码器加速度
GT_GetAxisError	读取 axis 合成规划位置与合成编码器位置的误差
GT_GetClock	读取运动控制器系统时钟
设置控制参数	
GT_SetControlFilter	设定 PID 索引，支持 4 组 PID 参数
GT_GetControlFilter	读取当前 PID 索引
GT_SetPid	设置 PID 参数
GT_GetPid	读取 PID 参数
点位模式	
GT_PrfTrap	切换到点位运动模式
GT_SetTrapPrm	设置点位模式运动参数
GT_GetTrapPrm	读取点位模式运动参数
GT_SetPos	设置目标位置
GT_GetPos	读取目标位置
GT_SetVel	设置目标速度
GT_GetVel	读取目标速度
GT_Update	启动点位运动
Jog 模式	
GT_PrfJog	切换到 Jog 模式
GT_SetJogPrm	设置 Jog 模式运动参数
GT_GetJogPrm	读取 Jog 模式运动参数
GT_SetVel	设置目标速度
GT_GetVel	读取目标速度
GT_Update	启动 Jog 运动
PT 模式	
GT_PrfPt	切换到 PT 模式
GT_SetPtLoop	设置循环次数
GT_GetPtLoop	读取循环次数
GT_PtSpace	查询 PT 指定 FIFO 的剩余空间
GT_PtData	向 PT 指定 FIFO 传送数据
GT_PtClear	清除 PT 指定 FIFO 中的数据
GT_PtStart	启动 PT 运动
电子齿轮模式	
GT_PrfGear	切换到电子齿轮模式
GT_SetGearMaster	设置电子齿轮模式的跟随主轴
GT_GetGearMaster	读取电子齿轮模式的跟随主轴
GT_SetGearRatio	设置电子齿轮模式的传动比
GT_GetGearRatio	读取电子齿轮模式的传动比
GT_GearStart	启动电子齿轮运动
Follow 模式	

GT_PrfFollow	切换到 Follow 模式
GT_SetFollowMaster	设置 Follow 模式的跟随主轴
GT_GetFollowMaster	读取 Follow 模式的跟随主轴
GT_SetFollowLoop	设置 Follow 模式的循环次数
GT_GetFollowLoop	读取 Follow 模式的循环次数
GT_SetFollowEvent	设置 Follow 模式的启动条件
GT_GetFollowEvent	读取 Follow 模式的启动条件
GT_FollowSpace	查询 Follow 指定 FIFO 的剩余空间
GT_FollowData	向 Follow 指定 FIFO 传送数据
GT_FollowClear	清除 Follow 指定 FIFO 中的数据
GT_FollowStart	启动 Follow 运动
GT_FollowSwitch	切换 Follow 的工作 FIFO
运动程序	
GT_Download	下载运动程序到运动控制器
GT_GetFunId	读取运动程序中函数的标识
GT_Bind	绑定线程、函数、数据页
GT_RunThread	启动线程
GT_StopThread	停止正在运行的线程
GT_PauseThread	暂停正在运行的线程
GT_GetThreadSts	读取线程的状态
GT_GetVarId	读取运动程序中变量的标识
GT_SetVarValue	设置运动程序中变量的值
GT_GetVarValue	读取运动程序中变量的值

第十二章 加密机制

目前支持两类加密形式：

1） 软件加密：即应用程序开发人员在 OtoStudio 环境下开发应用程序过程中，设置密码保护程序。具体分为两种：

[1] 保护程序代码：在 OtoStudio->选项->密码：设置密码和保护密码。

[2] 保护程序运行：例如，在 OtoStudio 程序开发中，在程序中给出一个密码比较语句，作为是否运行程序的条件。

2） 硬件加密：固高可以提供两种硬件加密方式：

[1] 在运行 GRT.exe 时候，自动检查硬件版本号，如果版本号不正确，则不能正常启动。（版本号由固高提供，同一系列的固高控制器，版本号是相同的）

[2] 提供绑定网卡地址的函数 GetMacAddress()，应用程序开发人员可通过读取网卡地址来加密应用程序。（每套硬件平台都有唯一的网卡地址，且不可更改）

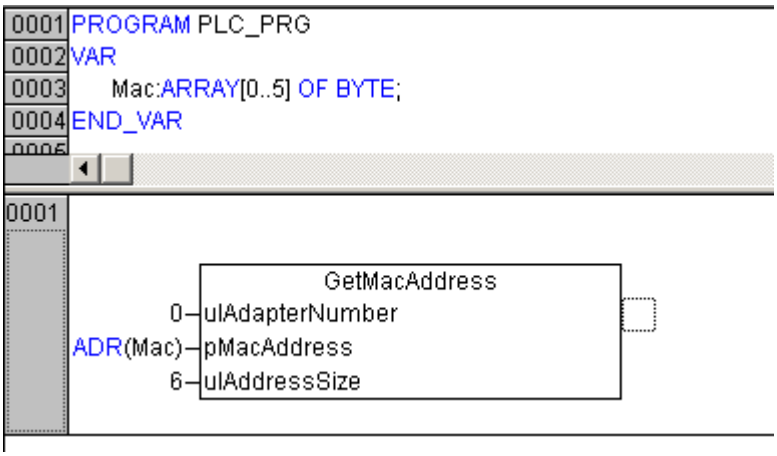
关于回款加密：固高可以提供参考方案如下

在应用程序中设置定时器，读取 CPU 的时钟，例如，希望客户在 3 个月内付款，否则应用程序无法工作。则可通过此种方式，先等待三个月的系统时钟，然后验证软件加密[2]。

GetMacAddress

GetMacAddress (ulAdapterNumber, pMacAddress , ulAddressSize)	
ulAdapterNumber: UINT	当前控制器网卡端口号，默认为 0
pMacAddress: POINTER TO BYTE	输出网卡地址，为数组首地址
ulAddressSize: UINT	网卡数组长度，单位为 Byte

范例程序如下：



CPAC-OtoStudio IO 扩展模块编程手册

第一章 概述

1.1 I/O 概述

针对自动化很多应用领域的需求，OtoStudio 软件平台上集成了种类丰富的数字量输入输出模块，模拟量输入输出模块，热电偶输入模块，继电器输出模块及其它 IO 模块，用户可以方便的配置，操作实际的模块以满足了各种应用行业的逻辑控制及各种信号的输入输出功能要求。

此功能为选配功能，请确认具有该功能的许可，否则造成的不便，我们不负有任何责任。

第二章 OtoStudio 中的 I/O 配置

目标系统平台：CPAC GUC-X00-TPX

当系统支持 I/O 模块功能扩展的情况下，用户可以在 PLC 配置中，右键 CPAC Platform，添加 I/O 扩展从站通讯模块 IM 153-1GL01 (CPAC-IOET300)，可以通过设置从站的节点 ID，对应于实际从站上的拨号；在从站模块上右键添加配套的数字量、模拟量 I/O 模块。对于一个从站 IM153 来说，最多可以挂接 16 个模块，且模块的种类没有限制。当挂接超过 16 个模块时候，系统会提示错误，不能正确工作。I/O 模块的配置如下图 2-1 所示。

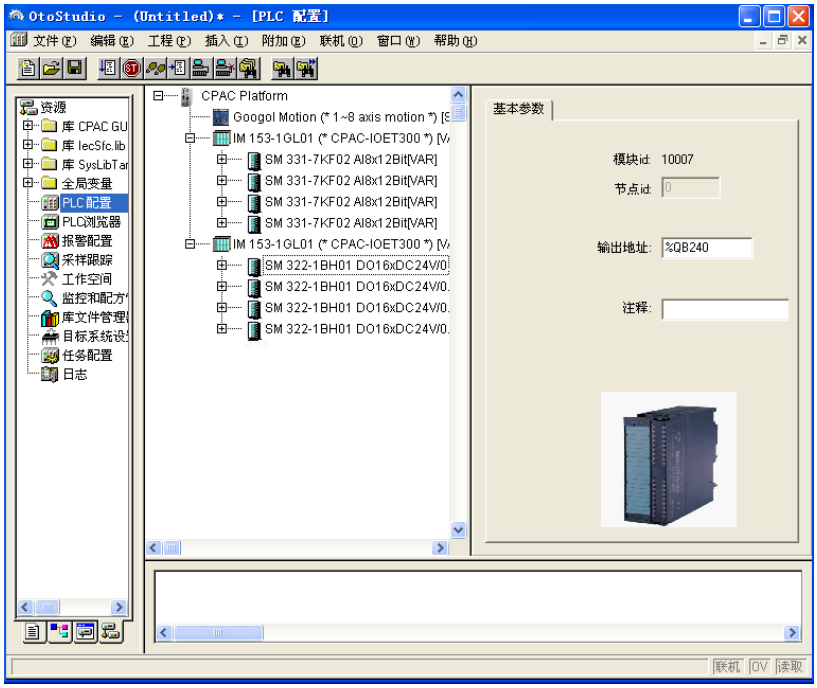


图 2-1

针对每个模块，双击会显示出对应的 I/O 资源点数和地址，输入用%I 表示，输出用%Q 表示。对于数字量模块有：8, 16, 32 通道 24V 数字量模块，对于模拟量模块有：4 通道模拟量输出，可支持电压输出+/-10V, 0~5V, 0~10V；可支持电流输出+/-20mA, 0~20mA, 4~20mA。8 通道模拟量输入，可支持电压输入+/-10V, 1~5V, +/-5V, +/-2.5V, +/-1.5V, +/-500Mv, +/-250mV, +/-80Mv；可支持电流输入+/-20mA, 0~20mA, 4~20mA, +/-10mA, +/-3.2mA；可支持电阻输入 0~150ohm, 0~300ohm, 0~600ohm。如图 2-2

第十章 指令列表

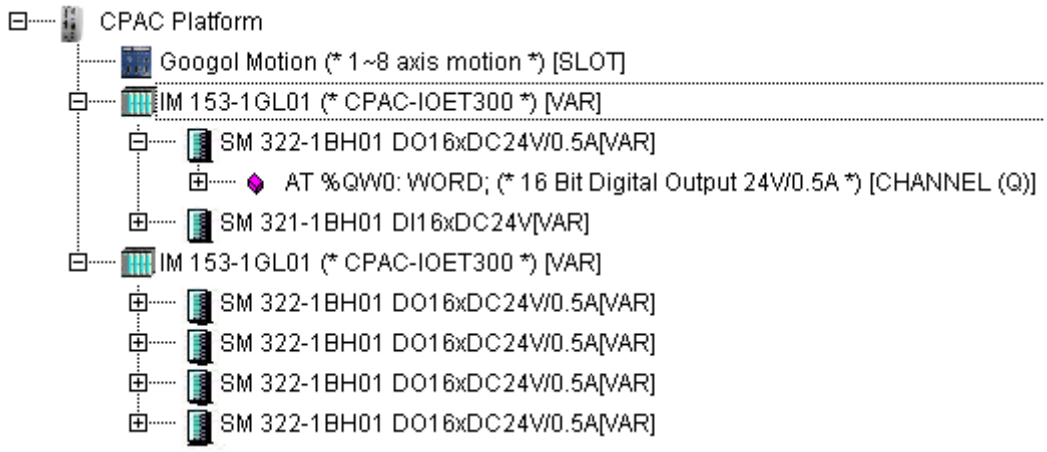


图 2-2

针对于每一个模块都提供了系统指定的地址，16 位数据用 W 表示，如图 2-3，DO16 对应的地址是%QW0，也可以按位寻址，例如%QX0.0 表示的是 D016 的第一位。

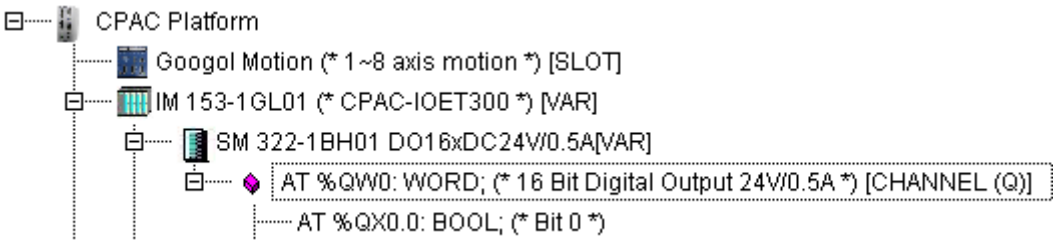


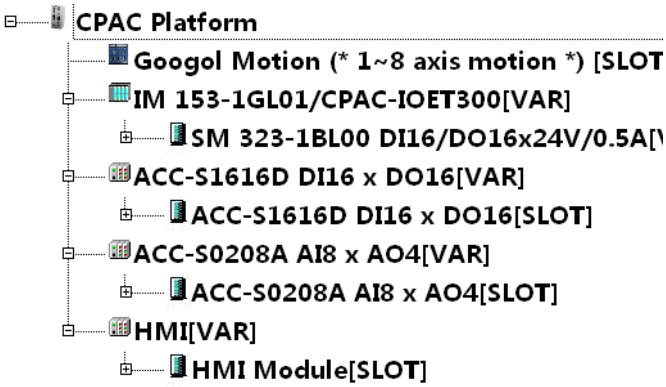
图 2-3

也可以添加 ACC 系列 IO 模块和 HMI 操作面板（不需要 IM153 通讯模块），主要有两种模块：

ACC-S1616D 16 DI x 16 DO： 16 入 16 出数字量模块

ACC-S0208A 4 AO x 8 AI：8 入 4 出 12 位模拟量模块

HMI：HMI 人机专用操作面板

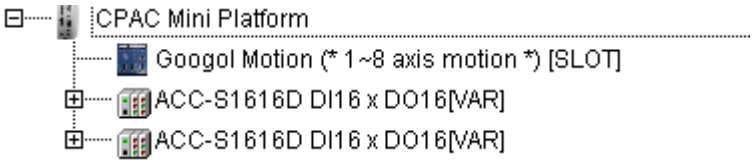


- 注意：必须先配置所有的 300 模块，然后再配置其他模块。（站号可以分别设置）

目标系统平台：CPAC GUC-X00-TPX Mini

当系统支持 I/O 模块功能扩展的情况下，用户可以在 PLC 配置中，右键 CPAC Mini Platform，只能添加 ACC 系列 I/O 扩展模块，主要有两种模块：

ACC-S1616D 16 DI x 16 DO： 16 入 16 出数字量模块
ACC-S0208A 4 AO x 8 AI：8 入 4 出 12 位模拟量模块



第三章 如何对 I/O 编程

3.1 PLC 编程

当完成 PLC 配置后，配置出对应的硬件扩展模块，接着就要对这些扩展模块进行编程，即可编程。在编程中，扩展的模块地址通过专用的定义表示，有两种方法：
[1] 可以在变量声明里，将地址信息映射，例如

```
(*将 counter_heat7 写到第 0 个字的第 0 位*)
counter_heat7 AT %QX0.0: BOOL;

(*将第 7 个字的第 2 位赋值给 lightcabinetimpulse *)
lightcabinetimpulse AT %IX7.2: BOOL;
```

[2] 直接表示。例如

```
  %QW0:=NOT SHL(1,i);
  %QW1:=NOT SHL(1,i);
  i:=i+1;
  IF i>16 THEN
    i:=0;
  END_IF
```

3.2 梯形图编写的例程

例程如下：应用梯形图完成 I/O 的操作，输出模块的第一个位在定时翻转。电机运动的速度切换通过数字量输入模块的第一位和第二位控制。程序如图 3-1

第十章 指令列表

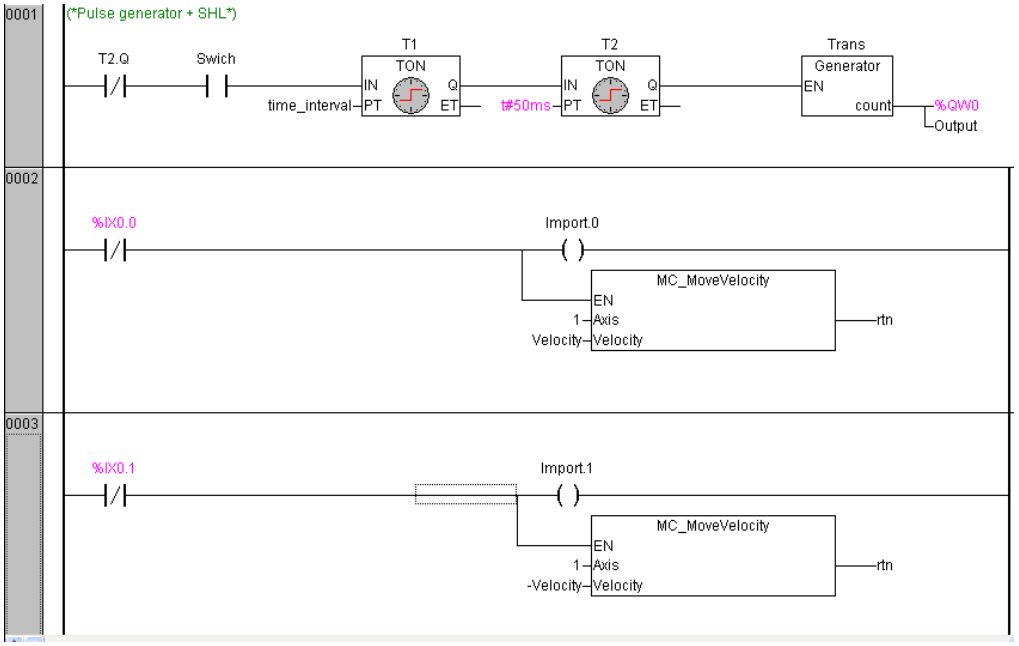


图 3-1

该程序的配置请参考图 2.2；应用扩展模块的 I/O 点数可以实现复杂的逻辑功能，尤其是应用梯形图的编程语言，更加方便。

第四章 程序下载

当编译完程序，并且编译无错误无警告时，点击登陆（login）下载程序并完成实际的配置。

当正确配置完成，从站模块的 LK 会点亮，且在界面上会提示：PLC Station * configuration is completed. 否则 SF 会点亮（报警）且界面上会提示：PLC Station * configuration failed, config error is [a]。a 代表错误信息的代码，从-1到31。其实际意义见下面列表

-1	通讯不正常
0 （缺省）	成功
1	模块不支持
2	传入的指针为空
3	从站不存在报警
4	输入输出刷新长度大于240
7	配置信息不正确,参数不正确
10	连接超时
11	从站报警：其他错误
12	从站报警：通讯错误
13	从站报警：配置信息不匹配
14	扫描从站：从站数不合逻辑
15	从站报警：从站中断
16	从站报警：从站处于停机状态
31	从站无应答
32	总线校验错误

错误类型及可能的原因：

第十章 指令列表

错误号	可能原因	解决方法
0	运行良好	---
10 12 17 18 19	1、 固件不正确或不匹配 2、 外部原因造成通讯不正常，如干扰等。	尝试多次，如果问题频繁出现，请联系我们
13	配置的模块与实际总线上挂接的模块不匹配	1、 仔细检查 OtoStudio 上的 PLC 配置情况和实际的模块挂接状况，确定模块匹配。 2、 也可以通过 OtoStudio 的 PLC-Browser 的 Scan 函数，自动读取实际的模块挂接情况，然后与配置比对。
15	从站上的某些模块出现断线、溢出、供电断开等	请通过诊断库读取诊断信息，找出产生中断的模块及产生中断的原因（请参考后面部分对中断信息的分析部分） 根据实际情况采取相应的措施。
16	从站处于停机状态	如果将从站配置成为‘station stop after interrupt’,则当有中断产生时，从站将进入停机状态，此时，从站只会响应用户的读取中断信息的指令。 在这种情况下，只有将从站断电重启后重新配置，才能使得从站重新开始工作。
20 31 32	1、 从站通讯失败，可能连接电缆断开、从站号不对应、从站没有上电等。 2、 通讯过程中数据校验失败，可能通讯受到干扰。	1、 检查从站是否上电 2、 检查 OtoStudio 上配置的从站的节点号是否跟从站的编码开关对应 3、 检查通讯电缆是否连接正确和牢固。

说明：对于 OtoStudio 的配置下载过程中遇到的错误代码，同样可以参考上表。

当配置完成后，点击运行程序，从站模块上的 RD 指示灯开始闪烁，代表程序开始刷新硬件通道

第五章 I/O 功能库的使用

在具有 I/O 扩展功能的 OtoStudio 中，提供了针对逻辑控制的库。可以在 POU 中调用。

5.1 数模转化库 IO_EXTENTION 函数说明

AnologData_To_DigitalData	
ValEU : LREAL	模拟量数值 （输入）
MinEU : LREAL	模拟量量程最小值 （输入）
MaxEU : LREAL	模拟量量程最大值 （输入）
Raw : WORD	数字量 （输出）
DigitalData_To_AnologData	
Raw : WORD	数字量 （输入）
MinEU : LREAL	模拟量量程最小值 （输入）
MaxEU : LREAL	模拟量量程最大值 （输入）
ValEU : LREAL	模拟量数值 （输出）

例如：当对模拟量输出模块进行操作时候，添加一个模拟量输出4通道模块。且配置为电压输出模式，输出量程为-10V..10V。配置如图4-1所示

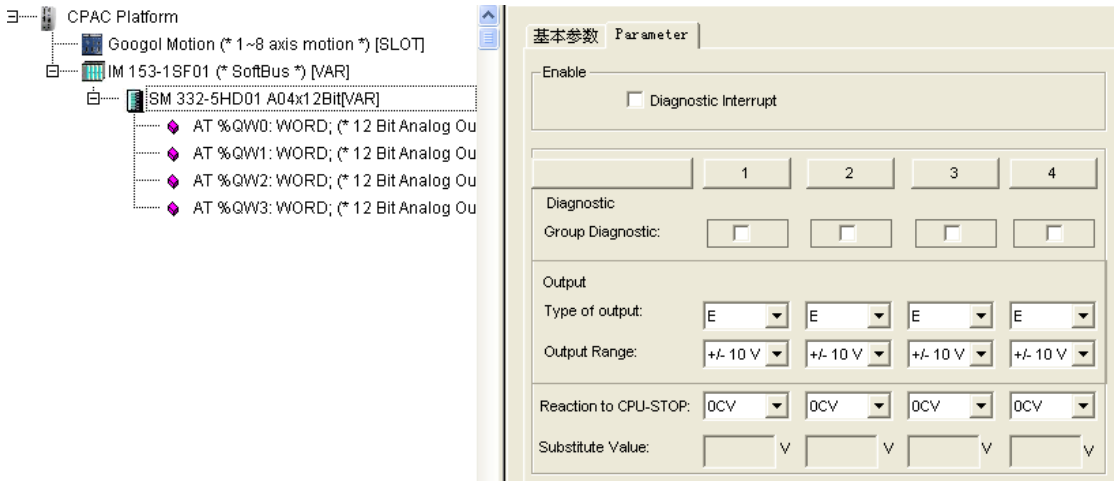


图4-1

则用户若想对该模块的第一个通道输出电压 nV，首先添加对应的库，如图4-2所示

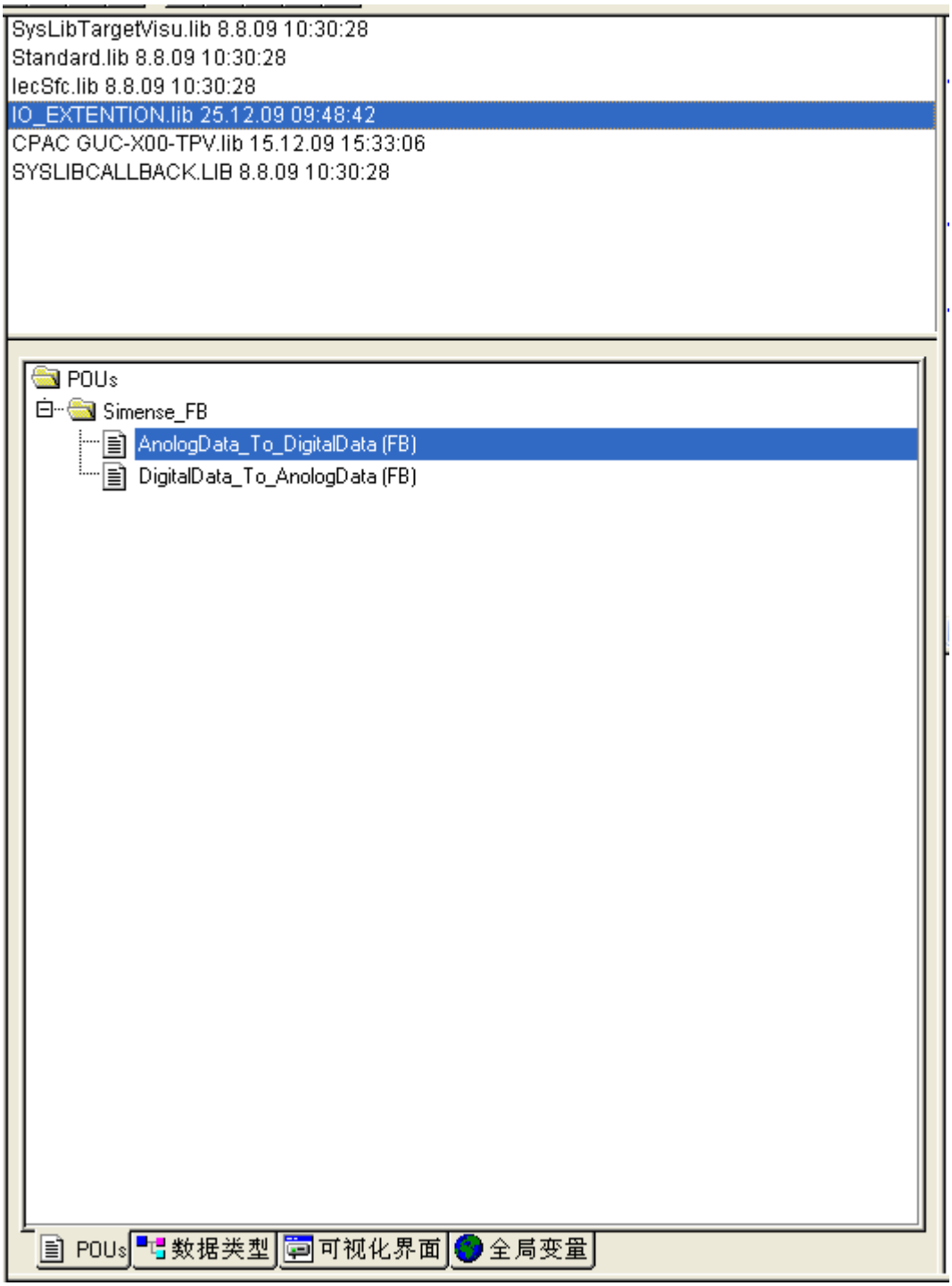


图4-2

需调用功能块 AnologData_To_DigitalData 将模拟量转换成数字量再进行输出。相关的例程如下图所示4-3所示

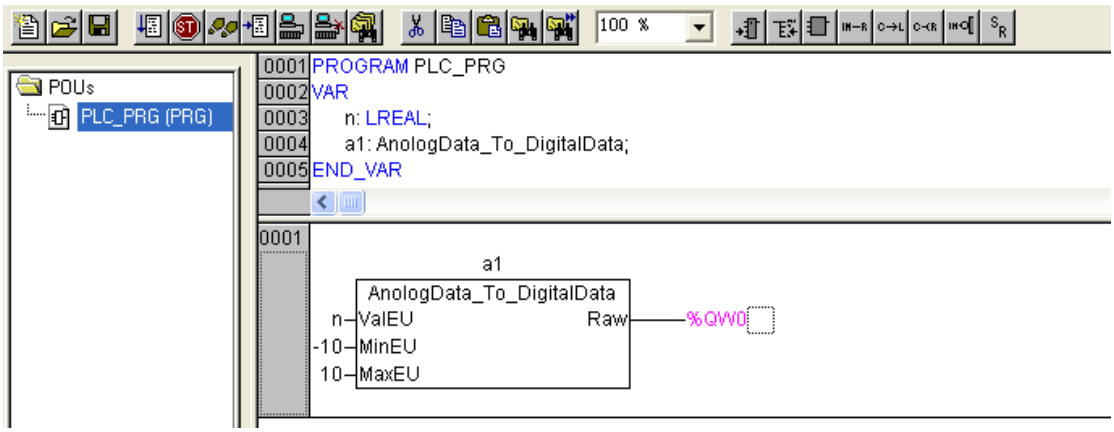


图4-3

注意	**I/O 功能库只能在具有 I/O 扩展功能的前提下应用，与 I/O 扩展功能一并提供**

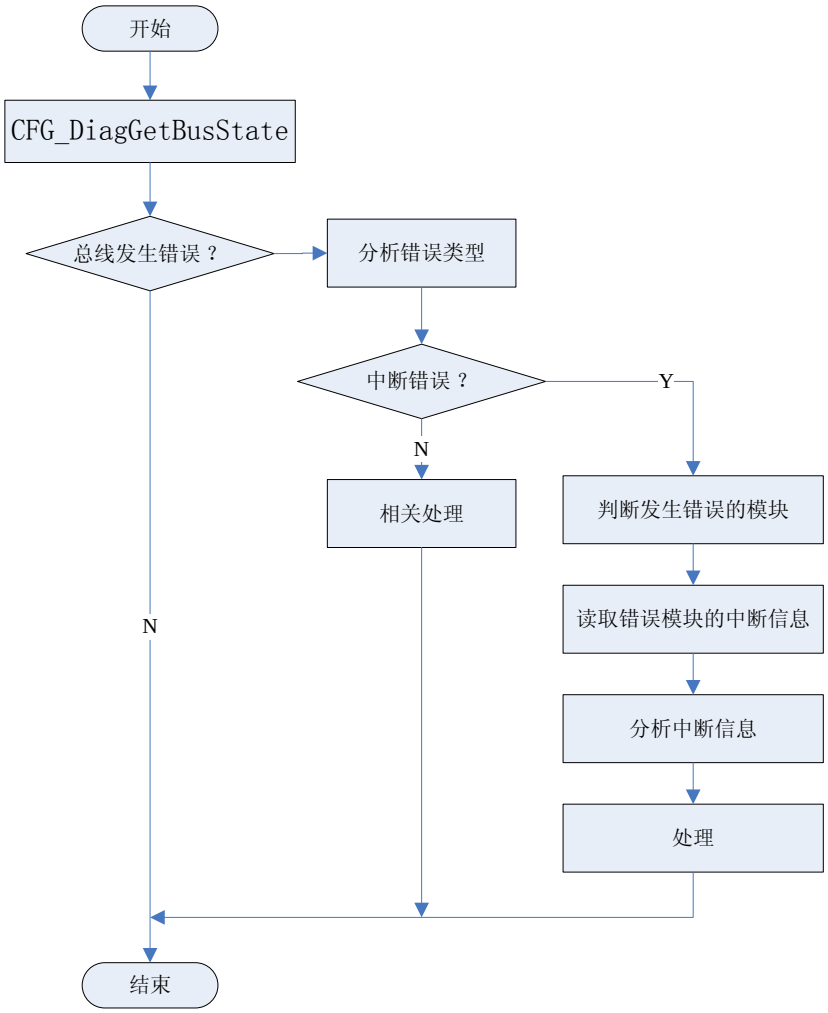
5.2 诊断函数库 PLCDiag 库说明

IO_SlaveReset	所有从站输出复位为 0。说明：该指令只复位被配置的从站
无参数	
IO_DiagGetSlaveInterrupt	读取从站中断的详细信息
station : INT	从站号，取值范围[0,15];
module : INT	模块号，取值范围[0,15];
interruptInfo : POINTER TO ARRAY[0..15] OF BYTE	中断信息。 中断信息代表的含义，请参见相关说明。
IO_DiagGetBusState	获取总线及从站的状态
busState : POINTER TO INT	总线状态，0--正常，1--不正常。
slaveState : POINTER TO ARRAY[0..15] OF DINT	返回的各从站的状态
errModuleFlag : POINTER TO ARRAY[0..15] OF DINT	如果 slaveState[i]为中断错误，则按位指示发生错误的模块，否则为 0； 注意：这里只会判断已经配置成功的从站
IO_DiagSlaveStateHandler	处理从站报警
station : INT	从站号，取值范围[0,15];
handlerId : INT	处理方法， 0 //无须理会 1 //关闭当前从站 2 //关闭所有从站 3 //清除报警

	注意：关闭从站后需要重新配置才能使得从站工作正常

相关宏定义：

诊断处理流程：



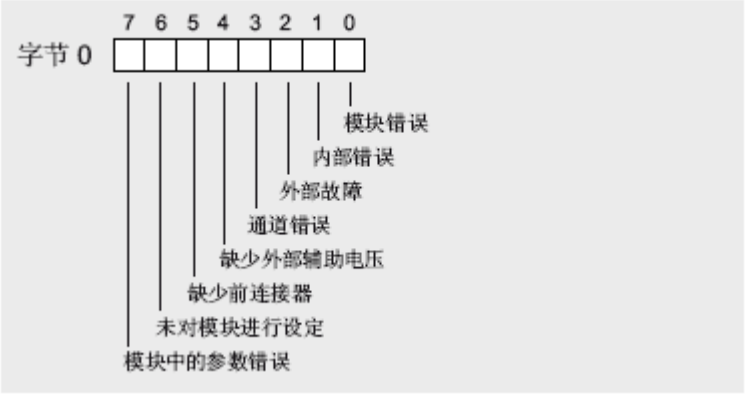
例程：

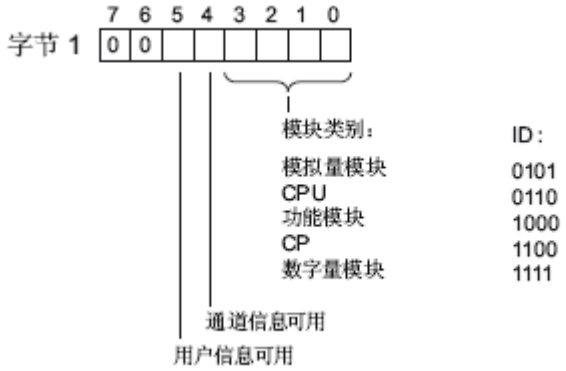
```
PROGRAM PLC_PRG
rtn : INT;
busState : INT;
slaveState,moduleFlag : ARRAY[0..15] OF INT;
interruptInfo : ARRAY[0..15] OF BYTE;
i,j : INT;

//获取总线状态
rtn :=
CFG_DiagGetBusState(ADR(busState),ADR(slaveState[0]),ADR(moduleFlag[0]));
if(busState) THEN                                    //总线上有错误发生
    for i:=0 TO 15 BY 1 DO
        if(15 = slaveState[i]) THEN                  //错误类型为：中断错误
```

```
            for j:=0 TO 15 BY 1 DO
                if (moduleFlag[i] AND (SHL(1,j)) THEN//是否第j个模块发生错误
                    //读取第i个从站的第j个模块的中断信息
                 rtn := CFG_DiagGetSlaveInterrupt(i,j,ADR(interruptInfo[0]));
                    //处理第i个从站的状态
                    rtn := CFG_DiagSlaveStateHandler(i,1);
                END_IF
            END_FOR
        END_IF
    END_FOR
END_IF
```

5.3 中断信息

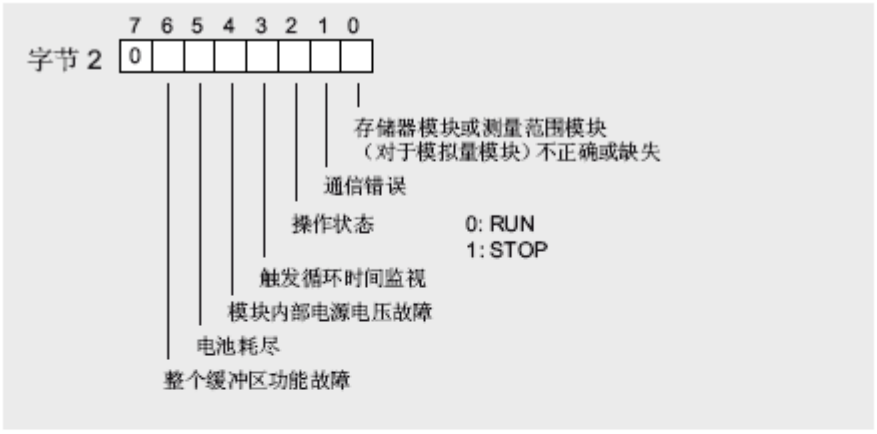
模块的诊断数据存储在数据记录中，数据记录长度最长为16个字节。
下面对各个字节进行描述（相应位中，逻辑‘1’代表错误）：
字节 0 和字节 1：

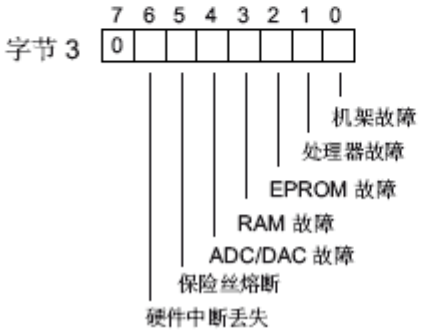




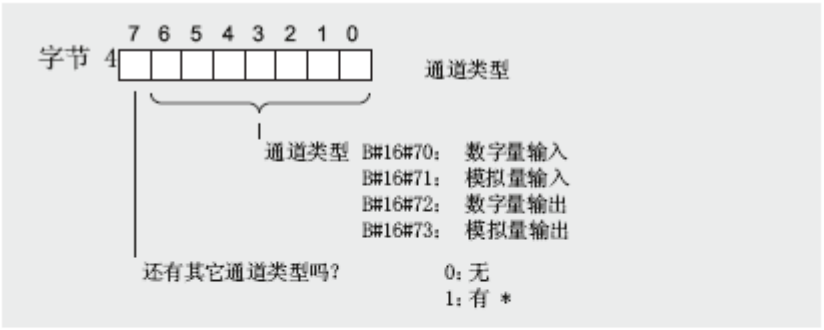
字节 2 和字节 3：

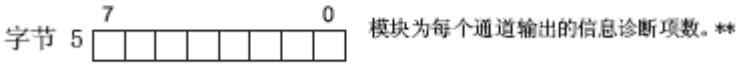
第十章 指令列表

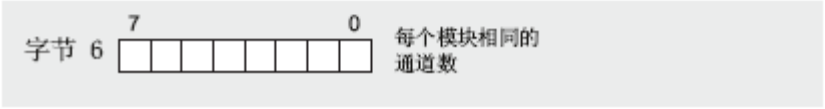




字节 4 到字节 6，包含有关通道类型、诊断信息长度和通道数的信息。

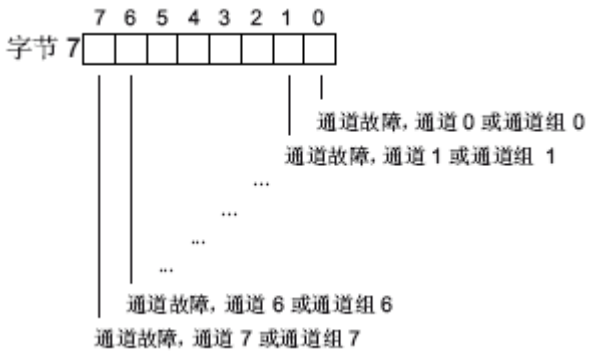






*当存在其他通道类型（通道类型的位 7 为 1）时，下一个通道类型跟在数据记录 1 之后，其实的数据类型是紧跟在上一个通道类型的通道特定诊断数据之后的数据类型。
**每个通道用于通道特定诊断数据的字节数，对应于此处指定的位数。

字节 7 通道故障载体和更高字节



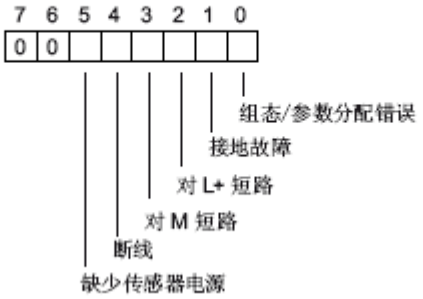
通道故障载体的长度最短为 1 个字节。对于 8 个以上通道的模块，通道故障载体占用多个字节。

通道特定的诊断数据跟在通道故障载体后面，各个通道用于通道特定诊断数据的字节数取决于自己 5‘诊断信息长度’中输入的位数。

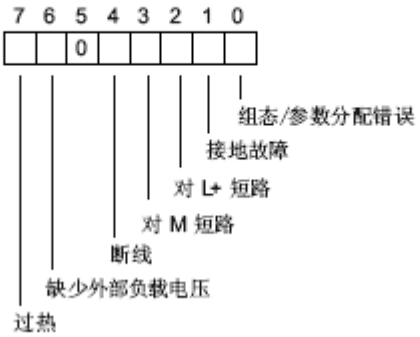
下图显示了模块特定的通道或通道组的诊断字节的分配。

在‘模块诊断’部分介绍了可能的出错原因以及相应的解决方法。

SM 321；DI 16 x DC 24 V 的数字量输入通道；带有过程和诊断中断

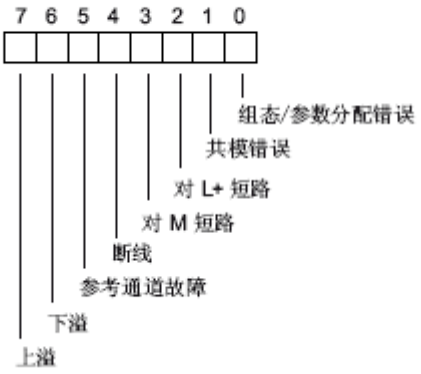


SM 322；DO 8 x DC 24 V/0.5 A 的数字量输出通道，带诊断中断

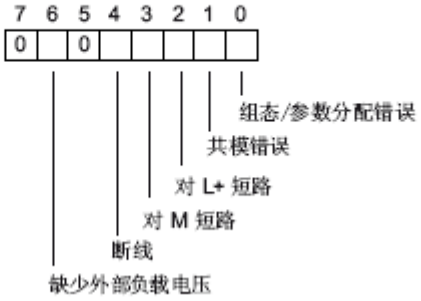


第十章 指令列表

带诊断功能的 SM 331 模块的模拟量输入通道



带诊断功能的模拟量输出通道



例如：

从站的第一个模块为 331-7KF01（AI8*12bit），当从站由于中断报警后，读取中断信息为（16 进制表示）：

0D 15 00 00 71 08 08 03 10 10 00 00 00 00 00 00

可以对这些数据进行分析，以确定产生中断的原因。

0D---代表从站存在模块错误、外部故障、通道错误。

15---该模块为模拟量模块，通道信息可用

00 00---说明没有字节 2 和字节 3 中对应的错误

71—没有其他通道，发生故障的通道类型为‘模拟量输入’通道。

08---特定的通道诊断数据长度为 1 个字节

08---相同的通道数为 8

03---故障通道为第一、二通道

10---断线（带诊断功能的模拟量输入通道）

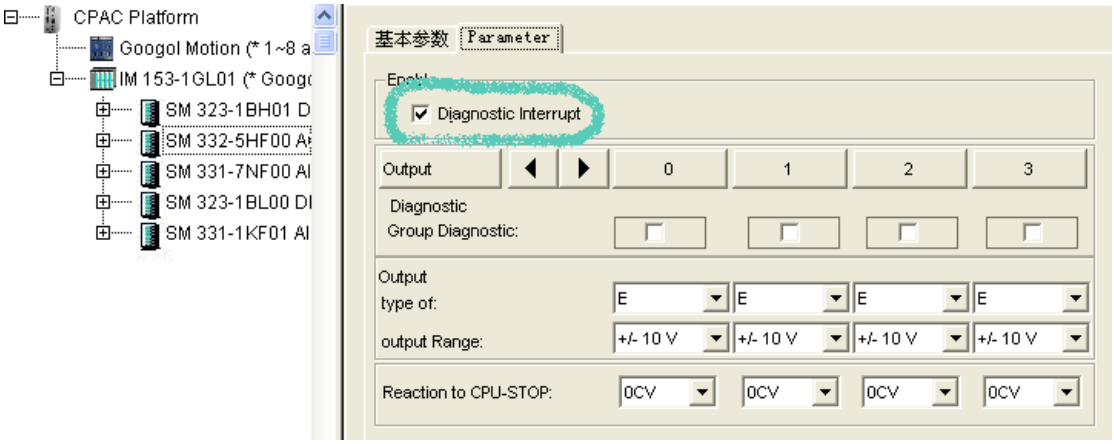
00 00 … ---无意义

有以上分析可以知道，该中断产生的原因是 331-7KF01 模块的第一二通道断线引起的。

第六章 中断功能

6.1 模块中断配置

OtoStudio 支持模块的故障中断诊断功能。当在 OtoStudio->PLC 配置中，支持中断的模块配置了中断诊断的功能，如下图所示：



注意	**只有支持中断诊断的模块才可以配置该参数** **目前包括：332-5HF00, 331-7NF00, 331-1KF01**

可诊断的故障类型包括：模块电源断开，外设接线错误，量程超限等。具体请参看故障诊断函数 IO_DiagGetBusState 说明。

当配置完成该参数后，需要选择是否进行 Group Diagnostic，当选择对应通道后，就可以诊断每一个通道的中断信息，否则只能诊断到模块的中断信息。具体请参看故障诊断函数 IO_DiagGetSlaveInterrupt 说明。

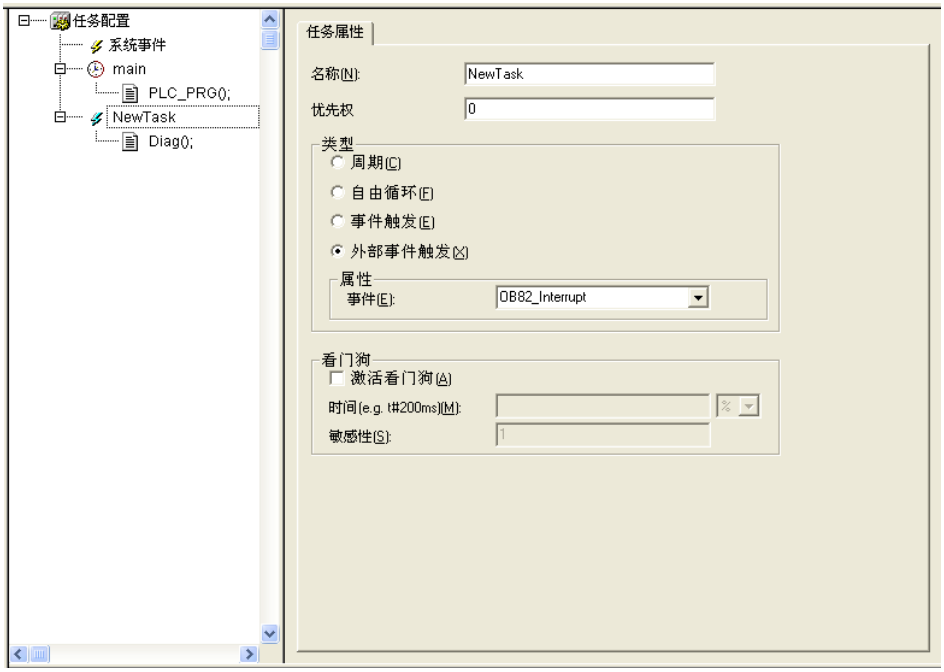
6.2 从站中断服务配置

当配置诊断中断完成后，在任务配置中如果默认不配置中断任务，则一旦中断产生，系统会自动停机。

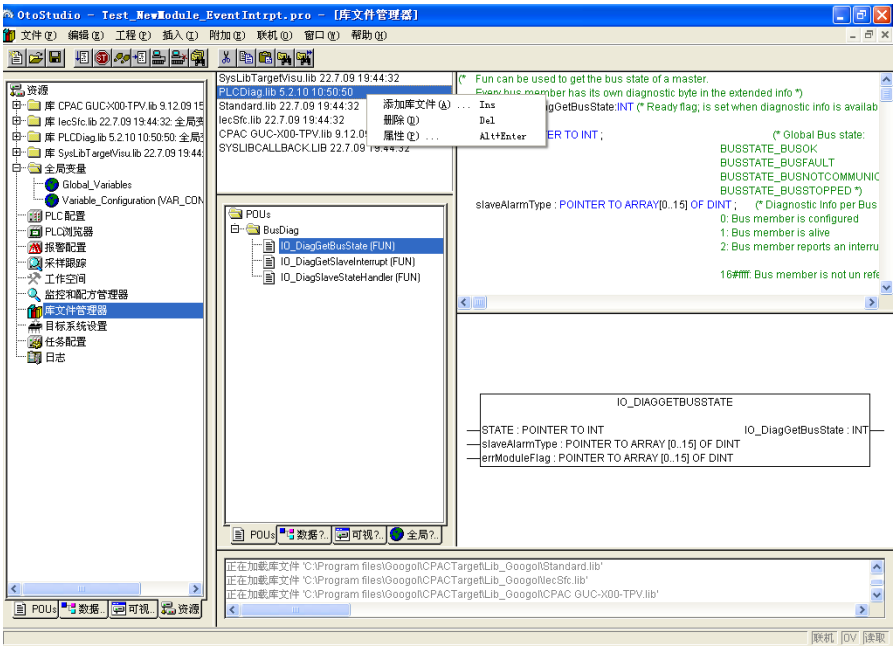
6.3 主机响应中断

当从站中断配置中，也提供了用户响应中断程序的功能，具体的操作步骤如下：
1：在任务配置中配置中断任务，添加一个任务，将任务类型选择为“外部事件触发”，在属性中，选择事件为 OB82_Interrupt，等同与 Step7 中的 OB82，当中断产生时，系统会检测到配置了 OB82 任务，则不停机，且会触发该任务，并只执行一次。也就是说：在任务下挂接

的 IEC 程序，只执行一次。如图所示。



在程序中可以调用中断诊断功能库的函数，也获取错误中断信息，以及错误类型。该库的添加见下图所示：



注：如果产生中断的错误是不可恢复的，例如：电压超限保护，则系统无论是否配置 OB82，均会自动停机。

6.4 主机响应中断程序例程

该程序中实现了当中断产生时候，获取中断信息，并判断是否处理中断的功能。

```
0001 PROGRAM Diag
0002 VAR
0003     State:INT;
0004     slaveAlarmType: ARRAY[0..15] OF WORD;
0005     errModuleFlag: ARRAY[0..15] OF WORD;
0006     SlaveId: INT;
0007     moduleId: INT;
0008     InterruptINFO: ARRAY[0..15] OF WORD;
0009     OK: BOOL;
0010     HandlerId: INT;
0011     i:INT;
0012     eiurieur: INT;
0013 END_VAR
```

```
0001 i:=i+10;
0002     IO_DiagGetSlaveInterrupt(SlaveId, moduleId, ADR(InterruptINFO));
0003 IF OK THEN
0004     IO_DiagSlaveStateHandler(SlaveId, HandlerId);
0005     eiurieur:=eiurieur+1;
0006 END_IF
0007
0008
0009
0010
0011
0012
0013
0014
0015
```

注意	**用户如果要在该程序中添加对 IO 的操作，只能对其他从站 IO 操作**

注意	**当应用程序中无 IO 操作时，且在目标系统配置中，没有选中“更新不使用的 IO”，则中断服务程序不会激活。**

第七章 I/O 模块说明

OtoStudio 软件中支持种类丰富的数字量输入输出模块，模拟量输入输出模块，热电偶输入模块，继电器输出模块及其它 IO 模块满足了各种应用行业的逻辑控制及各种信号的输入输出功能要求。

固高 PLC 从站模块产品列表

模拟量模块列表	
CPAC-300-331-7KF01	8 路*12 位模拟量输入模块
CPAC-300-331-7KF02	8 路*12 位模拟量输入模块

第十章 指令列表

CPAC-300-331-1KF01	8 路*13 位模拟量输入模块
CPAC-300-331-7NF00	8 路*16 位模拟量输入模块
CPAC-300-332-5HD01	4 路*12 位模拟量输出模块
CPAC-300-332-5HF00	8 路*12 位模拟量输出模块
数字量模块列表	
CPAC-300-321-1BH01	16 路数字量输入模块
CPAC-300-321-1BH02	16 路数字量输入模块
CPAC-300-321-1BH50	16 路数字量输入模块
CPAC-300-321-1BL00	32 路数字量输入模块
CPAC-300-322-1BF01	8 路数字量输出模块
CPAC-300-322-1BH01	16 路数字量输出模块
CPAC-300-322-1BL00	32 路数字量输出模块
CPAC-300-322-1HF01	8 路继电器输出，浮地
CPAC-300-322-1HH01	16 路继电器输出，浮地
CPAC-300-322-1HH50	16 路继电器输出，干接地
CPAC-300-323-1BH01	8 Di 8 Do 数字量输入输出模块
CPAC-300-323-1BL00	16 Di 16 Do 数字量输入输出模块

第十章 指令列表

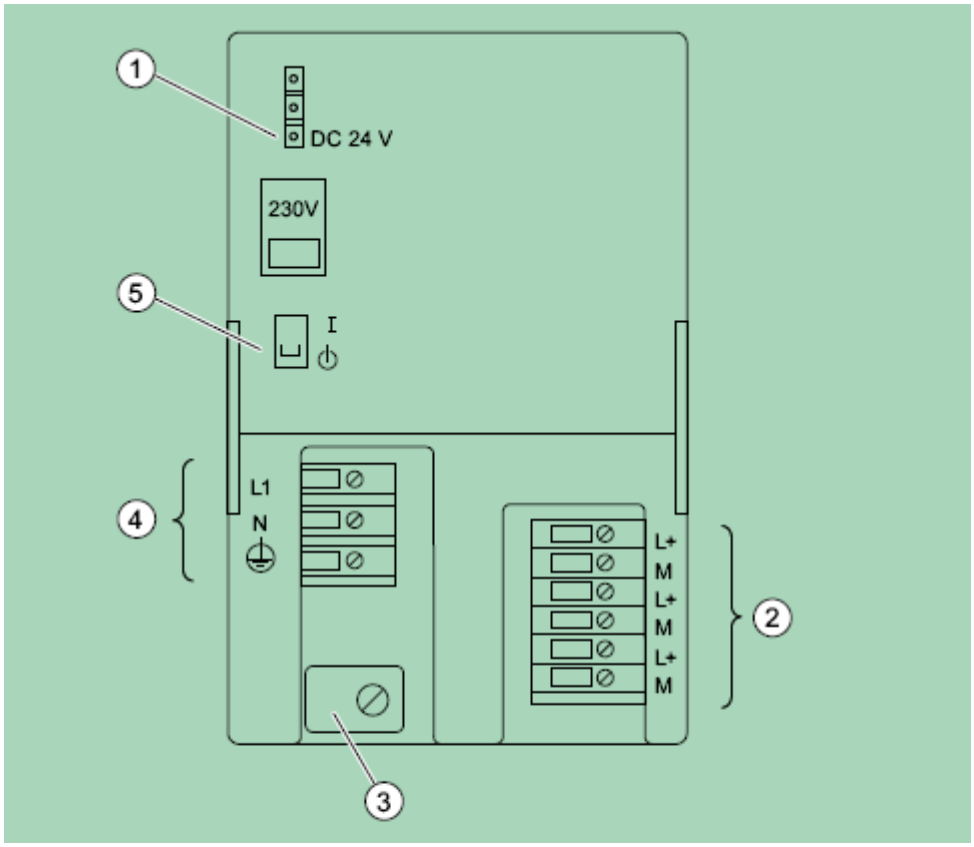
电源模块

属性及技术数据

电源模块主要的属性如下：

- 输出电流为5 A
- 输出电压为24 VDC；短路和断路保护
- 与单相交流电源连接
  (额定输入电压120/230 VAC，50/60 Hz)
- 安全隔离符合EN 60 950
- 可用作负载电源

外形及接线



图中：

1---‘24 VDC 输出电压工作’显示

2---24 VDC 输出电压接线端

3---固定装置

第十章 指令列表

4---主回路和保护性导体接线端
5---24 VDC 开关
6---电源选择器开关

第十章 指令列表

通讯转接模块：CPAC-IOET300

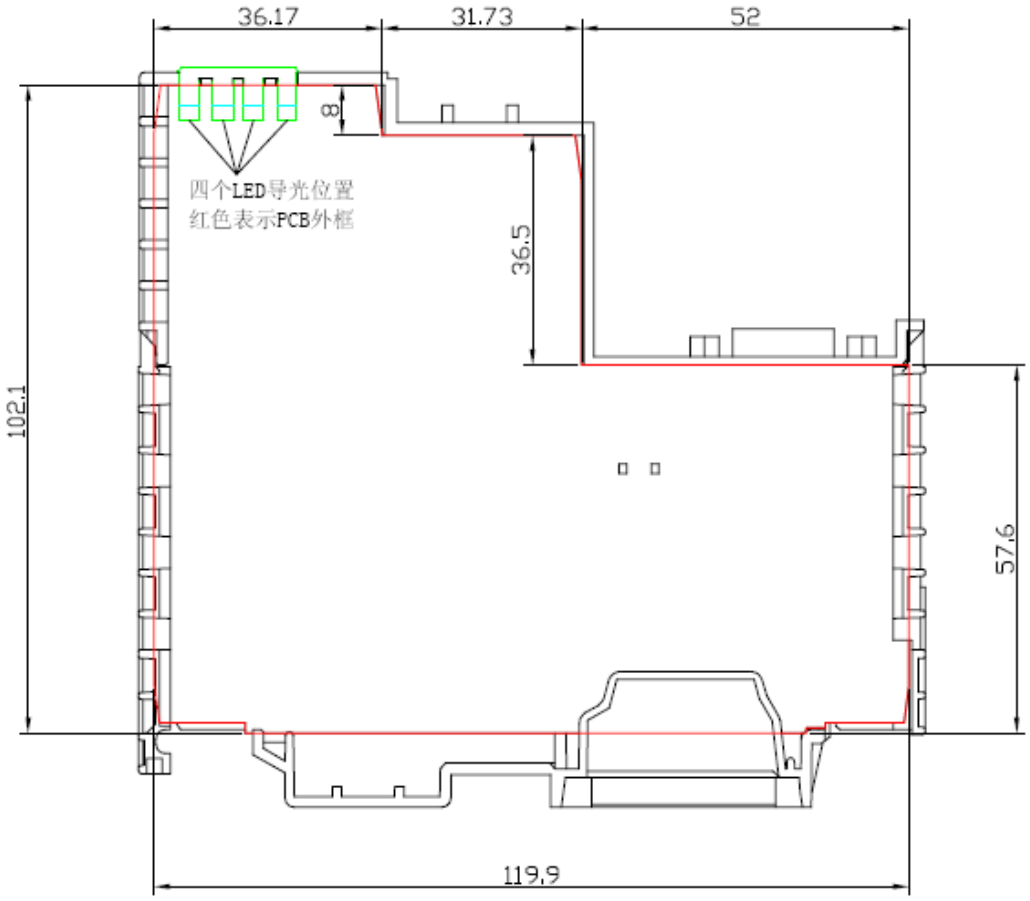
属性及技术数据

CPAC-IOET300是固高的通讯转接模块，其主要的属性如下：

- 输出电压：4.75V~5.25V
- 输出电流：最大3A
- 信号电平：5V TTL
- RS-485绝缘电压：1000V
- 输入电压：12V~24V
- 供电功率：15W

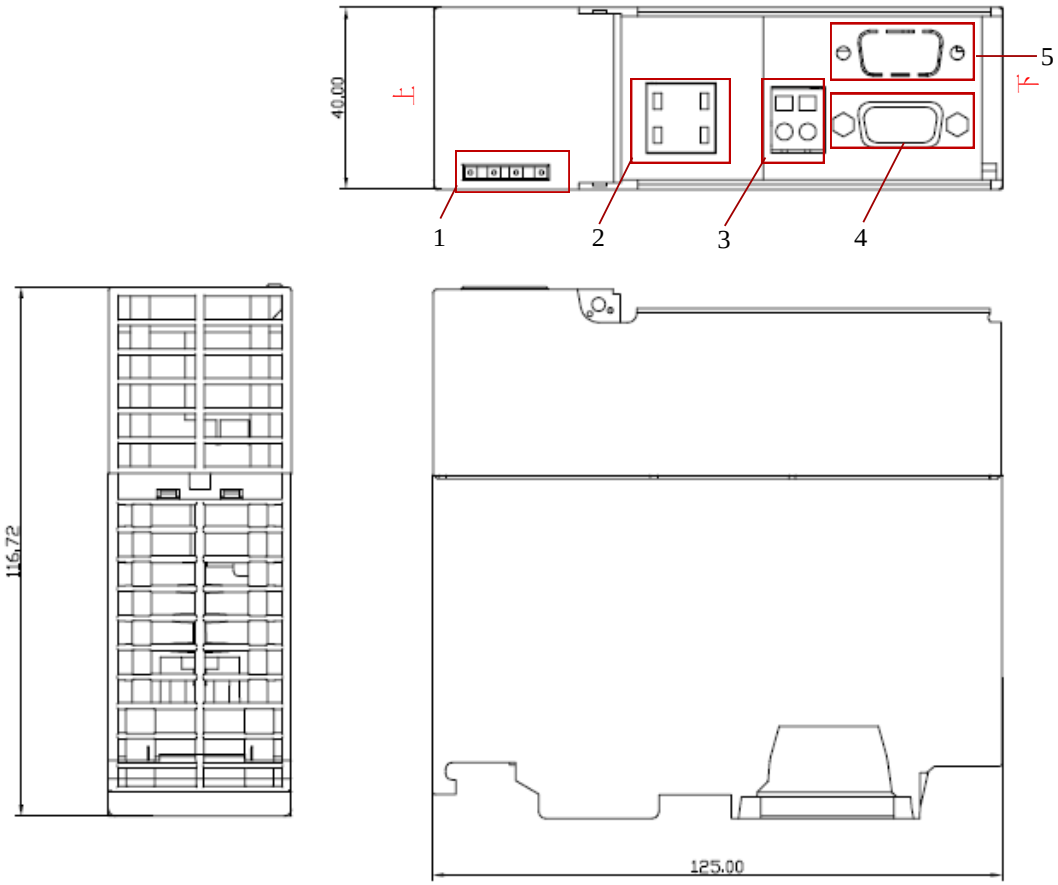
外形及接线

CPAC-IOET300 的外形尺寸如下图所示：



其外形及接口如下图所示：

第十章 指令列表



1---指示灯
2---从站地址拨码
3---电源输入
4---通讯连接端口 1
5---通讯连接端口 2
指示灯定义见下表（编号从上至下）：

序号	说明	标识	颜色	显示方式
LED1	电源指示	PWR	橙	上电后常亮
LED2	故障指示	SF	红	平时灭，故障时常亮
LED3	运行指示	RD	黄绿	平时灭，IO 刷新后闪烁
LED4	连接指示	LK	黄绿	平时灭，配制后常亮

电源接口定义如下：

引脚	信号名称	信号说明
1	VIN	12V~24V 电源输入
2	GND	电源地

通讯连接的两个端口完全等效（便于级联），其端口定义如下表：

引脚	信号名称	信号说明	默认电平
1			
2	RX+	主站发送，从站接收信号/正相	
3	TX+	从站发送，主站接收信号/正相	

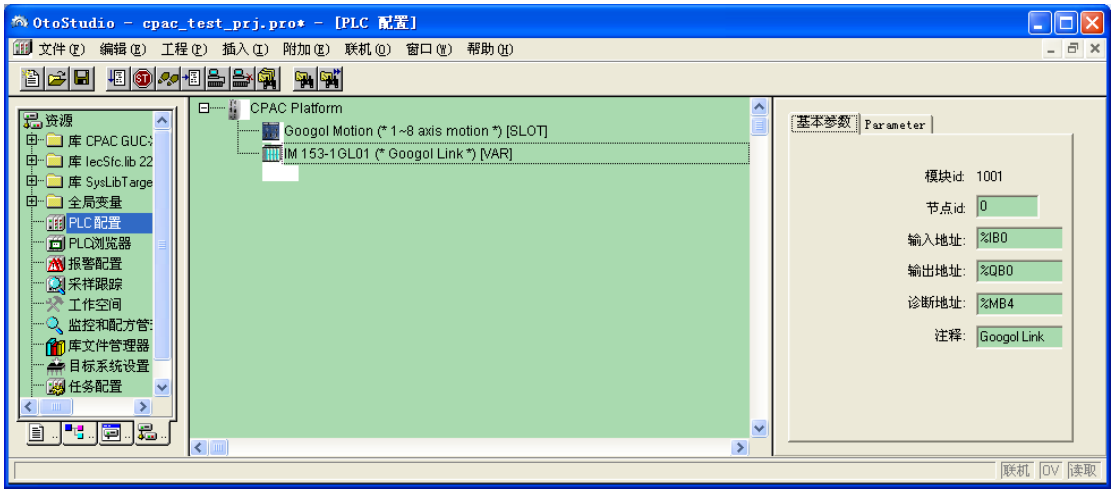
4			
5			
6			
7	RX-	主站发送，从站接收信号/负相	
8	TX-	从站发送，主站接收信号/负相	
9			

配置及编程

添加模块

转接从站是从站转接模块，所有的 IO 模块只能挂接在它的下面，最多支持 16 个从站，每个从站下面最多挂接 16 个 IO 模块。

进入 OtoStudio 的 PLC 配置，在 CPAC Platform 上点击右键，在弹出的菜单中选择‘添加 IM 153-1GL01/CPAC-IOET300’添加模块，如下图所示。添加转接模块后，用户就可以在从站下挂接不同的 IO 模块。

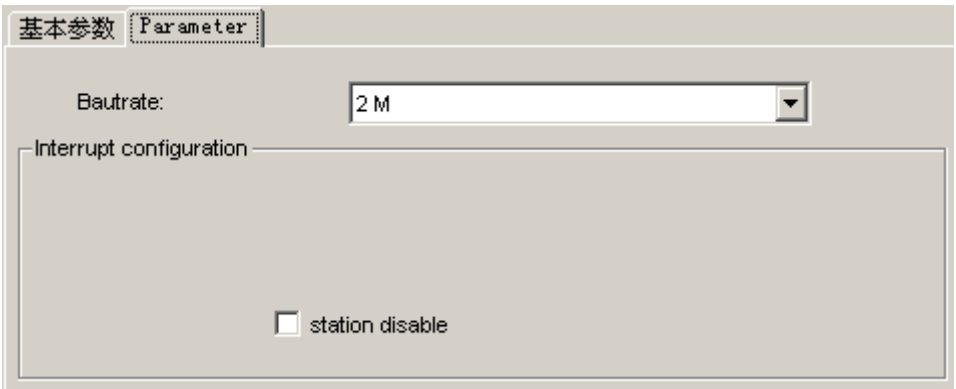


模块配置

基本参数包括输入输出地址的设置，该地址决定该从站下挂接模块的输入输出的起始地址。

节点 id 是基本参数中特别需要注意的一个参数，该参数必须与从站的地址编码开关对应。

CPAC-IOET300 模块的扩展参数设置界面如下图所示：



第十章 指令列表

Baudrate: Googollink 总线通讯波特率，当站点较多时，建议设置为 500K。默认为 2M
Station disable：如果该单选框被激活，则 OtoStudio 在下载配置时，将忽略该从站。

第十章 指令列表

模拟量模块

模拟量模块列表	
CPAC-300-331-7KF01	8 路*12 位模拟量输入模块
CPAC-300-331-7KF02	8 路*12 位模拟量输入模块
CPAC-300-331-1KF01	8 路*13 位模拟量输入模块
CPAC-300-331-7NF00	8 路*16 位模拟量输入模块
CPAC-300-332-5HD01	4 路*12 位模拟量输出模块
CPAC-300-332-5HF00	8 路*12 位模拟量输出模块

模拟量输出：CPAC-300-332-5HD01

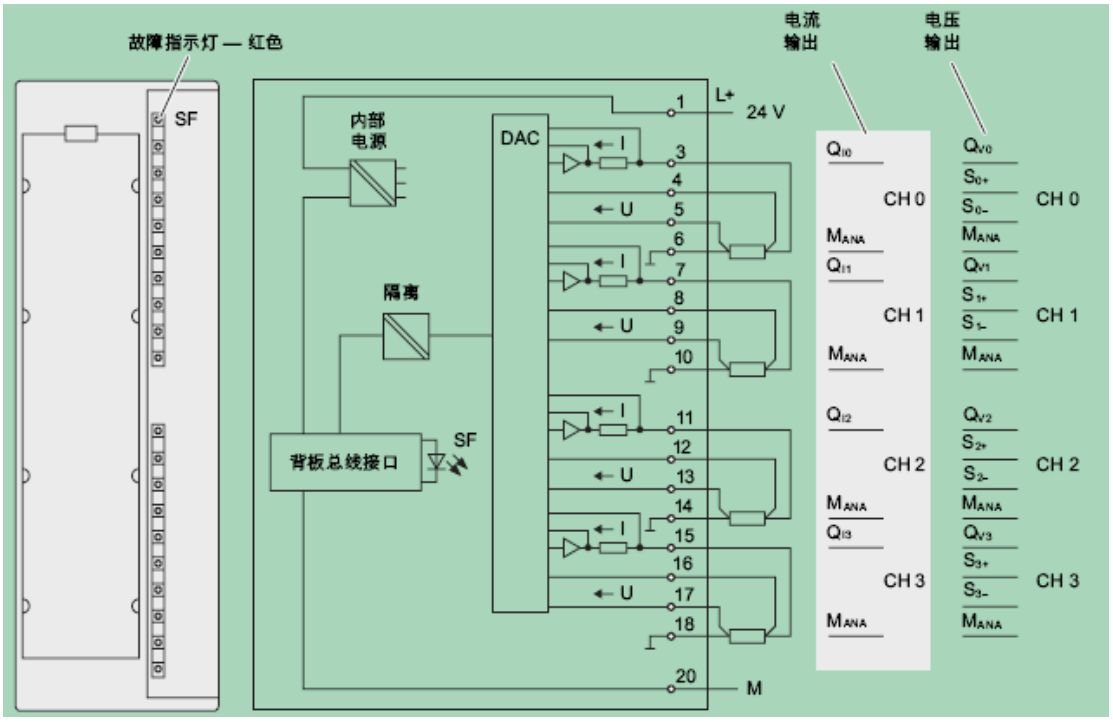
属性及技术数据

CPAC-300-332-5HD01是模拟量输出模块，其主要的属性如下：

- 4 输出通道
- 可将输出通道编程为
  --电压输出
  --电流输出
- 精度12 位
- 可编程诊断
- 可编程诊断中断
- 可编程替换值输出
- 与背板总线接口和负载电压电隔离

外形及接线

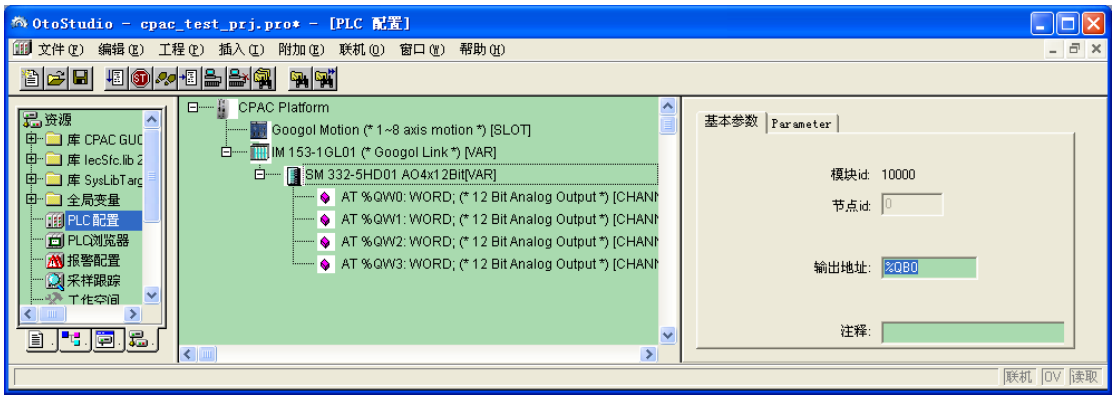
CPAC-300-332-5HD01 的外形及接线图如下图所示：



配置及编程

添加模块

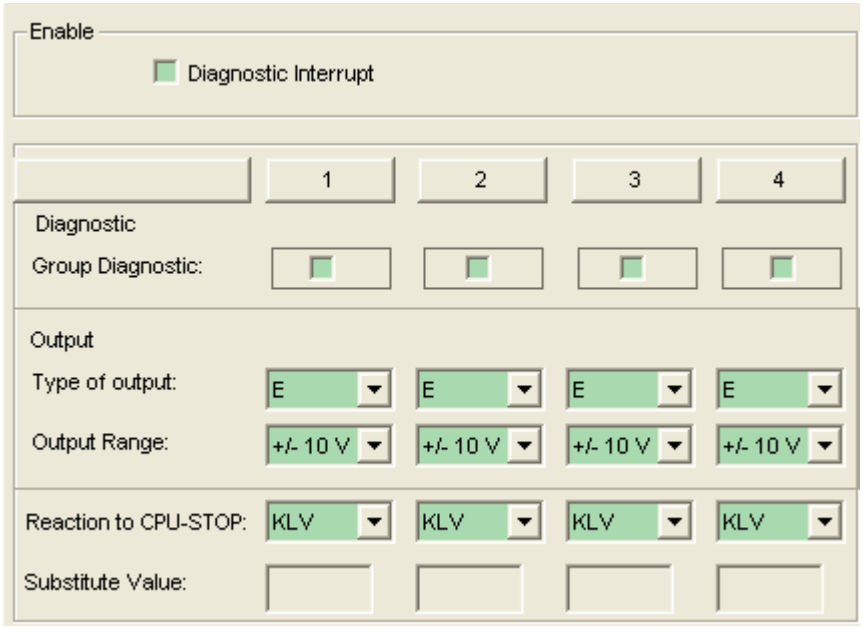
在 OtoStudio 的 PLC_Configuration 中添加一个从站模块，然后在从站模块下添加 CPAC-300-332-5HD01 模块，如下图所示：



模块配置

基本参数包括输出地址的设置，该地址决定模拟量输出模块各通道输出的起始地址。

CPAC-300-332-5HD01 模块的扩展参数设置界面如下图所示：



Diagnostic Interrupt 是针对整个模块的中断配置。如果该单选框被激活，则当一个诊断事件（如配置参数错误、模块短路、L+未接等）发生后，模块将产生一次中断。

Group diagnosis：如果该单选框被激活，则当一个诊断事件（如配置参数错误、模块短路、L+未接等）发生后，对应的诊断信息将被写入模块的诊断数据区。

Output 区包括对各通道的输出类型及范围的配置。E 代表电压，I 代表电流。

Reaction to CPU-STOP：当从站停机时，通道的输出选项设置，包括三个选项，OCV 代表输出关闭，KLV 代表保持 CPU-STOP 前的那个值，SV 选项选中后，对应通道下面的 Substitute Value 将可用，需要用户输入一个值，当 CPU-Stop 时，该通道将输出该值。

模块编程

完成模块的配置后，则模块的通道列表将列出该模块的 4 个通道，通道名详细地描述了该通道的地址等信息，如‘AT %QW0:WORD;(* 12 Bit Analog Output*)[CHANNEL(Q)]’。

在您的 POU 中，可以直接对通道赋值，如 ‘%QW0 := 2048;’，如果该通道配置为+/-10V 的电压模式，则该语句执行后，对应通道将输出 0.73V 的电压，计算方法如下：

OutVol := volMin + (volMax - volMin)*(outValue - valueMin)/(valueMax - valueMin) ;

outVol ---通道上输出的模拟量值；

volMin---该通道配置的模拟量的最小值，如上例的-10v；

volMax---该通道配置的模拟量的最大值，如上例的 10v；

valueMin---该通道对应的数值最小值，见相应模块数值范围表；

valueMax---该通道对应的数值最大值，见相应模块数值范围表；

outValue---通道上输出的数值；

CPAC-300-332-5HD01 模块的数值范围表如下：

量程	输出数值	
	最大值	最小值
±10V	27684	-27684
0~10V	27684	0
1~5V	27684	0
±20mA	27684	-27684
4~20mA	27684	0
0~20mA	27684	0

上面的转换也可以通过 OtoStudio 提供的转换函数实现，请参考函数 AnologData_To_DigitalData 的相关说明。

第十章 指令列表

模拟量输入：CPAC-300-331-7KF01

属性及技术数据

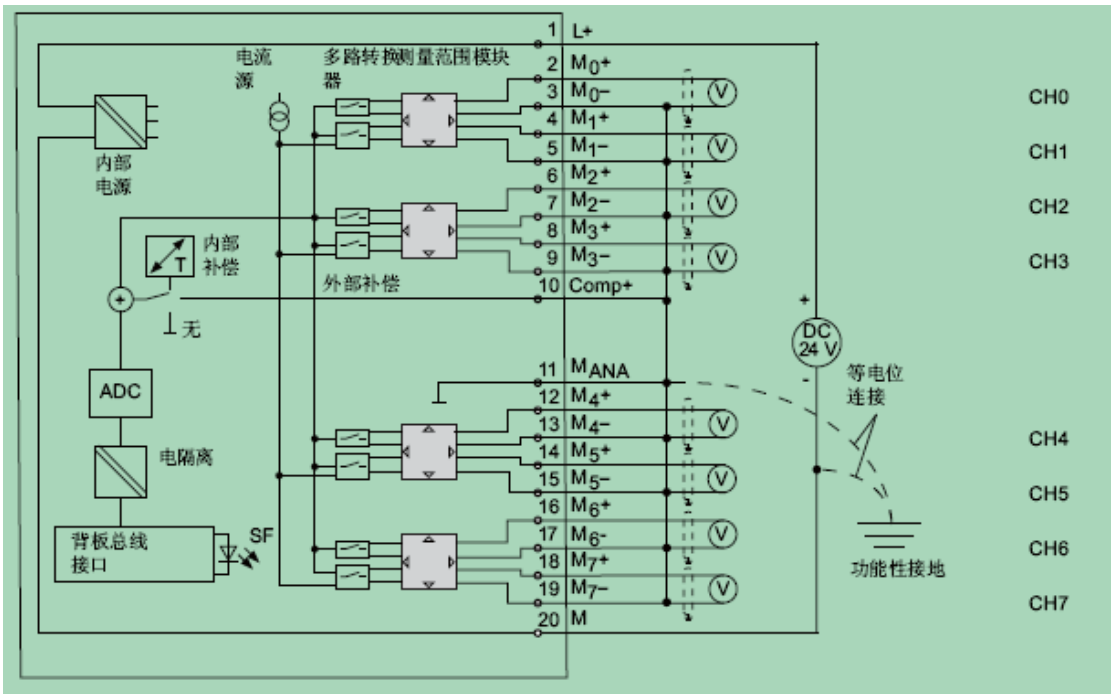
CPAC-300-300-7KF01是模拟量输入模块，其主要的属性如下：

- 4 个通道组中的 8 点输入
- 在每个通道组，测量类型可编程
  - 电压
  - 电流
  - 电阻
  - 温度
- 每个通道组的分辨率均可编程（9/12/14 位 + 符号）
- 每个通道组的任意测量范围选择
- 可编程诊断和诊断中断
- 2 个通道的可编程限制值监视
- 越限时的硬件中断可编程
- 电隔离 CPU 和负载电压（不适用于 2 线制传感器）

测量值分辨率直接与所选的积分时间成比例，即在模拟量输入通道，测量值分辨率与积分时间的长短成正比。

外形及接线

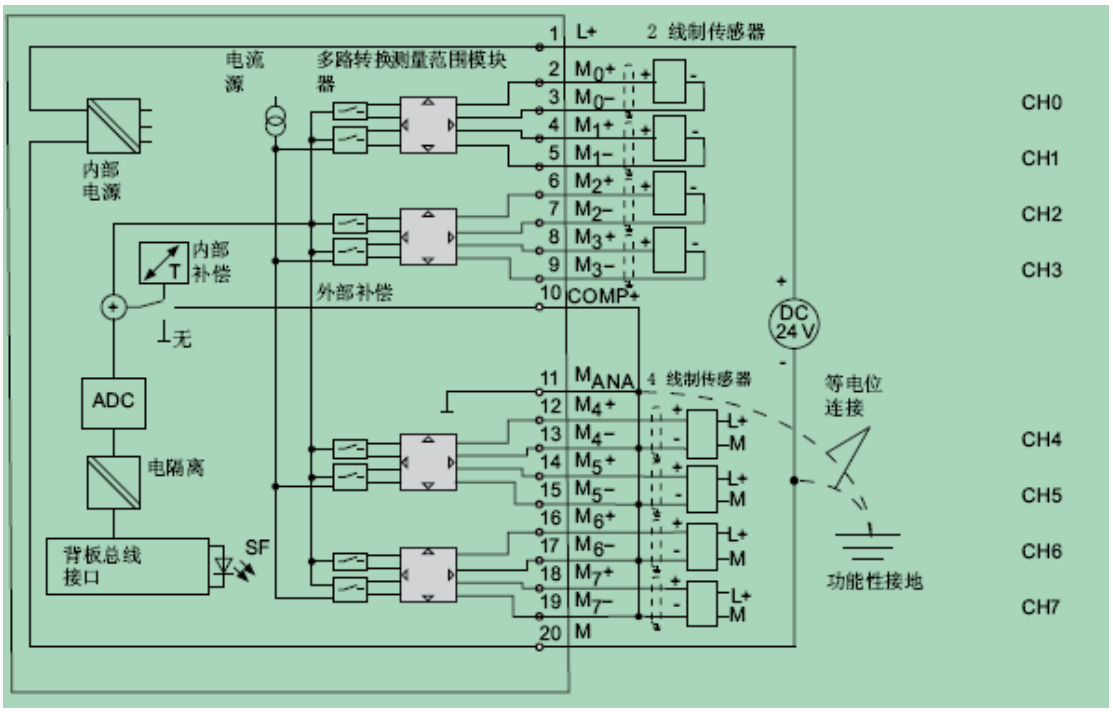
电压测量时的接线如下图所示：



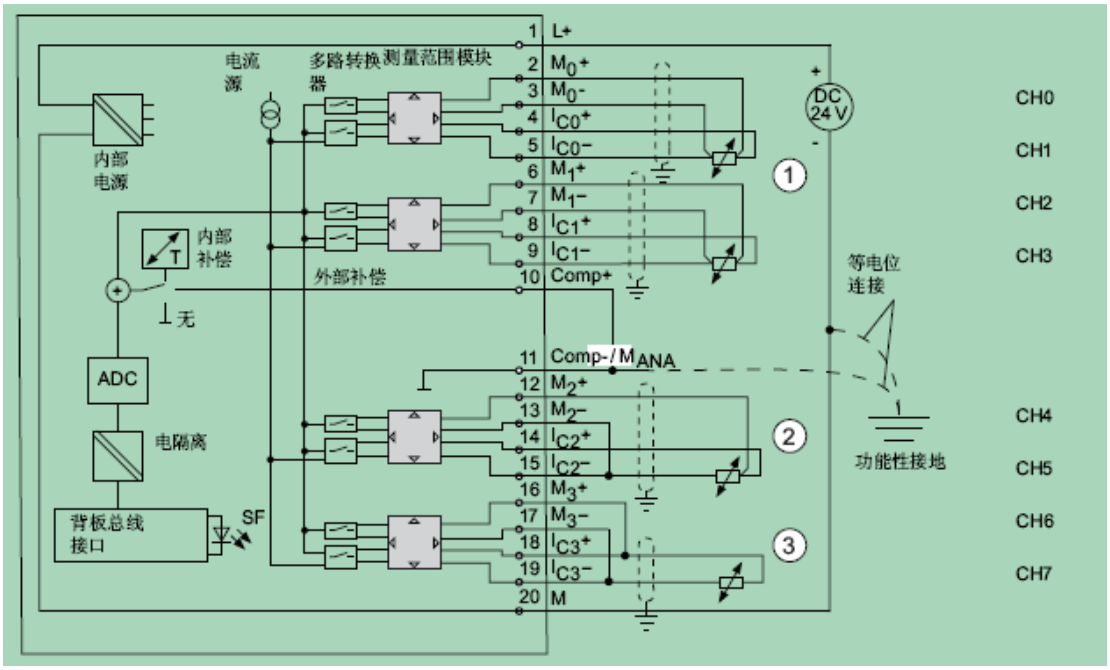
用于电流测量的2线制和4线制测量传感器：

说明：使用非隔离电源的接地 4 线制传感器时，无须互联 M_{ANA} 和 M-（端子 11、13、15、17、

19）。



电阻传感器或温度计的 2 线制、3 线制和 4 线制连接。



上图中：①为 4 线制连接

②为 3 线制连接

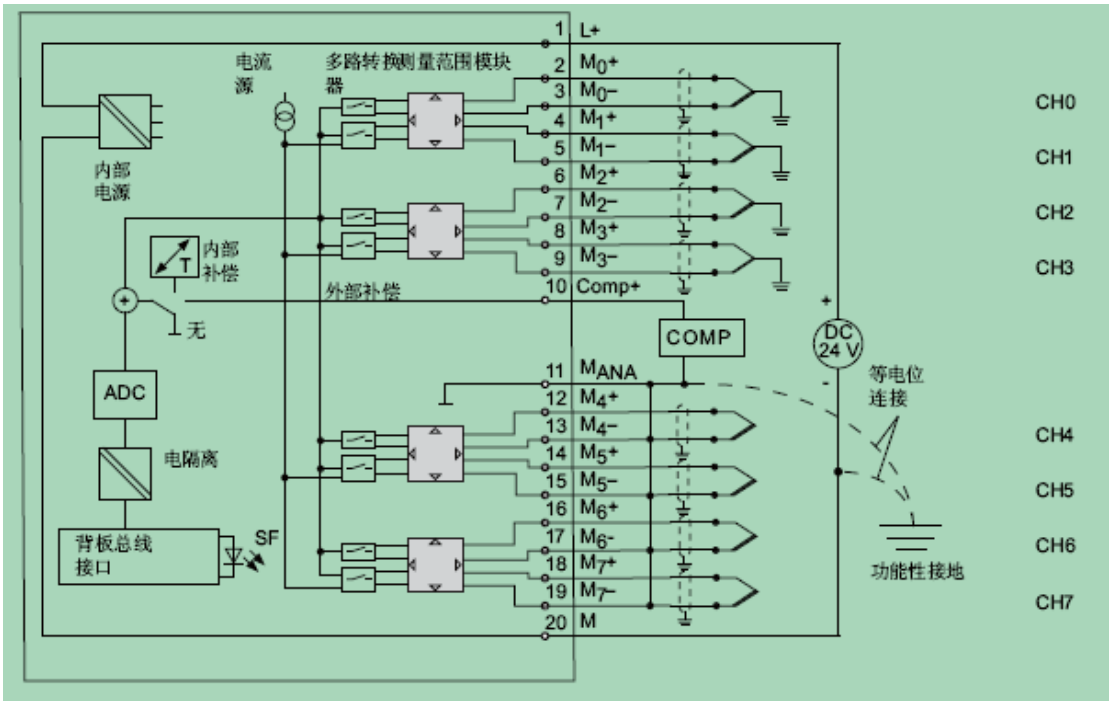
③为 2 线制连接

说明：

1、‘电阻测量’仅在每组的一个通道中可用。相应地，改组的“第二个”通道用于电流测量模块(Ic)。改组的“第 1 个”通道将返回测量值。改组的“第二个”通道分配缺省上溢值“$7FFF_H$”；

2、“2 线制”和“3 线制”连接没有电源阻抗补偿。

带外部补偿的热电偶接线如下图：



说明：

1、使用内部补偿时，必须在 Comp+和 M_{ANA} 间进行桥接。

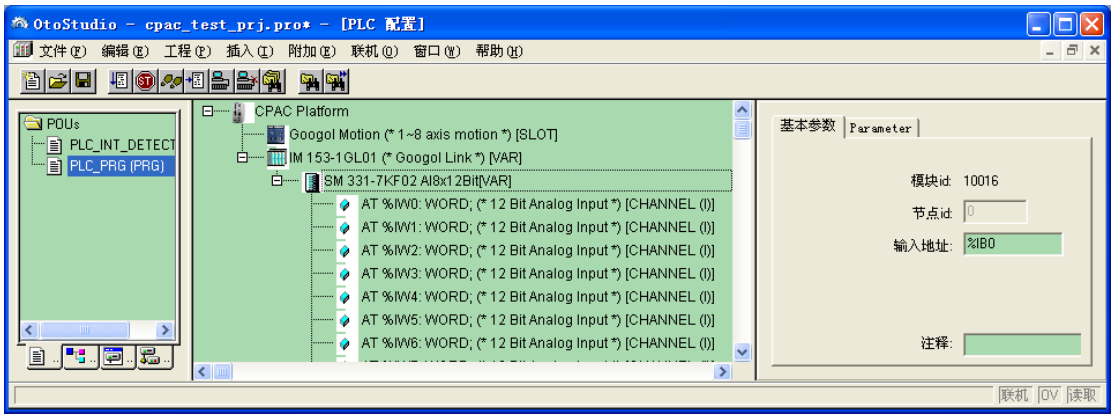
2、使用接地热电偶时，禁止互连 M-和 MANA。 在这种情况下，必须确保适当进行低电阻等电位连接，以便不超过允许的共模电压。

3、在使用非接地热电偶时，将 M- 和 MANA 互连。

配置及编程

添加模块

在 OtoStudio 的 PLC_Configuration 中添加一个从站模块，然后在从站模块下添加 CPAC-300-331-7KF02 模块，如下图所示：

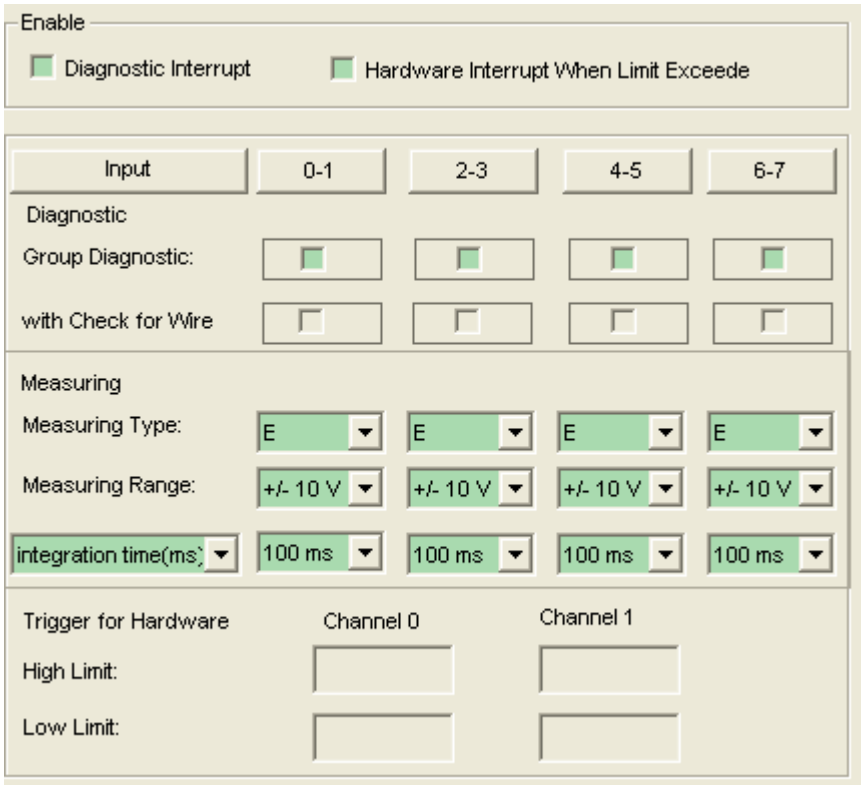


模块配置

基本参数包括输出地址的设置，该地址决定模拟量输出模块各通道输入的起始地址。

CPAC-300-331-7KF02 模块的扩展参数设置界面如下图所示：

第十章 指令列表



Diagnostic Interrupt 是针对整个模块的中断配置。如果该单选框被激活，则当一个诊断事件（如配置参数错误、模块短路、L+未接等）发生后，模块将产生一次中断。

Hardware interrupt when limit exceeded：如果该单选框被激活，则配置对话框下部的‘Trigger for Hardware’的上下限输入框变为可用，用户可在此输入上、下限值。模块运行过程中，一旦输入值高于上限或低于下限，则模块触发一次硬件中断。

Group diagnosis：如果该单选框被激活，则当一个诊断事件（如配置参数错误、模块短路、L+未接等）发生后，对应的诊断信息将被写入模块的诊断数据区。

With wire break check：各通道的断线检查使能。

Measuring 区包括对各通道的输入类型及范围的配置。E 代表电压，4DMU 代表 4 线制电流，2DMU 代表 2 线制电流，R-4L 及 Rt 代表电阻，其他的代表热电偶，下表。

热电偶 TC-I （内部比较）（热敏电压测量）忽略线性化	类型 N [NiCrSi-NiSi] 类型 E [NiCr-CuNi] 类型 J [Fe-CuNi] 类型 K [NiCr-Ni] 类型 L [Fe-CuNi]
热电偶 TC-E （外部比较）（热电电压测量）忽略线性化	
热电偶（线性，内部比较）（温度测量）TC-IL	类型 N [NiCrSi-NiSi] 类型 E [NiCr-CuNi] 类型 J [Fe-CuNi] 类型 K [NiCr-Ni] 类型 L [Fe-CuNi]
热电偶（线性，外部比较）（温度测量）TC-EL	

Position of Measuring Range 下面是对各通道积分时间的选择。

模块编程

第十章 指令列表

完成模块的配置后，则模块的通道列表将列出该模块的 4 个通道，如‘AT %IW0:WORD;(* 12 Bit Analog Input*)[CHANNEL(I)]’。

在您的 POU 中，可以直接采集对通道的值，如 ‘value := %IW0;’，如果得到的 value 变量的值是 2048，且该通道配置为+/-10V 的电压输入，则可以知道对应通道有 0.73V 的电压输入，计算方法如下：

OutVol := volMin + (volMax - volMin)*(value - valueMin)/(valueMax - valueMin) ;

outVol ---通道上输出的模拟量值；

volMin---该通道配置的模拟量的最小值，如上例的-10v；

volMax---该通道配置的模拟量的最大值，如上例的 10v；

valueMin---该通道对应的数值最小值，见相应模块数值范围表；

valueMax---该通道对应的数值最大值，见相应模块数值范围表；

value ---通道上输入的数值；

上面的转换也可以通过 OtoStudio 提供的转换函数实现，请参考函数 DigitalData_To_AnologData 的相关部分。

第十章 指令列表

模拟量输入：CPAC-300-331-1KF01

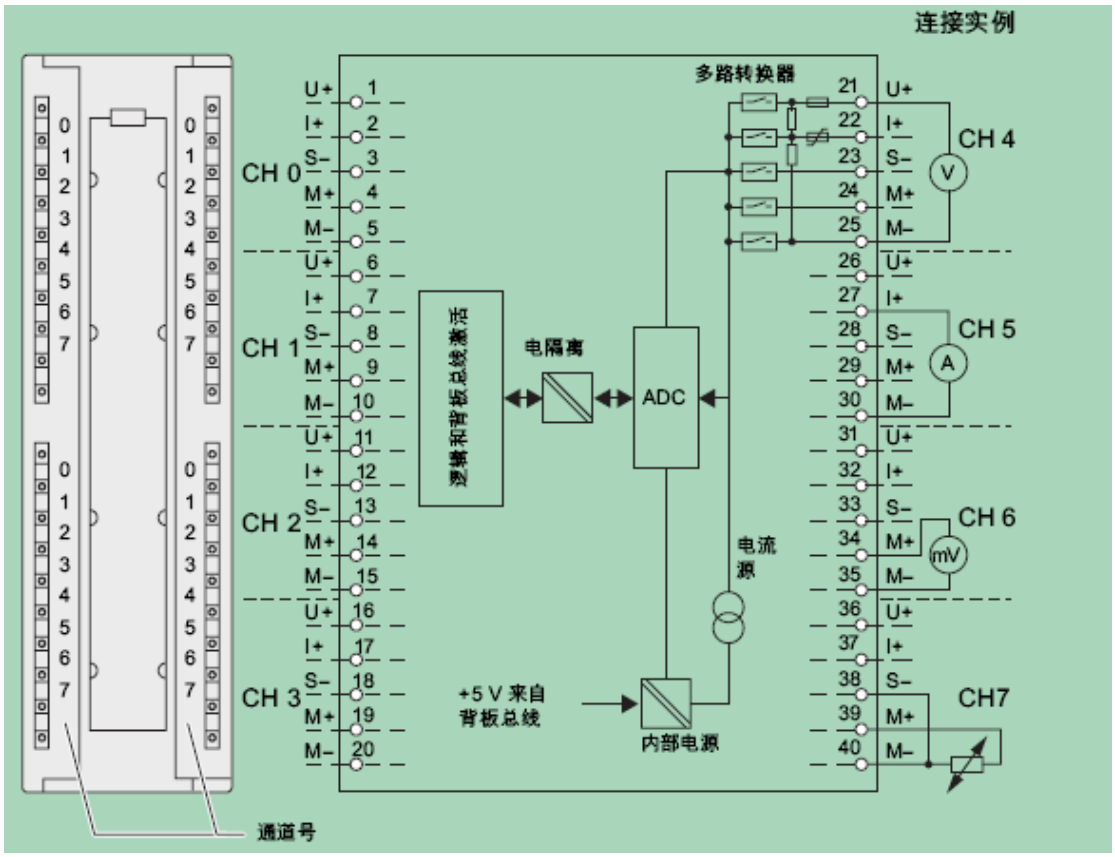
属性及技术数据

CPAC-300-331-1KF01是模拟量输入模块，其主要的属性如下：

- SM 331; AI 8 x 13 位的特性：
- 8 点输入
- 测量值精度12 位 + 符号
- 可选测量方法：
  - 电压
  - 电流
  - 电阻
  - 热电阻
- 与背板总线接口电气隔离

外形及接线

针对不同的测量类型，下图显示了通道 4 到 7 的不同接线实例。这些连接实例适用于所有通道（通道 0 到 7）。



说明：

第十章 指令列表

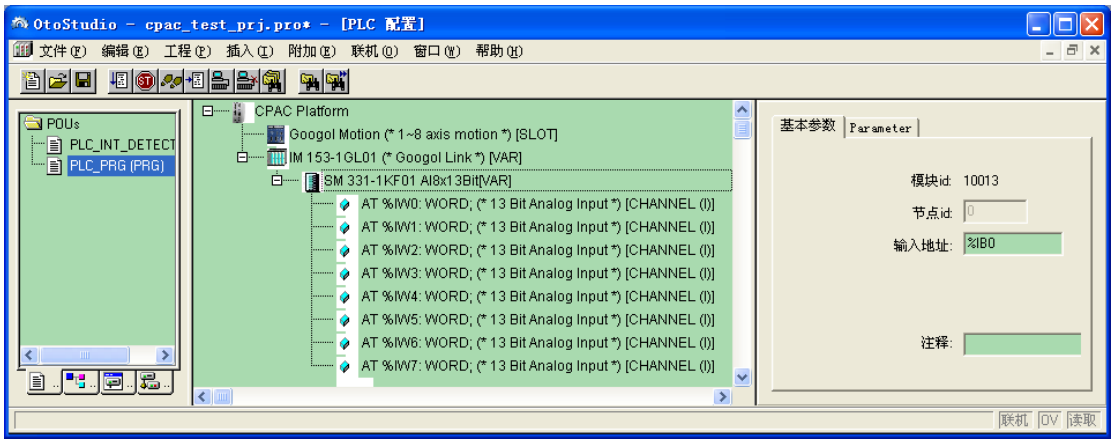
1、 连接电压和电流传感器时，请确保输入之间不超过运行的2V最大共模电压C_{MV}。为了防止出现测量错误，请将各个M-端子连接起来。

2、 在测量电阻和电阻温度计时，不必将M-端子互连。

配置及编程

添加模块

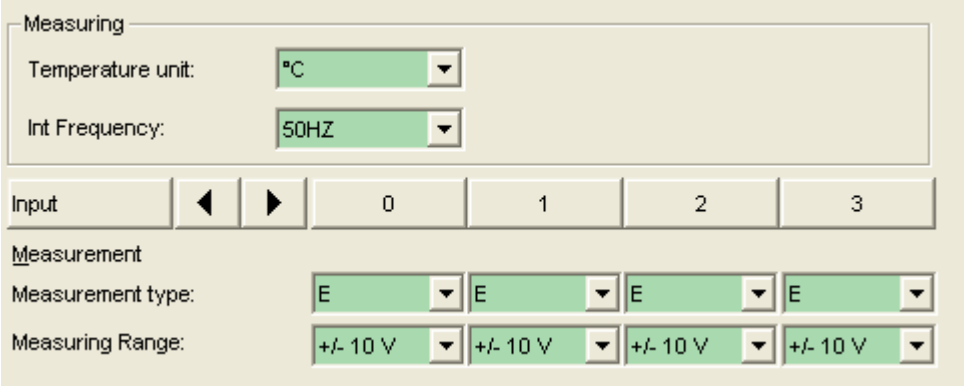
在 OtoStudio 的 PLC_Configuration 中添加一个从站模块，然后在从站模块下添加 CPAC-300-331-1KF01 模块，如下图所示：



模块配置

基本参数包括输出地址的设置，该地址决定模拟量输出模块各通道输出的起始地址。

CPAC-300-331-1KF01 模块的扩展参数设置界面如下图所示：



Temperature Unit 用来选择测量时的温度单位。

Int Frequency:干扰抑制频率的设置，对应着积分时间设置，对应关系如下表：

干扰抑制频率（Hz）	积分时间（ms）
400	10
60	16.6
50	20
10	100

Group diagnosis：如果该单选框被激活，则当一个诊断事件（如配置参数错误、模块短路、L+未接等）发生后，对应的诊断信息将被写入模块的诊断数据区。

Measurement 区包括对各通道的输入类型及范围的配置。E 代表电压，I 代表电流，R 代表电

阻，RTD 代表热电偶。

通过左右移按钮可以切换不同的通道。

模块编程

完成模块的配置后，则模块的通道列表将列出该模块的 4 个通道，如‘AT %IW0:WORD;(* 12 Bit Analog Input*)[CHANNEL(I)]’。

在您的 POU 中，可以直接采集对通道的值，如 ‘value := %IW0;’，如果得到的 value 变量的值是 2048，且该通道配置为+/-10V 的电压输入，则可以知道对应通道有 0.73V 的电压输入，计算方法如下：

OutVol := volMin + (volMax - volMin)*(value - valueMin)/(valueMax - valueMin) ;

outVol ---通道上输出的模拟量值；

volMin---该通道配置的模拟量的最小值，如上例的-10v；

volMax---该通道配置的模拟量的最大值，如上例的 10v；

valueMin---该通道对应的数值最小值，见相应模块数值范围表；

valueMax---该通道对应的数值最大值，见相应模块数值范围表；

value ---通道上输入的数值；

上面的转换也可以通过 OtoStudio 提供的转换函数实现，请参考函数 DigitalData_To_AnologData 的相关部分。

第十章 指令列表

模拟量输入：CPAC-300-331-7NF00

属性及技术数据

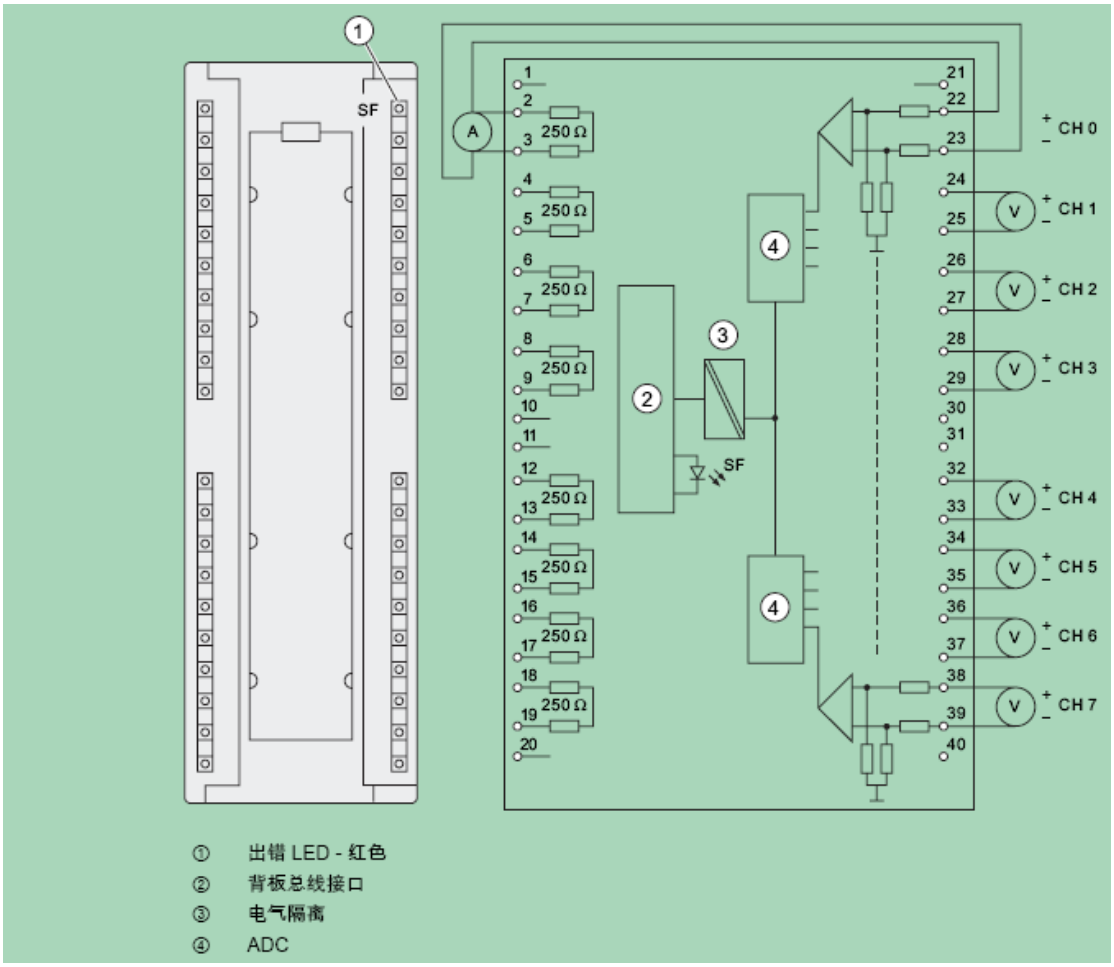
CPAC-300-331-7NF00是模拟量输入模块，其主要的属性如下：

- 4 个通道组中的8 点输入
- 测量值精度 = 15 位 + 符号（独立于积分时间）
- 每个通道组的可选测量方法：
  - 电压
  - 电流
- 用户定义的测量范围设置和每个通道组的过滤/刷新率
- 可编程诊断
- 可编程诊断中断
- 带限制值监视功能的2 个通道
- 超限时的可编程中断
- 与背板总线接口电气隔离
- 通道间的允许CMV：最大50 VDC

外形及接线

接线及方框图如下：

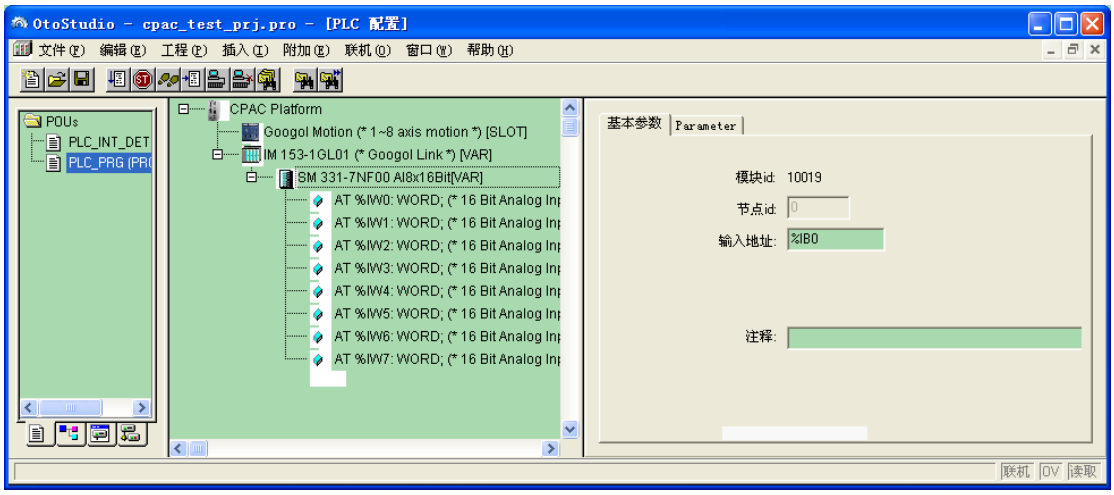
通道 0 用于电流测量，通道 7 用于电压测量。



配置及编程

添加模块

在 OtoStudio 的 PLC_Configuration 中添加一个从站模块，然后在从站模块下添加 CPAC-300-331-7NF00 模块，如下图所示：

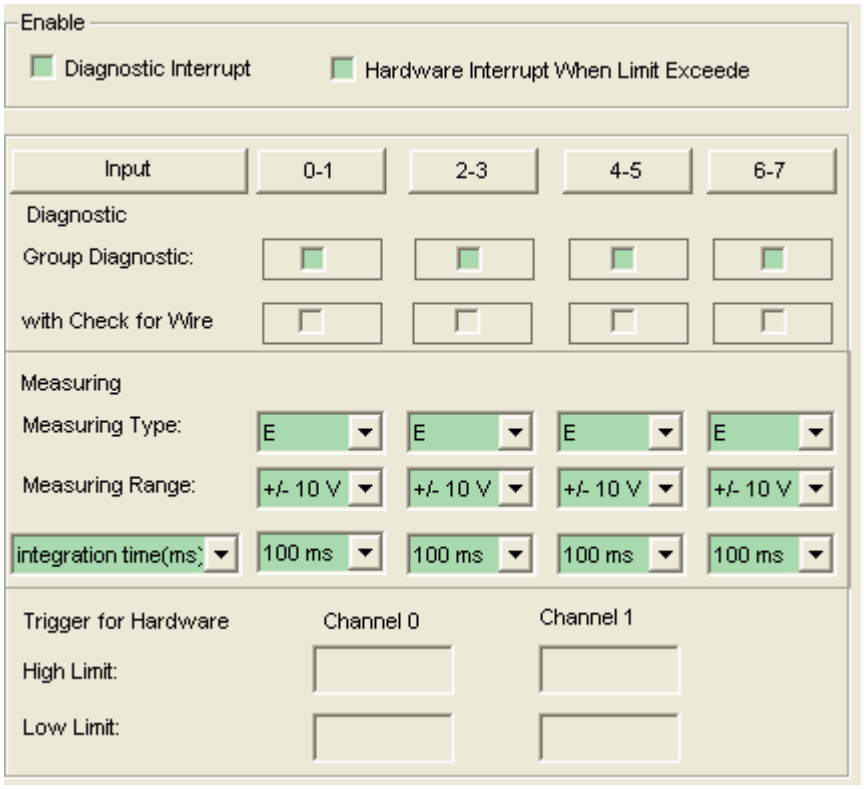


模块配置

第十章 指令列表

基本参数包括输出地址的设置，该地址决定模拟量输出模块各通道输入的起始地址。

CPAC-300-331-7NF00 模块的扩展参数设置界面如下图所示：



Diagnostic Interrupt 是针对整个模块的中断配置。如果该单选框被激活，则当一个诊断事件（如配置参数错误、模块短路、L+未接等）发生后，模块将产生一次中断。

Hardware interrupt when limit exceeded：如果该单选框被激活，则配置对话框下部的‘Trigger for Hardware’的上下限输入框变为可用，用户可在此输入上、下限值。模块运行过程中，一旦输入值高于上限或低于下限，则模块触发一次硬件中断。

Group diagnosis：如果该单选框被激活，则当一个诊断事件（如配置参数错误、模块短路、L+未接等）发生后，对应的诊断信息将被写入模块的诊断数据区。

With wire break check：各通道的断线检查使能。

Measuring 区包括对各通道的输入类型及范围的配置。E 代表电压，4DMU 代表 4 线制电流。

Position of Measuring Range 下面是对各通道积分时间的选择。

模块编程

完成模块的配置后，则模块的通道列表将列出该模块的 4 个通道，如‘AT %IW0:WORD;(* 12 Bit Analog Input*)[CHANNEL(I)]’。

在您的 POU 中，可以直接采集对通道的值，如 ‘value := %IW0;’，如果得到的 value 变量的值是 2048，且该通道配置为+/-10V 的电压输入，则可以知道对应通道有 0.73V 的电压输入，计算方法如下：

OutVol := volMin + (volMax - volMin)*(value - valueMin)/(valueMax - valueMin) ;

outVol ---通道上输出的模拟量值；

volMin---该通道配置的模拟量的最小值，如上例的-10v；

volMax---该通道配置的模拟量的最大值，如上例的 10v；

valueMin---该通道对应的数值最小值，见相应模块数值范围表；

valueMax---该通道对应的数值最大值，见相应模块数值范围表；

value ---通道上输入的数值；

上面的转换也可以通过 OtoStudio 提供的转换函数实现，请参考函数 DigitalData_To_AnologData 的相关部分。

第十章 指令列表

模拟量输出：CPAC-300-332-5HF00

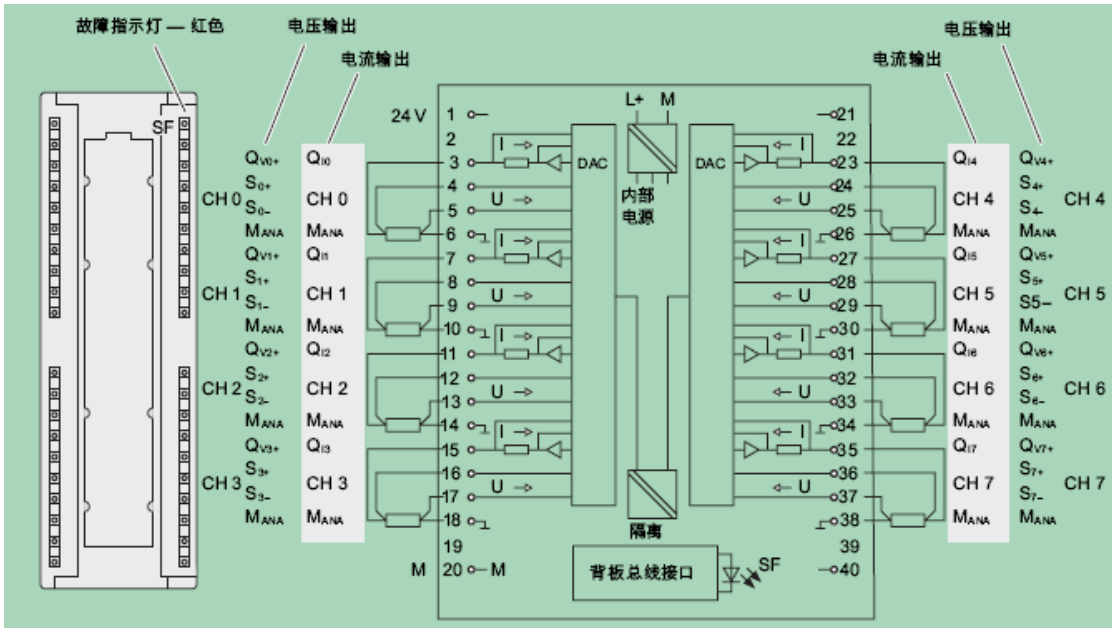
属性及技术数据

CPAC-300-332-5HF00是模拟量输出模块，其主要的属性如下：

- 8 输出通道
- 可将输出通道编程为
  - 电压输出
  - 电流输出
- 精度12 位
- 可编程诊断
- 可编程诊断中断
- 与背板总线接口和负载电压电隔离

外形及接线

CPAC-300-332-5HF00 的外形及接线图如下图所示：

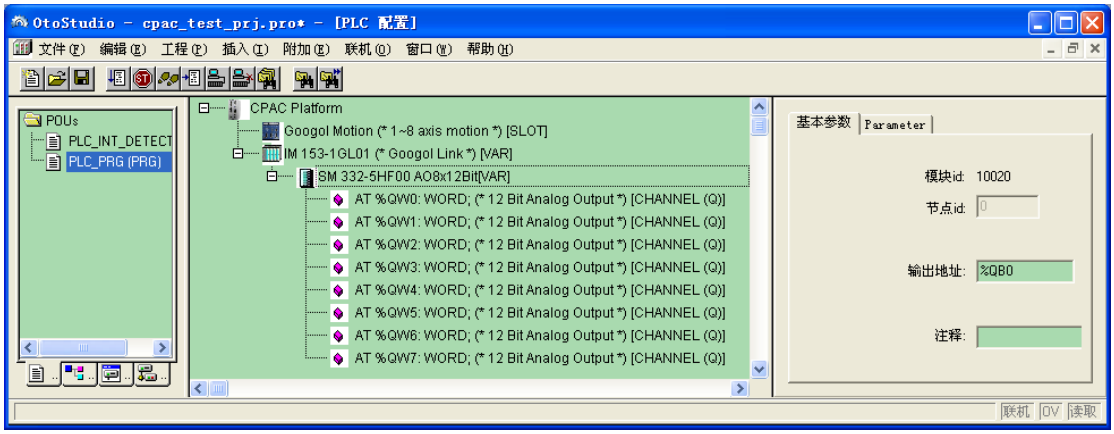


配置及编程

添加模块

在 OtoStudio 的 PLC_Configuration 中添加一个从站模块，然后在从站模块下添加 CPAC-300-332-5HF00 模块，如下图所示：

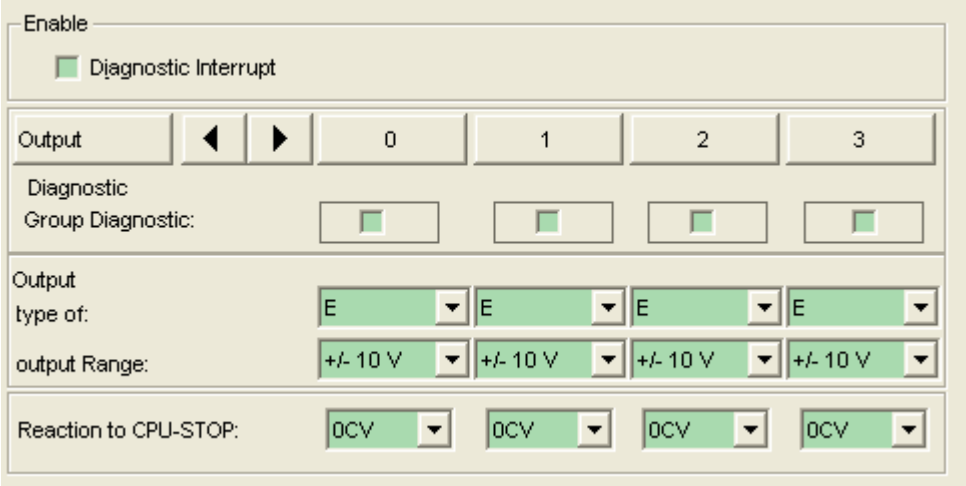
第十章 指令列表



模块配置

基本参数包括输出地址的设置，该地址决定模拟量输出模块各通道输出的起始地址。

CPAC-300-332-5HF00 模块的扩展参数设置界面如下图所示：



Diagnostic Interrupt 是针对整个模块的中断配置。如果该单选框被激活，则当一个诊断事件（如配置参数错误、模块短路、L+未接等）发生后，模块将产生一次中断。

Group diagnosis：如果该单选框被激活，则当一个诊断事件（如配置参数错误、模块短路、L+未接等）发生后，对应的诊断信息将被写入模块的诊断数据区。

Output 区包括对各通道的输出类型及范围的配置。E 代表电压，I 代表电流。

Reaction to CPU-STOP：当从站停机时，通道的输出选项设置，包括三个选项，0CV 代表输出关闭，KLV 代表保持 CPU-STOP 前的那个值。

模块编程

完成模块的配置后，则模块的通道列表将列出该模块的 4 个通道，通道名详细地描述了该通道的地址等信息，如‘AT %QW0:WORD;(* 12 Bit Analog Output*)[CHANNEL(Q)]’。

在您的 POU 中，可以直接对通道赋值，如 ‘%QW0 := 2048;’，如果该通道配置为+/-10V 的电压模式，则该语句执行后，对应通道将输出 0.73V 的电压，计算方法如下：

OutVol := volMin + (volMax - volMin)*(outValue - valueMin)/(valueMax - valueMin) ;

outVol ---通道上输出的模拟量值；

volMin---该通道配置的模拟量的最小值，如上例的-10v；

volMax---该通道配置的模拟量的最大值，如上例的 10v；

valueMin---该通道对应的数值最小值，见相应模块数值范围表；

valueMax---该通道对应的数值最大值，见相应模块数值范围表；

outValue---通道上输出的数值；

上面的转换也可以通过 OtoStudio 提供的转换函数实现，请参考函数 AnologData_To_DigitalData 的相关说明。

模拟量输入：CPAC-300-331-7KF02

请参考 CPAC-300-331-7KF01 模块。

第十章 指令列表

数字量模块

数字量模块列表	
CPAC-300-321-1BH01	16 路数字量输入模块
CPAC-300-321-1BH02	16 路数字量输入模块
CPAC-300-321-1BH50	16 路数字量输入模块
CPAC-300-321-1BL00	32 路数字量输入模块
CPAC-300-322-1BF01	8 路数字量输出模块
CPAC-300-322-1BH01	16 路数字量输出模块
CPAC-300-322-1BL00	32 路数字量输出模块
CPAC-300-322-1HF01	8 路继电器输出，浮地
CPAC-300-322-1HH01	16 路继电器输出，浮地
CPAC-300-322-1HH50	16 路继电器输出，干接地
CPAC-300-323-1BH01	8 Di 8 Do 数字量输入输出模块
CPAC-300-323-1BL00	16 Di 16 Do 数字量输入输出模块

数字量输入：CPAC-300-321-1BH01

CPAC-300-321-1BH02 与 CPAC-300-321-1BH01 完全等同。

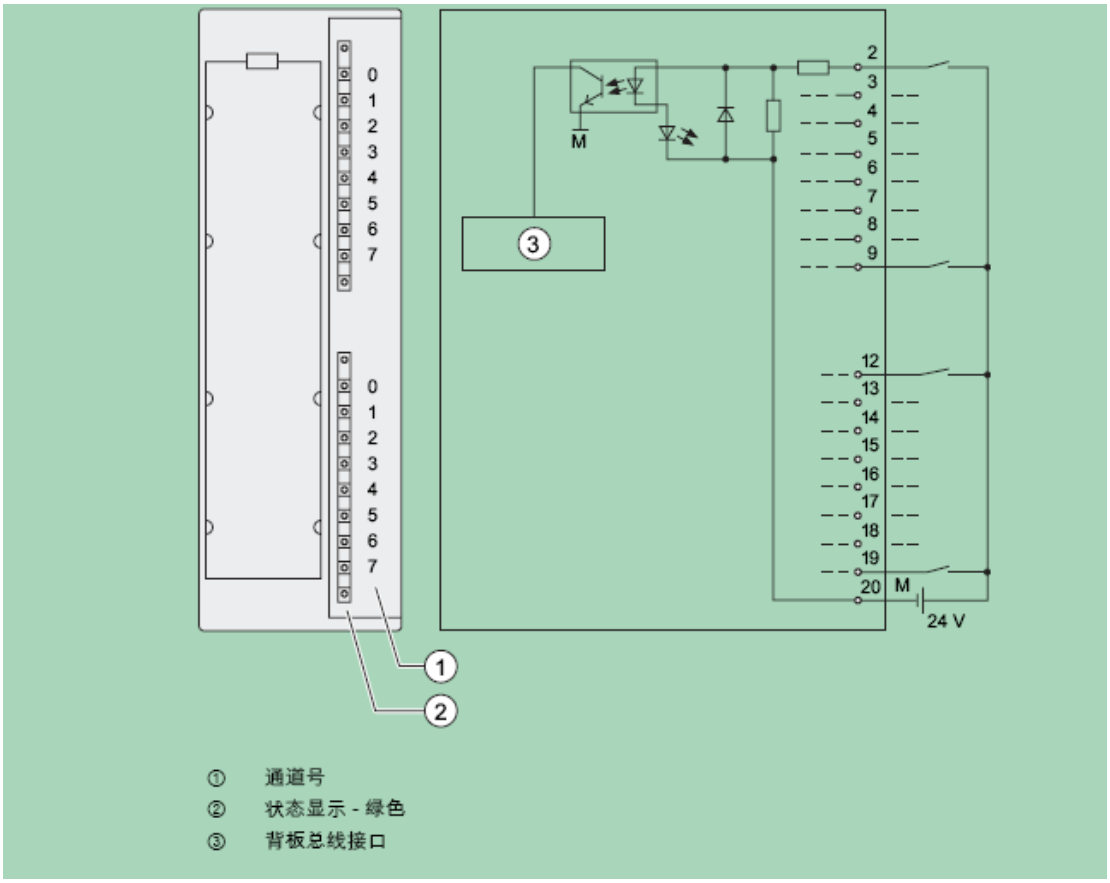
属性及技术数据

CPAC-300-321-1BH01及CPAC-300-321-1BH02是数字量的输入模块，其主要的属性如下：

- 16 点输入，电气隔离为16 组
- 额定输入电压24 VDC
- 适用于开关以及2-/3-/4-线接近开关 (BERO)

外形及接线

CPAC-300-321-1BH01 的外形及接线图如下图所示：

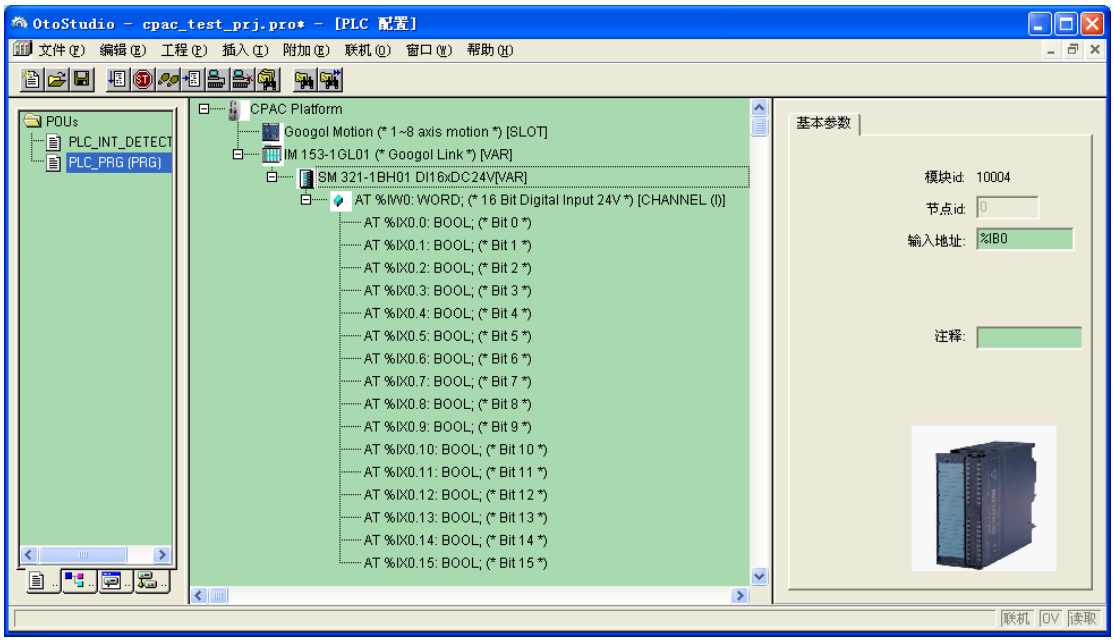


配置及编程

添加模块

在 OtoStudio 的 PLC_Configuration 中添加一个从站模块，然后在从站模块下添加 CPAC-300-321-1BH01 模块，如下图所示：

第十章 指令列表



模块配置

配置参数包括输入地址的设置，该地址即模块各通道输入的地址。

模块编程

完成模块的配置后，则模块的通道列表将列出该模块的每个通道，通道名详细地描述了该通道的地址、类型等信息。

在您的 POU 中，可以直接采集整个 16 位通道的输入值，如 ‘diValue := %IW0;’，也可以采集单个通道的输入值，如‘diBitValue := %IX0.0’。

第十章 指令列表

数字量输入：CPAC-300-321-1BH50

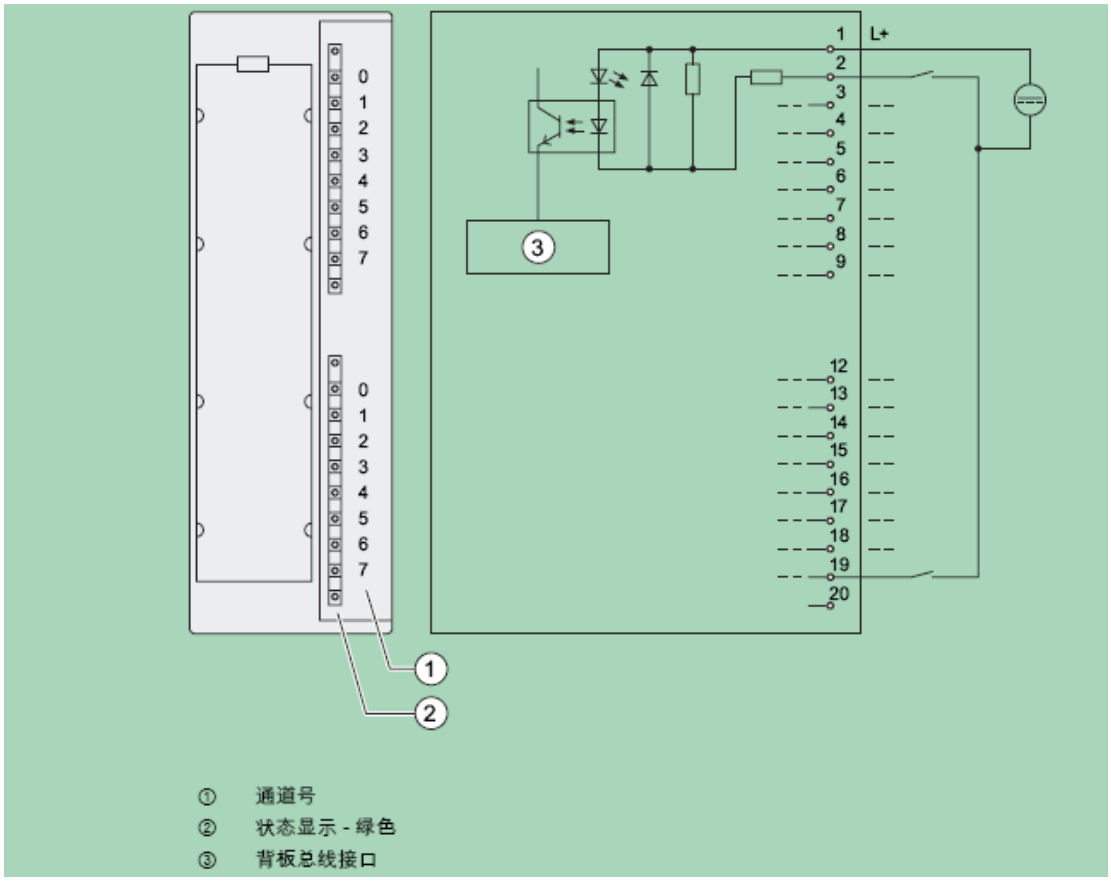
属性及技术数据

CPAC-300-321-1BH50是数字量的输入模块，其主要的属性如下：

- 16 个输入端，源输入，电气隔离为16 组
- 额定输入电压24 VDC
- 适用于开关以及2-/3-/4-线接近开关 (BERO)

外形及接线

CPAC-300-321-1BH50 的外形及接线图如下图所示：



配置及编程

请参考 CPAC-300-321-1BH01 的配置及编程。

数字量输入：CPAC-300-321-1BL00

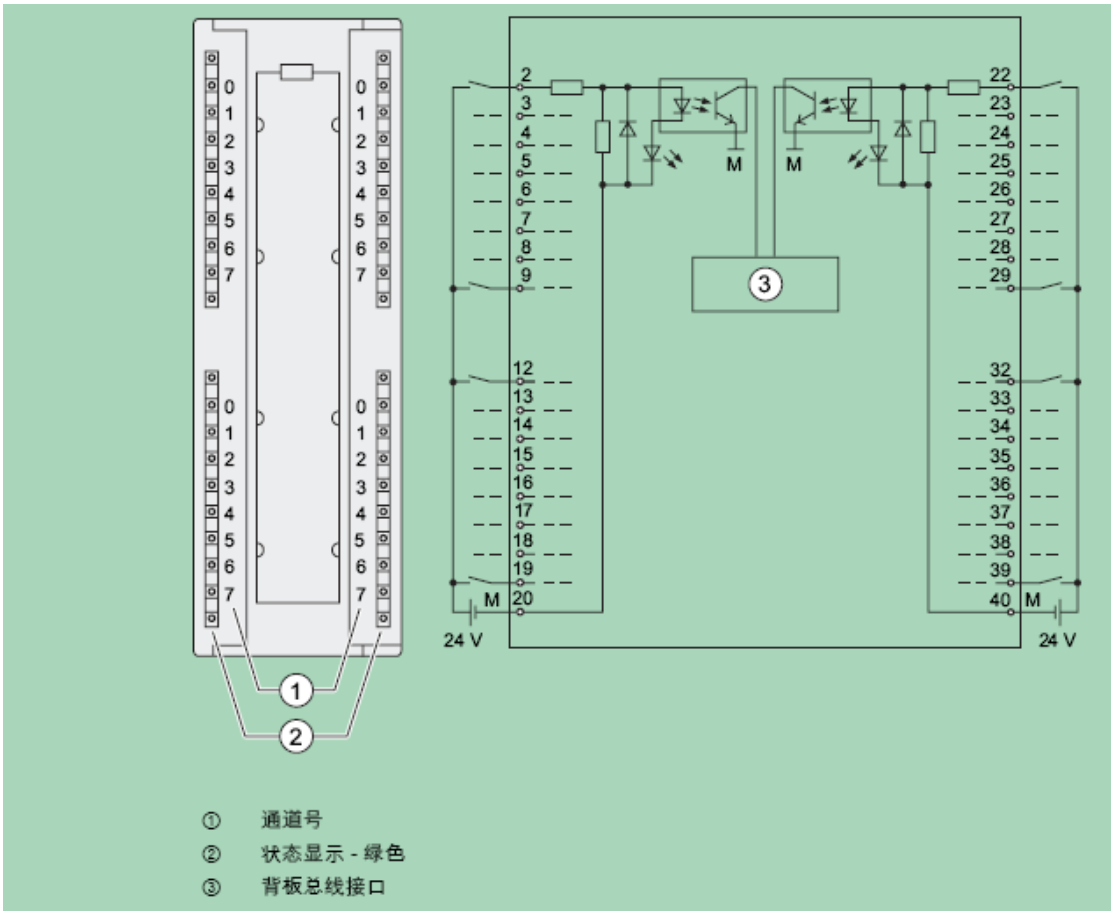
属性及技术数据

CPAC-300-321-1BL00是数字量的输入模块，其主要的属性如下：

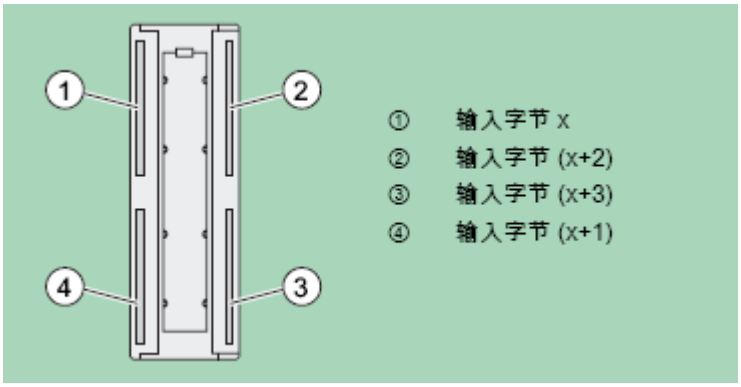
- 32 点输入，电气隔离为16 组
- 额定输入电压24 VDC
- 适用于开关以及2-/3-/4-线接近开关 (BERO)

外形及接线

CPAC-300-321-1BL00 的外形及接线图如下图所示：



通道与地址的对应关系如下图：



配置及编程

请参考 CPAC-300-321-1BH01 的配置及编程。

数字量输入：CPAC-300-321-1BH02

请参考 CPAC-300-321-1BH01。

数字量输出：CPAC-300-322-1BF01

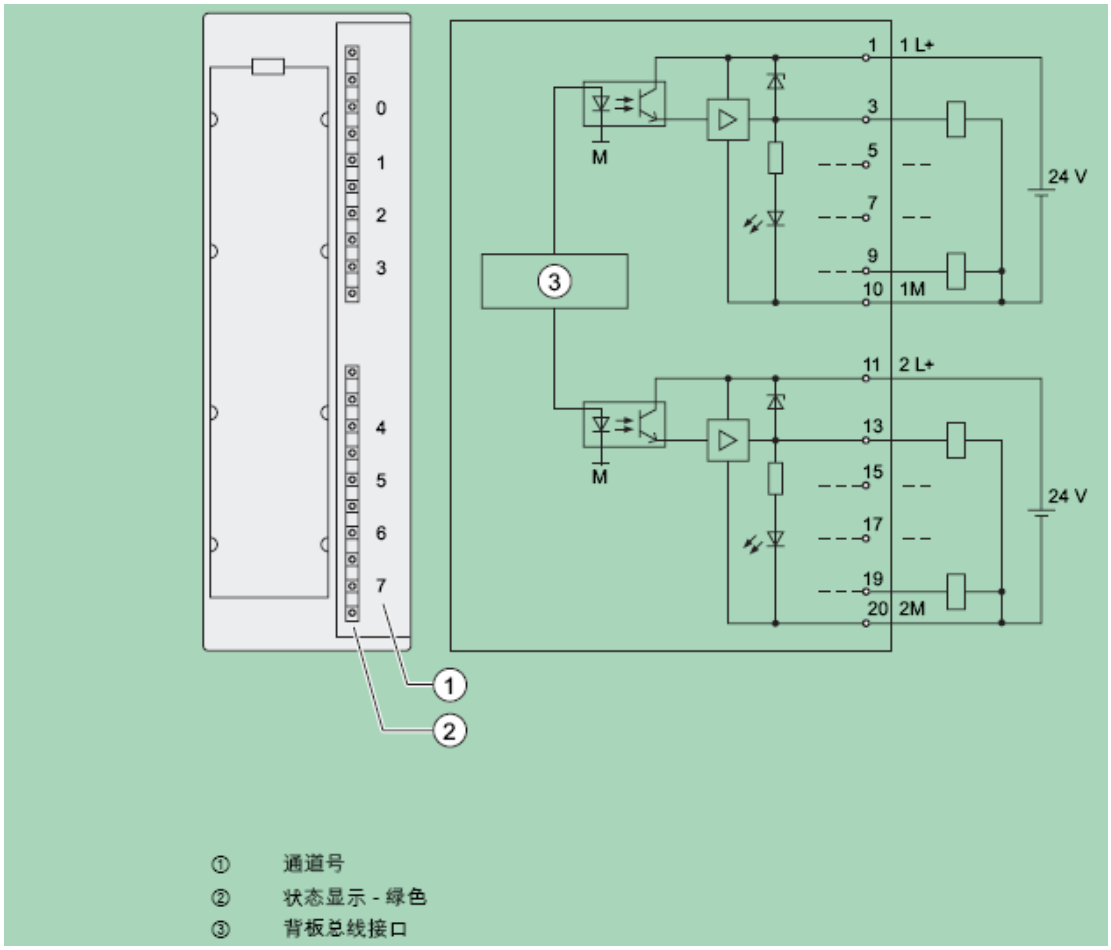
属性及技术数据

CPAC-300-322-1BF01是数字量的输出模块，其主要的属性如下：

- 8 点输出，电气隔离为4 组
- 输出电流为2 A
- 额定负载电压24 VDC
- 适用于电磁阀、DC 接触器和信号灯

外形及接线

CPAC-300-322-1BF01 的外形及接线图如下图所示：

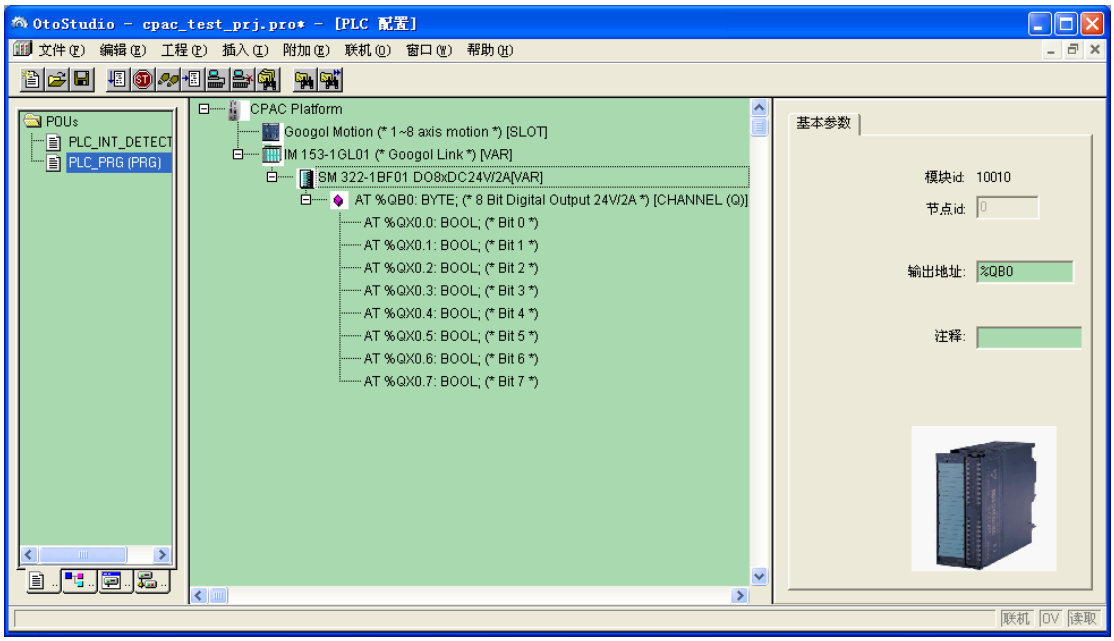


配置及编程

添加模块

第十章 指令列表

在 OtoStudio 的 PLC_Configuration 中添加一个从站模块，然后在从站模块下添加 CPAC-300-322-1BF01 模块，如下图所示：



模块配置

配置参数包括输出地址的设置，该地址即模块各通道输出的地址，通道名详细地描述了该通道的地址、类型等信息。

模块编程

完成模块的配置后，则模块的通道列表将列出该模块的每个通道。

在您的 POU 中，可以直接对整个 16 位通道赋值，如 ‘%IB0 := 255;’，也可以对单个通道赋值，如‘%QX0.0 := 1’。

数字量输出：CPAC-300-322-1BL00

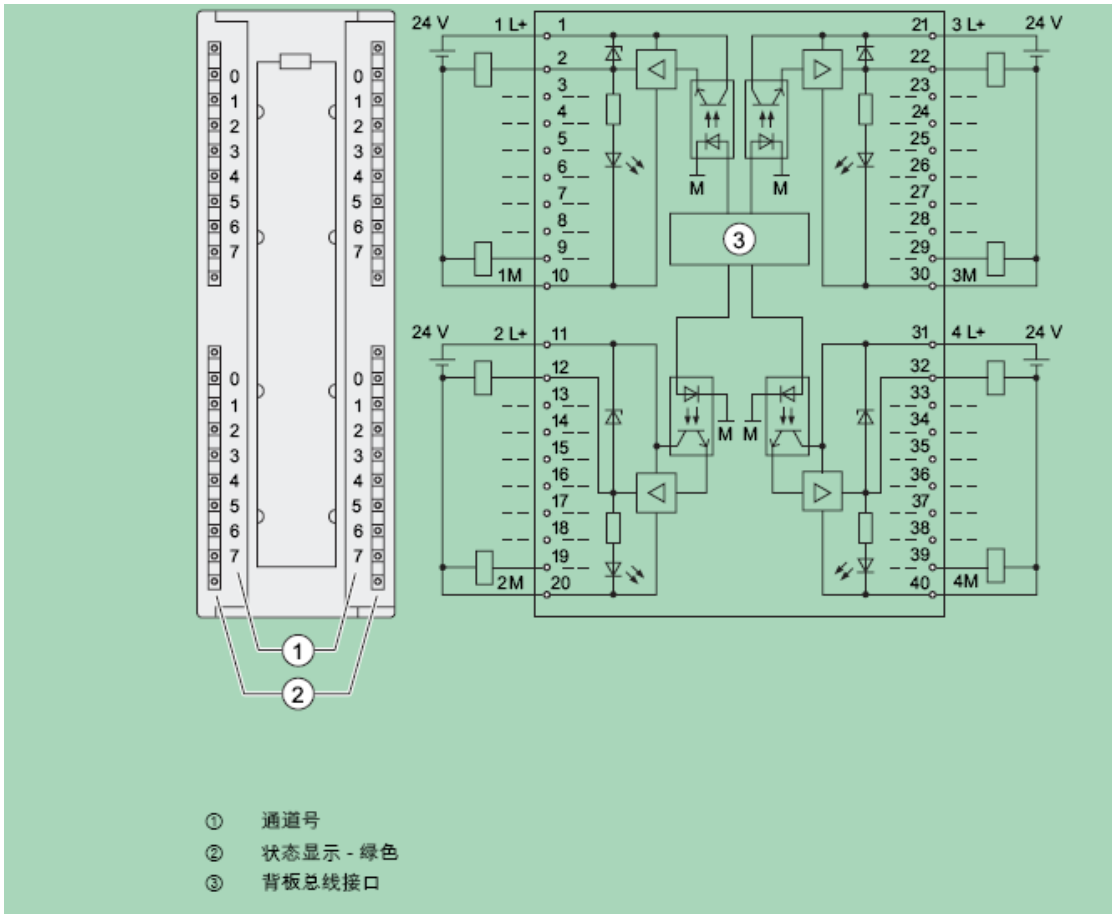
属性及技术数据

CPAC-300-322-1BL00是数字量的输出模块，其主要的属性如下：

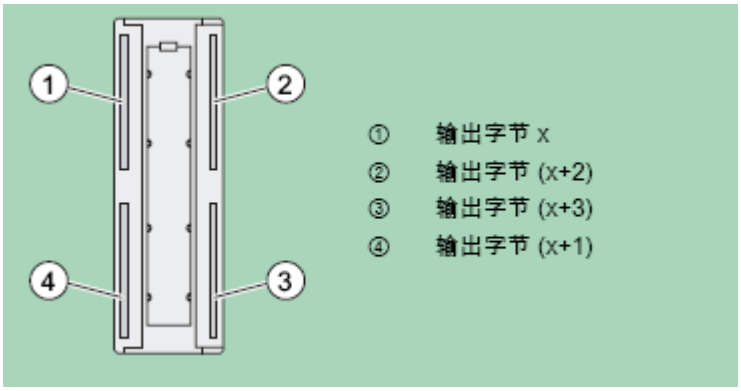
- 32 点输出，电气隔离为8 组
- 输出电流为0.5 A
- 额定负载电压24 VDC
- 适用于电磁阀、DC 接触器和信号灯

外形及接线

CPAC-300-322-1BL00 的外形及接线图如下图所示：



输出通道及输出地址对应关系如下图：



配置及编程

请参考 CPAC-300-322-1BF01 的配置及编程。

数字量输出：CPAC-300-322-1BH01

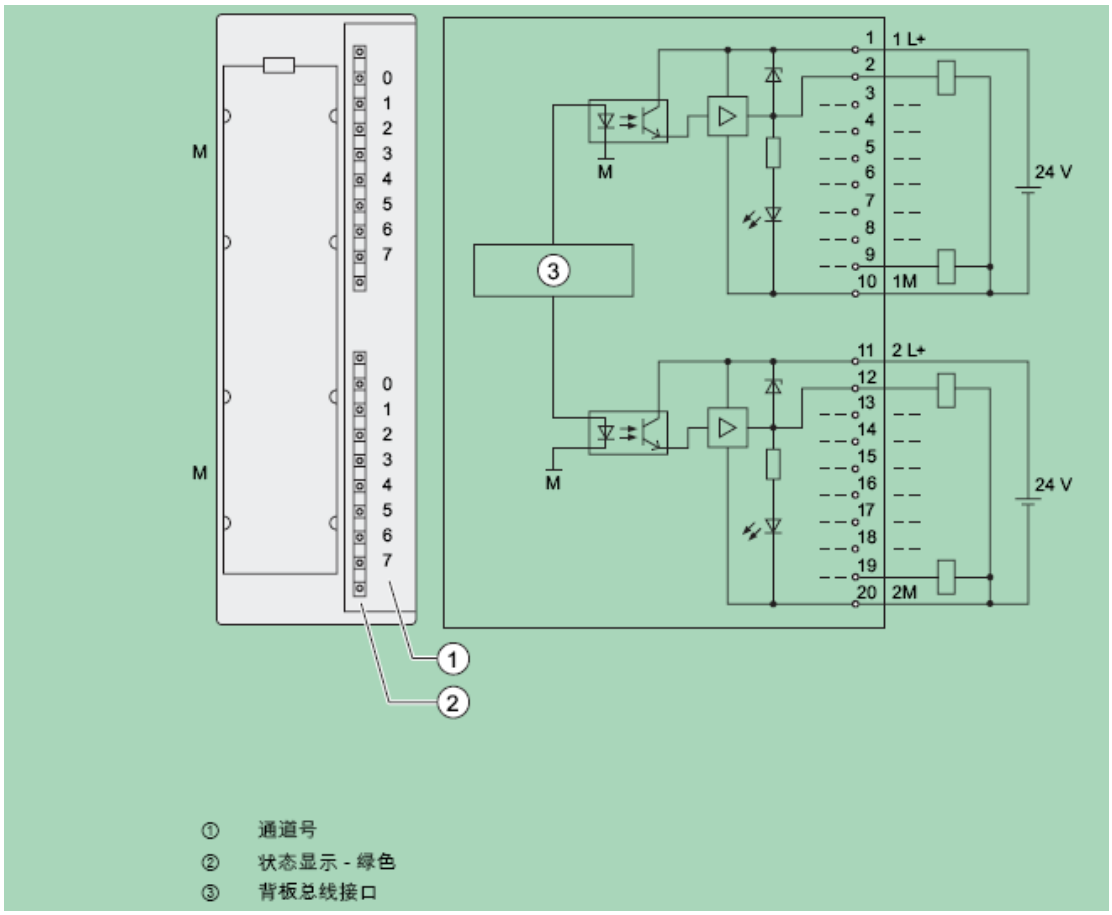
属性及技术数据

CPAC-300-322-1BH01是数字量的输出模块，其主要的属性如下：

- 16 点输出，电气隔离为8 组
- 输出电流为0.5 A
- 额定负载电压24 VDC
- 适用于电磁阀、DC 接触器和信号灯

外形及接线

CPAC-300-322-1BH01 的外形及接线图如下图所示：



配置及编程

请参考 CPAC-300-322-1BF01 的配置及编程。

数字量输出：CPAC-300-322-1HF01

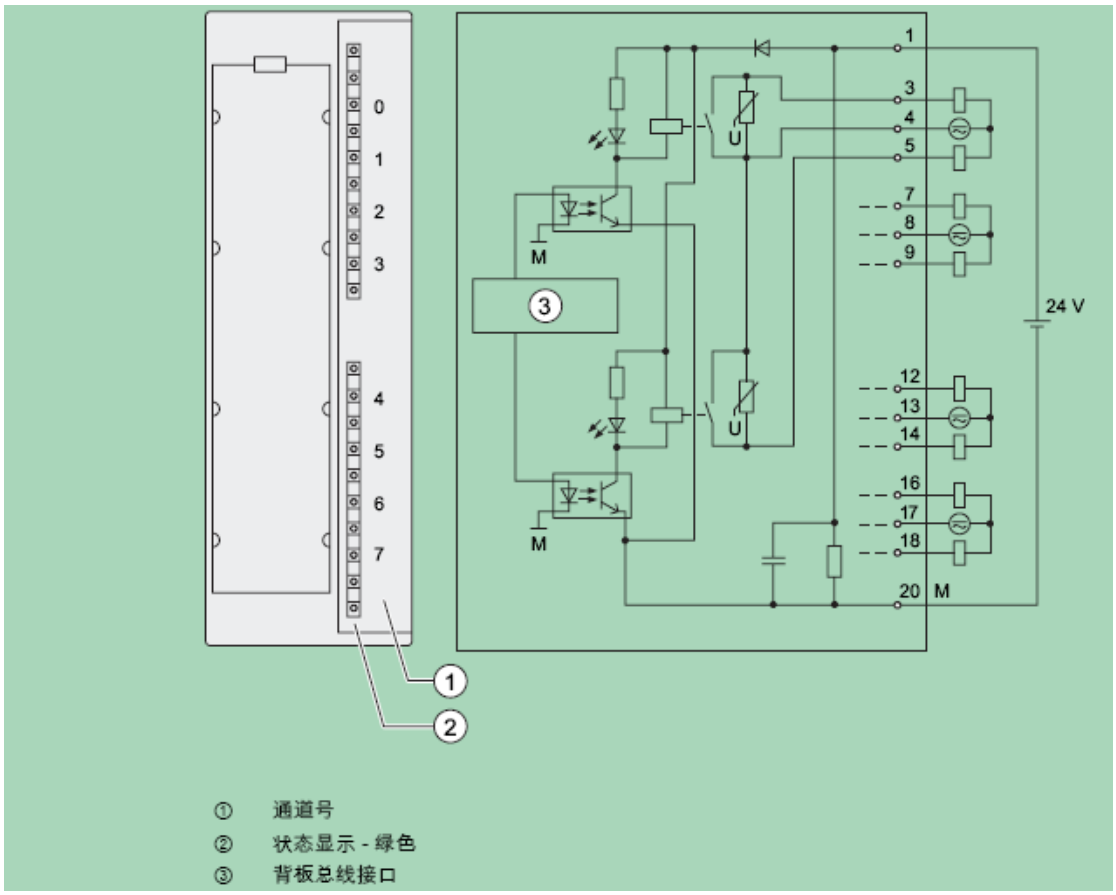
属性及技术数据

CPAC-300-322-1HF01是继电器输出模块，其主要的属性如下：

- 8 点输出，电隔离为2 组
- 额定负载电压为24 VDC 到120 VDC，48 VAC 到230 VAC
- 适用于AC/DC 电磁阀、接触器、电机起动器、FHP 电机和信号灯。

外形及接线

CPAC-300-322-1HF01 的外形及接线图如下图所示：



配置及编程

请参考 CPAC-300-322-1BF01 的配置及编程。

数字量输出：CPAC-300-322-1HH01

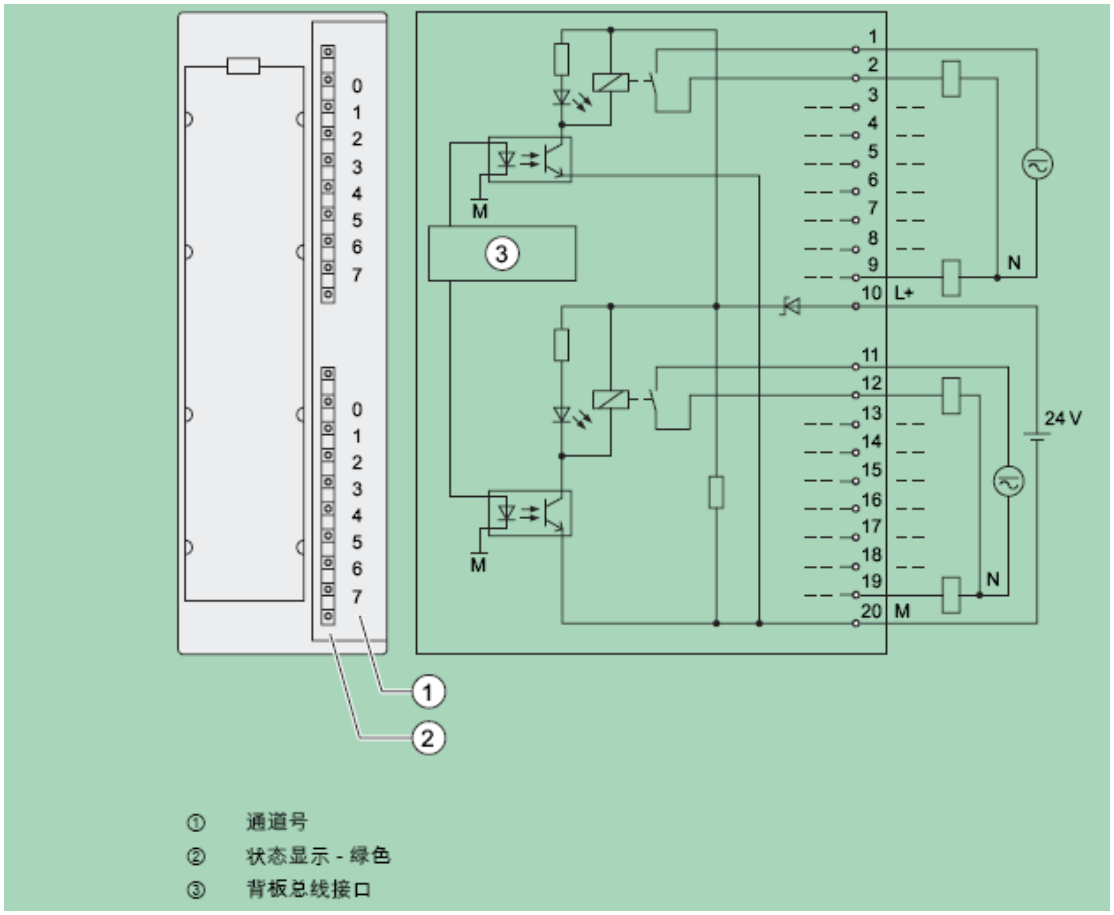
属性及技术数据

CPAC-300-322-1HH01是继电器输出模块，其主要的属性如下：

- 16 点输出，电隔离为8 组
- 额定负载电压为24 VDC 到120 VDC，48 VAC 到230 VAC
- 适用于AC/DC 电磁阀、接触器、电机起动器、FHP 电机和信号灯。

外形及接线

CPAC-300-322-1HH01 的外形及接线图如下图所示：



配置及编程

请参考 CPAC-300-322-1BF01 的配置及编程。

数字量输出：CPAC-300-322-1HH50

请参考 CPAC-300-322-1HH01。

数字量输入输出：CPAC-300-323-1BH01

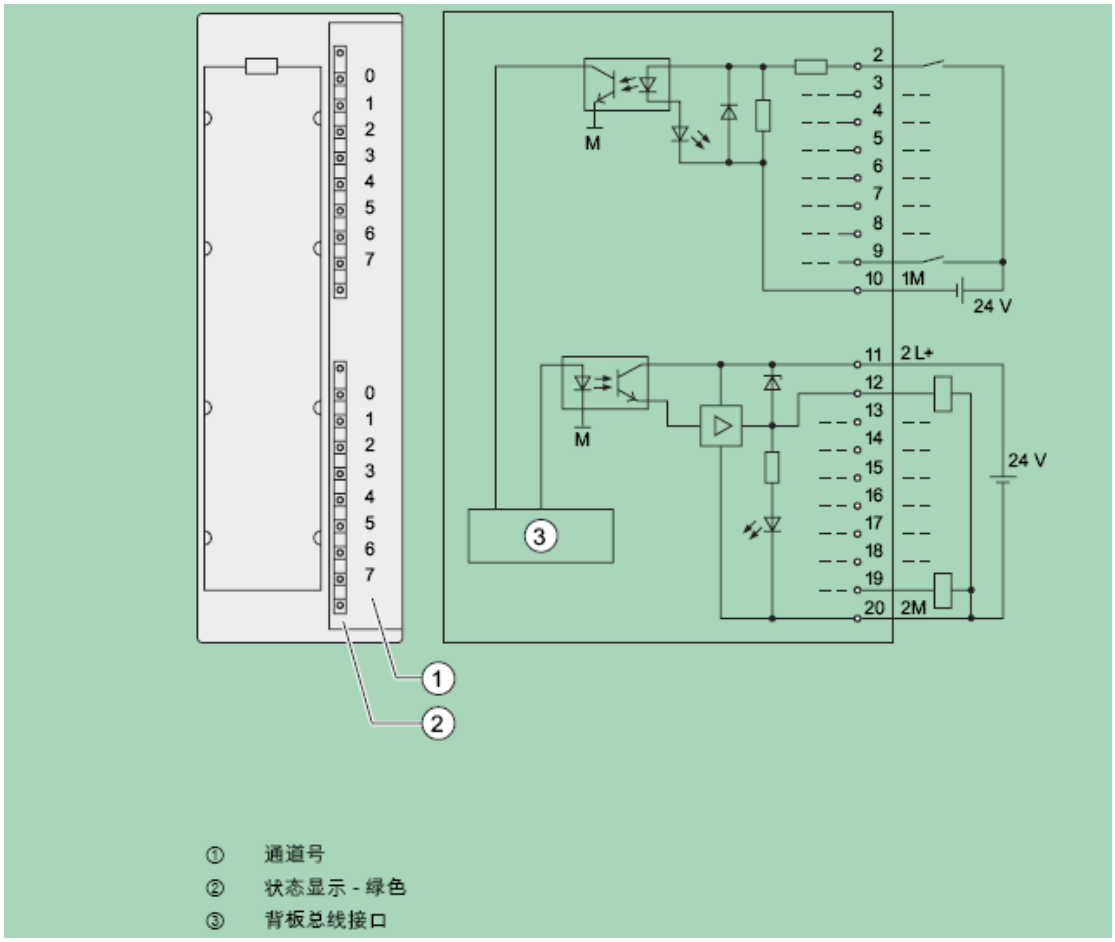
属性及技术数据

CPAC-300-323-1BH01是数字量的输出模块，其主要的属性如下：

- 8 点输入，电隔离为8 组
- 8 点输出，电隔离为8 组
- 额定输入电压24 VDC
- 额定负载电压24 VDC
- 输入适用于开关以及2/3/4 线接近开关 (BERO)
- 输出适用于电磁阀、DC 接触器和指示灯

外形及接线

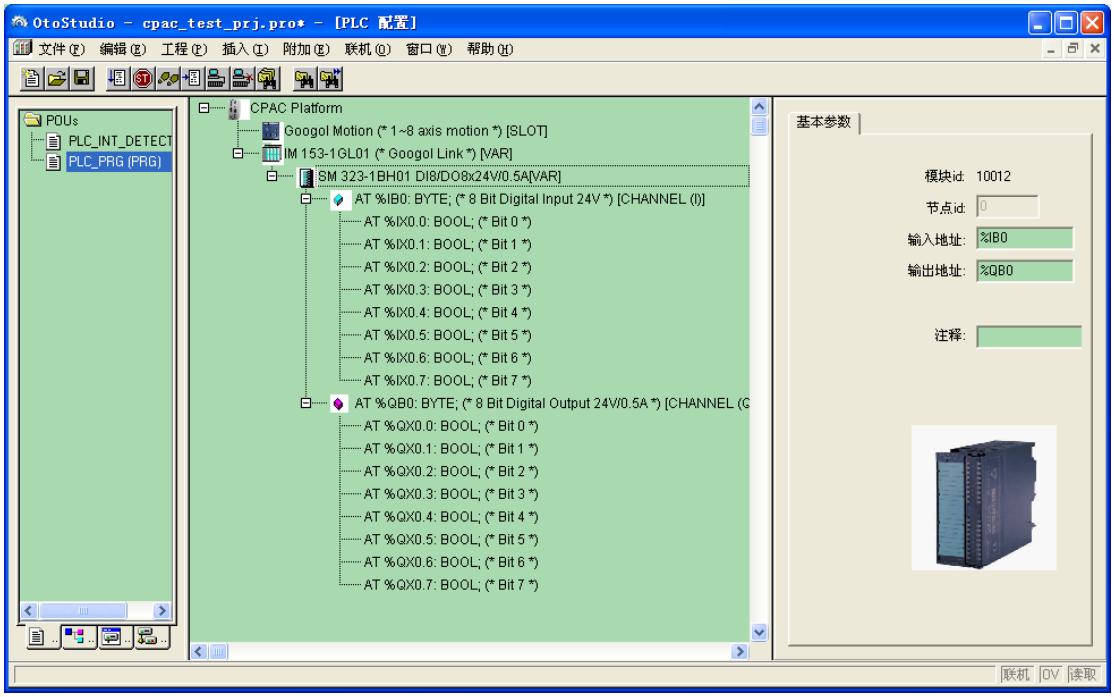
CPAC-300-323-1BH01 的外形及接线图如下图所示：



配置及编程

添加模块

在 OtoStudio 的 PLC_Configuration 中添加一个从站模块，然后在从站模块下添加 CPAC-300-323-1BH01 模块，如下图所示：



模块配置

配置参数包括输入输出地址的设置，输入地址即模块各通道输出的起始地址，输出地址即模块各通道输出的起始地址。

模块编程

完成模块的配置后，则模块的通道列表将列出该模块的每个通道。

在您的 POU 中，可以直接对整个 8 位的输出通道赋值，如 ‘%IB0 := 255;’，也可以对单个输出通道赋值，如‘%QX0.0 := 1’。

可以直接采集整个 8 位输入通道的输入值，如 ‘diValue := %IW0;’，也可以采集单个通道的输入值，如‘diBitValue := %IX0.0’。

数字量输入输出：CPAC-300-323-1BL00

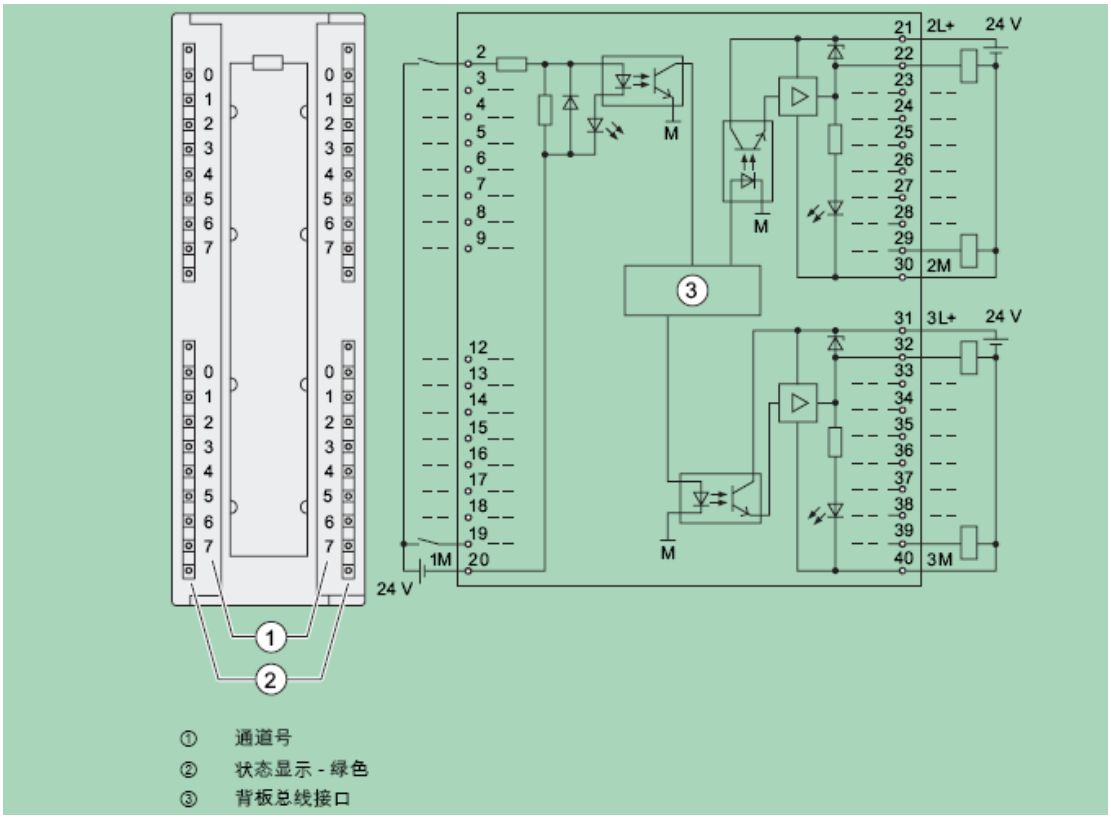
属性及技术数据

CPAC-300-323-1BL00是数字量的输出模块，其主要的属性如下：

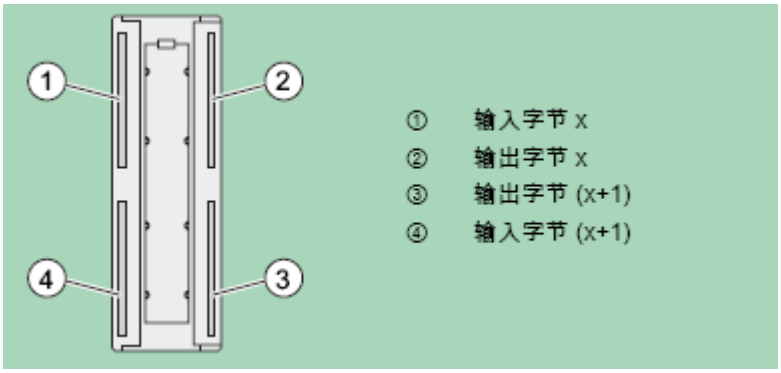
- 16 点输入，电隔离为16 组
- 16 点输出，电隔离为8 组
- 额定输入电压24 VDC
- 额定负载电压24 VDC
- 输入适用于开关以及2/3/4 线接近开关 (BERO)
- 输出适用于电磁阀、DC 接触器和指示灯

外形及接线

CPAC-300-323-1BL00 的外形及接线图如下图所示：



输入输出通道及地址分配，如下图：



配置及编程

请参考 CPAC-300-323-1BH01 模块。

附录 1：备件及附件

前连接器的说明：前连接器是配合 I/0 模块使用的必须组件，一个 I/0 模块配必须配一个前连接器，才能正常使用，下面表格说明前连接器的选型详细说明

CPAC-300-332-5HD01	配 20 针前连接器
CPAC-300-331-7KF01	配 20 针前连接器
CPAC-300-331-7KF02	配 20 针前连接器
CPAC-300-321-1BH02	配 20 针前连接器
CPAC-300-321-1BH50	配 20 针前连接器
CPAC-300-322-1BH01	配 20 针前连接器
CPAC-300-321-1BL00	配 40 针前连接器
CPAC-300-322-1BL00	配 40 针前连接器
CPAC-300-322-1BF01	配 20 针前连接器
CPAC-300-322-1HF01	配 20 针前连接器
CPAC-300-322-1HH00	配 20 针前连接器
CPAC-300-322-1HH01	配 20 针前连接器
CPAC-300-322-1HH50	配 20 针前连接器
CPAC-300-331-1KF01	配 40 针前连接器
CPAC-300-323-1BH01	配 20 针前连接器
CPAC-300-323-1BL00	配 40 针前连接器
西门子 300-332-5HF00	配 40 针前连接器
西门子 300-331-7NF00	配 40 针前连接器

GoogolLink 总线线缆说明：

I/O 模块通讯线缆	**CPAC-Cable-0015**	线缆长度为 1.5 米，其它长度需定制，最长 500 米

附录 2：服务及支持

附录 3：缩略语表

CPAC-OtoStudio 远程诊断功能手册

1 前言

CPAC 远程诊断功能，包括PLC浏览器远程操作功能。

2 使用方法

PLC浏览器的操作所需的步骤在OtoStudio工程中的描述如下。

2.1 创建一个 OtoStudio 工程

- 打开OtoStudio软件
- 创建一个新的工程通过“文件/新建”
- 选择对应的Otobox控制器，如：CPAC-X00-TPX控制器
- 创建新的POU “PLC_PRG“ (选择编程语言，如：FBD). 不需要编辑任何代码

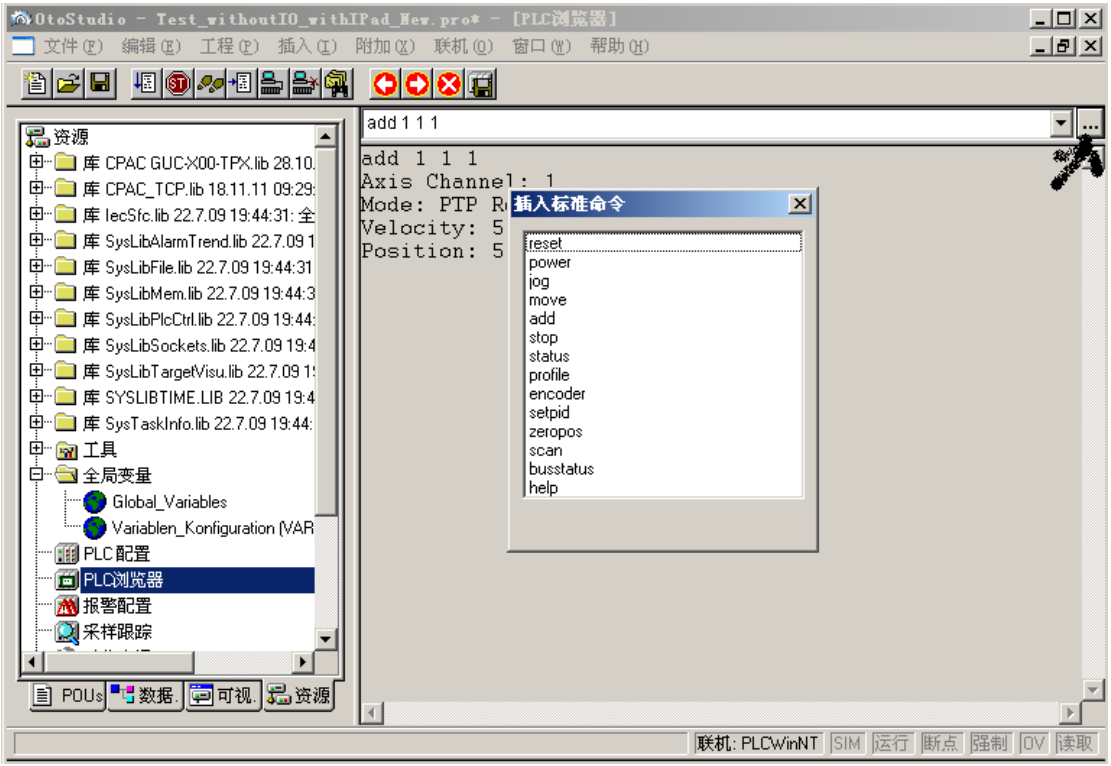
2.2 联机

选择对应的通讯参数，登入操作，建立联机。

2.3 使用 PLC 浏览器功能

在资源中，选中 PLC 浏览器，在命令行中键入 help, 即可得到所有可操作的 PLC 浏览器指令集合的帮助信息。如下图所示：

第十章 指令列表



命令	功能
reset	复位控制器
power [axis]	伺服使能指定轴
jog [axis] [vel]	JOG 模式指定轴运动
move [axis] [vel] [pos]	PTP 模式指定轴绝对运动
add [axis] [vel] [pos]	PTP 模式指定轴相对运动
gear [master] [slave] [masterratio] [slaveratio] [mastervel]	GEAR 模式指定轴运动
stop [axis]	停止指定轴运动
status [aixs]	获取指定轴状态寄存器值
profile [axis]	获取指定轴规划位置
encoder [axis]	获取指定轴编码器位置
setpid [axis] [kp] [ki] [kd]	设定指定轴 PID 参数
zeropos [axis]	清楚指定轴规划位置和编码器位置
di [ditype]	读取本地 IO 输入
do [dotype] [channel] [value]	写指定本地 IO 输出
adc [adchannel]	读取本地 adc 通道输入（仅适用于 CPAC-OTOBOX-UCT2-4XX）
Scan	扫描从站模块信息（仅在全部为 CPAC-300 系列 I/O 模块时有效）
busstatus	诊断从站在线
help	帮助文档
version	获取版本信息

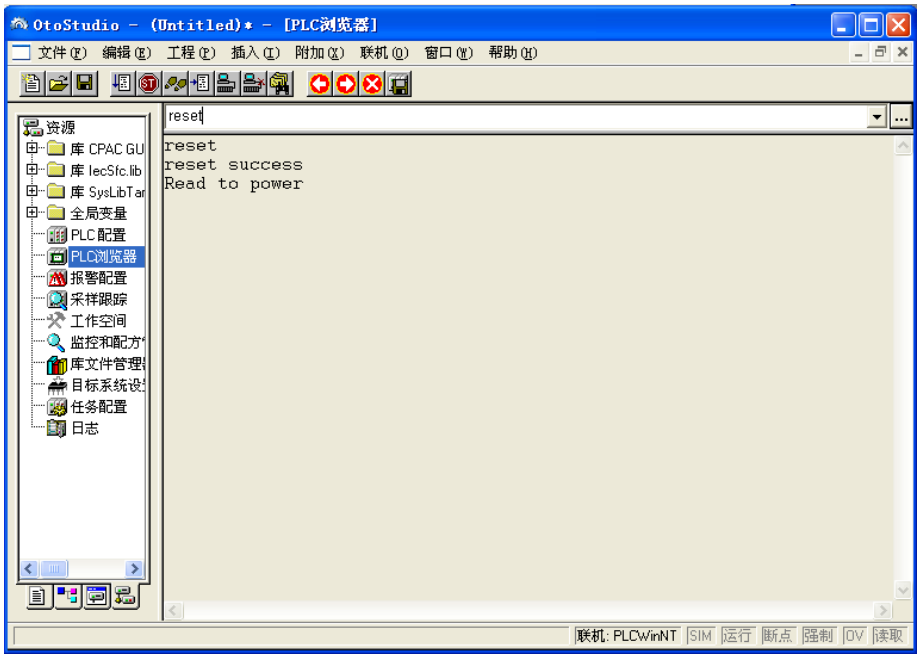
3 具体命令说明

reset

参数列表

参数	取值	意义
None		

操作实例如图：



power

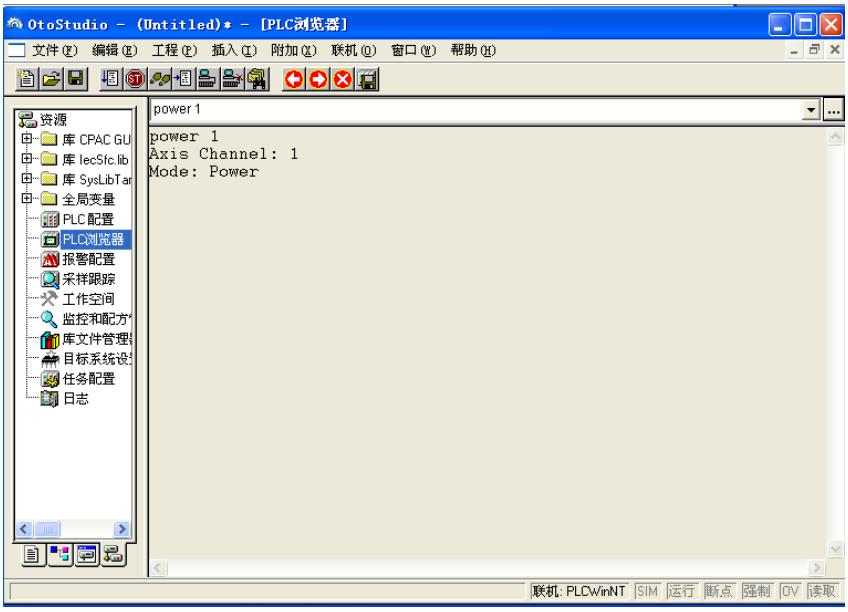
参数列表

参数	取值	意义
Axis	[1,8]	轴号

注：当没有输入参数，系统会自动使能所有轴通道。

操作实例如图：

第十章 指令列表

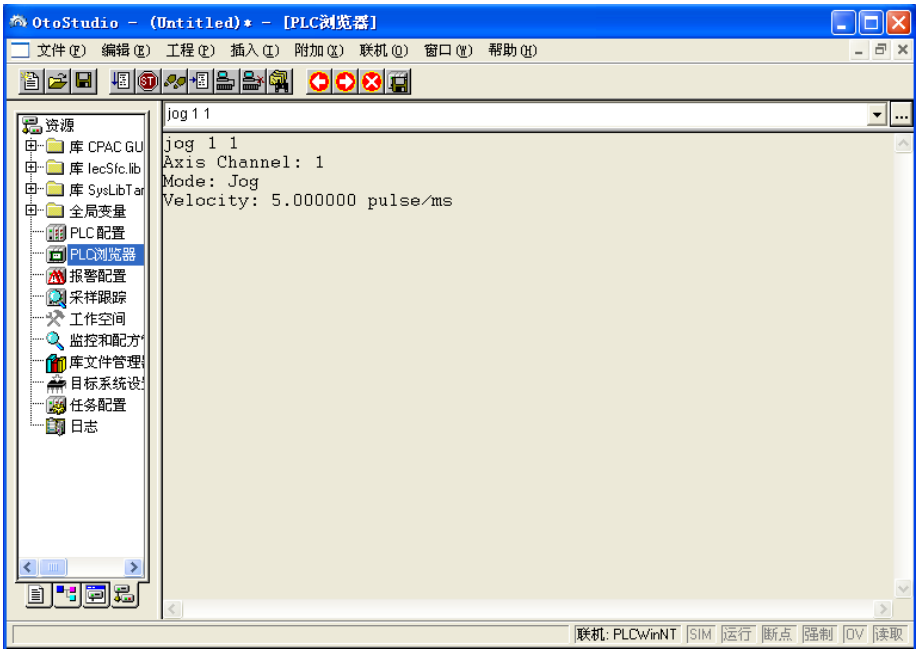


jog

参数列表

参数	取值	意义
Axis	[1,8]	轴号
vel	[-9,9]	速度等级，-9*5 pulse/ms ~ 9*5 pulse/ms

操作实例如图：



第十章 指令列表

move

参数列表

参数	取值	意义
Axis	[1,8]	轴号
vel	[-9,9]	速度等级，-9*5 pulse/ms ~ 9*5 pulse/ms
pos	[-9,9]	位置等级，-9*10000 pulse ~ 9*10000 pulse

操作实例如图：

stop

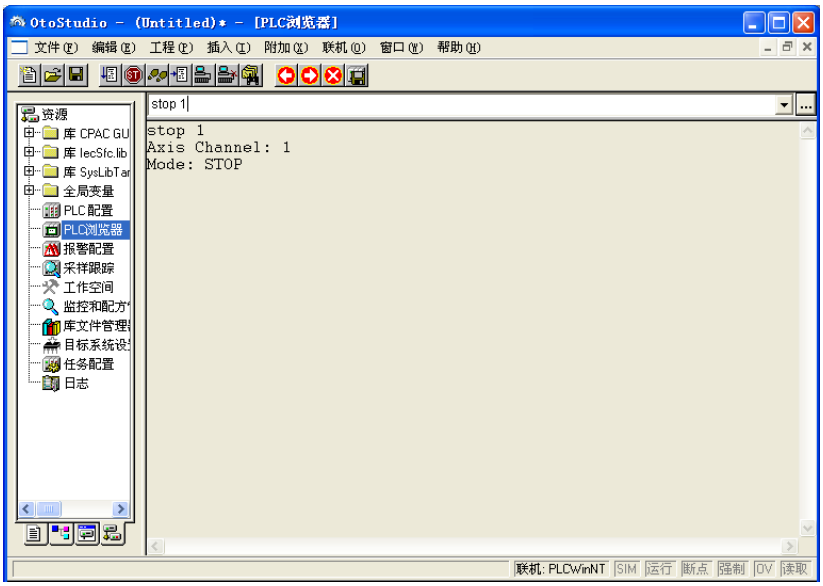
参数列表

参数	取值	意义
Axis	[1,8]	轴号

注：当没有输入参数，系统会自动停止所有轴通道。

操作实例如图：

第十章 指令列表

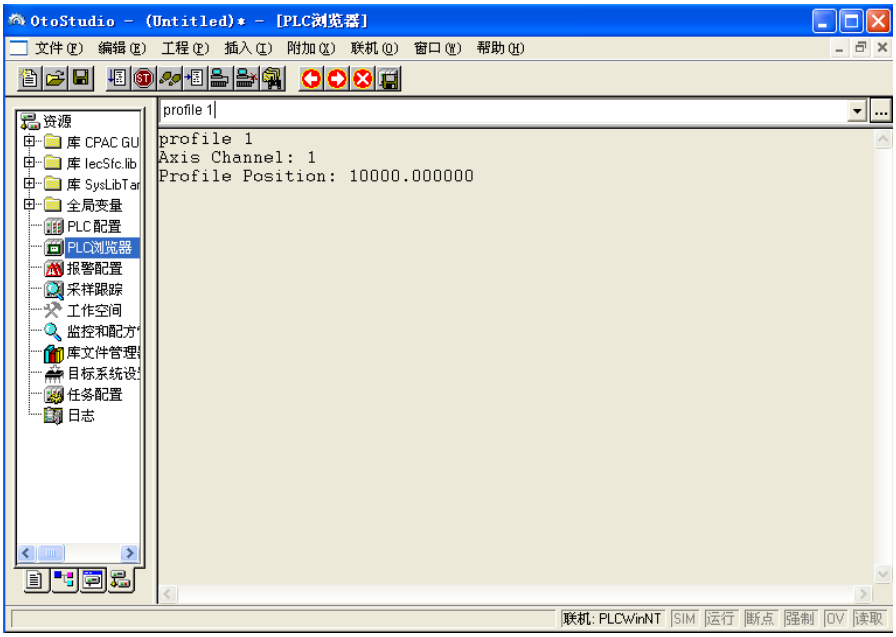


profile

参数列表

参数	取值	意义
Axis	[1,8]	轴号

操作实例如图：



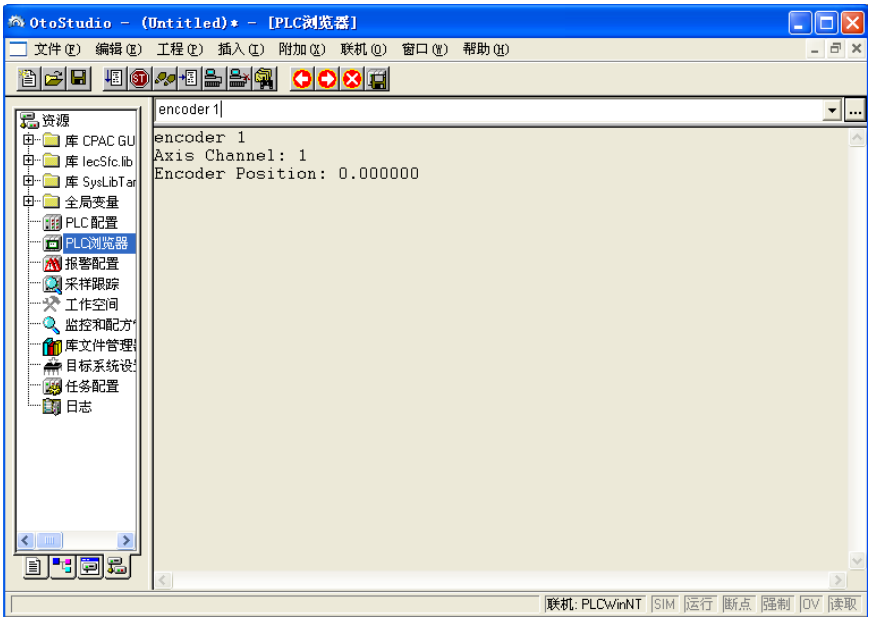
encoder

参数列表

参数	取值	意义

Axis	[1,8]	轴号

操作实例如图：



status

参数列表

参数	取值	意义
Axis	[1,8]	轴号

操作实例如图：

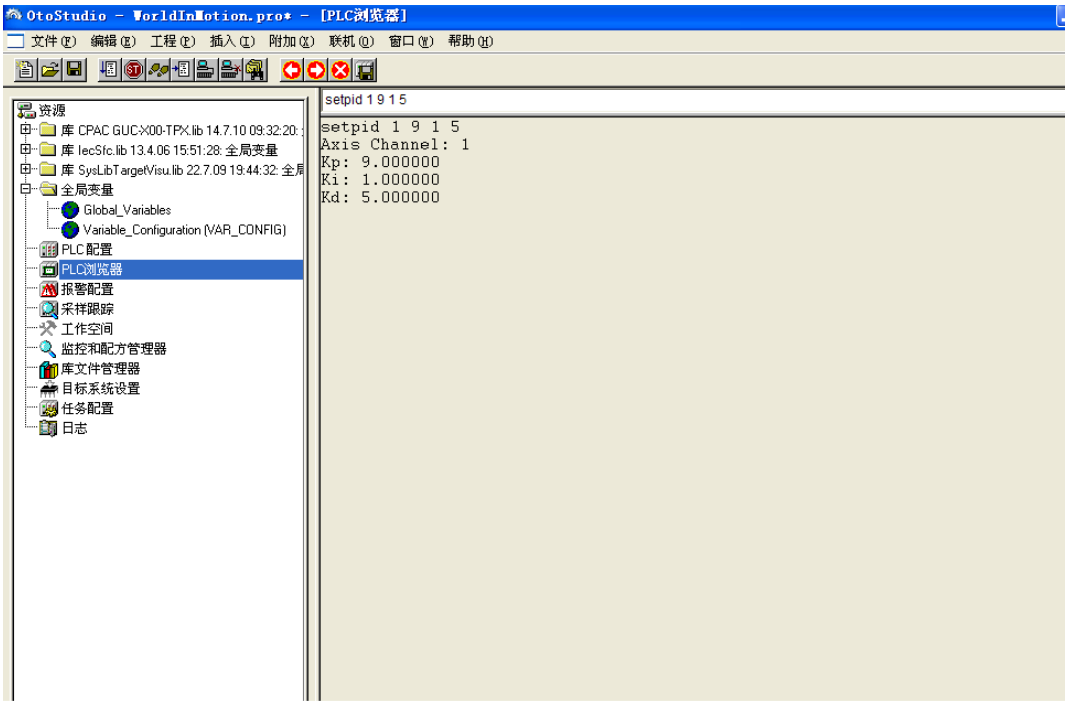
第十章 指令列表

setpid

参数列表

参数	取值	意义
Axis	[1,8]	轴号
Kp	[0,9]	比例参数
Ki	[0,9]	积分参数
Kd	[0,9]	微分参数

操作实例如图：



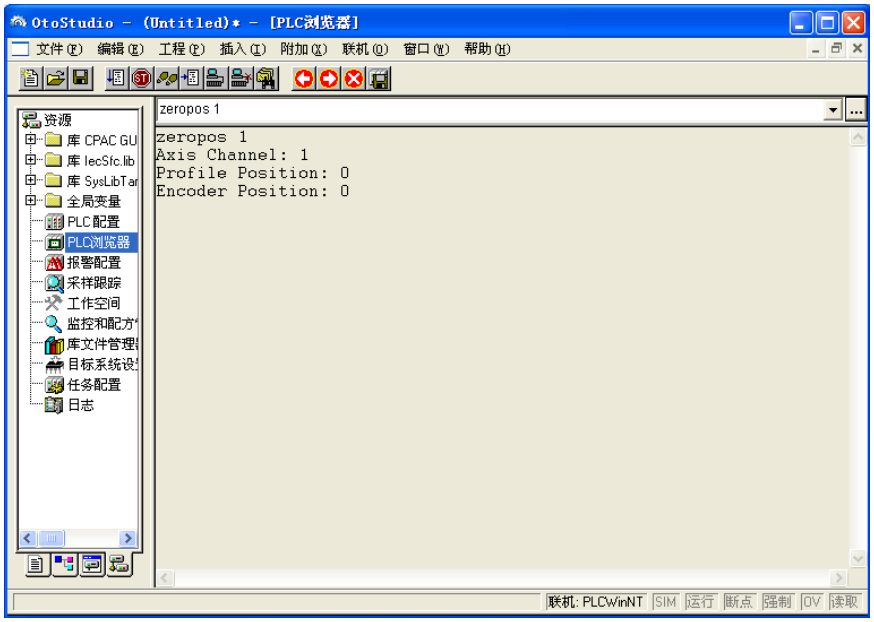
zeropos

参数列表

参数	取值	意义
Axis	[1,8]	轴号

操作实例如图：

第十章 指令列表

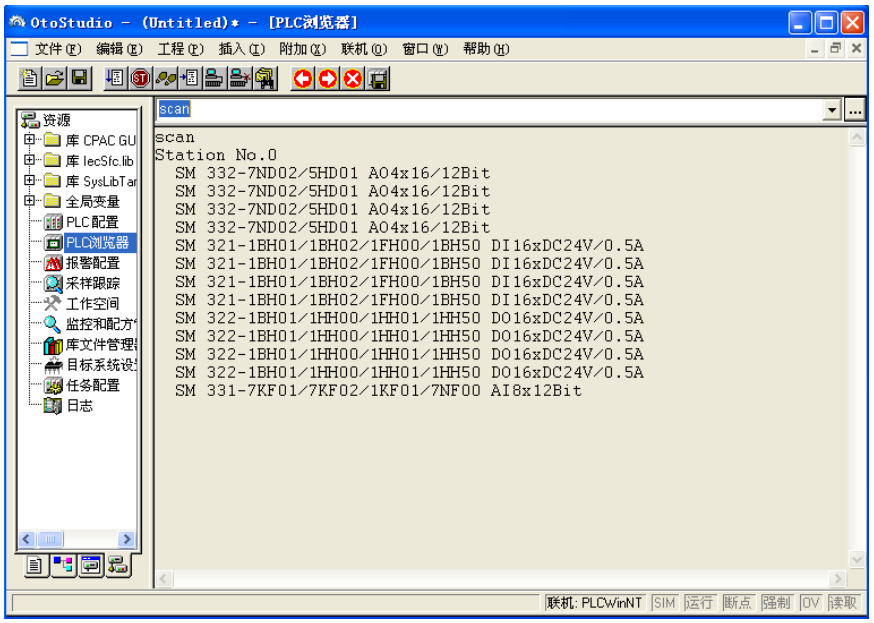


scan

参数列表

参数	取值	意义
None		

操作实例如图：



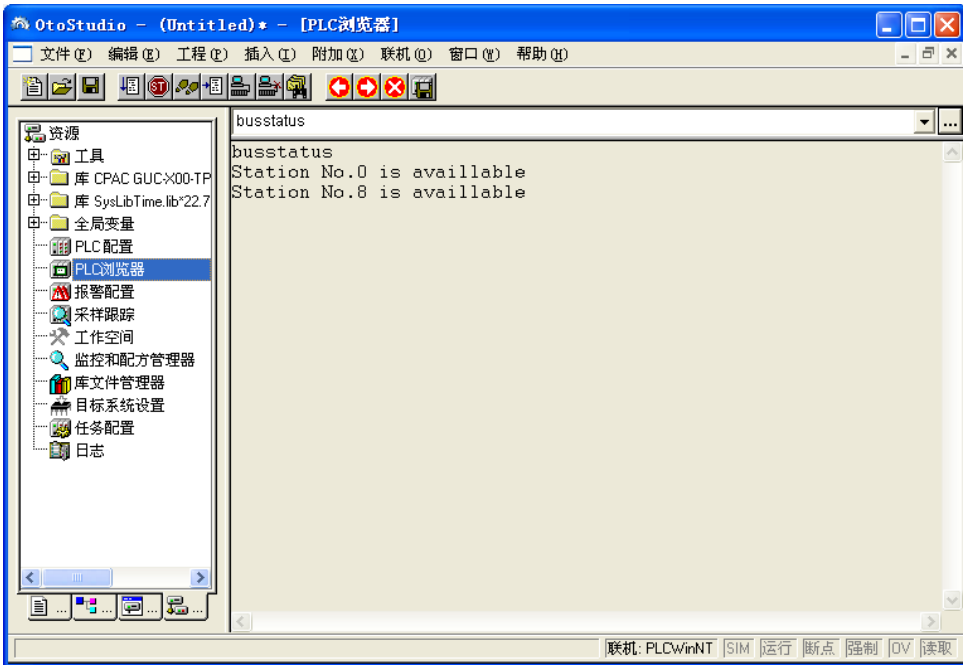
busstatus

参数列表

参数	取值	意义

None		

操作实例如图：



help

参数列表

参数	取值	意义
None		

操作实例如图：

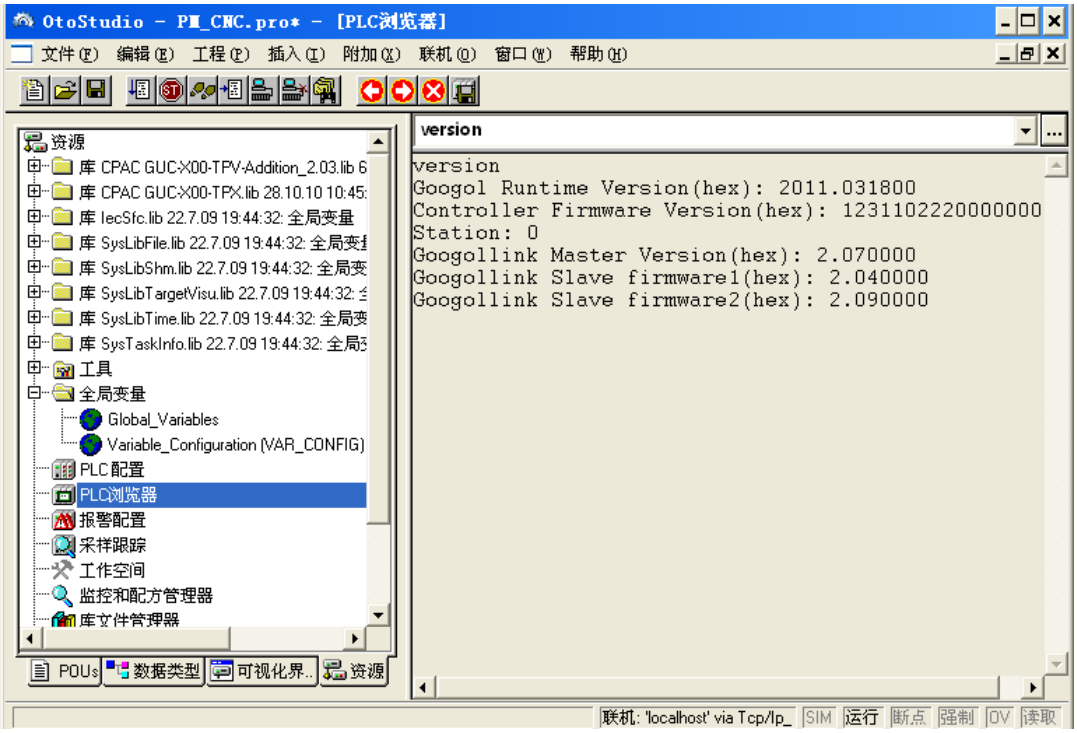
第十章 指令列表

version

参数列表

参数	取值	意义
None		

操作实例如图：



其中：

Googol Runtime Version 表示 GRT 内核版本

Controller Firmware Version 表示控制器固件版本

可选：Station X 表示连接的从站号

Googollink Master Version 表示与该从站连接的 Googollink 总线主站版本

Googollink Slave firmware1 表示与该从站 1 号 IC 固件版本

Googollink Slave firmware2 表示与该从站 2 号 IC 固件版本

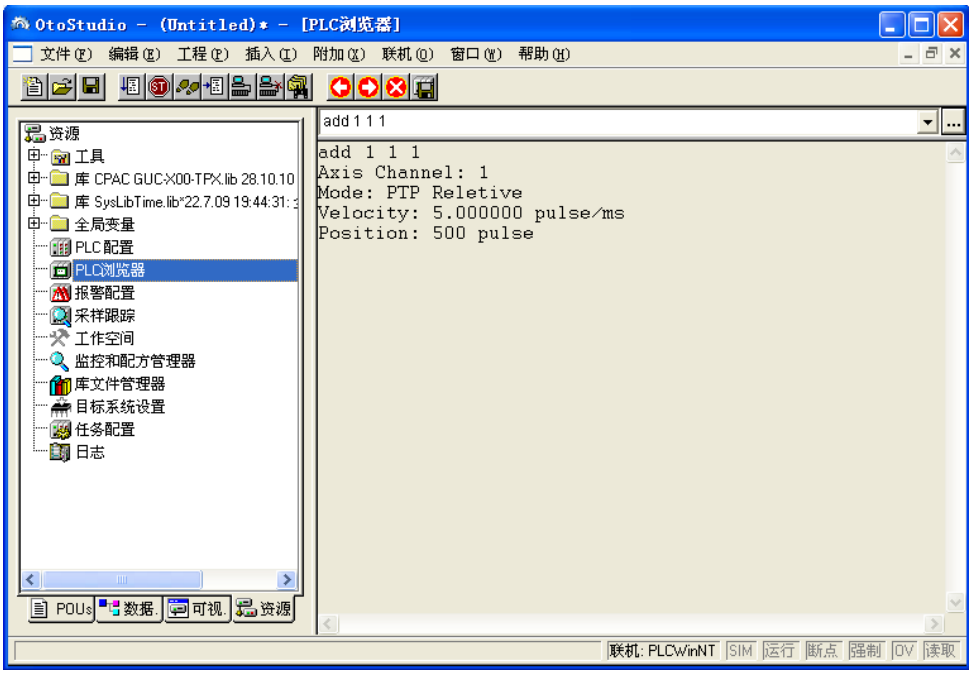
add

参数列表

参数	取值	意义
Axis	[1,8]	轴号
vel	[-9,9]	速度等级，-9*5 pulse/ms ~ 9*5 pulse/ms
pos	[-9,9]	相对位置等级，500 pulse ~ 9*500 pulse

第十章 指令列表

操作实例如图：



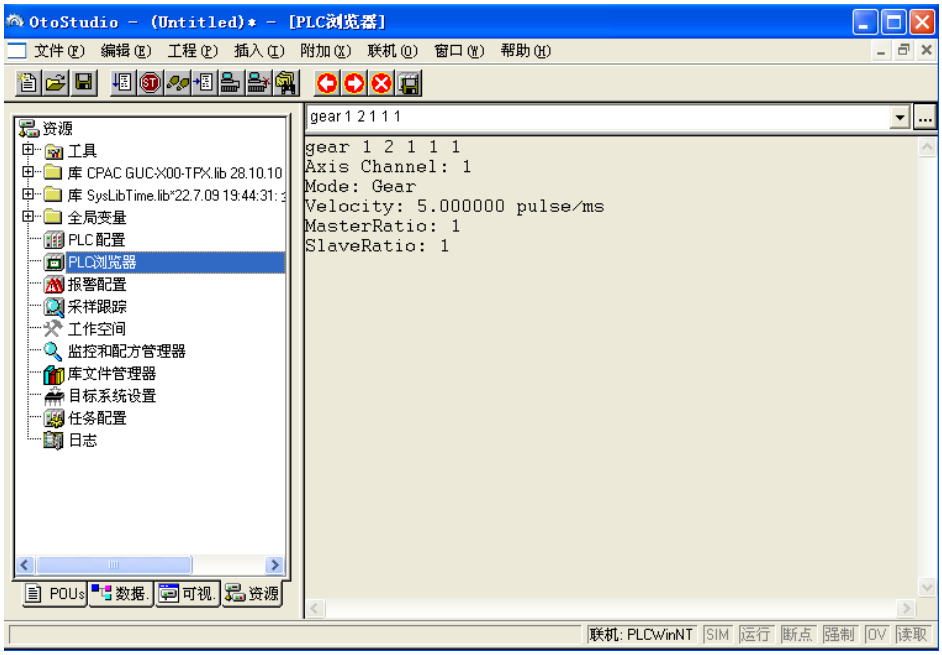
gear

参数列表

参数	取值	意义
Master	[1,8]	主轴轴号
Slave	[1,8]	从轴轴号
MasterRatio	[1,9]	电子齿轮比分母（主轴比）
SlaveRatio	[1,9]	电子齿轮比分子（从轴比）
Master Vel	[-9,9]	速度等级，-9*5 pulse/ms ~ 9*5 pulse/ms

操作实例如图：

第十章 指令列表



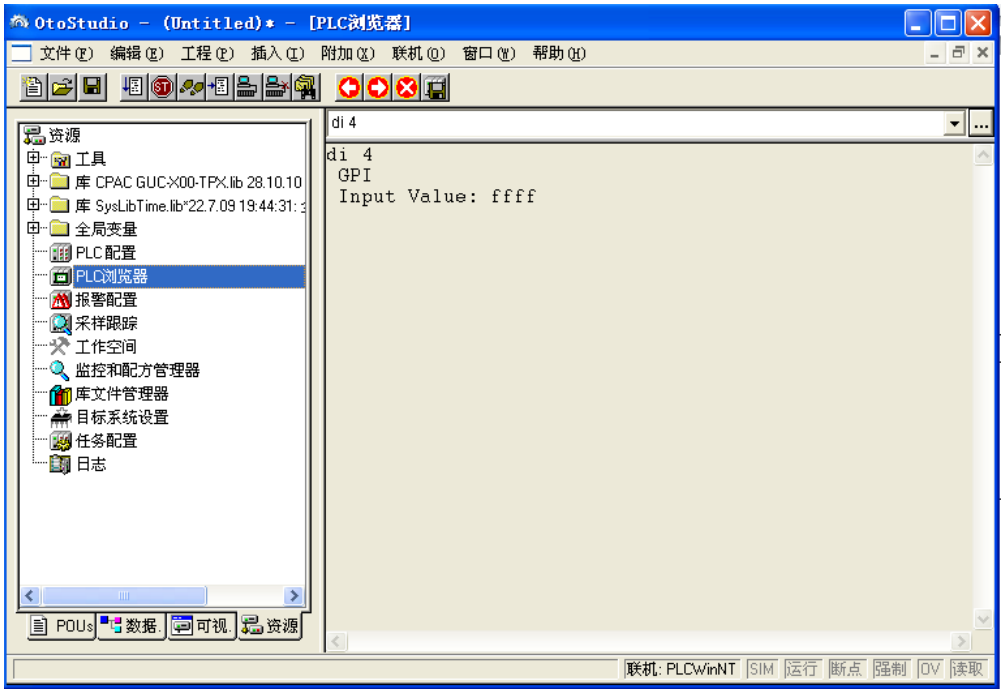
di

参数列表

参数	取值	意义
ditype	[0,4]	0：正限位 1：负限位 2：报警 3：home 4：通用输入，0 为有效输入

操作实例如图：

第十章 指令列表

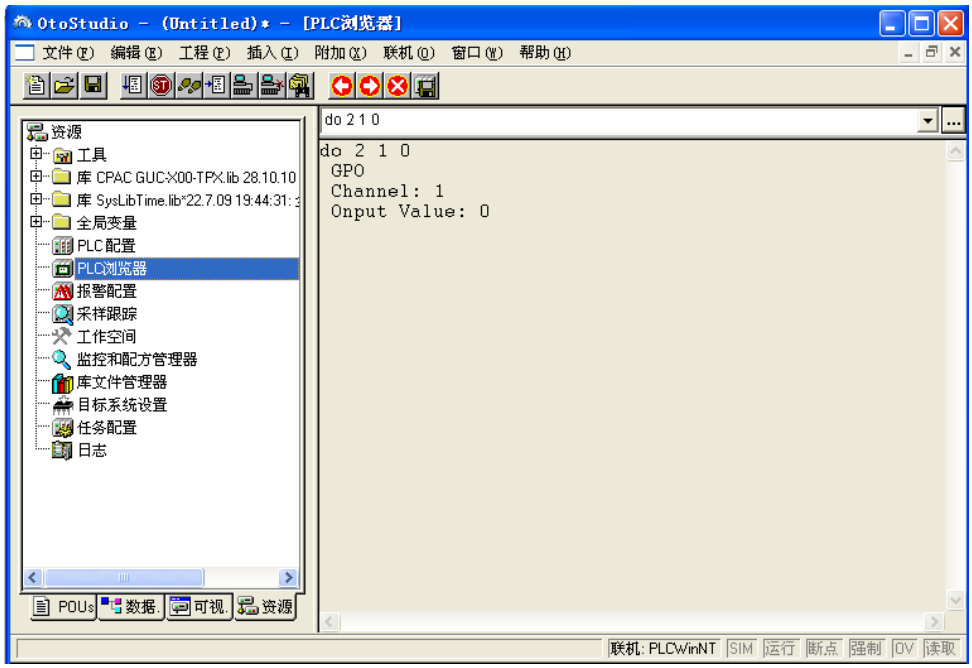


do

参数列表

参数	取值	意义
dotype	[0,4]	0：使能 1：复位轴 2：通用输出
dochannel	[0,F]	对应 DO0~DO15
dovalue	0/1	输出值，0 为有效输出

操作实例如图：



第十章 指令列表

adc

参数列表

参数	取值	意义
Adcchannel	[1,8]	获取指定通道的 ADC 电压值，-10V~10V

操作实例如图：

CPAC-OtoStudio_Util 应用库功能使用手册

1 前言

Util.lib 是基于CPAC-OtoStudio软件基础库，用于实现基本的PLC控制应用的基本功能。

该文档描述了该库提供的模型以及其函数的功能和使用方法。

2 使用方法

TCP协议库的使用方法非常简单。所需的步骤在OtoStudio工程中的描述如下。

2.1 创建一个 OtoStudio 工程

- 打开OtoStudio软件
- 创建一个新的工程通过“文件/新建”
- 选择对应的Otobox控制器，如：CPAC-X00-TPV/TPX控制器
- 创建新的POU “PLC_PRG“ (选择编程语言，如：FBD).

2.2 添加库

- 打开资源->库文件管理器
- 右键添加库：Util.LIB

2.3 功能函数列表

此库包含可以用于 BCD 转换，位/字节功能，数字辅助功能等各种块的附加集，以作模拟值处理控制器，信号生成器和功能操控器。

2.3.1 BCD 转换

BCD 格式中一个字节含有 0-99 之间的整数。每个十进位的空间占用 4 个位。10 个十进位的空间被存在 4-7 位之间。这样，BCD 格式则与十六进制表达方式是相似的。唯一不同的是 BCD 格式中只存 0-99 的数字，而十六进制字节是从 0-FF。
示例：整数 51 应转换位 BCD 格式。 5 在二进制中为 0101， 1 在二进制中为 0001，这样 BCD 字节便为 01010001，
此对应的值为$51=81。
参照:

BCD_TO_INT

此功能把 BCD 格式的一个字节转换成 INT 值：
此功能的输入值为 BYTE 类型，输出为 INT 类型。
字节应转换的地方，如不是 BCD 格式，其输出结果为 1。
ST 例子:

```
i:=BCD_TO_INT(73); (* 结果是 49 *)
k:=BCD_TO_INT(151); (* 结果是 97 *)
l:=BCD_TO_INT(15); (* 输出 -1, 因为它不是 BCD 格式 *)
```

INT_TO_BCD_
此功能把 INTEGER 值转换成 BCD 格式 的一个字节:
此功能的输入值为 INT 类型，输出为 BYTE 类型。
数字 255 出现在 INTEGER 应该转换，然而却不能转换为字节的地方。
ST 例子:

```
i:=INT_TO_BCD(49); (* 结果是 73 *)
k:=BCD_TO_INT(97); (* 结果是 151 *)
l:=BCD_TO_INT(100); (* 错误! 输出: 255 *)
```

2.3.2 位/字节功能

EXTRACT

这功能的输入是一个 DWORD X，以及一个 BYTE N。此输出为一个 BOOL 值，其中含有输入 X 的第 N 个位的内容，由此，此功能开始从零位计数。
ST 例子:

```
FLAG:=EXTRACT(X:=81, N:=4); (* 结果: TRUE, 因为 81 的二进制数为 1010001, 所以第四位是 1*)
FLAG:=EXTRACT(X:=33, N:=0); (* 结果: TRUE, 因为 33 的二进制数为 100001, 所以第零位是 1 *)
```

PACK

此功能可以回送 8 个输入位 B0，B1，…，B7，从类型 BOOL 作为一个 BYTE。功能块 UNPACK 与此功能紧密相关。

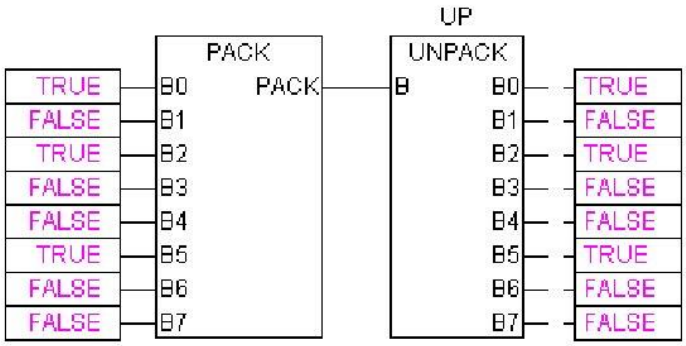
PUTBIT

此项功能的输入含有一个 DWORD X，一个 BYTE N 和一个布尔值 B。PUTBIT 把值 B 上从 X 设到第 N 位，由此从零位计数。
ST 例子:

```
A:=38;
(* 二进制数为 100110 *)
B:=PUTBIT(A,4,TRUE); (* 结果: 54 = 2#110110 *)
C:=PUTBIT(A,1,FALSE); (* 结果: 36 = 2#100100 *)
```

UNPACK

UNPACK 把输入变量 B 从类型 BYTE 转换为类型 BOOL 的 8 个输出变量 B0，…，B7，这是与 PACK 相对的。
FBD 例子: 输出:

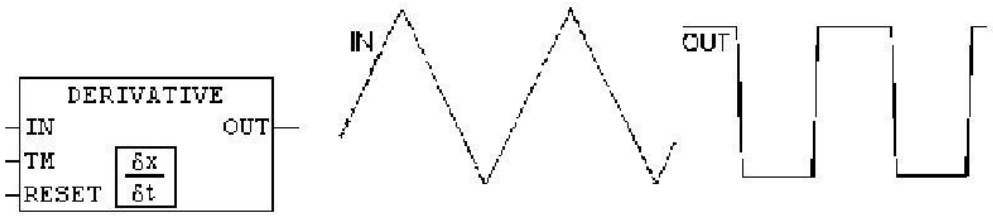


2.3.3 数学辅助功能

DERIVATIVE

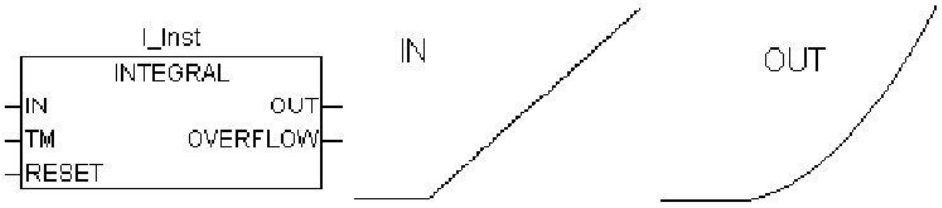
微分功能块。
功能值是使用 IN 以 REAL 类型变量表达的。TM 包含在 DWORD 中以毫秒为单位已通过的时间，类型 BOOL 的 RESET 输入可使此功能块通过释放值为 TRUE 而使其重新开始。
输出 OUT 的类型为 REAL。
为了取得最佳结果，派生约计使用最后的 4 个值，以便尽量减少输入参数不精确所产生的

错误。
FBD 下的功能块:



INTEGRAL

此功能块大致确定功能的积分。
在派生的模拟方式中，功能值的方式是使用 IN 的 REAL 变量。
TM 含有在 DWORD 中以毫秒为单位所记录的时间并且类型为 BOOL 的输入 RESET 可以使功能块以 TRUE 值开始重新启动。输出 OUT 类型为 REAL。积分为两个步进功能的约数。这些平均值显示为约定的积分数。
FBD 下的功能块: 例如: 线性功能的综合:



LIN_TRAFO

此功能块(util.lib)转换一个 REAL 值，此致在已定义有上下值限的值域内为一个相应的 REAL 值，此值则在另一个已定义的有上下值限的值域内。下面的等式便是此项转换的基础:
(IN - IN_MIN) : (IN_MAX - IN) = (OUT - OUT_MIN) : (OUT_MAX - OUT)

输入变量:

变量	数据类型	描述
IN	REAL	输入值
IN_MIN	REAL	输入值域的下限
IN_MAX	REAL	输入值域的上限
OUT_MIN	REAL	输出值域的下限
OUT_MAX	REAL	输出值域的上限

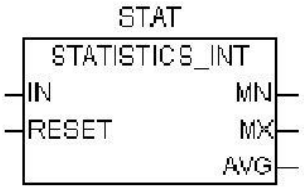
输出变量:

变量	数据类型	描述
OUT	REAL	输出值
ERROR	BOOL	产生错误：TRUE，如IN_MIN=IN_MAX, 或如IN不在输入值域内。

应用例子:
温度传感器提供电压值(输入 IN)。将此温度值转换为摄氏度值(输出值 OUT)。输入（伏特）值域已由 IN_MIN＝0 和 IN_MAX＝10 定义的上下限。其输出(摄氏度)值域则由 OUT_MIN=-20 和 OUT_MAX=40 为上下限。这样，输入为 5 伏的温度其摄氏度的结果为 10 度。

STATISTICS_INT

此功能块计算部分标准的统计学值。
输入值 IN 为 INT 类型。当布尔输入变量 RESET 为 TRUE，所有的值都会被重新初始化。输出 MN 包含 IN 中的最小值和最大值的 MX。AVG 描述的是平均值，那也是所期望的 IN 值，三个输出变量都是 INT 类型。
FBD 下的功能块:



STATISTICS_REAL

此功能块 STATISTICS 相呼应，除了其输入变量 IN，像 MN，MX，AVG 都是 REAL 类型。

VARIANCE

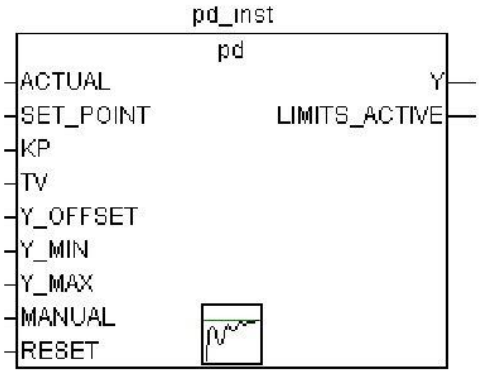
VARIANCE 计算已键入值的变异。

输入变量 IN 为 REAL 类型，RESET 为 BOOL 类型，输出 OUT 也是 REAL 类型。此块计算的输入值的变异。
VARIANCE 可以用 RESET＝TRUE 重新设置。标准偏差可以用 VARIANCE 的平方根的方法很容易地被计算出来。

2.3.4 控制器

PD

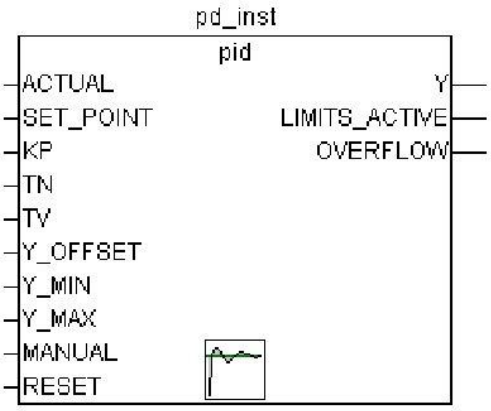
PD 控制器功能块:



ACTUAL（当前值）和 SET_POINT(期望或额定值)以及 KP，均衡系数，都是 REAL 类型的输入值。TV 为 DWORD 类型，含有以秒为单位的(如 0.5 秒代表 500 毫秒)派生活动时间。Y_OFFSET，Y_MIN 和 Y_MAX 为 REAL 类型，被用于已指定域内操纵量的转换。类型为 BOOL 类型，被用于重置控制器。
Y 也被限定在所允许的 Y_MIN 和 Y_MAX 的范围之间。如果 Y 超出此范围，作为布尔输出变量的 LIMIT_ACTIVE 便成为 TRUE。如果想让操纵量无限制，Y_MIN 和 Y_MAX 设为零。
若 MANUAL 为 TRUE，那么调节器被暂停使用，即 Y 不被更改（被控制器），如果 MANUAL 变为 FALSE，此时则会重新初始化控制器。
P 控制器可通过设置 TV，很容易使其生成一个固定的值零。

PID

PID 控制器功能块:



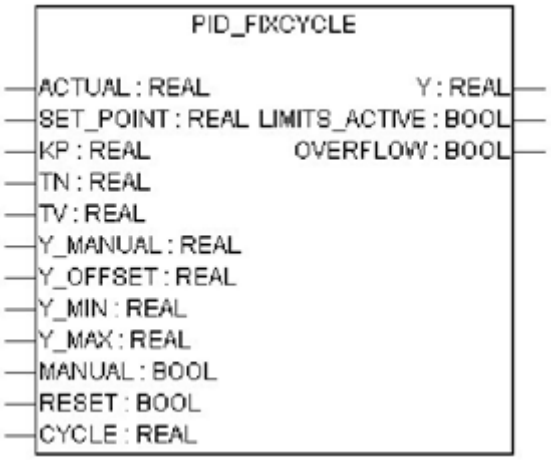
此功能不像 PD 控制器功能，它含有一个更深层次的 DWORD 输入变量 TN，以秒为单位(如 0.5 秒代表 500 毫秒)调整时间。输出操纵量（Y）还是 REAL 类型，并不像 PD 控制器那样含有一个附件的积分部分：

$Y = KP \times (D + 1/TN \int edt + TV\ dD/dt) + Y\_OFFSET$

PID 控制器可以很方便地通过将 TV 设置成零转为 PI 控制器。因为有了附加的积分部分，所以控制器参数输入出现错误就会造成溢出，条件是错误的积分变得过大。因此，为了安全起见，便会把布尔输出的变量 OVERFLOW 调用出来。此时其值应该是 TRUE。同时，控制器暂停，只有重新初始化时才会被再次激活。

PID_FIXCYCLE

PID_FIXCYCLE 控制器功能块:

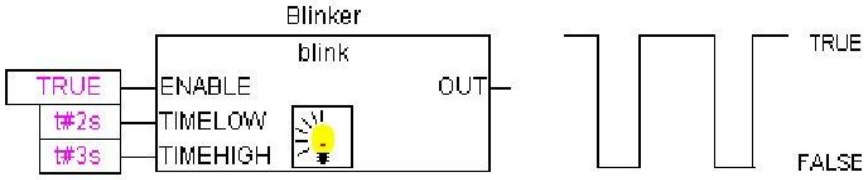


此功能块与 PID 控制器相应，除了周期不是有内部功能自动测量，而是被输入变量 CYCLE（以秒为单位）进行设定。

3.4.5 信号生成

BLINK

BLINK 功能块可生成脉冲信号。其输入变量由布尔类型的 ENABLE 以及 TIMELOW 和 TIMEHIGH 时间类型组成的。输出变量 OUT 类型为 BOOL。
如果 ENABLE 被设为 TRUE，BLINK 便开始生效，将时间输出变量 TIMEHIGH 设为 TRUE，然后将时间 TIMELOW
的变量值设为 FALSE。
CFC 例子:



FREQ_MEASURE

此功能块用于测量（平均）布尔输入信号的频率。可以详细列出应当平均为多少个时间段。一个时间段便为
两个输入信号的两个边界的上升之间的时间。

输入变量:

变量	数据类型	描述
IN	BOOL	输入信号
PERIODS	INT	时段数量，即，边界上升的间隔时间，应当计算输入信号的平均频率。可能的值为1-10。
RESET	BOOL	把所有参数重置为0

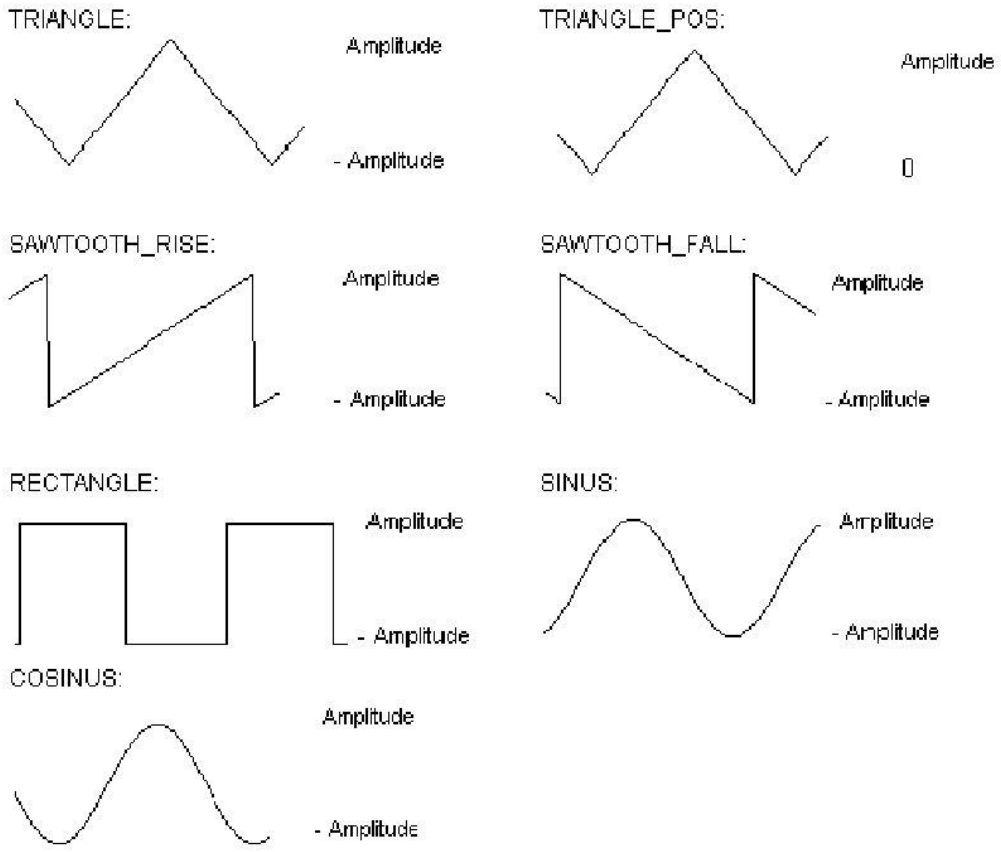
输出变量:

变量	数据类型	描述
OUT	REAL	得到频率(赫兹)
VALID	BOOL	首次测量之前为FALSE，或者如果时间长度>3*OUT（显示输入变量出了问题）

GEN

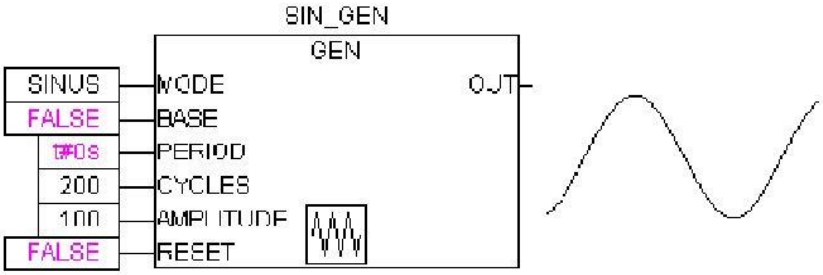
功能生成器生成典型的周期功能:
输入变量为一个由 MODE 组成，其中包括预定的计数类型 GEN_MODE，BOOL 类型的 BASE， TIME 类型的 PERIOD，组成部分中已有两个 INT 型的值 CYCLE 和 AMPLITUDE 以及布尔 RESET 输入变量。
MODE 说明所生成功能利用列举值 TRIANGLE 和 TRIANGLE_POS 传送两个三角形功

能。SAWTOOTH_RISE 为一个上升功能，而 SAWTOOTH_FALL 为一个下降的锯齿，RECTANGLE 为一个矩形信号，SINE 和 COSINE 分别为正弦和余弦:



BASE 定义周期是否真的与所定义的时间（BASE＝TRUE）有关，或者它是如何与 PERIOD 或 CYCLE 定义相应的周期时间。AMPLITUDE 从细节方式定义所生成的功能的周期变量的最大绝对值。此功能生成器在 RESET＝TRUE 时就会再次被设为 0。

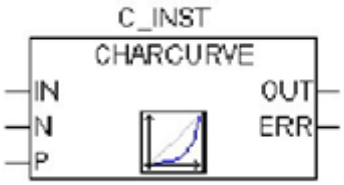
FBD 例子:



3.4.6 功能操作器

CHARCURVE

此项功能块用于在线性功能中一一表现值:



INT 类型中的 IN 要用值作操作源进行操作。BYTE N 指定点数，此点定义表现功能。此特征线随后便用类型 POINT 中的 P 在一个 ARRAY P[0……10]中生成，其中的 POINT 为一个基于 2 个 INT 值（X 和 Y）两个值的一个结构。输出的组成部分有 INT 中的 OUT。操作的值以及 BYTE ERR 此项主要在必须时显示错误。

ARRAY 中的点 P[0]…P[N-1]必须根据其 X 值进行分类，否则，ERR 便会接收到值 1。如果输入的 IN 不在 P[0].X 和 P[N-1].X，则 ERR＝2 并且含有相应限值 P[0].Y 或 P[N-1].Y。如果 N 不在所允许的 2－11 之间的值域内，那么 ERR＝4。

ST 例子:

首先，ARRAY P 必须在页首被定义:

```
VAR
...
CHARACTERISTIC_LINE:CHARCURVE;
KL:ARRAY[0..10] OF POINT:=(X:=0,Y:=0),
(X:=250,Y:=50),
(X:=500,Y:=150),(X:=750,Y:=400),7((X:=1000,Y:=1000));
COUNTER:INT;
...
END_VAR
```

再提供一个值一直增加的情况下 CHARCURVE 的例子:

```
COUNTER:=COUNTER+10;
CHARACTERISTIC_LINE(IN:=COUNTER,N:=5,P:=KL);
```

结果跟踪显示说明其结果:

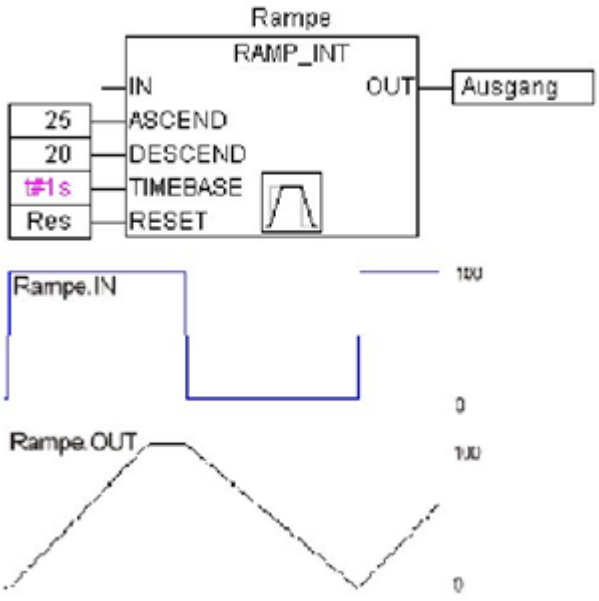
RAMP_INT

输入变量由三个 INT 值的 OUT 变量组成：IN，功能输入以及 ASCEND 和 DESCEND，每个间隔之间的最大增或减量，它又是被 TIME 类型中的 TIMEBASE 来定义的。将 RESET 设为 TRUE 会导致 RAMP_INT 被重新初始化。

INT 类型中的 OUT 输出变量中含有上升和下降的有限功能值。

把 TIMEBASE 设为 t#0s，ASCEND 和 DESCEND 虽与时间间隔无关，但是却能保持不变。

CFC 例子:



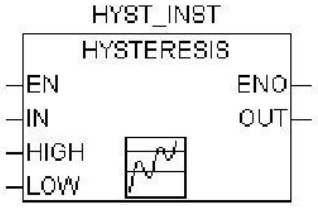
RAMP_REAL

RAMP_REAL 与 RAMP_INT 中的功能一样，不同的是，其中的输入变量为 IN，ASCEND，DESCEND 和输出变量 OUT
皆为 REAL 类型。

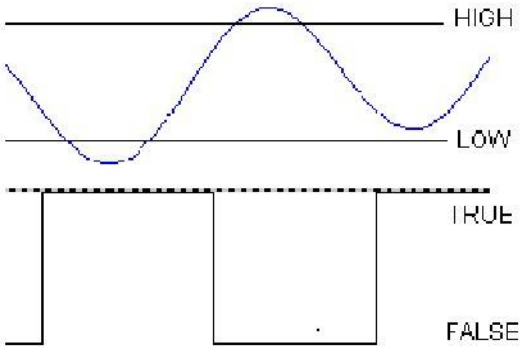
3.4.7 模拟值的处理

HYSTERESIS

此功能块的输入变量包括三个 INT 值：IN，HIGH 和 LOW。输出变量 OUT 为 BOOL 类型。



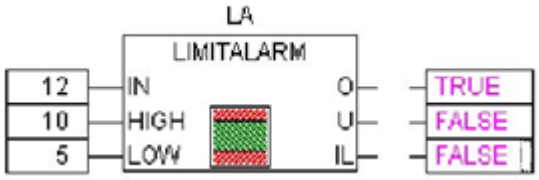
如果 IN 值低于 LOW 的限值，OUT 则变成 TRUE。若 IN 超过了上限 HIGH 的值，便会显示 FALSE 的结果。
说明示例：



LIMITALARM

限制警告

此项功能块说明输入变量的值是否在设定范围内，如果不在范围内便超出了限制范围。输入值 IN，HIGH 和 LOW 都是 INT 类型，而输出变量 O，U 和 IL 则都是 BOOL 类型。



CPAC-OtoStudio 系统库功能使用手册

1 前言

系统库函数SysLibxxx.lib是OtoStudio上针对OEM用户访问底层操作的系统函数功能，在仿真运行下无效。只能在硬件上下载运行。

该文档描述了该库提供的功能以及其函数的功能和使用方法。

系统库预览下表

库	功能
SysLibCallback.lib	为实时事件激活已定义的回调功能
SysLibCom.lib	目标系统的RS232串行通讯
SysLibDir.lib	在目标系统上管理文件目录系统
SysLibEvent.lib	同步控制两个 IEC 任务的处理
SysLibFile.lib	在目标系统中操作一个文件系统
SysLibFileAsync.lib	在目标系统中操作一个文件系统（异步访问）
SysLibGetAddress.lib	回调数据段的起始地址功能
SysLibIECTasks.lib	适合于IEC步的特殊功能（参见SysLibTasks.lib）
SysLibInt.lib	在一个功能中申请中断响应
SysLibMem.lib	内存管理
SysLibOS.lib	调用OS级别应用程序
SysLibPlcCtrl.lib	控制 PLC 中的保留变量和看门狗
SysLibProjectInfo.lib	读取工程信息
SysLibRtc.lib	读取计算机的实时时钟
SysLibSem.lib	为任务同步创建和使用旗语
SysLibShm.lib	创建并访问一个共用存储器
SysLibSockets.lib	同步访问通过 TCP/IP 和 UDP 通讯的接口
SysLibSocketsAsync.lib	异步访问通过 TCP/IP 和 UDP 通讯的接口
SysLibStr.lib	字符串处理功能（如复制，比较）
SysLibTasks.lib	任务操作(参见 SysLibIECTasks.lib)
SysLibTime.lib	实时获取本地系统时钟

2 各个系统库及使用方法

2.1 SysLibCallback.lib

这个库提供功能函数 SysCallbackRegister 和 SysCallbackUnregister，分别用来激活 callback 功能和 runtime 事件的定义。

两个功能函数都是布尔型的，当必需的 callback 功能被成功的注册或注消时，它们会分别返回 TRUE，且整个过程都是实时的。

这callback功能的原型必须如下所示：

```
FUNCTION Callback : DWORD
VAR_INPUT
dwEvent:DWORD; // 事件
dwFilter:DWORD; // 过滤
dwOwner:DWORD; // 来源
END_VAR
```

当功能函数被用来注册或注消功能调用时，都将采用下面的参数接口描述：

输入变量	数据类型	描述
iPOUIndex	INT	要用来注册/注消而调用 callback 功能的 POU 的指针，这个指针必须在以前通过帮助操作 INDEXOF(<function name>)命令已经获得。
Event	RTS_EVENT	被 callback 功能呼叫的 runtime 事件，其详细说明在 RTS_EVENT 的列举中。

RTS_EVENT

RTS_EVENT 列举定义如下：

```
TYPE RTS_EVENT :
(
  EVENT_ALL,

  (* 一般事件 *)
  EVENT_START,
  EVENT_STOP,
  EVENT_BEFORE_RESET,
  EVENT_AFTER_RESET,
  EVENT_SHUTDOWN,
```

```
  (* 实时运行产生的例外 *)

  EVENT_EXCPT_CYCLETIME_OVERFLOW,
  EVENT_EXCPT_WATCHDOG,
  EVENT_EXCPT_HARDWARE_WATCHDOG,
  EVENT_EXCPT_FIELDBUS,
  EVENT_EXCPT_IOUPDATE);

END_TYPE
```

详细的使用方法请参考示例程序SysLibCallbackTest.pro

2.2 SysLibCom.lib

此库函数支持的是与目标系统的RS232串口通讯。要是目标系统可以支持串口通讯，则下列库中的函数将能实现串口的打开和关闭，以及通过串口进行数据的读取和写入。（整个过程也都是
实时的）

SysComOpen
SysComSetSettings
SysComSetSettingsEx
SysComClose
SysComRead
SysComWrite

SysComOpen

此功能用来打开串口。

这个功能块将为串口返回一个handle，当调用库中其它的的功能块时可通过这个handle来进行传递。要是串口不能被打开，则返回的handle值为0xFFFFFFFF。

输入变量	数据类型	描述
Port	PORTS	指定要打开的串口（COM1，...）；端口号数在 PORTS 列举后面。（参考SysSetSettings）

SysComSetSettings

此功能块可以提供像波特率、停止位、奇偶性、功能超时时间、缓冲器大小和扫描时间等的
串口设置。参数值是指向 COMSETTINGS 的指针，COMSETTINGS 是已经定义好的结构体。

当参数被正确成功设置后，功能块将返回 TRUE，否则返回 FALSE。

输入变量	数据类型	描述
ComSettings	POINTER TO COMSETTINGS	指向机构体 COMSETTINGS 的指针。可通过 ADR 操作命令获得。（参考下面的例子）
dwHandle	DWORD	通过 SysComOpen 已获得的串口 handle

COMSETTINGS

COMSETTINGS 结构体也是库函数的一部分，详细说明如下：

```
TYPE COMSETTINGS :
STRUCT
    Port:PORTS;              (*串口号，参考下面的 PORTS 列举*)
    dwBaudRate:DWORD;        (* 4800, 9600, 19200, 38400, 57600, 115200 *)
    byStopBits:BYTE;         (*0 = 1 停止位，1 = 1.5 停止位，2 = 2 停止位*)
    byParity:BYTE;           (*0 = 无奇偶校验，1 = 奇校验，2 = 偶校验*)
    dwTimeout:DWORD;         (*以 ms 计算的连接超时，默认是 0*)
    dwBufferSize:DWORD;      (*内在设备缓冲器大小，默认是 0*)
    dwScan:DWORD;            (*串口连接时轮流检测时间，应该设置成 0*)
END_STRUCT
END_TYPE
```

SysComSetSettingsEx

此功能块的参数值也是指向 COMSETTINGS 的指针，用来设置所有和串口通讯相关的参数。不仅上面提到的参数可以设置 ,而且溢出控制和特性大小值也可以在这个功能块中得到设置。这是通过向 COMSETTINGSEX 结构体中添加相应设置项完成的。

若参数被成功设置，则功能块会返回 TRUE 值，否则返回 FALSE。在串口打开时设置是否可以多次更改取决于硬件的选择。当设置更改时可能需要对串口进行重新打开才能使新的设置生效。

输入变量	数据类型	描述
ComSettingsEx	POINTER TO COMSETTINGSEX	指向机构体 COMSETTINGSEX 的指针。可通过 ADR 操作命令获得。（参考下面的例子）
dwHandle	DWORD	通过 SysComOpen 已获得的串口 handle

COMSETTINGSEX

COMSETTINGSEX 结构体：

```
TYPE COMSETTINGSEX :
STRUCT
    Size:INT; (*结构体的字节数，用 sizeof（）操作命令填写，可以向后
兼容*)
    Port:PORTS; (*COM1,COM2...*)
    dwBaudRate:DWORD;        (* 4800, 9600, 19200, 38400, 57600, 115200 *)
    byStopBits:BYTE;         (*0 = 1 停止位，1 = 1.5 停止位，2 = 2 停止位*)
    byParity:BYTE;                (*0 = 无奇偶校验，1 = 奇校验，2 = 偶校验*)
    dwTimeout:DWORD;         (*以 ms 计算的连接超时，默认是0 *)
    dwBufferSize:DWORD;      (*内在设备缓冲器大小，默认是0 *)
    dwScan:DWORD;            (*串口连接时轮流检测时间，应该设置成 0，除非硬件的说明
文档要求更改*)

    cByteSize : BYTE; (*4...8，用位描述的特性大小*)
    fOutxCtsFlow :  BOOL; (*指出输出流控制的CTS信号是否被监听。如果成员值是TRUE
而且CTS设置位中断，则输出停止直到下个CTS发出*)
            fDtrControl : BYTE; (* 0：当设备开放时使 DTR 线失效。
```

1：当设备开放时使 DTR线有效
2： 授权 DTR 握手*)
fDsrSensitivity : BOOL;(* 指出通讯驱动是否对 DSR 信号状态敏感。如果这个参数设置为 TRUE，则驱动器忽略收到的任何字节，直到 DSR 调制解调器输入高电平信号。*)
fRtsControl : BYTE;(* 0：当设备开放时使 RTS 线失效。1：当设备开放时使 RTS线有效。2：授权 RTS 握手。驱动器在输入缓冲器小于总缓冲器一半时增强 RTS 线而在其大于 3/4 总缓冲器时减弱 RTS线。3：指定传输字节可用时 RTS 线为高，当所有的缓存器数据全部发送完时 RTS 线为低。*)
fOutxDsrFlow : BOOL;(* 指出输出流控制的 DSR 信号是否被监听。如果成员值是 TRUE 而且 DSR 设置位中断，则输出停止直到下个 DSR 发出。*)
END_STRUCT
END_TYPE

完成硬件参数设置的举例：

```
dwHandle: DWORD;

pt_comsettingsex:COMSETTINGSEX:=(Port:=COM1,

dwBaudRate:=38400,

byStopBits:=0,

dwTimeout:=5000,

cByteSize:=7,

byParity := 2,

fOutxCtsFlow := FALSE,

fOutxDsrFlow:=TRUE,

DtrControl := 2,

fRtsControl := 2);
```

实现部分:

```
pt_comsettingsex.Size := sizeof(pt_comsettingsex);
SysComSetSettingsEx(dwHandle := Handle, ComSettingsExt :=ADR(pt_comsettingsex));
```

在这里，dwHandle是通过调用SysComOpen(COM1)获取的返回值.

SysComClose

这个功能块的输出类型是布尔型，用来实现关闭 COM 口。为了实现这个功能它的输入参数变、量必须是以前用 SysComOpen 功能块获得的串口 handle 值。操作成功返回 TRUE 值，否则返回 FALSE。

输入变量	数据类型	描述
dwHandle	DWORD	通过 SysComOpen 已获得的串口 handle

SysComWrite

这个功能块以 DWORD 类型向前面 SysComOpen 功能块指定的串口写入数据。除了前

面提到的handle 为输入外，要输入字符串的地址及字符串的大小都要为输入对象，还有输入超时也是功能块的输入值。
功能块将返回实际写入的字节数。

输入变量	数据类型	描述
dwHandle	DWORD	通过 SysComOpen 已获得的串口 handle
dwBufferAddress	DWORD	要被写入串口的字符串的地址，可通过 ADR 操作命令来获得
dwBytesToWrite	DWORD	要被写入的字节数
dwTimeout	DWORD	这个时间后功能块将返回到最新值，单位是ms

SysComRead

这个功能块以 DWORD 类型从串口读取数据，输入参数有前面 SysComOpen 块获得的handle值和期望读取的字节数以及功能块的超时时间，除此之外还有读取的数据要被拷贝到的位置的地址，这个地址是要在前面已经定义好的。
功能块将返回实际读取的字节数。

输入变量	数据类型	描述
dwHandle	DWORD	通过 SysComOpen 已获得的串口 handle
dwBufferAddress	DWORD	读取的字符串要被拷贝出来的地址 ,可通过 ADR 操作命令来获得
dwBytesToRead	DWORD	要被读取的字节数
dwTimeout	DWORD	这个时间后功能块将返回到最新值，单位是ms

详细的使用方法请参考示例程序SysLibComTest.pro

2.3 SysLibDir.lib

此库支持 runtime 系统，可用下面的功能块处理目标系统上的文件系统。可以读取修改目录名。且操作是实时的。

SysDirCreate
SysDirOpen
SysDirRead
SysDirRemove
SysDirRename

SysDirCreate

这个功能块（BOOL 型）可以创建一个新的目录。如果目录被成功创建则此功能块返回 TRUE 值，否则返回 FALSE。

输入变量	数据类型	描述
stName	STRING	目录名称

SysDirOpen

这个功能块（DWORD 型）可以打开一个已有的目录，以便通过 SysDirRead 功能块读取目录内的条目（文件、子目录）。

此功能块返回一个 handle，SysDirRead 功能块将以这个 handle 为输入。

输入变量	数据类型	描述
stDirectory	STRING	目录名称

SysDirRead

这个功能块（UDINT 型）可以读取目录下的条目。每次调用这个功能块，都可以读取目录下的一个条目，只要返回 1，则说明目录下还有其它条目。要想读取目录下所有的条目必须反复调用此功能块知道其返回 0 为止。

关于条目的细节信息将在 DIRECTORY_INFO 结构体中写出。

输入变量	数据类型	描述
hDir	DWORD	目录的 handle，也就是上面 SysDirOpen 功能块的返回值
stDirEntry	STRING	目录下条目的名字，可以是文件也可以是子目录名，最大 80 个字符
pDirInfo	POINTER TO DIRECTORY_INFO	指向 DIRECTOPY_INFO 结构体的指针，此结构体将加载要读取条目的详细信息，如果不需要读取在这里可以输入0。

SysDirRemove

这个功能块（BOOL 型）可以删除一个已有的目录。如果目录被成功删除则返回 TRUE 值，否则返回 FALSE。

输入变量	数据类型	描述
stName	STRING	将要删除的目录名称

SysDirRename

这个功能块（BOOL 型）可以重命名一个已有的目录。如果目录被成功重命名则返回 TRUE 值，否则返回 FALSE。

输入变量	数据类型	描述
stOldName	STRING	旧的目录名称
stNewName	STRING	新的目录名称

DIRECTORY_INFO

这个结构体包含了通过 SysDirRead 功能块读取的目录的目录信息。组成如下：

变量	数据类型	描述
ftTime	DIRFILETIME	包括创建日期、修改日期、访问日期等
dwSize	DWORD	目录下条目的大小
bDirectory	BOOL	如果确实是目录则返回 TRUE；若是条目则返回 FALSE。

DIRFILETIME

这个结构体包含了对目录访问的日期信息，它是通过被DIRECTORY_INFO结构体调用所应用的。组成如下：

变量	数据类型	描述
dtCreation	DT	创建日期
dtLastAccess	DT	最新一次访问的日期
dtLastModification	DT	最近一次修改的日期

详细的使用方法请参考示例程序SysLibDirTest.pro

2.4 SysLibFile.lib

这个库支持处理目标计算机上的文件系统。如果目标对象支持，此库可以实现对文件的打开、关闭、删除、重命名、读和写等操作。更多的功能像获得文件的大小以及最后一次的访问（读取、修改）等也是可以应用的。执行过程也是实时的。
功能块如下：

SysFileOpen
SysFileWrite
SysFileRead
SysFileClose
SysFileDelete
SysFileCopy
SysFileEOF
SysFileGetPos
SysFileGetSize
SysFileGetTime
SysFileRename
SysFileSetPos

SysFileOpen

这个功能（DWORD 型）可以用来打开已有的文件或创建一个新文件。它的返回值是一个文件编号，这个编号将在其它功能块如 SysFileWrite、SysFileRead、SysFileClose 中作为一个输入变量。要是发生错误将返回 0。

输入变量	数据类型	描述
FileName	STRING	文件名称
Mode	STRING	访问模式： w 写模式（可以创建或更新文件） r 读模式(文件以只读形式打开，要是文件不存在，则返回一个错误） rw 读写模式(文件可以被更新，要是文件不存在，则返回一个错误） a 附加模式（文件以“w”模式打开，更新的内容附加在文件的结尾）

SysFileClose

这个功能(BOOL 型)用来关闭一个前面用 SysFileOpen 功能块打开的文件，返回值是 1(正确)或 0(错误)。

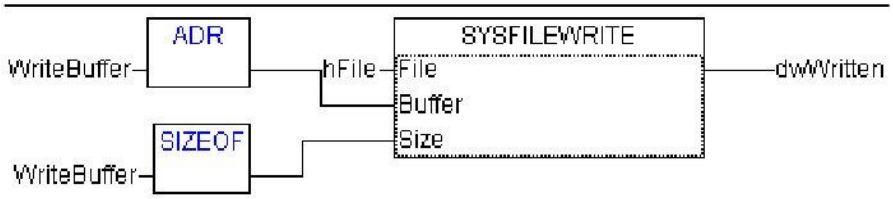
输入变量	数据类型	描述
File	DWORD	文件编号（参看 SysFileOpen 功能块）

SysFileWrite

这个功能（DWORD 型）将数据写入到用 SysFileOpen 功能块打开的文件中，返回值是成功写入文件的字节数。

输入变量	数据类型	描述
File	DWORD	文件编号（参看 SysFileOpen 功能块）
Buffer	DWORD	要写入的数据在缓冲器中的地址(可通过 ADR 操作命令获得）
Size	DWORD	要写入数据的字节数（可通过 SIZEOF 操作命令获得）

例子:
WriteBuffer : ARRAY[0..5] OF BYTE:=0,1,2,4,5,6;
DwWritten : DWORD;
hFile : DWORD;

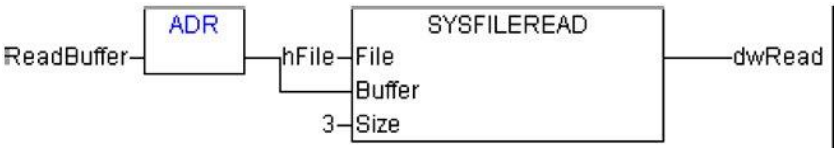


SysFileRead

这个功能（DWORD 型）用来从 SysFileOpen 功能块打开的文件中读取数据，返回值是成功读取的字节数。

输入变量	数据类型	描述
File	DWORD	文件编号（参看 SysFileOpen 功能块）
Buffer	DWORD	要读取的数据在缓冲器中的地址(可通过 ADR 操作命令获得）
Size	DWORD	要读取数据的字节数（可通过 SIZEOF 操作命令获得）

例子:
ReadBuffer : ReadBuffer:ARRAY[0..5] OF BYTE;
hFile : DWORD;
dwRead : DWORD;



SysFileDelete

这个功能（BOOL 型）用来删除一个文件。返回值是 1（正确）或 0（错误）。

输入变量	数据类型	描述
FileName	STRING	文件名称

SysFileGetPos

这个功能（DINT 型）用来返回文件中当前设置的偏移量，这个文件由 SysFileOpen 功能返回的文件编号来确定的。

输入变量	数据类型	描述
File	DWORD	文件编号（参看 SysFileOpen 功能块）

SysFileSetPos

这个功能（DINT 型）用来设置文件的偏移量，这个文件由 SysFileOpen 功能返回的文件编号来确定的。

输入变量	数据类型	描述
File	DWORD	文件编号（参看 SysFileOpen 功能块）
Pos	DWORD	在文件中的访问 offset 值

SysFileEOF

如果当前的偏移量已经在文件的末尾，这个功能块（BOOL 型）将返回 1，否则返回 0。

输入变量	数据类型	描述
File	DWORD	文件编号（参看 SysFileOpen 功能块）

SysFileGetSize

这个功能（DWORD 型）用来返回文件的字节数。

输入变量	数据类型	描述
FileName	STRING	文件名

SysFileGetTime

这个功能（BOOL 型）用来返回文件的创建日期、最后一次访问日期以及最后一次修改日期。习惯的格式是 DT。可以通过访问 FILETIME 结构体获得这些数据。返回值是 1（正确）或 0（错误）。

输入变量	数据类型	描述
FileName	STRING	文件名称
ftFileTime	POINTER TO FILETIME	指向 FILETIME 结构体的指针，可以应用 ADR 操作命令来获得。

FILETIME 结构体是像下面这样定义的：

```
TYPE FILETIME
STRUCT
dtCreation:DT; (* 创建日期 *)
dtLastAccess:DT; (*上次访问的日期 注意在 VxWork 系统中可能只有日期,而没有时间*)
dtLastModification:DT; (* 上次修改的日期 *)
END_STRUCT
END_TYPE
```

SysFileCopy

这个功能（UDINT 型）可以拷贝文件内容到另一个文件下（不同文件名）。它将返回实际拷贝的字节数。

输入变量	数据类型	描述
FileDest	STRING	要拷贝的目标文件名称
FileSource	STRING	要拷贝的源文件名称

SysFileRename

这个功能（BOOL 型）可以重命名一个文件。它将返回 1（正确）或 0（错误）。

输入变量	数据类型	描述
FileOldName	STRING	旧文件名
FileNewName	STRING	新文件名

详细的使用方法请参考示例程序SysLibFileTest.pro

2.5 SysLibFileAsync.lib

这个库可以实现访问来自 IEC-application 的 非实时操作文件。以下面的功能块来举例：

SysFileOpenAsync
SysFileCloseAsync
SysFileWriteAsync
SysFileReadAsync
SysFileDeleteAsync
SysFileGetPosAsync
SysFileEOFAsync
SysFileGetSizeAsync
SysFileGetTimeAsync
SysFileCopyAsnc
SysFileRenameAsync
SysFileCloseAllOpenAsync

共同点：

这个库中的功能块包含一些公共成员：

– 输入参数 bEnable:BOOL

– 输出参数 bDone：BOOL

– 输出参数 bBusy：BOOL

– 输出参数 bError：BOOL

– 输出参数 wErrorId：WORD

所有这些功能块在输入参数 bEnable 输入一个上升沿时开始运行。一旦当功能块开始运行只有当输出参数 bDone 有输出时才可以再次调用功能块。此时输出是有效的。从现在开始。

这些公共参数在文档中不再做明确的解释。

这 I/O 中的 I 表示输入，O 表示输出。

SysFileOpenAsync

这个功能块用来打开一个已经存在的文件或建立一个新文件。它的输出是一个 hFile，也就是文件的一个 handle。一个文件的 handle 就是这个文件的标识符，可以是其它功能块调用文件时的一个输入参数。

输入变量	数据类型	描述
FileName	STRING	文件名称
Mode	STRING	访问模式： w 写模式（可以创建或更新文件） r 读模式（文件以只读形式打开，要是文件不存在，则返回一个错误） rw 读写模式（文件可以被更新，要是文件不存在，则返回一个错误） a 附加模式（文件以“w”模式打开，更新的内容附加在文件的结尾）

SysFileCloseAsync

这个功能块用来关闭一个文件。从此时开始这个文件的 handle 将失效，可以对文件进行其它的操作处理。

输入变量	数据类型	描述
File	DWORD	文件编号（参看 SysFileOpenAsync 功能块）

SysFileWriteAsync

这个功能块用来向文件中写入数据，文件是通过 SysFileOpenAsync 模块打开的。

输入变量	数据类型	描述
File	DWORD	文件编号（参看 SysFileOpenAsync 功能块）
Buffer	DWORD	要写入的数据在缓冲器中的地址（可通过 ADR 操作命令获得）
Size	DWORD	要写入数据的字节数（可通过 SIZEOF 操作命令获得）
dwWrite	DWORD	实际写入的字节数

数据是以二进制形式写入文件的，这意味着没有进行任何的转换。

SysFileReadAsync

这个功能块用来向文件中写入数据，文件是通过 SysFileOpenAsync 模块打开的。

输入变量	数据类型	描述
File	DWORD	文件编号（参看 SysFileOpenAsync 功能块）
Buffer	DWORD	要读取的数据在缓冲器中的地址(可通过 ADR 操作命令获得）
Size	DWORD	要读取数据的字节数（可通过 SIZEOF 操作命令获得）
dwRead	DWORD	实际读取的字节数

pBuffer 参数必须通过 ADR 操作命令来取得，对文件都是直接的读写二进制，没有进行任何的转换而拷贝到 pBuffer 中。

SysFileDeleteAsync

这个功能块用来删除一个文件。

输入变量	数据类型	描述
FileName	STRING	要删除的文件名称

SysFileGetPosAsync

这个功能块将重新找到文件读/写的当前位置。

输入变量	数据类型	描述
File	DWORD	文件编号（参看 SysFileOpenAsync 功能块）

SysFileSetPosAsync

这个功能块将重新设置文件读/写的当前位置。

输入变量	数据类型	描述
File	DWORD	文件编号（参看 SysFileOpenAsync 功能块）
Pos	DWORD	在文件中的访问 offset 值

SysFileEOFAsync

这个功能块用来确定对文件进行读/写的指针是否已经到达文件的结尾。

输入变量	数据类型	描述
File	DWORD	文件编号（参看 SysFileOpenAsync 功能块）
bEOF	DWORD	表明是否文件的 handle 已经到达

SysFileGetSizeAsync

这个功能块用来返回文件的字节数。

输入变量	数据类型	描述
File	DWORD	文件编号（参看 SysFileOpenAsync 功能块）
dwSize	DWORD	获取的文件字节数

SysFileGetTimeAsync

这个功能块用来获得文件的修改时间。

输入变量	数据类型	描述
stFileName	STRING	文件名称
ftFileTime	POINTER TO FILETIME	指向 FILETIMEAsync 结构体的指针，可以应用 ADR 操作命令来获得。

FILETIMEAsync结构体是像下面这样定义的：

```
TYPE FILETIMEAsync
STRUCT
    dtCreation:DT; (* 创建日期 *)
    dtLastAccess:DT; (*上次访问的日期 注意在 VxWork 系统中可能只有日期,而没有时间*)
    dtLastModification:DT; (* 上次修改的日期 *)
END_STRUCT
END_TYPE
```

SysFileCopyAsync

这个功能块用来拷贝一个文件到另一个文件或位置。

输入变量	数据类型	描述
FileDest	STRING	要拷贝的目标文件名称
FileSource	STRING	要拷贝的源文件名称
dwCopied	DWORD	拷贝的字节数

SysFileRenameAsync

这个功能块用来重命名一个文件。

输入变量	数据类型	描述
FileOldName	STRING	旧文件名
FileNewName	STRING	新文件名

SysFileCloseAllOpenAsync

通过这个功能块用户可以关闭当前打开的所有文件，不用知道每个文件的 handle 或名称。系统可以自己默认这些 handle 值。

详细的使用方法请参考示例程序SysLibFileAsyncTest.pro

2.6 SysLibEvent.lib

这个库用来提供同步和控制两个（IEC-）任务的处理。一个任务等待一个事件的发生使其能被重新激活，第二个任务可以设置这个事件的发生。

下面是功能块的定义说明，可以删除、开始一个事件和设置超时时间。（执行是实时的）

SysEventCreate
SysEventDelete
SysEventSet
SysEventWait

SysEventCreate

这个功能块（DWORD 型）用来新建及命名一个事件，然后返回一个 handle 值，库中的其它功能块可以通过这个 handle 值对此事件进行访问。

输入变量	数据类型	描述
stName	STRING	新事件的名称

SysEventDelete

这个功能块（BOOL 型）用来删除一个事件，这个事件通过上面的 SysEventCreate 功能块创建并返回了 handle 值。如果事件被成功删除则返回 TRUE 值，否则返回 FALSE。

输入变量	数据类型	描述
DwHandle	DWORD	通过 SysEventCreate 功能块返回的 handle 值

SysEventSet

这个功能块（DWORD 型）用来设置一个事件，事件是前面用 SysEventCreate 功能块创建并返回了 handle 值。如果事件被成功设置则返回 TRUE，否则返回 FALSE。

输入变量	数据类型	描述
DwHandle	DWORD	通过 SysEventCreate 功能块返回的 handle 值

SysEventWait

这个功能块（DWORD 型）用来设置一个事件的超时时间，事件是前面用 SysEventCreate 功能块创建并返回了 handle 值。如果超时时间被成功设置则返回 TRUE，否则返回 FALSE。

输入变量	数据类型	描述
DwHandle	DWORD	通过 SysEventCreate 功能块返回的 handle 值
dwTimeout	DWORD	超过这个时间后功能块将返回到最新，单位是 ms

2.7 SysLibIECTask.lib

这个库可以用来访问 IEC 任务的配置信息。执行过程是实时的。（可以通过 SysLibTasks.lib 这个库来创建、删除、设置优先级、停止和重起一个任务）获得信息的功能块：

SysIECTaskGetConfig
SysIECTaskGetInfo
附加功能：
SysIECGetFctPointer
SysIECTaskResetEvent

SysIECTaskGetConfig

这个功能块（BOOL 型）用来重新得到 IEC 任务的参数配置。任务已经通过它的名字索引标志了它的地址，这些在 OtoStudio 任务配置中已经赋值。当任务被找到则返回 TRUE 值，否则返回 FALSE。

输入变量	数据类型	描述
udiTaskId	UINT	任务的 Id（在 OtoStudio 任务配置中的指针）（也可以选用 stTaskName）
pTaskInfo	POINTER TO SYSIECTASKCONFENTRY	任务配置信息（结构体，参考如下）

结构体 SysIECTaskConfEntry 的组成：

```
TYPE SYSIECTASKCONFENTRY :
STRUCT
    byTaskNr: USINT; 任务编号
    byPriority: USINT; 任务优先级 (参考otostudio->任务配置)
    lInterval: DINT; 任务循环时间间隔 (参考otostudio->任务配置)
    ldrEvent: LDATAREF_TYPE;事件任务的事件，具体参考后面LDATAREF_TYPE声明
    wIndex: UINT; 被任务调用的 POU 的指针（与通过 INDEXOF()得到的指针相匹配）
    ulNameLen: UDINT; 任务名称的长度
    szName: STRING(80);任务名称(参考otostudio->任务配置)
END_STRUCT
END_TYPE
```

结构体 LdataRef_Type 的组成：

```
TYPE LDATAREF_TYPE :
STRUCT
    POURef: UINT; 事件变量的 POU_ID
    Offset: UDINT; 事件变量的偏移量
    Size: UDINT; 事件变量的大小
END_STRUCT
END_TYPE
```

SysIECTaskGetInfo

这个功能块（BOOL 型）用来返回当前 IEC 任务的时间值。这任务可以通过任务名称或指针得到，这些也都在任务配置中设置了。如果任务被找到则返回 TRUE 值，否则返回 FALSE。

输入变量	数据类型	描述
stTaskName	STRING	任务名称（可随意应用输入变量 UdiTaskId，参考下面）
pTaskInfo	POINTER TO SYSIECTASKINFO	指向当前 IEC 任务数据的指针（结构体，参考下面）

结构体 SysIECTaskInfo 组成：

```
TYPE SYSIECTASKINFO :
STRUCT
    dwCount:DWORD; 开始任务后的周期数
    dwCycleTime:DWORD; 当前循环周期时间
    dwCycleTimeMin:DWORD; 最小循环周期时间
    dwCycleTimeMax:DWORD; 最大循环周期时间
    dwCycleTimeAvg:DWORD; 平均循环周期时间
    wStatus:WORD; PLC 的当前状态：0 = RUN，1 = STOP
    wMode:WORD; 当前任务模式 :0 = running ,1 = stopped( maybe by a runtime error )
END_STRUCT
END_TYPE
```

附加功能：

SysIECGetFctPointer

对于用来创建一个新任务的 SysTaskCreate 功能块(参考 SysLibTasks.lib 库)有一个必须的输入参数，这个辅助的功能块（DWORD 型）为这个输入参数返回一个功能指针。此功能块需要一个被任务调用的 POU 的内部指针作为输入参数。这个指针可以通过INDEXOF 操作命令来获得。

输入变量	数据类型	描述
wIndexOf	WORD	被任务调用的 POU 的内部指针

SysIECTaskResetEvent

这个辅助功能块(BOOL 型)用来重新安排一个已经触发的 IEC 任务的事件变量。此功能块没有输入参数。它在当前任务下调用。如果成功返回 TRUE，否则返回 FALSE。（例如如果任务不是一个触发的任务事件）
这个功能块设置布尔量的用来作为一个事件的 IEC 变量，要是 FALSE，Googol runtime 系统任务管理的内在标志设置为 0。所以它将在一个程序循环周期的事件变量的上升沿时完成。

详细的使用方法请参考示例程序SysLibIECTaskTest.pro

2.8 SysLibTask.lib

这个库提供的功能块可以用来管理任务。也就是说可以创建、删除、修改优先级、停止和重起一个任务。执行过程同步。

注意：这些程序不支持多线程，在一般的状态下没有问题，但是在应用程序状态下一种情况可能发生，就是几个 IEC 任务的创建和处理额外的任务，应该同步调用这些功能块。这种情形下可以应用 SysLibSem.lib 库。

（如果需要功能块得到 IEC 任务的配置信息，可应用 SysLibIECTasks.lib 库）

管理任务的功能块如下：

SysTaskCreate
SysTaskDestroy
SysTaskGetInfo
SysTaskGetPriority
SysTaskSetPriority
SysTaskSuspend
SysTaskResume

任务内部调用的功能块如下：

SysTaskSleep
SysTaskEnd
SysTaskGetCurrent

SysTaskCreate

这个功能块（UDINT 型）用来创建一个新任务。它将为任务返回一个唯一的 ID 号，这个 ID 号对于库中其它的功能块来说是一个必需的输入参数。

输入变量	数据类型	描述
stName	STRING	任务的名称
byPriority	BYTE	任务的优先级，可能值为：0～255 保留：为系统留下 0..31 IEC 任务：32..63 通讯任务：64 及以上
udiInterval	UDINT	以毫秒计算的任务时间间隔
pfFunction	DWORD	指向功能的块指针，它必须通过 SysIECGetFctPointer 功能块已获得
pArgument	DWORD	指向新任务的参数的指针

SysTaskDestroy

这个功能块（BOOL 型）用来删除一个任务。如果操作成功返回 TRUE，否则返回 FALSE。

输入变量	数据类型	描述
udiTaskId	UDINT	要删除的任务的 ID 号，这个号是SysTaskCreate 功能块的返回值

SysTaskGetInfo

这个功能块（BOOL 型）将返回通过任务 ID 来识别的任务的信息。

输入变量	数据类型	描述
udiTaskId	UDINT	想获得信息的任务的 ID 号，这个号是 SysTaskCreate 功能块的返回值
pSysTaskInfo	POINTER TO SYSTASKINFO	指向包含任务信息的 SysTaskInfo 结构体的指针，参考下面

SysTaskInfo 结构体的组成：

```
TYPE SYSTASKINFO :
STRUCT
    dwHandle:DWORD; 任务操作系统的 handle
    dwId:DWORD; 任务索引
    dwSem:DWORD; 保留，仅用在 runtime 系统
    wIECTaskNr:WORD; 如果它是 IEC 任务，则为 IEC 任务索引
    stName:STRING; 任务的名称
END_STRUCT
```

END_TYPE

SysTaskGetPriority

这个功能块（BYTE 型）将返回通过任务 ID 来识别的任务的优先级。这任务优先级的可取值在 0（最高优先级）到 255（最低优先级）之间。

输入变量	数据类型	描述
udiTaskId	UDINT	想获得优先级的任务的 ID 号，这个号是 SysTaskCreate 功能块的返回值

SysTaskSetPriority

这个功能块（BOOL 型）可用来设置通过任务 ID 来识别的任务的优先级。如果操作成功返回 TRUE，否则返回 FALSE。
这任务优先级的可取值在 0（最高优先级）到 255（最低优先级）之间。

输入变量	数据类型	描述
udiTaskId	UDINT	想设置优先级的任务的 ID 号，这个号是 SysTaskCreate 功能块的返回值
byPriority	BYTE	任务的优先级，可能值为：0～255 保留：为系统留下 0..31 IEC 任务：32..63 通讯任务：64 及以上

SysTaskSuspend

这个功能块（BOOL 型）可停止一个正在运行的任务。这任务可通过任务 ID 号来识别。（可通过调用 SysTaskResume 功能块来使任务继续执行）

如果成功的停止任务返回 TRUE，否则返回 FALSE。

输入变量	数据类型	描述
udiTaskId	UDINT	想要停止的任务的 ID 号，这个号是 SysTaskCreate 功能块的返回值

SysTaskResume

这个功能块（BOOL 型）可使上面 SysTaskSuspend 功能块先前停止的任务继续运行。

如果任务成功重新运行返回 TRUE，否则返回 FALSE。

输入变量	数据类型	描述
udiTaskId	UDINT	想要继续运行的任务的 ID 号，这个号是 SysTaskCreate 功能块的返回值

SysTaskSleep

这个功能块（BOOL 型）可以中断一个正在运行的任务，然后使其在定义好的一段时间后、继续运行。如果休眠功能成功执行返回 TRUE，否则返回 FALSE。

输入变量	数据类型	描述
udiMilliseconds	UDINT	以毫秒计算的任务停止运行的时间，这个时间后任务将继续运行

SysTaskEnd

这个功能块（BOOL 型）在一个任务处理过程结束时应该被调用。典型的是当一个任务离开是这个功能块应该立即被调用执行。

输入变量	数据类型	描述
udiExitCode	UDINT	应该设置为 0
udiTaskId	UDINT	想要终止的任务的 ID 号，这个号是 SysTaskCreate 功能块的返回值

SysTaskGetCurrent

这个功能块（UDINT 型）可被当前运行的任务调用来获得自己的任务 ID 号。

输入变量	数据类型	描述
Dummy	BOOL	TRUE 就激活这个功能块

2.9 SysLibTime.lib

下面描述的这些功能块将可以用来读取本地系统的实时时间。执行过程是同步的。这个库需要 otostudio 在任务编辑器里分配显示任务。

CurTime
CurTimeEx

相关的数据结构：

SystemTimeDate
SysTime64

CurTime

这个功能块以毫秒级来提供本地系统的实时时间。应用 SysTime64 结构体。

输入输出变量	数据类型	描述
SystemTime	SysTime64	我们以毫秒从本地系统读取的时间。参考 SysTime64结构体

CurTimeEx

这个功能块提供本地系统当前实时时间的额外信息。

输入输出变量	数据类型	描述
SystemTime	SysTime64	我们以毫秒从本地系统读取的时间。参考 SysTime64结构体
TimeDate	SystemTimeDate	从本地系统读取的时间的额外信息 （SystemTimeDate 结构体）。参考下面

SysTime64

这个结构体包含以微秒给出的本地系统的时间。为此应用低加高DWORD，要用 64 位。它被CurTimeEx和CurTime功能块所调用。

组成：

```
TYPE SysTime64 :
STRUCT
    ulLow:DWORD; 实际时间的低32位的值（微秒）
    ulHigh:DWORD; 实际时间的高32位的值（微秒）
END_STRUCT
END_TYPE
```

SystemTimeDate

这个结构体包含本地系统时钟给出的实时时间的下面信息。它被CurTimeEx功能块所调用。

组成：

```
TYPE SystemTimeDate :
STRUCT
    dwLowMSecs : DWORD;
    dwHighMsec : DWORD;
以微秒返回的时间，用低 DWORD 加上高 DWORD，参考 SysTime6 结构体
    Year : UINT;        年
    Month : UINT;       月
    Day : UINT;         日
    Hour : UINT;        小时
```

```
        Minute : UINT;     分钟
        Second : UINT;     秒
        Milliseconds : UINT;    毫秒
        DayOfWeek : UINT;     星期几
    END_STRUCT
    END_TYPE
```

详细的使用方法请参考示例程序SysLibFileTimeTest.pro

2.10 SysLibGetAddress.lib

这个库提供两个功能块 ,将返回某个确定的数据段(Memory ,Input ,Output ,Retain 或 Global) 的开始地址和大小。

SysLibGetAddress

这个功能块（DWORD 型）可以用来返回某个确定的数据段（Memory，Input，Output，Retain或 Global）的开始地址。它已将数据段的编号列入清单。

输入变量	数据类型	描述
iSegment	INT	数据段编号

SysLibGetSize

这个功能块（DWORD 型）可以用来返回某个确定的数据段（Memory，Input，Output，Retain或 Global）的大小。它已将数据段的编号列入清单。

输入变量	数据类型	描述
iSegment	INT	数据段编号

举例：

ADDRESS_SEGMENTS

这个举例描述了不同数据段的编号:

```
TYPE ADDRESS_SEGMENTS :
(
    DATAID_MEMORY,
    DATAID_INPUT,
    DATAID_OUTPUT,
    DATAID_RETAIN,
    DATAID_GLOBVARS
);
END_TYPE
```

详细的使用方法请参考示例程序SysLibGetAddressTest.pro

2.11 SysLibInt.lib

这个库可以用来对一个功能块设置和移除一个中断 handle。这个执行过程是实时的。这功能块如下：

SysInstallIntHandler
SysRemoveIntHandler

SysInstallHandler

这个功能块（BOOL 型）可以通过对功能块地址的识别给予一个编号来设置中断。这个地址可以用 SysIECGetFctPointer 功能块来得到。（参考 SysLibIECTasks.lib 库）若操作成功则返回 TRUE，否则返回 FALSE。这个中断 handle 可以用 SysRemoveIntHandler 功能块来移除。

输入变量	数据类型	描述
iInterrupt	INT	中断编号
dwFctAddress	DWORD	功能块指针，可用 SysIECGetFctPointer (SysLibIECTasks.lib)来获得

SysRemoveHandler

这个功能块（BOOL 型）可以通过对功能块地址的识别给予的编号来移除一个中断。这个地址可以用 SysIECGetFctPointer 功能块来得到。（参考 SysLibIECTasks.lib 库）
这中断 handle 是通过 SysInstallIntHandler 功能块来设置的。

输入变量	数据类型	描述
iInterrupt	INT	中断编号
dwFctAddress	DWORD	功能块指针，可用 SysIECGetFctPointer (SysLibIECTasks.lib)来获得

详细的使用方法请参考示例程序SysLibIntTest.pro

2.12 SysLibMem.lib

此库可以用来对内存进行管理。下面这些功能块将可以用来分配、释放、定义、比较一片内存区域，也能用来在不同的内存区域间来拷贝、移动、交换数据。执行过程是实时的。

SysMemAlloc
SysMemFree
SysMemMove
SysMemSet
SysMemCmp
SysMemCpy
SysMemSwap

SysMemAlloc

这个功能块（DWORD 型）可用来动态分配内存空间。返回值是一个指向分配的内存空间的指针，要是这没有足够的需要的内存空间将返回 0。即使分配一个小的内存空间，这个返回值也是需要被检查的。

输入变量	数据类型	描述
dwSize	DWORD	要分配的字节数

SysMemFree

这个功能块（BOOL 型）用来释放内存空间。若操作成功返回 TRUE，否则返回 FALSE。

输入变量	数据类型	描述
dwAddress	DWORD	当前分配的内存空间的地址（参考 SysMemAlloc）
dwSize	DWORD	要释放的字节数

SysMemCmp

这个功能块(DWORD 型)用来比较两个 dwCount 大小的内存缓冲器里的内容 ,dwbuf1 和 dwbuf2 分别表示两个缓冲器的开始地址。这个功能块将返回两个缓冲器的内容差别。

<0 表示buf1 比 buf2 小

=0 表示buf1 和 buf2 大小相等

>0 表示buf1 比 buf2 大

输入变量	数据类型	描述
dwBuf1	DWORD	内存缓冲器 1 的地址
dwBuf2	DWORD	内存缓冲器 2 的地址
dwCount	DWORD	需要比较的字节数

SysMemCpy

这个功能块（DWORD 型）可用来从一个缓冲器拷贝一定字节数的内存区域内容到另一个缓冲器中。这个功能块将返回一个指向目标缓冲器地址的指针。此与 SysMemMove 功能块的不同在于这个功能块只允许在两个不邻接的缓冲器间拷贝。

输入变量	数据类型	描述
dwDest	DWORD	目标缓冲器的地址
dwSrc	DWORD	源缓冲器的地址
dwCount	DWORD	需要拷贝的字节数

SysMemMove

这个功能块（DWORD 型）用来移动一个缓冲器中的内容到令一个缓冲器。这个功能块将返回一个指向目标缓冲器地址的指针。此与 SysMemCpy 功能块的不同在于这个功能块允许在两个邻接甚至重叠的缓冲器间拷贝。

输入变量	数据类型	描述
dwDest	DWORD	目标缓冲器的地址
dwSrc	DWORD	源缓冲器的地址
dwCount	DWORD	需要拷贝的字节数

SysMemSet

这个功能块（DWORD 型）用来初始化一个已经定义好值的内存区域。它将返回一个指向目标缓冲器地址的指针。

输入变量	数据类型	描述
dwDest	DWORD	指向初始化内存区域地址的指针
bCharacter	BYTE	用来初始化内存区域的特性或数字值
dwCount	DWORD	内存区域的字节数

SysMemSwap

该功能块被禁止使用。

2.13 SysLibOS.lib

此库可以用来对操作系统资源操作。用于实现创建进程，调用操作系统应用程序或者可执行程序。执行过程是实时的。

SysCreateProcess
SysExecuteCommand

SysCreateProcess

这个功能块（DWORD 型）可用来创建一个进程，并且返回该进程的Handle

输入变量	数据类型	描述
pszCommandLine	STRING	开始一个新的进程的命令行，一般这个参数是该

		新进程的模块名称
bHide	BOOL	隐藏该命令行执行的窗口

SysExecuteCommand

这个功能块（DINT 型）执行系统命令，并且返回结果。

输入变量	数据类型	描述
pszCommandLine	STRING	开始一个新的进程的命令行，一般这个参数是该新进程的模块名称

详细的使用方法请参考示例程序SysLibOSTest.pro

2.14 SysLibPlcCtrl.lib

此库包含用来 PLC 控制的下列功能块。执行过程是实时的。
SysStartPlcProgram
SysResetPlcProgram
SysStopPlcProgram
SysShutdownPlc
SysEnableScheduling
SysGetPlcLoad
另外这有功能块用来处理/保留变量：
SysRestoreRetains
SysSaveRetains
还有一个功能块用来激活看门狗：
SysWdgEnable
这些功能块除了 SysResetProgram 执行过程都是实时的。这功能块创建一个任务来执行命令。

SysStartPlcProgram

这个功能块（BOOL 型）用来启动一个 PLC。若成功返回一个 TRUE，否则返回 FALSE。

输入变量	数据类型	描述
Dummy	BOOL	没有功能

SysResetPlcProgram

这个功能块（BOOL 型）用来重起一个 PLC。重起模式可通过 Reset_Mode 列举帮助来设置。此功能块永远返回 TRUE。
这个功能块不是实时执行的，但它可以创建一个任务。此任务的优先级比最低用户任务的优先级还要低。
注意：在一个 callback 中这个功能块可能不被调用，尤其在没有用户任务创建或删除时。例如 EVENT_BEFORE_RESET, EVENT_AFTER_RESET, EVENT_SHUTDOWN, EVENT_STOP 。

输入变量	数据类型	描述
rmRESETMODE	RESET_MODE	选取一个希望的 PLC 重起命令列举 值： 0=RESET_WARM 1=RESET_COLD 2=RESET_HARD RESET_WARM 相当于在 OtoStudio 在线菜单里的热复位，RESET_HARD 相当于复位到初始化状态

SysStopPlcProgram

这个功能块（BOOL 型）用来停止一个 PLC。若操作成功返回 TRUE，否则返回 FALSE。

输入变量	数据类型	描述
Dummy	BOOL	没有功能

SysEnableScheduling

这个功能块（DWORD 型）用来使 PLC 丧失或拥有调度 IEC 任务的能力。

输入变量	数据类型	描述
bEnable	BOOL	没有功能

SysGetPlcLoad

这个功能块（DWORD 型）返回当前处理器加载的 IEC 任务。

输入变量	数据类型	描述
Dummy	BOOL	没有功能

SysSaveRetains

这个功能块（DINT 型）可以用来保存文件中 retain 变量的值。将返回下面值中的一个：
1：OK
0：没有程序加载
-1：文件无法打开

输入变量	数据类型	描述
stFileName	STRING	想保存 retain 变量文件的文件名，注意：默认情况下为 ‘ ’；系统会自动分配 ‘retain.ret’

SysRestoreRetains

这个功能块(DINT 型)可以用来从文件中恢复 retain 变量的值。将返回下面值中的一个：
1：OK
0：没有程序加载
-1：文件无法打开
-2：连接的文件比 retain 区域大

输入变量	数据类型	描述
stFileName	STRING	保存 retain 变量文件的文件名，注意：默认情况下系统会在开机时自动调用该功能，恢复所有的保持类型变量的值

SysWdgEnable

这个功能块（BOOL 型）可用来为指定的 IEC 任务加载或解除看门狗。要是操作成功返回 TRUE，否则返回 FALSE。详细的使用方法请参考示例程序SysLibPlcCtrlTest.pro

输入变量	数据类型	描述
bEnable	BOOL	如果 TRUE：看门狗功能激活；如果 FALSE：看门狗功能解除。
byIECTaskIndex	BYTE	用来激活/解除看门狗功能的 IEC 任务的指针
stIECTaskName	POINTER TO STRING	IEC 任务的名字，可以是指空指针

2.15 SysLibProjectInfo.lib

这个库包含的功能块可以各自工程 ID 来读取工程信息。这个执行过程是实时的。
这还有一个功能块用来得到工程的 ID。

SysGetProjectInfo

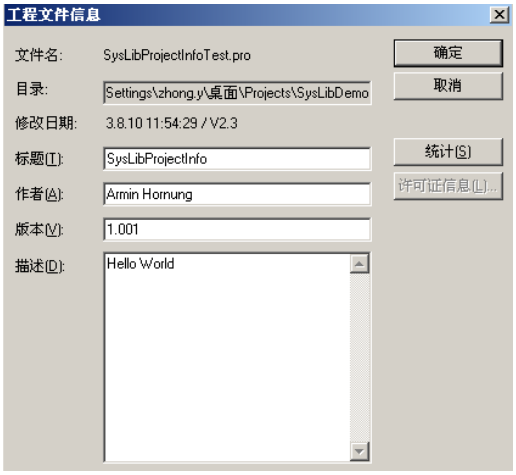
这个功能块(BOOL 型)可以提供在系统程序上登陆的工程信息的组成。‘工程’‘工程信息’）；结构体 PROJECT_INFO 用来存储这些信息。要是操作成功返回 TRUE，否则返回 FALSE。

输入变量	数据类型	描述
ProjectInfo	POINTER TO PROJECT_INFO	指向结构体 PROJECT_INFO 的指针，这里存有工程信息。可以用 ADR 操作来得到偏移量。

PROJECT_INFO

这个结构体包含在 OtoStudio 登陆的工程的工程信息（‘工程’ ‘工程信息’）。它被 SysGetProjectInfo 功能块调用。
组成：

```
TYPE PROJECT_INFO :
STRUCT
    dtDate : DT;                    修改时间
    stProject : STRING(255);        文件名
    stTitle : STRING(255);          标题
    stVersion : STRING(255);        版本
    stAuthor : STRING(255);         作者
    stDescription : STRING(255);    描述
END_STRUCT
END_TYPE
```



详细的使用方法请参考示例程序SysLibPlcProjectInfoTest.pro

2.16 SysLibRtc.lib

这个库包含的功能块可用来访问目标系统的 realtime 时钟。realtime 时钟可以被读取和设置。另外当电池重新装入时可以设置时间的显示模式。执行过程是实时的。

功能块如下：

SysRtcCheckBattery
SysRtcGetHourMode
SysRtcGetTime
SysRtcSetTime

提示：在这个上下文中需要 RTC 功能块，它是 OtoStudio 标准库的一部分，它可以返回正在运行的日期时间和涉及开始的时间。

因为开始时间必须被明确的设置，所以 RTC 并不是一个真实的 realtime 时钟功能块。但用 RTC 可以保存系统的运行时间。可以考虑用 SysRtcGetTime 功能块来设置 RTC的开始时间，SysRtcGetTime 功能块在 SysLibRtc.lib 库中有描述。

SysRtcCheckBattery

这个功能块（BOOL 型）用来检查本地系统 RTC 电源状态，这对于精确的时钟显示是非常重要的。

如果电源状态不正确则返回 FALSE，否则返回 TRUE。

输入变量	数据类型	描述
bDummy	BOOL	TRUE 就开始功能

SysRtcGetHourMode

这个功能块（BOOL 型）可以用来读取本地系统 RTC 的显示模式。若是 12 小时模式则返回 FALSE，若是 24 小时模式则返回 TRUE。

输入变量	数据类型	描述
bDummy	BOOL	TRUE 就开始功能

SysRtcGetTime

这个功能块（DATA-AND-TIME）返回读取本地系统 RTC 的当前时间。

输入变量	数据类型	描述
bDummy	BOOL	TRUE 就开始功能

SysRtcSetTime

这个功能块(DATA-AND-TIME)可以用来设置本地系统 RTC。若操作成功则返回 TRUE，否则返回 FALSE。

输入变量	数据类型	描述
ActDateAndTime	DATE_AND_TIME	设置电脑的真实时间时钟

详细的使用方法请参考示例程序SysLibRtcTest.pro

2.17 SysLibSem.lib

这个库可为了任务的同步创建和应用 semaphores。这 semaphores 服务可避免任何的几个任务同时访问同一数据。执行是实时的。

下面的功能块是可以应用的：

SysSemCreate 用来创建一个 semaphore
SysSemDelete 用来删除一个 semaphore
SysSemEnter 用来占用一个 semaphore
SysSemLeave 用来离开一个 semaphore
SysSemTry 用来检查是否这个 semaphore 被另一个任务占用

SysSemCreate

这个功能块（DWORD）可以用来创建一个 semaphore。这个功能块返回一个 handle，用来标识 semaphore，这个 handle 在库中其它功能块中可以作为一个输入参数。

输入变量	数据类型	描述
bDummy	BOOL	如果bDummy = TRUE ,则创建一个semaphore

SysSemDelete

这个功能块（BOOL 型）可以通过前面 SysSemCreate 功能块得到的 handle 标识删除一个 semaphore。若操作成功返回 TRUE，否则返回 FALSE。

输入变量	数据类型	描述
dwHandle	DWORD	通过 SysSemCreate 功能块得到的 semaphore 的一个handle

SysSemEnter

当数据被其它任务同时调用时，这个功能块（BOOL 型）必须在这个任务访问数据前调用。这样这些数据将为其它任务而锁定，直到应用 SysSemLeave 功能块释放 semaphore时而得到释放。
这 semaphore 将通过前面 SysSemCreate 功能块得到的 handle 来标识。若操作成功返回 TRUE，否则返回 FALSE。

输入变量	数据类型	描述
dwHandle	DWORD	通过 SysSemCreate 功能块得到的 semaphore 的一个handle

SysSemLeave

当数据被其它任务同时调用时，这个功能块（BOOL 型）必须在访问数据后调用。这是由于必须释放前面 SysSemEnter 功能块访问数据前锁定的 semaphore。
这 semaphore 将通过前面 SysSemCreate 功能块得到的 handle 来标识。若操作成功返回 TRUE，否则返回 FALSE。

输入变量	数据类型	描述

dwHandle	DWORD	通过 SysSemCreate 功能块得到的 semaphore 的一个handle

SysSemTry

这个功能块（BOOL 型）（参考 SysLibSem.lib 库）可以用来检查一个 semaphore当前是否被另外一个任务占用（通过 SysSemEnter）。
这 semaphore 将通过前面 SysSemCreate 功能块得到的 handle 来标识。若操作成功返回 TRUE，否则返回 FALSE。

输入变量	数据类型	描述
dwHandle	DWORD	通过 SysSemCreate 功能块得到的 semaphore 的一个handle

详细的使用方法请参考示例程序SysLibSemTest.pro

2.18 SysLibShm.lib

有的内存区域是被几个处理过程分别涉及到的物理地址所公共应用的，这个库提供的功能块可以访问这些内存区域。（共享内存，shortcut ShM ）

这个库提供的功能块将可以用来打开和关闭这个 shM，以及从中读取和写入数据。这读、写、关闭功能块需要一个打开功能块返回的 handle 值。执行过程是实时的。

SysShmOpen
SysShmClose
SysShmRead
SysShmWrite

SysShmOpen

这个功能块（DWORD 型）打开一个共享内存。
它为 shM 返回一个 handle 值，这个 handle 可以当做一个指针来用。这个 handle 值是其它功能块一个必要的输入参数。

输入变量	数据类型	描述
stName	STRING	可以设置希望的共享内存名称
dwPhysicalAddress	DWORD	设置下面中的一个： 1、 希望的 shM 的物理地址，必须有效。 2、 如果 shM 区域的位置可以任意，则为 0。
pdwSize	DWORD	指向 shM 区域大小的指针 1、 要是 shM 区域以及存在，将返回实际的大小 2、 要是 shM 不存在，将根据所给的大小创建一个 shM 区域。如果输入‘0’这个功能将失效。因此这个功能也可以用来检查 shM 是否已被创建。

SysShmClose

这个功能块(BOOL 型)将关闭通过前面 SysShmOpen 功能块返回的 handle 值所确定的共享内存。如果操作成功返回 TRUE，否则返回 FALSE。

输入变量	数据类型	描述
hShm	DWORD	SysShmOpen 功能块返回的共享内存的 handle 值

SysShmRead

这个功能块（DWORD 型）可以从共享内存读取指定的字节数，开始位置在一个确定的偏移量。它将返回实际读取的字节数。

输入变量	数据类型	描述
hShm	DWORD	SysShmOpen 功能块返回的共享内存的 handle 值
dwOffset	DWORD	开始读取数据区域的偏移量
pData	DWORD	要读取的数据缓冲器的地址
dwSize	DWORD	要读取的字节数

SysShmWrite

这个功能块（DWORD 型）可以向共享内存写入指定的字节数。它将返回实际写入的字节数。

输入变量	数据类型	描述
hShm	DWORD	SysShmOpen 功能块返回的共享内存的 handle 值
dwOffset	DWORD	开始写入数据区域的偏移量
pData	DWORD	要写入的数据缓冲器的地址
dwSize	DWORD	要写入的字节数

详细的使用方法请参考示例程序SysLibShmTest.pro

2.19 SysLibSockets.lib

这个库支持通过 TCP_IP 和 UDP 通讯访问 sockets。
下面列出的功能块表将可用，对应操作系统调用相应的功能块。
执行过程是同步的。

要是许多 sockets 同时打开/关闭，可能将花费很长的时间。

推荐使用优先级低的另一个任务，以便减小对主控制任务的影响。同样也是针对异步功能块，请参考 SysLibSocketsAsync.lib 库。

SysSockAccept

SysSockBind
SysSockClose
SysSockConnect
SysSockCreate
SysSockGetHostByName
SysSockGetHostName
SysSockGetOption
SysSockGetLastError
SysSockHtonI
SysSockHtons
SysSockInetAddr
SysSockInetNtoa
SysSockIoctl
SysSockListen
SysSockNtohI
SysSockNtohs
SysSockSelect
SysSockSetIPAddress
SysSockSetOption
SysSockShutdown

特殊的 TCP：

SysSockRecv
SysSockSend

特殊的 UDP：

SysSockRecvFrom,
SysSockSendTo

SysSockAccept

这个功能块（DINT 型）调用操作系统的同意功能，这样可以同意一个 socket 的连接请求。将为 socket 返回一个新的描述符（handle）。最初的 socket 将被重新设置为‘倾听’状态。（参考 SysSockListen）

输入变量	数据类型	描述
diSocket	DINT	识别被倾听功能块设置为倾听状态的socket 的描述符。与 socket 的连接实际上是通过 SysSockListen 功能块返回的。 被请求的连接然后和那个 socket 建立，为此 SysSockAccept 功能块返回一个handle。
pSockAddr	DWORD	指向 SOCKADDR 类型变量的指针；可调用来加载地址。
piSocketAddrSize	DWORD	指向 DINT 类型变量的指针。这个变量将赋值为结构体 SockAddr 的长度。（可通过SIZEOF 命令来得到）

SOCKADDR 结构体：

sin_family : INT; (* Adress-family, 定义地址格式 *)

sin_port : UINT; (* 连接请求个体的端口*)

sin_addr : UDINT; (* 请求个体的 IP 地址*)

sin_zero : ARRAY [0..7] OF SINT; (* 缓冲器 *)

SysSockBind

这个功能块（BOOL 型）将调用操作系统的绑定功能。它将为前面表示的 socket 分配一个本地地址，这个地址属于 SysSockCreate 功能块创建的地址范围。通常情况下，‘绑定’功能在像 SysSockListen 或 SysSockAccept 功能块调用 socket 前完成。如果操作成功返回 TRUE，否则返回 FALSE。

输入变量	数据类型	描述
diSocket	DINT	由 SysSockCreate 返回的 socket 的描述符。
pSockAddr	DWORD	指向 SOCKADDR 类型变量的指针；可调用来加载地址。
diSockAddrSize	DINT	结构体 SockAddr 的长度。（可通过 SIZEOF 命令来得到）

SysSockClose

这个功能块（BOOL 型）用来调用操作系统的关闭 socket 功能，以关闭一个 socket。如果操作成功返回 TRUE，否则返回 FALSE。

输入变量	数据类型	描述
diSocket	DINT	由 SysSockCreate 返回的 socket 的描述符。

SysSockConnect

这个功能块（BOOL 型）用来调用操作系统的连接功能。要是 socket 还没有被 SysSockBind 功能块‘绑定’，就自动的给它分配一个本地地址。然后这 socket 就准备好了发送或接收数据。如果操作成功返回 TRUE，否则返回 FALSE。

输入变量	数据类型	描述
diSocket	DINT	由 SysSockCreate 返回的 socket 的描述符。
pSockAddr	DWORD	指向 SOCKADDR 类型变量的指针；可调用来加载地址。
diSockAddrSize	DINT	结构体 SockAddr 的长度。（可通过 SIZEOF 命令来得到）

SysSockCreate

这个功能块（DINT 型）用来调用操作系统的 socket 功能。可创建一个新的 socket并分配一个服务供应者。

这个功能块返回新 socket 的描述符，此描述符将作为库中其它功能块的一个输入参数。例如 SysSockConnect，SysSockBind。

输入变量	数据类型	描述
diAddressFamily	DINT	Address family。

diType	DINT	下面两种类型的一种，例如对于 Windows Sockets1.1：SOCK_STREAM，SOCK_DGRAM
diProtocol	DINT	依赖于所选 Address family 的协议

SysSockGetHostByName

这个功能块（DWORD 型）将调用 Win32 系统的 gethostbyname 功能。如果操作成功将返回主机的地址，否则返回 SOCKET_INADDR_NONE（库中定义的一个全局常量）

输入变量	数据类型	描述
stHostName	POINTER TO STRING	主机的名称

SysSockGetHostName

这个功能块（BOOL 型）将调用操作系统的 gethostname 功能然后返回主机名称。
如果操作成功返回 TRUE，否则返回 FALSE。

输入变量	数据类型	描述
stHostName	POINTER TO STRING	主机的名称
diNameLength	DINT	主机名称的长度

SysSockGetOption

这个功能块（BOOL 型）将调用操作系统的 getsockopt 功能，来获得 socket 选项细节值。如果操作成功返回 TRUE，否则返回 FALSE。

输入变量	数据类型	描述
diSocket	DINT	由 SysSockCreate 返回的 socket 的描述符。
diLevel	DINT	明确的协议级别；可能的值：SOL_SOCKET, IPPROTO_TCP
diOption	DINT	想得到的当前选项名称；参考 SysSockSetOption 功能选项列表。
diOptionValue	DWORD	指向要写入选项当前值的变量的指针。
diOptionLength	DWORD	指向要写入选项当前值的变量的长度的指针。

SysSockGetLastError

这个功能块（INT 型）将调用操作系统的 getlasterror 操作，它将返回给定socket的最新发生的错误编号。

输入变量	数据类型	描述
diSocket	DINT	由 SysSockCreate 返回的 socket 的描述符。

SysSockHtons

这个功能块（DWORD 型）将调用操作系统的 htonl 功能，它将一个 u_long 值从主机字节次序转换到 TCP/IP 网络次序。这个功能块返回转换后的值。

输入变量	数据类型	描述
wHost	WORD	想要转换的值

SysSockInetAddr

这个功能块（BOOL 型）将调用Win32 系统的inet_ntoa 功能，它将转换一个因特网地址串为标准格式。如果操作成功返回 TRUE，否则返回 FALSE。

输入变量	数据类型	描述
stIPAddr	STRING	IP 地址(如 ’192.168.0.101’)

SysSockIoctl

这个功能块（DINT 型）将调用操作系统的 ioctl 功能，它可以控制 socket 的 I/O模块。如果操作成功返回 TRUE，否则返回 FALSE。

输入变量	数据类型	描述
diSocket	DINT	由 SysSockCreate 返回的 socket 的描述符。
diCommand	DINT	在 socket 上应用的命令；（相应的参数例如 Win32：cmd）有效命令：SOCKET_FIONBIO, SOCKET_FIONREAD
piParameter	DWORD	指向命令参数的指针。

SysSockListen

这个功能块（BOOL 型）将调用操作系统的 listen 功能。它将使 socket 倾听连接请求并把它们排成一队直到 SysSocketAccept 功能块接收它们。如果操作成功返回 TRUE。
当超过功能块的最大请求连接数时返回 FALSE。

输入变量	数据类型	描述
diSocket	DINT	由 SysSockCreate 返回的 socket 的描述符。
diMaxConnections	DINT	可放入 socket 输入队列的最大请求连接数。

SysSockNtohl

这个功能块(DWORD 型)将调用操作系统的 ntohl 功能 ,它将一个 u_long 值从 TCP/IP 网络次序转换到主机字节次序。
这个功能块返回转换后的主机字节次序。

输入变量	数据类型	描述
dwNet	DWORD	想要转换的 u_long 值

SysSockNtohs

这个功能块(WORD 型)将调用操作系统的 ntohs 功能 ,它将一个 u_short 值从 TCP/IP 网络次序转换到主机字节次序。
这个功能块返回转换后的主机字节次序。

输入变量	数据类型	描述
dwNet	WORD	想要转换的 u_short 值

SysSockSelect

这个功能块(DINT 型)将调用操作系统的 select 功能 ,它可以检查一个或几个sockets 是否已经准备好某些信息通讯。这个请求应用的 sockets 组可以通过结构体SOCKET_FD_SET 来定义。这个功能块将返回 select 功能的结果。

输入变量	数据类型	描述
diWidth	DINT	结构体 SOCKET_FD_SET 的大小
fdRead	DWORD	的指针。也可以用 0 来忽略。结构体 SOCKET_FD_SET 参考如下。
fdWrite	DWORD	随意一个指向定义 socket 设置选择写状态的结构体的指针。也可以用 0 来忽略。结构体 SOCKET_FD_SET 参考如下。
fdExcept	DWORD	随意一个指向定义 socket 设置选择错误状态的结构体的指针。也可以用 0 来忽略。结构体 SOCKET_FD_SET 参考如下。
ptvTimeout	DWORD	SysSockSelect 功能块等待响应的最长时间。结构体 SOCKET_TIMEVAL 参考如下。

SOCKET_FD_SET 结构体：

fd_count: UDINT; (*sockets 编号*)

fd_array:ARRAY[0..63] OF DINT (*socket 描述符指的区域*)

SOCKET_TIMEVAL 结构体：

tv_sec:DINT; (*秒*)

tv_usec:DINT; (*微秒*)

SysSockSetIPAddress

这个功能块被禁止使用

SysSockSetOption

这个功能块（BOOL 型）将调用操作系统的 setsockopt 功能，它来设置 socket 选项的细节。
如果操作成功返回 TRUE，否则返回 FALSE。

输入变量	数据类型	描述
diSocket	DINT	由 SysSockCreate 返回的 socket 的描述符。
diLevel	DINT	明确的协议级别
diOption	DINT	选项名称
diOptionValue	DWORD	选项值；通过设置‘0’解除布尔选项值，其它情况设置值。
diOptionLength	DWORD	保存选项值的缓冲器的大小

SysSockShutdown

这个功能块（BOOL 型）将调用操作系统的 shutdown 功能，它来禁止发送或接收的继续通讯。这个功能块不关闭 socket！必须通过 SysSockClose 功能块来关闭。
如果操作成功返回 TRUE，否则返回 FALSE。

输入变量	数据类型	描述
diSocket	DINT	由 SysSockCreate 返回的 socket 的描述符。
diHow	DINT	在这里定义哪种类型的通讯要被禁止

SysSockHtonl

这个功能块（DWORD 型）将调用操作系统的 htonl 功能，它将一个 u_long 值从主机字节次序转换到 TCP/IP 网络次序。
这个功能块返回转换后的值。

输入变量	数据类型	描述
wHost	WORD	想要转换的值

SysSockRecv

这个特殊的 TCP/IP 功能块（DINT 型）将调用操作系统（VxWorks 或 Win32）的 read 功能，它用来接收发向 socket 的数据。

这个功能块将返回读取的字节数。

输入变量	数据类型	描述
diSocket	DINT	由 SysSockCreate 返回的 socket 的描述符。
pbyBuffer	DWORD	要读取数据的缓冲器的地址
diBufferSize	DINT	要读取数据的缓冲器的大小
diFlags	DINT	根据 socket 选项来定义功能块的调用方式

SysSockSend

这个特殊的 TCP/IP 功能块（DINT 型）将调用操作系统的 send 功能，它可以发送存在 socket 缓冲器中的数据。

这个功能块将返回发送的字节数。如果 socket 已经“gracefully closed”，将返回 0，否则返回 1。

输入变量	数据类型	描述
diSocket	DINT	由 SysSockCreate 返回的 socket 的描述符。
pbyBuffer	DWORD	要发送数据的缓冲器的地址
diBufferSize	DINT	要发送的缓冲器的大小
diFlags	DINT	根据 socket 选项来定义功能块的调用方式

SysSockRecvFrom

这个特殊的 UDP 功能块（DINT 型）将调用操作系统的 recvfrom 功能，用它来读取发向 socket 的数据。

这个功能块将返回读取的字节数。如果 socket 已经“gracefully closed”，将返回 0，否则返回 1。

输入变量	数据类型	描述
diSocket	DINT	由 SysSockCreate 返回的 socket 的描述符。
pbyBuffer	DWORD	要接受的缓冲器的地址
diBufferSize	DINT	要接受冲器的大小
diFlags	DINT	根据 socket 选项来定义功能块的调用方式
pSockAddr	DWORD	指向 SOCKADDR 类型变量的指针
diSockAddize	DINT	结构体 SockAddr 的长度

SysSockSendTo

这个特殊的 UDP 功能块（DINT 型）将调用操作系统的 send 功能，用它来发送存储在 socket 的数据。这个功能块将返回读取的字节数。如果 socket 已经“gracefullyclosed”，将返回 0，否则返回 1。

输入变量	数据类型	描述
diSocket	DINT	由 SysSockCreate 返回的 socket 的描述符。
pbyBuffer	DWORD	要发送数据的缓冲器的地址
diBufferSize	DINT	要发送的缓冲器的大小
diFlags	DINT	根据 socket 选项来定义功能块的调用方式
pSockAddr	DWORD	指向 SOCKADDR 类型变量的指针

diSockAddize	DINT	结构体 SockAddr 的长度

详细的使用方法请参考示例程序SysLibSocketsTest.pro

2.20 SysLibSocketsAsync.lib

这个库像 SysLibSockets.lib 库一样提供同样的功能，只是功能块代替且执行过程是异步的。
请注意：
– 功能块的调用是针对明确目标的。
– 要是很多 socket 同时打开/关闭将需要很长的时间。
– 推荐在单独的低优先级任务下应用同步的功能块。
这个库的功能块与 SysLibSockets.lib 库相应的功能块具有同样影响效果的输入参数。
这个库的功能块与 SysLibSockets.lib 库相应的功能块具有同样影响效果的返回值。
参考2.19的各个功能块的描述信息。
另外的，对于所有的功能块下面的输入输出参数都是可用的。

输入变量	数据类型	描述
bEnable	BOOL	上升沿：这个功能块被激活
输出变量	数据类型	描述
bDone	BOOL	TRUE 显示这个功能块完成处理
bBusy	BOOL	TRUE 显示这个功能块正在处理
bError	BOOL	TRUE 显示这个功能块发生错误
wErrorId	WORD	错误编号

每个功能块将在 bEnable 处发现上升沿时开始各自的功能。接着它将被循环调用直到bDone =TRUE。然后输出变量 bError 和 wErrorId 将作为功能块特殊的输出参数来起作用。对于每一个 socket 将产生一个新的任务。另外，这还有一个不和 socket 任务并联的针对功能块的任务。功能块如下：
SysSockAcceptAsync
SysSockBindAsync
SysSockCloseAsync
SysSockConnectAsync
SysSockCreateAsync
SysSockGetHostByNameAsync
SysSockGetHostNameAsync
SysSockGetOptionAsync
SysSockHtonIAsync
SysSockHtonsAsync
SysSockInetAddrAsync
SysSockInetNtoaAsync
SysSockloctlAsync
SysSockListenAsync
SysSockNtohIAsync
SysSockNtohsAsync
SysSockSelectAsync
SysSockSetIPAddressAsync
SysSockSetOptionAsync
SysSockShutdown
TCP specific:
SysSockRecvAsync, SysSockSendAsync
UDP specific:
SysSockRecvFromAsync, SysSockSendToAsync

详细的使用方法请参考示例程序SysLibSocketsAsyncTest.pro

2.21 SysLibStr.lib

这个库提供字符串操作功能块。下面的功能块将可用。执行过程是同步的。

用来比较字符串：

SysStrCmp
SysStrCmpI
SysStrCmpN
SysStrCmpNI
SysStrCpy
SysStrLen

SysStrCmp

这个功能块（DINT 型）用来比较词典编纂式的两个字符串，并返回下面值中的一种：

返回值<0　　字符串 1 比字符串 2 小
返回值=0　　字符串 1=字符串 2
返回值>0　　字符串 1 比字符串 2 大

输入变量	数据类型	描述
dwString1	STRING	第一个字符串
dwString2	STRING	第二个字符串

SysStrCmpI

这个功能块（DINT 型）用来检查是否两个字符串相同，并返回下面值的一种：

返回值<0　　字符串 1 比字符串 2 小
返回值=0　　字符串 1=字符串 2
返回值>0　　字符串 1 比字符串 2 大

输入变量	数据类型	描述
dwString1	STRING	第一个字符串
dwString2	STRING	第二个字符串

SysStrCmpN

字符数的计算从字符串的开始来考虑，这个功能块（DINT 型）将通过定义的字符数来比较两个字符串的大小。返回下面值的一种：

返回值<0　　字符串 1 比字符串 2 小
返回值=0　　字符串 1=字符串 2
返回值>0　　字符串 1 比字符串 2 大

输入变量	数据类型	描述
dwString1	STRING	第一个字符串
dwString2	STRING	第二个字符串
diChars	DINT	从字符串的开始计数的位置编号 ,字符串的大小将用来被比较

SysStrCmpNI

这个功能块（DINT 型）将检查两个字符串定义的字符数是否相同（从字符串的开始开始计算）。返回下面值的一种：

返回值<0　　字符串 1 比字符串 2 小
返回值=0　　字符串 1=字符串 2
返回值>0　　字符串 1 比字符串 2 大

输入变量	数据类型	描述
dwString1	STRING	第一个字符串
dwString2	STRING	第二个字符串
diChars	DINT	从字符串的开始计数的位置编号 ,将被检查在两个字符串中是否相同

SysStrCpy

这个功能块（DWORD 型）可以用来拷贝一个字符串到另一个字符串。他将返回一个指向目标字符串的指针。

输入变量	数据类型	描述
dwString1	STRING	想要拷贝到的字符串（目标字符串）
dwString2	STRING	想要被拷贝的字符串（源字符串）

SysStrLen

这个功能块（DINT 型）可用来得到一个字符串的长度。它将返回一个字符数，不包括字符串的结束符（NULL）。

输入变量	数据类型	描述
dwString1	STRING	想要计算长度的字符串

2.22 SysLibSymbols.lib

这个库使用户可以获取到IEC程序中得符号（局部变量、全局变量）的物理地址。下面的功能块将可用。执行过程是同步的。

SysLibGetSymbolAddress

SysLibGetSymbolAddress

这个功能块返回指定符号变量的物理地址。这些变量必须用完成的名称，例如PLC_PRG.a or ‚.global_var'
用户依然可以通过ADR（PLC_PRG.a）获取到地址，但是与前者的区别是：后者必须在IEC工程编译时，添加该变量到符号表中。否则ADR无法获取到地址。而前者则无需添加。
如果该功能块返回为0，表示无法找到指定符号的地址或者指定符号不存在。

输入变量	数据类型	描述
pszSymbol	STRING	符号变量名

例如：

```
0001 PROGRAM PLC_PRG
0002 VAR
0003     a: INT;
0004     dwAdr1: DWORD;
0005     dwAdr2: DWORD;
0006 END_VAR
0007
```

```
0001 (* Be sure to
0002 - Dump symbol entries (/Project/Options/Symbol File)
0003 - Download symbol file (target settings )
0004 *)
0005 a:=a+1;
0006 (*Address from symbol management *)
0007 dwAdr1 := SysLibGetSymbolAddress('PLC_PRG.a');
0008 (* Address from ASDR-operator *)
0009 dwAdr2 := ADR(PLC_PRG.a);
0010 (* Both values must be the same! *)
```

固高科技（深圳）有限公司

地　　址：深圳市高新技术产业园南区深港产学研基地西座二层 W211 室

电　　话：0755-26970823　26970817　26970824

传　　真：0755-26970846

电子邮件：support@googoltech.com

网　　址：http://www.googoltech.com.cn

固高科技（香港）有限公司

地　　址：香港九龙清水湾香港科技大学新翼楼 3639 室

电　　话：(852) 2358-1033

传　　真：(852) 2358-4931

电子邮件：info@googoltech.com

网　　址：http://www.googoltech.com/

OtoStudio 高级运动控制编程手册

（2011-12-06）

版权申明

运动中的机器有危险！使用者有责任在机器中设计有效的出错处理和安全保护机制，固高科技没有义务或责任对由此造成的附带的或相应产生的损失负责。

文档版本

版本号	修订日期
1.0	2010 年 09 月 07 日
2.01	2010 年 09 月 08 日
2.03	2011 年 7 月 22 日
2.2	2011 年 12 月 6 日

目录

CPAC-OtoStudio 高级运动控制库使用手册

第一章 OtoStudio 中运动函数库的使用

1.1 OtoStudio 软件库的使用

在 CPAC 软件平台下使用运动控制器的高级功能库，方法与使用 CPAC-X00-TPX.lib 方法一致。可同时使用 CPAC-X00-TPX.lib。CPAC-OtoBox 控制器的库文件名为 CPAC GUC_X00_TPX-Addition 2.2.lib。

1 OtoStudio 平台中库的使用

4. 启动OtoStudio.exe，新建一个工程；
5. 系统自动添加CPAC GUC-X00-TPX.lib
6. 手动在库文件管理器中添加 CPAC GUC-X00-TPX-Addition 2.2.lib

至此，用户就可以在 OtoStudio 中调用函数库中的任何函数，开始编写应用程序。

第二章 命令返回值及其意义

2.1 指令返回值

CPAC 控制器按照主机发送的指令工作。CPAC 控制器指令封装在动态链接库中。用户在编写应用程序时，通过调用 CPAC 控制器 中运动控制库 GUC-X00-TPX-Addition 2.2.lib 指令来操纵 CPAC 控制器的运动控制。

CPAC 控制器在接收到主机发送的指令时，将执行结果反馈到主机，指示当前指令是否正确执行。指令返回值的定义如下。

运动控制器指令返回值定义

返回值	意义	处理方法
0	指令执行成功	
1	指令执行错误	2. 检查当前指令的执行条件是否满足
7	指令参数错误	1. 检查当前指令输入参数的取值
2	license 不支持	1. 如果需要此功能，请与生产厂商联系。
-1	主机和运动控制器通讯失败	5. 是否正确安装运动控制器驱动程序 6. 检查运动控制器是否接插牢靠 7. 更换主机 8. 更换控制器
-6	打开控制器失败	4. 是否正确安装运动控制器驱动程序 5. 是否调用了 2 次 GT_Open 指令 6. 其他程序是否已经打开运动控制器
-7	运动控制器没有响应	2. 更换运动控制器

注意	建议在用户程序中，检测每条指令的返回值，以判断指令的执行状态。并建立必要的错误处理机制，保证程序安全可靠地运行。

第三章 系统配置

系统配置除使用 OtoStudio 中的配置工具外，还支持使用 CPAC GUC-X00-TPX-Addition 2.2.lib 中的指令来在程序运行中配置。

3.1 配置信息修改指令

用户除了可以使用上面所述的配置文件的方式实现运动控制器的初始化配置外，还可以使用指令的方式来实现初始化配置。

配置信息指令列表

指令	说明
GT_AlarmOff	控制轴驱动报警信号无效
GT_AlarmOn	控制轴驱动报警信号有效
GT_LmtsOn	控制轴限位信号有效
GT_LmtsOff	控制轴限位信号无效
GT_ProfileScale	设置控制轴的规划器当量变换值
GT_EncScale	设置控制轴的编码器当量变换值
GT_StepDir	将脉冲输出通道的脉冲输出模式设置为“脉冲+方向”
GT_StepPulse	将脉冲输出通道的脉冲输出模式设置为“CW/CCW”
GT_SetMtrBias	设置模拟量输出通道的零漂电压补偿值
GT_GetMtrBias	读取模拟量输出通道的零漂电压补偿值
GT_SetMtrLmt	设置模拟量输出通道的输出电压饱和极限值
GT_GetMtrLmt	读取模拟量输出通道的输出电压饱和极限值
GT_EncSns	设置编码器的计数方向
GT_EncOn	设置为“外部编码器”计数方式
GT_EncOff	设置为“脉冲计数器”计数方式
GT_SetPosErr	设置跟随误差极限值
GT_GetPosErr	读取跟随误差极限值

GT_SetStopDec	设置平滑停止减速度和急停减速度
GT_GetStopDec	读取平滑停止减速度和急停减速度
GT_LmtSns	设置运动控制器各轴限位开关触发电平
GT_CtrlMode	设置控制轴为模拟量输出或脉冲输出
GT_SetStopIo	设置平滑停止和紧急停止数字量输入的信息
GT_GpiSns	设置运动控制器数字量输入的电平逻辑
GT_SetAdcFilter	设置模拟量输入的滤波器时间参数(仅适用于 GTS-400-PX 控制器)

GT_AlarmOff

控制轴驱动报警信号无效

GT_AlarmOff(axis)	
Axis:INT	控制轴号

GT_AlarmOn

控制轴驱动报警信号有效

GT_AlarmOn(axis)	
Axis:INT	控制轴号

GT_LmtsOn

控制轴限位信号有效

GT_LmtsOn(axis, limitType)	
Axis:INT	控制轴号
LimitType:INT	需要有效的限位类型 MC_LIMIT_POSITIVE：需要将该轴的正限位有效 MC_LIMIT_NEGATIVE：需要将该轴的负限位有效 -1：需要将该轴的正限位和负限位都有效，默认为该值

GT_LmtsOff

控制轴限位信号无效

GT_LmtsOff(axis, limitType)	
Axis:INT	控制轴号
LimitType:INT	需要无效的限位类型 MC_LIMIT_POSITIVE：需要将该轴的正限位无效 MC_LIMIT_NEGATIVE：需要将该轴的负限位无效 -1：需要将该轴的正限位和负限位都无效，默认为该值

GT_ProfileScale

设置控制轴的规划器当量变换值

GT_ProfileScale(axis, alpha, beta)	
Axis:INT	控制轴号
Alpha:INT	规划器当量的 alpha 值，取值范围：[-32768,32767]，请参见 ii.1
Beta:INT	规划器当量的 beta 值，取值范围：[-32768,32767]，请参见 ii.1

GT_EncScale

设置控制轴的编码器当量变换值

GT_EncScale(axis, alpha, beta)	
Axis:INT	控制轴号
Alpha:INT	编码器当量的 alpha 值，取值范围：[-32768,32767]，请参见 ii.1
Beta:INT	编码器当量的 beta 值，取值范围：[-32768,32767]，请参加 ii.1

GT_StepDir

将脉冲输出通道的脉冲输出模式设置为“脉冲+方向”

GT_StepDir(step)

Step:INT	脉冲输出通道号

GT_StepPulse

将脉冲输出通道的脉冲输出模式设置为“CW/CCW”

GT_StepPulse(step)	
Step:INT	脉冲输出通道号

GT_SetMtrBias

设置模拟量输出通道的零漂电压补偿值

GT_SetMtrBias(dac, bias)	
Dac:INT	模拟量输出通道号
Bias:INT	零漂补偿值，取值范围：[-32768,32767]

GT_GetMtrBias

读取模拟量输出通道的零漂电压补偿值

GT_GetMtrBias(dac, pBias)	
Dac:INT	模拟量输出通道号
pBias:POINTER TO INT	读取的零漂补偿值

GT_SetMtrLmt

设置模拟量输出通道的输出电压饱和极限值

GT_SetMtrLmt(dac, limit)	
Dac:INT	模拟量输出通道号
iLimit:INT	输出电压饱和极限值，取值范围：(0,32767]

GT_GetMtrLmt

读取模拟量输出通道的输出电压饱和极限值

GT_GetMtrLmt(dac, pLimit)	
Dac:INT	模拟量输出通道号
PLimit:POINTER TO INT	读取的输出电压饱和极限值

GT_EncSns

设置编码器的计数方向

GT_EncSns(sense)	
Sense:UINT	按位标识编码器的计数方向，bit0~bit7 依次对应编码器 1~8， bit8 对应辅助编码器 0：该编码器计数方向不取反 1：该编码器计数方向取反 请参见 3.2.4

GT_EncOn

设置为“外部编码器”计数方式

GT_EncOn(encoder)	
Encoder:INT	编码器通道号

GT_EncOff

设置为“脉冲计数器”计数方式

GT_EncOff(encoder)	
Encoder:INT	编码器通道号

GT_SetPosErr

设置跟随误差极限值

GT_SetPosErr(control, error)	
Control:INT	伺服控制器编号
Error:DINT	跟随误差极限值，取值范围：(0, 2147483648]

GT_GetPosErr

读取跟随误差极限值

GT_GetPosErr(control, pError)	
Control:INT	伺服控制器编号
PError:POINTER TO DINT	读取的跟随误差极限值

GT_SetStopDec

设置平滑停止减速度和急停减速度

GT_SetStopDec(profile, decSmoothStop, decAbruptStop)	
Profile:INT	规划器的编号
DecSmoothStop:LREAL	平滑停止减速度，取值范围：(0,32767]
DecAbruptStop:LREAL	急停减速度，取值范围：(0,32767]

GT_GetStopDec

读取平滑停止减速度和急停减速度

GT_GetStopDec(profile, pDecSmoothStop, pDecAbruptStop)	
Profile:INT	规划器的编号
PDecSmoothStop:POINTER TO LREAL	读取的平滑停止减速度
PDecAbruptStop:POINTER	读取的急停减速度

TO LREAL	

GT_LmtSns

设置运动控制器各轴限位开关触发电平

GT_LmtSns(sense)	
Sense:UINT	按位标识轴的限位的触发电平状态，具体请参加重点说明

GT_CtrlMode

设置控制轴为模拟量输出或脉冲输出

GT_CtrlMode(axis, mode)	
Axis:INT	控制轴号
Mode:INT	切换的模式 0：将指定轴切换为闭环控制模式(电压控制方式) 1：将指定轴切换为开环控制模式(脉冲控制方式)

GT_SetStopIo

设置平滑停止和紧急停止数字量输入的信息

GT_SetStopIo(axis, stopType,inputType, inputIndex)	
Axis:INT	需要设置停止 IO 信息的轴的编号，取值范围：[1,8]
StopType:INT	需要设置停止 IO 信息的停止类型 0：紧急停止类型 1：平滑停止类型
InputType:INT	设置的数字量输入的类型 MC_LIMIT_POSITIVE(该宏定义为 0)　正限位 MC_LIMIT_NEGATIVE(该宏定义为 1)　负限位 MC_ALARM(该宏定义为 2)　驱动报警

	MC_HOME(该宏定义为 3) 原点开关 MC_GPI(该宏定义为 4) 通用输入 MC_ARRIVE(该宏定义为 5) 电机到位信号(仅适用于 GTS-400-PX 控制器)
InputIndex:INT	设置的数字量输入的索引号，取值范围根据 inputType 的取值而定 当 inputType= MC_LIMIT_POSITIVE 时，取值范围：[1,8] 当 inputType= MC_LIMIT_NEGATIVE 时，取值范围：[1,8] 当 inputType= MC_ALARM 时，取值范围：[1,8] 当 inputType= MC_HOME 时，取值范围：[1,8] 当 inputType= MC_GPI 时，取值范围：[1,16] 当 inputType= MC_ARRIVE 时，取值范围：[1,8]

GT_GpiSns

设置运动控制器数字量输入的电平逻辑

GT_GpiSns(sense)	
Sense:UINT	按位表示各数量输入的电平逻辑，从 bit0~bit15，分别对应数字量输入 1 到 16。 0：输入电平不取反，通过 GT_GetDi()指令读取到 0 表示输入低电平，通过 GT_GetDi()指令读取到 1 表示输入高电平； 1：输入电平取反，通过 GT_GetDi()指令读取到 0 表示输入高电平，通过 GT_GetDi()指令读取到 1 表示输入低电平；

GT_SetAdcFilter

设置模拟量输入的滤波器时间参数(仅适用于 CPAC-OtoBox-UC*-4**控制器)

GT_SetAdcFilter(adc, filterTime)	
Sdc:INT	数字量输入的编号，取值范围：[1,8]
FilterTime:INT	数字量输入信号的滤波器时间参数，取值范围：[0,50]

3.2 重点说明

编码器计数方向设置

指令 GT_EncSns()可以修改运动控制器各编码器的计数方向，当指令参数的某个状态位为 1 时，将所对应的控制轴的编码器计数方向取反，指令参数的状态位定义如下表所示：

状态位	8	7	6	5	4	3	2	1	0
编码器	辅助编码器	Enc8	Enc7	Enc6	Enc5	Enc4	Enc3	Enc2	Enc1

设置限位开关触发电平

运动控制器默认的限位开关为常闭开关，即各轴处于正常工作状态时，其限位开关信号输入为低电平；当限位开关信号输入为高电平时，与其对应轴的限位状态将被触发。如果使用常开开关，需要通过调用指令 GT_LmtSns()改变限位开关触发电平。

指令 GT_LmtSns()的参数设置各轴正负限位开关的触发电平，当该参数的某个状态位为 0 时，表示将对应的限位开光设置为高电平触发，当某个状态位为 1 时，表示将对应的限位开关设置为低电平触发。指令参数和各轴限位的对应关系如下所示：

状态位	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
限位开关	轴 8		轴 7		轴 6		轴 5		轴 4		轴 3		轴 2		轴 1	
	—	+	—	+	—	+	—	+	—	+	—	+	—	+	—	+

第四章 运动模式

除 GUC-800-TPX 控制器每个轴可以独立工作在点位、Jog、PT、电子齿轮或 Follow 运动模式下外，本章主要介绍 CPAC GUC-X00-TPX-Addition 2.0.lib 实现 PVT 运动模式，以及插

补运动。

4.1 插补运动模式

插补运动模式可以实现多轴的协调运动，从而完成一定的运动轨迹。该插补运动模式具有以下一些功能，可以实现直线插补和圆弧插补；可以同时有两个坐标系进行插补运动；每个坐标系含有两个缓存区，可以实现缓存区暂停、恢复等功能；具有缓存区延时和缓存区数字量输出的功能；具有前瞻预处理功能，能够实现小线段高速平滑的连续轨迹运动。

4.1.1 指令列表

设置插补运动指令列表

指令	说明
GT_SetCrdPrm	设置坐标系参数，确立坐标系映射，建立坐标系
GT_GetCrdPrm	查询坐标系参数
GT_CrdData	向插补缓存区增加插补数据
GT_LnXY	缓存区指令，两维直线插补
GT_LnXYZ	缓存区指令，三维直线插补
GT_LnXYZA	缓存区指令，四维直线插补
GT_LnXYG0	缓存区指令，两维直线插补(终点速度始终为 0)
GT_LnXYZG0	缓存区指令，三维直线插补(终点速度始终为 0)
GT_LnXYZAG0	缓存区指令，四维直线插补(终点速度始终为 0)
GT_ArcXYR	缓存区指令，XY 平面圆弧插补(以终点位置和半径为输入参数)
GT_ArcXYC	缓存区指令，XY 平面圆弧插补(以终点位置和圆心位置为输入参数)
GT_ArcYZR	缓存区指令，YZ 平面圆弧插补(以终点位置和半径为输入参数)
GT_ArcYZC	缓存区指令，YZ 平面圆弧插补(以终点位置和圆心位置为输入参数)
GT_ArcZXR	缓存区指令，ZX 平面圆弧插补(以终点位置和半径为输入参数)
GT_ArcZXC	缓存区指令，ZX 平面圆弧插补(以终点位置和圆心位置为输入

	参数)
GT_BufIO	缓存区指令，缓存区内数字量 IO 输出设置指令
GT_BufDelay	缓存区指令，缓存区内延时设置指令
GT_BufDA	缓存区指令，缓存区内输出 DA 值
GT_BufLmtsOn	缓存区指令，缓存区内有效限位开关
GT_BufLmtsOff	缓存区指令，缓存区内无效限位开关
GT_BufSetStopIo	缓存区指令，缓存区内设置 axis 的停止 IO 信息
GT_BufMove	缓存区指令，实现刀向跟随功能，启动某个轴点位运动
GT_BufGear	缓存区指令，实现刀向跟随功能，启动某个轴跟随运动
GT_SetUserSegNum	缓存区指令，设置自定义插补段段号
GT_GetUserSegNum	读取自定义插补段段号
GT_GetRemainderSegNum	读取未完成的插补段段数
GT_CrdSpace	查询插补缓存区剩余空间
GT_CrdClear	清除插补缓存区内的插补数据
GT_CrdStart	启动插补运动
GT_CrdStatus	查询插补运动坐标系状态
GT_SetOverride	设置插补运动目标合成速度倍率
GT_SetCrdStopDec	设置插补运动平滑停止、急停合成加速度
GT_GetCrdStopDec	查询插补运动平滑停止、急停合成加速度
GT_GetCrdPos	查询该坐标系的当前坐标位置值
GT_GetCrdVel	查询该坐标系的合成速度值
GT_InitLookAhead	初始化插补前瞻缓存区

GT_SetCrdPrm

设置坐标系参数，确立坐标系映射，建立坐标系

GT_SetCrdPrm(crd, pCrdPrm)	
Crd:INT	坐标系号，取值范围：[1,2]
PCrdPrm:POINTER TO	设置坐标系的相关参数

TCrdPrm	TYPE TCrdPrm : STRUCT dimension:INT; profile:ARRAY[0..7] OF INT; synVelMax:LREAL; synAccMax:LREAL; evenTime:INT; setOriginFlag:INT; originPos:ARRAY[0..7] OF DINT; END_STRUCT END_TYPE dimension：坐标系的维数，取值范围：[1,4]。 Profile[8]：坐标系与规划器的映射关系，每个元素的取值范围：[0,4]。 synVelMax：该坐标系的最大合成速度。取值范围：(0,32767)，单位：pulse/ms。 synAccMax：该坐标系的最大合成加速度。取值范围：(0,32767)，单位：pulse/(ms*ms)。 evenTime：每个插补段的最小匀速段时间。取值范围：[0,32767)，单位：ms。 setOriginFlag：表示是否需要指定坐标系的原点坐标的规划位置，0：不需要指定原点坐标值，则坐标系的原点在当前规划位置上；1：需要指定原点坐标值，坐标系的原点在 originPos 指定的规划位置上。 originPos[8]：指定的坐标系原点的规划位置值

GT_GetCrdPrm

查询坐标系参数

GT_GetCrdPrm(crd, pCrdPrm)	
Crd:INT	坐标系号，取值范围：[1,2]
PCrdPrm:POINTER TO TCrdPrm	读取坐标系的相关参数 结构体的成员含义参照 GT_SetCrdPrm

GT_CrdData

向插补缓存区增加插补数据

GT_CrdData(crd, pCrdData, fifo)	
Crd:INT	坐标系号，取值范围：[1,2]
PCrdData:POINTER TO TCrdData	插补数据 TYPE TCrdData : STRUCT motionType:INT; circlePlat:INT; pos:ARRAY[0..3] OF DINT; radius:DINT; circleDir:INT; center:ARRAY[0..1] OF DINT; vel:LREAL; acc:LREAL; velEndZero:INT; operation:TCrdBufOperation; cosI:ARRAY[0..3] OF LREAL; velEnd:LREAL; velEndAdjust:LREAL; SchyR:LREAL; END_STRUCT END_TYPE

	pos：该插补段终点坐标位置值。取值范围：[-1073741823, 1073741823]，单位：pulse。 vel：该插补段的目标速度。取值范围：(0,32767)，单位：pulse/ms。 acc：该插补段的加速度。取值范围：(0,32767)，单位：pulse/(ms*ms)。 velEndZero：表示该插补段的终点速度是否强制为 0。 operation：缓存区内的延时和数字量输出操作。 cos：插补模块内部使用变量。 velEnd：该插补段的终点速度，取值范围：[0,32767)，单位：pulse/ms。 velEndAdjust：插补模块内部使用变量。 r：插补模块内部使用变量。 其他成员变量：均为内部使用。 TYPE TCrdBufOperation : STRUCT flag:INT; delay:UINT; doType:INT; doMask:UINT; doValue:UINT; dataExt:ARRAY[0..1] OF UINT; END_STRUCT END_TYPE flag：表示该插补段是否含有延时或者数字量输出操作，取值范围：[0,1]，0：该插补段没有延时或者数字量输出操作；1：该插补段含有延时或者数字量输出操作。 delay：延时操作的延时时间，取值范围：[0,16383]，单位：ms。 doType：数字量输出的类型： 0：没有数字量输出的操作。

	MC_ENABLE：输出驱动器使能。 MC_CLEAR：输出驱动器报警清除。 MC_GPO：输出通用输出。 doMask：从 bit0~bit15 按位表示指定的数字量输出是否有操作，0：该路数字量输出无操作；1：该路数字量输出有操作。 doValue：从 bit0~bit15 按位表示指定的数字量输出的值。 dataExt：内部使用变量。
Fifo:INT	插补缓存区号，取值范围：[0,1]，默认为：0

GT_LnXY

缓存区指令，两维直线插补

GT_LnXY(crd, x, y, synVel, synAcc, velEnd, fifo)	
Crd:INT	坐标系号，取值范围：[1,2]
X:DINT	插补段 x 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
Y:DINT	插补段 y 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
SynVel:LREAL	插补段的目标合成速度。取值范围：(0,32767)，单位：pulse/ms。
SynAcc:LREAL	插补段的合成加速度。取值范围：(0,32767)，单位：pulse/(ms*ms)。
VelEnd:LREAL	插补段的终点速度。取值范围：[0,32767)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认为：0
Fifo:INT	插补缓存区号，取值范围：[0,1]，默认为：0

GT_LnXYZ

缓存区指令，三维直线插补

GT_LnXYZ(crd, x, y, z, synVel, synAcc, velEnd, fifo)	
Crd:INT	坐标系号，取值范围：[1,2]
X:DINT	插补段 x 轴终点坐标值。取值范围:[-1073741823, 1073741823]，单位：pulse。
Y:DINT	插补段 y 轴终点坐标值。取值范围:[-1073741823, 1073741823]，单位：pulse。
Z:DINT	插补段 z 轴终点坐标值。取值范围:[-1073741823, 1073741823]，单位：pulse。
SynVel:LREAL	插补段的目标合成速度。取值范围：(0,32767)，单位：pulse/ms。
SynAcc:LREAL	插补段的合成加速度。取值范围：(0,32767)，单位：pulse/(ms*ms)。
VelEnd:LREAL	插补段的终点速度。取值范围：[0,32767)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认为：0
Fifo:INT	插补缓存区号，取值范围：[0,1]，默认为：0

GT_LnXYZA

缓存区指令，四维直线插补

GT_LnXYZA(crd, x, y, z, a, synVel, synAcc, velEnd, fifo)	
Crd:INT	坐标系号，取值范围：[1,2]
X:DINT	插补段 x 轴终点坐标值。取值范围:[-1073741823, 1073741823]，单位：pulse。
Y:DINT	插补段 y 轴终点坐标值。取值范围:[-1073741823, 1073741823]，单位：pulse。
Z:DINT	插补段 z 轴终点坐标值。取值范围:[-1073741823, 1073741823]，单位：pulse。
A:DINT	插补段 a 轴终点坐标值。取值范围:[-1073741823, 1073741823]，单位：pulse。

SynVel:LREAL	插补段的目标合成速度。取值范围：(0,32767)，单位：pulse/ms。
SynAcc:LREAL	插补段的合成加速度。取值范围：(0,32767)，单位：pulse/(ms*ms)。
VelEnd:LREAL	插补段的终点速度。取值范围：[0,32767)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认为：0
Fifo:INT	插补缓存区号，取值范围：[0,1]，默认为：0

GT_LnXYG0

缓存区指令，两维直线插补(终点速度始终为 0)

GT_LnXYG0(crd, x, y, synVel, synAcc, fifo)	
Crd:INT	坐标系号，取值范围：[1,2]
X:DINT	插补段 x 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
Y:DINT	插补段 y 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
SynVel:LREAL	插补段的目标合成速度。取值范围：(0,32767)，单位：pulse/ms。
SynAcc:LREAL	插补段的合成加速度。取值范围：(0,32767)，单位：pulse/(ms*ms)。
Fifo:INT	插补缓存区号，取值范围：[0,1]，默认为：0

GT_LnXYZG0

缓存区指令，三维直线插补(终点速度始终为 0)

GT_LnXYZG0(crd, x, y, z, synVel, synAcc, fifo)	
Crd:INT	坐标系号，取值范围：[1,2]
X:DINT	插补段 x 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。

Y:DINT	插补段 y 轴终点坐标值。取值范围:[-1073741823, 1073741823]，单位：pulse。
Z:DINT	插补段 z 轴终点坐标值。取值范围:[-1073741823, 1073741823]，单位：pulse。
SynVel:LREAL	插补段的目标合成速度。取值范围：(0,32767)，单位：pulse/ms。
SynAcc:LREAL	插补段的合成加速度。取值范围：(0,32767)，单位：pulse/(ms*ms)。
Fifo:INT	插补缓存区号，取值范围：[0,1]，默认为：0

GT_LnXYZG0

缓存区指令，四维直线插补(终点速度始终为 0)

GT_LnXYZAG0(crd, x, y, z, a, synVel, synAcc, fifo)	
Crd:INT	坐标系号，取值范围：[1,2]
X:DINT	插补段 x 轴终点坐标值。取值范围:[-1073741823, 1073741823]，单位：pulse。
Y:DINT	插补段 y 轴终点坐标值。取值范围:[-1073741823, 1073741823]，单位：pulse。
Z:DINT	插补段 z 轴终点坐标值。取值范围:[-1073741823, 1073741823]，单位：pulse。
A:DINT	插补段 a 轴终点坐标值。取值范围:[-1073741823, 1073741823]，单位：pulse。
SynVel:LREAL	插补段的目标合成速度。取值范围：(0,32767)，单位：pulse/ms。
SynAcc:LREAL	插补段的合成加速度。取值范围：(0,32767)，单位：pulse/(ms*ms)。
Fifo:INT	插补缓存区号，取值范围：[0,1]，默认为：0

GT_ArcXYR

缓存区指令，XY 平面圆弧插补(以终点位置和半径为输入参数)

GT_ArcXYR(crd, x, y, radius, circleDir, synVel, synAcc, velEnd, fifo)	
Crd:INT	坐标系号，取值范围：[1,2]
X:DINT	圆弧插补 x 轴的终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
Y:DINT	圆弧插补 y 轴的终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
Radius:LREAL	圆弧插补的圆弧半径值。取值范围：[-1073741823, 1073741823]，单位：pulse。 半径为正时，表示圆弧为小于等于 180° 圆弧 半径为负时，表示圆弧为大于 180° 圆弧 半径描述方式不能用来描述整圆
CircleDir:INT	圆弧的旋转方向 0：顺时针圆弧 1：逆时针圆弧
SynVel:LREAL	插补段的目标合成速度。取值范围：(0,32767)，单位：pulse/ms。
SynAcc:LREAL	插补段的合成加速度。取值范围：(0,32767)，单位：pulse/(ms*ms)。
VelEnd:LREAL	插补段的终点速度。取值范围：[0,32767)，单位：pulse/ms。 该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认为：0
Fifo:INT	插补缓存区号，取值范围：[0,1]，默认为：0

GT_ArcXYC

缓存区指令，XY 平面圆弧插补(以终点位置和圆心位置为输入参数)

GT_ArcXYC(crd, x, y, xCenter, yCenter, circleDir, synVel, synAcc, velEnd, fifo)	
Crd:INT	坐标系号，取值范围：[1,2]

X:DINT	圆弧插补 x 轴的终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
Y:DINT	圆弧插补 y 轴的终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
xCenter:LREAL	圆弧插补的圆心相对于起点位置的偏移量（x 方向）
yCenter:LREAL	圆弧插补的圆心相对于起点位置的偏移量（y 方向）
CircleDir:INT	圆弧的旋转方向 0：顺时针圆弧 1：逆时针圆弧
SynVel:LREAL	插补段的目标合成速度。取值范围：(0,32767)，单位：pulse/ms。
SynAcc:LREAL	插补段的合成加速度。取值范围：(0,32767)，单位：pulse/(ms*ms)。
VelEnd:LREAL	插补段的终点速度。取值范围：[0,32767)，单位：pulse/ms。 该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认为：0
Fifo:INT	插补缓存区号，取值范围：[0,1]，默认为：0

GT_ArcYZR

缓存区指令，YZ 平面圆弧插补(以终点位置和半径为输入参数)

GT_ArcYZR(crd, y, z, radius, circleDir, synVel, synAcc, velEnd, fifo)	
Crd:INT	坐标系号，取值范围：[1,2]
Y:DINT	圆弧插补 y 轴的终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
Z:DINT	圆弧插补 z 轴的终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
Radius:LREAL	圆弧插补的圆弧半径值。取值范围：[-1073741823, 1073741823]，单位：pulse。 半径为正时，表示圆弧为小于等于 180° 圆弧

	半径为负时，表示圆弧为大于 180° 圆弧 半径描述方式不能用来描述整圆
CircleDir:INT	圆弧的旋转方向 0：顺时针圆弧 1：逆时针圆弧
SynVel:LREAL	插补段的目标合成速度。取值范围：(0,32767)，单位：pulse/ms。
SynAcc:LREAL	插补段的合成加速度。取值范围：(0,32767)，单位：pulse/(ms*ms)。
VelEnd:LREAL	插补段的终点速度。取值范围：[0,32767)，单位：pulse/ms。 该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认为：0
Fifo:INT	插补缓存区号，取值范围：[0,1]，默认为：0

GT_ArcYZC

缓存区指令，YZ 平面圆弧插补(以终点位置和圆心位置为输入参数)

GT_ArcYZC(crd, y, z, yCenter, zCenter, circleDir, synVel, synAcc, velEnd, fifo)	
Crd:INT	坐标系号，取值范围：[1,2]
Y:DINT	圆弧插补 y 轴的终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
Z:DINT	圆弧插补 z 轴的终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
yCenter:LREAL	圆弧插补的圆心相对于起点位置的偏移量（x 方向）
zCenter:LREAL	圆弧插补的圆心相对于起点位置的偏移量（y 方向）
CircleDir:INT	圆弧的旋转方向 0：顺时针圆弧 1：逆时针圆弧
SynVel:LREAL	插补段的目标合成速度。取值范围：(0,32767)，单位：pulse/ms。
SynAcc:LREAL	插补段的合成加速度。取值范围：(0,32767)，单位：

	pulse/(ms*ms)。
VelEnd:LREAL	插补段的终点速度。取值范围：[0,32767)，单位：pulse/ms。 该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认为：0
Fifo:INT	插补缓存区号，取值范围：[0,1]，默认为：0

GT_ArcZXR

缓存区指令，ZX 平面圆弧插补(以终点位置和半径为输入参数)

GT_ArcZXR(crd, z, x, radius, circleDir, synVel, synAcc, velEnd, fifo)	
Crd:INT	坐标系号，取值范围：[1,2]
Z:DINT	圆弧插补 z 轴的终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
X:DINT	圆弧插补 x 轴的终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
Radius:LREAL	圆弧插补的圆弧半径值。取值范围：[-1073741823, 1073741823]，单位：pulse。 半径为正时，表示圆弧为小于等于 180° 圆弧 半径为负时，表示圆弧为大于 180° 圆弧 半径描述方式不能用来描述整圆
CircleDir:INT	圆弧的旋转方向 0：顺时针圆弧 1：逆时针圆弧
SynVel:LREAL	插补段的目标合成速度。取值范围：(0,32767)，单位：pulse/ms。
SynAcc:LREAL	插补段的合成加速度。取值范围：(0,32767)，单位：pulse/(ms*ms)。
VelEnd:LREAL	插补段的终点速度。取值范围：[0,32767)，单位：pulse/ms。 该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认为：0

Fifo:INT	插补缓存区号，取值范围：[0,1]，默认为：0

GT_ArcZXC

缓存区指令，ZX 平面圆弧插补(以终点位置和圆心位置为输入参数)

GT_ArcZXC(crd, z, x, zCenter, xCenter, circleDir, synVel, synAcc, velEnd, fifo)	
Crd:INT	坐标系号，取值范围：[1,2]
Z:DINT	圆弧插补 z 轴的终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
X:DINT	圆弧插补 x 轴的终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
zCenter:LREAL	圆弧插补的圆心相对于起点位置的偏移量（x 方向）
xCenter:LREAL	圆弧插补的圆心相对于起点位置的偏移量（y 方向）
CircleDir:INT	圆弧的旋转方向 0：顺时针圆弧 1：逆时针圆弧
SynVel:LREAL	插补段的目标合成速度。取值范围：(0,32767)，单位：pulse/ms。
SynAcc:LREAL	插补段的合成加速度。取值范围：(0,32767)，单位：pulse/(ms*ms)。
VelEnd:LREAL	插补段的终点速度。取值范围：[0,32767)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认为：0
Fifo:INT	插补缓存区号，取值范围：[0,1]，默认为：0

GT_BufIO

缓存区指令，缓存区内数字量 IO 输出设置指令

GT_BufIO(crd, doType, doMask, doValue, fifo)	
Crd:INT	坐标系号，取值范围：[1,2]

DoType:UINT	数字量输出的类型： MC_ENABLE：输出驱动器使能。 MC_CLEAR：输出驱动器报警清除。 MC_GPO：输出通用输出。
DoMask:UINT	从 bit0~bit15 按位表示指定的数字量输出是否有操作，0：该路数字量输出无操作；1：该路数字量输出有操作。
DoValue:UINT	从 bit0~bit15 按位表示指定的数字量输出的值
Fifo:INT	插补缓存区号，取值范围：[0,1]，默认为：0

GT_BufDelay

缓存区指令，缓存区内延时设置指令

GT_BufDelay(crd, delayTime, fifo)	
Crd:INT	坐标系号，取值范围：[1,2]
DelayTime:UINT	延时时间，取值范围：[0,16383]，单位：ms。
Fifo:INT	插补缓存区号，取值范围：[0,1]，默认为：0

GT_BufDA

缓存区指令，缓存区内输出 DA 值

GT_BufDA(crd, chn, daValue, fifo)	
Crd:INT	坐标系号，取值范围：[1,2]
Chn:INT	模拟量输出的通道号，取值范围：[1,8]
DaValue:INT	模拟量输出的值，取值范围：[-32768,32767]，其中：-32768 对应-10V，32767 对应+10V
Fifo:INT	插补缓存区号，取值范围：[0,1]，默认为：0

GT_BufLmtsOn

缓存区指令，缓存区内有效限位开关

GT_BufLmtsOn(crd, axis, limitType, fifo)	
Crd:INT	坐标系号，取值范围：[1,2]
Axis:INT	需要将限位有效的轴的编号，取值范围：[1,8]
LimitType:INT	需要有效的限位类型 MC_LIMIT_POSITIVE：需要将该轴的正限位有效 MC_LIMIT_NEGATIVE：需要将该轴的负限位有效 -1：需要将该轴的正限位和负限位都有效，默认为该值
Fifo:INT	插补缓存区号，取值范围：[0,1]，默认为：0

GT_BufLmtsOff

缓存区指令，缓存区内无效限位开关

GT_BufLmtsOff(crd, axis, limitType, fifo)	
Crd:INT	坐标系号，取值范围：[1,2]
Axis:INT	需要将限位无效的轴的编号，取值范围：[1,8]
LimitType:INT	需要无效的限位类型 MC_LIMIT_POSITIVE：需要将该轴的正限位无效 MC_LIMIT_NEGATIVE：需要将该轴的负限位无效 -1：需要将该轴的正限位和负限位都无效，默认为该值
Fifo:INT	插补缓存区号，取值范围：[0,1]，默认为：0

GT_BufSetStopIo

缓存区指令，缓存区内设置 axis 的停止 IO 信息

GT_BufSetStopIo(crd, axis, stopType, inputType, inputIndex, fifo)	
Crd:INT	坐标系号，取值范围：[1,2]
Axis:INT	需要设置停止 IO 信息的轴的编号，取值范围：[1,8]

StopType:INT	需要设置停止 IO 信息的停止类型 0：紧急停止类型 1：平滑停止类型
InputType:INT	设置的数字量输入的类型 MC_LIMIT_POSITIVE 正限位 MC_LIMIT_NEGATIVE 负限位 MC_ALARM 驱动报警 MC_HOME 原点开关 MC_GPI 通用输入
InputIndex:INT	设置的数字量输入的索引号，取值范围根据 inputType 的取值而定 当 inputType= MC_LIMIT_POSITIVE 时，取值范围：[1,8] 当 inputType= MC_LIMIT_NEGATIVE 时，取值范围：[1,8] 当 inputType= MC_ALARM 时，取值范围：[1,8] 当 inputType= MC_HOME 时，取值范围：[1,8] 当 inputType= MC_GPI 时，取值范围：[1,16]
Fifo:INT	插补缓存区号，取值范围：[0,1]，默认为：0

GT_BufMove

缓存区指令，实现刀向跟随功能，启动某个轴点位运动

GT_BufMove(crd,moveAxis,pos,vel, acc,modal,fifo)	
Crd:INT	坐标系号，取值范围：[1,2]
MoveAxis:INT	需要进行点位运动的轴号，取值范围：[1,8]，该轴不能处于坐标系中
Pos:DINT	点位运动的目标位置，单位：pulse
Vel:lreal	点位运动的目标速度，单位：pulse/ms
Acc:lreal	点位运动的加速度，单位：pulse/(ms*ms)
Modal:INT	点位运动的模式： 0：该指令为非模态指令，即不阻塞后续的插补缓存区指令的执行

	1：该指令为模态指令，将会阻塞后续的插补缓存区指令的执行
Fifo:INT	插补缓存区号，取值范围：[0,1]，默认为：0

GT_BufGear

缓存区指令，实现刀向跟随功能，启动某个轴跟随运动

GT_BufGear(crd,gearAxis, pos,fifo)	
Crd:INT	坐标系号，取值范围：[1,2]
GearAxis:INT	需要进行跟随运动的轴号，取值范围：[1,8]，该轴不能处于坐标系中
Pos:DINT	跟随运动的位移量，单位：pulse
Fifo:INT	插补缓存区号，取值范围：[0,1]，默认为：0

GT_SetUserSegNum

缓存区指令，设置自定义插补段段号

GT_SetUserSegNum(crd, segNum,fifo)	
Crd:INT	坐标系号，取值范围：[1,2]
SegNum:DINT	设置用户自定义的插补段段号
Fifo:INT	插补缓存区号，取值范围：[0,1]，默认为：0

GT_GetUserSegNum

读取自定义插补段段号

GT_GetUserSegNum(crd,pSegment, fifo)	
Crd:INT	坐标系号，取值范围：[1,2]
pSegment:POINTER TO DINT	读取的用户自定义的插补段段号
Fifo:INT	插补缓存区号，取值范围：[0,1]，默认为：0

GT_GetRemainderSegNum

读取未完成的插补段段数

GT_GetRemainderSegNum(short crd,long *pSegment,short fifo=0)	
Crd:INT	坐标系号，取值范围：[1,2]
pSegment:POINTER TO DINT	读取的剩余插补段的段数
FifoINT	插补缓存区号，取值范围：[0,1]，默认为：0

GT_CrdSpace

查询插补缓存区剩余空间

GT_CrdSpace(crd, pSpace, fifo)	
Crd:INT	坐标系号，取值范围：[1,2]
PSpace:POINTER TO DINT	读取插补缓存区中的剩余空间。
Fifo:INT	插补缓存区号，取值范围：[0,1]，默认为：0

GT_CrdClear

清除插补缓存区内的插补数据

GT_CrdClear(crd, fifo)	
Crd:INT	坐标系号，取值范围：[1,2]
Fifo:INT	所要清除的插补缓存区号，取值范围：[0,1]

GT_CrdStart

启动插补运动

GT_CrdStart(mask, option)	
Mask:INT	从 bit0~bit1 按位表示需要启动的坐标系，其中，bit0 对应坐标

	系 1，bit1 对应坐标系 2；0：不启动该坐标系，1：启动该坐标系。
Option:INT	从 bit0~bit1 按位表示坐标系需要启动的缓存区的编号，其中，bit0 对应坐标系 1，bit1 对应坐标系 2；0：启动坐标系中 FIFO0 的运动，1：启动坐标系中 FIFO1 的运动。

GT_CrdStatus

查询插补运动坐标系状态

GT_CrdStatus(crd, pRun, pSegment, fifo)	
Crd:INT	坐标系号，取值范围：[1,2]
PRun:POINTER TO INT	读取插补运动状态，0：该坐标系的该 FIFO 没有在运动；1：该坐标系的该 FIFO 正在进行插补运动。
PSegment:POINTER TO DINT	读取当前已经完成的插补段数，当重新建立坐标系或者调用 GT_CrdClear 指令后，该值会被清零。
Fifo:INT	所要查询运动状态的 fifo 号，取值范围：[0,1]，默认为：0

GT_SetOverride

设置插补运动目标合成速度倍率

GT_SetOverride(crd, synVelRatio)	
Crd:INT	坐标系号，取值范围：[1,2]
SynVelRatio:LREAL	设置的插补目标速度倍率，取值范围：(0,1]，系统默认该值为：1。

GT_SetCrdStopDec

设置插补运动平滑停止、急停合成加速度

GT_SetCrdStopDec(crd, decSmoothStop, decAbruptStop)

Crd:INT	坐标系号，取值范围：[1,2]
DecSmoothStop:LREAL	设置的坐标系合成平滑停止加速度，取值范围：(0,32767)，单位：pulse/(ms*ms)。
DecAbruptStop:LREAL	设置的坐标系合成急停加速度，取值范围：(0,32767)，单位：pulse/(ms*ms)。

GT_GetCrdStopDec

查询插补运动平滑停止、急停合成加速度

GT_GetCrdStopDec(crd, pDecSmoothStop, pDecAbruptStop)	
Crd:INT	坐标系号，取值范围：[1,2]
PDecSmoothStop:POINTER TO LREAL	查询坐标系合成平滑停止加速度，单位：pulse/(ms*ms)。
PDecAbruptStop:POINTER TO LREAL	查询坐标系合成急停加速度，单位：pulse/(ms*ms)。

GT_GetCrdPos

查询该坐标系的当前坐标位置值

GT_GetCrdPos(crd, pPos)	
Crd:INT	坐标系号，取值范围：[1,2]
Ppos:POINTER TO LREAL	读取的坐标系的坐标值，单位：pulse。该参数应该为一个数组首元素的指针，数组的元素个数取决于该坐标系的维数。

GT_GetCrdVel

查询该坐标系的合成速度值

GT_GetCrdVel(crd, pSynVel)	
Crd:INT	坐标系号，取值范围：[1,2]

PSynVel: POINTER TO LREAL	读取的坐标系的合成速度值，单位：pulse/ms。

GT_InitLookAhead

初始化插补前瞻缓存区

GT_InitLookAhead(crd, fifo, T, accMax, n, pLookAheadBuf)	
Crd:INT	坐标系号，取值范围：[1,2]
Fifo:INT	插补缓存区编号，取值范围：[0,1]
T:LREAL	拐弯时间，单位：ms
AccMax:LREAL	最大加速度，单位：pulse/(ms*ms)
N:INT	前瞻缓存区大小，取值范围：[0,32767)
PLookAheadBuf:POINTER TO TCrdData	前瞻缓存区内存区指针

4.1.2 重点说明

4.1.2.1 建立坐标系

运动控制器初始状态下，所有的规划轴都处于单轴运动模式下，两个坐标系也是无效的。所以，当需要进行插补运动时，首先需要建立坐标系，将规划轴映射到相应的坐标系中。每个坐标系最多支持四维(X-Y-Z-A)，用户根据自己的需求，也可以利用二维(X-Y)、三维(X-Y-Z)坐标系描述运动轨迹。

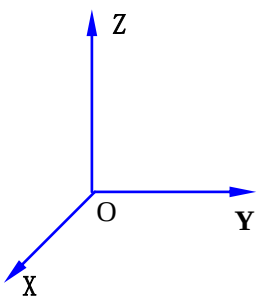


图 4-4-1 右手坐标系

用户通过调用 GT_SetCrdPrm()指令将在坐标系内描述的运动通过映射关系映射到相应的规划轴上。运动控制器根据坐标映射关系，控制各轴运动，实现要求的运动轨迹。调用 GT_SetCrdPrm()指令时，所映射的各规划轴必须处于静止状态。

建立坐标系的例程如下：

…… …… ……

```
Rtn:INT;
CrdPrm: TCrdPrm;                                (* 定义坐标系结构体变量*)
First:BOOL:=TRUE;
-------------------------------------------------------------------------------------------
IF First THEN
SysMemSet(ADR(crdPrm),0,sizeof(crdPrm));          (* 将变量初始化为全 0 *)
crdPrm.dimension:=2;                            (* 坐标系为二维坐标系*)
crdPrm.synVelMax:=500;                          (* 最大合成速度：500pulse/ms*)
crdPrm.synAccMax:=1;                            (* 最大加速度：1pulse/ms^2*)
crdPrm.evenTime := 50;                          (* 平滑时间：50ms*)
crdPrm.profile[0] := 1;                         (* 规划器 1 对应到 X 轴*)
crdPrm.profile[1] := 2;                         (* 规划器 2 对应到 Y 轴*)
crdPrm.setOriginFlag := 1;                      (* 需要明确指定坐标系原点的规划位置*)
crdPrm.originPos[0] := 100;
crdPrm.originPos[1] := 100;
rtn := GT_SetCrdPrm(1,ADR(crdPrm));             (* 建立 1 号坐标系，设置坐标系参数*)
First:=FALSE;
END_IF
```

…… …… ……

例程说明：

dimension：表示所建立的坐标系的维数，取值范围为[1,4]，该例程中所建立的坐标系是二维，即 X-Y 坐标系。

synVelMax：表示该坐标系所能承受的最大合成速度，如果用户在输入插补段的时候所设置的目标速度大于了该速度，则将会被限制为该速度。

synAccMax：表示该坐标系所能承受的最大合成加速度，如果用户在输入插补段的时候所设置的加速度大于了该加速度，则将会被限制为该加速度。

evenTime：每个插补段的最小匀速时间。当用户设置的插补段比较短时，而该插补段的目标速度又设置的比较大，则会造成合成速度的曲线如图 5-6-2 所示，只有加速段和减速段，

形成一个速度尖角，加速度在尖角处瞬间由正值变为了负值，造成较大的冲击；设置了 evenTime 之后，可以减小目标速度，使速度曲线如图 5-6-3 所示，减小了加速度突变的冲击。

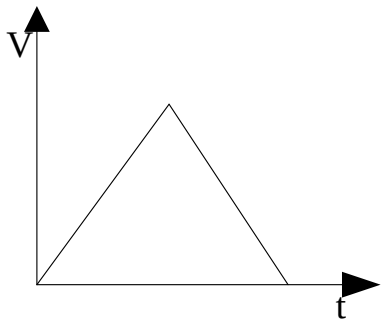


图 5-6-2　evenTime=0

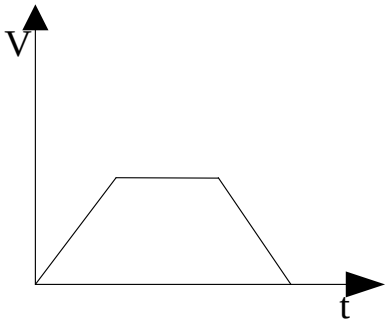


图 5-6-3　evenTime>0

profile[x]：规划轴与坐标轴之间的对应关系。Profile[0..7]对应规划轴 1~8，如果规划轴没有对应到该坐标系，则 profile[x]的值为 0；如果对应到了 X 轴，则 profile[x]为 1，Y 轴对应为 2，Z 轴对应为 3，A 轴对应为 4。不允许多个规划轴映射到相同坐标系的相同坐标轴，也不允许把相同规划轴对应到不同的坐标系，否则该指令将会返回错误值。

setOriginFlag：表示是否需要指定坐标系原点的规划位置值，该参数可以方便用户建立区别于机床坐标系的加工坐标系。如果该参数为 0，则加工坐标系的原点在当前的规划位置上，如果该参数为 1，则加工坐标系的原点在用户指定的规划位置上，通过 originPos 来指定。

originPos[x]：加工坐标系原点的规划位置值，即相对于机床坐标系的偏移量。建立的加工坐标系如图 5-6-4 所示。

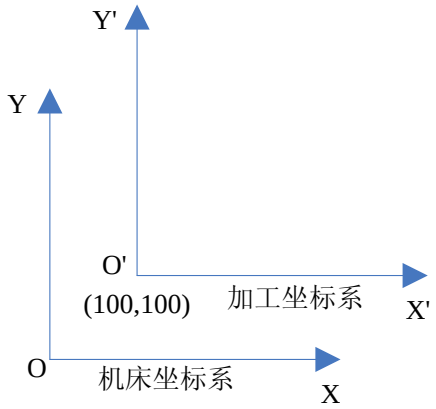


图 5-6-4 加工坐标系偏移量示意图

4.1.2.2 坐标系运动

坐标系运动采用缓存区运动方式，即用户需要向插补缓存区中传递插补数据，然后，启动插补运动，运动控制器则会依次执行用户所传递的插补数据，直到所有的插补数据全部运动完成。

每个坐标系包含两个缓存区(FIFO)：FIFO0 和 FIFO1，其中 FIFO0 为主要运动 FIFO，FIFO1 为辅助运动 FIFO，每个 FIFO 都含有 4096 段插补数据的空间。FIFO 支持动态管理的方式，即插补数据运动完成之后，其所占用的缓存区空间将会被释放，用户可以继续传递新的插补数据，通过这种方式，就可以支持大于 4096 段的用户插补数据。

坐标系运动的直线插补例程如下：

```
…… …… ……
Rtn:INT;
Run:INT;                                    (* 定义坐标系运动状态查询变量*)
Segment:DINT;                               (* 定义坐标系运动完成段查询变量*)
First:BOOL:=TRUE;
-----------------------------------------------------------------------------------------------
IF First THEN
rtn := GT_AxisOn(1);
rtn := GT_AxisOn(2);

rtn := GT_CrdClear(1,0);                    (* 清除坐标系 1 的 FIFO0 中的数据*)
(* 第一段插补数据*)
rtn := GT_LnXY(1,200000,0,100,0.1,0,0);     (*向坐标系 1 的 FIFO0 传递直线插补数据*)
                                            (* 该插补段的终点坐标(200000,0)*)
                                            (* 该插补段的目标速度：100pulse/ms*)
                                            (* 插补段的加速度：0.1pulse/ms^2*)
                                            (* 终点速度为 0*)
(*第二段插补数据*)
rtn = GT_LnXY(1,100000,173205,100,0.1,0,0);
(* 缓存区数字量输出*)
rtn = GT_BufIO(1,MC_GPO,16#ffff,16#55,0);   (* 数字量输出类型：通用输出*)
                                            (* bit0~bit15 全部都输出*)
                                            (* 输出的数值为 0x55*)
(* 第三段插补数据*)
rtn := GT_LnXY(1,-100000,173205,100,0.1,0,0);
(* 缓存区数字量输出*)
rtn := GT_BufIO(1,MC_GPO,16#ffff,16#aa,0);
(* 第四段插补数据*)
rtn := GT_LnXY(1,-200000,0,100,0.1,0,0);
```

```
(* 缓存区延时指令*)
rtn := GT_BufDelay(1,400,0);                    (* 该处延时 400ms*)
(* 第五段插补数据*)
rtn := GT_LnXY(1,-100000,-173205,100,0.1,0,0);
(* 缓存区数字量输出*)
rtn := GT_BufIO(1,MC_GPO,16#ffff,16#55,0);
(* 缓存区延时指令*)
rtn := GT_BufDelay(1,100,0);
(* 第六段插补数据*)
rtn := GT_LnXY(1,100000,-173205,100,0.1,0,0);
(* 第七段插补数据*)
rtn := GT_LnXY(1,200000,0,100,0.1,0,0);
rtn := GT_CrdSpace(1,ADR(space),0);             (*查询坐标系 1 的 FIFO0 所剩余的空间*)
rtn := GT_CrdStart(1,0);                        (*启动坐标系 1 的 FIFO0 的插补运动*)
First:=FALSE;
END_IF

(* 坐标系在运动,查询到的 run 的值为 1*)
rtn := GT_CrdStatus(1,ADR(run),ADR(segment),0);
…… …… ……
```

例程说明：

通过 GT_LnXY()指令向插补缓存区传递数据，在该指令中包含了终点坐标、加速度、目标速度，还可以调用 GT_BufIO()指令在缓存区中进行数字量输出，调用 GT_BufDelay()指令在缓存区中进行延时操作。该例程共向插补缓存区传递了 7 段运动数据，例程的运行结果如图 5-6-5 所示，是一个六边形。其中，在运动过程中进行了缓存区延时和数字量输出的操作。

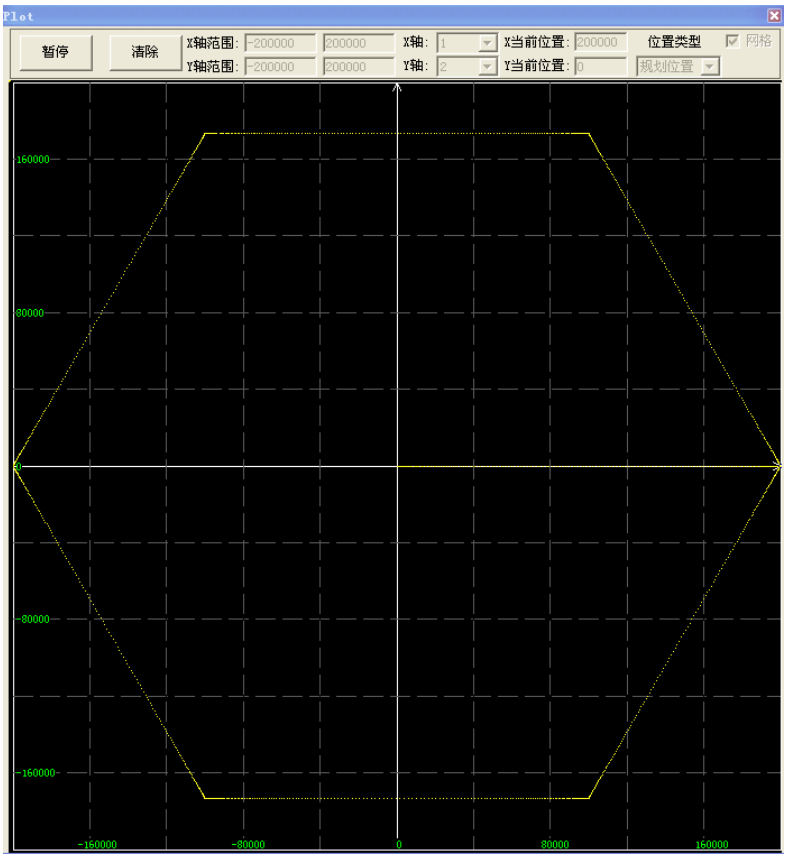


图 5-6-5 直线插补例程运动结果

GT_CrdStatus()指令可以用来查询坐标系指定 FIFO 的运动状态(运动或静止)，以及当前已经完成的插补运动的段数，该段数从建立坐标系之后的第一个 GT_CrdStart()的调用之后开始累加，直到销毁坐标系或者调用 GT_CrdClear()指令时才被清零。

坐标系运动的圆弧插补例程如下：

…… …… ……

```
Rtn:INT;
Run:INT;                                (* 定义坐标系运动状态查询变量*)
Segment:DINT;                           (* 定义坐标系运动完成段查询变量*)
First:BOOL:=TRUE;
-------------------------------------------------------------------------------------------------
IF First THEN
rtn := GT_AxisOn(1);
rtn := GT_AxisOn(2);

rtn := GT_CrdClear(1,0);                (* 清除坐标系 1 的 FIFO0 中的数据*)

(* 直线插补数据*)
rtn := GT_LnXY(1,200000,0,100,0.1,0,0);
```

```
(* 圆弧插补数据*)
rtn := GT_ArcXYC(1,200000,0,-100000,0,0,100,0.1, 0,0);
                                        (* 使用圆心描述方法描述一个整圆*)
                                        (* 圆心坐标(100000,0)*)
                                        (* 终点坐标与起点坐标重合(200000,0)*)
                                        (* 顺时针圆弧*)
                                        (* 该插补段的目标速度：100pulse/ms*)
                                        (* 插补段的加速度：0.1pulse/ms^2*)
                                        (* 终点速度为 0*)
(* 圆弧插补数据*)
rtn := GT_ArcXYR(1,0,200000,200000,1,100,0.1,0,0);
                                        (* 使用半径描述方法描述一个 1/4 圆弧*)
                                        (* 终点坐标为：(0,200000)*)
                                        (* 半径：200000*)
                                        (* 逆时针圆弧*)
rtn := GT_LnXY(1,0,0,100,0.1,0,0);          (* 回到原点位置*)
rtn := GT_CrdSpace(1, ADR(space),0);            (* 查询坐标系 1 的 FIFO0 所剩余的空
间*)
rtn := GT_CrdStart(1,0);                    (* 启动坐标系 1 的 FIFO0 的插补运动*)
First:=FALSE;
END_IF

rtn := GT_CrdStatus(1,ADR(run),ADR(segment),0);
(* 查询坐标系 1 的 FIFO 的插补运动状态*)
(* 坐标系在运动,查询到的 run 的值为 1*)
IF run=1 THEN
…… …… ……
```

圆弧插补例程的运行结果如图 5-6-6 所示。

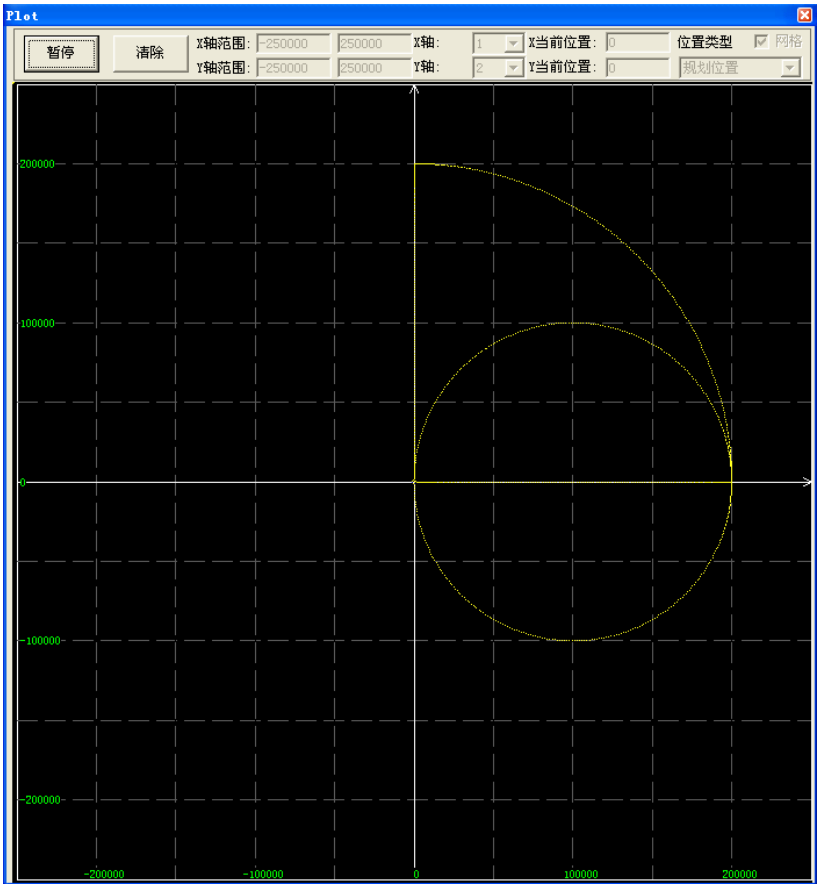


图 5-6-6 圆弧插补例程运动结果

该控制器支持在 XY 平面、YZ 平面和 ZX 平面的圆弧插补，其中圆弧插补的旋转方向按照右手螺旋定则定义为：从坐标平面的“上方”(即垂直于坐标平面的第三个轴的正方向)看，来确定逆时针方向和顺时针方向。可以这样简单记忆：将右手拇指前伸，其余四指握拳，拇指指向第三个轴的正方向，其余四指的方向即为逆时针方向。映射坐标系为二维坐标系(X-Y)时，XOY 坐标平面内的圆弧插补逆时针方向同样定义。

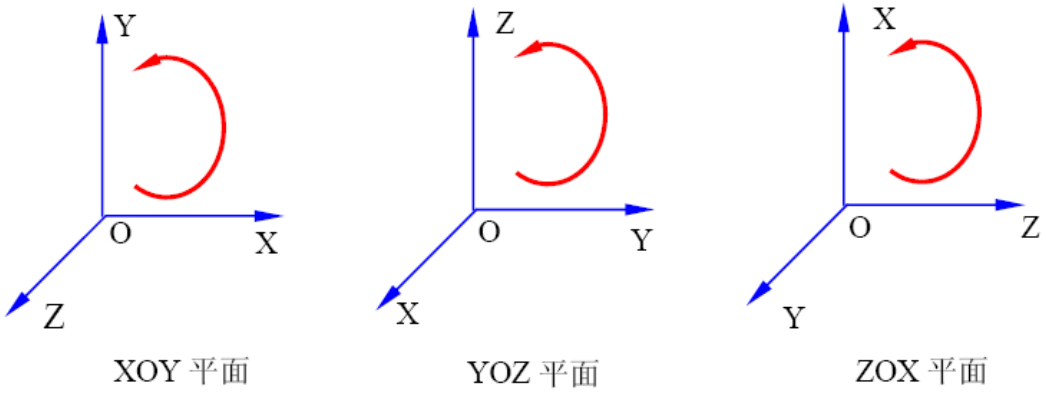


图 5-6-7 圆弧插补逆时针方向

圆弧插补有两种描述方法：半径描述方法和圆心坐标描述方法，用户可以根据加工数据

选择合适的描述方法来编程。所使用的描述方法遵循 G 代码的编程标准。

1．半径描述方法

半径描述方法，即调用指令 GT_ArcXYR()、GT_ArcYZR()、GT_ArcZXR()对圆弧进行描述，用户需要输入圆弧终点坐标、圆弧半径、圆弧的旋转方向、速度和加速度等。其中参数半径可为正值，也可为负值，其绝对值为圆弧的半径，正值表示圆弧的旋转角度≤180°，负值表示圆弧的旋转角度＞180°，如图 5-6-8 所示，半径描述方法无法描述 360°的整圆。

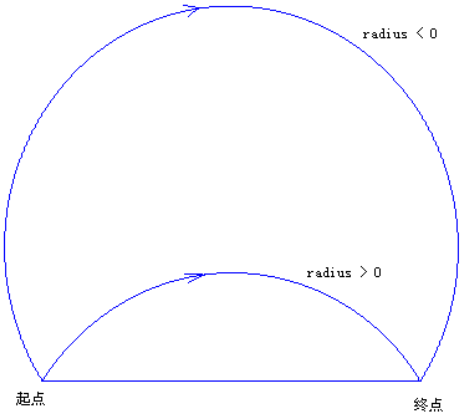


图 5-6-8 半径取正值/负值圆弧插补示意图

2．圆心描述方法

圆心描述方法，即调用指令 GT_ArcXYC()、GT_ArcYZC()、GT_ArcZXC()对圆弧进行描述，用户需要输入圆弧终点坐标、圆心相对于起点坐标的相对位置值、圆弧的旋转方向、速度和加速度等。其中，圆心位置值参数的定义如下图所示：

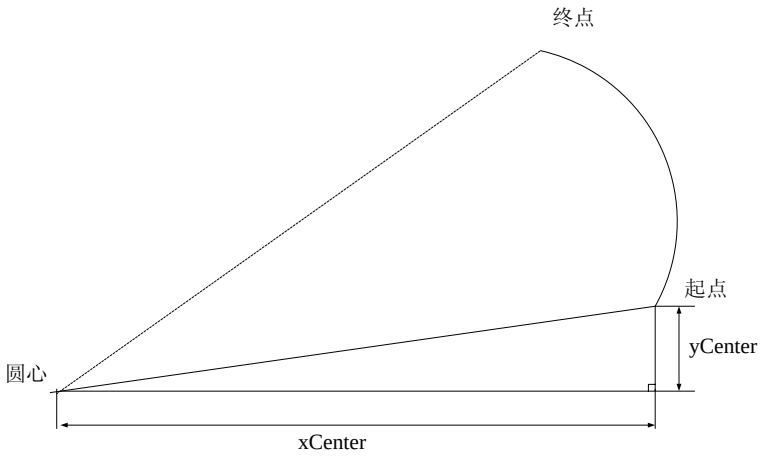


图 5-6-9 圆心表示方法示意图

圆心参数为相对于起点位置的增量值，带正负号，如果起点的坐标为(xStart,yStart)，用户设置的圆心参数为(xCenter,yCenter)，则圆心坐标值为(xStart+xCenter,yStart+yCenter)。用

户设置的起点坐标和终点坐标重合时，则表示将要进行一个整圆的运动。

用户应该保证圆弧描述指令可以正确描述一段圆弧，如果用户所设置的参数不能生成一段正确的圆弧，指令会返回 7(参数错误)。

4.1.2.3 前瞻预处理

小线段插补加工的特点：为保证刀具与加工工件接触面的光顺，尽量保持轨迹运动过程中切向速度的恒定，同时又必须保证一定的轨迹加工精度。

观察图 5-6-10，可以了解该图形中每条线段终点处都是拐点(轨迹特征发生明显改变的点)，且必须降速，但是否应该降到 0，需要根据线段长度、速度、加速度以及拐点速度变化限值等与加工工艺相关的参数来计算出各段的终点速度。

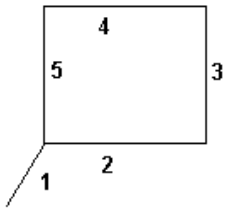


图 5-6-10 X-Y 平面多段轨迹图形

观察图 5-6-11，可以了解对于以小线段拟合曲线轨迹的线段组合，应在加工过程中尽量保持切向速度的恒定，但又必须保证在拐点(第 8 点)处将速度降到一个合理的值(合理的终点速度)，以保证加工执行机构(机械本体和电机)能够承受由于拐点处轨迹特征发生变化而带来的速度变化量。

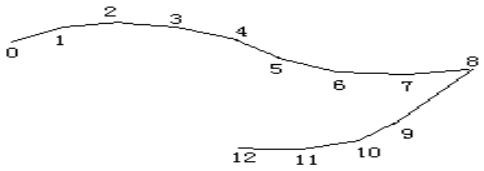


图 5-6-11 X-Y 平面小线段轨迹图形

为了解决高速和高精度这对矛盾，运动控制器对运动过程中的速度规划采用基于前瞻预处理的处理方式。

用户可根据本身机床的工艺特征参数(脉冲当量、目标速度、最大加速度、允许拐弯时

间等)，调用运动控制器提供的前瞻预处理模块，给出每段的终点速度，运动控制器则严格按照每段终点速度进行加减速控制。运动控制器将这套实现速度规划预处理功能的指令称为前瞻预处理(又称 LookAhead)指令。

从图 5-6-12 可以直观地了解，使用前瞻预处理功能模块来规划速度，在小线段加工过程中的对速度的显著提升：

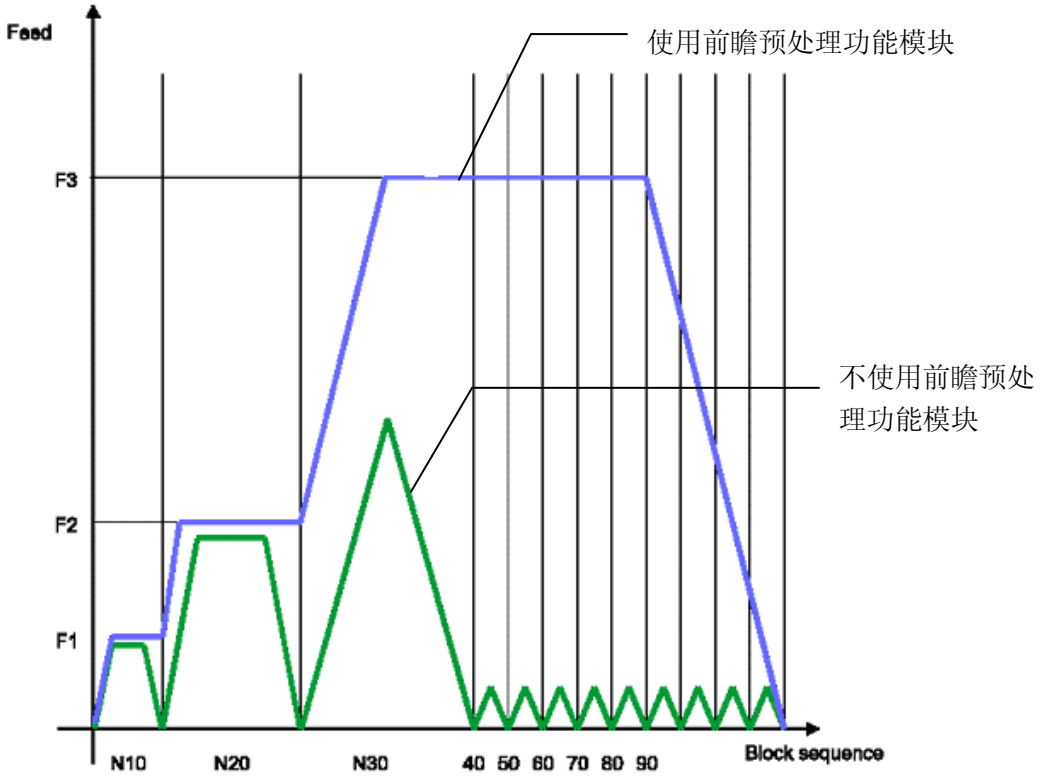


图 5-6-12 使用和不使用前瞻预处理功能模块的速度曲线对比图

使用前瞻预处理的例程如下：

…… …… ……

```
Rtn:INT;
I:INT;
CrdDataSend: TCrdData;
CrdData:ARRAY[0..199] OF TCrdData;                    (* 定义前瞻缓存区内存区*)
PosTest:ARRAY[0..1] OF DINT;
First:BOOL:=TRUE;
IF First THEN
rtn := GT_AxisOn(1);
rtn := GT_AxisOn(2);

rtn := GT_InitLookAhead(1,0,5,1,200,crdData); (* 初始化坐标系 1 的 FIFO0 的前瞻模块*)
(* 压插补数据：小线段加工*)
```

```
posTest[0] := 0;
posTest[1] := 0;
FOR I:=1 TO 300 BY 1 DO
    rtn := GT_LnXY(1,8000+posTest[0],9000+posTest[1],100,0.8,0,0);
    posTest[0] := posTest[0]+1600;
    posTest[1] := posTest[1]+1852;
END_FOR
rtn := GT_CrdData(1,0,0);                    (* 将前瞻缓存区中的数据压入控制器*)
rtn := GT_CrdStart(1,0);
First:=FALSE;
END_IF
…… …… ……
```

例程说明：

拐弯时间(T)：GT_InitLookAhead()指令的第三个参数，单位：ms。T 的经验范围是：1ms~10ms，T 越大，计算出来的终点速度越大，但却降低了加工精度；反之，提高了加工的精度，但计算出的终点速度偏低。因此要合理选择 T 值。

最大加速度(accMax)：GT_InitLookAhead()指令的第四个参数，单位：pulse/(ms*ms)。系统能承受的最大加速度，根据不同的机械系统和电机驱动器取值不同。

前瞻缓存区大小和前瞻缓存区内存区指针：该前瞻模块采用用户提供前瞻缓存区内存区的方式，因此，用户可以根据自己的需要以及计算机的条件定义合适的缓存区大小，前瞻缓存区越大，占用的内存区就越大。用户需要先定义一个插补数据数组变量，申请一定的内存区，然后通过 GT_InitLookAhead()指令把内存区的指针传递给运动控制器的前瞻模块，在进行前瞻预处理的过程中，用户不能再对该内存区进行任何操作，否则将会破坏前瞻缓存区中的数据，造成数据的错误。

当前瞻缓存区的段数不为 0 时，用户调用缓存区指令传递的插补数据先进入前瞻缓存区，当前瞻缓存区放满之后，如果再有新的数据传入，最先进入前瞻缓存区的数据，则会进入插补缓存区。

如果用户所有的插补数据已经输入完毕，前瞻缓存区中还有数据没有进入插补缓存区，这时，需要调用 GT_CrdData(1,0,0)，运动控制器会将前瞻缓存区的数据依次传递给插补缓存区，直到前瞻缓存区被清空为止。

在数据量比较大的时候，用户需要配合 GT_CrdSpace()指令查询插补缓存区的剩余空间，在有空间的时候再调用缓存区指令传递数据，如果插补缓存区已满，调用缓存区指令将会返回错误，说明该段插补数据没有输入成功，需要再次输入该段插补数据。

如果不使用前瞻预处理功能，则运动控制器不会对插补段的终点速度和目标速度进行优化，运动控制器将严格按照用户指定的目标速度和终点速度进行速度规划。如果用户调用 GT_LnXYG0()、GT_LnXYZG0()和 GT_LnXYZAG0()指令，则该运动指令将会完成一个完整的加减速过程，即每段运动的合成速度都是从 0 开始，结束的时候也是 0。如果调用其他插补运动指令(包括直线插补和圆弧插补指令)，则用户可以指定插补段的目标速度和终点速度，运动控制器会按照用户指定的目标速度和终点速度进行速度规划。

使用前瞻预处理功能之后，控制器会根据用户设置的参数将每段的终点速度设置为一个合理的值，该值不一定为 0，如果用户的工艺要求某段插补数据的终点速度必须为 0，则需要调用 GT_LnXYG0()、GT_LnXYZG0()或者 GT_LnXYZAG0()，该指令会将该直线插补段的终点速度设置为 0。如果调用其他插补运动指令(包括直线插补和圆弧插补指令)，则用户设置的终点速度将会无效，实际的终点速度是前瞻预处理模块根据用户设置的前瞻预处理参数和运动轨迹计算出来的一个合理的终点速度。另外，如果某段插补运动数据与下段插补运动数据之间存在缓存区延时，则该段插补运动的终点速度会被设置为 0。

前瞻预处理功能只支持 3 轴或者 3 轴以下的插补运动，如果建立的坐标系大于 3 轴，则在使用前瞻预处理功能时，会返回错误值，调用缓存区指令时，会返回 7(参数错误)。

如果没有进行前瞻预处理的合成速度曲线如图 5-6-13 所示，合成速度在不停的变化。进行前瞻预处理的合成速度曲线如图 5-6-14 所示，由于例程中的插补运动都是在同一条直线上，所以速度可以一直维持在目标速度，大大提高了加工效率。

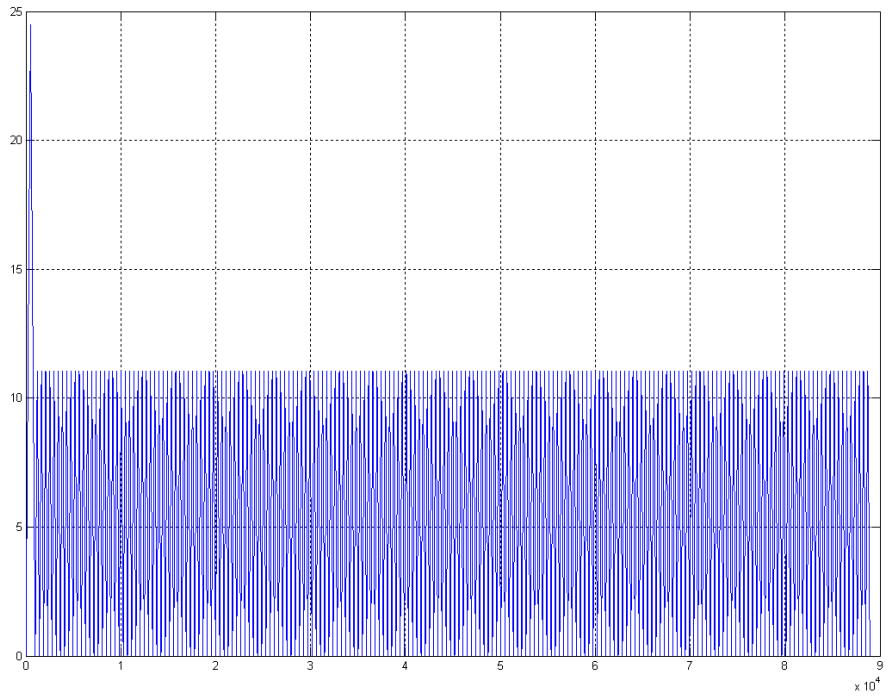


图 5-6-13 没有进行前瞻预处理的合成速度曲线

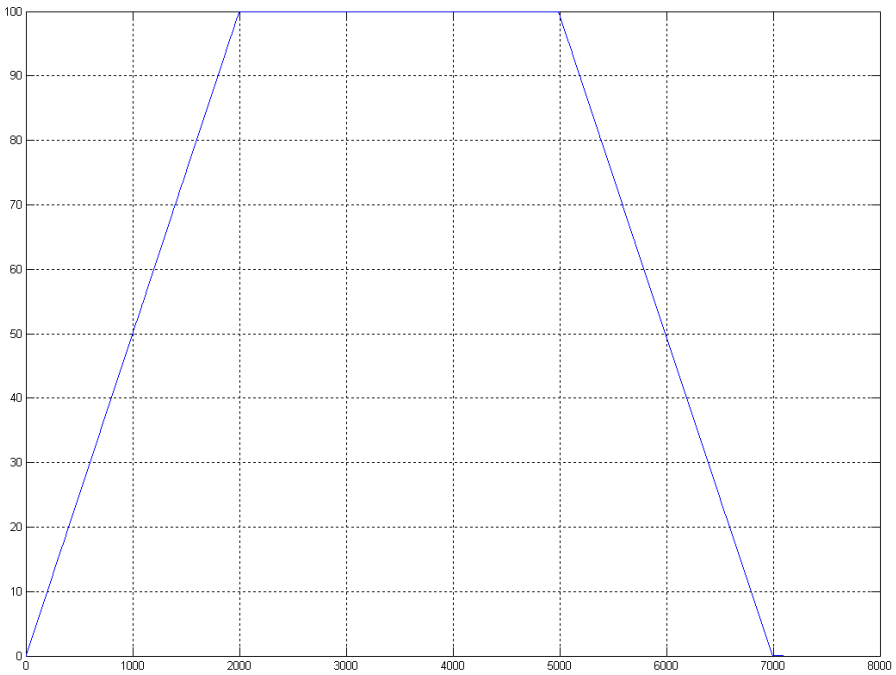


图 5-6-14 进行了前瞻预处理后的合成速度曲线

4.1.2.4 缓存区暂停、恢复

在缓存区插补的过程中，用户可能会需要暂停加工，查看加工效果，或者在暂停之后进行其他操作，如换刀等，该运动控制器支持上述操作过程。为了实现上述的操作过程，每个坐标系提供了两个插补缓存区：FIFO0 和 FIFO1。两个缓存区都有 4096 段插补缓存区，并且可以独立设置各自的前瞻预处理缓存区。

其中，FIFO0 是主运动 FIFO，用户的主体插补运动的插补数据应该放在 FIFO0 中。FIFO0 的插补运动可以被中断，中断后可以进行辅助 FIFO1 的插补运动，辅助 FIFO1 的插补运动完成后，FIFO0 可从断点处继续恢复原来的运动。

FIFO1 是辅助运动 FIFO，用户的辅助插补运动的插补数据可以放在 FIFO1 中，FIFO1 的插补数据必须在 FIFO0 的运动停止的情况下才能输入，如果 FIFO0 在运动，向 FIFO1 中传递插补数据，将会提示错误。FIFO1 的插补运动也可以暂停、恢复，但是，在暂停、恢复之间不可以进行 FIFO0 的插补运动，否则，FIFO1 缓存区将会被清空，不可以再恢复 FIFO1 的运动。

在主运动 FIFO0 的运动暂停之后，又进行了 FIFO1 的运动，如果用户希望在 FIFO1 运动结束之后，继续进行 FIFO0 的运动，则用户必须保证，FIFO1 运动结束后，坐标位置值与 FIFO0 停止时的坐标位置值(断点位置)相同，否则，FIFO0 将不接受启动运动指令，调用 GT_CrdStart()指令启动 FIFO0 的运动，将会提示错误。

4.1.2.5 刀向跟随功能

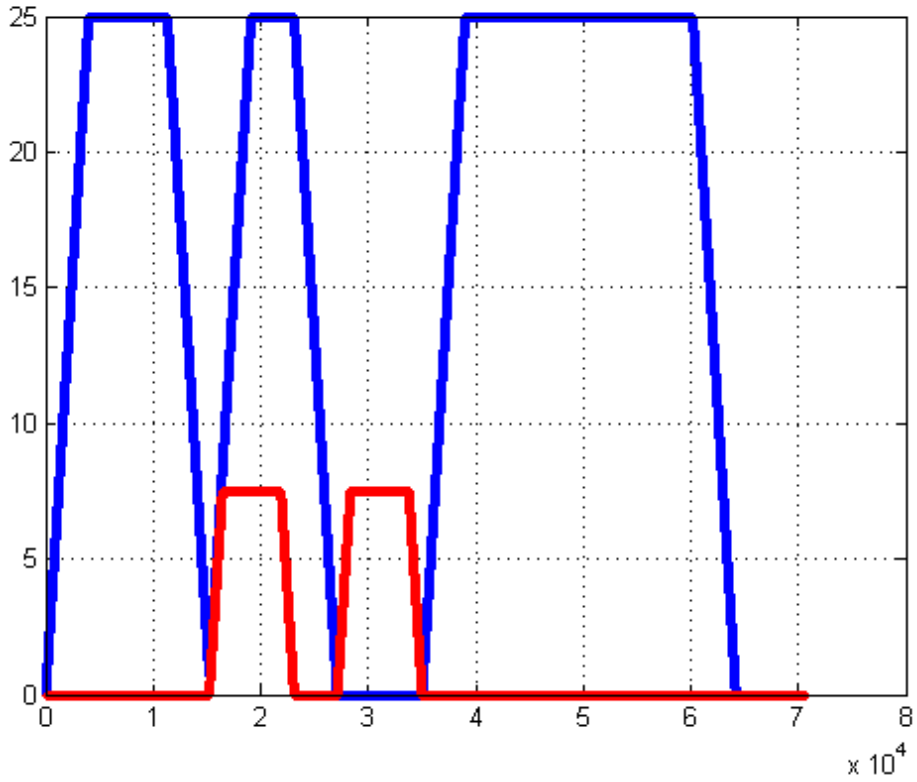
刀向跟随，就是在插补运动的过程中，部分轴会随着插补运动的合成位移的变化而变化，从而实现在加工过程中，刀具始终处于合适的加工方向的工艺。在本控制器的插补模块中有两条指令来实现该工艺：GT_BufMove()和GT_BufGear()。GT_BufMove()可以在插补运动的过程中插入模态和非模态的点位运动；GT_BufGear()可以在插补过程中实现其他轴跟随插补合成位移的运动。

1. 插补过程中的点位运动

插补过程中的点位运动通过在缓存区中压入 GT_BufMove()指令来实现，该指令的第二个参数是需要进行点位运动的轴号，这里需要注意的是，需要进行点位运动的轴不能是坐标系中的轴；当该轴的运动模式不是点位运动模式而且正在运动时，该指令将不能正常执行。该指令的第三个参数是点位运动的目标位置，该位置值是相对于机床原点的绝对位置。该指令的第四个参数是点位运动的目标速度，该值必须为正值。该指令的第五个参数是点位运动的加速度，该值必须为正值。该指令的第六个参数表示该点位运动是模态的还是非模态的，模态指令的意义是，在进行该点位运动时，后续的插补缓存区中的指令将会被暂停执行，直到该指令执行完毕后，才执行下一条指令；非模态指令的意义是，该指令启动了一个轴的点位运动后，立即取下一条缓存区中的指令执行，不会等待点位运动的结束。使用的具体例程如下：

```
PROGRAM PLC_PRG
VAR
   Enable : BOOL;
   Rtn : INT;
   Run : INT; (*  定义坐标系运动状态查询变量  *)
   Segment : DINT; (*  定义坐标系运动完成段查询变量  *)
END_VAR
----------------------------------------------------------------------------------------------
If Enable THEN
      rtn := GT_CrdClear(1,0); (*清除坐标系 1 的 FIFO0 中的数据*)
      rtn := GT_LnXY(1,200000,200000,100,0.1,0,0); (*  直线插补指令  *)
      rtn := GT_BufMove(1,4,50000,30,0.1,0,0); (*  缓存区内的点位运动指令
                                                点位运动的轴号：第 4 轴
                                                点位运动的目标位置：50000 pulse
                                                点位运动的目标速度：100 pulse/ms
                                                点位运动的目标加速度：0.1 pulse/(ms*ms)
                                                该点位运动是非模态指令  *)
      rtn := GT_LnXY(1,200000,0,100,0.1,0,0); (*  直线插补指令*)
      rtn := GT_BufMove(1,4,100000,30,0.1,1,0); (*  缓存区内的点位运动指令
                                                点位运动的轴号：第 4 轴
                                                点位运动的目标位置：100000 pulse
                                                点位运动的目标速度：100 pulse/ms
                                                点位运动的目标加速度：0.1 pulse/(ms*ms)
                                                该点位运动是模态指令  *)
      rtn := GT_ArcXYC(1,-200000,0,-200000,0,0,100,0.1,0,0); (*  圆弧插补指令  *)
      Enable := FALSE;
END_IF
```

例程的运行结果如下图所示：



其中蓝色为插补运动的合成速度，红色为点位运动轴的速度值，可以看出第一个点位运动是非模态指令，与插补运动同时运动，而第二个点位运动是模态指令，会阻塞插补运动，等点位运动结束之后，再进行插补运动。 在使用插补缓存区中的点位运动功能时，需要注意以下内容：

a. 点位运动的目标位置是相对于机床原点的绝对位置。

b. 如果在上一次的缓存区点位运动没有运动完成，又发送了新的点位运动，则会按照新的点位运动指令进行规划，即可以在插补缓存区中修改点位运动的目标位置和目标速度。

c. 如果在运动过程中停止插补缓存区的运动，则点位运动将不会停止，如果需要停止点位运动，则需要调用GT_Stop()指令停止响应轴的运动。恢复缓存区运动时，用户需自行保证之前点位运动的轴在合适的位置上。

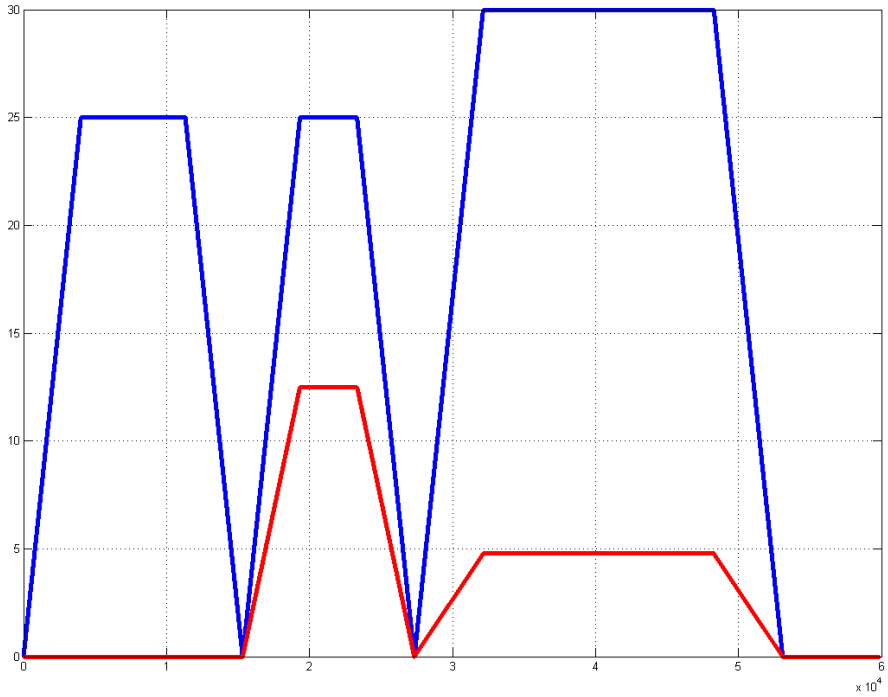
d. 当在模态点位运动的过程中，进行点位运动的轴由于触发限位等异常原因停止时，插补缓存区将不会再继续运行，此时用户需排查异常情况，重新设置相应参数，使系统正常后才可以工作。

2. 插补过程中的跟随运动

插补过程中的跟随运动通过在缓存区中压入 GT_BufGear()指令来实现，该指令的第二个参数是需要进行跟随运动的轴号，这里需要注意的是，需要进行跟随运动的轴不能是坐标系中的轴；如果在发送跟随指令 GT_BufGear()时该轴正在运动时，该指令将不能正常执行。该指令的第三个参数是跟随运动的位移量，该位移量是相对值，即下一段插补段运动过程中，跟随轴需要运动的位移量。使用的具体例程如下：

```
PROGRAM PLC_PRG
VAR
   Enable : BOOL;
   Rtn : INT;
   Run : INT; (*  定义坐标系运动状态查询变量  *)
   Segment : DINT; (*  定义坐标系运动完成段查询变量  *)
END_VAR
-------------------------------------------------------------------------------------
If Enable THEN
      rtn = GT_CrdClear(1,0); (*  清除坐标系 1 的 FIFO0 中的数据  *)
      rtn = GT_LnXY(1,200000,200000,100,0.1,0,0); (*  直线插补指令  *)
      rtn = GT_BufGear(1,4,50000, 0); (*  缓存区内的跟随运动指令
                                          跟随运动的轴号：第 4 轴
                                          跟随运动的位移量：50000 pulse *)
      rtn = GT_LnXY(1,200000,0,100,0.1,0,0); (*  直线插补指令  *)
      rtn = GT_BufGear(1,4,50000,0); (*  缓存区内的跟随运动指令
                                          跟随运动的轴号：第 4 轴
                                          跟随运动的位移量：50000 pulse *)
      rtn = GT_ArcXYC(1,-200000,0,-200000,0,0,100,0.1,0,0); (*  圆弧插补指令  *)
      Enable := FALSE;
END_IF
```

例程的运行结果如下图所示：



其中蓝色为插补运动的合成速度，红色为点位运动轴的速度值，跟随轴的速度跟随插补运动的合成速度的变化而变化。 在使用插补缓存区中的跟随运动功能时，需要注意以下内容：

a．GT_BufGear()指令需要在所要跟随的插补段前，不要间隔其他种类的指令，可以同时调用多个 GT_BufGear()指令来实现多个轴跟随插补运动。

b．当暂停坐标系运动时，当插补的合成速度减速到 0 时，跟随轴的速度也会为 0，如果希望重新恢复坐标系运动时，跟随轴仍能够实现跟随，不要调用 GT_Stop()指令停止跟随轴的运动，否则重新启动插补缓存区的运动时，跟随轴将无法完成正在进行的跟随运动。

4.2 PVT 模式

4.2.1 指令列表

PVT 指令列表

指令	说明
GT_PrfPvt	设置指定轴为 PVT 模式
GT_SetPvtLoop	设置循环次数
GT_GetPvtLoop	查询循环次数
GT_PvtTable	向指定数据表传送数据，采用 PVT 描述方式
GT_PvtTableComplete	向指定数据表传送数据，采用 Complete 描述方式
GT_PvtTablePercent	向指定数据表传送数据，采用 Percent 描述方式
GT_PvtPercentCalculate	计算 Percent 描述方式下各数据点的速度
GT_PvtTableContinuous	向指定数据表传送数据，采用 Continuous 描述方式
GT_PvtContinuousCalculate	计算 Continuous 描述方式下各数据点的时间
GT_PvtTableSelect	选择数据表
GT_PvtStart	启动运动
GT_PvtStatus	读取状态

GT_PrfPvt

设置指定轴为 PVT 模式

GT_PrfPvt(profile)	
Profile:INT	轴号

GT_SetPvtLoop

设置循环次数

GT_SetPvtLoop(profile, loop)	
Profile:INT	轴号

Loop:DINT	指定循环执行的次数 0 表示无限循环

GT_GetPvtLoop

查询循环次数

GT_GetPvtLoop(profile, pLoopCount, pLoop)	
Profile:INT	轴号
PLoopCount: POINTER TO DINT	查询已经循环的次数
PLoop: POINTER TO DINT	查询循环执行的总次数

GT_PvtTable

向指定数据表传送数据，采用 PVT 描述方式

GT_PvtTable(tableId, count, pTime, pPos, pVel)	
TableId:INT	指定数据表
Count:DINT	数据点个数，每个数据表具有 1024 个存储空间 每个数据点占用 1 个存储空间
PTime:POINTER TO LREAL	数据点时间数组，单位是“毫秒”，数组长度为 count
PPos:POINTER TO LREAL	数据点位置数组，单位是“脉冲”，数组长度为 count
PVel:POINTER TO LREAL	数据点速度数组，单位是“脉冲/毫秒”，数组长度为 count

GT_PvtTableComplete

向指定数据表传送数据，采用 Complete 描述方式

GT_PvtTableComplete(tableId,count, pTime, pPos, pA, pB, pC, velBegin, velEnd)	
TableId:INT	指定数据表
Count:DINT	数据点个数，每个数据表具有 1024 个存储空间 每个数据点占用 1 个存储空间
PTime:POINTER TO LREAL	数据点时间数组，单位是“毫秒”，数组长度为 count
PPos:POINTER TO LREAL	数据点位置数组，单位是“脉冲”，数组长度为 count
pA、pB、pc:POINTER TO LREAL	工作数组，内部使用，数组长度为 count 该数组用户不必赋值
velBegin:LREAL	起点速度，单位是“脉冲/毫秒”

velEnd:LREAL	终点速度，单位是“脉冲/毫秒”

GT_PvtTablePercent

向指定数据表传送数据，采用 Percent 描述方式

GT_PvtTablePercent(tableId, count, pTime, pPos, pPercent, velBegin)	
TableId:INT	指定数据表
Count:DINT	数据点个数，每个数据表具有 1024 个存储空间 每个数据点占用 1~3 个存储空间
pTime:POINTER TO LREAL	数据点时间数组，单位是“毫秒”，数组长度为 count
pPos:POINTER TO LREAL	数据点位置数组，单位是“脉冲”，数组长度为 count
pPercent:POINTER TO LREAL	数据点百分比数组，数组长度为 count 百分比的取值范围[0,100]
velBegin:LREAL	起点速度，单位是“脉冲/毫秒”

GT_PvtPercentCalculate

计算 Percent 描述方式下各数据点的速度

GT_PvtPercentCalculate(count, pTime, pPos, pPercent, velBegin, pVel)	
Count:DINT	数据点个数，该指令用来计算各数据点的速度，不会将数据点下载到运动控制器
pTime:POINTER TO LREAL	数据点时间数组，单位是“毫秒”，数组长度为 count
pPos:POINTER TO LREAL	数据点位置数组，单位是“脉冲”，数组长度为 count
pPercent:POINTER TO LREAL	数据点百分比数组，数组长度为 count 百分比的取值范围[0,100]
velBegin: LREAL	起点速度，单位是“脉冲/毫秒”
pVel:POINTER TO LREAL	返回各数据点的速度，单位是“脉冲/毫秒”

GT_PvtTableContinuous

向指定数据表传送数据，采用 Continuous 描述方式

GT_PvtTableContinuous(tableId, count, pPos, pVel, pPercent, pVelMax, pAcc, pDec, timeBegin)	
TableId:INT	指定数据表
Count:DINT	数据点个数，每个数据表具有 1024 个存储空间 每个数据点占用 1~8 个存储空间

pPos:POINTER TO LREAL	数据点位置数组，单位是“脉冲”，数组长度为 count
pVel:POINTER TO LREAL	数据点速度数组，单位是“脉冲/毫秒”，数组长度为 count
pPercent:POINTER TO LREAL	数据点百分比数组，数组长度为 count 百分比的取值范围[0,100]
pVelMax:POINTER TO LREAL	数据点最大速度数组，单位是“脉冲/毫秒”，数组长度为 count
pAcc:POINTER TO LREAL	数据点加速度数组，单位是“脉冲/毫秒 ^{2}”，数组长度为 count
pDec:POINTER TO LREAL	数据点减速度数组，单位是“脉冲/毫秒 ^{2}”，数组长度为 count
TimeBegin:LREAL	起点时间，单位是“毫秒”

GT_PvtContinuousCalculate

计算 Continuous 描述方式下各数据点的时间

GT_PvtContinuousCalculate(count, pPos, pVel, pPercent, pVelMax, pAcc, pDec, pTime)	
Count:DINT	数据点个数，该指令用来计算各数据点时间，不会将数据点下载到运动控制器
pPos:POINTER TO LREAL	数据点位置数组，单位是“脉冲”，数组长度为 count
pVel:POINTER TO LREAL	数据点速度数组，单位是“脉冲/毫秒”，数组长度为 count
pPercent:POINTER TO LREAL	数据点百分比数组，数组长度为 count 百分比的取值范围[0,100]
pVelMax:POINTER TO LREAL	数据点最大速度数组，单位是“脉冲/毫秒”，数组长度为 count
pAcc:POINTER TO LREAL	数据点加速度数组，单位是“脉冲/毫秒 ^{2}”，数组长度为 count
pDec:POINTER TO LREAL	数据点减速度数组，单位是“脉冲/毫秒 ^{2}”，数组长度为 count
pTime:POINTER TO LREAL	返回各数据点的时间，单位是“毫秒”

GT_PvtTableSelect

选择数据表

GT_PvtTableSelect(profile, tableId)	
Profile:INT	轴号
TableId:INT	指定数据表 PVT 模式提供 32 个数据表，取值范围[1,32]

GT_PvtStart

启动运动

GT_PvtStart(mask)	
Mask:DINT	按位指示需要启动的轴号 bit0 表示 1 轴，bit1 表示 2 轴，…… 当 bit 位为 1 时表示启动对应的轴 在启动运动之前，可以调用 GT_PvtTableSelect 选择数据表 如果没有选择数据表，默认使用数据表 1 如果数据表为空，则启动失败

GT_PvtStatus

读取状态

GT_PvtStatus(profile, pTableId, pTime, count)	
Profile:INT	轴号
PTableId:POINTER TO INT	当前正在使用的数据表
pTime:POINTER TO LREAL	当前轴已经运动的时间，单位是“毫秒”
Count:INT	读取的轴数

4.2.2 重点说明

PVT 模式使用一系列数据点的“位置、速度、时间”参数来描述运动规律。

位置、速度和时间满足如下函数关系：

$$p = at^3 + bt^2 + ct + d$$

$$v = \frac{dp}{dt} = 3at^2 + 2bt + c$$

如果给定相邻 2 个数据点的“位置、速度、时间”参数，可以得到如下方程组：

$$\begin{cases} at_1^3 + bt_1^2 + ct_1 + d = p_1 \\ 3at_1^2 + 2bt_1 + c = v_1 \\ at_2^3 + bt_2^2 + ct_2 + d = p_2 \\ 3at_2^2 + 2bt_2 + c = v_2 \end{cases}$$

求解该方程组，可以得到 a、b、c、d，因此相邻 2 个数据点的运动规律就可以确定下来。

运动控制器提供 32 个数据表存储数据点。每个数据表具有 1024 个存储空间。数据表和轴之间相互独立，一个数据表可以供多个轴使用。

调用 GT_PvtTable、GT_PvtTableComplete、GT_PvtTablePercent 或 GT_PvtTableContinuous 指令向数据表中传递数据。这些指令会删除数据表中原先的数据，因此所有数据应当一次传送完毕。如果使用数据表的轴正在运动，禁止更新数据表。

调用 GT_PvtTableSelect 指令选择数据表。可以在运动状态下切换数据表，但是不会立即切换。只有当前数据表执行完毕以后，才会切换到新的数据表。

调用 GT_PvtStart 启动运动。启动以后，各轴时间清 0。如果第一个数据点的时间为 0 则立即启动，否则会延时启动，延时时间等于第一个数据点的时间。

数据表可以循环执行，调用 GT_SetPvtLoop 设置循环次数，循环次数为 0 表示无限循环。当遍历完数据表以后，时间初始化为第一个数据点的时间，而不是 0。

假设有如下 4 个数据点，采用 PVT 方式进行描述。

数据点	时间（毫秒）	位置（脉冲）	速度（脉冲/毫秒）
P1	1,000	0	0
P2	2,000	5,000	10
P3	3,000	15,000	10
P4	4,000	20,000	0

1. 调用 GT_PrfPvt 将轴切换到 PVT 模式
2. 调用 GT_PvtTable 将 4 个数据点传递到数据表
3. 调用 GT_SetPvtLoop 设置为循环执行

4. 调用 GT_PvtStart 启动运动

由于 P1 的时间为 1000 毫秒，因此调用 GT_PvtStart 以后延时 1000 毫秒启动。由于是循环执行，到达 P4 以后返回到 P1，速度曲线如下图所示。

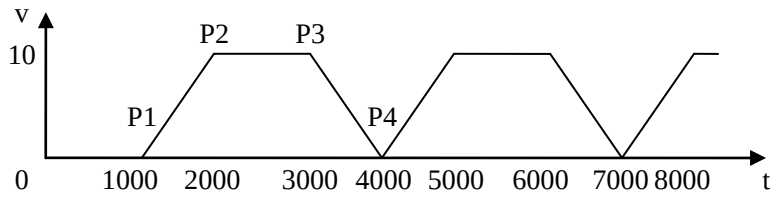


图 5-7-1 循环执行数据表

PVT 模式有 4 种方式描述运动规律，PVT、Complete、Percent 和 Continuous，下面对此进行详细说明。

4.2.2.1 PVT 描述方式

PVT 描述方式直接定义各数据点的“位置、速度、时间”。相邻 2 个数据点之间，运动控制器使用 3 次多项式对位置进行插值，使用 2 次多项式对速度进行插值。因此当给出各数据点“位置、速度、时间”参数以后，相应的运动规律也就确定下来。

例如下面 4 组数据点，采用 PVT 描述方式。

数据点	时间（毫秒）	位置（脉冲）	速度（脉冲/毫秒）
P1	0	0	0
P2	1,000	5,000	10
P3	2,000	15,000	10
P4	3,000	20,000	0
数据点	时间（毫秒）	位置（脉冲）	速度（脉冲/毫秒）
P1	0	0	0
P2	1,000	5,000	9
P3	2,000	15,000	9
P4	3,000	20,000	0
数据点	时间（毫秒）	位置（脉冲）	速度（脉冲/毫秒）

P1	0	0	0
P2	1,000	5,000	7.5
P3	2,333	15,000	7.5
P4	3,333	20,000	0
数据点	时间（毫秒）	位置（脉冲）	速度（脉冲/毫秒）
P1	0	0	0
P2	750	1,667	6.6669
P3	2,250	18,333	6.6669
P4	3,000	20,000	0

这 4 组数据点对应的运动规律如图所示。

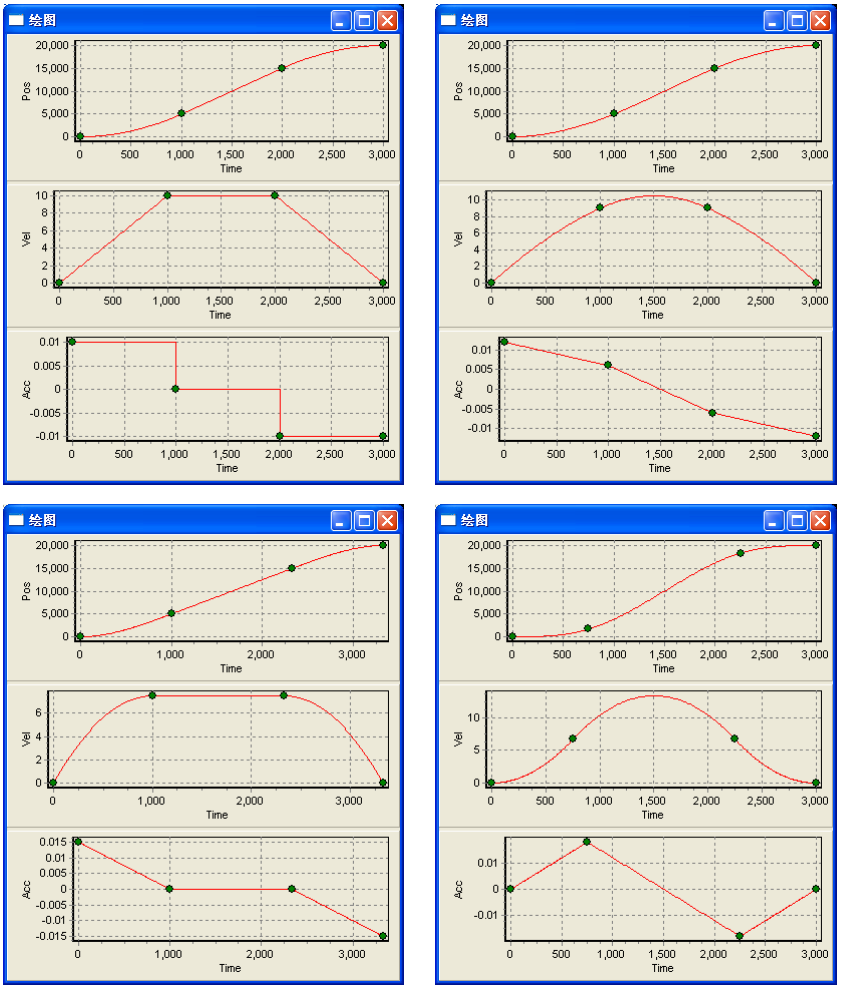


图 5-7-2 合理的 PVT 数据点参数

可以看出，PVT 描述方式非常灵活。给定数据点的“位置、速度、时间”参数，就能

够得到相应的运动规律。

需要注意的是，数据点参数需要仔细设计，否则难以得到理想的运动规律。

例如下面 2 组数据点参数不合理，得到的速度曲线不够平滑。

数据点	时间（毫秒）	位置（脉冲）	速度（脉冲/毫秒）
P1	0	0	0
P2	1,000	5,000	15
P3	2,000	15,000	15
P4	3,000	20,000	0
数据点	时间（毫秒）	位置（脉冲）	速度（脉冲/毫秒）
P1	0	0	0
P2	1,000	5,000	5
P3	2,000	15,000	5
P4	3,000	20,000	0

这 2 组数据点对应的运动规律如图所示。

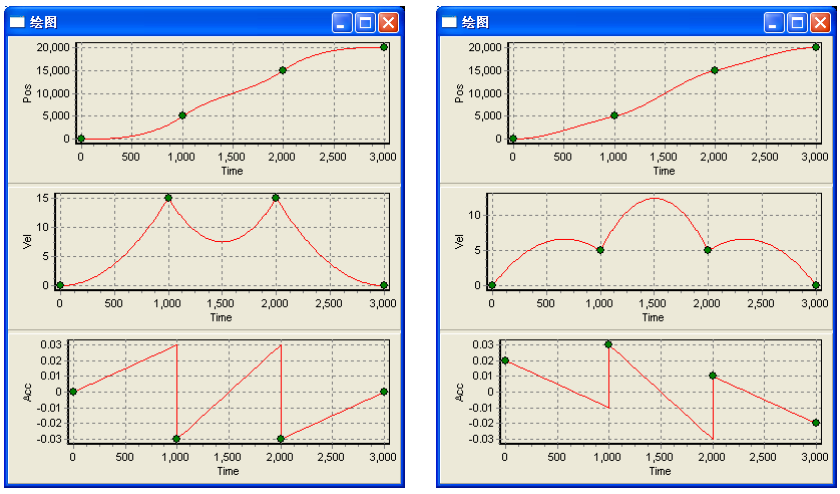


图 5-7-3 不合理的 PVT 数据点参数

4.2.2.2 Complete 描述方式

Complete 描述方式定义各数据点的“位置、时间”，以及起点速度和终点速度。

Complete 方式只定义了起点速度和终点速度。运动控制器根据各数据点的“位置、时间”参数计算中间各点的速度，确保各数据点速度连续和加速度连续。

例如下面这组数据点，采用 Complete 描述方式，可以轻松得到光滑的速度曲线。

数据点	时间（毫秒）	位置（脉冲）	速度（脉冲/毫秒）
P1	0	0	0
P2	1,000	5,000	不指定
P3	2,000	15,000	不指定
P4	3,000	20,000	0

这组数据点对应的运动规律如图所示。

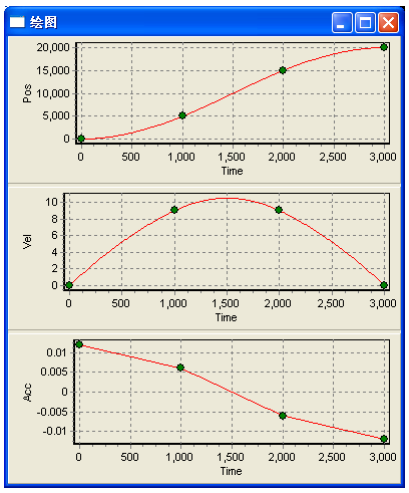


图 5-7-4 Complete 描述方式

Complete 适合描述光滑的速度曲线，例如三角函数等。

假设位置和时间之间的关系由函数 $P=50000\sin^2(\pi/2000*t)$确定。在一个函数周期[0,2000]内取 5 个时间点计算相应的位置，如下表所示。

数据点	时间（毫秒）	位置（脉冲）	速度（脉冲/毫秒）
P1	0	0	0
P2	500	25,000	不指定
P3	1,000	50,000	不指定
P4	1,500	25,000	不指定

P5	2,000	0	0

这组数据点对应的运动规律如图所示。

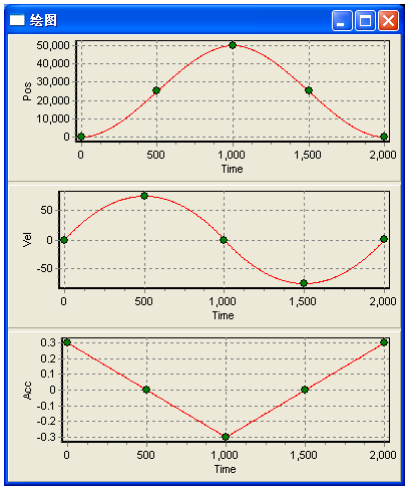


图 5-7-5 Complete 方式描述三角函数

增加数据点可以减小与函数 $P=50000\sin^2(\pi/2000*t)$的逼近误差。下图给出了数据点数为 5、10、50 时的位置误差。

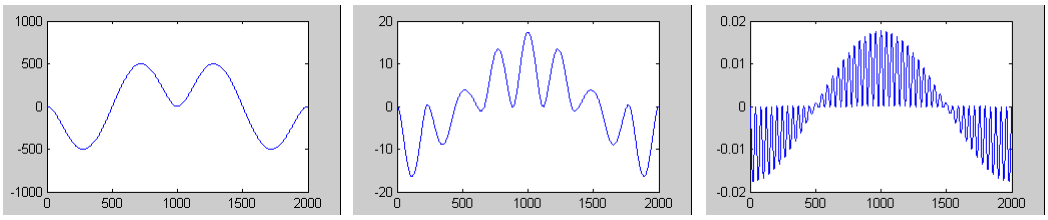


图 5-7-6　数据点数分别为 5、10、50 时的位置误差

4.2.2.3 Percent 描述方式

Percent 描述方式定义各数据点的“位置、时间、百分比”，以及起点速度。

Percent 描述方式能够精确定义加速段、匀速段、减速段的位移、速度和时间。

Percent 描述方式假设相邻 2 个数据点之间速度为线性变化，利用起点速度以及各数据点的“位置、时间”参数，通过如下递推公式可以计算出各数据点的速度。

$$v_{i+1}=\frac{2(p_{i+1}-p_i)}{t_{i+1}-t_i}-v_i$$

因此指定了各数据点的“位置、时间”参数以后，各数据点的速度实际上也就已经确定下来。

通过“百分比”参数可以调整速度曲线的光滑性。数据点的百分比参数是指“相邻 2 个数据点之间加速度的变化时间占速度变化时间的百分比”。以下图为例来进行说明。

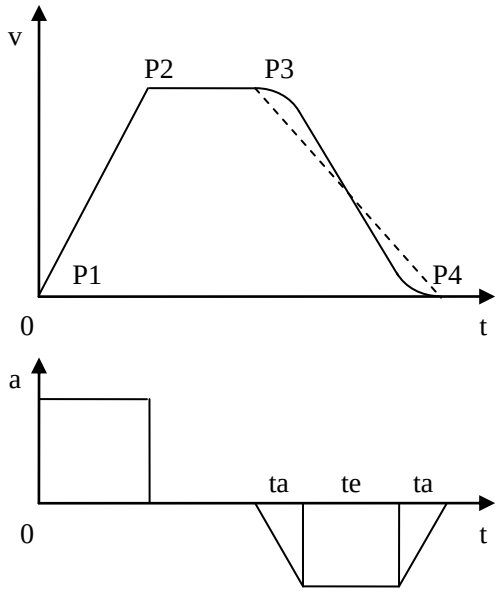


图 5-7-7 百分比的定义

数据点 P1 和 P2 之间加速度不变，因此数据点 P1 的百分比为 0。

数据点 P2 和 P3 之间加速度不变，因此数据点 P2 的百分比为 0。

数据点 P3 和 P4 之间加速度变化时间为 2ta，运动时间为 2ta+te，因此数据点 P3 的百分比为 2ta/(2ta+te)*100%。

调整百分比参数，不会影响数据点的“位置、时间参数”。以上图为例，当数据点 P3 的百分比为 0 时，数据点 P3 和 P4 之间的速度曲线为虚线；当数据点 P3 的百分比不为 0 时，数据点 P3 和 P4 之间的速度曲线为实线。

例如下面这组数据点，采用 Percent 描述方式。

数据点	时间（毫秒）	位置（脉冲）	百分比	速度（脉冲/毫秒）
P1	0	0	60	0
P2	1,000	5,000	0	不指定
P3	2,000	15,000	100	不指定

P4	3,000	20,000	0	不指定

这组数据点对应的运动规律如图所示。

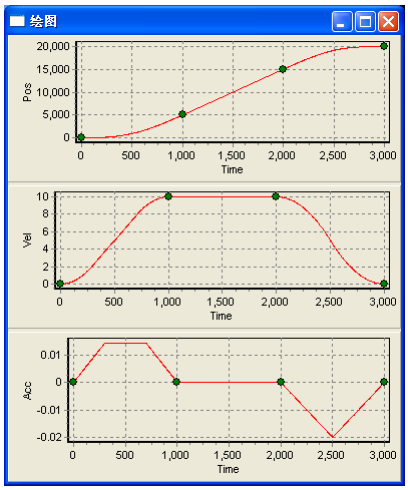


图 5-7-8 Percent 描述方式

4.2.2.4 Continuous 描述方式

Continuous 描述方式定义各数据点的“位置、速度、最大速度、加速度、减速度、百分比”。不用指定数据点的时间。运动控制器根据数据点参数，自动将相邻 2 个数据点之间拆分为加速段、匀速段和减速段。

数据点 P_i 的最大速度是指从数据点 P_i 到数据点 P_{i+1} 之间的速度上限。

数据点 P_i 的加速度是指从数据点 P_i 到数据点 P_{i+1} 之间的加速段所使用的加速度。

数据点 P_i 的减速度是指从数据点 P_i 到数据点 P_{i+1} 之间的减速段所使用的减速度。

数据点 P_i 的百分比是指从数据点 P_i 到数据点 P_{i+1} 之间的加减速段中，加速度变化时间占速度变化时间的百分比。

相邻 2 个数据点之间能够拆分出来的段数和这 2 个数据点的参数有关，下图示例了一些可能的分段情况。

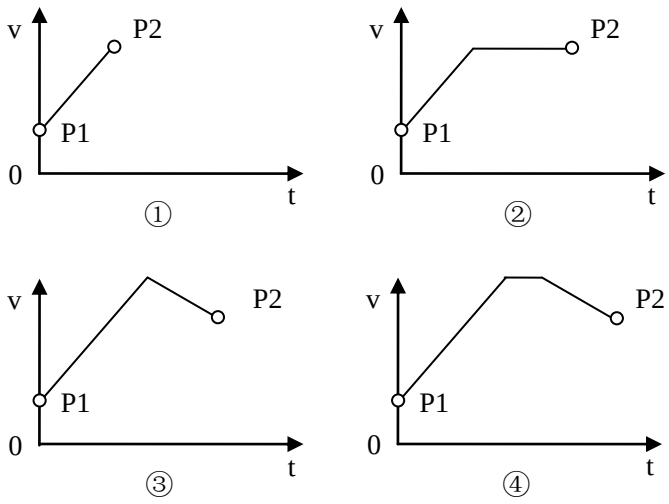


图 5-7-9 Continuous 描述方式

例如下面这 2 组数据点，采用 Continuous 描述方式。

数据点	位置	速度	最大速度	加速度	减速度	百分比
P1	0	0	10	0.01	0.01	60
P2	20,000	0	10	0.01	0.01	0
数据点	位置	速度	最大速度	加速度	减速度	百分比
P1	0	0	10	0.01	0.01	60
P2	19,800	2	2	0.02	0.02	0
P3	21,800	0	2	0.02	0.02	0

这 2 组数据点对应的运动规律如图所示。

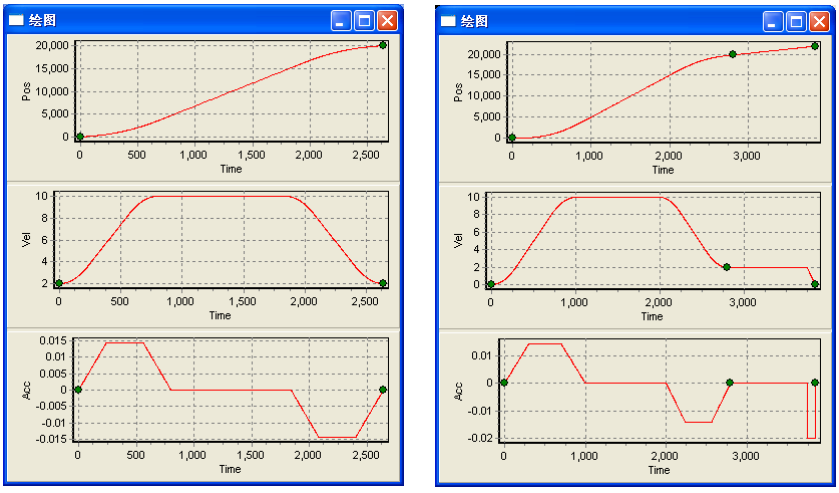


图 5-7-10 Continuous 描述方式

4.2.3 例程

4.2.3.1 PVT 描述方式

整个速度曲线由 5 段组成，并且带有起跳速度。

第一段速度增大，加速度保持不变。

第二段速度增大，加速度减小。

第三段速度不变，加速度为 0。

第四段速度减小，加速度增大。

第五段速度减小，加速度不变。

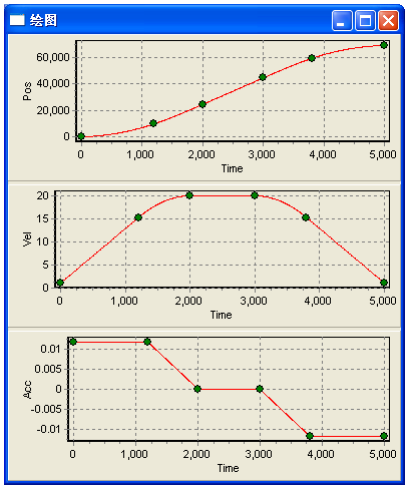


图 5-7-11 PVT 例程速度曲线

可以满足上述要求的一组数据表如下。

数据点	时间（毫秒）	位置（脉冲）	速度（脉冲/毫秒）
P1	0	0	1
P2	1,200	9,750	15.25
P3	2,000	24,483	20
P4	3,000	44,483	20
P5	3,800	59,216	15.25

P6	5,000	68,966	1

```
VAR_GLOBAL CONSTANT
AXIS:INT:=1
TABLE:INT:=1
END_VAR

PROGRAM MAIN
VAR
    Rtn:INT;
    Mask:DINT;
    (* X 轴的数据点参数*)
    AlTime:ARRAY[0..5] OF LREAL:=0,1200,2000,3000,3800,5000;
    Pos:ARRAY[0..5] OF LREAL:=0,9750,24483,44483,59216,68966;
    Vel:ARRAY[0..5] OF LREAL:=1,15.25,20,20,15.25,1;
    prfVel,prfPos,t:LREAL;
    tableId:INT;
    First:BOOL:=TRUE;
END_VAR
-----------------------------------------------------------------------------------------------------
IF First THEN
    rtn := GT_AxisOn(AXIS);

    (*  设置为 PVT 模式*)
    rtn := GT_PrfPvt(AXIS);

    (*  发送数据*)
    rtn := GT_PvtTable(TABLE,6,ADR(alTime[0]),ADR(pos[0]),ADR(vel[0]));

    (*  选择数据表*)
    rtn := GT_PvtTableSelect(AXIS,TABLE);

    mask := SHL(1,(AXIS-1));
    rtn := GT_PvtStart(mask);

    First:=FALSE;
END_IF

(*  读取数据表和运动时间*)
rtn := GT_PvtStatus(AXIS,ADR(tableId),ADR(t),1);

(*  读取规划速度*)
```

```
rtn = GT_GetPrfVel(AXIS,ADR(prfVel),1,0);

(* 读取规划位置*)
rtn = GT_GetPrfPos(AXIS,ADR(prfPos),1,0);
```

4.2.3.2 Complete 描述方式

假设位置和时间之间的关系由函数 $P=40000\sin^2(\pi/2000*t)$确定。要求启动以后能够循环运动，按 A 键幅值增大 50%，按 B 键幅值减小 50%。

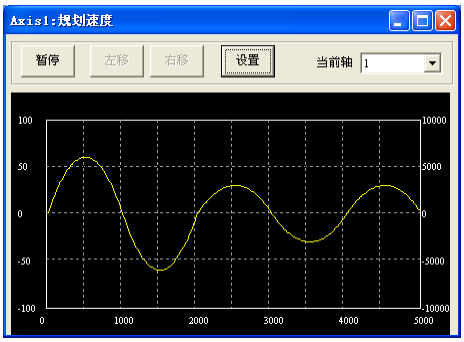


图 5-7-12 速度曲线

```
VAR_GLOBAL CONSTANT
   AXIS:INT:=1
   TABLE1:INT:=1
   TABLE2:INT:=2
   PI:LREAL:=3.1415926
END_VAR

FUNCTION Calculate : INT
VAR_INPUT
   Amplitude:LREAL;
   n:DINT;
   pTime:POINTER TO LREAL;
   pPos:POINTER TO LREAL;
END_VAR

VAR
      i:DINT;
END_VAR
-----------------------------------------------------------------------------------------------------------
FOR i:=0 TO n-1 BY 1 DO
```

```
    pPos[i] := amplitude*sin(PI/2000*pTime[i])*sin(PI/2000*pTime[i]);
END_FOR



PROGRAM MAIN
VAR
    Rtn:INT;
    Mask:DINT;
    (* X 轴的数据点参数*)
    AlTime:ARRAY[0..4] OF LREAL:= 0,500,1000,1500,2000;
    Pos:ARRAY[0..4] OF LREAL;
    a,b,c: ARRAY[0..4] OF LREAL;
    prfVel,prfPos,t:LREAL;
    tableId:INT;
    amplitude:LREAL:=40000;
    table:INT:= TABLE1;
    key:STRING(1);
    First:BOOL:=TRUE;
END_VAR
------------------------------------------------------------------------------------------------------------
IF First THEN
    rtn := GT_AxisOn(AXIS);

    (*  设置为 PVT 模式*)
    rtn := GT_PrfPvt(AXIS);

    Calculate(amplitude,5,ADR(alTime[0]),ADR(pos[0]));

    (*  发送数据*)
    rtn:=GT_PvtTableComplete(table,5,ADR(alTime[0]),ADR(pos[0]),ADR(a[0]),ADR(b[0]),A
DR(c[0]),0,0);

    (*  选择数据表*)
    rtn := GT_PvtTableSelect(AXIS,table);

    (*  设置为循环执行*)
    rtn := GT_SetPvtLoop(AXIS,0);
    mask := SHL(1,(AXIS-1));
    rtn := GT_PvtStart(mask);
    First:=FALSE;
END_IF

(*  读取数据表和运动时间*)
```

```
rtn := GT_PvtStatus(AXIS,ADR(tableId),ADR(t),1);

(*  读取规划速度*)
rtn = GT_GetPrfVel(AXIS,ADR(prfVel),1,0);

(*  读取规划位置*)
rtn = GT_GetPrfPos(AXIS,ADR(prfPos),1,0);

If key<>'' THEN
    IF ( 'A' = key ) THEN
        amplitude := amplitude *1.5;
    END_IF
    IF ( 'B' = key) THEN
        amplitude := amplitude *0.5;
    END_IF
    IF ( 'A' = key ) OR ( 'B' = key ) THEN
        Calculate(amplitude,5,ADR(alTime[0]),ADR(pos[0]));
        table := TABLE1 + TABLE2 - tableId;

        (*发送数据*）
        rtn := GT_PvtTableComplete(table, 5, ADR(alTime[0]), ADR(pos[0]), ADR(a[0]),
ADR(b[0]), ADR(c[0]), 0, 0);

        (*选择数据表*)
        rtn := GT_PvtTableSelect(AXIS,table);
    END_IF

    IF ( 'Q' = key) THEN
        (*停止轴运动*)
        mask := SHL(1,(AXIS-1));
        GT_Stop(mask,0);
    END_IF
END_IF
```

4.2.3.3 Percent 描述方式

X 轴往复运动，Y 轴正向进给。X 轴加减速时 Y 轴开始进给，X 轴匀速运动时，Y 轴保持静止。

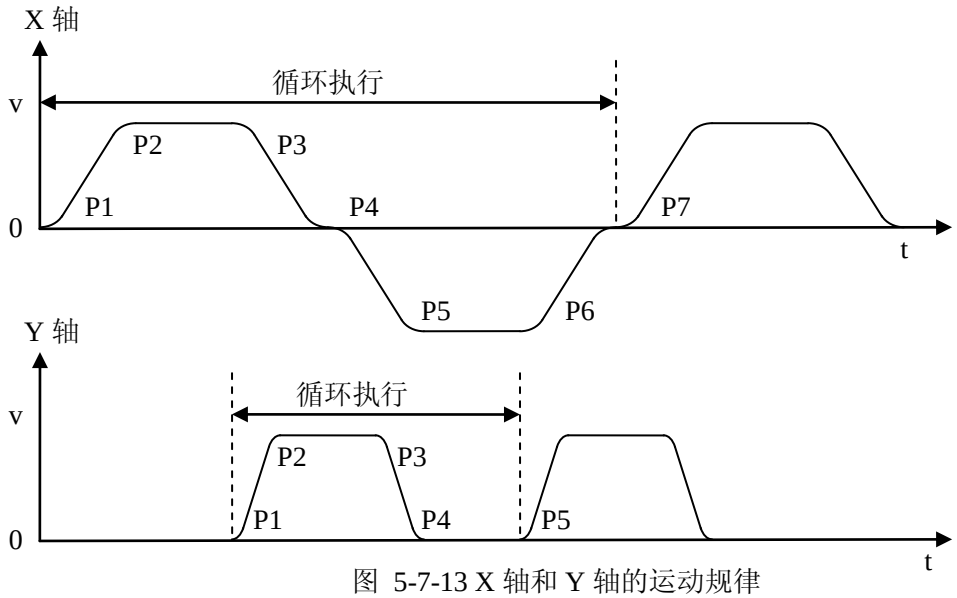


图 5-7-13 X 轴和 Y 轴的运动规律

X 轴取 7 个数据点，设置为循环模式。数据点参数如下：

数据点	时间（毫秒）	位置（脉冲）	百分比	速度（脉冲/毫秒）
P1	0	0	60	0
P2	1,000	5,000	0	不指定
P3	2,000	15,000	60	不指定
P4	3,000	20,000	60	不指定
P5	4,000	15,000	0	不指定
P6	5,000	5,000	60	不指定
P7	6,000	0	0	不指定

根据 X 轴数据点参数，可以计算出各数据点的速度，百分比参数对数据点的速度计算没有影响。

数据点	时间(毫秒)	位置（脉冲）	速度（脉冲/毫秒）
P1	0	0	0
P2	1,000	5,000	2(5000-0)/(1000-0)-0=10
P3	2,000	15,000	2(15000-5000)/(2000-1000)-10=10
P4	3,000	20,000	2(20000-15000)/(3000-2000)-10=0
P5	4,000	15,000	2(15000-20000)/(4000-3000)-0=-10
P6	5,000	5,000	2(5000-15000)/(5000-4000)-(-10)=-10

P7	6,000	0	2(0-5000)/(6000-5000)-(-10)=0

Y 轴取 5 个数据点，设置为循环模式。X 轴启动以后到达数据点 P3 时 Y 轴才启动，因此第 1 个数据点的时间设置为 2000 毫秒。当 Y 轴到达 P5 以后，返回到 P1 循环执行。数据点参数如下：

数据点	时间（毫秒）	位置（脉冲）	百分比	速度（脉冲/毫秒）
P1	2,000	0	60	0
P2	2,500	2,500	0	不指定
P3	3,500	12,500	60	不指定
P4	4,000	15,000	0	不指定
P5	5,000	15,000	0	不指定

根据 Y 轴数据点参数，可以计算出各数据点的速度，百分比参数对数据点的速度计算没有影响。

数据点	时间(毫秒)	位置（脉冲）	速度（脉冲/毫秒）
P1	2,000	0	0
P2	2,500	2,500	2(2500-0)/(2500-2000)-0=10
P3	3,500	12,500	2(12500-2500)/(3500-2500)-10=10
P4	4,000	15,000	2(15000-12500)/(4000-3500)-10=0
P5	5,000	15,000	2(15000-15000)/(5000-4000)-0=0

X 轴循环 n 次，Y 轴需要循环 2n-1 次。下图是当 X 轴的循环次数为 2，Y 轴循环次数为 3 时的 XY 位置图。横轴是 X 轴的位置，纵轴是 Y 轴的位置。黄线是实际的运动轨迹。

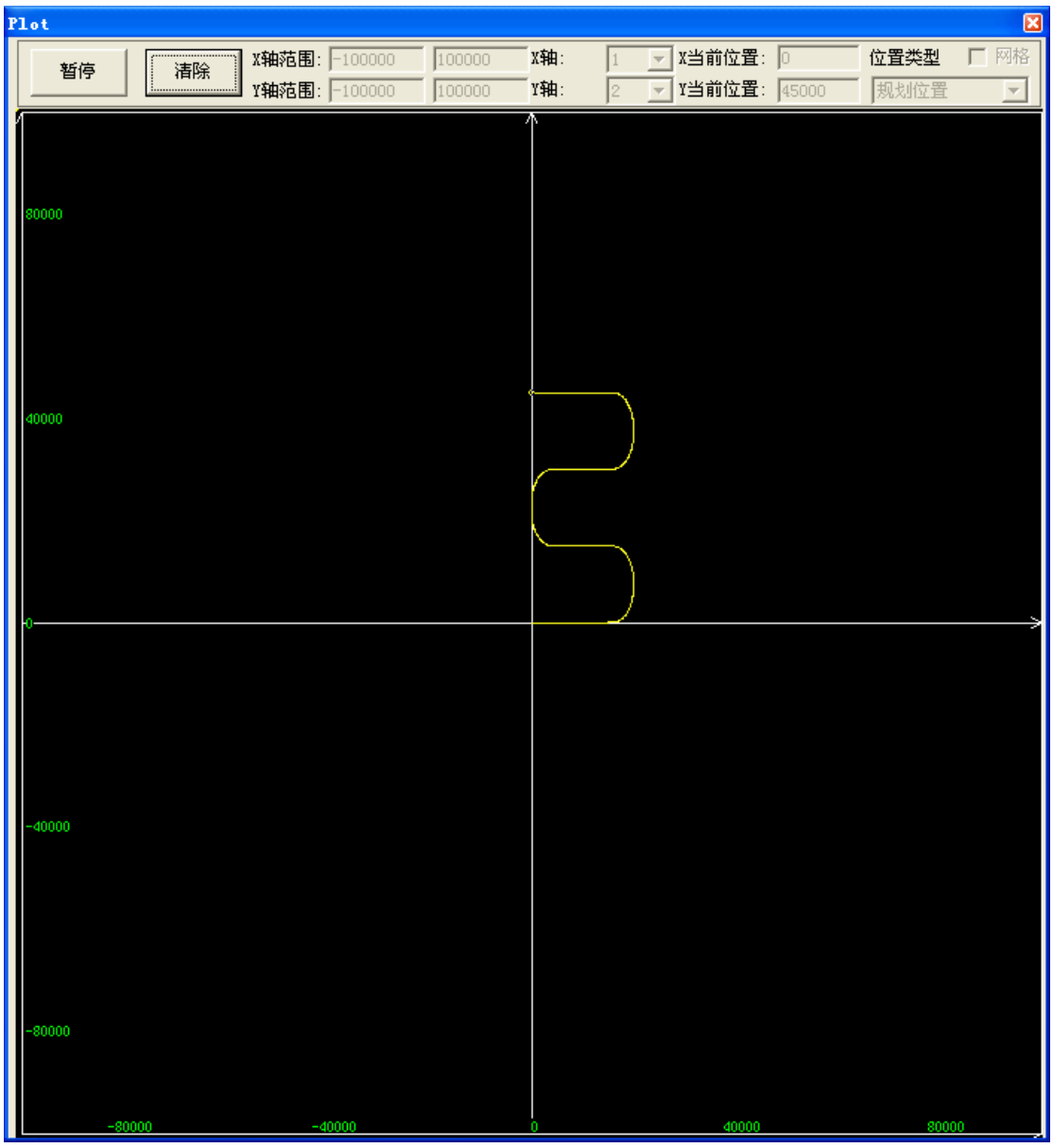


图 5-7-14 X-Y 位置图

```
VAR_GLOBAL CONSTANT
   AXIS_X:INT:=1
   AXIS_Y:INT:=2

   TABLE_X:INT:=1
   TABLE_Y:INT:=2
   LOOP_COUNT:INT:=2;
END_VAR

PROGRAM MAIN
VAR
      Rtn:INT;
      Mask:DINT;
```

```
    (* X 轴的数据点参数*)
    Time_x:ARRAY[0..6] OF LREAL:= 0,1000,2000,3000,4000,5000,6000;
    pos_x:ARRAY[0..6] OF LREAL:= 0,5000,15000,20000,15000,5000,0;
    percent_x:ARRAY[0..6] OF LREAL:= 60,0,60,60,0,60,0;
    Time_y:ARRAY[0..4] OF LREAL:= 2000,2500,3500,4000,5000;
    pos_y:ARRAY[0..4] OF LREAL:= 0,2500,12500,15000,15000;
    percent_y:ARRAY[0..4] OF LREAL:= 60,0,60,0,0;

    prfVel, prfPos, alTime:ARRAY[0..1] OF LREAL;
    tableId:INT;
    First:BOOL:=TRUE;
END_VAR
-----------------------------------------------------------------------------------------------------------
IF First THEN

    rtn := GT_AxisOn(AXIS_X);
    rtn := GT_AxisOn(AXIS_Y);

    (*  将 X 轴设置为 PVT 模式*)
    rtn := GT_PrfPvt(AXIS_X);

    (*  将 Y 轴设置为 PVT 模式*)
    rtn := GT_PrfPvt(AXIS_Y);

    (*  向 X 轴的数据表发送数据*)
rtn := GT_PvtTablePercent(TABLE_X,7,ADR(time_x[0]),ADR(pos_x[0]),ADR(percent_x[0]),0);

    (*  向 Y 轴的数据表发送数据*)
rtn := GT_PvtTablePercent(TABLE_Y,5,ADR(time_y[0]),ADR(pos_y[0]),ADR(percent_y[0]),0);

    (* X 轴选择数据表 TABLE_X*)
    rtn := GT_PvtTableSelect(AXIS_X,TABLE_X);

    (* Y 轴选择数据表 TABLE_Y*)
    rtn := GT_PvtTableSelect(AXIS_Y,TABLE_Y);

    (*  设置循环次数*)
    rtn := GT_SetPvtLoop(AXIS_X,LOOP_COUNT);

    (*  设置循环次数*)
    rtn := GT_SetPvtLoop(AXIS_Y,2*LOOP_COUNT-1);

    (*  同时启动 X 轴和 Y 轴*)
    (*  由于 Y 轴的第 1 个数据点时间为 2000ms*)
```

```
    (* 因此 X 轴启动 2000ms 以后，Y 轴才开始运动*)
    mask := SHL(1,(AXIS_X-1));
    mask := mask OR SHL(1,(AXIS_Y-1));
    rtn := GT_PvtStart(mask);
    First:=FALSE;
END_IF

(* 读取数据表和运动时间*)
rtn := GT_PvtStatus(AXIS_X,ADR(tableId[0]),ADR(alTime[0]),1);
rtn := GT_PvtStatus(AXIS_Y,ADR(tableId[1]),ADR(alTime[1]),1);

(* 读取规划速度*)
rtn := GT_GetPrfVel(AXIS_X,ADR(prfVel[0]),1,0);
rtn := GT_GetPrfVel(AXIS_Y,ADR(prfVel[1]),1,0);

(* 读取规划位置*)
rtn := GT_GetPrfPos(AXIS_X,ADR(prfPos[0]),1,0);
rtn := GT_GetPrfPos(AXIS_Y,ADR(prfPos[1]),1,0);
```

4.2.3.4 Continuous 描述方式

X 轴从 A 点运动到 B 点，Y 轴从 C 点运动到 D 点。要求 X 轴到达 B 点时，Y 轴同时到达 D 点。

首先调用 GT_PvtContinuousCalculate 指令计算 X 轴的运动时间 t_x 和 Y 轴的运动时间 t_y。该指令不会把数据点发送到运动控制器。如果 $t_x>t_y$，Y 轴延时 t_x-t_y 启动；如果 $t_x<t_y$，X 轴延时 t_y-t_x 启动。

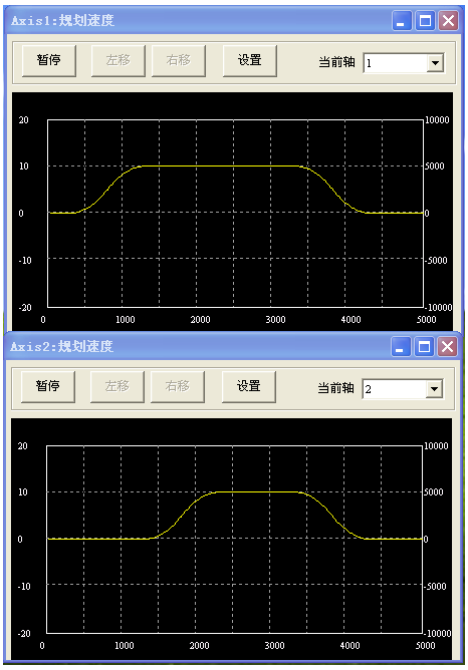


图 5-7-15 X 轴和 Y 轴的速度曲线

```
VAR_GLOBAL CONSTANT
   AXIS_X:INT:=1
   AXIS_Y:INT:=2

   TABLE_X:INT:=1
   TABLE_Y:INT:=2
END_VAR
-------------------------------------------------------------------------------
PROGRAM MAIN
VAR
      Rtn:INT;
      Mask:DINT;
      (* X 轴的数据点参数*)
      pos_x:ARRAY[0..1] OF LREAL:= 0,30000;
      vel_x:ARRAY[0..1] OF LREAL:= 0,0;
      velMax_x:ARRAY[0..1] OF LREAL:= 10,10;
      percent_x:ARRAY[0..6] OF LREAL:=100,100;
      acc_x:ARRAY[0..1] OF LREAL:= 0.01,0.01;
      dec_x:ARRAY[0..1] OF LREAL:= 0.01,0.01;
      time_x:ARRAY[0..1] OF LREAL;
      timeBegin_x:LREAL;

      (* Y 轴的数据点参数*)
      pos_y:ARRAY[0..1] OF LREAL:= 0,20000;
      vel_y:ARRAY[0..1] OF LREAL:= 0,0;
```

```
    velMax_y:ARRAY[0..1] OF LREAL:= 10,10;
    percent_y:ARRAY[0..6] OF LREAL:=100,100;
    acc_y:ARRAY[0..1] OF LREAL:= 0.01,0.01;
    dec_y:ARRAY[0..1] OF LREAL:= 0.01,0.01;
    time_y:ARRAY[0..1] OF LREAL;
    timeBegin_y:LREAL;

    prfVel, prfPos, alTime:ARRAY[0..1] OF LREAL;
    tableId: ARRAY[0..1] OF INT;
    First:BOOL:=TRUE;
END_VAR
-----------------------------------------------------------------------------------------------------------
IF First THEN
    rtn := GT_AxisOn(AXIS_X);

    rtn := GT_AxisOn(AXIS_Y);

    (* 将 X 轴设置为 PVT 模式*)
    rtn := GT_PrfPvt(AXIS_X);

    (* 将 Y 轴设置为 PVT 模式*)
    rtn := GT_PrfPvt(AXIS_Y);

    (* 计算 X 轴运动时间*)
    rtn :=GT_PvtContinuousCalculate(2,ADR(pos_x[0]),ADR(vel_x[0]),ADR(percent_x[0]),AD
R(velMax_x[0]), ADR(acc_x[0]), ADR(dec_x[0]), ADR(time_x[0]));

    (* 计算 Y 轴运动时间*)
    rtn := GT_PvtContinuousCalculate(2, ADR(pos_y[0]), ADR(vel_y[0]), ADR(percent_y[0]),
ADR(velMax_y[0]), ADR(acc_y[0]), ADR(dec_y[0]), ADR(time_y[0]));

    (* 计算启动延时*)
    IF time_x[1] < time_y[1] THEN
        timeBegin_x := time_y[1] - time_x[1];
        timeBegin_y := 0;
    ELSE
        timeBegin_x := 0;
        timeBegin_y := time_x[1] - time_y[1];
    END_IF

    (* 发送 X 轴数据点*)
    rtn := GT_PvtTableContinuous(TABLE_X,2, ADR(pos_x[0]), ADR(vel_x[0]),
ADR(percent_x[0]), ADR(velMax_x[0]), ADR(acc_x[0]), ADR(dec_x[0]),timeBegin_x);
```

```
    (*  发送 Y 轴数据点*)
    rtn := GT_PvtTableContinuous(TABLE_Y,2, ADR(pos_y[0]), ADR(vel_y[0]),
ADR(percent_y[0]), ADR(velMax_y[0]), ADR(acc_y[0]), ADR(dec_y[0]),timeBegin_y);

    (* X 轴选择数据表 TABLE_X*)
    rtn := GT_PvtTableSelect(AXIS_X,TABLE_X);

    (* Y 轴选择数据表 TABLE_Y*)
    rtn := GT_PvtTableSelect(AXIS_Y,TABLE_Y);

    (*  同时启动 X 轴和 Y 轴*)
    (*  由于 Y 轴的第 1 个数据点时间为 2000ms*)
    (*  因此 X 轴启动 2000ms 以后，Y 轴才开始运动*)
    mask := SHL(1,(AXIS_X-1));
    mask := mask OR SHL(1,(AXIS_Y-1));
    rtn := GT_PvtStart(mask);
    First:=FALSE;
END_IF

(*  读取数据表和运动时间*)
rtn := GT_PvtStatus(AXIS_X,ADR(tableId[0]),ADR(alTime[0]),1);
rtn := GT_PvtStatus(AXIS_Y,ADR(tableId[1]),ADR(alTime[1]),1);

(*  读取规划速度*)
rtn := GT_GetPrfVel(AXIS_X,ADR(prfVel[0]),1,0);
rtn := GT_GetPrfVel(AXIS_Y,ADR(prfVel[1]),1,0);

(*  读取规划位置*)
rtn := GT_GetPrfPos(AXIS_X,ADR(prfPos[0]),1,0);
rtn := GT_GetPrfPos(AXIS_Y,ADR(prfPos[1]),1,0);
```

第五章 运动程序

5.1 简介

为了表述方便，直接在 PC 机上调用动态链接库发送指令访问控制器的程序称为“应用程序”，下载到运动控制器上执行的程序称为“运动程序”。

C 语言编写的运动程序编译以后能够下载到运动控制器中执行。运动程序能够脱离主机在运动控制器上独立执行，从而将主机从繁琐的运动逻辑管理中解放出来。一方面主机能够将 CPU 资源分配给其它任务，另一方面运动程序能够直接访问运动控制器，不需要进行频繁通讯，从而具有更高的实时性。

当然，如果需要，主机仍然可以在任何时候向控制器发送指令，即使运动控制器上的运

动程序正在执行。注意：当主机指令和运动控制器上的运动程序控制相同的轴时，需要仔细设计运动逻辑，以免造成混乱。

运动控制器允许多达 32 个运动程序在运动控制器上同时执行。

运动控制器内建的线程调度机制保证多线程环境下运动程序所有指令的执行都是完整的。

在多线程环境下，一个线程中连续的 2 条指令在执行时有可能被插入其它线程的指令。当启动多个线程并行执行时，应当仔细考虑线程之间是否会相互影响。

5.2 编写运动程序

5.2.1 指令列表

运动程序指令列表

指令	说明
GT_Download	下载运动程序到运动控制器
GT_GetFunId	读取运动程序中函数的标识
GT_GetVarId	读取运动程序中变量的标识
GT_Bind	绑定线程、函数、数据页
GT_RunThread	启动线程
GT_StopThread	停止正在运行的线程
GT_PauseThread	暂停正在运行的线程
GT_GetThreadSts	读取线程的状态
GT_SetVarValue	设置运动程序中变量的值
GT_GetVarValue	读取运动程序中变量的值

GT_Download

下载运动程序到运动控制器

GT_Download(pFileName)	
pFileName:POINTER TO STRING	下载到运动控制器的运动程序文件名

GT_GetFunId

读取运动程序中函数的标识

GT_GetFunId(pFunName, pFunId)	
pFunName:POINTER TO STRING	运动程序函数名称

pFunId:POINTER TO INT	根据运动程序函数名称查询函数标识

GT_GetVarId

读取运动程序中变量的标识

GT_GetVarId(pFunName, pVarName, pVarInfo)	
pFunName:POINTER TO STRING	全局变量输入 NULL 局部变量所在函数的名称
pVarName:POINTER TO STRING	运动程序变量名称
pVarInfo:POINTER TO TVarInfo	根据运动程序函数名称和变量名称查询变量标识

GT_Bind

绑定线程，函数，数据页

GT_Bind(thread, funId, page)	
thread:INT	线程编号，取值范围[0,31]。
funId:INT	函数标识，可以调用 GT_GetFunId 查询
page:INT	数据页编号，取值范围[0,31]

GT_RunThread

启动线程

GT_RunThread(thread)	
thread:INT	线程编号，取值范围[0,31]

GT_StopThread

停止正在运行的线程

GT_StopThread(thread)	
thread:INT	线程编号，取值范围[0,31]

GT_PauseThread

暂停正在运行的线程

GT_PauseThread(thread)

thread:INT	线程编号，取值范围[0,31]

GT_GetThreadSts

读取线程的状态

GT_GetThreadSts(thread, pThreadSts)	
thread:INT	线程编号，取值范围[0,31]

pThreadSts:POINTER TO TThreadSts	读取线程状态
	typedef struct ThreadSts
	{
	short run; // 运行状态
	short error; // 指令返回值
	double result; // 函数返回值
	short line; // 当前执行的指令行号
	} TThreadSts;

GT_SetVarValue

设置运动程序中变量的值

GT_SetVarValue(page, pVarInfo, pValue, count=1)	
page:INT	数据页编号 全局变量为-1 局部变量取值范围[0,31]
pVarInfo:POINTER TO TVarInfo	需要访问的变量标识
pValue:POINTER TO LREAL	需要写入的变量值
count:INT	需要写入的变量值的数量，取值范围[1,8]

GT_GetVarValue

读取运动程序中变量的值

GT_GetVarValue(page, pVarInfo, pValue, count=1)	
page:INT	数据页编号 全局变量为-1 局部变量取值范围[0,31]
pVarInfo:POINTER TO TVarInfo	需要访问的变量标识
pValue:POINTER TO LREAL	需要读取的变量值

count:INT	需要读取的变量值的数量，取值范围[1,8]

5.2.2 重点说明

编写运动程序的方法如下：

1) 使用文本编辑器编写运动程序的 C 语言源程序。

2) 使用 MCT2008 编译运动程序，生成目标程序文件（*.bin）和符号文件（*.ini）。

3) 调用 GT_Download 指令将运动程序目标程序下载到运动控制器中。

4) 调用 GT_GetFunId 指令获取函数 ID

5) 调用 GT_GetVarId 指令获取变量 ID

6) 调用 GT_Bind 指令绑定线程、函数和数据页。

7) 调用 GT_SetVarValue 指令初始化局部变量和全局变量。

8) 调用 GT_RunThread 指令，启动线程。

9) 调用 GT_GetThreadSts 指令查询线程状态，或者调用 GT_GetVarValue 指令查询变量。

调用 GT_Download 指令可以将运动程序下载到运动控制器的 SDRAM 中。当下载新的运动程序时会覆盖原有的运动程序。运动控制器每次上电以后需要重新下载运动程序。在发布应用程序时，应同时发布目标文件和符号文件。否则运动程序无法正确下载执行。

运动程序下载到运动控制器以后还不能立即执行，必须调用 GT_Bind 指令绑定线程、函数和数据页，然后调用 GT_RunThread 指令启动线程。运动控制器支持 32 个线程同时运行，一个线程只能分配一个函数，但是一个函数可以分配给多个线程同时执行，例如多轴回零时，可以让多个线程绑定同一个回零函数，然后同时启动这些线程就可以实现多轴同时回零。在线程执行过程中不允许绑定新的函数，除非线程执行完毕。

各函数的局部变量放在相互独立的数据页中。运动控制器提供 32 个数据页。在绑定线程和函数时，必须指明所使用的数据页。一个数据页只能分配给一个线程，但是一个线程可以使用多个数据页。线程在执行过程中可以切换数据页。

应用程序可以随时调用 GT_GetThreadSts 指令查询线程的执行状态。

应用程序可以随时调用 GT_SetVarValue 指令更新运动程序中所有变量的值。

应用程序可以随时调用 GT_GetVarValue 指令查询运动程序中所有变量的值。

运动程序指令返回值定义

返回值	意义	处理方法
0	指令执行成功	
2001	指令执行错误	1. 检查当前指令的执行条件是否满足
2007	指令参数错误	1. 检查当前指令输入参数的取值
2008	打开文件错误	1. 检查变量、函数名称大小写, C 语言编写的

		程序对大小写是敏感的 2. 检查生成的目标文件是否已经下载到指定路径下

5.2.3 例程

5.2.3.1 单线程累加求和

运动程序完成累加求和任务。定义了全局变量 sum 用于保存累加和，局部变量 begin 用于保存累加起点，局部变量 end 用于保存累加终点。累加完成以后程序结束。

```
//---------------------------------------------
//  累加求和
// begin  累加起点
// end    累加终点
//---------------------------------------------
int sum;

int add(int begin,int end)
{
    int i;
    int cc;

    i=begin;
lbl_loop:
    cc = i > end;
    if(cc) goto lbl_end;
    sum = sum + i;
    i = i + 1;
    goto lbl_loop;
lbl_end:
    return sum;
}
```

应用程序负责编译、下载、初始化、启动运动程序。

```
PROGRAM SingleSum
VAR
    rtn: INT;
    sDownloadfilename:STRING:='sum.bin';
    sFunname:STRING:='add';
    iFunid: INT;
```

```
    sVarname:STRING:='sum';
    strSum:TVarInfo;
    strBegin:TVarInfo;
    strEnd:TVarInfo;
    lrValue:LREAL;
    strThread:TThreadSts;
    xFirst: BOOL := TRUE;
END_VAR
--------------------------------------------------------------------------------
IF xFirst THEN

    (*下载运动程序 sum.bin*)
    rtn:=GT_Download(ADR(sDownloadfilename));

    (*获取函数 ID*)
    rtn:=GT_GetFunId(ADR(sFunname), ADR(iFunid));

    (*获取全局变量 sum 的 ID*)
    rtn:=GT_GetVarId(0, ADR(sVarname), ADR(strSum));

    (*获取局部变量 begin 的 ID*)
    sVarname:='begin';
    rtn:=GT_GetVarId(ADR(sFunname), ADR(sVarname), ADR(strBegin));

    (*获取局部变量 end 的 ID*)
    sVarname:='end';
    rtn:=GT_GetVarId(ADR(sFunname), ADR(sVarname), ADR(strEnd));

    (*绑定线程，函数，数据页*)
    rtn:=GT_Bind(0, iFunid, 0);

    lrValue:=0;
    (*初始化运动程序的全局变量 sum*)
    rtn:=GT_SetVarValue(-1, ADR(strSum), ADR(lrValue), 1);

    lrValue:=1;
    (*初始化运动程序的局部变量 begin*)
    rtn:=GT_SetVarValue(0, ADR(strBegin), ADR(lrValue), 1);

    lrValue:=100;
    (*初始化运动程序的局部变量 end*)
    rtn:=GT_SetVarValue(0, ADR(strEnd), ADR(lrValue), 1);

    (*启动线程*)
```

```
    rtn:=GT_RunThread(0);

    xFirst:=FALSE;
END_IF

(*查询线程状态*)
rtn:=GT_GetThreadSts(0,ADR(strThread));

(*查询全局变量 sum 的值*)
rtn:=GT_GetVarValue(-1, ADR(strSum), ADR(lrValue), 1);
```

5.2.3.2 多线程累加求和

运动程序代码和例程 9.2.3.1 相同。

应用程序负责编译、下载、初始化、启动运动程序。和例程 9.2.3.1 不同之处在于启动 2 个线程完成累加运算任务。

```
PROGRAM MultiSum
VAR
    rtn:INT;
    iFunId:INT;
    strSum,strBegin,strEnd:TVarInfo;
    lrValue:LREAL;
    strThread:TThreadSts;
    xFirst: BOOL:=TRUE;
    sDownloadfilename: STRING:='sum.bin';
    sFunname: STRING:='add';
    sVarname:STRING:='sum';
END_VAR
-------------------------------------------------------------------------------
IF xFirst THEN

    (*下载运动程序 sum.bin*)
    rtn:=GT_Download(ADR(sDownloadfilename));

    (*获取函数 ID*)
    rtn:=GT_GetFunId(ADR(sFunname), ADR(iFunId));

    (*获取全局变量 sum 的 ID*)
    rtn:=GT_GetVarId(0, ADR(sVarname), ADR(strSum));

    (*获取局部变量 begin 的 ID*)
    sVarname:='begin';
    rtn:=GT_GetVarId(ADR(sFunname), ADR(sVarname), ADR(strBegin));
```

```
    (*获取局部变量 end 的 ID*)
    sVarname:='end';
    rtn:=GT_GetVarId(ADR(sFunname), ADR(sVarname), ADR(strEnd));

    (*绑定线程，函数，数据页*)
    rtn:=GT_Bind(0, iFunId, 0);
    rtn:=GT_Bind(1, iFunId, 1);

    (*初始化运动程序的全局变量 sum*)
    lrValue:=0;
    rtn:=GT_SetVarValue(-1, ADR(strSum), ADR(lrValue), 1);

    (*初始化线程 0 运动程序的局部变量 begin*)
    lrValue:=1;
    rtn:=GT_SetVarValue(0, ADR(strBegin), ADR(lrValue), 1);

    (*初始化线程 0 运动程序的局部变量 end*)
    lrValue:=50;
    rtn:=GT_SetVarValue(0, ADR(strEnd), ADR(lrValue), 1);

    (*初始化线程 1 运动程序局部变量 begin*)
    lrValue:=51;
    rtn:=GT_SetVarValue(1, ADR(strBegin), ADR(lrValue), 1);

    (*初始化线程 1 运动程序局部变量 end*)
    lrValue:=100;
    rtn:=GT_SetVarValue(1, ADR(strEnd), ADR(lrValue), 1);

    (*启动线程 0, 1*)
    rtn:=GT_RunThread(0);
    rtn:=GT_RunThread(1);

    xFirst:=FALSE;
END_IF

(*查询线程状态*)
rtn:=GT_GetThreadSts(0, ADR(strThread));
rtn:=GT_GetThreadSts(1, ADR(strThread));

(*查询全局变量 sum 的值*)
rtn:=GT_GetVarValue(-1, ADR(strSum), ADR(lrValue), 1);
```

5.2.3.3 组合运动

该例程实现 3 个运动轴协调运动，其应用场景为半导体加工设备，定义 3 个运动轴分别为摆臂轴、送料轴和点胶轴。其中各个轴的运动关系如下所述。

摆臂轴：点胶轴离开工作区域时，启动摆臂轴向工作区域运动，完成工作后无需等待任何信号，即可撤离工作区域。

送料轴：摆臂轴离开工作区域时，即可启动送料轴运动。

点胶轴：摆臂轴离开工作区域之后，启动点胶轴向工作区域运动；当送料轴运动到位时，点胶轴完成点胶工作，然后离开工作区域。

运动程序将三个轴的运动分别用三个函数来实现，ArmMotion 控制摆臂轴，GlueMotion 控制点胶轴，PieceMotion 控制物料轴，通过四个全局同步变量，来实现各个运动之间的互斥和关联。

pieceArrival：标志送料轴已经到达，可以启动点胶轴的运动离开工作区域。

pieceStart：启动送料轴标志位，标志送料轴可以启动。

glueStart：启动点胶轴标志位，标志点胶轴可以开始运动到工作区域。

armStart：启动摆臂轴标志位，标志摆臂轴可以开始向工作区域运动。

ArmMotion 函数与线程 0 和数据页 0 绑定；GlueMotion 函数与线程 1 和数据页 1 绑定；PieceMotion 函数与线程 2 和数据页 2 绑定。每个线程独立控制一个轴的运动。运动程序的变量初始值设置如下：

全局变量 pieceArrival 值：0；(物料轴未到达)

全局变量 armStart 值：0；(初始状态下摆臂轴运动条件不满足)

全局变量 pieceStart 值：1；(初始状态下物料轴运动条件满足)

全局变量 glueStart 值：1；(初始状态下点胶轴运动条件满足)

下图中 1 轴是摆臂轴规划速度，2 轴是点胶轴规划速度，3 轴是物料轴规划速度。

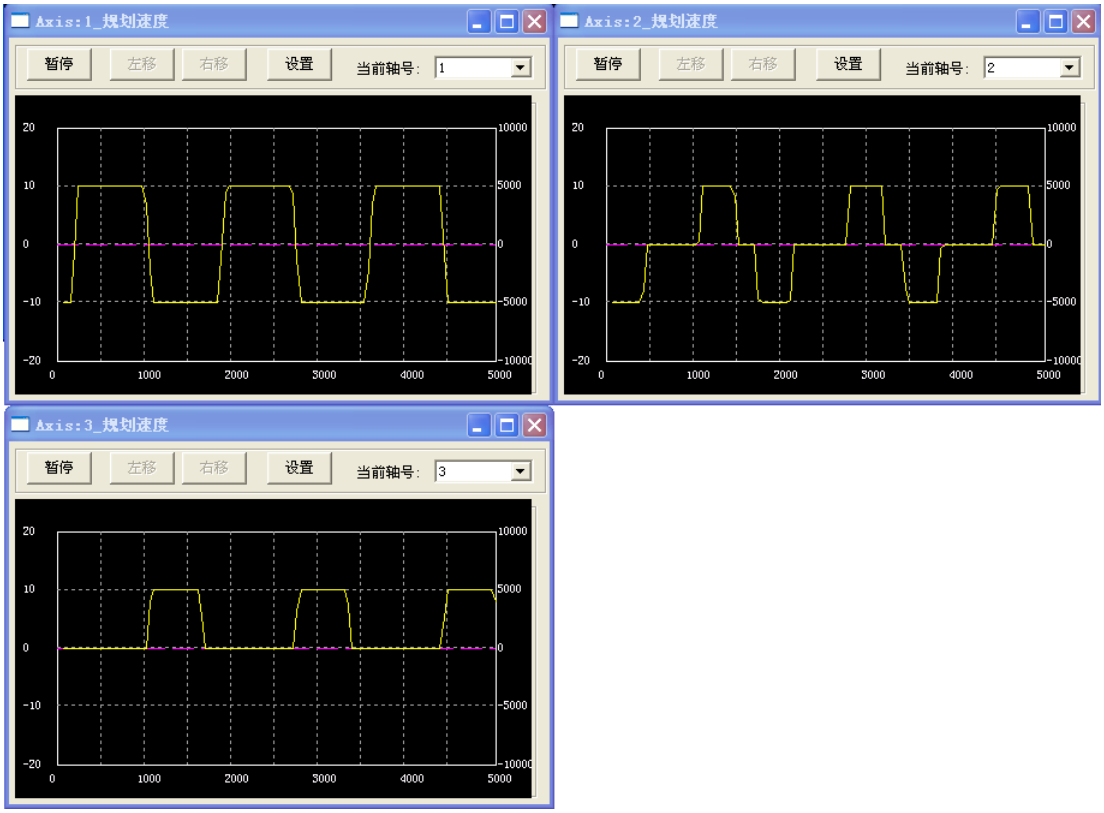


图 9-1 组合运动时序图

运动程序代码如下：

```
//-----------------------------------------------
// 组合运动
//-----------------------------------------------

int pieceArrival;       //物料轴到达标志
int pieceStart;             //物料轴启动标志
int glueStart;              //点胶轴启动标志
int armStart;           //摆臂轴启动标志

// 摆臂轴运动
void ArmMotion(short arm)
{
     long armStep;
     short cc;
    long clock;
    long sts;
    short axis;

lbl_loop:
     axis=arm-1;
```

```
    axis=1<<axis;

    // 等待摆臂启动标志
lbl_wait_for_arm_start:
    cc = !armStart;
    if(cc) goto lbl_wait_for_arm_start;

    // 清除摆臂启动标志
    armStart = 0;

    // 摆臂轴运动
    GT_SetPos(arm,armStep);
    GT_Update(axis);

    // 等待摆臂轴运动停止
lbl_wait_for_arm_stop:
    GT_GetSts(arm,&sts,1,&clock);
    cc = sts & 0x400;
    if(cc) goto lbl_wait_for_arm_stop;

    // 摆臂轴返回
    GT_SetPos(arm,0);
    GT_Update(axis);

    // 置起点胶轴启动标志
    glueStart = 1;

    // 置起物料轴启动标志
    pieceStart = 1;

    // 等待摆臂轴运动停止
lbl_wait_for_arm_return:
    GT_GetSts(arm,&sts,1,&clock);
    cc = sts & 0x400;
    if(cc) goto lbl_wait_for_arm_return;

   goto lbl_loop;
}

// 点胶轴运动
void GlueMotion(short glue)
{
    long glueStep;
    short cc;
```

```
    long clock;
    long sts;
    short axis;

lbl_loop:
     axis=glue-1;
     axis=1<<axis;

     // 等待点胶轴启动标志
lbl_wait_for_glue_start:
     cc = !glueStart;
     if (cc) goto lbl_wait_for_glue_start;

     // 清除点胶轴启动标志
    glueStart = 0;

    // 点胶轴运动
    GT_SetPos(glue,glueStep);
    GT_Update(axis);

    // 等待点胶轴运动停止
lbl_wait_for_glue_stop:
    GT_GetSts(glue,&sts,1,&clock);
    cc = sts & 0x400;
    if(cc) goto lbl_wait_for_glue_stop;

    // 等待物料轴到达
lbl_wait_for_piece_arrival:
     cc = !pieceArrival;
     if(cc) goto lbl_wait_for_piece_arrival;

    // 点胶轴返回
    GT_SetPos(glue,0);
    GT_Update(axis);

     // 置起摆臂轴启动标志
    armStart = 1;

    // 等待运动停止
lbl_wait_for_glue_return:
    GT_GetSts(glue,&sts,1,&clock);
    cc = sts & 0x400;
    if(cc) goto lbl_wait_for_glue_return;
```

```
    goto lbl_loop;
}

// 物料轴运动
void PieceMotion(short piece)
{
     long pieceStep;
     short cc;
    long clock;
    long sts;
    long pos;
    short axis;
    pos = 0;

lbl_loop:
     axis=piece-1;
     axis=1<<axis;

     // 等待启动物料标志
lbl_wait_for_piece_start:
     cc = !pieceStart;
     if (cc) goto lbl_wait_for_piece_start;

     // 清除物料轴启动标志
    pieceStart = 0;

     // 清除物料轴到达标志
     pieceArrival = 0;

    // 启动物料轴运动
    pos = pos + pieceStep;
    GT_SetPos(piece,pos);
    GT_Update(axis);

     // 等待物料轴运动停止
lbl_wait_for_piece_stop:
    GT_GetSts(piece,&sts,1,&clock);
    cc = sts & 0x400;
    if(cc) goto lbl_wait_for_piece_stop;

     // 置起物料轴到达标志
    pieceArrival = 1;

    goto lbl_loop;
```

}

应用程序负责下载、初始化、启动运动程序。代码如下：

```
PROGRAM MultiMotion
VAR
    iARMAxis:INT:=1;
    iGLUEAxis:INT:=2;
    iPIECEAxis:INT:=3;
    lrARM_VEL:LREAL:=10;
    lrGLUE_VEL:LREAL:=10;
    lrPIECE_VEL:LREAL:=10;
    diARM_STEP:DINT:=6000;
    diGLUE_STEP:DINT:=4000;
    diPIECE_STEP:DINT:=8000;

    rtn:INT;
    strTrap:TTrapPrm;
    iArmMotion,iGlueMotion,iPieceMotion:INT;
    strPieceArrival,strPieceStart,strGlueStart,strArmStart:TVarInfo;
    strArm,strArmStep,strGlue,strGlueStep,strPiece,strPieceStep:TVarInfo;
    lrValue:LREAL;
    prfPos:ARRAY [1..8] OF LREAL;

    sDownloadfilename:STRING:='led.bin';
    sFunname:STRING;
    sVarname:STRING;
    xFirst: BOOL:=TRUE;
END_VAR
-------------------------------------------------------------------------------
IF xFirst THEN
    (*清除报警和限位*)
    rtn:=GT_ClrSts(1,8);
    rtn:=GT_AxisOn(1);
    rtn:=GT_AxisOn(2);
    rtn:=GT_AxisOn(3);

    (*设置 ARM 运动参数*)
    rtn:=GT_PrfTrap(iARMAxis);
    rtn:=GT_GetTrapPrm(iARMAxis,ADR(strTrap));
    strTrap.acc:=0.25;
    strTrap.dec:=0.25;
    rtn:=GT_SetTrapPrm(iARMAxis,ADR(strTrap));
    rtn:=GT_SetVel(iARMAxis,lrARM_VEL);
```

```
rtn:=GT_Update(SHL(DWORD#1,iARMAxis-1));

(*设置 GLUE 运动参数*)
rtn:=GT_PrfTrap(iGLUEAxis);
rtn:=GT_GetTrapPrm(iGLUEAxis, ADR(strTrap));
strTrap.acc:=0.25;
strTrap.dec:=0.25;
rtn:=GT_SetTrapPrm(iGLUEAxis, ADR(strTrap));
rtn:=GT_SetVel(iGLUEAxis, lrGLUE_VEL);
rtn:=GT_Update(SHL(DWORD#1,iGLUEAxis-1));

(*设置 PIECE 运动参数*)
rtn:=GT_PrfTrap(iPIECEAxis);
rtn:=GT_GetTrapPrm(iPIECEAxis, ADR(strTrap));
strTrap.acc:=0.25;
strTrap.dec:=0.25;
rtn:=GT_SetTrapPrm(iPIECEAxis, ADR(strTrap));
rtn:=GT_SetVel(iPIECEAxis, lrPIECE_VEL);
rtn:=GT_Update(SHL(DWORD#1,iPIECEAxis-1));

(*下载运动程序*)
rtn:=GT_Download(ADR(sDownloadfilename));

(*获取函数 ID*)
sFunname:='ArmMotion';
rtn:=GT_GetFunId(ADR(sFunname),ADR(iArmMotion));
sFunname:='GlueMotion';
rtn:=GT_GetFunId(ADR(sFunname), ADR(iGlueMotion));
sFunname:='PieceMotion';
rtn:=GT_GetFunId(ADR(sFunname),ADR(iPieceMotion));

(*获取全局变量 ID*)
sVarname:='pieceArrival';
rtn:=GT_GetVarId(0, ADR(sVarname), ADR(strPieceArrival));
sVarname:='pieceStart';
rtn:=GT_GetVarId(0, ADR(sVarname), ADR(strPieceStart));
sVarname:='glueStart';
rtn:=GT_GetVarId(0, ADR(sVarname), ADR(strGlueStart));
sVarname:='pieceStart';
rtn:=GT_GetVarId(0, ADR(sVarname), ADR(strPieceStart));

(*获取局部变量 ID*)
sFunname:='ArmMotion';
sVarname:='arm';
```

```
rtn:=GT_GetVarId(ADR(sFunname),ADR(sVarname), ADR(strArm));
sVarname:='armStep';
rtn:=GT_GetVarId(ADR(sFunname),ADR(sVarname), ADR(strArmStep));

sFunname:='GlueMotion';
sVarname:='glue';
rtn:=GT_GetVarId(ADR(sFunname), ADR(sVarname), ADR(strGlue));
sVarname:='glueStep';
rtn:=GT_GetVarId(ADR(sFunname), ADR(sVarname), ADR(strGlueStep));

sFunname:='PieceMotion';
sVarname:='piece';
rtn:=GT_GetVarId(ADR(sFunname), ADR(sVarname), ADR(strPiece));
sVarname:='pieceStep';
rtn:=GT_GetVarId(ADR(sFunname), ADR(sVarname), ADR(strPieceStep));

(*绑定线程，函数，数据页*)
rtn:=GT_Bind(0, iArmMotion, 0);
rtn:=GT_Bind(1, iGlueMotion, 1);
rtn:=GT_Bind(2, iPieceMotion, 2);

(*初始化运动程序的全局变量*)
lrValue:=0;
rtn:=GT_SetVarValue(-1, ADR(strPieceArrival), ADR(lrValue), 1);
rtn:=GT_SetVarValue(-1, ADR(strArmStart), ADR(lrValue), 1);

lrValue:=1;
rtn:=GT_SetVarValue(-1, ADR(strPieceStart), ADR(lrValue), 1);
rtn:=GT_SetVarValue(-1, ADR(strGlueStart), ADR(lrValue), 1);

(*初始化运动程序局部变量*)
lrValue:=INT_TO_LREAL(iARMAxis);
rtn:=GT_SetVarValue(0, ADR(strArm), ADR(lrValue), 1);
lrValue:=DINT_TO_LREAL(diARM_STEP);
rtn:=GT_SetVarValue(0, ADR(strArmStep), ADR(lrValue), 1);

lrValue:=INT_TO_LREAL(iGLUEAxis);
rtn:=GT_SetVarValue(1, ADR(strGlue), ADR(lrValue), 1);
lrValue:=DINT_TO_LREAL(diGLUE_STEP);
rtn:=GT_SetVarValue(1, ADR(strGlueStep), ADR(lrValue), 1);

lrValue:=INT_TO_LREAL(iPIECEAxis);
rtn:=GT_SetVarValue(2, ADR(strPiece), ADR(lrValue), 1);
lrValue:=DINT_TO_LREAL(diPIECE_STEP);
```

```
    rtn:=GT_SetVarValue(2, ADR(strPieceStep), ADR(lrValue), 1);

    (*启动线程*)
    rtn:=GT_RunThread(0);
    rtn:=GT_RunThread(1);
    rtn:=GT_RunThread(2);

    xFirst:=FALSE;
END_IF

GT_GetPrfPos(1, ADR(prfpos), 8 ,0);
```

5.3 语言元素

5.3.1 数据类型

支持整型和浮点型 2 种数据类型。

整型 32 位，取值范围是-2,147,483,648 ~ 2,147,483,647。

浮点型采用定点格式，32 位整数，16 位小数。所能表示的最小精度为(1/2)^16=0.0000152587890625。

5.3.2 常量

可以在程序中直接使用立即数和宏。立即数可以是 10 进制整数、16 进制整数和浮点数。

5.3.3 变量

可以声明局部变量和全局变量。每个函数最多可声明 1024 个局部变量。全局变量最多可声明 1024 个。整型类型说明符为 int。浮点型类型说明符为 double。

5.3.4 数组

支持一维数组，支持常量下标索引和变量下标索引。

不支多维数组，不支持用数组元素进行下标索引。

5.3.5 函数

函数可以定义返回值类型和输入形参类型。

不支持在函数中调用自定义函数，但是可以调用 GT 运动控制指令。

5.3.6 数据类型转换

支持强制数据类型转换。强制数据类型转换符有 int,double。

1. 数据类型转换符必须加括号，如 a=(int)b
2. 数据类型转换不会改变变量本身的数据类型定义

5.4 运算指令

支持算术运算、逻辑运算、关系运算、位运算，语法规则和 C 语言相同，但是不支持复杂表达式，只能使用 2 个操作数进行运算，而且这 2 个操作数的数据类型必须相同。

注意：由于运动程序中的浮点数据类型只有 16 位小数精度，请不要在运动程序中进行高精度浮点运算。

5.4.1 算数运算

用于各类数值运算。包括加(+)、减(-)、乘(*)、除(/)、求余(或称模运算，%)共五种。

5.4.2 逻辑运算

逻辑运算包括与(&&)、或(||)、非(!)三种。参数可以是整型变量、或者整型常数。

5.4.3 关系运算

关系运算符用于比较运算。包括大于(>)、小于(<)、等于(==)、 大于等于(>=)、小于等于(<=)和不等于(!=)六种。参与比较的参数类型必须一致。

5.4.4 位运算

参与运算的量，按二进制位进行运算。包括位与(&)、位或(|)、位非(~)、位异或(^)、左移(<<)、右移(>>)六种。

5.5 流程控制

支持条件跳转、条件返回，语法规则和 C 语言相同。

条件跳转如下所示：

if(var) goto label；

当条件变量 var 非 0 时，跳转到标记为 label 的指令。

不支持表达式作为判断条件。

条件返回如下所示：

if(var) return value；

当条件变量 var 非 0 时，程序返回，返回值为 value。

不支持表达式作为判断条件

CPAC-OtoStudio PLCOpen_Fun 运动控制库使用手册

1 前言

PLCOpen_Fun.lib是OtoStudio上针对不同用户需求开发的运动控制的专用库。

该文档描述了该库提供的功能以及其函数的功能和使用方法。

2 使用方法

PLCOpen_Fun库的使用方法非常简单。所需的步骤在OtoStudio工程中的描述如下。

2.1 创建一个 OtoStudio 工程

- 打开OtoStudio软件
- 创建一个新的工程通过“文件/新建”
- 选择对应的Otobox控制器，如：CPAC-X00-TPX控制器
- 创建新的POU “PLC_PRG” (选择编程语言，如：FBD).

2.2 添加库

- 打开资源->库文件管理器
- 右键添加库：PLCOpen_Fun.LIB
- 库CPAC GUC-X00-TPX.lib将会自动被添加. 如果没有被自动添加，请重复上一步操作，将这个库也添加进来

2.3 函数功能列表

函数名	功能
MC_Power	控制伺服使能
MC_Stop	停止轴的规划运动
MC_Reset	清除目标轴状态
MC_Home	实现单轴回零运动全过程
MC_MoveVelocity	给定速度值，持续运动，并且可以实时更新速度。
MC_MoveAbsolute	控制轴运动到指定的绝对位置
MC_MoveRelative	控制轴从当前位置运动到指定距离
MC_PositionProfile	通过给定一组“位置-时间”参数控制目标轴运动
MC_ReadActualPosition	读取目标轴的实际位置
MC_ReadActualVelocity	读取目标轴的实际速度
MC_ReadStatus	读取目标轴的运动状态
MC_CamIn	设置并启动目标轴凸轮跟随运动
MC_CamOut	使目标轴退出凸轮跟随
MC_GearIn	设置并启动目标轴齿轮跟随运动
MC_GearOut	使目标轴退出齿轮跟随

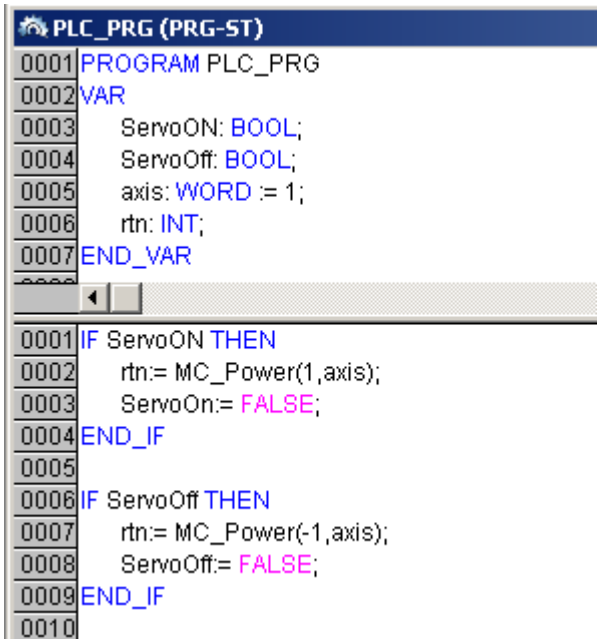
2.4 功能函数的参数介绍

MC_Power

该函数（INT型）控制电机伺服使能打开或者关闭。完成时返回1，否者返回-1。

输入参数	类型	意义
Enable	INT	使能信号： Enable=1 时打开驱动器使能； Enable=-1 时关闭驱动器使能。
Axis	WORD	轴号

参数声明示例：



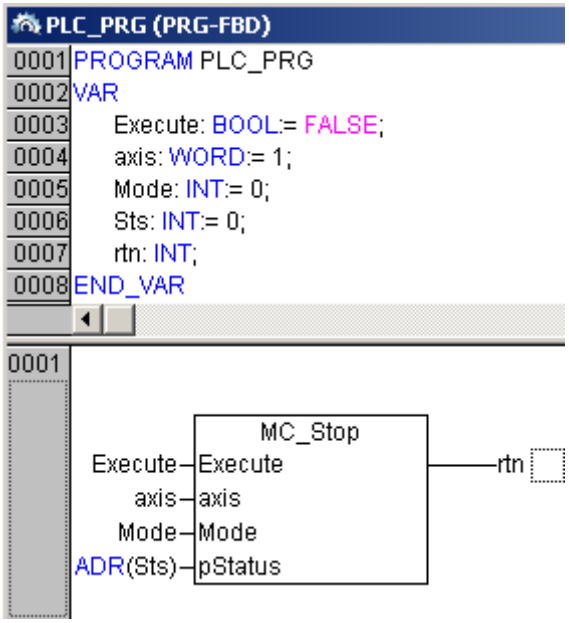
```
PLC_PRG (PRG-ST)
0001 PROGRAM PLC_PRG
0002 VAR
0003     ServoON: BOOL;
0004     ServoOff: BOOL;
0005     axis: WORD := 1;
0006     rtn: INT;
0007 END_VAR

0001 IF ServoON THEN
0002     rtn:= MC_Power(1,axis);
0003     ServoOn:= FALSE;
0004 END_IF
0005
0006 IF ServoOff THEN
0007     rtn:= MC_Power(-1,axis);
0008     ServoOff:= FALSE;
0009 END_IF
0010
```

MC_Stop

该函数（INT型）用于停止目标轴的规划运动。停止运动中Status=-1，运动正常结束后MC_Stop=1（保持一个周期），Status=1（一直保持）

输入参数	类型	意义
Execute	BOOL	使能信号，上升沿触发
Axis	WORD	轴号
Mode	INT	停止模式： 当 Mode=0 时，平滑停止目标轴； 当 Mode=1 时，急停目标轴
pStatus	POINTER TO INT	记录停止过程的状态，初次调用时该指针指向的值必须为 0； 当 Execute=FALSE 时，pStatus^=0

参数声明示例：

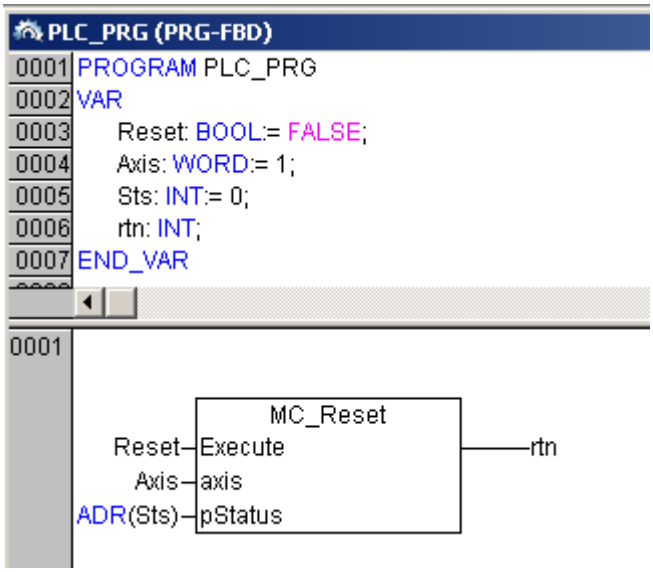


MC_Reset

该函数（INT 型）用于清除目标轴状态。清除状态完成返回 1，否则返回-1。

输入参数	类型	意义
Execute	BOOL	使能信号，上升沿触发
Axis	WORD	轴号
pStatus	POINTER TO INT	该指针指向状态标志 Status，初次调用时该指针指向的值必须为 0； 当 Execute=FALSE 时，pStatus^=0

参数声明示例：



MC_Home

该函数（INT 型）用于实现单轴回零运动全过程，捕获方式为 Home 捕获和 Home&Index 捕获。当捕获成功时，返回值为 3；当返回值为 1 时，表示没有找到 Home 信号 ;当返回值为 2 时 ,表示找到了 Home 信号 ,但是寻找 Index 信号失败。

输入参数	类型	意义
Execute	BOOL	使能信号，高电平有效
axis	WORD	轴号
Mode	INT	编码器捕获模式： Mode=0：Home 捕获； Mode=1：Home&Index 捕获
High_Vel	LREAL	寻找原点的速度，单位“脉冲/毫秒”
Low_Vel	LREAL	到原点的定位速度，单位“脉冲/毫秒”
acc	LREAL	加速度值，单位“脉冲/毫秒 ^{2}”
pPos	POINTER TO DINT	用于保存内部捕获的位置值，该指针指向的值必须由用户初始化为一个绝对值较大的值，单位“毫秒” （如果原点在当前位置的负方向，该值为负，否则为正）
offset	DINT	回零偏移量，单位“脉冲”
pStatus	POINTER TO INT	记录回零过程的状态，初次调用时该指针指向的值必须为 0； 当 Execute=FALSE 时，pStatus^=0

回零过程状态标志 Status 的意义：

Home 回零点过程状态描述各个阶段意义：

1：没有找到 HOME 开关；

2：找到 HOME 开关，运动停止；

3：启动回 HOME 运动；

4：回 HOME 运动中；

5：Home 回零完成；

Home&Index 回零点过程状态描述各个阶段意义：

1：没有找到 HOME 开关

2：找到 HOME 开关，运动停止

3：回 HOME 运动

4：HOME 回零完成

5：启动 INDEX 回零

6：没有找到 INDEX 开关

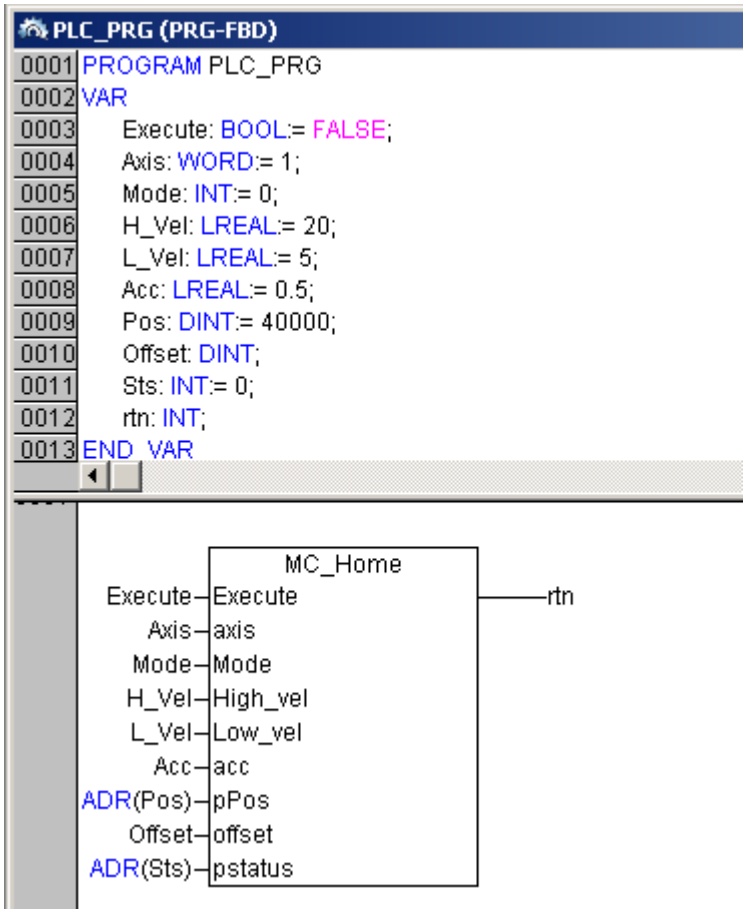
7：找到 INDEX 开关，运动停止

8：启动回 INDEX 运动

9：回 INDEX 运动中

10：Home&Index 回零完成

参数声明示例：



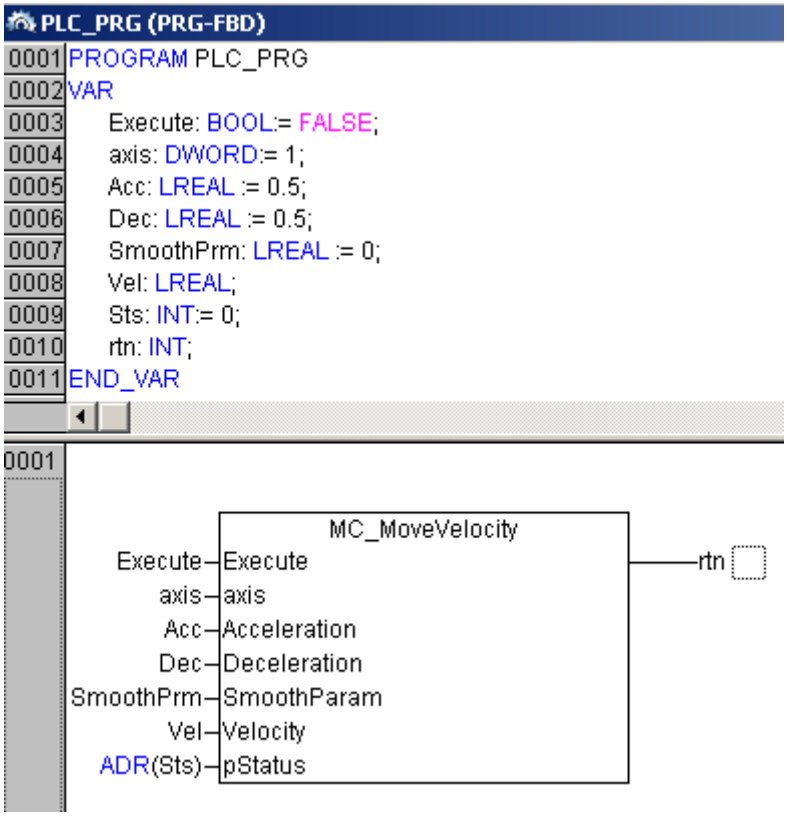
MC_MoveVelocity

该函数（INT 型）用于控制目标轴根据给定的速度持续运动。可以实时更新目标轴的速度值。

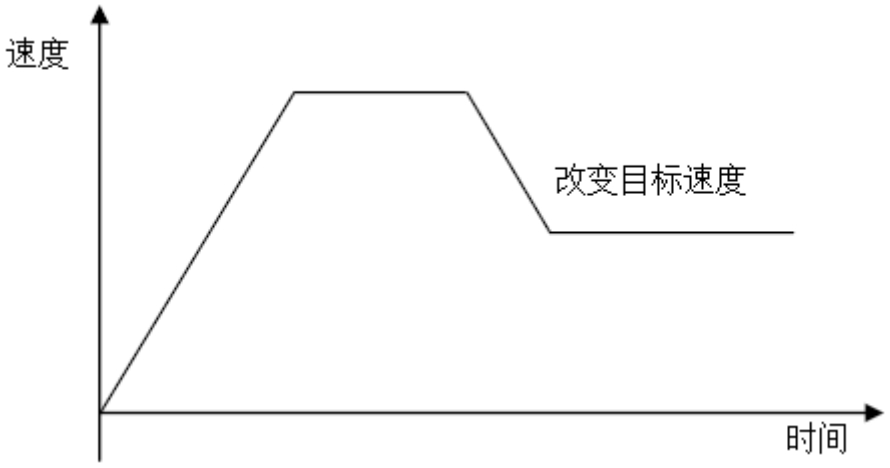
输入参数	类型	意义
Execute	BOOL	使能信号，上升沿触发
axis	WORD	轴号
Acceleration	LREAL	加速度值，单位“脉冲/毫秒 ^{2}”
Deceleration	LREAL	减速度值，单位“脉冲/毫秒 ^{2}”

SmoothParam	LREAL	平滑系数，取值范围:[0,1)，越接近 1，加速度变化越平稳
Velocity	LREAL	目标速度值，单位“脉冲/毫秒”
pStatus	POINTER TO INT	该指针指向运动状态标志 Status，初次调用时该指针指向的值必须为 0； 当 Execute=FALSE 时，pStatus^=0

参数声明示例：



速度曲线：

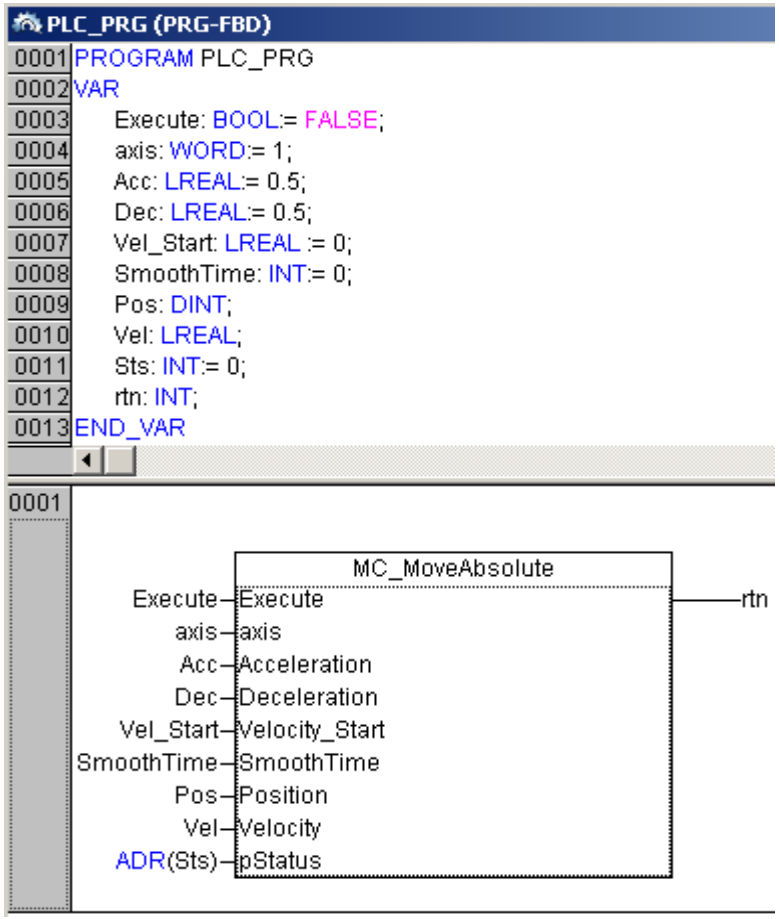


MC_MoveAbsolute

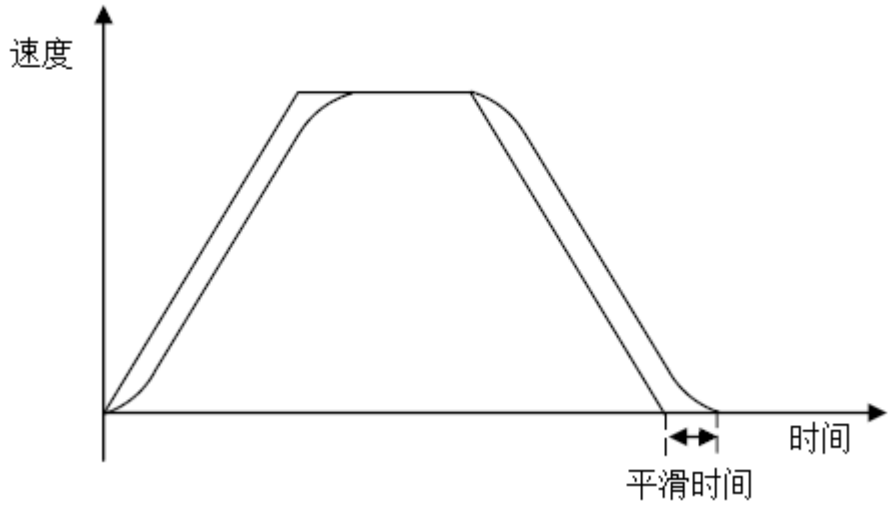
该函数（INT 型）控制轴以梯形曲线速度规划运动到指定的绝对位置。运动中 Status=-1，运动正常结束后 MC_MoveAbsolute=1（保持一个周期），Status=1（一直保持）。

输入参数	类型	意义
Execute	BOOL	使能信号，上升沿触发
axis	WORD	轴号
Acceleration	LREAL	设置加速度值，单位“脉冲/毫秒 ^{2}”
Deceleration	LREAL	设置减速度值，单位“脉冲/毫秒 ^{2}”
Velocity_Start	LREAL	设置起跳速度，单位“脉冲/毫秒”
SmoothTime	INT	设置平滑时间，即加速度变化时间 单位“毫秒”，取值范围:[0,50]
Position	DINT	设置目标位置，单位“脉冲”
Velocity	LREAL	设置最大速度值，单位“脉冲/毫秒”
pStatus	POINTER TO INT	记录运动过程的状态，初次调用时该指针指向的值必须为 0； 当 Execute=FALSE 时，pStatus^=0

参数声明示例：



速度曲线：

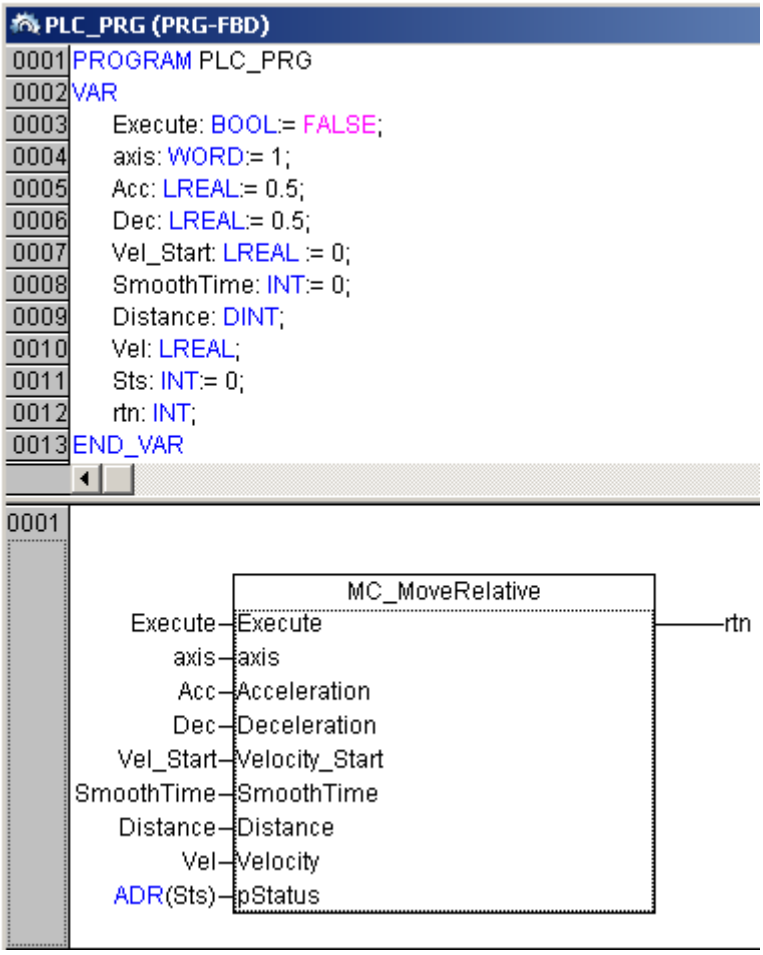


MC_MoveRelative

该函数（INT 型）控制轴以梯形曲线速度规划从当前位置运动到指定距离。运动中 Status=-1，运动完成后，MC_MoveRelative=1（保持一个周期）,Status=1（一直保持）。

输入参数	类型	意义
Execute	BOOL	使能信号，上升沿触发
Axis	WORD	轴号
Acceleration	LREAL	设置加速度值，单位“脉冲/毫秒 ^{2}”
Deceleration	LREAL	设置减速度值，单位“脉冲/毫秒 ^{2}”
Velocity_Start	LREAL	设置起跳速度，单位“脉冲/毫秒”
SmoothTime	INT	设置平滑时间，即加速度变化时间 单位“毫秒”，取值范围:[0,50]
Distance	DINT	设置目标距离，单位“脉冲”
Velocity	LREAL	设置最大速度值，单位“脉冲/毫秒”
pStatus	POINTER TO INT	记录运动过程的状态，初次调用时该指针指向的值必须为 0； 当 Execute=FALSE 时，pStatus^=0

参数声明示例：



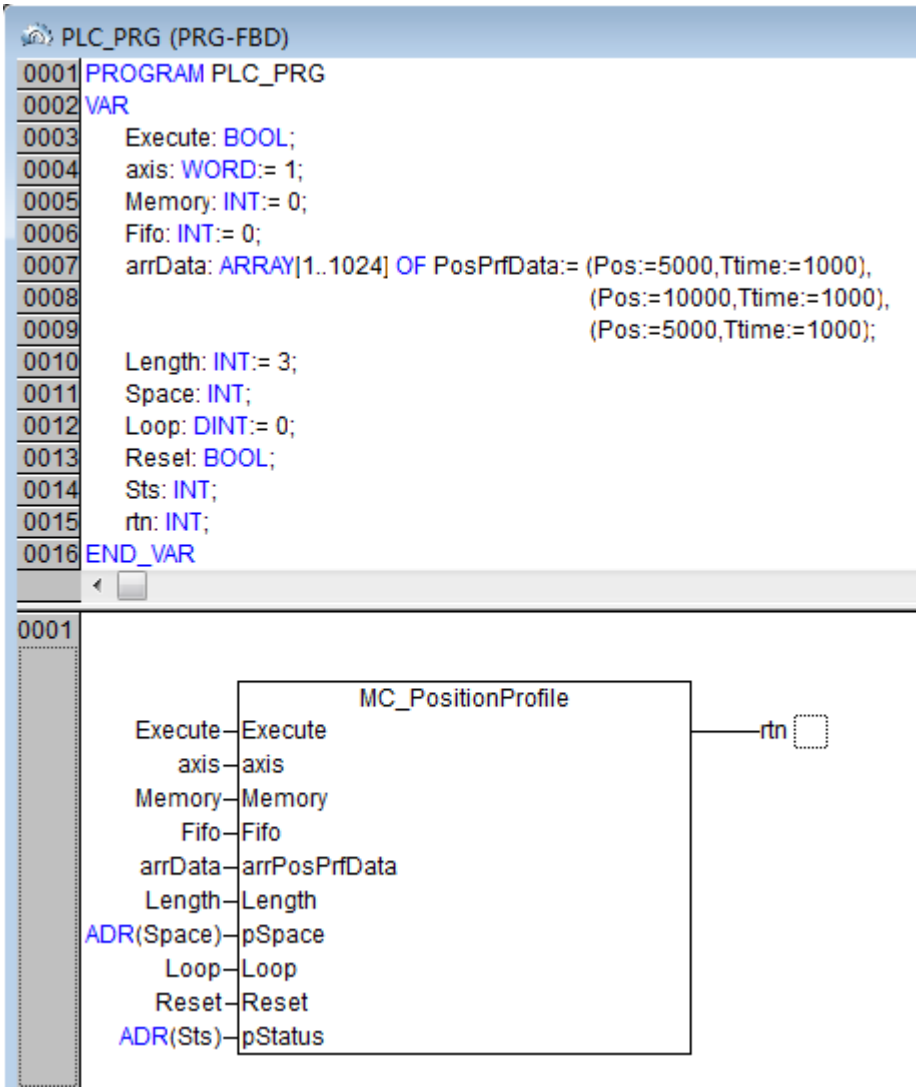
MC_PositionProfile

该函数（INT 型）通过给定一组“位置、时间”参数控制目标轴运动。

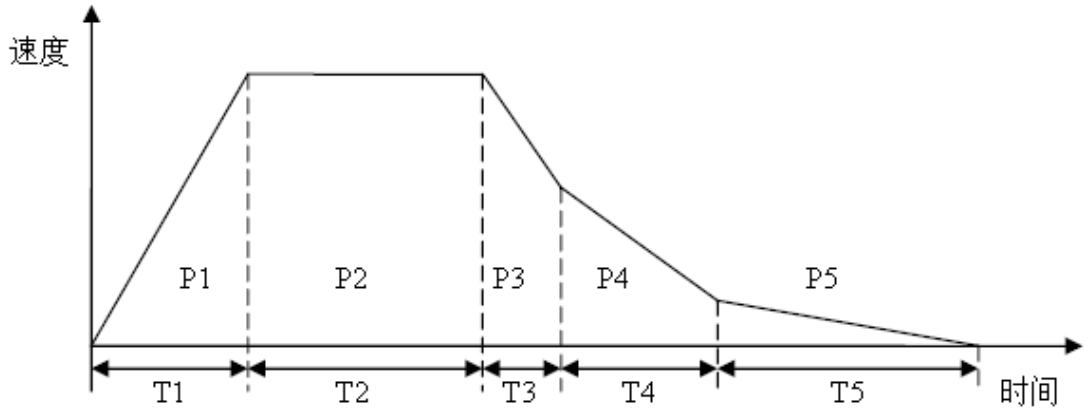
输入参数	类型	意义
Execute	BOOL	使能信号，上升沿触发
Axis	WORD	轴号
Memory	INT	缓存区大小标志： 当 Memory= 0 时，每个缓存区有 32 段空间； 当 Memory= 1 时，每个缓存区有 1024 段空间
Fifo	INT	指定存放运动数据的缓存区，共有 2 个 FIFO 供选择（0 或者 1），默认为 0
arrPosPrfdata	ARRAY[1..1024] OF PosPrfData	各段的位置和时间参数 TYPE PosPrfData : STRUCT Pos: LREAL;(*各段的相对位置*)

		Ttime: DINT;(*各段的相对时间*) SegType:INT:= 0; (*各段的类型：普通段:0；匀速段:1；停止段:2*)
Length	INT	压入缓存区数据的段数
pSpace	POINTER TO INT	该指针指向目标缓存区的剩余空间
Loop	DINT	设置循环次数，默认为 0 ，即无限循环
Reset	BOOL	复位电平，当 Reset=1 时，Execute=0，并清空缓存区
pStatus	POINTER TO INT	记录运动过程的状态，初次调用时该指针指向的值必须为 0； 当 Execute=FALSE 时，pStatus^=0

参数声明示例：



速度曲线：



如图所示，该函数使用一系列“位置、时间”数据点描述速度规划，用户需要将速度曲线分割成若干段。

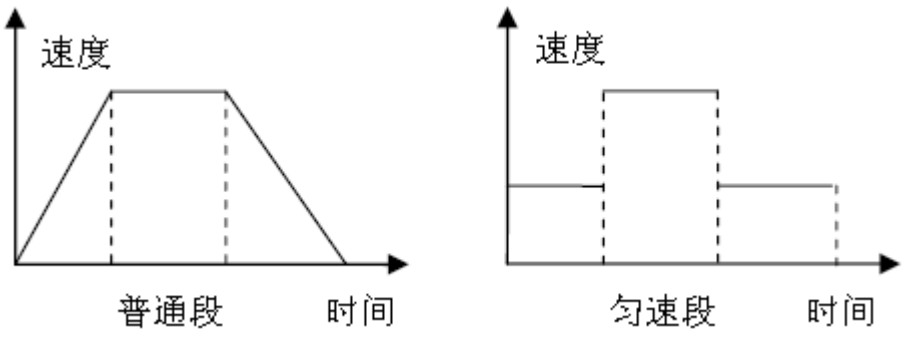
整个速度曲线被分割成 5 段，第 1 段起点速度为 0，经过时间 T1 运动位移 P1，因此第 1 段的终点速度为v1=2(P1/T1)；第 2 段起点速度为v1，经过时间 T2 运动位移 P2，因此第 2 段的终点速度为v2=2(P2/T2)- v1；第 3、4、5 段依此类推。

用户只需要给出每段所需时间和位移 ,运动控制器会计算段内各点的速度和位置，生成一条连续的速度曲线。为了得到光滑的速度曲线，可以增加速度曲线的分割段数。

数据段类型：

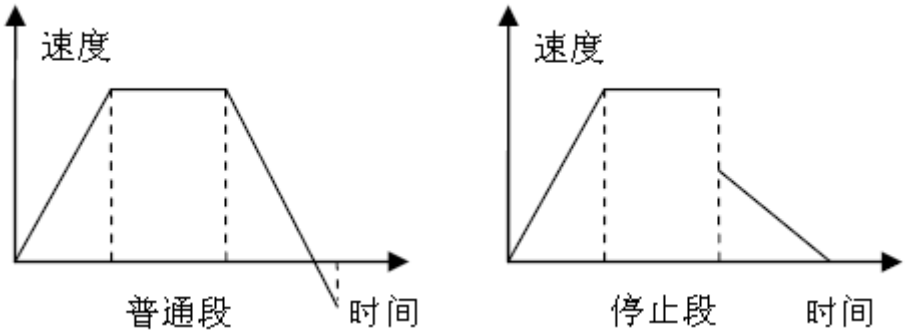
普通段：FIFO 中的第 1 段的起点速度为 0，从第 2 段开始每段的起点速度等于上一段的终点速度。

匀速段：FIFO 中各段的段内速度保持不变，段内速度=段内位移/段内时间。如图所示：



停止段：该段的终点速度为 0，起点速度根据段内位移和段内时间计算得到，

和上一段的终点速度无关。如图所示：

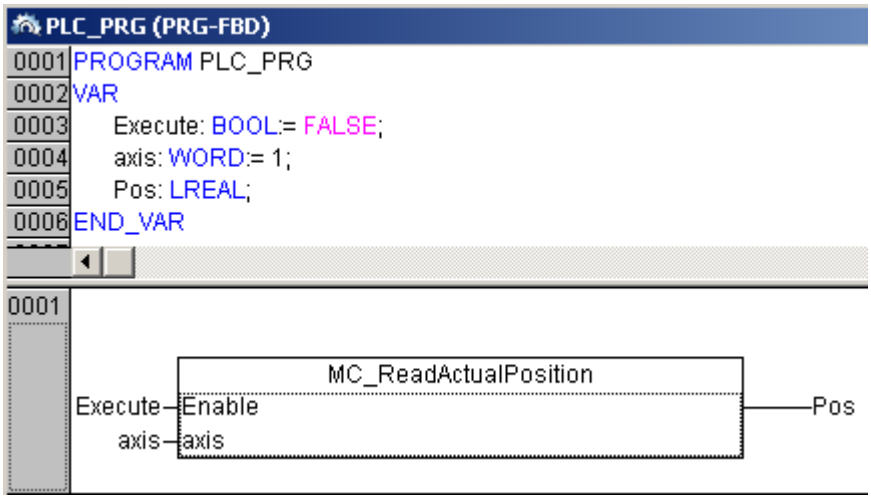


MC_ReadActualPosition

该函数（LREAL 型）用于读取目标轴的实际位置。返回值是位置值，单位“脉冲”，数据类型是 LREAL。

输入参数	类型	意义
Enable	BOOL	使能信号，高电平有效
Axis	WORD	轴号

参数声明示例：

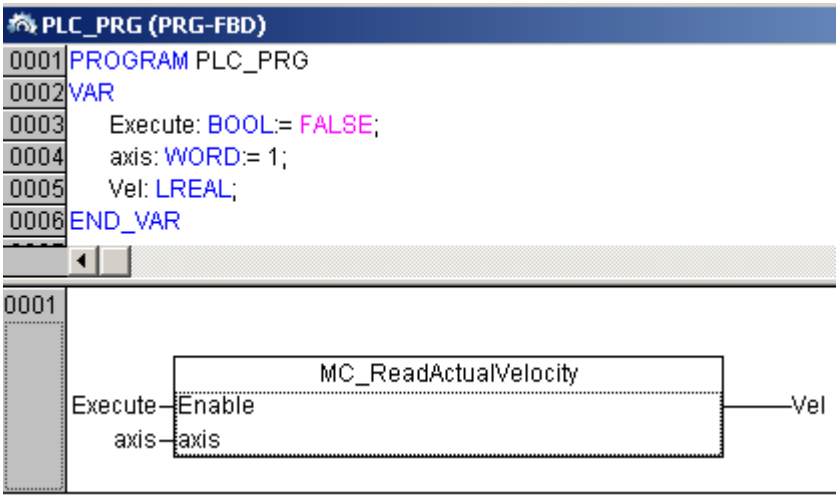


MC_ReadActualVelocity

该函数（LREAL 型）用于读取目标轴的实际速度。返回值是速度值，单位“脉冲/毫秒”，数据类型是 LREAL。

输入参数	类型	意义
Enable	BOOL	使能信号，高电平有效
axis	WORD	轴号

参数声明示例：



MC_ReadStatus

该函数用于读取目标轴的当前运动状态。返回值是结构体 Axis_Status。

输入参数	类型	意义
Enable	BOOL	使能信号，高电平有效
axis	WORD	轴号

STRUCT Axis_Status

Alarm: BOOL; //驱动器报警标志
控制轴连接的驱动器报警时置 1

MotionError: BOOL; //跟随误差越限标志
控制轴规划位置和实际位置的误差大于设定极限时置 1

HwLimitPos: BOOL; //正限位触发标志
正限位开关电平状态为限位触发电平时置 1
规划位置大于正向软限位时置 1

HwLimitNeg: BOOL; //负限位触发标志
负限位开关电平状态为限位触发电平时置 1
规划位置小于负向软限位时置 1

IO_Smth: BOOL; //IO 平滑停止触发标志
如果轴设置了平滑停止 IO，当其输入为触发电平时置 1，并自动平滑停止该轴

IO_Abrpt: BOOL; //IO 急停触发标志
如果轴设置了急停 IO，当其输入为触发电平时置 1，并自动急停该轴

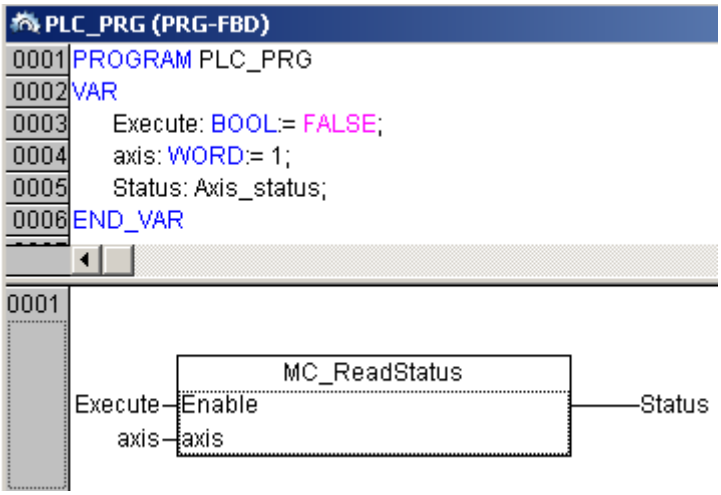
AxisOn: BOOL; //电机使能标志
电机使能时置 1

Motion: BOOL; //规划运动标志
规划器运动时置 1

Serve_Done: BOOL; //电机到位标志

规划器静止，规划位置和实际位置的误差小于设定误差带，并且在误差带内保持设定时间后，置起到位标志

参数声明示例：



MC_CamIn

该函数（INT 型）启动目标轴凸轮跟随运动。数据全部存入缓存区并且正常运动时，返回 1；否则返回-1。

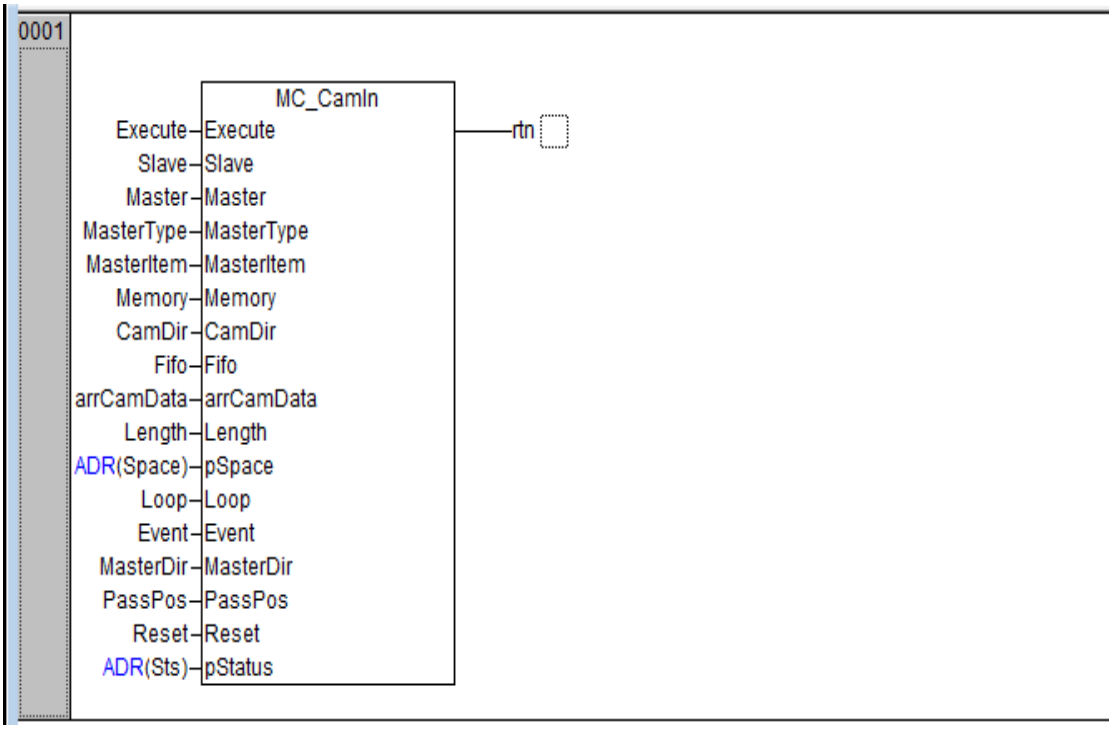
输入参数	类型	意义
Execute	BOOL	使能信号，上升沿触发
Slave	WORD	轴号
Master	WORD	主轴索引
MasterType	INT	跟随类型： 1 表示跟随编码器； 2 表示跟随规划器； 3 表示跟随合成轴。
MasterItem	INT	合成轴类型： 0 表示合成轴规划位置； 1 表示合成轴编码器位置。
Memory	INT	缓存区大小标志： 当 Memory= 0 时，每个缓存区有 16 段空间； 当 Memory= 1 时，每个缓存区有 512 段空间
CamDir	INT	设置跟随方式： 0 表示双向跟随； 1 表示正向跟随； 2 表示负向跟随。
Fifo	INT	指定存放运动数据的缓存区，共有

		2 个 FIFO 供选择（0 或者 1），默认为 0
arrCamData	ARRAY[1..512] OF CamData	各段主从轴数据。 TYPE CamData : STRUCT MasterOffset: DINT;(*主轴位置*) SlaveOffset: DINT;(*从轴位置*) CamType: INT;(*数据段类型：0 表示普通段；1 表示匀速段；2 表示减速到 0 段；3 表示保持 FIFO 之间速度连续段*)
Length	INT	压入缓存区数据的段数
pSpace	POINTER TO INT	该指针指向目标缓存区的剩余空间
Loop	INT	设置循环执行次数，默认为 0 ，即无限循环
Event	INT	设置启动跟随条件： 1 表示立即启动跟随； 2 表示主轴穿越设定位置后启动跟随。
MasterDir	INT	主轴运动方向： 1 表示正向运动； -1 表示负向运动。
PassPos	DINT	穿越位置，当 Event 为 2 时有效。
Reset	BOOL	复位电平，当 Reset=1 时，Execute=0，并清空缓存区
pStatus	POINTER TO INT	记录运动过程的状态，初次调用时该指针指向的值必须为 0； 当 Execute=FALSE 时，pStatus^=0

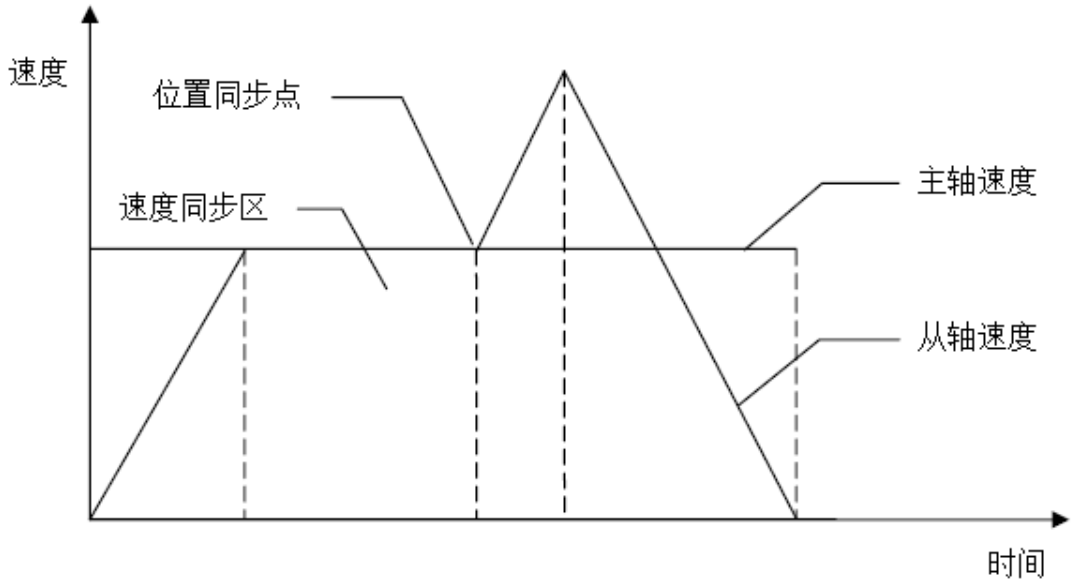
参数声明示例：

PLC_PRG (PRG-FBD)

```
0001 PROGRAM PLC_PRG
0002 VAR
0003     Execute: BOOL;
0004     Slave: WORD := 2;
0005     Master: WORD := 1;
0006     MasterType: INT := 1;
0007     MasterItem: INT;
0008     Memory: INT := 0;
0009     CamDir: INT := 0;
0010     Fifo: INT := 0;
0011     arrCamData: ARRAY[1..512] OF CamData:=(MasterOffset:= 40000, SlaveOffset:= 10000, CamType:= 0),
0012                                           (MasterOffset:= 80000, SlaveOffset:= 40000, CamType:= 0),
0013                                           (MasterOffset:= 40000, SlaveOffset:= 10000, CamType:= 0);
0014     Length: INT := 3;
0015     Space: INT;
0016     Loop: INT := 0;
0017     Event: INT := 1;
0018     MasterDir: INT := 1;
0019     PassPos: DINT;
0020     Reset: BOOL;
0021     Sts: INT;
0022     rtn: INT;
0023 END_VAR
```



速度曲线：



如图所示，在一些应用中，2 轴或多轴之间需要保证位置同步和速度同步。位置同步点表示主轴和从轴必须同时到达各自指定位置 ;速度同步区表示主轴和从轴之间必须保持准确的速度比。

第 1 段是加速区，从轴逐渐加速，直至到达同步速度。第 2 段是速度同步区，从轴和主轴保持设定的速度比，速度同步区结束时，主轴和从轴同时到达位置同步点。第 3 段是加速区，从轴穿越位置同步点以后迅速加速，脱离速度同步区。第 4 段是减速区，从轴逐渐减速到 0。

数据段类型：

普通段，FIFO 中第 1 段的起点速度为 0，从第 2 段起每段的起点速度等于上一段的终点速度。

匀速段，FIFO 中各段的段内速度保持不变。

停止段 ,该段的终点速度为 0 ,起点速度根据段内位移和段内时间计算得到，和上一段的终点速度无关。

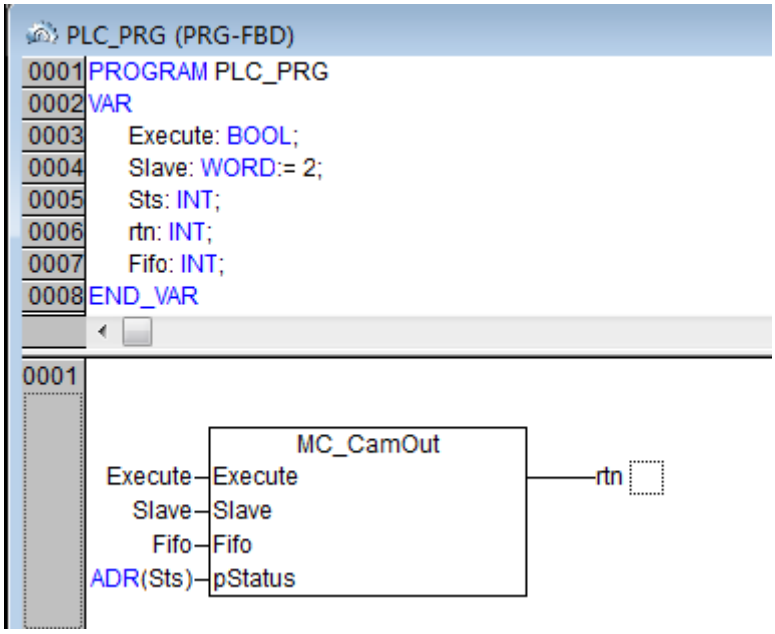
连续段，FIFO 中第一段的起点速度等于上个 FIFO 的终点速度，从第 2 段起每段的起点速度等于上一段的终点速度。

MC_CamOut

该函数（INT 型）使目标轴退出凸轮跟随并停止运动，目标轴必须已经启动凸轮跟随运动。

输入参数	类型	意义
Execute	BOOL	使能信号，上升沿触发
Slave	WORD	从轴轴号
Fifo	INT	之前指定存放数据的缓存区
pStatus	POINTER TO INT	记录运动过程的状态，初次调用时该指针指向的值必须为 0； 当 Execute=FALSE 时，pStatus^=0

参数声明示例：



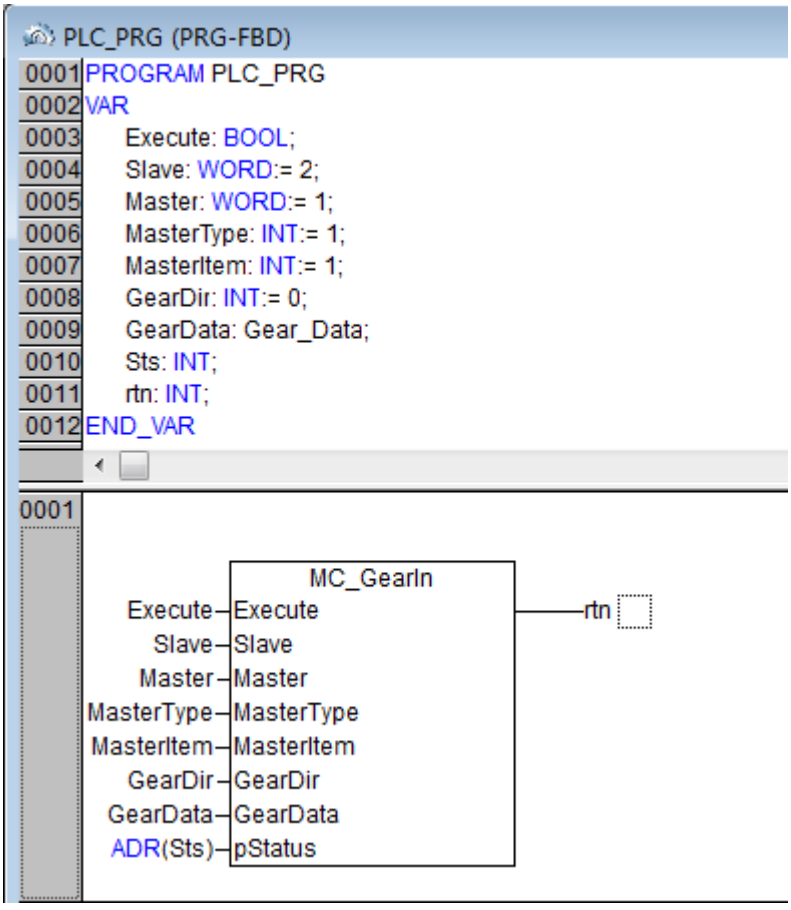
MC_GearIn

该函数（INT 型）启动目标轴齿轮跟随运动。

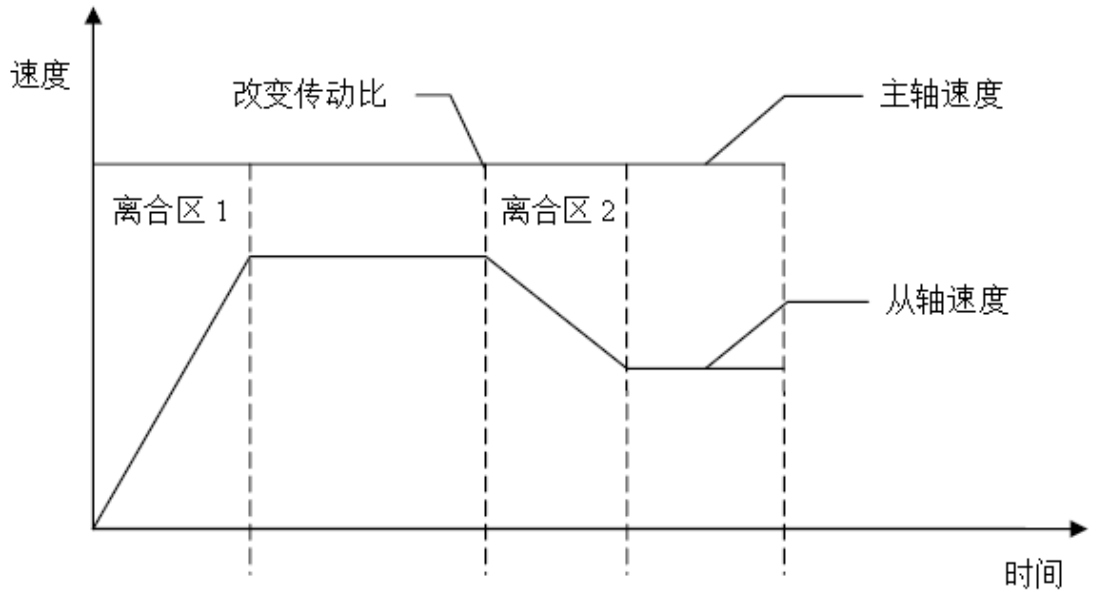
输入参数	类型	意义
Execute	BOOL	使能信号，上升沿触发
Slave	WORD	轴号
Master	WORD	主轴索引
MasterType	INT	跟随类型： 1 表示跟随编码器； 2 表示跟随规划器； 3 表示跟随合成轴。
MasterItem	INT	合成轴类型： 0 表示合成轴规划位置； 1 表示合成轴编码器位置。
GearDir	INT	设置跟随方式： 0 表示双向跟随； 1 表示正向跟随；

		2 表示负向跟随。
GearData	Gear_Data	主从轴传动比及离合区位移 TYPE Gear_Data : STRUCT MasterEven:DINT;(*传动比，主轴位移*) SlaveEven: DINT;(*传动比，从轴位移*) MasterSlope: DINT;(*主轴离合区位移*)
pStatus	POINTER TO INT	记录运动过程的状态，初次调用时该指针指向的值必须为 0； 当 Execute=FALSE 时，pStatus^=0

参数声明示例：



速度曲线：



电子齿轮模式能够将 2 轴或多轴联系起来，实现精确的同步运动，从而替代传统的机械齿轮连接。电子齿轮模式能够灵活的设置传动比，节省机械系统的安装时间。

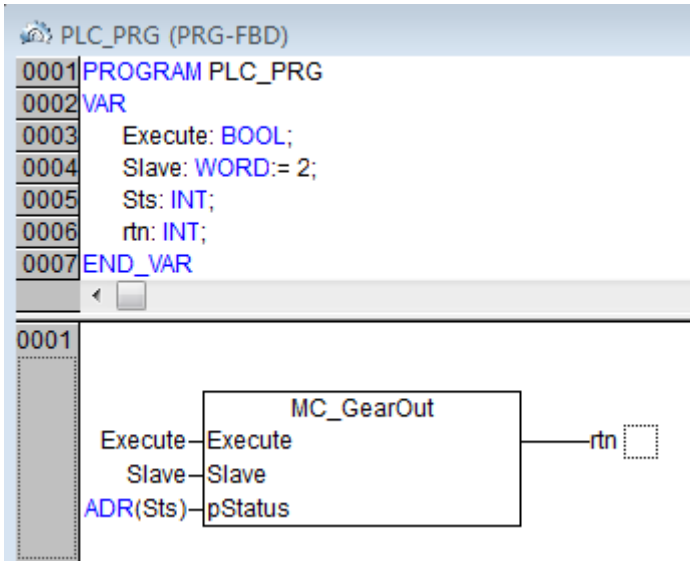
如图所示，电子齿轮模式能够在线修改传动比，当改变传动比时，可以设置离合区间，实现平滑变速。主轴匀速运动，从轴为电子齿轮模式，在离合区 1 从轴的传动比从 0 逐渐增大到设定传动比。当改变传动比时，在离合区 2 从轴的传动比逐渐变化到新的传动比。离合区越大，从轴传动比的变化过程越平稳。

MC_GearOut

该函数（INT 型）使目标轴退出齿轮跟随并停止运动，目标轴必须已经启动齿轮跟随运动。

输入参数	类型	意义
Execute	BOOL	使能信号，上升沿触发
Slave	WORD	从轴轴号
pStatus	POINTER TO INT	记录运动过程的状态，初次调用时该指针指向的值必须为 0； 当 Execute=FALSE 时，pStatus^=0

参数声明示例：

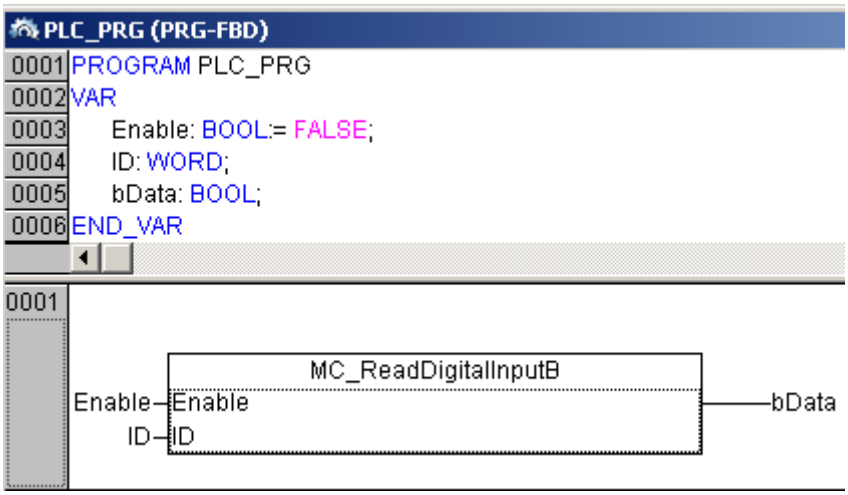


MC_ReadDigitalInputB

按位读取数字 IO 输入值，返回值是 BOOL 型。

输入参数	类型	意义
Enable	BOOL	使能信号，高电平有效
ID	WORD	通用输入 IO 的索引号，from bit0-bit15

参数声明示例：

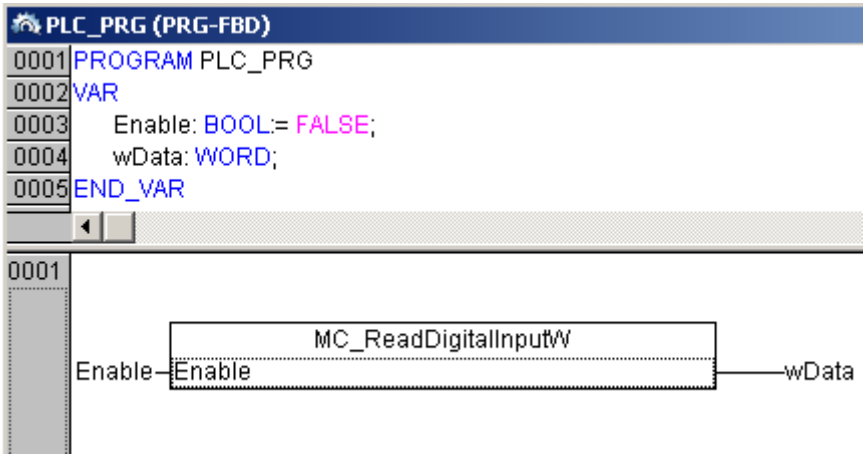


MC_ReadDigitalInputW

读取数字量 IO 输入值，返回值是 WORD 型。

输入参数	类型	意义
Enable	BOOL	使能信号，高电平有效

参数声明示例：

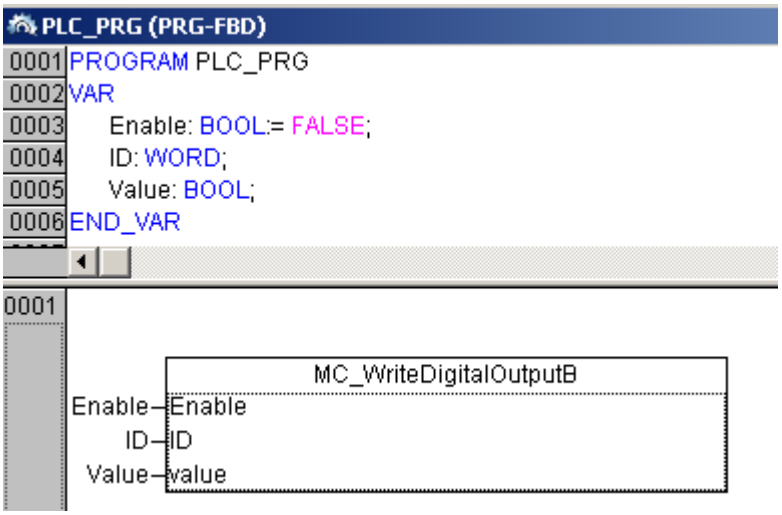


MC_WriteDigitalOutputB

按位设置数字 IO 的输出值。

输入参数	类型	意义
Enable	BOOL	使能信号，高电平有效
ID	WORD	通用输出的索引号，from bit0-bit15
Value	BOOL	设置目标输出值

参数声明示例：



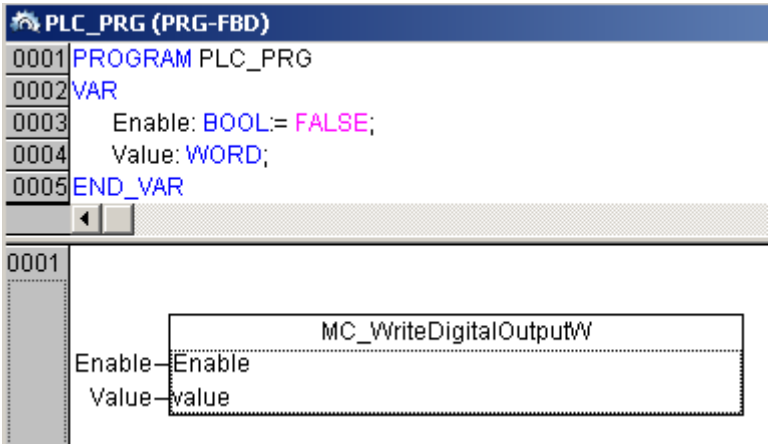
MC_WriteDigitalOutputW

设置数字 IO 的输出值。

输入参数	类型	意义

Enable	BOOL	使能信号，高电平有效
Value	DINT	设置目标输出值

参数声明示例：



CPAC_TCP 库的函数使用说明
CPAC_TCP 库的函数使用说明

CPAC-OtoStudio TCP 通讯库使用手册

1 前言

CPAC_TCP.lib 是基于CPAC-OtoStudio软件基础库SysLibSocket.lib开发的，SysLibSocket.lib是OtoStudio自带的标准系统库，用于实现TCP协议数据交互的基本功能。

该库采用非常简单的配置与参数，包含TCP server和TCP client的标准通讯函数

该文档描述了该库提供的模型以及其函数的功能和使用方法。

2 使用方法

TCP协议库的使用方法非常简单。所需的步骤在OtoStudio工程中的描述如下。

2.1 创建一个 OtoStudio 工程

- 打开OtoStudio软件
- 创建一个新的工程通过“文件/新建”
- 选择对应的Otobox控制器，如：CPAC-X00-TPX控制器
- 创建新的POU “PLC_PRG“ (选择编程语言，如：FBD).

2.2 添加库

- 打开资源->库文件管理器

- 右键添加库：CPAC_TCP.LIB

- 库STANDARD.LIB 以及 SYSLIBSOCKET.LIB, SYSLIBMEM.LIB 将会自动被添加. 如果没有被自动添加，请重复上一步操作，将这两个库也添加进来.

2.3 功能函数列表

函数名	功能
TcpServerOpenSocket	打开 TCP/IP 服务器
TcpClientOpenSocket	打开 TCP/IP 客户端
TcpServerWaitForConnect	诊断客户端是否连接服务器成功
TcpReceiveData	TCP/IP 客户端或者服务器接受数据
TcpSendData	TCP/IP 客户端或者服务器发送数据

2.4 功能函数的参数介绍

TcpServerOpenSocket

Opens a TCP server socket. Return: Socket-ID

输入参数	类型	意义
uiPort	UINT	Port number
diMaxConnections	DINT	Max possible client connections of the server

TcpClientOpenSocket

Opens a client socket to connect to a server. Return: Socket-ID for connection session

输入参数	类型	意义
iPort	INT	Port number of TCP socket to open
stIpAddress	STRING	IP-Address of server to connect to

TcpServerWaitForConnect

Wait for a client to connect.Return: Socket and IP-Address of connected client

输入参数	类型	意义
diSocket	DINT	Socket-ID
diTimeout	DINT	Timeout in ms

TcpReceiveData

Receive data via TCP socket.Return: Number of bytes sent or a value < 0 for an error (-1:

gracefully closed; -2: error)

输入参数	类型	意义
diSocket	DINT	Socket-ID
pbyData	DWORD	Address of data buffer
diDataSize	DINT	Size of data to send
diTimeout	DINT	Timeout in ms

TcpSendData

Send data via TCP socket.Return: Number of bytes sent or a value < 0 for an error (-1:

gracefully closed; -2: error)

输入参数	类型	意义
diSocket	DINT	Socket-ID
pbyData	DWORD	Address of data buffer
diDataSize	DINT	Size of data to send
diTimeout	DINT	Timeout in ms

3 示例程序

下面是用ST语言编写的使用CPAC_TCP开发通讯的示例程序。

3.1 TcpServer example

```
PROGRAM TCPServer
(*
    Example for a TCP server.
*)
VAR
    diSocket: DINT:= SOCKET_INVALID;
    uiPort: UINT:= 4444;
    Client: CLIENT_ACCEPT;
    abySend: ARRAY [1..10] OF BYTE;
    abyRecv: ARRAY [1..10] OF BYTE;
    abyRecvRaw: ARRAY [1..10] OF BYTE;
    bActive: BOOL;
    bInit: BOOL:= TRUE;
    bSend: BOOL;
    bEcho: BOOL:= TRUE;
    bSimulate: BOOL;
    bSimulationMode: BOOL:= TRUE; (*TRUE: Counter; FALSE: Random*)
    SimulateTimer: TON:= (PT:= t#250ms);
    byCounter: BYTE;
    bReset: BOOL;
    iIndex: INT;
    fbRandom: Random;
    diReceived: DINT;
    diSent: DINT;
    bSent: BOOL;
    bReceived: BOOL;
    bSendActivity: BOOL;
    bReceiveActivity: BOOL;
    LifeSignTimer: TON;
    tLifeSign: TIME:= t#500ms;
    LifeSignTimeoutTimer: TON;
    bLifeSignTimeout: BOOL;
    tLifeSignTimeout: TIME:= t#1s;
    byLifeSign: BYTE;
    bCheckConnection: BOOL:= TRUE;
END_VAR


IF bInit THEN
    (*Register Callbacks*)
    RegisterCallbacks();
```

```
    (*Initialising random number generator*)
    fbRandom(Seed:=TIME_TO_DINT(TIME()));
    bInit:= FALSE;
END_IF
IF bActive THEN
    (*Open listening socket*)
    IF diSocket = SOCKET_INVALID THEN
        diSocket := TcpServerOpenSocket(uiPort, 1);
    ELSE
        (*Wait for a client connection*)
        IF Client.diSocket = SOCKET_INVALID THEN
            Client := TcpServerWaitForConnect(diSocket, 250);
        ELSE

            (*Lifesign*)
            IF bCheckConnection THEN
                LifeSignTimer(IN:= TRUE, PT:= tLifeSign);
                LifeSignTimeoutTimer(IN:= TRUE, PT:= tLifeSignTimeout);
                IF LifeSignTimeoutTimer.Q THEN
                    bLifeSignTimeout:= TRUE;
                ELSE
                    bLifeSignTimeout:= FALSE;
                END_IF
            ELSE
                bLifeSignTimeout:= FALSE;
            END_IF

            (*Simulate data*)
            IF bSimulate THEN
                SimulateTimer(IN:= TRUE);
                IF SimulateTimer.Q THEN
                    IF bSimulationMode THEN (*Count bytes*)
                        byCounter:= byCounter + 1;
                        FOR iIndex:= 1 TO 10 DO
                            abySend[iIndex]:= byCounter;
                        END_FOR
                    ELSE (*Random bytes*)
                        FOR iIndex:= 1 TO 10 DO
                            fbRandom();
                            abySend[iIndex]:= DINT_TO_BYTE(fbRandom.Value);
                        END_FOR
                    END_IF
                    bSend:= TRUE;
                    SimulateTimer(IN:= FALSE);
```

```
            END_IF
        END_IF

        (*Transmit*)
        IF bSend THEN
            diSent:= TcpSendData(Client.diSocket, ADR(abySend), SIZEOF(abySend),
10);
            IF diSent <= 0 THEN (*Problem occured*)
                CloseSocket(ADR(Client.diSocket));
                Client.stIPAddress:= '';
                LifeSignTimeoutTimer(IN:= FALSE);
            ELSE
                bSent:= TRUE;
                LifeSignTimer(IN:= FALSE);
            END_IF
            bSend:= FALSE;
        ELSIF LifeSignTimer.Q THEN
            diSent:= TcpSendData(Client.diSocket, ADR(byLifeSign),
SIZEOF(byLifeSign), 10);
            IF diSent <= 0 THEN (*Problem occured*)
                CloseSocket(ADR(Client.diSocket));
                Client.stIPAddress:= '';
                LifeSignTimeoutTimer(IN:= FALSE);
            ELSE
                bSent:= TRUE;
                LifeSignTimer(IN:= FALSE);
            END_IF
        END_IF

        (*Receive*)
        REPEAT
            diReceived:= TcpReceiveData(Client.diSocket, ADR(abyRecvRaw),
SIZEOF(abyRecvRaw), 10);
            IF diReceived < 0 THEN (*Problem occured*)
                CloseSocket(ADR(Client.diSocket));
                Client.stIPAddress:= '';
                LifeSignTimeoutTimer(IN:= FALSE);
            ELSIF diReceived > 0 THEN (*Something received*)
                LifeSignTimeoutTimer(IN:= FALSE);
                bReceived:= TRUE;
                IF NOT (diReceived = 1 AND abyRecvRaw[1] = 0) THEN
                    abyRecv:= abyRecvRaw;
                    IF bEcho THEN (*Echo received data*)
                        abySend:= abyRecv;
```

```
                                diSent:=    TcpSendData(Client.diSocket,    ADR(abySend),
SIZEOF(abySend), 10);
                                IF diSent <= 0 THEN (*Problem occured*)
                                    CloseSocket(ADR(Client.diSocket));
                                    Client.stIPAddress:= '';
                                ELSE
                                    bSent:= TRUE;
                                    LifeSignTimer(IN:= FALSE);
                                END_IF
                            END_IF
                        END_IF
                    END_IF
                UNTIL
                    diReceived <= 0
                END_REPEAT;

                IF bSent THEN
                    bSendActivity:= NOT bSendActivity;
                    bSent:= FALSE;
                END_IF
                IF bReceived THEN
                    bReceiveActivity:= NOT bReceiveActivity;
                    bReceived:= FALSE;
                END_IF

            END_IF
        END_IF
    ELSE
        (*Shutdown server*)
        IF diSocket <> SOCKET_INVALID THEN
            CloseSocket(ADR(diSocket));
            IF Client.diSocket <> SOCKET_INVALID THEN
                CloseSocket(ADR(Client.diSocket));
                Client.stIPAddress:= '';
            END_IF
        END_IF
        IF LifeSignTimer.IN THEN
            LifeSignTimer(IN:= FALSE);
        END_IF
        IF LifeSignTimeoutTimer.IN = TRUE THEN
            LifeSignTimeoutTimer(IN:= FALSE);
        END_IF
        IF bLifeSignTimeout THEN
            bLifeSignTimeout:= FALSE;
```

```
        END_IF
    END_IF
    IF bReset THEN
        (*Reset data*)
        SysMemSet(dwDest:= ADR(abySend), bCharacter:= 0, dwCount:= SIZEOF(abySend));
        SysMemSet(dwDest:= ADR(abyRecv), bCharacter:= 0, dwCount:= SIZEOF(abyRecv));
        bReset:= FALSE;
    END_IF
```

3.2 TcpClient example

```
PROGRAM TCPClient
(*
    Example for a TCP client.
*)
VAR
    diSocket: DINT:= SOCKET_INVALID;
    uiPort: UINT:= 4444;
    stDestIPAddress: STRING:= '127.0.0.1';
    abySend: ARRAY [1..10] OF BYTE;
    abyRecv: ARRAY [1..10] OF BYTE;
    abyRecvRaw: ARRAY [1..10] OF BYTE;
    bActive: BOOL;
    bInit: BOOL:= TRUE;
    bEcho: BOOL;
    bSimulate: BOOL:= TRUE;
    bSimulationMode: BOOL:= TRUE; (*TRUE: Counter; FALSE: Random*)
    SimulateTimer: TON:= (PT:= t#250ms);
    byCounter: BYTE;
    bReset: BOOL;
    iIndex: INT;
    fbRandom: Random;
    bSend: BOOL;
    diReceived: DINT;
    diSent: DINT;
    bSent: BOOL;
    bReceived: BOOL;
    bSendActivity: BOOL;
    bReceiveActivity: BOOL;
    LifeSignTimer: TON;
    tLifeSign: TIME:= t#500ms;
    LifeSignTimeoutTimer: TON;
    bLifeSignTimeout: BOOL;
```

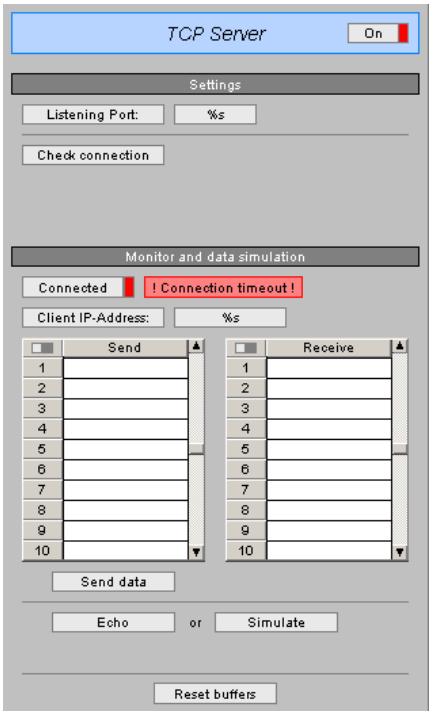
```
    tLifeSignTimeout: TIME:= t#1s;
    byLifeSign: BYTE;
    bCheckConnection: BOOL:= TRUE;
END_VAR

IF bInit THEN
    (*Register Callbacks*)
    RegisterCallbacks();
    (*Initialising random number generator*)
    fbRandom(Seed:=TIME_TO_DINT(TIME()));
    bInit:= FALSE;
END_IF
IF bActive THEN
    IF diSocket = SOCKET_INVALID THEN
        diSocket := TcpClientOpenSocket(uiPort, stDestIPAddress);
    ELSE

        (*Lifesign*)
        IF bCheckConnection THEN
            LifeSignTimer(IN:= TRUE, PT:= tLifeSign);
            LifeSignTimeoutTimer(IN:= TRUE, PT:= tLifeSignTimeout);
            IF LifeSignTimeoutTimer.Q THEN
                bLifeSignTimeout:= TRUE;
            ELSE
                bLifeSignTimeout:= FALSE;
            END_IF
        ELSE
            bLifeSignTimeout:= FALSE;
        END_IF

        (*Simulate data*)
        IF bSimulate THEN
            SimulateTimer(IN:= TRUE);
            IF SimulateTimer.Q THEN
                IF bSimulationMode THEN (*Count bytes*)
                    byCounter:= byCounter + 1;
                    FOR iIndex:= 1 TO 10 DO
                        abySend[iIndex]:= byCounter;
                    END_FOR
                ELSE (*Random bytes*)
                    FOR iIndex:= 1 TO 10 DO
                        fbRandom();
                        abySend[iIndex]:= DINT_TO_BYTE(fbRandom.Value);
                    END_FOR
```

```
                END_IF
                bSend:= TRUE;
                SimulateTimer(IN:= FALSE);
            END_IF
        END_IF

        (*Transmit*)
        IF bSend THEN
            diSent:= TcpSendData(diSocket, ADR(abySend), SIZEOF(abySend), 10);
            IF diSent <= 0 THEN (*Problem occured*)
                CloseSocket(ADR(diSocket));
                LifeSignTimeoutTimer(IN:= FALSE);
            ELSE
                bSent:= TRUE;
                LifeSignTimer(IN:= FALSE);
            END_IF
            bSend:= FALSE;
        ELSIF LifeSignTimer.Q THEN
            diSent:= TcpSendData(diSocket, ADR(byLifeSign), SIZEOF(byLifeSign), 10);
            IF diSent <= 0 THEN (*Problem occured*)
                CloseSocket(ADR(diSocket));
                LifeSignTimeoutTimer(IN:= FALSE);
            ELSE
                bSent:= TRUE;
                LifeSignTimer(IN:= FALSE);
            END_IF
        END_IF

        (*Receive*)
        REPEAT
            diReceived:= TcpReceiveData(diSocket, ADR(abyRecvRaw),
SIZEOF(abyRecvRaw), 10);
            IF diReceived < 0 THEN (*Problem occured*)
                CloseSocket(ADR(diSocket));
                LifeSignTimeoutTimer(IN:= FALSE);
            ELSIF diReceived > 0 THEN (*Something received*)
                LifeSignTimeoutTimer(IN:= FALSE);
                bReceived:= TRUE;
                IF NOT (diReceived = 1 AND abyRecvRaw[1] = 0) THEN
                    abyRecv:= abyRecvRaw;
                    IF bEcho THEN (*Echo received data*)
                        abySend:= abyRecv;
                        diSent:= TcpSendData(diSocket, ADR(abySend), SIZEOF(abySend),
10);
```

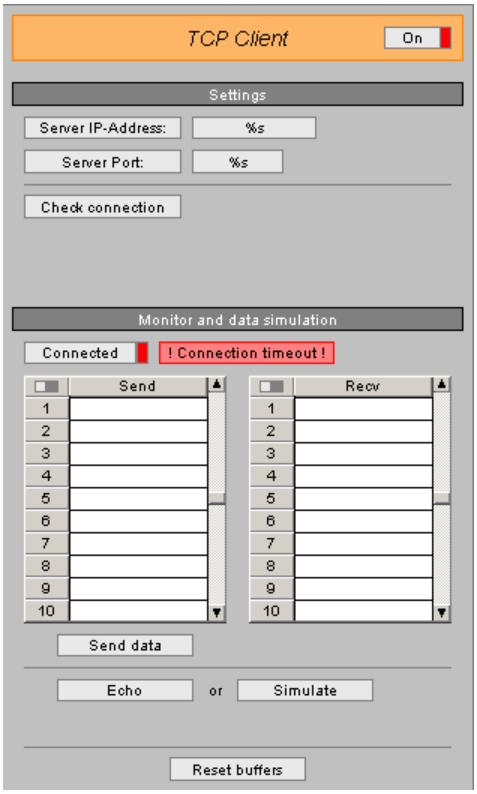
```
                        IF diSent <= 0 THEN (*Problem occured*)
                            CloseSocket(ADR(diSocket));
                        ELSE
                            bSent:= TRUE;
                            LifeSignTimer(IN:= FALSE);
                        END_IF
                    END_IF
                END_IF
            END_IF
        UNTIL
            diReceived <= 0
        END_REPEAT;

        IF bSent THEN
            bSendActivity:= NOT bSendActivity;
            bSent:= FALSE;
        END_IF
        IF bReceived THEN
            bReceiveActivity:= NOT bReceiveActivity;
            bReceived:= FALSE;
        END_IF

    END_IF
ELSE
    (*Shutdown client*)
    IF diSocket <> SOCKET_INVALID THEN
        CloseSocket(ADR(diSocket));
    END_IF
    IF LifeSignTimer.IN THEN
        LifeSignTimer(IN:= FALSE);
    END_IF
    IF LifeSignTimeoutTimer.IN THEN
        LifeSignTimeoutTimer(IN:= FALSE);
    END_IF
    IF bLifeSignTimeout THEN
        bLifeSignTimeout:= FALSE;
    END_IF
END_IF
IF bReset THEN
    (*Reset data*)
    SysMemSet(dwDest:= ADR(abySend), bCharacter:= 0, dwCount:= SIZEOF(abySend));
    SysMemSet(dwDest:= ADR(abyRecv), bCharacter:= 0, dwCount:= SIZEOF(abyRecv));
    bReset:= FALSE;
END_IF
```

4 人机界面

4.1 TcpServer example



4.2 TcpClient example



CPAC-OtoStudio 掉电保存数据功能使用手册

1 前言

CPAC-OtoBox针对需要应用需要实现在控制器断电前记录一些数据，并在控制器再次启动时将这些数据保持上次断前的值的需求。提供了掉电保存功能。

该文档描述了掉电保存功能的模型以及其函数的功能和使用方法及例程。

2 使用方法

掉电保存功能的使用方法非常简单。所需的步骤在OtoStudio工程中的描述如下。

2.1 创建一个 OtoStudio 工程

- 打开OtoStudio软件
- 创建一个新的工程通过“文件/新建”
- 选择对应的Otobox控制器，如：CPAC-X00-TPX控制器
- 创建新的POU “PLC_PRG“ (选择编程语言，如：FBD).

2.2 添加库

- 打开资源->库文件管理器
- 右键添加库：SysLibPlcctrl.LIB

2.3 掉电保存功能函数列表

函数名	功能
SysSaveRetains	记录所有定义为 RETAIN 类型的变量到控制器.该函数的参数为一个空字符串

2.4 功能函数的参数介绍

2.4.1 SysSaveRetains

保存所有掉电保存类型变量成功，返回0

输入参数	类型	意义
stFileName	**STRING**	**‘’ 强制为空字符串**

3 示例程序

下面是用ST语言编写的使用CPAC掉电保存功能的示例程序。

该例子的效果是：当我们给掉电保存变量a,b赋值后，将save：=TRUE时，程序会调用保存数据的函数一次。用户可以周期调用该函数 ,亦可以再判断每次需要保存的数据发生变化时调用一次，这个由用户决定。但是：该函数由于需要读写硬盘，所以会花费时间（大约在200微妙左右）。

```
PROGRAM PLC_PRG
VAR RETAIN
	a: INT;
	b: STRING;
END_VAR
VAR
	save: BOOL;
END_VAR
----------------------------------------------
a;
b;
IF save THEN
   SysSaveRetains('');
   save:=FALSE;
END_IF
```

重点说明：将该例程下载到控制器中，然后必须要“创建引导工程”，掉电保存功能才能生效。这是因为如果不创建引导工程，那么程序就会再掉电后丢失，掉电保存也没有意义。

请参考安装程序中的例子：\Projects\other\Retain.pro

CPAC-OtoStudio 控制器设置库使用手册

1 前言

ControllerSetting.lib是OtoStudio上针对OEM用户修改CPAC-OtoBOX的IP地址的专用库。

该文档描述了该库提供的功能以及其函数的功能和使用方法。

2 使用方法

ControllerSetting库的使用方法非常简单。所需的步骤在OtoStudio工程中的描述如下。

2.1 创建一个 OtoStudio 工程

- 打开OtoStudio软件
- 创建一个新的工程通过“文件/新建”
- 选择对应的Otobox控制器，如：CPAC-X00-TPX控制器
- 创建新的POU “PLC_PRG“ (选择编程语言，如：FBD).

2.2 添加库

- 打开资源->库文件管理器
- 右键添加库：IP_Extension.LIB

2.3 功能函数列表

函数名	功能
CE_SetIPInfo	设置 Internet 协议（TCP/IP）信息
CE_SetResolusion	设置分辨率
CE_Reboot	重启 CPAC-OtoBox

2.4 功能函数的参数介绍

CE_SetIPInfo

设置控制器IP信息，如果设置成功，返回0。设置成功后，系统会自动重启。

输入参数	类型	意义
IPAddress	**POINTER TO BYTE**	**IP 地址** **例如：** **STRING_IP : STRING;** **ADR(STRING_IP)**
SubnetMask	**POINTER TO BYTE**	**子网掩码地址** **例如：** **STRING_SubNet : STRING;** **ADR(STRING_SubNet)**
DefaultGateway	**POINTER TO BYTE**	**网关地址** **例如：** **STRING_ Gateway : STRING;** **ADR(STRING_Gateway)**

下面是用ST语言编写的使用CE_SetIPInfo开发通讯的示例程序。

```
PROGRAM PLC_PRG
VAR
    rtn:INT;
    IPAdress:STRING:='192.168.0.3';
    SubNetMask:STRING:='255.255.255.0';
    Gateway:STRING:='192.168.0.1';
END_VAR
-----------------------------------------------
rtn:= CE_SetIPInfo(ADR(IPAdress), ADR(SubNetMask), ADR(Gateway));
```

CE_SetResolusion

设置控制器显示分辨率信息，如果设置成功，返回0。设置成功后，系统会自动重启。注意：首先要设置BIOS的分辨率（开机Delete），必须保证控制器的分辨率必须是BIOS支持的。

输入参数	类型	意义
Display_Width	**INT**	**分辨率宽度像素** **例如：** **800**
Display_Hight	**INT**	**分辨率高度像素** **例如：** **600**

下面是用ST语言编写的使用CE_SetResolusion开发通讯的示例程序。

```
PROGRAM PLC_PRG
VAR
    rtn:INT;
    Width:INT:=800;
    Hight:INT:=600;
END_VAR
----------------------------------------------
rtn:= CE_SetResolusion(Width, Hight);
```

CE_Reboot

冷启动控制器

输入参数	类型	意义

下面是用ST语言编写的使用CE_Reboot开发通讯的示例程序。

```
PROGRAM PLC_PRG
VAR
    rtn:INT;
END_VAR
----------------------------------------------
rtn:= CE_Reboot();
```

CPAC-OtoStudio 版本更新手册

1 前言

CPAC-OtoStudio升级包针对是OtoStudio的阶段性升级，但又无需卸载整个软件再重装，快捷安全。

该文档描述了如何升级的使用方法。

升级包的类别有两种：简易升级包，完整升级包

2 简易升级包升级

使用简易升级包升级的条件是：升级包中为找到 TNF 文件时。

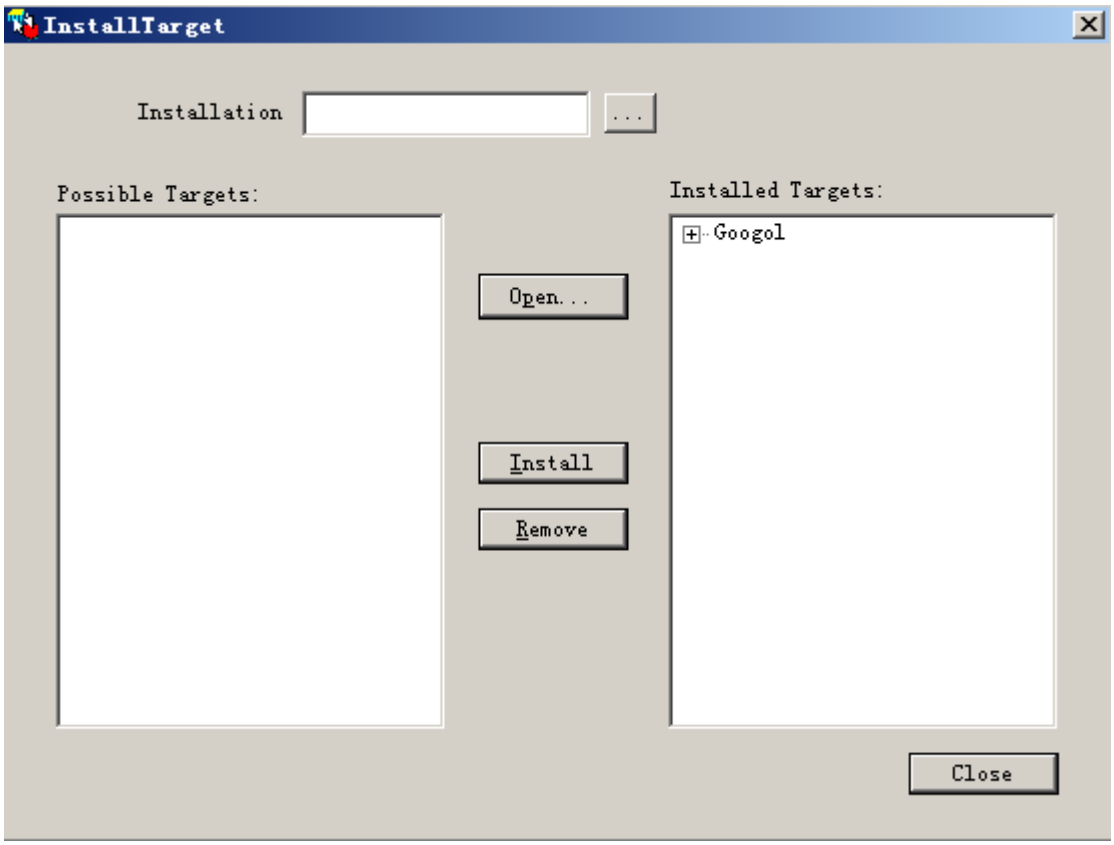
升级方法非常简单，直接将升级包中得文件全部拷贝到：“安装盘:\Program files\Googol\CPACTarget 下。然后重启 OtoStudio 软件。

3 完整安装包升级

所需的步骤在OtoStudio工程中的描述如下。

3.1 启动目标系统安装程序

- 打开OtoStudio软件安装路径，找到“安装盘:\Program files\Googol\CoDeSys V2.3\InstallTarget.exe”

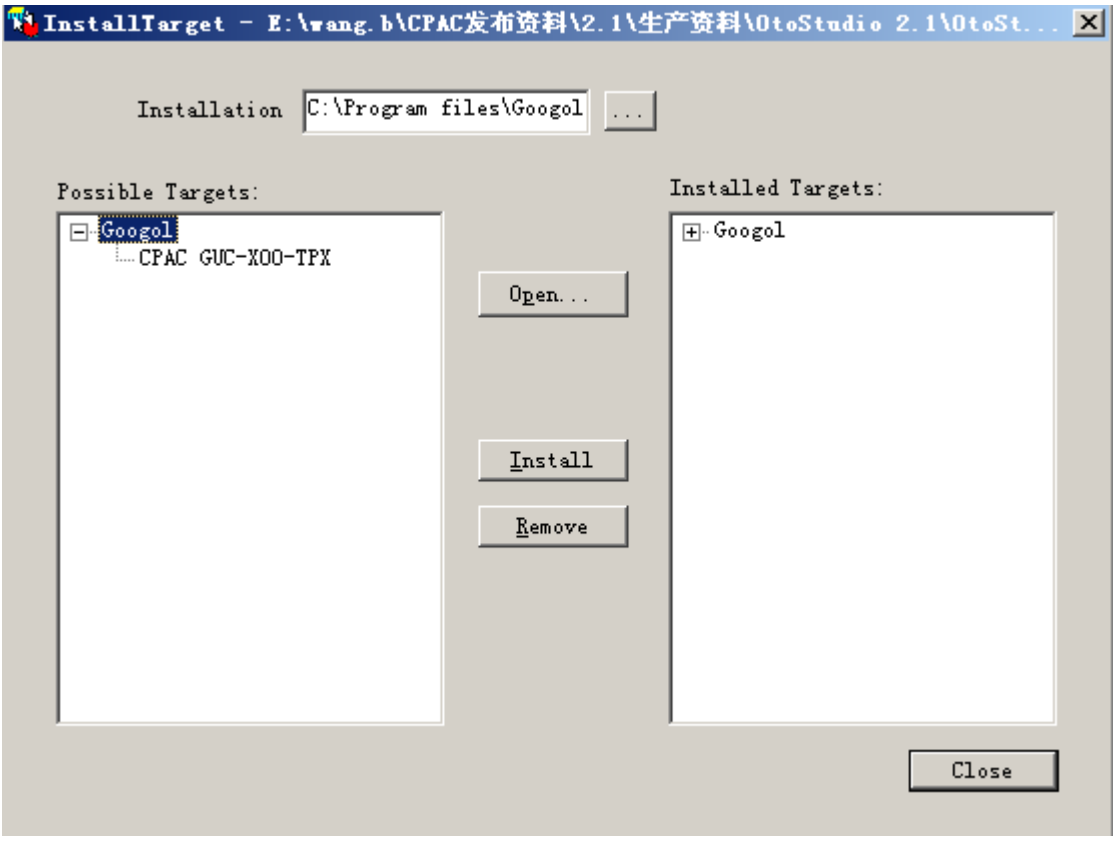


3.2 添加升级包

- 首先建议备份原有的配置文件：

“安装盘:\Program files\Googol\CPACTarget\GTS800.cfg, PlcConfGoogol 文件夹”

- 在InStallTarget软件中，点击 “Open”，打开安装包文件夹中的Googol.tnf文件，如下图所示：



- 选中坐标的Googol标题，然后点击“Install”. 系统会提示你是否覆盖原有的程序。点击”

Yes”如下图所示：

然后，点击”Close”完成更新了。

- 接着需要步骤一中还原原来备份的文件。然后重启OtoStudio软件。

CPAC-OtoStudio 编程命名规则手册

1 前言

CPAC-OtoStudio软件编程命名规则是OtoStudio自带的针对软件开发人员的编程规范，以方便编程人员内部之间的交互，代码共享。

该文档描述了该规则的主要内容。

2 主要内容

2.1 操作数命名

- 标准变量命名
- 用户自定义数据类型命名”
- 函数，功能块，程序命名
- 可视化界面命名

2.2 标准变量命名

应用程序和库中使用的变量的命名尽可能遵循**匈牙利标志法" (Hungarian Notation)**
针对每一个变量基本名称，应该能找到该变量的意思、简单描述。每一个单词的第一个字母应该大写，其他要小写（例如：FileSize）. 在基本名称前要加变量类型的前缀，前缀必须是小写。前缀的对应如下：

Data type	lower limit	upper limit	Information content	Prefix	Comment
BOOL	FALSE	TRUE	1 Bit	x *	
				b	reserved
BYTE			8 Bit	by	Bit string, not for arithm. operations
WORD			16 Bit	w	Bit string, not for arithm. operations
DWORD			32 Bit	dw	Bit string, not for arithm. operations
LWORD			64 Bit	lw	not for arithm. operations
SINT	-128	127	8 Bit	si	
USINT	0	255	8 Bit	usi	
INT	-32.768	32.767	16 Bit	i	
UINT	0	65.535	16 Bit	ui	
DINT	-2.147.483.648	2.147.483.647	32 Bit	di	
UDINT	0	4.294.967.295	32 Bit	udi	
LINT	-2^{63}	$2^{63} - 1$	64 Bit	li	
ULINT	0	$2^{64} - 1$	64 Bit	uli	
REAL			32 Bit	r	
LREAL			64 Bit	lr	
STRING				s	

TIME				tim	
TIME_OF_DAY				tod	
DATETIME				dt	
DATE				date	
ENUM			16 Bit	e	
POINTER				p	
ARRAY				a	

例如：

```
bySubIndex: BYTE;
sFileName: STRING;
udiCounter: UDINT;
pabyTelegramData: POINTER TO ARRAY [0..7] OF BYTE;
```

功能块实例和用户定义的类型的前缀如下例所示：

cansdoReceivedTelegram: CAN_SDOTelegram;

TYPE CAN_SDOTelegram :

(* 前缀: sdo *)

STRUCT

wIndex:WORD;

```
        bySubIndex:BYTE;
        byLen:BYTE;
        aby: ARRAY [0..3] OF BYTE;
    END_STRUCT
END_TYPE
```

局部常数类型变量以“c_”加变量类型作为前缀。例如，

```
VAR CONSTANT
c_uiSyncID: UINT := 16#80;
END_VAR
```

全_局变量”g_”和全局常变量”gc_”+变量类型作为前缀。例如，

```
VAR_GLOBAL
CAN_g_iTest: INT;
END_VAR
VAR_GLOBAL CONSTANT
CAN_gc_dwExample: DWORD;
END_VAR
```

其中 CAN 表示库的前缀（如果当前的数据是包含在一个库中）

2.3 用户自定义数据类型（DUT）变量命名

用户自定义类型都包括一个库名的前缀，如：CAN_。例如：

```
TYPE CAN_SDOTelegram :

(* prefix: CAN_*)

STRUCT
wIndex:WORD;
bySubIndex:BYTE;
byLen:BYTE;
abyData: ARRAY [0..3] OF BYTE;
END_STRUCT
END_TYPE
```

2.4 函数，功能块，程序命名

所有函数，功能块，程序包含库前缀（如：CAN_）,以及一个简要的描述（如：SendTelegram）。和变量一样，每个单词的首字母要大写，其他字母小写。强烈推荐 POU 的名称由一个动词加一个名词构成。示例如下,

FUNCTION_BLOCK CAN_SendTelegram (* prefix: canst *)
在 POU 的声明部分，需要给出注释，且所有输入，输出也要给出注释，且要安装变量定义的规则定义。针对功能块，创建实例的前缀要紧跟在名字后面。
动作（action）没有前缀；对于只在 POU 内部使用的动作，需要前缀“prv_”
函数必须至少有一个参数，外部库函数不能用结构体作为返回值。

针对功能块，功能，程序，在某些场合，为区分其类型，可使用前缀：FUN_, FB_, PRG_

2.5 可视化界面命名

界面部分也应与函数，功能块，程序类似命名。主界面：PLC_PRG 不能修改。
例如，
VISU_AAA, VISU_BBB
对于占位符控件，可以使用前缀：ph_
注意：尽量避免可视化界面与 POU（函数，功能块，程序）重名。

CPAC-OtoStudio 常见错误与警告手册

警告

如果在工程 编译过程中出现错误，将在 信息窗口出现提示。警告也同时显示出来。在此窗口按〈F4〉跳
到消息的下一行，与此相关的POU将会打开。在错误和警告之前有一个唯一的编号。在消息窗口选择了一个消
息行，按〈F1〉打开相应的在线帮助窗口。

1100

未知库中的功能名‘<name>’
使用一个外部库。请检查在 hex 文件中定义的所用功能是否也定义在 lib 文件内。

1101

″不能识别的符号‘<symbol>’″
代码生成器期望有一个名称为<symbol>的 POU。它没有在目标项目中定义。并使用这个名称定义了一个
功能/程序。
1102

″对符号‘<symbol>’的无效接口″
代码发生器期望有一个名为<symbol>的功能和恰好是一个标量的输入，或一个名为<symbol>和无输入或
输出的程序。

1103

″在代码地址‘<address>’的常数‘<name>’改写一个 16K 页的边界!″
超过 16K 页边界的一串常数。系统不能处理这种边界。这取决于运行时的系统，通过目标文件中的一个
登录项是否能解决这个问题。请与 PLC 制造商联系。

1200

″在任务调用的名为'<name>'的 POU 中的存取变量内容在参数列表中未获更新″
在任务配置中调用的任务块中所使用的变量将不会列在交叉参考列表中。

1300

″文件找不到'<name>'名″
指示全局变量的文件不存在。请检查其路径是否正确。

1301

″分析库未找到。分析代码无法生成″
系统使用了分析功能，但相应的库.lib 没找到。将它加入到库管理程序内。

1302

″新添加了外部引用功能，因此不能使用在线变更功能″
由于最后装载，你链接的一个库它所包含的功能尚未在运行时系统中引用。为此，你应装载整个项目。

1400

″未知的编译指令'<name>'，编译器忽略″
这个附注不受编译程序支持。受支持的命令见关键字的附注。

1401

"名为'<name>'的结构未定义任何元素"
此结构未定义任何元素，但是系统仍然会分配 1B 的内存容量。

1410

"功能中定义本地变量时'RETAIN'和'PERSISTENT'关键字无效"
系统将按照一般内部变量处理。

1411

"变量配置中定义的变量'<name>'未在任何任务调用中得到更新"
没有在任何调用中引用变量的高水平的例子。因此其不能在过程镜象中被拷贝。
例如:
变量配置:
VAR_CONFIG
plc_prg.aprg.ainst.in AT %IB0 : INT;
END_VAR
plc_prg:
index := INDEXOF(aprg);
程序 aprg 已经引用但是没有调用. 因此 plc_prg.aprg.ainst.in 永远得不到 %IB0 的实际值。

1412

"编译指令{pragma name}出现未知代号"
用户使用的编译指令包含有在该场合不能正确使用的指令。详情请参看 CPAC 在线帮助。

1413

"'<Name>'在名字列表中的索引非法，此索引将被忽略"
在编译指令中使用了不存在的参数列表。请参看参数管理器获取当前列表。

1414

名为'<name>'的编译指令定义太多组成部分
编译指令包含的定义（指括号里面的内容）超过了相应数组、功能块或者结构所能容纳的元素个数。

1500

"表达式未包含赋值。此语句无代码产生"
程序未使用该表达式的运算结果，因此该语句无代码产生。

1501

"字符串常量的内容被改写"
在 POU 中定义的字符串常量的内容不允许更改，因为系统忽略其大小检查。

1502

"POU 与其定义的某个变量重名。导致 POU 不能被正确调用"
POU 与其定义的某个变量重名。
例如:

```
PROGRAM a
...
VAR_GLOBAL
a: INT;
END_VAR
...


a; (* 没有 POU a 被调用，但是变量已经加载 *)
```

1503

"名为'<name>'的 POU 无输出，系统设定其输出为'TRUE'"
连接 FBD 或 KOP 的 POU 无输出。 系统自动将其设定为 TRUE。

1504

“’<name>’ (‘<number>’): 逻辑表达式的此分支将不会被执行，因为整个表达式的值已可确定”
逻辑表达式的此分支将不会被执行，因为整个表达式的值已可确定
例如:

```
IF a AND funct(TRUE) THEN ....
```

如果 a 是 FALSE 则 funct 不被调用。

1505

"'<name>'分支有歧义，可能不会被执行"
POU 的第一个输入为 FALSE，由于这个原因旁边的第二个输入将不被执行。

1506

"变量与某个本地动作同名，将导致该动作不能被正确调用"
重命名变量或行为。

1507

"创建的实例与某个功能名同名，将导致该实例不能被正确调用"
在 ST 编程语言中，如果调用与某功能同名的实例，系统将执行同名功能而不是实例。

1550

"名为'<name>'的 POU 在一个执行网络中被多重调用，可能导致不可预料的结果"
检查，对 POU 的多重调用是否是必要的。通过多重的调用可能导致不可预料的后果。

1600

"不明确打开 DB (可能产生错误的代码)."
没有表明 POU 打开的最初西门子程序。

1700

"输入框未联接上"
在 CFC 中有输入框未连接上，将不能产生执行代码。

1750

"步 '<名称>': 设定的时间下限超过时间上限"
请使用该步的'步属性'对话框进行时间的正确设定。

1751

"'<Name>'变量的使用警告：该变量使用于固定代码中并且影响步的执行顺序"
推荐对该变量重命名以使其具有唯一的标识符，从而避免负面影响。

1800

″非法的监控表达式'<Name>'″
可视化元素中包含了无法监控的表达式，检查变量名称和占位符的替换是否无误。

1801

″表达式不能作为输入变量使用″
可视化组件的输入框使用了组合表达式，请使用单个变量替换之。

1802

″名为'<Name>'的位图文件未找到″

1803

″网页可视化和目标平台可视化不支持打印操作″

1804

″目标平台不支持'<name>'型字体″

1805

″要使用可视化组件实现 PLC 趋势数据的存储应先进行相关的设定″

1806

″应对'目标设定'中的'PLC 报警处理'选项进行设定″

1807

″<name> (<number>): 针对可视目标没有警告消息窗口″

1850

″映射在%IB<number>位置的输入变量在任务'<name>'中使用，但在另外任务中更新″

1851

″映射在%IQ<number>位置的输出入变量在任务'<name>'中使用，但在另外任务中更新″

1852

″CanOpenMaster 功能块在任务'<name>'事件中未能实现循环调用！若要实现其循环调用，请使用'模块
参数'对话框正确设置运行参数并更新任务!″

1853

″PDO(index: '<number>')可能在任务'<name>'事件中未能实现周期性更新″

1900

″POU'<name>'（主程序）在库中找不到；当工程作为一个程序库使用时，主入口程序不可用″ 当项目作
为库使用时，不能利用“启动 POU”

1901

″入口变量和变量配置表未存入相应库文件中!″
存取变量和变量配置没有保存在库内。

1902

″'<name>'库不支持当前硬件平台类型!″
库的对象文件是为其他设备而生成的。

1903

″'<name>'库是非法库；该库文件不符合实际目标平台要求的格式″
文件没有实际目标所需要的格式。

1904

″ '<name>'常量覆盖了链接库的一个同名常量″
如果用户在工程中定义了与链接库同名的常量，库中该常量将被覆盖。

1970

"参数管理器：'<Name>'列，'<Name>'行的'<Name>'值不能被引入!"
请检查数据引入文件*.prm 的格式是否符合当前参数管理器的设置。

1980

"对全局变量'<Name>'同时读、写可能导致数据丢失"

1990

" '<name>'参数未能在参数设置中获的可用的内存地址映射"
请在资源选项卡中的 参数设置窗口对其进行正确设置。

2500

"任务 '<任务名称>': 对循环任务没有指定循环时间"

错误

3100

"用户程序代码太大。系统支持代码最大容量为'<number>'字节(<number>K)"
超过最大程序规模。减小项目规模。

3101

"用户程序使用数据太多，最大容量为'<number>'字节(<number>K)"
超出内存容量范围。请减少应用程序的数据使用量。

3110

"'<Name>'库出现错误"
该库文件*.hex 不是 INTEL 16 进制格式。

3111

″ ’<Name>’库太大。系统支持最大容量 64K″
.hex 文件超过设定的最大容量。

3112

″库中存在不可重复定位指令″
该库代码无法进行链接。

3113

″库代码覆盖功能表格.″
代码和功能表的范围重叠。

3114

″库使用了多个内存段.″
.hex 文件中的表和代码利用不止一个字段。

3115

″不能把常量赋值给 VAR_IN_OUT 型变量，数据类型不匹配.″
由于数据设置在“near”(近的),但是串常数设置在“huge”（大的）或“far”（远的），因而串常数的
内部指针格式不能转换成 VAR_IN_OUT 的内部指针格式。如有可能，改变这些目标设置。

3116

″功能表格覆盖库代码或段分界线。″

3117

″<Name> (<Zahl>):表达式过于复杂，无法用寄存器装载处理″

3120

″当前代码段超出 64K.″
当前生成的代码大于 64K。最终建立太多的初始化代码。

3121

″POU 太大.″
POU 的大小应限制在 64K 以下。

3122

″功能或结构的初始化代码太大，超出 64K 上限″
用于一个功能或一个结构化 POU 的初始化代码不应超过 64K。

3123

″数据段太大：段'<Number>%s', 大小<size> 字节(最大为 <number>字节)″

3124

″字符串常量太大: '<number>' 字节 (最大 253 字节)″
所给出的常量必须缩短字符数

3130

″用户堆栈太小：需要'<number>' 双字容量的堆栈，只能提供'<number>'双字容量的堆栈.″
POU 调用的嵌套深度太大。在目标设置中输入一个更大的堆栈尺寸，或编译没有任选项
“debug”(可在
对话框“project”(项目)“(options)”任选项“build”(建立)中设置)的建立项目。

3131

″用户堆栈太小：需要'<number>' 双字容量的堆栈，只能提供'<number>'双字容量的堆栈.″
请联系 PLC 制造商。

3132

″系统堆栈太小：需要'<number>' 双字容量的堆栈，只能提供'<number>'双字容量的堆栈.″
请联系 PLC 制造商。

3150

″功能 '<名称>'的参数 <编号>: 不能作为 C－函数的一个字符串参数来传递 IEC－功能的

结果.″
使用一个中间变量，将 IEC 功能的结果赋值给该中间变量。

3160

″不能打开'<name>'库文件.″
工程通过库管理器包含了'<name>'库，但是在指定的路径未能找到该库文件。

3161

″库文件'<name>'缺少代码段″
一个程序库的一个.obj 文件至少应包含一个 C 功能。将一个亚功能插入到.lib 文件内，在.lib 文件内
尚未定义这个亚功能。

3162

″无法解决库 '<名字>'涉及到的(符号 '<名称>', 类'<名称>', 类型'<名称>')″
.obj 文件包含一个对其他符号不能分辨的引用。请检查 C 编译程序的设置。

3163

″'<name>'库包含未知的引用类型″
.obj 文件包含一个引用类型，它不能被代码发生器所分辨。请检查 C 编译程序的设置。

3200

″'<name>'：布尔表达式过于复杂，请使用中间变量简化之″
目标系统的暂时存储器容量对表达式的规模来说显得不够。将表达式分成几个部分表达式，为此可使用
对几个中间变量进行赋值。

3201

″<名称> (<网络>): 一个网络必须不能导致代码超过 512 字节。″
不能分辨内部跳转。启动选项“利用 16 位跳转补偿”中的 68K 目标设置。

3202

″功能嵌套调用导致堆栈溢出″

使用一嵌套的功能调用 CONCAT(x,f(i))。这会导致数据丢失。将调用分成两个表达式。

3203

"表达式过于复杂（无法使用寄存器进行处理）."
将赋值分成几个表达式。

3204

"程序跳转超出范围 32k 字节"
跳转范围应不超过 32767 字节。

3205

"内部错误：包含太多常量字符串"
在一个 POU 中，最多能使用 3000 个常量字符串。

3206

"功能块程序代码溢出"
功能块程序代码上限为 32Kb。

3207

"数组优化失败"
计算索引时功能被调用导致最优化数组存取失败。

3208

"转化未成功"
使用的转换功能，不是为实际代码发生器而实现的。

3209

"操作未成功"
使用的操作符，不是为这种数据类型和实际代码发生器而实现的。MIN(字符串 1，字符串 2)

3210

"'<Name>'功能未找到"
调用的功能不能用于项目内。

3211

"字符串使用超出范围"
字符串变量一个表达式中可最多使用 10 次。

3212

"在 POU <POU 名称>错误的库命令 "

3250

"8 位控制器不支持 REAL 类型"
目标当前未得到支持。

3251

" 8 位控制器不支持日期类型。"
目标当前未得到支持。

3252

"堆栈溢出<number>字节"
当前目标系统不支持。

3253

"找不到此 hex 文件：'<Name>'"
目标当前未得到支持。

3254

"调用的外部库功能不能被解析."
目标当前未得到支持。

3255

"8 位控制器不支持指针."
运行于 8 位系统上的程序应避免使用指针.

3260

"功能'<名称>' 拥有太多的自变量: 增加目标系统自变量堆的大小."

3400

"读取输入变量时发生错误"
.exp 文件包含一个正确的存取变量段。

3401

"装载变量配置时发生错误"
.exp 文件包含一个正确的配置变量段。

3402

"装载全局变量时发生错误"
.exp 文件包含一个正确的全局变量段。

3403

"目标<name>不能被装载"
.exp 文件中的对象<name>段不正确。

3404

"装载任务配置时发生错误"
.exp 文件中的任务配置段不正确。

3405

"装载 PLC 配置时发生错误"
.exp 文件中的 PLC 配置段不正确。

3406

″SFC 程序组织单元存在同名步″。第二步将不能载入。″
.exp 文件中的 SFC POU 段包含有相同名的两个步。在输出文件中重新命名其中的一步。

3407

″当前步的上一步未找到″
.exp 文件中丢失步<name>。

3408

″当前步的后续步未找到″
.exp 文件中丢失步<name>。

3409

″ ’<′ name>’步转换条件不正确 ″
.exp 文件中，丢失转换名，它需要步<name>作为先行步。

3410

″ 对应转换条件’<name>’的步不正确″
.exp 文件中，丢失一个步，它需要转换名<name>作为先行条件。

3411

″ ’<name>’步与初始步的联系丢失″
.exp 文件中，丢失步<name>与初始步之间的连接。

3412

″宏’<name>’不能被载入″

3413

″CAM 文件载入时发生错误.″

3414

"CNC 文件载入时发生错误"

3415

"报警配置载入时发生错误"

3450

"'<PDO-name>':COB-ID 丢失"
为此模块，点出 PLC 配置对话框中的“properties”（属性），并输入用于 PDD<PDD 名>的一个 COB ID。

3451

"EDS 文件载入错误：硬件配置引用了此文件，但是找不到该文件"
CAN 配置所需的设备文件不在正确的目录内。检查“project”（项目）“options”（选项）“目录”
中的配置文件的目录设置。

3452

"'<Name>'模块不能被创建!"
<Name>'模块的设备文件不兼容当前的设置，应根据 CPAC 的相关设置对其进行修改。

3453

"'<Name>'通道不能被创建!"
'<Name>'通道的设备文件不兼容当前的设置，应根据 CPAC 的相关设置对其进行修改。

3454

"'<name>'地址指向的内存已使用!"
在对话框 PLC 配置的“setting” （设置）中，已启动选项“check for overlapping addresses” （检
查重叠地址），并已检测到一个重叠。注意，区域检查是基本尺寸，它与模块的数据类型有关。在配置文件中，
通过登录项“size”（尺寸），给出区域尺寸。

3455

"载入 GSD 文件出错：硬件配置中要引用此文件，但是找不到该文件!"
Profibu 配置所需的设备文件不在正确的目录内。检查“project”（项目）“options”(选项）“目
录”中的配置文件的目录设置。

3456

"总线设备驱动'<name>'无法被创建!"
用于模块<name>的设备文件不适应于当前的配置。自建立配置以来已作了修改，或者它已损坏。

3457

"模块解析错误!"
请检查这个模块的数据文件。

3458

"PLC 配置无法完成，请检查配置文件."

3459

"不支持所选波特率."

3460

3S_CanDrv.lib 库版本号错误。

3461

"3S_CanOpenMaster.lib 库版本号错误."

3462

"3S_CanOpenDevice.lib 库版本号错误."

3463

"3S_CanOpenManager.lib 库版本号错误."

3464

"3S_CanOpenNetVar.lib 库版本号错误."

3465

"CAN 通讯从模块的副索引应连续标号"

3466

"CAN 总线变量：PLC 配置中未发现 CAN 通讯控制器"

3468

"CAN 总线驱动：任务配置中未实现任务更新."

3469

"CANOpen 通讯主模块不能被调用，请手动分配其任务."

3470

"任务更新常数名称非法"

3500

"'<name>'变量未在变量列表中声明"
将用于变量的说明插入到全局变量表内，该表包含“variable_configuration”（变量—配置）。

3501

"在变量列表中声明的'<name>'变量未分配相应的内存地址."
将这个变量的一个地址分配给全局变量表，该表包含“variable_configuration”（变量—

配置）。

3502

″变量配置列表中的变量'<name>'数据类型声明有误″
在包含“variable_configuration”（变量—配置）。的全局变量表中，变量以不同于 POU 中的数据类型
说明。

3503

″变量配置列表中的变量'<name>'数据类型有误″
在包含“variable_configuration”（变量—配置）。的全局变量表中，变量以不同于 POU 中的数据类型
说明。

3504

″变量配置中不支持初始赋值″
“variable_configuration”（变量—配置）的一个变量是以地址和初始值说明的。但是，一个初始值
只能为没有地址分配的输入变量而定义的。

3505

″'<name>'路径不是合法的实例路径名 ″
“variable_configuration”（变量—配置）包括一个存在的变量。

3506

″存取路径未指定″
在“access variables” （存取变量）的全局变量表中，一个变量的存取路径是不正确的。正确的应
该是：《标示符》：《存取路径》：《类型》《存取方式》

3507

″存取变量对应的内存使用不合法″

3550

″标识符'<name>'重复定义″
使用一个相同的名来定义两个任务。重新命名其中的一个。

3551

″任务'<name>'中至少应包含一个程序调用″
插入一个程序调用或删除任务。

3552

″任务'<name>'使用的事件变量'<name>'未定义″
在任务属性的“single”（单）区域对话框中设置的一个事件变量，在项目未经全局说明。利用其他变
量或全局定义该变量。

3553

″任务'<name>'中的事件变量'<name>'必须是布尔类型变量″
在任务属性的“single”（单）区域对话框中，使用一个类型 BOOL 的变量作为事件变量。

3554

″任务入口 POU 必须是程序或者全局功能块实例″
在该字段中，输入“program call” (程序调用)一个功能或一个不定义的 POU。输入一个有效的程序名。

3555

″任务入口 POU 包含非法参数″
在该字段中，所用的”append program call”（附加程序）参数不符合程序 POU 的说明。

3556

″当前目标系统不支持此任务″

3557

″任务数超出最大范围″

3558

″任务'<name>'的优先权超出合法范围>'″

3559

″任务'<name>'：当前目标系统不支持任务间隔调用模式″

3560

″任务'<name>'：当前目标系统不支持任务 freewheeling 调用模式″

3561

″任务'<name>'：当前目标系统不支持任务触发调用模式″

3562

″任务'<name>'：当前目标系统不支持任务外部触发调用模式″

3563

″任务'<name>'的调用间隔设置超出合法范围″

3564

″ 当前目标系统不支持任务'<name>'所设置的外部触发事件'<name>'″

3565

″定义的事件触发任务数超出最大范围″

3566

"定义的间隔运行任务数超出最大范围"

3567

"定义的 freewheeling 任务数超出最大范围"

3568

"定义的外部事件触发任务数超出最大范围"

3569

"设定为系统事件触发的 POU 未定义"

3570

"多个任务设定的优先权相同"

3571

"工程未包含 SysLibCallback.lib 库，不能生成系统事件."

3572

"任务'<name>'的看门狗时间间隔设定超出合法范围"

3573

"看门狗时间设定超出合法范围"

3574

"事件变量对应的内存地址不能在事件中重复使用"

3575

″任务'<名称>': 循环时间应该是 '<数字>'的整倍数 .″

3600

″隐含变量未找到!″
使用命令”rebuild all”（重建所有）。如依然得到出错消息，请于 PLC 制造商联系。

3601

″'<name>'变量名称是系统保留用名，请更改″
给出的变量在项目中说明，虽然它是保持用于代码发生器。重新命名该变量。

3610

″ 不支持'<name>'″
给出的属性不被编译系统的当前版本支持。

3611

″提供的编译目录名'<name>'非法″
用于编译文件的“project” （项目）—“options” （选项）—“directories”（目录）中给出的目
录是无效的。

3612

″超出系统所能处理最大 POU 数量，编译被中止.″
在项目中所用的 POU 和数据类型太多。在“target setting/memory layout”（目标设置/存储器布局）
中，修改 POU 的最大数。

3613

″工程编译取消″
编译过程被用户删除。

3614

″工程必须包含一个名为 PLC_PRG 的主程序或者进行任务配置″
建立一个类型”program”（程序）的初始 POU，或建立一个任务配置。

3615

″<名称>（主要的程序）必须有程序类型″
在项目中所用的一个初始 POU 不是” program”（程序）类型。

3616

″程序不能在外部库中执行″
应作为外部库保存的项目包含一个程序。当使用该库时，不能使用这个程序。

3617

″内存不足″
增加你的计算机的虚拟存储器的容量。

3618

″当前代码生成器不支持位存取!″
当前设置的目标系统的代码发生器，不支持对变量的位存取。

3619

″库文件'<name>'与其对应的目标文件版本不合!″

3620

″PLC_PRG 不能出现在库文件中″

3621

″无法对编译文件'<name>'进行写操作″

3622

″无法创建符号文件'<name>'″

3623

″无法对引导工程文件'<name>'进行写操作″

3624

″目标设置 <目标设置 1>=<设置值> 与 <目标设置 2>=<设置值> 不兼容″
在目标设置对话框内检查并纠正这些设置(资源标签)。如果这些设置不可见且不可在那里编辑，请联系
PLC 制造商。

3700

″ 工程中存在与库中同名的 POU″
一个 POU 名已用于项目内，这一名早已用于一个库 POU。重新命名 POU。

3701

″ 目标名不能与 POU 同名″
使用命令“project”(项目)--“rename object”(重新命名对象),重新命名“object organizer（对
象组织程序）”中的 POU 或在说明窗内改变 POU 的名。在那里，POU 名必须放置在临近的关键字 PROGRAM，
FUNCTION 或 FUNCTIONBLOCK 中的一个。

3702

″标识符声明过量″
最大为 100 个标示符可输入到一个变量说明内。

3703

″标识符'<name>'重复定义 ″
要注意，在 POU 的说明部分中，只允许一个标示符有给定的名。

3704

″数据存在递归调用″

3705

″PLC_PRG 中不允许使用 VAR_IN_OUT 型变量″

3706

″标识符 ’常量’ 只允许在 ’VAR’, ’VAR_INPUT’, ’VAR_EXTERNAL’ 和’VAR_GLOBAL’ 中应用。″

3720

″AT 关键字应紧接内存地址″
在关键字 AT 后面加一个有效地址，或修改关键字。

3721

″只有变量以及全局变量能定位于某个内存地址″
将说明放置在一个 VAR 或 VAR-GLOBAL 说明区。

3722

″只有布尔型变量允许使用位地址″
修改地址或修改分配地址的变量类型。

3726

″常量不能存储于指定的内存中″
这种类型的变量不能放置在给定的地址上。

3727

″该地址不允许定义数组″

3728

″非法的内存地址″

3729

″内存地址'<name>'中存储的数据类型非法 ″

3740

″非法类型 ″
在一个变量说明中，使用无效的数据类型。

3741

″非法的数据类型″
使用一个关键字或一个操作符以代替一个有效类型标示符。

3742

″应指定枚举值″
在枚举类型定义中，在打开的括号后，或在括号之间的逗号后面，丢失一个标示符。

3743

″枚举应使用整型数字进行初始化″
枚举只能以类型 INT 的数初始化。

3744

″枚举常量名已被定义过″
检查一下，你是否遵循了以下的枚举值定义规则：
. 在一个枚举定义中，所有的值都应是唯一的。
. 在所有的全局枚举定义中，所有的值均应是唯一的。
. 在所有的局部枚举定义中，所有的值均应是唯一的。

3745

″子区域中只允许使用整型数据类型!″

子范围类型只能在整数数据类型上定义。

3746

″子区域'<name>'与基本数据类型不兼容″
子范围类型的范围极限设定中，有一个极限设定超出其基本类型的有效范围

3747

″字符串'<name>'长度未知″
没有一个有效的常数用于串长的定义。

3748

″最多只能定义 3 维数组″
在一个数据的定义中，给出允许的维数即大于三维。若可应用的话，使用“ARRAY OF ARRAY”（数组的
数组）。

3749

″下限'<name>'未定义″
使用一个为定义的常数来定义一个子范围或数组类型的下限。

3750

″上限'<name>'未定义″
使用一个为定义的常数来定义一个子范围或数组类型的上限。

3751

″字符串长度非法″

3752

″数组嵌套使用最大不能超过 9 维″

3760

″初始值错误″
使用一个与类型定义相当的初始值。为了更改说明，你可使用变量的说明对话框。(SHIFT F2 或 EDIT（编
辑）“autodeclare”(自动说明))。

3761

″ VAR_IN_OUT 型变量不允许赋初值.″
在 VAR_IN_OUT 变量的说明部分，去除初始化。

3780

″期待'VAR', 'VAR_INPUT', 'VAR_OUTPUT' 或 'VAR_IN_OUT'等类型变量″
POU 名后的第一行必须包含这些关键字中的一个关键字。

3781

″期待 END_VAR 标识符″
在说明窗内的给出行的起始位置输入一个 END—VAR 的有效标示符。

3782

″意外的结束″
在说明性编辑器中：在说明部分结束处加上关键字 END—VAR。在编译部分的文本编译器中：加上一个指
令，它中止最后的指令顺序。

3783

″期待 END_STRUCT 标识符″
应确实弄清楚，类型说明已正确的终止。

3784

″当前目标系统不支持设定的此属性″

3800

″全局变量占用大量内存，请在工程选项增加内存使用量.″
增加在对话框”project” （项目）--“options” （选项）--“build”（建立）的设定中所给出的程
序段数。

3801

″'<name>'变量太大″
变量使用大于 1 个数据段的类型。段尺寸是一个目标特定的参数并可在目标设定/存储器布局中修改。如
果你在当前的目标设定中找不到它，请与你的 PLC 制造商联系。

3802

″保留内存溢出.″
可供保变量用的内存空间已耗尽。在目标设定存储器布局中可设定目标专用的内保存区域尺寸。如果你
在对话框中找不到设定字段，请与你的 PLC 制造商联系。

3803

″全局数据存储内存溢出.″
可供全局变量用的内存空间已耗尽，在目标设定/存储器布局中可设定目标专用的内保存区域尺寸。如果
你在对话框中找不到设定的字段。请与你的 PLC 供应商联系。

3820

″功能中不允许使用 VAR_OUTPUT 和 VAR_IN_OUT 型变量″
在一个功能中，可以定义无输出或输入-输出变量。

3821

″功能至少应有一个输入接口″
为这种功能至少附加一个输入参数。

3840

″未知的全局变量'<name>'!″
在 POU 中，使用一个 VAR_EXTERNAL 变量，对它未说明其全局变量。

3841

″'<name>'声明与全局声明不匹配!″
在 VAR_EXTERNAL 变量说明中给出的类型与全局说明中的类型不匹配。

3850

″在包装的'<名称>'结构体内部对未包装'<名称>'的结构体的声明是不允许的!″

3900

″标识符的多重强调″
在标示符名下除去多重下划线。

3901

″地址表达式至多允许包含 4 个数字段″
有一种表达式允许直接赋值给一个地址，它有不止四层的数值域。

3902

″关键字必须大写″
对关键字应使用大写字母，或启动选项，“project”（项目）—“options”--（选项）中的
“autoformat”（自动格式化）。

3903

″非法的 duration 型常量″
常数的计数法不符合 IEC61131-3 格式。

3904

″duration 型常量溢出″

用于时间常数值不能以内部格式表示。可表示的最大值为 t#49d17h2m47s295ms

3905

″非法的日期常量″
常数的计数法不符合 IEC61131-3 格式。

3906

″非法的时间常量″
常数的计数法不符合 IEC61131-3 格式。

3907

″非法的日期和时间常量″
常数的计数法不符合 IEC61131-3 格式。

3908

″非法的字符串常量″
串常数包含一个无效的字符。

4000

″期望标识符″
在这个位置输入一个有效的标示符。

4001

″没有声明变量’〈名称〉’″
说明变量是局部或全局。

4010

″类型搭配错误：不能转换’〈名称〉’为’〈名称〉’.″
检查一下，操作符期望的是何种数据类型，并改变引起错误的变量类型，或选择其他变量。

4011

″在’<名称>’中参数类型搭配错误：不能转换’<名称>’为’<名称>’.″
实际参数的数据类型不能自动转换成格式化参数的类型。使用某种类型转换方式或使用其他变量类型。

4012

″在’<名称>’中参数类型搭配错误：不能转换’<名称>’为’<名称>’.″
将一个无效类型《类型 2》变量分配给输入变量’<name>’。将变量或常数以一个类型《类型 1》变量代
之，或采用一种类型转换。相应的，一个带有类型前缀的常数。

4013

″在’<名称>’中输出类型搭配错误：不能转换’<名称>’为’<名称>’.″
将一个无效类型《类型 2》变量分配给输入变量’<name>’。将变量或常数以一个类型《类型 1》变量代
之，或采用一种类型转换。相应的，一个带有类型前缀的常数。

4014

″无夸张类型：不能转换’<名称>’为’<名称>’″
常数类型与前缀类型不兼容。

4015

″对于直接的位存储数据类型’<名称>’不合法″
直接位编址只允许用于整数和“位串”数据类型。你在位存取<var1>.<bit>中，使用 RIAL/LREAL 类型的
变量 var1,或一个常数。

4016

″对于’<名称>’类型的变量的位索引’<编号>’超出范围″
你正式试图存取的一位，它不是为变量的数据类型而规定的。

4017

″对于’REAL’， ’MOD’没有定义″
操作符 MOD 只可用于整数和位串数据类型。

4020

″具有写权的变量或直接地址需要’ST’,’STN’,’S’,’R’″
使用一个具有写存取的变量取代第一个操作数。

4021

″变量’<名称>’不具有写权″
使用一个具有写存取的变量取代该变量。

4022

″期望操作数″
在注释后面加上一个操作数。

4023

″在’＋’或’－’后需要数字″
输入一个数字。

4024

″在’<名称>’前需要<操作数 0>或<操作数 1>’或……″
在命名的位置处，输入一个有效的操作数。

4025

″在’<名称>’前需要’:=’ 或 ’=>’″
在命名的位置处，输入二种操作符中的一种。

4026

″‘BITADR’需要一个位地址或关于位地址的一个变量″
使用一个有效的位地址。

4027

″需要一个整数符号或常数符号″
输入一个整数或一个有效常数的标示符。

4028

″‘INI’操作数需要功能块实例或数据单元类型实例″
检查 INI 操作符所使用的变量的数据类型。

4029

″不允许对同一个功能进行嵌套.″
在不是可重入的目标系统和在仿方式时，一个功能调用并不包含作为参数的自身调用。

4030

″‘ADR’操作数不允许用表达式或常量″
使用一个变量或一个直接地址取代常数或表达式。

4031

″位操作不允许用’ADR’！请用‘BITADR’来代替.″
使用 BITADR。请注：BITADR 功能不返回一个物理的内存地址。

4032

″对于’〈名称〉’来说，’〈数字〉’操作数太少。至少需要’〈数字〉’个″
检查一下，命名的操作符需要多少操作数，并添加缺少的操作数。

4033

″对于’〈名称〉’来说，’〈数字〉’操作数太多。至少需要’〈数字〉’个″
检查一下，命名的操作符需要多少操作数，并除去多余的操作数。

4034

″用 0 来除″
你正在一个常数表达式中以 0 作为除数。如果你要引起一个运行时的错误，使用—如可以

应用的话—一
个有值为 0 的变量。

4035

″如果’代替常量’被激活，’VAR’必须在’VAR CONSTANT’中应用″
使用直接值的一个常数地址存取是不可能的。如可应用的话，取消”project”（项目）---“options”
（选项）---“build”（建立）

4040

″标签’<名称>’没有定义″
使用名（标记名）定义一个标记，或将名（标记名）改变成一个定义的标记名。

4041

″标签’<名称>’进行了多重定义″
在 POU 中，标记<name>是重复定义的。重新命名标记或从重复定义中除去一个名。

4042

″在序列中没有比’<数字>’更多的标签可以被允许″
跳转标记数限制在“<Anzahl>”。插入一个哑指令。

4043

″标签格式错误。一个标签在冒号后必须定义一个随意的名字.″
标记名是无效的，或在表达式中丢失了冒号。4050

4050

″没有定义 POU’%S’″
利用命令”project”（项目）--“Add Object”（附加对象）来定义有名<name>的一个 POU，或将 <name>
改变成一个已定义的 POU 名。

4051

″‘%S’没有功能″

替代<name>，使用一个在项目或在库中定义的功能名。

4052

″’<名称>’必须是 FB’<名称>’声明的一个实例″
使用一个在项目中定义的数据类型<name>的实例，或将（实例名）类型改为<name>。

4053

″’<名称>’没有有效的盒子或操作符″
使用一个 POU 名或一个在项目定义的操作符来取代<name>。

4054

″所给参数没有有效的 POU 名称″
给出的参数不是一个有效的 POU 名。

4060

″’<名称>’的‘VAR_IN_OUT‘参数’<名称>’需要写权作为输入″
具有写存取的变量必须移交给 VAR_IN_OUT 参数,这是因为,一个 VAR_IN_OUT 可在 POU 内修改。

4061

″’<名称>’的‘VAR_IN_OUT‘参数’<名称>’必须被用.″
VAR_IN_OUT 参数必须有移交的具有写存取的变量，这是因为，一个 VAR_IN_OUT 可在 POU 中修改。

4062

″’<名称>’的‘VAR_IN_OUT‘参数’<名称>’没有外部接口.″
VAR_IN_OUT 参数只能在 POU 内写入或读取，这是因为它们是由引用转交的。

4063

″’<名称>’的‘VAR_IN_OUT‘参数’<名称>’不准应用位地址.″
一个位地址不是一个有效的物理地址。转交一个变量或一个直接的非位地址。

4064

″在本地调用时‘VAR_IN_OUT‘必须复写!″
在局部动作调用中，删除用于 VAR_IN_OUT 变量的参数集。

4070

″POU 包含太复杂的表达式″
通过将表达式分成几个表达式来降代嵌套深度。为此，可使用中间变量。

4071

″网络太复杂″
将网络分成几个网络。

4072

″在 FB 类型’<名称>’和实例’<名称>’的动作标识符应用不一致.″

4100

″‘^‘需要指针类型″
你正在试图将一个不作为指示符说明的变量非关联化。

4110

″[<索引>]需要数组变量″
[<index>]用于一个变量，它不是作为有 ARRAY OF 的一个数组加以说明的。

4111

″一个数组的索引表达式必须是‘INT‘类型″
使用正确类型的表达式，或类型转换。

4112

″数组有太多索引″
检查索引数（1.2 或.3），这些索引是说明数组的，除去多余的索引。

4113

″数组有太少索引″
检查索引数（1.2 或.3），这些索引是说明数组的，添加缺少的索引。

4114

″一个常量索引超出数组范围″
确实弄清楚，所用的索引是在数组界限之内。

4120

″‘.‘需要变量结构体″
句点左边的标识符必须是一个类型 STRUCT 或 FUNCTION_BLOCK 的变量，或是名为一个 FUNCTION 或一个
PROGRAM 的变量。

4121

″’<名称>’不是’<对象名称>’的成分″
成分<name>不包括在对象<object name>的定义内。

4122

″’<名称>’不是调用的功能块的输入变量″
检查一下，被调用的功能块的输入变量，并将’<name>’改变成这些变量中的一个变量。

4200

″需要 LD″
在 IL 编辑器中，在跳转符号后面，至少应插入一个 LD 指令。

4201

″需要 IL 操作符″
每个 IL 指令必须用一个操作符或一个跳转符开始。

4202

″括号内文本意外结束″
在文本后插入一个关闭括号。

4203

″’<名称>’在括号内不允许″
操作符<name>在一个 IL 括号表达式中是无效的。（无效的是：JMP,RET,CAL,LDN,LD,TIME）

4204

″对于结束括号缺少相应的开始括号″
插入一个打开括号，或除去这个关闭括号。

4205

″在‘）‘后不允许逗号″
在关闭括号后面除去逗号。

4206

″括号内不允许有标签″
移去跳转符号，使其在括号外面。

4207

″‘N‘修正符需要操作数类型’BOOL‘’ BYTE‘’ WORD‘或’DWORD‘″
N 修改符需要可执行布尔求反的数据类型。

4208

″条件操作符需要‘BOOL‘类型″
确实弄清楚，表达式给出布尔结果，或应用类型转换。

4209

″在此不允许功能名″
使用一个变量或一个常数来替换这个功能。

4210

″'CAL', 'CALC' 和 'CALN'需要功能块实例作为操作数″
说明一个你要调用的功能块的实例。

4211

″在 IL 程序中只允许在行的末尾进行注释″
将注释移到行的结束处，或移到附加行。

4212

″条件声明前的累加器错误″
未定义这个累加器。如果一个指令是领先的，它未提交一个结果。

4213

″'S'和'R'需要'BOOL'操作数″
在这个位置使用一个布尔变量。

4250

″在 POU 结尾需要另一个'ST'声明″
该行不是以一个有效的 ST 指令开始。

4251

″在功能'<名称>'内有太多参数″
给出的参数多于功能定义中所说明的参数。

4252

″在功能'<名称>'内有太少参数″
给出的参数少于功能定义中所说明的参数。

4253

″'IF'或'ELSIF'需要'BOOL'表达式作为条件″
确实弄清楚，IF 或 ELSIF 的条件是一个布尔表达式。

4254

″’ WHILE‘需要’BOOL‘表达式作为条件″
确实弄清楚，WHILE 后的条件是一个布尔表达式。

4255

″’ UNTIL‘需要’BOOL‘表达式作为条件″
确实弄清楚，UNTIL 后的条件是一个布尔表达式。

4256

″‘NOT’ 需要’BOOL‘操作数″
确实弄清楚，NOT 后的条件是一个布尔表达式。

4257

″‘FOR‘声明的变量必须是’INT‘类型″
确实弄清楚，计数器变量是一个整数或位串数据类型。（例如：DINT，DWORD）

4258

″‘FOR‘声明的表达式没有具有可写权的变量″
使用一个有写存取的变量来替换计数器变量。

4259

″‘FOR‘声明的开始值没有具有可写权的变量″
“FOR”指令中的起始值必须与计数器变量类型相兼容。

4260

″‘FOR‘声明的结束值没有具有可写权的变量″
“FOR”指令中的结束值必须与计数器变量类型相兼容。

4261

″‘FOR‘声明的增加值没有具有可写权的变量″
FOR”指令中的起始值必须与计数器变量类型相兼容。

4262

″循环外需要退出″
只能在“FOR”，“WHILE”或“UNTIL”指令内使用“EXIT”。

4263

″需要数字，‘ELSE‘或‘END_CASE‘″
在一个“CASE”表达式内，你只能使用一个数或一个“ELSE”指令，或结束指令“END_CASE”。

4264

″‘CASE‘需要整数类型选择器″
确实弄清楚，选择器是一个整数或位串数据类型的（例如 DINT，DWORD）。

4265

″‘，‘后需要编号″
在 CASE 选择器的枚举时，在逗号后面必须插入一个其他选择器。
4266
″至少需要一个声明″
插入一个指令，至少一个分号。

4267

″功能块的调用需要功能块实例″
功能块的调用中的标识符没有实例。说明所需要的功能块的一个实例，或利用一个早已定义的实例。

4268

″需要表达式″
在这里插入一个表达式。

4269

″在'ELSE'分支后需要有'END_CASE'″
使用一个“END_CASE”来终止“ELSE”部分后面的“CASE”指令。

4270

"'CASE' 常量'<名称>' 已经被应用"
一个“CASE”选择器在一个“CASE”指令内只能使用一次。

4271

"范围的下限值比上限值大."
修改选择器的区域边界，使其下部边界不大于上部边界。

4272

"在调用'<名称>'的地方<位置>需要参数'<名称>'!"
你可以这样的方式编辑一个功能调用，使它还包含参数名，而不仅包含参数值。但是，无论如何，参数
的位置（顺序）必须与功能定义中相同。

4273

"部分'CASE'范围'<范围>'在'范围'<范围>'中已经被应用"
确实弄清楚，用于 CASE 指令中的选择器区域不出现重叠。

4274

"在'CASE'声明时出现多重的 'ELSE'分支"
一个 CASE 指令不能包含多于一个 ELSE 指令。

4300

"跳转需要'BOOL'作为输入类型"
确实弄清楚，相应于 RETURN 指令的跳转输入是一个布尔表达式。

4301

"POU '<名称>' need exactly 需要正确的 <数字> 输出"
输入数不对应于在 POU 定义中给出的 VAR_INPUT 和 VAR_IN_OUT 变量数。

4302

″POU ’<名称>’ 需要正确的输出 %d ″.
输出数不对应于在 POU 定义中给出的 VAR_OUTPUT 变量数。

4303

″’<名称>’ 没有操作符″
使用一个有效的操作符来替换<name>。

4320

″一个触点的开关信号必须是布尔类型的表达式″
用于一个接点的开关信号必须是一个布尔表达式。

4321

″线圈的输出变量必须是布尔类型。″
一个线圈的输出变量必须是类型 BOOL。

4330

″功能块 ’<名称>’的输入’EN’处期望一个表达式 ″
将一个输入或一个表达式分配给 POU“<name> ”的输入 EN。

4331

″功能块 ’<名称>’的输入’<编号>’处期望一个表达式 ″
没有分配操作符 POU 的输入<number>。

4332

″ 功能块 ’<名称>’的输入’<名称>’处期望一个表达式″
POU 的输入是类型 VAR_IN_OUT 而且没有分配。

4333

″跳转处期望一个标识符″
给出的跳转标记不是一个有效的标识符。

4334

″跳转的输入处期望一个表达式″
将一个布尔表达式分配给 RETURN 指令的输入。如这是 TRUE，则将执行这个跳转。

4335

″返回的输入处期望一个表达式″
将一个布尔表达式分配给 RETURN 指令的输入。如这是 TRUE，则将执行着个跳转。

4336

″输出的输入处期望一个表达式″
将一个合适的表达式分配给输出框。

4337

″期望输入的标识符″
在输入框中插入一个有效的表达式或标识符。

4338

″ ’<名称>’功能块没有输入″
对操作符 POU<name>输入分配有效的表达式。

4339

″输出时的类型搭配错误: 不能转换 ’<名称>’ 到 ’<名称>’.
输出框中的表达式类型与应分配给它的表达式类型不兼容。

4340

″跳转需要 ’BOOL’ 作为输入类型″
确实弄清楚，跳转是一个布尔表达式。

4341

″返回需要一个布尔输入″
确实弄清楚，用于 RETURN 指令的输入是一个布尔表达式。

4342

″在功能块 ’<名称>’的输入’EN’处需要一个表达式″
将一个有效的布尔表达式分配给框的 EN 输入。

4343

″常量的值和声明的不一致″
框<name>的输入<name>作为 VAR_INPUT CONSTANT 说明。但是，在对话框“编辑参数”中，已将一个类型
不兼容的表确定式发配给这个 POU 框。

4344

″’S’ 和’R’ 需要’BOOL’操作数″
在相应的“复位”指令的设定后面，插入一个有效的布尔表达式。

4345

″ ’<名称>’的无效的参数类型’<名称>’ : 不能转换 ’<类型>’ 到’<类型>’.″
分配给 POU 框<name> 的输入<name>的一个表达式。类型是不兼容。

4346

″不允许一个常量作为输出″
你只能将一个输出分配给一个变量或一个有写存取的直接地址。

4347

″’VAR_IN_OUT’ 参数需要可写的输入变量″
只有写存取的变量才可移交给 VAR_IN_OUT 参数，这是因为这些变量可在 POU 内修改。

4348

″无效的程序名称 ’<名称>’. 具有同样名称的一个变量已经存在.″

4349

″在 POU <名称>中的输入或输出已经被删除: 检查功能块的所有连接.只有当 CFC 编辑后

此错误消息才会
消失。″

4350

″不能从外部访问一个 SFC-行为动作!″
SFC 动作只能在它们被定义的 SFC POU 内才能调用。

4351

″步的名称没有标识符: '<名称>'″
重新命令步名，或选择一个有效的标识符作为步名。

4352

″有效的步名称后有额外的字符:'<名称>'″
在步名中除去无效的字符。

4353

″步名称被复写: '<名称>'″
重新命名一个步名。

4354

″跳转到没有定义的步: '<名称>'″
选择一个现有的步名作为相应的跳转的目标。插入一个有名的步，“<name>”

4355

″一个转换必须没有任何的边缘效应(分配, FB-调用等.)″
一次转换必须是一个布尔表达式。

4356

″跳转时没有有效的步名称: '<名称>' ″
使用一个有效的标识符作为跳转目标（标记）。

4357

″没有找到 IEC-库″
检查一下，库 iecsfc.lib 是否插入在库管理程序内，以及在”project”（项目）——“options”（选
项）——“paths”(路径)中规定的库路径是否正确。

4358

″行为动作没有声明：’<名称>’″
确实弄清楚，在对象组织程序内，IEC 步的动作是在 SFC POU 下面插入的，而且在编辑器中，动作名插
入在限定符右边的框内。

4359

″无效的限定词：’<名称>’″
在动作名左边的框内，输入一个用于 IEC 动作的限定符。

4360

″ 限定词’<名称>’后期望时间常数″
紧邻动作名左侧的框，在一个限定符后输入一个时间常数。

4361

″’<名称>’ 不是行为动作的名称″
紧邻限定线右侧的框，输入一个动作名，或在项目中定义的一个变量名。

4362

″行为动作中没有布尔表达式的应用：’<名称>’″
插入一个布尔变量或一个有效的动作名。

4363

″IEC-步名称已经被变量应用：’<名称>’″
请重新命名步或变量。

4364

″一个转换必须是布尔量表达式″
转换表达式的结果必须是类型 BOOL。

4365

″限定词'<名称>'后期望有时间常数″
打开步”<name>的对话框“步属性”，并输入一个有效的时间变量或时间常数。

4366

″平行分支的标签没有有效的标识符: '<名称>'″
在三角形符号旁输入一个有效的标识符，它标记跳转符。

4367

″ '<名称>' 标签已经用过″
早已有一个跳转符或一个步使用过这个名。请相应的重新命名。

4368

″'<名称>'行为在多重的一系列步中应用，也就是一个包含其它的。″
动作<name>用于 POU 以及 POU 的一个或几个动作。

4369

″一个转换需要一个严密的网络″
对于一个转换使用了几个 FBD 相应的 LD 网络。请减少到一个网络。

4370

″正确的 IL-转换后有额外的行″
在转换结束处除去不必要的线路。

4371

″有效的表达式后有无效的字符: '<名称>″
在转换结束处除去不必要的字符。

4372

"'<名称>'步: 时间限制需要'TIME'类型"

4373

" 只在 SFC-POUs 下允许 IEC-行为"

4374

"期望步代替'<名称>'转换 "

4375

"期望转换代替'<name>'步"

4376

"'<名称>'转换后期望步"

4377

" '<名称>'步后期望转换"

4400

POU '<名称>'输入/ 转换 有错误。例如没有完成."
POU 不能完全转换到 IEC61131-3，相应的丢失一个操作数。

4401

"S5 时间常量 <名称> 秒数太大 (最大. 9990s)."
在累加器内没有有效的 BCD 编码时间。

4402

"只有 I/O 可进行直接访问."
确实弄清楚，你只存取定义为输入或输出的变量。

4403

"STEP5/7 结构体无效或不能转换成 IEC 61131-3."
有些 STEP5/7 命令不能转换到 IEC61131-3，相应的丢失一个操作数。

4404

"STEP5/7 操作数无效或不能转换成 IEC 61131-3."
有些 STEP5/7 操作数不能转换到 IEC61131-3，相应的丢失一个操作数。

4405

"重置一个 STEP5/7 定时器不能被转换成 IEC 61131-3."
相应的 IEC 定时器没有复位输入。

4406

"STEP5/7 计数器常量超出范围 (最大是 999)."
累加器中没有有效的 BCD 编码的计数器常数。

4407

"STEP5 结构体不能转换成 IEC 61131-3."
有些 STEP5/7 指令不能转换成 IEC61131-3，例如 DUF。

4408

"定时器 或 计数器 的位访问不能转换成 IEC 61131-3."
专用的定时器/计数器命令不能转换成 IEC61131-3。

4409

"ACCU1 或 ACCU2 的内容没有定义, 不能转换成 IEC 61131-3."
与两个累加器相连接的一个命令是不能转换的，这是因为未规定累加器的值。

4410

"没有在工程中调用 POU ."
输入被调用的 POU。

4411

"全局变量表出现错误."
请检查 SEQ 文件。

4412

"内部错误编号 11"
请与 PLC 制造商联系。

4413

"在数据块中格式化行时出错"
应输入的代码中有一个错误的日期。

4414

"FB/FX 名称丢失."
在需始的 S5D 文件中，丢失一个（扩展的）POU 的符号名。

4415

"不允许块结束后的说明."
不能输入一个受保护的 POU。

4416

"命令无效"
S5/S7 命令不能被取消。

4417

"注释没有封闭"
使用“*）”来关闭注释。

4418

"FB/FX-名称太长（最多 8 个字符)"
一个（扩展的）POU 的符号名太长。

4419

″期望对行″″(* 名称: 〈FB/FX-名称〉 *)″″格式化 ″
校正相应的行。

4420

″FB/FX 参数名称丢失″
检查 POU。

4421

″FB/FX 参数类型无效″
检查 POU。

4422

″FB/FX 参数类型丢失″
检查 POU。

4423

″ FB/FX 调用参数无效。″
检查 POU 的接口。

4424

″警告: 调用 FB/FX 时 丢失或参数错误或有'0'参数。″
被调用的 POU 尚未输入，或不正确，或无参数（在最后一种场合，你可忽略出错消息）。
4425
″标签定义丢失″
未定义跳转的目标（标记）。

4426

″POU 没有有效的 STEP 5 块名称, 例如 PB10″
修改 POU 名。

4427

″时间类型没有声明″
在全局变量表中加入一个定时器的说明。

4428

″超出打开 STEP5 支架的最大数量。″
不允许使用多于七个的开括号。

4429

″正式参数的名称错误。″
参数名不能超过四个字符。

4430

″参数的正式类型不是 IEC 可改变的。″
在 IEC61131-3 中，定时器，计数器和 POU 不能转换为形式参数。

4431

″调用 STEP5 STL 时其 'VAR_OUTPUT' 有太多的参数。″
一个 POU 不能包含多于 16 个形式参数作为输出。

4432

″不允许标签带表达式。″
在 IEC61131-3 中，跳转标记不能插入在任何需要的位置。

4434

″太多标签。″
一个 POU 不能包含多于 100 个标记。

4435

″在转移 / 调用后，必须启动一个新的表达式″
跳转或调用后，必须后随一个“load”（装入）命令 LD。

4436

″结果位没有定义，不能转换成 IEC 61131-3.″
VKE 使用的命令不能转换，这是因为，VKE 的值是未知的。

4437

″指令和操作数的类型不一致″
一个值命令用于一个字操作数（或反过来也如此）。

4438

″数据块没有打开 (在 DB 之前插入 C 指令)″
插入一个 ADB。

4500

″变量或地址未被承认″
在项目内未说明监视变量。通过按 F2 按钮，你得到列出说明变量的输入辅助。

4501

″有效监视表达式后的额外特性。″
除去多于的记号。

4520

″程序错误：在'<名称>'前应该有标记!″
附注不正确。检查一下，”<name>”是否是一个有效的标记。

4521

″程序错误：不期望的成分 '<名称>'!″
检查一下，附注是否正确编写。

4522

″标记超出程序的要求!″
丢失断开的附注，添加一个“flag off”（除去标记）指令。

4523

″程序{<程序名称>} 不允许的界面类型 '<名称>'″

4550

″索引超出定义范围 : 变量 OD ″<编号>, 行<行号>.″
保证索引是在目标设定/网络功能度中所规定的区域内。

4551

″分索引超出定义范围 : 变量 OD ″<编号>, 行<行号>.″
保证子索引是在目标设定/网络功能度中所规定的区域内。

4552

″索引超出定义范围 : 参数 OD ″<编号>, 行<行号>.″
保证索引是在目标设定/网络功能度中所规定的区域内。

4553

″分索引超出定义范围 : 参数 OD ″<编号>, 行<行号>.″
保证子索引是在目标设定/网络功能度中所规定的区域内。

4554

″变量名称错误: 变量 OD <编号>, 行<行号>.″
保证在字段”variable”(变量)中的一个有效项目变量。使用相应于全局变量(variable name)的语
法 POU 名（变量名）。

4555

″登陆表为空, 输入不能随意: 参数 OD <编号>, 行<行号>″
你必须在这个字段做出一个登录项。

4556

″登陆表为空, 输入不能随意: 变量 OD <编号>, 行<行号>″

你必须在这个字段做出一个登录项。

4557

″必须的参数内存太大″

4558

″必须的变量内存太大″

4560

″错误的值：路径’<名称>’，纵列’<名称>’，行’<行号>’″

4561

″纵列没有定义：’<名称>’″

4562

″索引/分索引 已经存在：路径’<名称>’，行’<行号>’″

4563

″标识符’<名称>’ 已经存在：路径’<名称>’，行’<行号>’″

4564

″索引’<名称>’ 超出范围：路径’<名称>’，行’<行号>’ ″
在目标设置定义范围内输入一个索引。

4565

″分索引’<名称>’ 超出范围：路径’<名称>’，行’<行号>’ ″
在目标设置定义范围内输入一个分索引。

4566

″参数管理器输入时发生错误″
用户应该输入一个在参数管理器中包含错误信息的输出文件。检查*.exp 文件。

4600

″网络变量: ’<名称>’ 表达式不是布尔类型。″

4601

″网络变量 ’<名称>’: 发现网络变量交换没有循环或无限循环。″

4602

″’<网络变量名称表>’: 对象用 UDP 端口’<端口号>’ 代替 ’<端口号>’″
在指定的网络变量表 设置中，一个端口号已经应用，它和在全局变量文件夹中找到的第一个网络变量表
应用的端口号不同。当心所有的网络变量表应用同一个端口！

4620

工程中发现没有使用的变量。请参考 ’工程’ ’检查’ 未使用变量 命令的描述。

4621

变量分配内存空间时发生交迭。请参考 ’工程’ ’检查’

’重叠内存范围’命令的描述。

4622

IEC地址分配的同一个内存空间涉及到多个任务。请参考 ’工程’ ’检查’ ’同时访问’命令的描述。

4623

工程在多于一个地方同时获得对同一个内存的写入权。请参考 ’工程’ ’检查’’ ’输出量的

多重访问’命
令的描述。

4650

″轴组’<名称>’: 任务’<名称>’ 不存在.″

4651

″轴组’<名称>’: 没有设置循环时间(dwCycle).″

4652

″驱动器’<名称>’: wDriveID 在此轴组中已经存在.″

4670

″CNC 程序’<名称>’: 没有找到全局变量 ’<名称>’ .

4671

″CNC 程序’<名称>’: 变量’<名称>’ 具有矛盾的类型.″

4685

″凸轮’<名称>’:未知的凸轮表类型.″

4686

″凸轮’<名称>’: 凸轮上的点超出了数据类型范围.″

4700

″’<编号>’ (’<名称>’): 监视表达式 ’<名称>’ 不是数字变量.″

4701

″’<名称>’ (’<编号>’): 监视表达式 ’<名称>’ 不是布尔类型.″

4702

″'<名称>' ('<编号>'): 监视表达式 '<名称>' 不是字符串类型.″

4703

″'<名称>' ('<编号>'): 监视表达式'<名称>'错误″
可视化窗口包含了错误的变量。

4704

″'<名称>'('<编号>'): 监视表'<名称>'初始化值错误.″
检查所用的表。

4705

″'<名称>' ('<编号>'): 报警表分配的报警组无效.″
在报警表配置对话框中输入一个有效的报警组。（报警种类表）

4900

″类型转换错误″
当前选择的代码转换器不支持你所用的类型转换。

4901

″内部错误：数组存取溢出!″
数组范围超过 32 位变量，减少数组索引范围。

5100

″<名称> (<Zahl>): 表达式太复杂。没有寄存器可用。″
对于可用寄存器来说指定的表达式太复杂。请尝试用中间变量来减少表达式的复杂度。