TECO
东元集团

MP300系列运动控制器编程手册

◆ **版本：V1.0**

◆ **发布日期：08/2023**

声明

版权声明

Copyright@2023

版权所有，保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文件内容的部分或全部，并不得以任何形式传播。

由于产品版本升级或其它原因，本文件内容会不定期进行更新。除非另有约定，本文件仅作为使用参考，本文件中的所有陈述、信息和建议不构成任何明示或暗示的担保。

前言

简介

本手册主要介绍了如何对 MP300 系列运动控制器进行编程操作。内容包括：

- 第 1 章主要介绍 CODESYS 的概念和基本组件；
- 第 2 章主要介绍 CODESYS 编辑器；
- 第 3 章主要介绍 CODESYS 编程参考；
- 第 4 章主要介绍 Standard.lib 和 Util.lib 库指令的使用说明；
- 第 5 章主要介绍 Softmotion 库的 SM3_Basic library 库指令使用说明；
- 第 6 章主要介绍 Softmotion 库的 SM3_CNC library 库指令使用说明；

在线支持

除本手册外，还可以在因特网上获取相关的产品资料和技术服务。
http://www.taian-technology.com/

目录

声明 II

前言 III

目录 IV

1 概念和基本组件 1

1.1 简介和基本概念 1

1.2 CODESYS 的结构 2

1.2.1 工程的构成 2

1.2.2 语言 4

1.2.3 联机调试功能 15

1.2.4 标准化 16

2 编辑器 1

2.1 CAM 编辑器 1

2.1.1 使用电子凸轮 7

2.1.2 CAM：成员属性 13

2.2 CFC 编辑器 14

2.3 数据源编辑器 22

2.3.1 添加数据服务器 22

2.3.2 添加数据源 23

2.3.3 配置数据源 25

2.3.4 在视图中使用数据源 29

2.4 CNC 编辑器 30

2.4.1 文本编辑器 30

2.4.2 图形编辑器 33

2.4.3 工具箱 36

2.4.4 CNC 对象 39

2.4.5 CNC 数据结构 43

2.4.6 CNC 在线 44

2.5 CNC 设置编辑器 44

2.6 声明编辑器 47

2.7 设备编辑器 51

2.7.1 通用设备编辑器 51

2.7.2 CAN 总线配置编辑器 68

2.7.3 EtherCAT 设备编辑器 83

2.7.4 Softmotion 设备编辑器 92

2.8 DUT 编辑器 105

2.9 FBD/LD/IL 编辑器 105

2.9.1 语言 106

2.9.2 IL 语言中的修饰符和操作符 107

2.9.3 FBD 和 LD 编辑器视图 109

2.9.4 IL 编辑器视图 111

2.9.5 菜单 114

2.9.6 FBD/LD/IL 元素 115
2.9.7 在线模式下的 FBD/LD/IL 编辑器 122
2.10 全局变量列表编辑器 125
2.11 库管理编辑器 126
2.12 网络变量列表编辑器 129
2.13 符号表配置编辑器 129
2.14 任务编辑器 130
2.14.1 任务编辑器用法 131
2.14.2 在线模式下的任务编辑器 134
2.15 跟踪编辑器 135
2.16 视图编辑器 150
2.17 视图管理编辑器 158
2.18 监控列表编辑器 163
3 编程参考 165
3.1 声明 165
3.1.1 变量声明 165
3.1.2 标识符命名建议 166
3.1.3 变量初始化 171
3.1.4 变量任意初始化 171
3.1.5 声明编辑器 171
3.1.6 自动声明对话框 172
3.1.7 快捷方式 172
3.1.8 AT 声明 172
3.1.9 关键字 173
3.1.10 局部变量-VAR 177
3.1.11 输入变量-VAR_INPUT 178
3.1.12 输出变量-VAR_OUTPUT 178
3.1.13 输入和输出变量-VAR_IN_OUT 178
3.1.14 全局变量-VAR_GLOBAL 178
3.1.15 临时变量-VAR_TEMP 179
3.1.16 静态变量-VAR_STAT 179
3.1.17 外部变量-VAR_EXTERNAL 179
3.1.18 变量类型的属性关键字 179
3.1.19 保持型变量-RETAIN、PERSISTANT 180
3.1.20 常量 CONSTANT，文字 182
3.1.21 变量配置-VAR_CONFIG 183
3.1.22 用户自定义数据类型的声明和初始化 184
3.1.23 可扩展函数-PARAMS 184
3.1.24 FB_init，FB_reinit 方法 185
3.1.25 FB_Exit 方法 186
3.1.26 pragma 命令 187
3.1.27 列表组件功能 210
3.2 数据类型 210
3.2.1 标准数据类型 210
3.2.2 IEC1131-3 标准的扩展数据类型 212
3.2.3 自定义数据类型 217

3.3 操作符 228
3.3.1 IEC 操作符和标准扩展函数 228
3.3.2 算术操作符 228
3.3.3 位操作符 233
3.3.4 移位操作符 235
3.3.5 选择操作符 238
3.3.6 比较操作符 241
3.3.7 地址操作符 244
3.3.8 调用操作符 245
3.3.9 类型转换操作符 245
3.3.10 数学函数 253
3.3.11 IEC 扩展操作符 257
3.4 操作数 267
3.4.1 常量 267
3.4.2 变量 271
3.4.3 地址 273
3.4.4 函数 274
4 Standard.lib 库和 Util.lib 库使用说明 276
4.1 CODESYS 中的操作符总结 276
4.2 Standard.lib 库指令 278
4.2.1 字符串功能 279
4.2.2 双稳态功能块 284
4.2.3 触发器 286
4.2.4 计数器 288
4.2.5 定时器 290
4.3 Util.lib 库指令 295
4.3.1 BCD 转换 296
4.3.2 位/字节算法功能块 296
4.3.3 数学辅助功能块 297
4.3.4 调节器 300
4.3.5 信号发生器 303
4.3.6 函数操纵功能块 307
4.3.7 模拟量监视功能块 309
5 SM3_Basic Library 库使用说明 311
5.1 驱动接口 314
5.2 数据类型 316
5.2.1 基本驱动 316
5.2.2 错误 320
5.2.3 PLCopen 327
5.3 POUs 330
5.3.1 基本驱动 330
5.3.2 错误 338
5.3.3 文件 338
5.3.4 PLCopen 339
5.3.5 Simple Test 377

6 SM3_CNC Library 库使用说明.....378
6.1 数据类型.....380
6.2 全局变量.....386
6.3 SMC_CNC_POUs.....390
6.3.1 File.....390
6.3.2 SoftMotion CNC.....392
6.3.3 SM_Trafo_POUs.....429

1 概念和基本组件

1.1 简介和基本概念

CoDeSys 是一种与设备无关的 PLC 编程系统。CoDeSys 不仅支持所有符合 IEC 61131-3 标准的编程语言，还支持C 语言。与CoDeSys 实时系统结合，可以在一个工程中对多个控制器设备进行配置。

使用 CoDeSys 编程时，请注意以下基本概念：

- ❒ 面向对象：CoDeSys V3.2 中，在编程元素、编程特性、工程结构、版本管理等各方面都体现了面向对象的思想。可以通过联合、实例化来实现一个工程中的多设备和多应用，由此，可以在同一个设备上运行多个应用、可以对应用进行拷贝、可以在一个工程中混合配置参数型和可编程型的设备。
- ❒ 基于组件的编程系统结构：在用户界面中（例如编辑器和菜单等）可以使用的功能，是由在配置文件中定义的所使用的组件（插件）来决定。组件分为系统组件和可选组件，其中系统组件是必需的基本组件。除了 3S 提供的这些组件之外，用户还可以使用自动化平台工具包（Automation Platform tool-kit）来创建自定义的组件。
- ❒ 版本管理：在 CoDeSys 中可以同时安装一个组件的多个版本，并且可以组合使用这些版本，编译器也可以安装和使用多个版本；而且无需更新整个版本，就可以新增独立的功能。
- ❒ 工程的组织方式也同样采用了面向对象的方法：
  在 CoDeSys 工程中，包含了由各种编程对象组成的PLC 程序对象，还包含了在目标系统（硬件设备）上运行 PLC 程序时，需要的“资源”对象。

由上所述，在一个工程中有两类主要的对象：

1、编程对象（POUs）：

编程对象 POU 包括程序、函数、功能块、方法、接口、动作、数据类型定义等。

在“POU 窗口”中管理的编程对象，在整个工程范围内都有效，可以被工程中所有的“应用”通过任务配置来调用，即实例化。在“设备窗口”中管理的编程对象（即针对特定应用的编程对象），只能被本应用使用，或被本应用的“子应用”实例化后使用。

2、资源对象（设备树）：

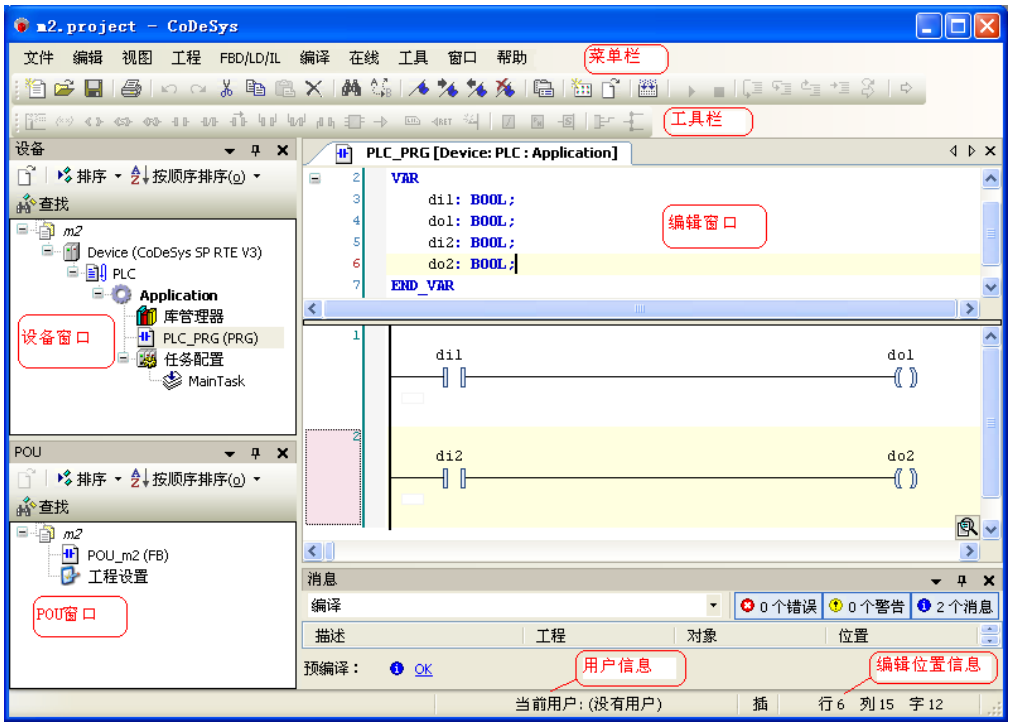
资源对象包括设备对象、应用、任务配置、配方管理等。资源对象只能在设备窗口中进行管理，即只能在设备树中进行管理。在设备树中添加对象后，需要按一定的“规则”与被控设备进行映射。对象（如库和 GVL 等）在工程中的有效范围，会依据设备树中应用和设备对象的层级关系而定，一般来说，一个应用中的对象对其“子应用”也有效，可以被使用。

- ❒ 由集成的编译器生成代码，并使用机器码以加快执行时间。
- ❒ 和控制器设备之间的传输数据：在 CoDeSys 与目标设备之间，通过 gateway 组件和实时系统进行数据的传输。提供了完善的在线功能对设备程序进行实时监控。
- ❒ 预定义的特性设置能够在有精简特性和低复杂度的“标准”用户接口和支持所有特性的“专业”环

境之间进行选择。编程系统在首次安装到系统上启动时，你可根据提示选择一种设置。但是之后仍然可以切换设置，也能对当前使用的特性设置进行自定义。标准版本和专业版本的区别请查看特性选择的描述。

1.2 CODESYS 的结构

CODESYS 编程用户界面由“组件”排列组成，参考下图：



1.2.1 工程的构成

一个工程包含了 PLC 程序中的所有对象，以工程文件命名存储。工程中包含下列对象：POU、数据类型、可视化界面、资源及库。

POU

POU 是指“程序组织单元”，可以是程序、功能块或函数类型，它们可以随时增补。

每一个 POU 都包含一个局部和主题声明定义，主体部分可以用 IEC 的语言来编写，这些语言包括指令列表（IL），结构化文本（ST），顺序功能图（SFC），功能模块图（FBD），梯形图（LD）或连续功能图表（CFC）。

CODESYS 支持所有 IEC 标准的 POUs，如果在工程文件中使用这些 POU，必须在工程文件中包含标准库文件 standard.lib。POUs 可以调用其它的 POUs，但递归调用是不允许的。

函数

一个函数是一个 POU，它正确地产生一个数据元素（可以包含若干元素，比如，字段或结构体），在处理过程中，可以作为文本化语言表达式中的一个操作数来调用它。在声明一个函数的时候，一定要给它一个类型，这就是说，在函数名后面加上一个冒号然后跟一个数据类型。

一个正确的函数声明可以参考下面的例子：

FUNCTION Fct: INT

另外，必须分配给函数一个结果，即把函数名作为一个输出变量。

函数的声明从关键字 FUNCTION 开始。

在 IL 中函数的调用被安排在单步操作或单个转换之内。在 ST 中一个函数的调用可以作为表达式中的操作数。下面是一个在 IL 中一个函数带有三个输入变量并返回前两个变量的乘积与第三个变量相除的结果。

功能块

一个功能块是一个 POU，在程序中提供一个或多个值。与函数相反，一个功能块没有返回值。功能块的声明用关键字 FUNCTION_BLOCK 开始，可以创建功能块的备份或实例。

程序

一个程序是一个 POU，它在操作过程中返回几个值，程序在工程文件中是全局的。程序的所有最终值将保留直到下一次程序运行。

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“Project（工程）”→“Object Add（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。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 POU PLC_PRG(假如没有使用任务配置)，PLC_PRG 是一个单任务程序中的主程序。

动作

动作能够被定义并分配给功能块和程序，动作表现为一个更进一步的执行，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，动作使用相同的输入/输出变量和局部变量来执行。

资源

你需要用资源来配置和组织的工程文件和追踪变量的值。

- 全局变量用于全部工程文件或网络中。
- 库管理器用于添加库文件到工程中。
- 日志文件记录在线期间的动作。
- PLC Configuration 配置可编程控制器的硬件。
- 任务配置通过任务划分来引导程序的工作。
- 监视和接收管理器来显示变量值和设置默认变量值。
- 目标系统设置用来选择和在必要时进行目标系统的最终配置。
- 工作区作为工程选项的映像。

在 CODESYS 中构建工程需要的目标系统和目标系统设置，也可能用到下列资源：

- 用于变量值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参数管理器。
- 作为控制器监视的 PLC 浏览器。
- 工具，与目标系统相关，在 CODESYS 内外调用外部工具程序。

库文件

可以在工程文件中包含一系列的库文件，可以象使用自定义的变量一样使用库文件的 POU，数据类型，和全局变量。库文件中的 standard.lib 和 util.lib 是可以自由调用的程序单元。

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类型。

可视化界面

CODESYS 提供了可视化界面，因此可以显示工程的变量，通过可视化的帮助可以在离线的情况下绘制几何图形，在联机模式下能够按照特定变量的值而改变他们的形状，颜色和文本输出。

可视化的界面可以用作带 CODESYS 的 HMI 的 PLC 实用操作接口，或者作为一个网页或目标系统显示，通过因特网或 PLC 直接可视化。

1.2.2 语言

CODESYS 提供了对应的编辑器，支持所有在 IEC_61131 标准中规定的编程语言：

- 文本形式的语言：指令表（IL）、结构化文本（ST）
- 图形化的语言：顺序功能图表（SFC）、功能模块图（FBD）、梯形图（LD）、基于功能模块图的连续功能图表（CFC）

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。

在指令列表中将用到下面的操作符和限定符：

限定符：

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。

N 用于其它情况：取操作数的反（不包括累加器）。

下面是操作符和它们可能的限定符以及相关的意义：

操作符及限定符意义	
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	当前的值为 TRUE 时，把布尔型操作数置为 FALSE

AND N，（	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 （	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	

单击这里可以得到所有 IEC 操作符的列表。

例子：使用限定符编写的程序

LD TRUE (* 把 TRUE 加载到累加器中*)

ANDN BOOL1 (* 执行 AND 和 BOOL1 变量的取反后“与” *)

JMPC mark (* 当上面的结果为 TRUE 时，跳转到标号“mark”处*)

LDN BOOL2 (* 保存 BOOL2 的反 *)

ST ERG (* 把 BOOL2 保存在 ERG*)

Lable:

LD BOOL2 (* 保存 BOOL2 的值 *)

ST ERG (*把 BOOL2 保存在 ERG*)

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；此时操作数 5 计算 MUL。

结构化文本（ST）

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）。

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、函数调用或其它表达式。

表达式的计算

依照一定的规则来处理操作数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。相同约束力的运算符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)	最强的约束力
函数调用	函数名（参数列表）	
求幂	EXPT	
取反	-	
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	
布尔运算“或”	OR 最弱的约束力	

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。

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

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

	END_IF；
CASE	CASE INT1 OF 1：BOOL1： = TRUE； 2：BOOL2：= TRUE； ELSE BOOL1：= FALSE； BOOL2：= FALSE； END_CASE；

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

WHILE	J：=1 ； WHILE J<= 100 AND ARR[J] <> 70 DO J： =J+2； END_WHILE；
REPEAT	J：=-1 ； REPEAT J：=J+2； UNTIL J= 101 OR ARR[J] = 70 END_REPEAT；
EXIT	EXIT ；
空指令	；

赋值操作符

赋值符号左边是一个操作数（变量，地址），“：=”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。

在结构化文本中调用功能块

通过写功能块的实例名，以及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。

RETURN 指令

返回指令可以用来按照条件离开一个 POU（ POU）。

IF 指令

IF 指令可以检验一个条件，根据这个条件，执行指令。

语法：

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。

CASE 指令

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判断的指令。句式：

CASE <Var1> OF

<Value1>: <Instruction 1>

Languages...

2-14 CoDeSys V2.3

<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 不是所指定的值，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;

FOR 循环

通过 FOR 循环程序可以编写重复执行的处理过程。

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。

WHILE 循环

WHILE 循环可以象FOR 循环那样使用，不同之处在于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 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。

因此在此类情况下，可以用这两种循环。如果循环数比较明确，那么因为FOR 没有死循环反而更好应用。

REPEAT 循环

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；

EXIT 指令

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。

顺序功能图表（SFC）

顺序功能图表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，以及采用过度转换变量来控制处理的顺序。

步（Step）

用顺序功能图编写的 POU 包含了一系列的步，这些步之间是通过定向连接（转换条件）实现的。

有两种类型的步：

• 简单类型：每步包括一个动作和一个标记，这个标记用来表示此步是否激活。如果单步动作正在执行，那么在步的右上角方向会出现一个小三角形。

• IEC 类型：每步包含一个标记和一个或多个赋值的动作或布尔变量。相关的动作出现在步的右边。

动作（Action）

一个动作可以包含一系列的指令表或结构化文本指令，功能模块图或梯形图许多的网络，或者又包含另外顺序功能图。

在简单步中，动作经常是和步连接在一起的，为了能编辑一个动作，在步上双击鼠标或选择此步，再选择菜单命令“Extras”“Zoom Action / Transition”。另外，每一个步中只允许一个输入或输出动作。

IEC 步的动作是附加在顺序功能图- POU 内的对象管理器中，通过双击或者在它的编辑器中按 Enter 键可以加载它。也可以通过“Project”“Add Action”来创建一个新的动作。可以为一个 IEC 步分配最多九个动作。

进入和退出动作

可以额外的为一个步添加一个进入和退出的动作，在一个步激活后，一个进入动作只能执行一次。退出动作只在步失效之前执行一次。

带进入动作的步左下角一个“E”来表示，退出动作用右下角的“X”表示。

转换/转换条件

在步和步之间有所谓的转换。

转换条件的值必须是TRUE 或 FALSE，因而它可以是一个布尔变量、布尔地址或布尔常量。在结构化文本句式（例如（ I<=100 ） AND b） 或者在任何一种期望的语言（ 参照‘Extras’‘Zoom Action/Transition’）中，它也能包括一系列有布尔结果的指令。转换中不能包括程序、功能块或赋值。

<注意> 除了转换外，也能用渐进模式跳到下一步，查看 SFCtip 和 SFCtipmode。

激活步

在调用顺序功能图的 POU 后，将首先执行初始化步的动作（被一个双边线包围）。正在执行的步动作，状态是激活的，在联机模式下，激活的步显示为蓝色。在一个控制循环中激活步的所有动作都将执行。所以，当激活步之后的转换条件是 TRUE 时，它之后的步被激活。当前激活的步将在下个循环中再执行。

<注意> 如果激活的步包含一个输出动作，譬如它下面转换条件是 TRUE，那么它只能在下个循环过程中执行。

IEC 步

在顺序功能图中可以使用标准的 IEC 步。

为了能使用 IEC 步，必须在工程文件中联接 Iesfc.lib 库文件。

一个 IEC 步中不能分配超过九个动作，IEC 的动作不象简单步那样固定地作为输入或输出到某个步的动作，而是和步分开存储并且能够在一个 POU 中重复使用多次。因此，它们必须用命令“Extras Associate action”和单个步联系在一起。

除了动作，布尔变量也能分配给步。

能够使用所谓的限定词来控制激活和未激活的动作和布尔变量。可能有时间延迟，如果一个动作依然激活中，而下一个步已经开始处理了，通过限定词 S（设置），可以取得并发的过程。

随着每一个顺序功能模块的调用，相关联的布尔变量被设置或复位，也就是说，随着每一次调用，这个值将在 TRUE 到 FALSE 之间来回变化。

IEC 步的关联动作在步右边的两长方形中表示，左边的区域包含了限定词，可能带有时间常量，右边的区域包含了动作名和各自的布尔变量名。

限定符

为了关联动作和 IEC 步，用到下面的限定词

N	非存储	动作和步一起激活
R	复位	动作是未激活的
S	设置	动作被激活，再复位前保持激活状态
L	时间限制	动作激活一段时间，最大值和步激活时间一致
D	时间延迟	如果步仍然激活，动作在一定时间后激活，然后只要步是激活的，它就保持激活
P	脉冲	如果步激活，动作只执行一次。
SD	存储和时间延迟	在一定时间之后动作激活并保持激活状态到下一个复位开始。
DS	延迟和保持	只要步仍然激活并且保持到下一个复位开始，那么在一定时间后动作被激活
SL	保持和时间限制	动作激活并保持一段时间

限定符 L、 D 、 SD 、 DS 和 SL 需要一个 TIME 常量格式的时间值。

<注意> 当一个动作失去激活时，它会再执行一次。这就是说每个动作至少执行两次。

SFC 中的隐含变量

在 SFC 中使用一些隐含声明的变量。

每一个步都有一个标记，标记中存储着步的状态。对于 IEC 步来说，步的标记（激活或未激活）被称为<StepName>.x 或者对一个简单的步来说称为<StepName>。当关联的步激活的时候这个布尔变量值为 TRUE，反之则值为 FALSE。它能够用在 SFC 模块中的每一个动作和转换中。

可以通过查询<ActionName>.x 来查询一个 IEC 步是否激活。

隐含变量<StepName>.t 能够用来查询步激活的时间。

隐含变量也能够被其它程序访问，例如，boolvar1：=sfc1.step1.x; 这里 step1.x 是隐含布尔变量，它代表了 POUsfc1 中的 IEC 步 step1 的状态。

SFC 标志符

标志符用来控制 SFC POU 操作，它在工程运行期间隐含创建。为了能读这些标志符，必须定义合适的全局变量或局部变量。例如，如果在一个 SFC POU 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初始化步的初始化之后，因而，例如 SFCReset 标志符可以在初始化步中被复位到 FALSE。

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 中预先定义 SFCError，同时注册任何超时设定的标志符。

SFCPause:当这个布尔变量值为 TRUE 时，SFC 图表就停止执行。

SFCError:当在 SFC 图表中有超时，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，这些状态将不会记录。如果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 SFCError。

SFCTrans:当一个转换被激活时，这个布尔变量得到真值。

SFCErrorStep:这个变量是一个字符串变量，如果 SFCError 中注册了一个超时，这个变量将存储这个超时步的名字。前提条件是在 SFC 中已定义了注册任何超时的标志符 SFCError。

SFCErrorPOU:这个字符串变量包含了发生超时的模块名字。前提条件是在 SFC 中已定义了注册任何超时的标志符 SFCError。

SFCCurrentStep:这个字符串变量存储了那个被激活步的名字，它与时间监控无关。在仿真的情况下，此步存储在外部权限的分支中。如果一个超时发生，超时将不再注册，而且SFCError 也不会复位。

SFCErrorAnalyzationTable:这是数组型变量 ARRAY [0..n]，它提供一个转换表达式的分析结果。对有影响的每个元素转换注册为 FALSE， 以及上一步超时记录， 所以要把下列信息写进 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 POU 显式使用。

上一步的超时注册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, SFCTipMode: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 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。

可选分支

在 SFC 中可以定义两个或两个以上的可选择分支。每一个分支的开始和结束必须带有一个转换。可选分支可以包含并行的分支和其它的选择分支，一个可选分支开始于一个水平线并终止于一个水平线（选择结束），或是一个跳跃。

如果在可选分支开始行前面的步是激活的，每一个可选分支的首次变换将从左到右被计算，最先的转换将从左边转换条件为 TRUE 的开始，然后下面的步被激活。

并行分支

在 SFC 中可以定义两个或两个以上的分支为并行分支。每一个并行分支在开始和结束处必须有一个步。并行分支可以包含可选择的分支或其它并行分支，一个并行分支开始于一个双划线，结束于一个双划线或者一个跳跃，它能提供一个跳跃标识。

如果并行分支的先前步是激活的，并且这个步之后的变换条件值是 TRUE 时，那么并行分支的第一步激活。这些分支彼此并行处理。当所有并行步激活并且这些步之后转换条件为 TRUE 时，那么并行分之末端线后的那一步被激活跳转。

跳转是对在跳转符号下面指明的步名的一个连接。当在不允许创建向上或互相交叉联络的时候，必须使用跳转。

功能模块图（FBD）

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提供算术或逻辑表达式、功能块的调用、跳转或返回指令的结构。

连续功能图表（CFC）

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

梯形图（LD）

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 POU 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。

触点

在梯形图中的每一个网络图的左边都有触点（触点是用两个并行线| |来表示），它用来表示电路的“开”“关”状态。

这些状态与布尔变量 TRUE 和 FALSE 相一致。布尔变量属于每一个触点。如果变量值为 TRUE，那么状态可以通过连接线从左传到右边。否则，右边接收到的是“断开”。

触点可以并联使用，其中的一个并联分支必须传递“开”状态时，并联分支才能传递“开”。或者触点串联连接，此时，触点必须传递“开”状态时，最后的触点才传递“开”，这些与串并联连接的电路一致。

线圈

在梯形网络图的右边有一些所谓的线圈，它们用（）表示，并且只能通过水平线来连接。线圈传递从左到右的连接状态，并且复制状态到布尔变量中，可以描述入口线的状态为“开”（对应布尔变量的 TRUE）或者“关闭”状态（对应布尔变量的 FALSE）。

触点和线圈也可以取反值（在上例中的触头 SWITCH1 和线圈%QX3.0 是取反值）。如果触点取反值（在触头符号中用“/”来表示），然后它复制反后的值到相应的布尔型变量中。如果一个触点取反值，仅当相应的布尔变量取到 FALSE 时，电路才能连通。

梯形图中的功能块

可以在网络图中添加功能块和程序，但它们必须具有布尔型值的输入和输出，并且可以象触点那样用在梯形图的左边。

触点的设置/复位

触点可以被定义为设置/复位触点。设置触点在触点符号中用“S”表示，它从不覆盖相应的布尔变量中的 TRUE 值，也就是说，如果变量一旦设置为 TRUE，它将保持状态不变。

复位触点用“R”来表示，它从不覆盖相应布尔变量中 FALSE 值，如果变量已经设置了 FALSE，它将保持这种状态不变。

当使用梯形图时候，可以用触点开关的结果来控制其它的 POU。一方面可以用线圈把结果输出给全局变量，这个全局变量可以在其它的地方使用。也可以通过引入一个带 EN 输入的 POU 来直接在梯形网络图中插入可能的调用。

这些 POU 是完整的正常的操作数、功能、程序或功能块。它们都有一个附加的输入标志符 EN。EN 输入是一个布尔变量，只有当 EN 值为 TRUE 时，带有 EN 输入的 POU 才会被计算。

1.2.3 联机调试功能

采样追踪

采样追踪允许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。CODESYS 允许对 20 个变量进行追踪，每一个变量可以追踪 500 个值。

调试

CODESYS 的调试功能使得错误很容易被找到。为了调试，运行命令“Project”“Options”，在自动弹出的创建选项的对话框中选择已经激活的选项。

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 POU 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。

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可以检查程序中的逻辑错误。

单循环

如果选择了单循环，每一个循环结束，执行也就结束。

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（write value），或者在每一个循环之后重新定义为特定的值（force value）。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 Write Variable xy 对话框，在这里可以编辑变量的真实值。

监视

在联机模式下，所有的变量可以从控制器中读出并及时的显示。可以在定义和程序编辑器中找到这些显示。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。如果要监视功能块的实例中的变量，相应的实例块必须已经打开。

仿真

在仿真模拟过程中，创建的 PLC 程序不在实际的 PLC 中运行，而是在 CODESYS 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 是不能运行在仿真模式的。

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。

1.2.4 标准化

IEC61131-3 是一种国际标准化的 PLC 编程语言。在 CODESYS 中提供的可编程语言符合标准化的要求。根据这个标准，一个程序包含以下元素：

- 结构
- POU
- 全局变量

通用的语言的元素在标识符、地址、类型、注释和常量部分中已经讲述。CODESYS 程序的处理是从一个特殊的 POU（POU）PLC_PRG 开始的，POU PLC_PRG 能调用其它的 POU。

2 编辑器

2.1 CAM 编辑器

CAM 编辑器仅对 CoDeSys SoftMotion 控制器可用，需要预先安装 CoDeSys SoftMotion。

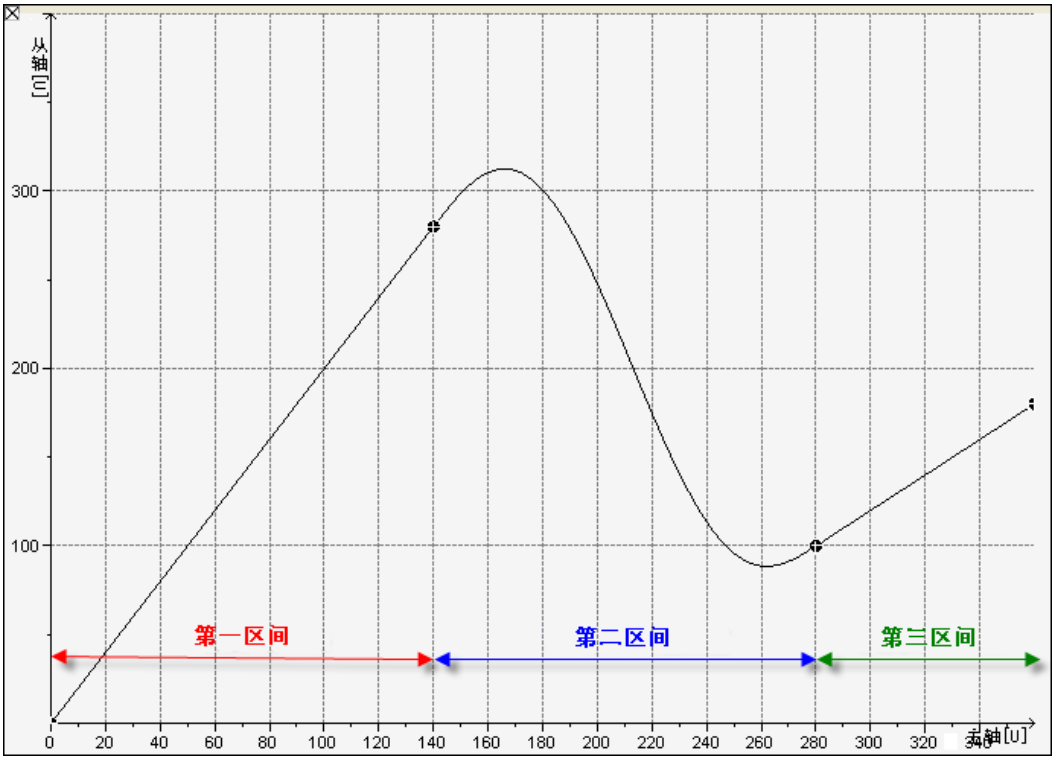
在 CoDeSys 开发环境中集成了 SoftMotion 电子凸轮盘。在电子凸轮编辑器中可以通过图形或列表的方式实现电子凸轮盘(或者凸轮开关功能)。当根据相关的应用程序产生代码时，将创建各种全局数据结构 (CAM 数据)，这些数据结构能够被 IEC 程序访问。因此在添加 SoftMotion 驱动时 SM3_Basic 库将会被自动包含。

SoftMotion 电子凸轮盘的定义

电子凸轮盘规定了主从轴参数间的功能依赖关系。这个依赖关系可以通过主从轴值之间的数值映射关系反应出来。更准确的讲，对主轴细分若干个区间，在每个区间中，从轴与主轴的映射关系，由直线或者五次多项式曲线（5 阶可导）构成。

示例：电子凸轮盘图形

一个电子凸轮盘图形：横轴代表主轴，纵轴代表从轴。



例如，主轴的数值范围从 0 到 360。将这个范围被划分为三个区间：

- 左边的第一个区间： [0, 140]
- 中间的第二个区间： [140, 280]

- 右边的第三个区间： [280, 360]

在第一个和第三个区间中，主从轴的映射关系是线性的，因此就用一条直线表示。由此而知，其一阶导数（即梯度）为常数，二阶导数等于零。

在第二个区间中的主从轴的映射关系为一个五阶可导的多项式。因此，其一阶导数是四阶可导的多项式，其二阶导数（即曲率）是三阶可导的多项式，其三阶导数是二阶可导的多项式（以此类推……）。

以主轴位置为参考，是从轴的运动与主轴的位置构成该映射关系，其一阶导数即为从轴相对于主轴运动的速度，其二阶导数即为从轴相对于主轴运动的加速度。

采用上述物理分析和解释，可以明显的得出映射必须连续的结论，即从图形上看不能有突变。特别是，两个相邻区间的图形需要光滑连接，并且其上面的每一个点都不能突变。此外，通常一阶导数及二阶导数也需要连续。（事实上，该举例中，要满足三重连续性要求，就决定了在两段直线之间的区间，其起点和终点间必须是唯一的插入一条五阶可导的多项式曲线。）

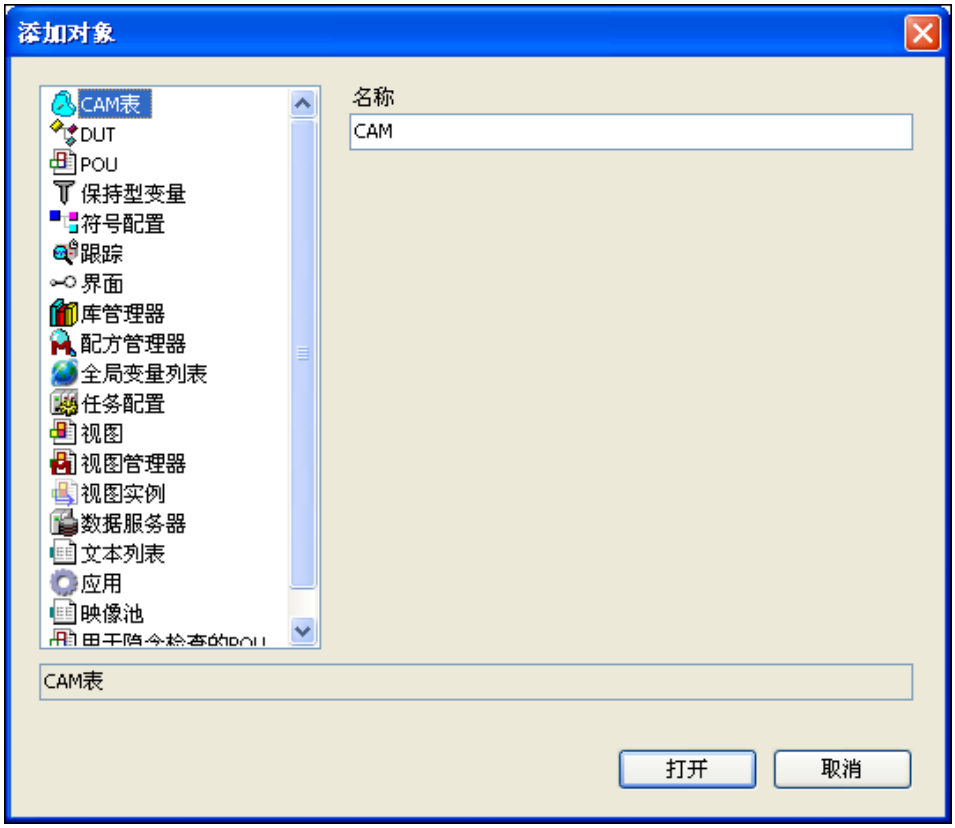
另外，如果电子凸轮盘定义为周期变化，映射值和从轴终点的一阶及二阶导数必须符合其起点的相应值。（在终点的映射值必须符合在起点值的倍数）。

此外，凸轮挺杆,即二进制开关，可以被添加到电子凸轮盘的任意位置。特别是，可以创建一种只实现开关控制的电子凸轮表，方法是将整个主轴的行程内对应的从轴位置值设为零，而表中仅含有凸轮挺杆（开关）信息。

创建电子凸轮盘

电子凸轮盘作为一个对象可以被添加到工程或应用程序中。因此选择工程（可以在设备窗口中，也可以在 POU 窗口中）或应用程序，激活菜单中的“添加对象……”菜单项，将会弹出“添加对象”对话框。

添加对象对话框



在窗口中点击“电子凸轮表”选项，输入名称（默认为“CAM”），点击“打开”确定。电子凸轮盘被插入

到设备树（或 POU 树）中，并且电子凸轮编辑器会打开这个新创建并采用默认设置方式的电子凸轮盘。

编辑电子凸轮

电子凸轮编辑器由四部分组成，其中前两部分在创建电子凸轮时会被自动打开：

- 工具箱中提供了各种选择及插入点工具。
- 主编辑器的标题为电子凸轮的路径名。（例如：CAM [PLCWinNT:Plc Logic:Application]）。该窗口中包含 4 个属性页，通过图形或表格方式对电子凸轮进行设置。
- 在成员属性视图 上点击一下后打开成员特定属性编辑器。
- 对象属性编辑器中包含电子凸轮的一般信息。其可以在设备窗口中，通过点击电子凸轮对象菜单内容中的“属性…”打开，也可以通过类别视图中的“属性… ( ) 命令打开。

工具箱

根据主编辑器中实际上所选的属性页，工具箱中包含如下各种电子凸轮编辑器工具：

- 选择工具总是有效的。通过点击 ，将其激活，在主编辑器中点击鼠标选择点或采用电子凸轮图形描述的映射段，将描述内容记录在表格中。通过图形表示法所选的点或映射段将会加粗显示，被选中的表项为窄蓝色边框，此外其所在的行呈现浅蓝色。
- 只有在属性页的标题为“CAM”时，添加点工具才有效。通过点击 ，将其激活，在主编辑器中的速度或加速度映射图的某个位置上点击鼠标，一个新的点将会被插入到该位置。注意，加加速度图形是无法编辑的。
- 只有在属性页的标题为“Tappets”时，添加凸轮挺杆工具才有效。通过点击 将其激活，在主编辑器中轨迹的某个位置上点击鼠标一个新的凸轮挺杆将会被插入到其相应位置上。

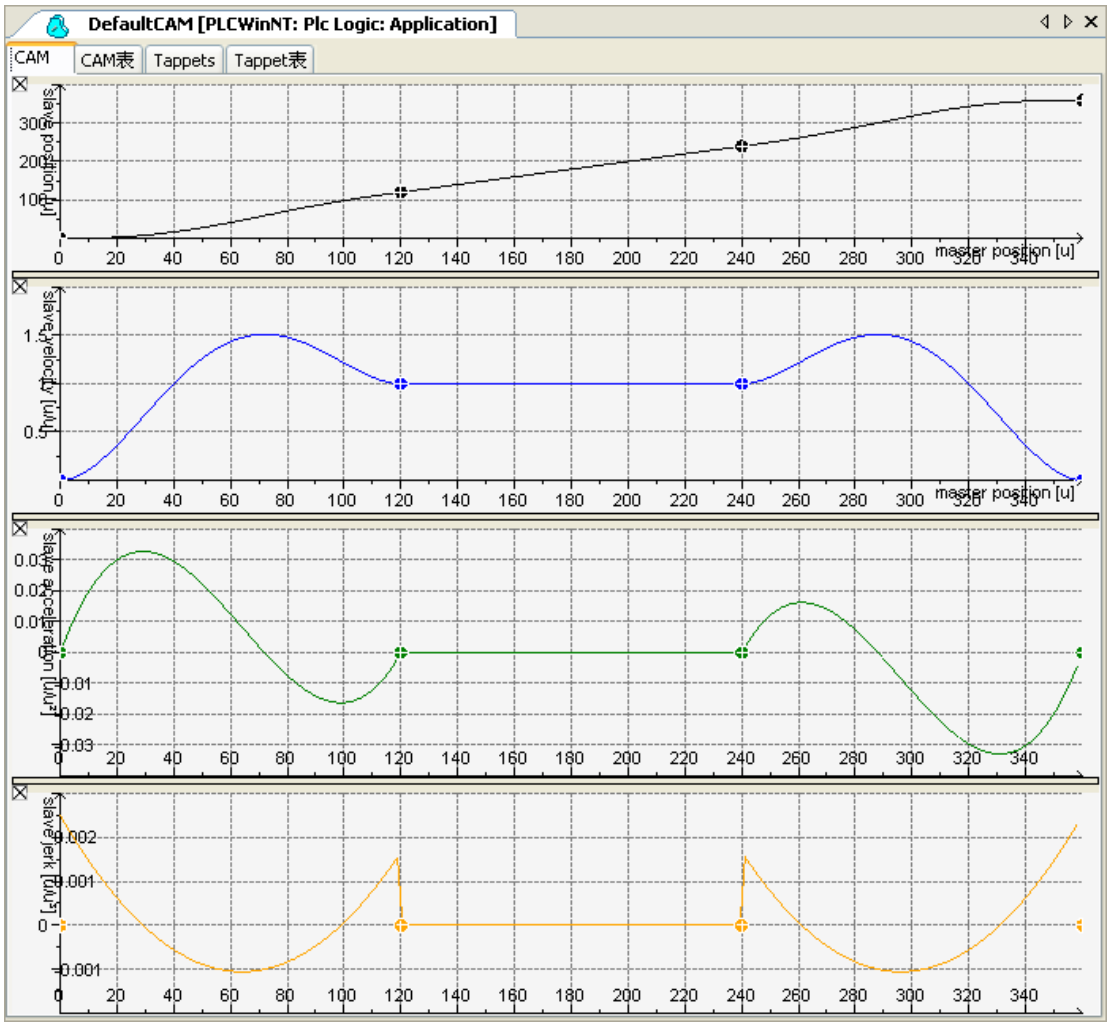
关于主编辑器中各种工具更详细的用法描述如下：

在主编辑器中编辑

在主编辑器中包含四个不同的属性页用来显示映射图及一阶导数图或者采用图形或表格帮助方式添加凸轮挺杆。图形编辑器可以代替表格编辑器进行修正，反之亦然。根据选择不同的属性页来确定相应的编辑模式：

❑ 标题为“CAM”的属性页：用图形编辑器定义一个电子凸轮；该属性页从上到下被分成的四个子窗口中显示的图形依次为：从轴位置图（黑色），从轴速度图（黄色），从轴加速度图（绿色），从轴加加速度图（黄色）。

主编辑器：CAM



通过按下鼠标左键不释放调整子窗口的高度来调整横坐标。也可以通过点击子窗口左上角的 来将其关闭。然后，被关闭的子窗口的位置上将显示一行字符串（位置坐标、速度、加速度或加加速度）。点击相应的字符串将会再次打开对应的子窗口。

在每个子窗口中，坐标系的横轴对应着主轴的范围（默认为[0,360]）。在这个位置关系图中，纵轴的幅值将根据预先在属性编辑器中设定的数值范围来确定，但是纵轴的幅值也会根据对应图表中速度、加速度或加加速度的数值范围来自动进行调整。

除了加加速度图外，其它图形都是可编辑的。正如速度、加速度以及加加速度都是由上级函数求导而得出的，所以对位置、速度、加速度任意一个进行修正，都会影响到另外三个图形的显示。

一个新建的电子凸轮，缺省条件下会用四个点定义出三个区间，这三个区间为[0,120], [120,240]和[240,360]。缺省的映射关系为指定的从轴位置值与主轴位置值相等，速度、加速度及加加速度值为 0，此外第二和第三点的速度被指定为 1。此时，在各区间的映射关系为五次多项式的函数类型。

可以通过下面的途径改变默认的映射方式：

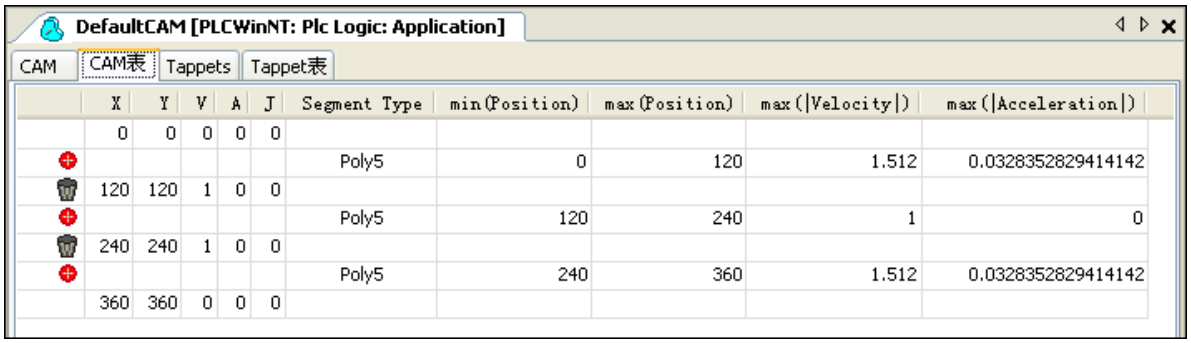
从工具箱中选择添加点工具插入一个远点。在上述三个图形的任何一个的坐标系中通过单击鼠标左键指定其到适当的位置。在包含新点的区间中，依据新点将主轴分成两部分，并在这两个部分分别创建两个新的映射段（用五次多项式类型），并会调整其起点及终点的值。

使用工具箱中的选择工具在对应的图形上选择一个点来更新其位置、速度或加速度，在此过程中保持按下鼠标左键且不释放。另外，也可以使用成员特殊属性编辑器 实现上述操作。

可以通过删除键将选中的点删除（当然被删除的点不能是主轴的起点或终点）。

❑ 标题为“CAM 表”的属性页：采用表格编辑器定义一个电子凸轮：前面的五列包含相关点的属性（即横坐标 X 和纵坐标的位置 Y，速度 V，加速度 A 和加加速度 J）。接下来为映射段的五个属性（区间类型，从轴位置极值（最小值,最大值），以及在相关区间内的速度和加速度的最大值）。保留表格中第一行的值作为主轴的起始位置（以及相关的从轴数值），最后一行作为终点位置。奇数行（最少一行）以交替的顺序确定映射段或点。

主编辑器：电子凸轮表



	X	Y	V	A	J	Segment Type	min(Position)	max(Position)	max(|Velocity|)	max(|Acceleration|)
	0	0	0	0	0					
⊕						Poly5	0	120	1.512	0.0328352829414142
🗑	120	120	1	0	0					
⊕						Poly5	120	240	1	0
🗑	240	240	1	0	0					
⊕						Poly5	240	360	1.512	0.0328352829414142
	360	360	0	0	0					

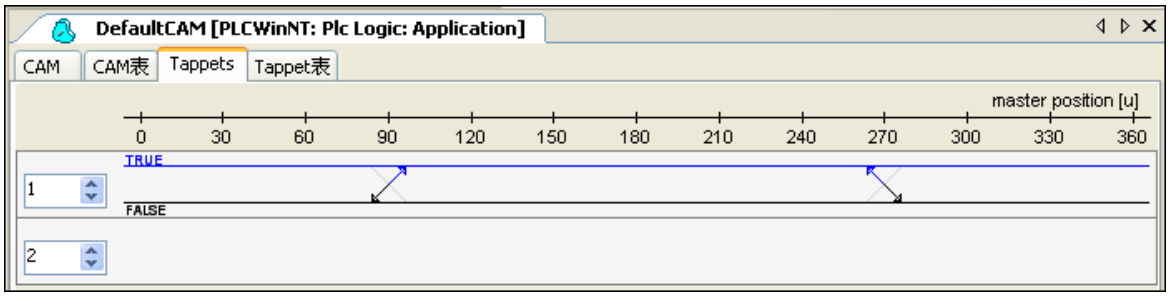
在一个已经存在的区间中，通过点击映射段属性行前的⊕符号添加一个点。在此之后将插入两行新内容，第一行内容是相应区间的中点的入口值，第二行内容是下一步映射段的属性。无论整个区间添加前的段类型如何，添加后的两部分均被默认设置为五次多项式类型。

通过点击鼠标来更新选中的表项。注意，不能编辑最后 4 列中设置的相关映射段的最大值。选中后，通过键盘输出入，或再次点击进入表项，从下拉列表中选择适当的内容来修改所选表项的段类型。必须注意曲线连续性的要求。否则错误的修改将不会被应用，或者被自动根据其他表中数据修正。

通过点点击删除的点所在行前面的🗑符号可以删除该点（只要该点不是主轴的起点或终点）。该行及随后的一行将会被从表格中移除。区间中左边的点将会被移除，其它部分相应的将会与区间中右边的点相接合。无论之前两个单独的段是何类型，新形成的映射段为五次多项式类型。

❑ 标题为“凸轮挺杆”的属性页：在图形编辑器中定义凸轮挺杆轨迹。凸轮挺杆轨迹中定义了一个或多个具有相同制动开关的由主轴位置坐标决定的凸轮挺杆（即凸轮挺杆事件）。横轴上反应了主轴的行程。可以通过调节右侧箭头 增加或减少 ID 号来控制接下来轨迹的走势（默认情况下有两条轨迹，第一条轨迹已经由两个凸轮挺杆的行为所决定）。

主编辑器：凸轮挺杆



从工具箱中选择添加凸轮挺杆工具来添加一个新的凸轮挺杆。可以在需要的轨迹处单击鼠标左键来指定一个适当的位置，在此处将插入一个新的凸轮挺杆，并且此时选择工具将会代替添加凸轮挺杆工具自动生效。通过点击其中心选中一个凸轮挺杆后，可以通过鼠标拖动来移动其位置（在移动的

过程中保持鼠标处于按下状态）或者通过删除键将其删除。

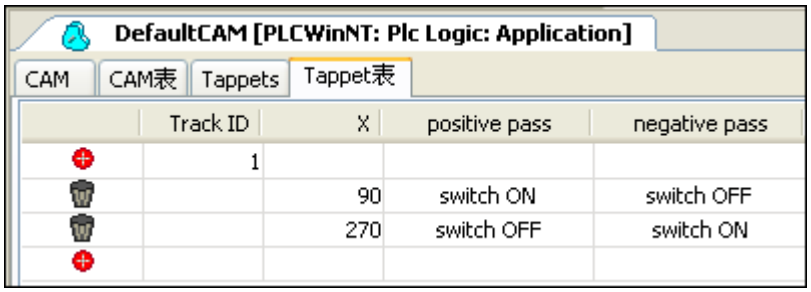
正向（主轴位置值增加的方向）或负向经过凸轮挺杆标记时，可以赋予不同的事件（“打开开关”,“关闭开关”,“转换开关状态”,“无操作”）。请看列表中的各种组合情况。一个轨迹中各凸轮挺杆的各种属性，即相同的 ID 号，指向同一挺杆位，其为布尔型变量。默认情况下，电子凸轮在正转情况下具有“打开开关”的属性，在反转 情况下具有“关闭开关”属性。因此该布尔变量在正转时的值被设置为 TRUE，当反转时被设置为 FALSE。“转换开关状态”事件将会使该布尔变量的值取反，“无操作”则对布尔变量的值无影响。通过单击可以使 中的交叉线打开或关闭，用来改变对应凸轮的各种属性。

凸轮挺杆属性组合表

凸轮挺杆符号	正转	反转
	无操作	无操作
	打开开关	无操作
	关闭开关	无操作
	无操作	打开开关
	无操作	关闭开关
	打开开关	关闭开关
	打开开关	关闭开关
	关闭开关	打开开关
	关闭开关	关闭开关
	转换开关状态	无操作
	无操作	转换开关状态
	打开开关	转换开关状态
	转换开关状态	打开开关
	转换开关状态	关闭开关
	关闭开关	转换开关状态
	转换开关状态	转换开关状态

❑ 标题为“凸轮挺杆表”的属性页：在表格编辑器中可以定义凸轮挺杆轨迹。在显示出轨迹 ID 号的行（行前有⊕符号）下面，该轨迹有多少凸轮挺杆就有多少行显示。 符号之前的这些行中每一行都代表一个凸轮挺杆事件。下表中的各列内容均是有效的：轨迹 ID 号（凸轮挺杆轨迹 ID 号）；X：主轴的相应位置坐标；正转/反转：从正向或反向通过主轴位置坐标的事件定义。

主编辑器：凸轮挺杆表



一个轨迹中的各凸轮挺杆的属性即相同的 ID 号，涉及到相同的从动件，其为布尔型变量。默认情况下，在正转情况下电子凸轮的属性为“打开开关”，在反转情况下电子凸轮的属性为"关闭开关"，因此在正转情况下布尔型变量的值设置为 TRUE，在反转情况下设置为 FALSE。“转换开关状态”事件对该布尔型变量的值取反，但“无操作”事件对布尔型变量的值没有影响。

通过点击轨迹 ID 号前面的⊕符号，可以向一个已经存在的轨迹上添加一个凸轮挺杆。在此，一个新的行将会被插入到紧跟主轴位置 0 之后，其内容为默认属性。

单击选中表项可以修改其内容。然后，可以输入一个新的轨迹 ID 号或者主轴的位置坐标，或者通过进一步点击相应表项从下拉列表中的正转/反转中选择属性。

通过点击表中相应行前面的符号删除凸轮挺杆。

点击表格中的最后一行中的 符号添加一个新的轨迹。在此，⊕将会插入两行新的内容，第一行包含一个尚未分配的轨迹号，第二行提供主轴位置 0 上默认的凸轮挺杆及默认的属性。一旦与轨迹对应的所有的凸轮挺杆全被移除，轨迹本身将会被移除。

在成员属性编辑器中进行编辑

在图形编辑器中，当前选中的电子凸轮或凸轮挺杆的成员属性，可以通过紧邻工具箱窗口的“属性”窗口查看并编辑，其每个默认值均有效。

在电子凸轮对象属性编辑器中进行编辑

在设备树中，当前被选中的电子凸轮对象，其基本设定可以在对象属性对话框中进行定义。

编译电子凸轮

在编译的过程中将创建 MC_CAM_REF 类型的结构体变量。这些结构体变量中包含的每个映射部分都是用来描述该电子凸轮。关于该类数据结构被用来作为 MC_CamIn 模块的输入参数。该结构是 SM3 Basic.library 库的一部分。

2.1.1 使用电子凸轮

SM3_Basic.library 库提供合适的模块来处理电子凸轮，可以通过库管理器的“添加库…”菜单命令将该库插入到工程中。否则，也会在 SoftMotion 驱动添加到设备树中时，自动插入。

下面的内容将会对具体参数（例如周期、偏移等）的含义作一些深入的解释，并说明将一个电子凸轮叠加到另一个电子凸轮上的可能性：

- 模块参数的作用
- 两个电子凸轮间的切换

模块参数的作用

电子凸轮属性（电子凸轮属性对话框）：

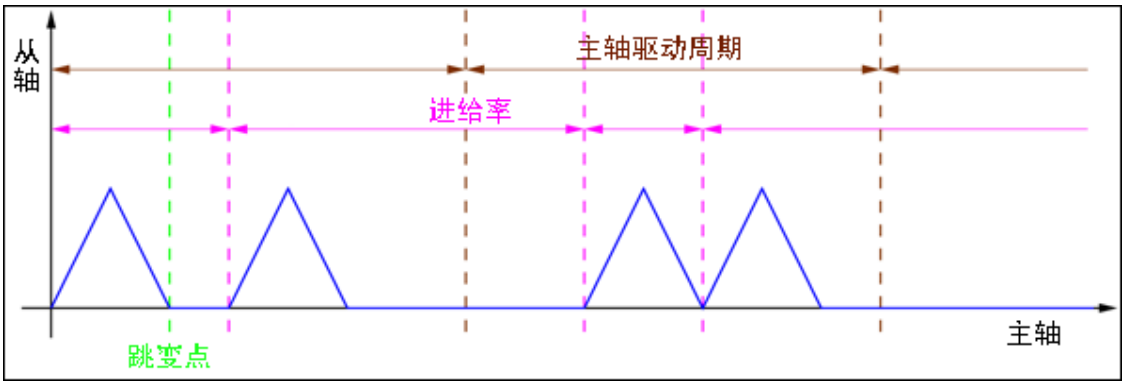
- **周期变化，周期**

一个周期性电子凸轮可以从一个周期到下一个周期重复运行而不出现“跳变”现象。从轴的位置与主轴位置相关，当选项“周期重复”前面的选择框被选定☑，从轴相对于主轴终止位置的位置值与从轴相对于主轴开始的位置值是成倍数关系，但是，从轴的速度和加速度的值还等于他们在起始处的值。周期或进给率用从轴幅值的单位来测算。即使“周期重复”选项没有被选中，相关的电子凸轮也可以连续的工作。在这种情况下，用户必须确保在转换处的动作与预想的一致。

MC_CAMTableSelect 模块：

- **周期的**

这个参数决定了主轴到终止位置后电子凸轮是否要再次进行。下面是参数设为 TURE 时的情形说明。下图显示了周期间的明显差别：这个例子，你可以把它想象成一个传送带传送一些相同物料的应用场景。主轴的值从 0 到所有物料对象的长度。在这个范围内，一个物料对象传送过来，其左后方会一个个的都传送过来，用一个工具（例如，用来给物料打印标记）其位置是由从轴来驱动的（蓝色的图）。



当然，一个电子凸轮主轴的范围（也就是所定义的主轴的整个行程值）并不等于主轴驱动的周期距离（即在例子中，皮带传送一个物料对象的距离）。因此，等式“SlavePosition = CAM（MasterPosition）”，指的是在电子凸轮关系中，从轴位置的定义依赖于主轴的位置，并且只在电子凸轮第一个循环是有效的。当有新的物料对象到来时，电子凸轮驱动设备开始一个新的循环。由于皮带上的物料对象之间有间距（并且间距距离可能还不同），以及生产速度的影响，也就是说在连续的两个电子凸轮之间的间隔时间不同，因此每个循环的电子凸轮主轴范围并不一致。如果参数设置为 FASLE，主轴到达终止位置时，模块 CamIn 的输出变量 EndOfProfile 将置为 TRUE，并且从轴将固定停止在它的当前实际位置。注意，即使主轴超出定义的范围，电子凸轮的动作也不会结束。当主轴重新进入有效范围时，从轴仍会根据电子凸轮进行控制。

- **主轴绝对位置**

如果 MasterAbsolute 输入为 TRUE，电子凸轮将在当前主轴位置处开始，该点也可能位于电子凸轮过程的中间部分。但是如果该点在电子凸轮指定的范围之外，将会弹出一个错误信息。如果输入为 FALSE，电子凸轮运动将在当前位置处开始，也就是说，电子凸轮会以当前主轴所在的位置做为电子凸轮主轴运动所设定的零点。电子凸轮的主轴范围如果不包括“0”，就不能在该模式下使用，

否则将会弹出一个错误信息（“…主轴超出指定范围…”）。

- **从轴绝对位置**

参数 CamTableSelect.SlaveAbsolute 影响从轴的启动模式（StartMode）（启动模式主要由参数 CamIn.StartMode 控制）。两个参数对启动模式的作用如下表所示：

CamIn.StartMode CamTableSelect.SlaveAbsolute	absolute	relative	ramp_in	ramp_in_pos	ramp_in_neg
TRUE	absolute	relative	ramp_in absolute	ramp_in_pos absolute	ramp_in_neg absolute
FALSE	relative	relative	ramp_in relative	ramp_in_pos relative	ramp_in_neg relative

模式的详细描述参见 MC_Camin.StartMode.

MC_CamIn 模块

- **主轴偏移，主轴幅值**

输入量 MasterOffset 和 MasterScaling 按照下面公式变换主轴位置，并且电子凸轮会用变换后的位置 X 进行计算：X = MasterScaling*MasterPosition + MasterOffset

因此，如果 MasterScaling 的值大于 1，电子凸轮将在较高速度下运行；如果值小于 1，电子凸轮将在较低速度运行。

- **从轴偏移，从轴幅值**

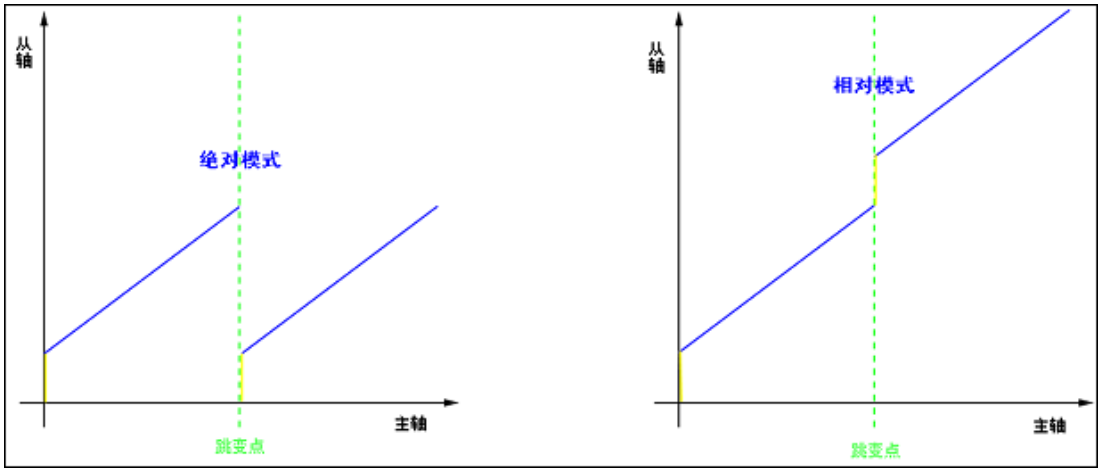
SlaveOffset 参数会使电子凸轮纵向（从轴方向）移动，SlaveScaling 参数则会在从轴方向拉伸电子凸轮。根据下述公式说明可知，第一步电子凸轮先进行拉伸动作，然后再进行移动：

Y = SlaveScaling*CAM(X) + SlaveOffset

A SlaveScaling >1 ，拉伸电子凸轮，从轴运动范围增大；同理，SlaveScaling<1，缩小电子凸轮从轴运动范围。

- **启动模式：绝对/相对/斜坡输入/正向斜坡输入/反向斜坡输入**

下面的图片展示是在 “绝对”与“相对”模式下对一个典型电子凸轮的不同作用。该电子凸轮定义在主轴范围为：从左侧纵坐标开始到右侧绿色垂直虚线之间的范围。因为我们关心的是周期运动的电子凸轮，因此在主轴位置超出范围后，它会继续重复运行。



绝对模式：在新的电子凸轮循环开始时，电子凸轮的计算与当前从轴的位置无关。如果从轴相对于主轴起始位置不同于从轴相对于主轴的终止位置，其将造成一个跳变。

相对模式：新的电子凸轮会根据当前从轴的位置进行改变；也就是说，从轴在上一个电子凸轮循环结束的位置，会被现在的电子凸轮运动作为“从轴偏移”进行位置相加的计算。但是，如果在电子凸轮定义中，与主轴起始位置对应的从轴位置不是 0，其将造成一个跳变。

斜坡输入：通过增加一个补偿运动（根据极限值 VelocityDiff、加速度、减速度得出的运动）来防止电子凸轮在开始时的潜在跳变。因此，只要从轴是旋转的方式，正向斜坡输入选项则只进行正向补偿，而反向斜坡输入则只进行反向补偿。对于线性运动的从轴，补偿方向可以自动实现，也就是说，正向斜坡输入和反向斜坡输入可以被用斜坡输入的方式进行解释。

电子凸轮间的切换

基本上，两个电子凸轮在任何时候都可以切换，但是需要考虑一些情况：

在电子凸轮编辑器中，从轴的位置定义为电子凸轮函数的计算输出，电子凸轮函数是以在主轴范围内的一个主轴位置为计算条件，由此可以用下述简单的公式来进行表达说明：

SlavePosition = CAM(MasterPosition)

因为主轴驱动的实际周期一般与电子凸轮定义的主轴范围是不同的，所以为满足电子凸轮函数的正确输入，主轴位置必须被按比例调整到函数定义域内：

SlavePosition = CAM(MasterScale*MasterPosition + MasterOffset)

类似的情况，如果一个电子凸轮在绝对值模式下启动，产生了一个向上的跳变，函数输出（也就是虚拟从轴位置）也会按比例被修正： SlavePosition = SlaveScale*CAM(MasterPosition) + SlaveOffset

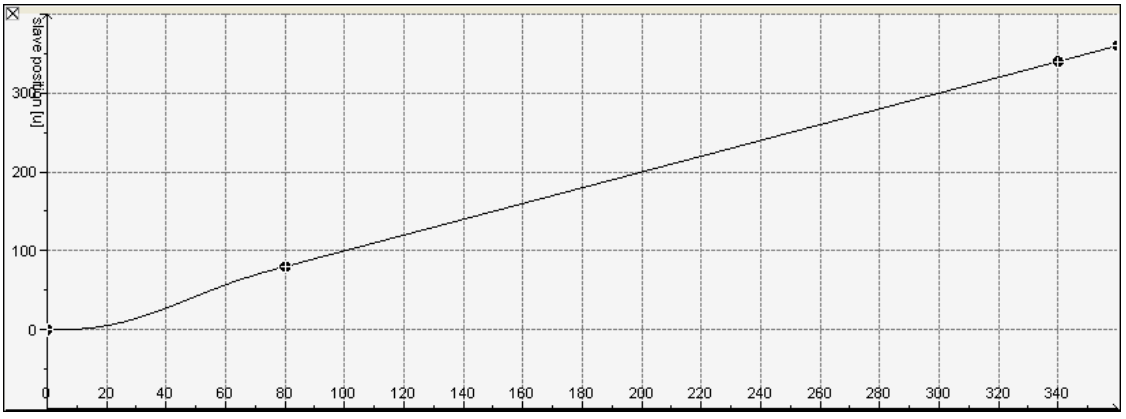
最坏情况下，这两种比例修正都必须被采用，所以，事实上从轴位置（SlavePosition）是由更为复杂的公式计算得出的： Slaveposition = SlaveScale*CAM(MasterScale*Masterposition + MasterOffset) + SlaveOffset

在每个电子凸轮周期结束时，比例和偏移可以改变以获得更为合适的参数。

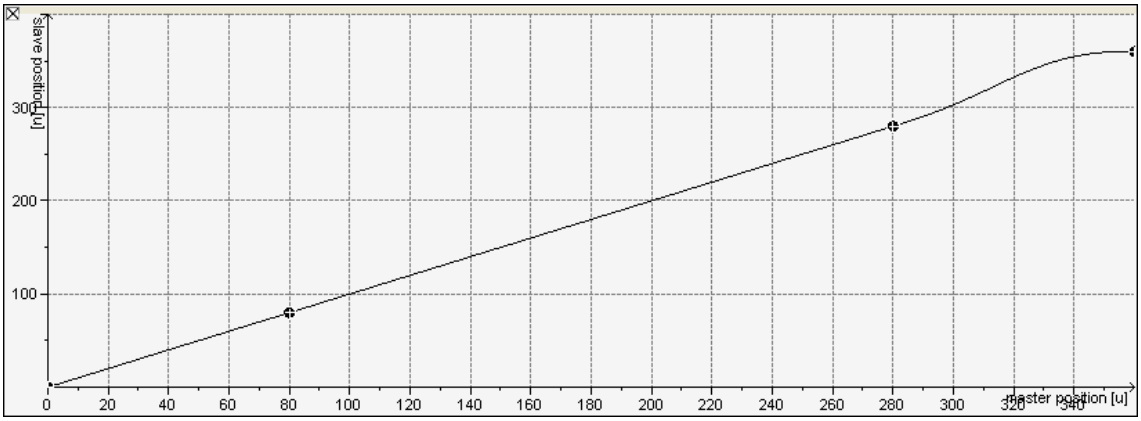
遗憾的是，电子凸轮的 MC_CamIn 模块的重新启动，将会删除它的内存并且包括比例值和偏移值。因此，所定义的电子凸轮函数会适应于一般各种不同的从轴数值。基于这个原因，建议只有在需要处理另外一个不同的电子凸轮时，才去重启 MC_CamIn-FB。

看下面从 CAM1 切换到 CAM2 的例子：

CAM1 是在两条直线之后而形成的五次多项式曲线构成。



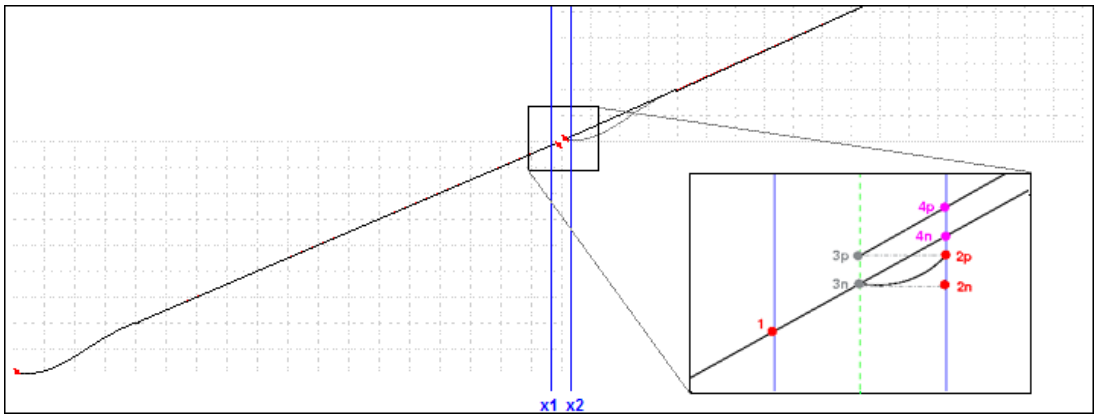
CAM2 是在五次多项式曲线后的两条直线组成。



在两个电子凸轮间进行切换需要考虑下面的问题：

1、为了避免跳变，第一个电子凸轮在结束点处的速度和加速度应该与第二个电子凸轮在开始点处的值对应相匹配。就现在这个例子来看，需要满足的条件为：CAM1 的结束点和 CAM2 的开始点，其速度都应为 1，加速度都应为 0。

2、如果第二个电子凸轮从轴的位置值在开始时被指定为 0，那么应该是在相对模式下启动的。但是第一个电子凸轮必须是在非周期模式下运行。否则，如果第一个电子凸轮为周期性的，则其在完成运行第一个周期时，从轴的位置就不再是唯一确定的：



该图框的右侧展示了由 CAM1 变换到 CAM2 的细节。蓝线表示电子凸轮函数在主轴位置 x1 和 x2 上的计算值。首先让我们考虑一下不顺利的周期情况：

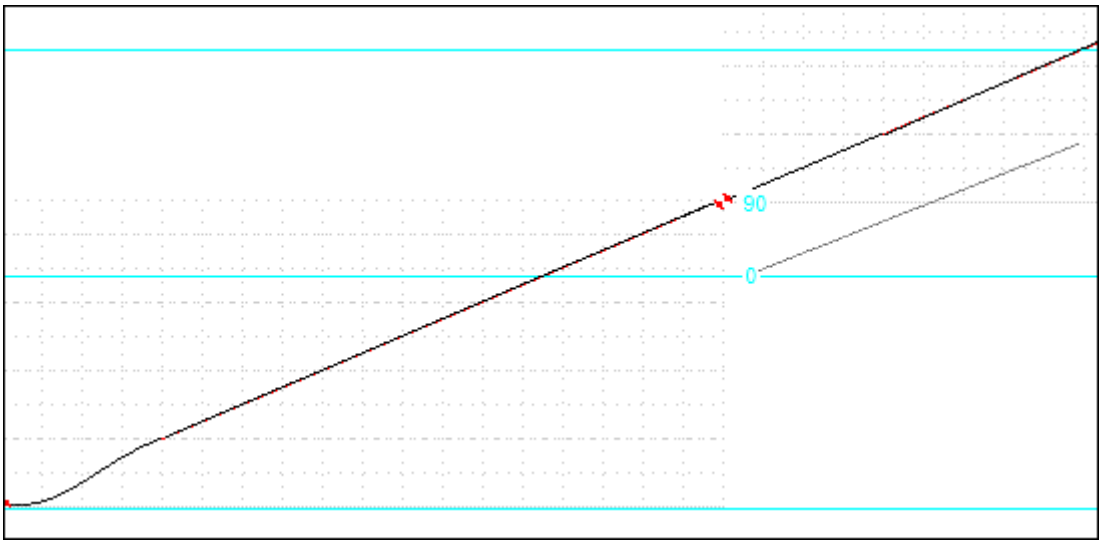
MasterAbsolute := TRUE; SlaveAbsolute := FALSE;	
CAM(x1, CAM1, PERIODIC:=TRUE);	这个调用计算出电子凸轮的主轴位置 x1，它小于 CAM1范围内终点位置的值。所以，用通常的方法，CAM1被计算出从轴的位置结果点为1。
CAM(x2, CAM1, PERIODIC:=TRUE);	当下次再调用这个模块时，计算得出的主轴位置 x2已经超出了 CAM1的终点位置（绿色虚线显示的，等于3p 点的横坐标）。因此，EndOfProfile 标志将被触发。因为 CAM1以周期方式启动，它将在 CAM1范围结束的地方重启，并且返回点2P 做为此次的调用结果。
CAM(EXECUTE:=FALSE);	切换到新的电子凸轮。
CAM(x2, CAM2, PERIODIC:=TRUE);	在主轴位置x2上进行第二次计算。这次以新的电子凸轮CAM2来进行计算。因为 CAM2用相对模式启动，其从轴的实际位置（2p）被作为偏移量添加到 CAM2映射中。所以，它的图形

	起始点移动到了3p 点。它在主轴位置 x2处的计算结果是4p 点，由此，将会得到一个不希望产生的跳变。

选择非周期模式来避免跳变：

MasterAbsolute := TRUE; SlaveAbsolute := FALSE;	
CAM(x1, CAM1, PERIODIC:=FALSE);	这个调用计算出电子凸轮在主轴位置 x1，它小于 CAM1范围内终点位置的值。所以，用通常的方法，CAM1被计算出从轴的位置结果为点1。
CAM(x2, CAM1, PERIODIC:=FALSE);	当下次再调用这个模块时，计算得出的主轴位置 x2已经超出了 CAM1的终点位置（绿色虚线显示的，等于3p 点的横坐标）。因此，EndOfProfile 标志将被触发。因为 CAM1以非周期方式启动，相对于 x2（2n）的从轴位置与相对于 CAM1主轴范围结束点的从轴位置是一致的，即都是点3n。
CAM(EXECUTE:=FALSE);	切换到新的电子凸轮。
CAM(x2, CAM2, PERIODIC:=FALSE);	在主轴位置 x2处进行第二次计算。这次用新电子凸轮 CAM2来进行计算。因为 CAM2用相对模式启动，其从轴的实际位置（2p）被作为偏移量添加到 CAM2映射中。所以，它的图形起始点移动到了3n 点。它在主轴位置 x2处的计算结果是4n 点，其与同一条线上的点1和3n 是吻合的。

3、在绝对模式下启动电子凸轮时需要注意，从轴要有合适的起始位置。不管电子凸轮是以周期模式还是非周期模式启动，如果主轴范围等于从轴的周期，切换起来就没有那么复杂了。 如果两个周期（主轴位置和从轴位置的周期）与 CAM2 主轴值域相一致，等于 360°，上述例子中的 CAM2就可以在绝对模式下启动。否则，如果假如从轴周期是 270°（淡蓝色线条所示），绝对模式选项就不可以，除非进行一些调整。



在 CAM1 与 CAM2 转换时，从轴的位置为 90°的情况下。CAM2 用绝对模式启动，会在 0°造成一个跳变，用灰色线表示。当然，可以通过设定从轴偏移到合适的 90°值时，就可以恰当地处理这种情况。

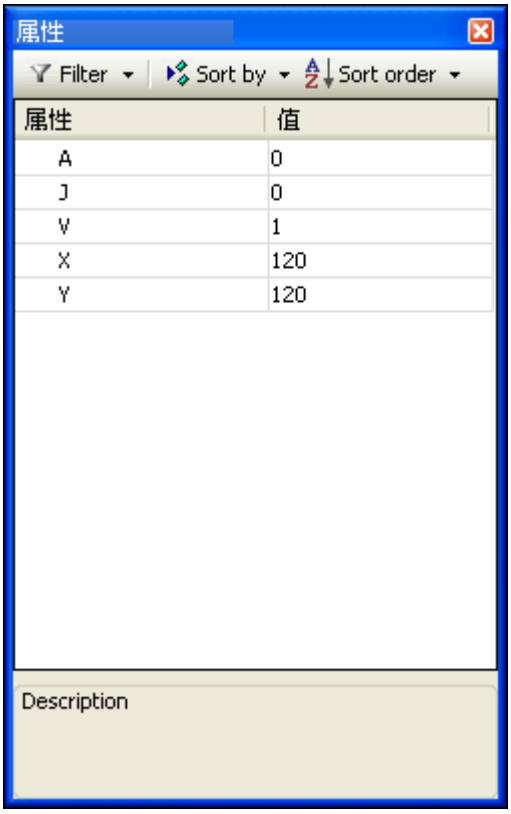
2.1.2 CAM：成员属性

电子凸轮编辑器中使用到的成员属性可以通过“属性”编辑窗体进行查看和修改。

在电子凸轮编辑器中，点击成员属性标志 将会出现属性窗体。如果当前没有成员（点，映射段，挺杆，……）被选中，那么编辑器的工作区为空，同时在窗体下部的“描述”区域中无任何内容。

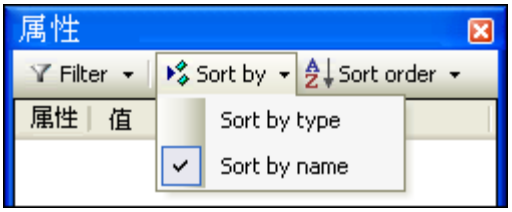
另外，如果在电子凸轮编辑器的绘图选项卡（标题为“CAM”或“Tappets”）中点击一个成员，那么工作区将以表格的方式列出与被选成员相关的所有属性。

示例：显示被选点的属性



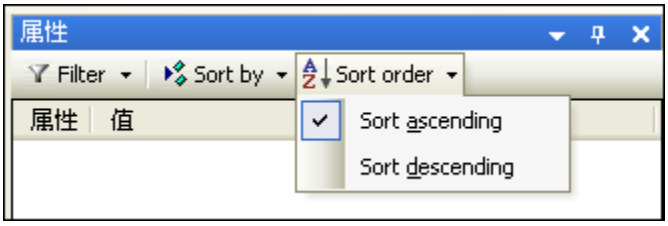
属性编辑器的顶端是一个小工具条。第一项是“过滤”命令。由于目前没有过滤功能，所以与选中成员相关的所有属性都显示出来。第二项是“排序”命令，可以通过选择“按类型”或者“按名字”来进行排序。

工具条：按. . .排序



第三项是关于升序和降序的“排序”命令。

工具条：排序



某个表项被选中时，该项会有一个窄蓝边框，并且它所在的行都会呈现浅蓝色。同时，相关属性的说明信息也会在窗体的“描述”区域显示。

根据相关属性，被选表项的值可以通过键入适当值进行修改；或者通过单击表格单元（table cell），从出现的选择列表中进行选取。在取消选定表格单元后，修改将生效；即单击另外一个表项或空白处，都可使修改生效。

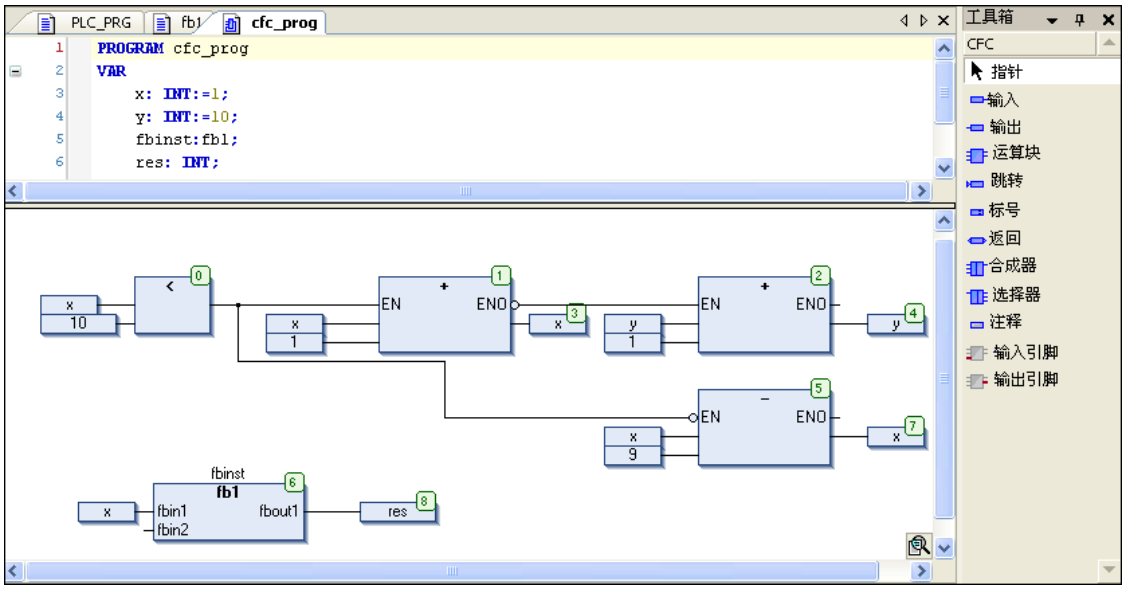
2.2 CFC 编辑器

CFC 编辑器由“CFC 编辑器”插件提供，用于编写 CFC（连续功能图）。CFC 语言是对 IEC 61131-3 编程语言的扩展。点击“工程”菜单下的“添加对象”，可以增加以 CFC 为编程语言的 POU。

CFC 编辑器是一个图形编辑器。

在编写 CFC POU 时，窗口的上半部分是声明编辑器，下半部分是 CFC 编辑器。

CFC 编辑器



与网格编辑器不同，CFC 编辑器允许把元素放在任何位置 ，例如，允许直接插入反馈回路。CFC 编辑器内部有一个链表，包含了所有已经插入的元素，链表的顺序决定了 CFC 元素的执行顺序，但是用户可以改变元素的执行顺序。

工具箱中包含下列元素：运算符（包括操作符、功能、功能块和程序），输入、输出、注释、标签、跳转、编排器和选择器；可以把这些元素插入到 CFC 编辑器中。

拖动鼠标，连接元素的输入和输出，会自动创建最短连接线。当元素移动的时候，连线也会自动调整。

用户可以通过缩放工具改变编辑窗口的尺寸：点击编辑窗口右下角的按钮 ，在打开的菜单中选择一个缩放倍数；还有另外一个方法，在打开的菜单中选择 ... 打开一个对话框，然后可以输入任意缩放倍数。

当 CFC 编辑器处于激活状态时，通过右键菜单或者“CFC”菜单，可以使用 CFC 命令。

CFC-连续功能图-语言

连续功能图是 IEC61131-3 标准编程语言的扩展，是基于功能块图的图形化编程语言，但它没有网络限制，可任意放置元素，例如允许插入反馈回路。

关于如何在 CoDeSys 中添加一个 CFC 编程对象，请参看 CFC 编辑器。

CFC 中的光标位置

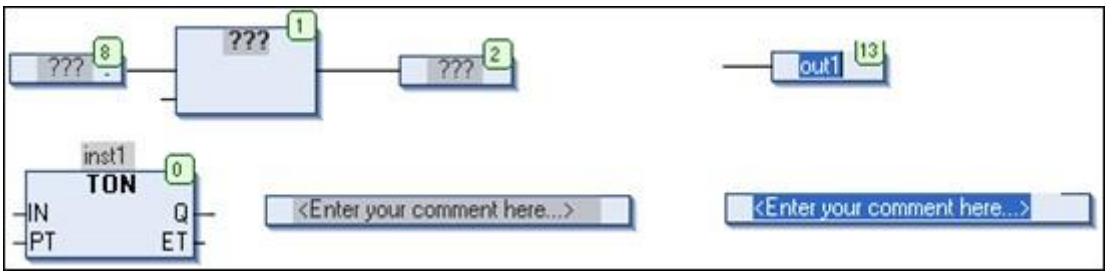
在 CFC 程序中，当光标在元素上移动时，以灰色阴影方式显示的区域可能是光标的位置。

点击其中一个阴影区域，该区域立刻变为红色。当释放鼠标键后，该区域立刻变成当前光标位置，相关元素或文本被选中并且用红色显示。

一共有三种类型光标位置。参看下图的灰色阴影区指示的光标位置。

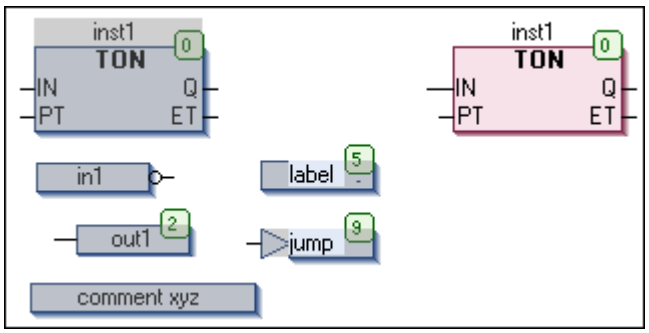
1、当光标在文本时，文本的背景色为蓝色，且可以编辑。点击 ... 按钮打开输入助手。当插入一个元素后，开始显示的是"???"，用于提示用户输入一个有效的标识符。然后，光标置于变量名或运算块参数名位置，并给出运算块参数或变量的类型提示。如果已经被定义，符号注释将显示在第二行。

可能的光标位置以及两个选中文本的例子



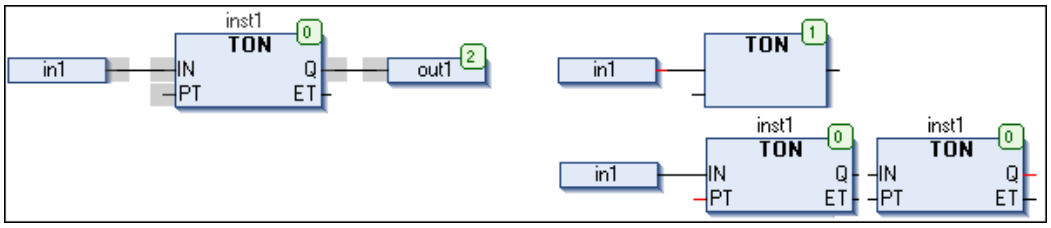
2、当光标在元素（运算块，输入，输出，跳转，标签，返回，注释）上时，该元素显示为红色，且可用鼠标移动。

可能的光标位置以及一个选中元素的例子



3、当光标指在元素的输入或输出连接线上时，连接点变红，可以对该连接线进行取反，复位或置位操作。

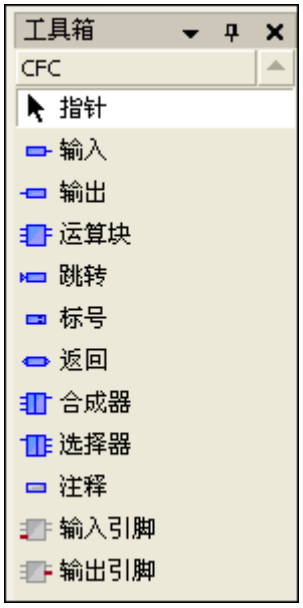
可能的光标位置以及一个选中输出输入位置的例子



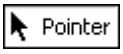
CFC 元素/工具箱

CFC 编辑器插件提供的工具箱中，提供了很多图形元素供用户在 CFC 编辑窗口中编程使用。缺省情况下点击“视图”菜单下“工具箱”，能够打开工具箱。

CFC 工具箱，缺省

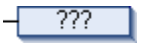
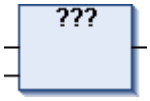
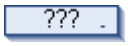
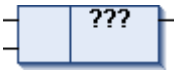


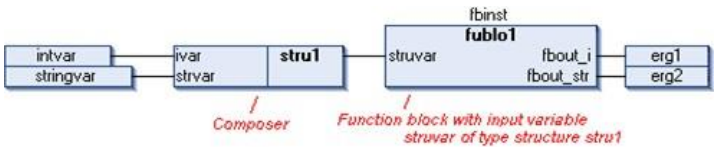
在工具箱中选择所需元素，通过拖&放操作可以插入到编辑窗口中。

缺省情况下，在工具箱列表的顶端有一个指针项 

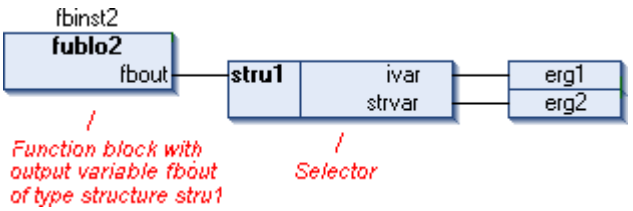
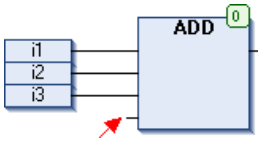
。选择该项后，光标变为箭头状，用户可在编辑窗口中选中元素，并布置、编辑它们。

元素：

输入		选中'???'文本，然后修改为变量或者常量。通过输入助手可以选择输入一个有效标识符。
输出		
运算块		运算块可用来表示操作符，函数，功能块和程序。选中运算块的'???'文本框，修改为一个操作符名，函数名，功能块名或者程序名。通过输入助手可以选择输入一个有效的对象。 在例子中，当插入一个功能块，随即运算块上出现另一个'???'，这时要把"???"修改为功能块实例名。 若运算块被修改为另一个运算块（通过修改运算块名），而且新运算块的最大输入或输出引脚数，或者最小输入或输出引脚数与前者不同。运算块的引脚会自动做相应的调整。若要删除引脚，则首先删除最下面的引脚。
跳转		跳转用来指示程序下一步执行到哪里，这个位置是由标签定义的（见下）。插入一个新标签后，要用标签名替代"???"。
标签		标签标识程序跳转的位置（见上文"跳转"） 在在线模式下，标识 POU 结束的返回标签会自动插入。
返回		注意：在线模式下，return 自动插入到编辑器第一列的最后那个元素之后。在单步调试中，在离开该 POU 之前，会自动跳转到该 return。
编排器		编排器用于结构体类型的运算块输入。编排器会显示结构体的所有成员，以方便编程人员使用它们。使用方法是：先增加一个编排器到编辑器中，修改"???"为要使用的结构体名字，然后连接编排器的输出引脚和运算块的输入引脚。 例如 CFC 程序 cfc_prog 中有一个功能块实例 fubblo1，它的一个输入引脚 struvar 是结构体类型。在这种情况下通过使用编排器，用户可以使用结构体的各个成员： 结构体 stru1的定义: TYPE stru1 : STRUCT ivar:INT; strvar:STRING:='hallo'; END_STRUCT END_TYPE Function block fublo1, declaration and implementation: FUNCTION_BLOCK fublo1 VAR_INPUT struvar:STRU1; END_VAR VAR_OUTPUT fbout_i:INT; fbout_str:STRING; END_VAR VAR

		fbvar:STRING:='world';
		END_VAR
		
		fbout_i:=struvar.ivar+2;
		fbout_str:=CONCAT (struvar.strvar,fbvar);
		Program cfc_prog, declaration and implementation:
		PROGRAM cfc_prog
		VAR
		intvar: INT;
		stringvar: STRING;
		fbinst: fublo1;
		erg1: INT;
		erg2: STRING;
		END_VAR
		 

选择器		选择器用于结构体类型的运算块输出。选择器会显示结构体的所有成员，以方便编程人员使用它们。使用方法是：先增加一个选择器到编辑器中，修改“???”为要使用的结构体名字，然后连接选择器的输出引脚和运算块的输出引脚。 例如 CFC 程序 cfc_prog 中有一个功能块实例 fubblo2，它的一个输出引脚 fbout 是结构体类型。在这种情况下通过使用选择器，用户可以使用结构体的各个成员： 结构体 stru1的定义:TYPE stru1 : STRUCT ivar:INT; strvar:STRING:='hallo'; END_STRUCT END_TYPE Function block fublo1, declaration and implementation: FUNCTION_BLOCK fublo2 VAR_INPUT CONSTANT fbin1:INT; fbin2:DWORD:=24354333; fbin3:STRING:='hallo'; END_VAR VAR_INPUT fbin : INT; END_VAR VAR_OUTPUT fbout : stru1; fbout2:DWORD; END_VAR

		VAR fbvar:INT; fbvar2:STRING; END_VAR Program cfc_prog, declaration and implementation: VAR intvar: INT; stringvar: STRING; fbinst: fublo1; erg1: INT; erg2: STRING; fbinst2: fublo2; END_VAR 
注释		用该元素可以为图表添加注释。选中文本，即可以输入注释。用户可以用<ctrl>+<enter>在注释中换行。
输入引脚		有些运算块可以增加输入引脚。首先在工具箱中选中 Input Pin，然后拖放到在 CFC 编辑器中的算法块上，该运算块就会增加一个输入引脚。
输出引脚		有些运算块可以增加输出引脚。首先在工具箱中选中 Output Pin，然后拖放到在 CFC 编辑器中的算法块上，该运算块就会增加一个输出引脚。

插入、排列元素

缺省情况下，打开 CFC 编辑器的同时，提供 CFC 编程元素 的工具箱也会打开。

CFC 编辑器选项定义了使用 CFC 编辑器时的常规设置。

插入

插入元素方法是：在工具箱中点击鼠标选中元素，用鼠标拖动被选元素至编辑窗口目标位置。拖动时，光标会显示为箭头加矩形框和一个加号； 释放鼠标按键，元素便会插入。

选择

用鼠标点击并选中一个元素、输入/输出引脚或者文本，可以做进一步的操作，比如编辑、重新排列等。按住<Shift>键可继续点击选中多个元素。用户也可以按住鼠标左键，拖动鼠标在欲选择的元素周围画出一个虚线矩形框，释放按键，这些元素即被选中。使用右键菜单中的选中全部命令，可以选中所有元素。

通过箭头键，可以改变选中元素。移动顺序依赖于执行顺序或者编号元素。请见下文。

选中一个输入引脚后，然后按<Ctrl>+<Leftarrow>键，则其对应的输出将被选中。选中一个输出引

脚后，然后按<Ctrl>+<Leftarrow>键，则其对应的输出将被选中。

替换运算块

通过修改运算块的名字就可以替换运算块。由于 POU 的定义不同，POU 输入和输出引脚的个数会自动调整，同时现有的一些赋值可能会被被删除。

移动

鼠标点击选中一个元素（请参看能够选中的光标位置），拖动它到目标位置，然后释放按键，即可以移动元素。也可以通过剪切和粘贴命令移动元素。

连线

拖动鼠标可以建立起输入元素和输出元素间的连线，该连线是最短连线，并且考虑了其它元素和连线的影响。若连接线为浅灰色，则表明元素之间有位置重叠。

复制

选择要复制的元素，然后使用复制 和粘贴命令。

编辑

缺省情况下，插入元素后，文本部分显示的是 “???”。可以鼠标点击选中文本，修改为想要的文本（如 POU 名，标签名，实例名，注释等）。也可以点击按钮 ... 打开输入助手，通过输入助手输入。

删除

可用右键菜单中的删除命令或<Del>键删除已选中的元素。

执行顺序，元素编号

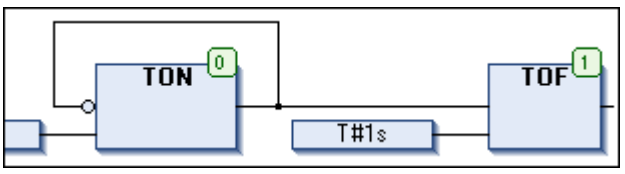
运算块、输出、跳转、返回和标签元素的右上角的数字，显示了在线模式下 CFC 中元素的执行顺序。执行流程从编号为 0 的元素开始。缺省情况下，可以使用 CFC 菜单中的“执行顺序”子菜单下命令中修改执行顺序。

添加一个新元素时，按照拓扑序列（从左到右，从上到下），该元素将自动获得一个编号。新元素得到是它的拓扑后继者的编号，此时若原有序列已被改变，则所有比该编号大的编号都将自动加一。

移动元素时，它的编号仍保持不变。

考虑到执行顺序会影响到结果，在一定情况下可以改变执行顺序。

元素编号的例子



改变工作表的大小

通过改变工作表尺寸，可为编辑窗口中 CFC 元素提供获得更多的空间。选中所有元素并拖动选中元素或通过剪切复制命令（请见上文：移动），可以改变工作域（工作表）大小。

另外，为节省时间，大图表时可以使用特殊尺寸设置对话框。请点击这里 ，查看编辑工作表对话框部分的描述。

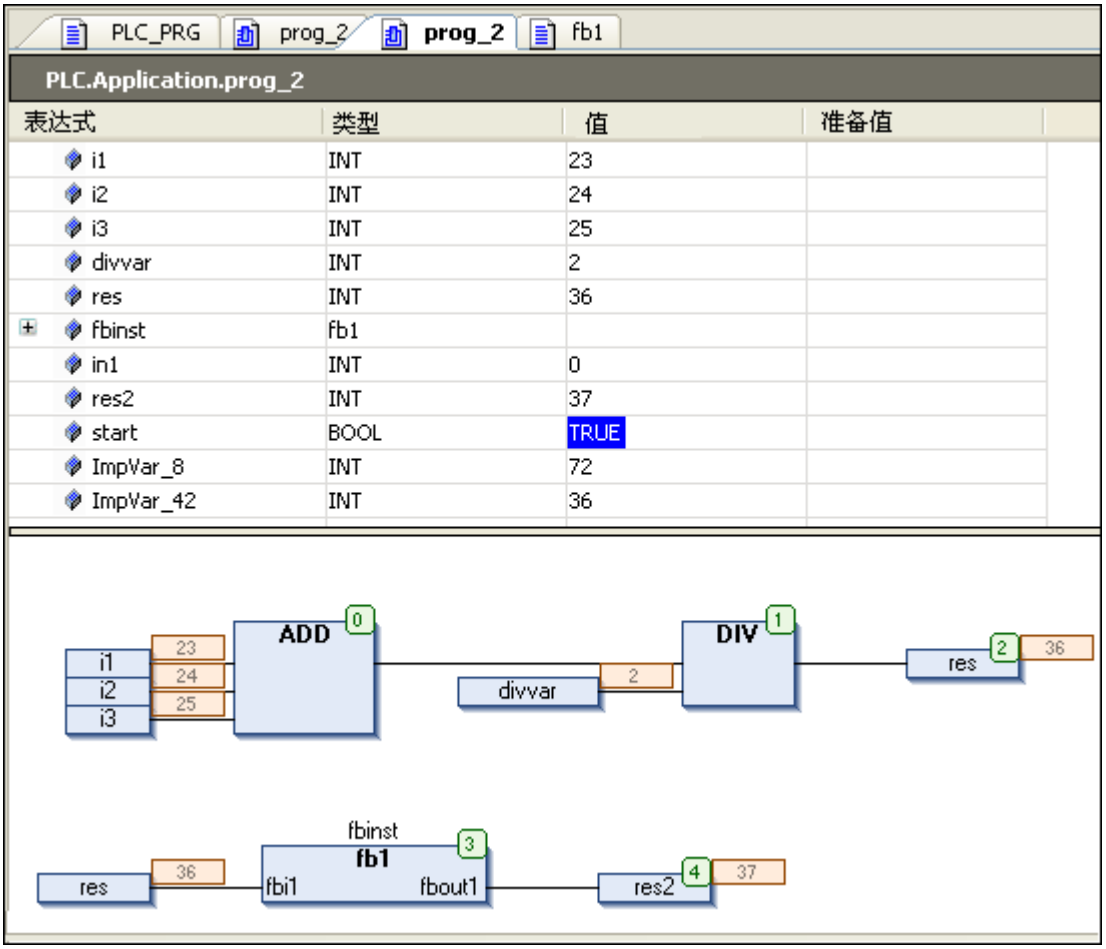
在线模式下 CFC 编辑器

在线模式下，CFC 编辑器可以监视变量，并可以写入/ 强制变量、表达式到控制器。调试功能（断点，单步调试等）也可以使用。

监视

实际值显示在每个变量后面的小监视窗口中（内嵌监视窗口）。

示例：程序对象 PLC_PRG 的在线视图

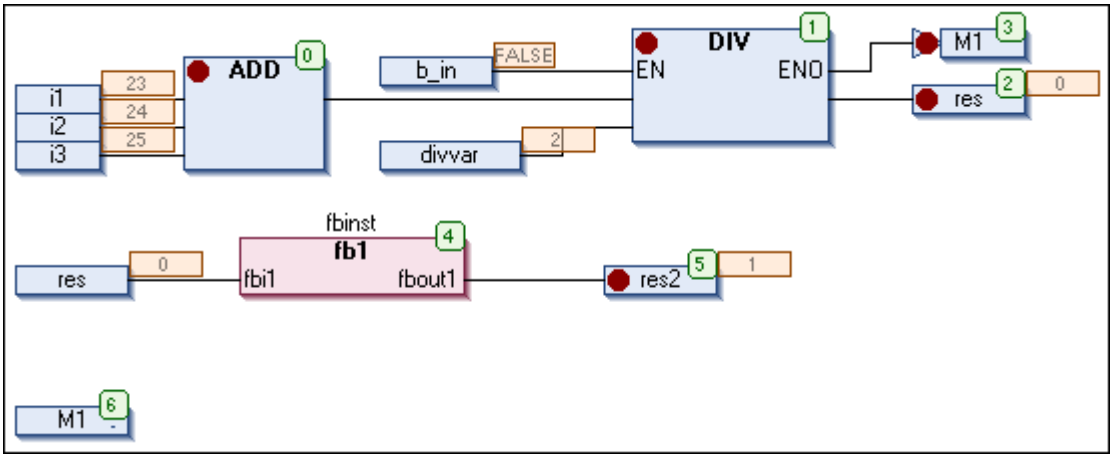


对于功能块 POU 的在线视图：在实现部分，监视窗口不会显示任何值，但是，表达式值栏和内嵌监视窗口中，分别会显示三个问号。

CFC 编辑器的断点位置:

在 POU 中变量值改变、程序流分支或调用其它 POU 的位置，可以设置断点。请看下图的断点位置。

CFC 编辑器中断点的位置



方法中的断点：被调用的所有方法都会自动设置一个断点。因此，如果调用一个接口方法，那么断点可以设置在实现接口的功能块的所有方法和所有派生功能块的该方法调用。如果在功能块中用指针调用一个方法，断点可以设置在功能块的该方法中和所有派生功能块的该方法调用中。

2.3 数据源编辑器

注意：只有被当前使用的特征设置支持时，此功能才有效。

数据服务器可以添加到一个本地应用上，并且用于在应用和远程数据源之间建立一个点对点的连接。通过这种方式，该本地应用可以使用和显示其他设备或 OPC 服务器（该功能尚未实现）提供的数据项（变量）。数据服务器可以对应多个数据源。

注意：数据服务器提供了一种新的数据交换方式，但是不能代替 CoDeSys V2.3 中便已使用的网络变量功能。网络变量是通过用户定义的变量列表和广播方式来进行数据交换的。

注意：视图中的按位访问，只有在与数据服务器连接的情况下，位索引为确定常数时可用（不能是可变的变量，即使是声明的常数变量也不行）。

数据服务器作为对象被添加到应用下的设备树中，每个应用只能配置一个数据服务器。数据服务器被添加之后，对应的数据服务器的库会自动包含到工程中。

受数据服务器控制的数据源也作为对象被添加 到数据服务器下。

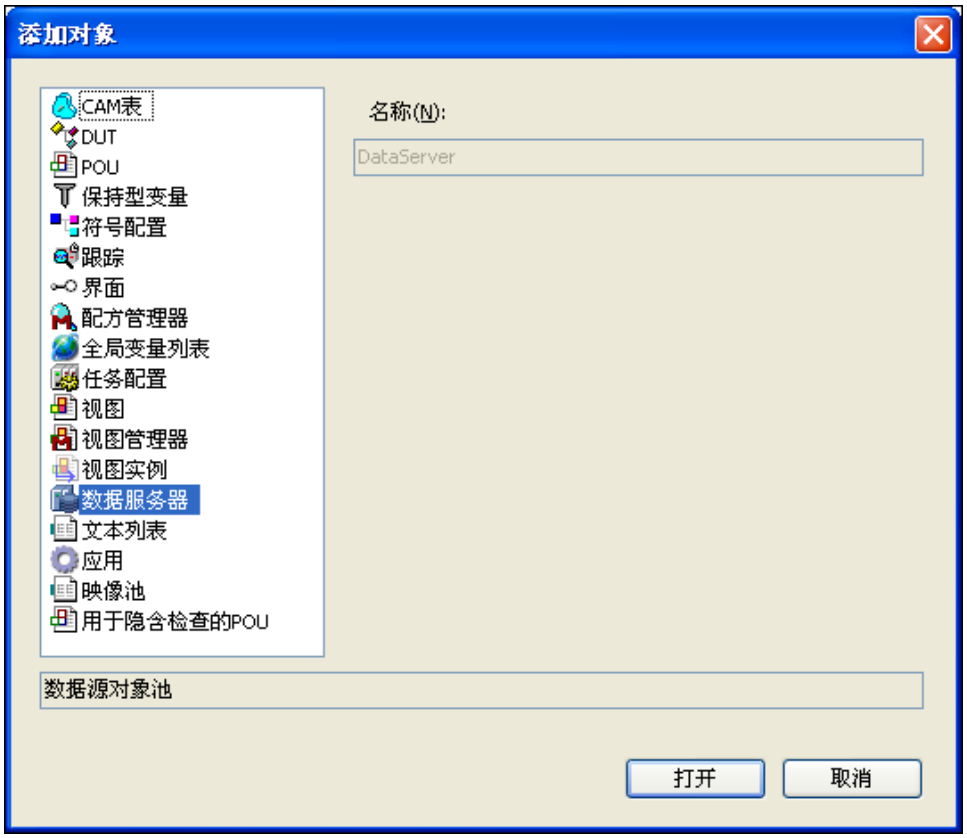
数据源的配置可以在数据源编辑器提供的对话框中进行设置。设置包括：使用哪些变量（数据项）以及它们的存取类型，与数据源之间的通信是怎样建立的，也就是关于数据值的更新。

数据服务器和数据源功能是由“数据源编辑器”和“数据源对象”这两种插件提供的。

2.3.1 添加数据服务器

通过“添加对象”命令，可以将数据服务器这个对象添加到应用中。一个应用只能添加一个数据服务器，且这个对象不能被重命名。

“添加对象”对话框，添加一个数据服务器



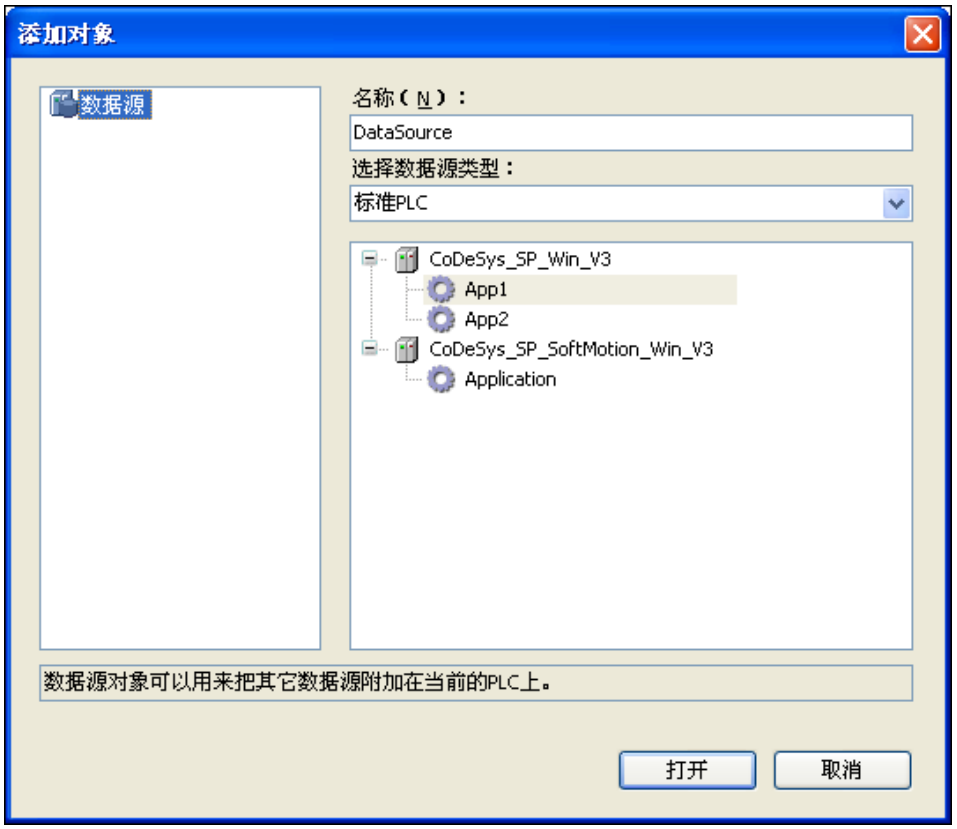
一旦在应用中添加了数据服务器，对应的数据服务器库将自动被添加到库管理器中，数据服务器任务也将被添加到任务配置中。

通过“添加对象”命令，在数据服务器对象下添加数据源对象。

2.3.2 添加数据源

通过“添加对象”命令，可以添加数据源到数据服务器中。

选择数据源添加对象对话框



在“添加对象”对话框中，提供了选择列表供用户选择数据源类型。当前可用的数据源类型有：3S 设备的“标准 PLC”；尚未实现的数据源类型有：OPC 服务器。

一旦用户选择了数据源类型，在选择数据源类型列表下面的窗口中，将会显示该类型中当前可用的数据源。选择所需要的数据源项，并且在名称栏中定义一个名称标识。

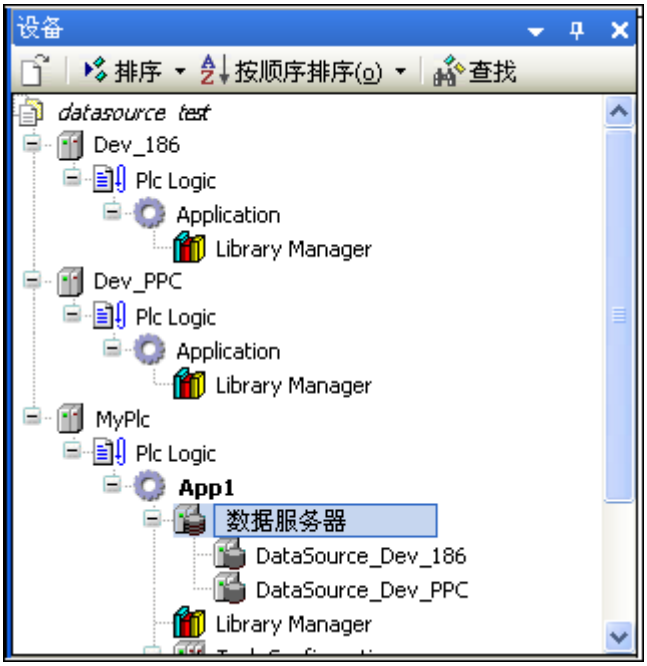
确定数据源后，点击打开按钮，选择的数据源将以定义的名称被添加在本应用下的设备树中。

例子如下：

从设备树中可以看到，当前有三个设备：Device1, Device2, MyPlc。

对于 MyPlc 下的应用 App1，想要使用设备 Device1 and Device2 中的数据，首先需要添加数据服务器和数据服务器下的数据源。在添加对象对话框中选中数据源类型“3S PLC"。 在数据源窗口中将会显示 Device1/Application<xy> and Device2/Application<xy>。选中 Device1 下的应用，输入名称"DataSource_Dev1" ，选中 Device2 下的应用，输入名称"DataSource_Dev2"。确认后，数据源对象将会添加在"App1 下的设备树中，此时，用户可以配置该数据源对象。

举例，数据服务器和数据源



2.3.3 配置数据源

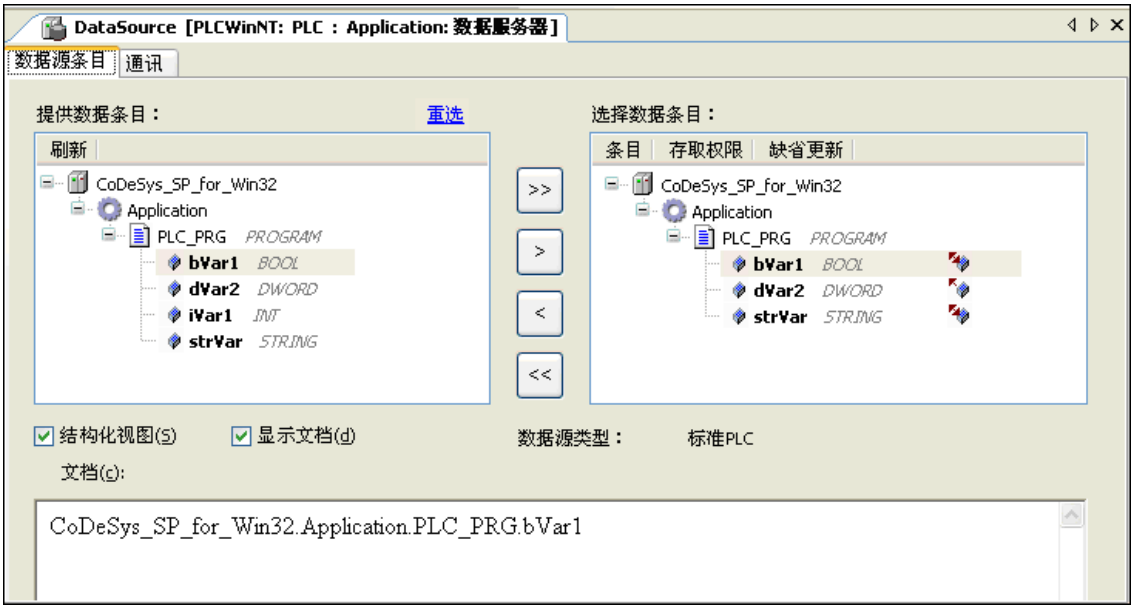
数据源添加到数据服务器后，需要配置数据项以及和真实数据源的通信方式。

通过“编辑对象”命令，或者通过双击数据源对象，可以打开配置对话框。

该对话框分两部分：数据源数据项，通信。

数据源数据项

数据源数据项对话框



在数据源数据项对话框中，可以定义本地应用使用远程数据源的哪些数据项（变量）类型，以及访问权限。

注意：视图中的按位访问，只有在与数据服务器连接的情况下，位索引为确定常数时可用（不能是可变的变量，即使是声明的常数变量也不行）。只有当相关程序的代码在之前已产生，变量才能被加入到列表中。

所选数据源提供的所有变量都在提供的数据项栏中显示，此栏上部的刷新按钮用于更新数据项列表。变量加入列表前需要为相关应用创建代码。

如果激活了结构化视图选项，数据项列表将会以树结构方式显示，否则将以平铺列表显示。所有被数据源应用使用的 POUS 以及使用的变量在“应用”节点下显示，所有实例化的功能块和数据类型以及它们的成员变量将在“数据类型”节点下显示。注意：如果没有激活“结构化视图”选项，则数据类型不会被列出。

激活了显示描述文档选项后，如果数据源提供了对数据项的描述，该描述会在文件栏中显示。

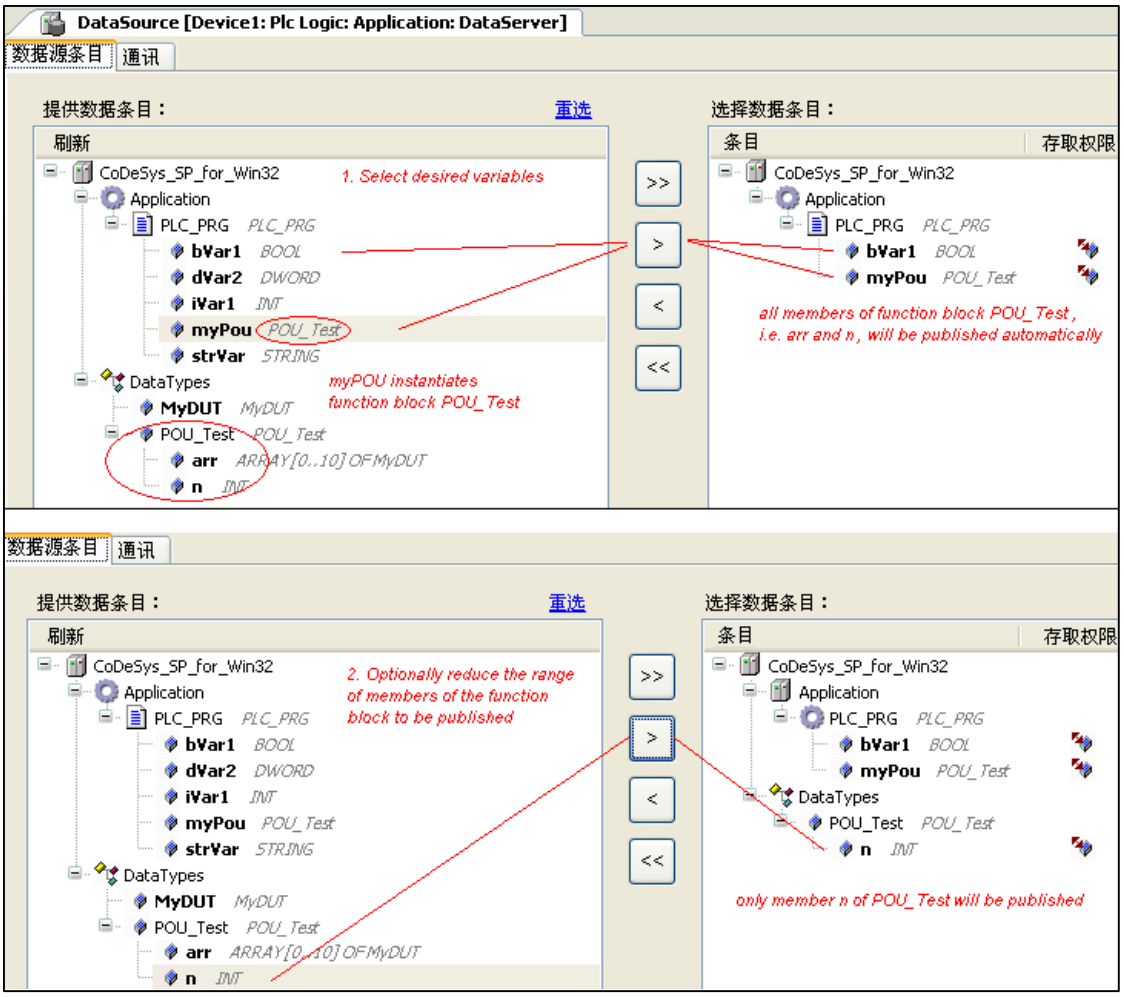
为当前应用选择变量（数据项）：

对于“提供的数据项”栏中的所选的数据项，可以通过双击数据项，或者单击>按钮（添加单个数据项）或单击>>按钮（添加全部数据项）将数据项添加到右边“已选的数据项”栏内。反之，已选择的数据项可通过单击<按钮或<<按钮来撤销添加。

在结构化视图中，对于功能块或数据类型，可以减少成员变量的显示。方法如下：

在“提供的数据项”栏中的“应用”节点下，选择所需变量，如果这些变量中有功能块实例或数据类型，那么这些实例或数据类型的成员变量都会自动显示在本栏的“数据类型”节点下。如果不需要所有的成员变量，可以在“数据类型”节点下选择具体的成员变量添加，那么，最终在“已选的数据项”栏的“数据类型”节点下，将只存在选择的那些成员变量。

点击此处，显示相应的示意图。



该配置选项可直接使用于该应用。

访问权限

在默认设置下：访问权限栏中每个数据项的权限都为“可读写”。如果要修改访问权限为“只读”，点击相应的访问权限栏，进入编辑模式。再次点击访问权限符号可以改变访问权限为“只读”。

“读写”符号：

“只读”符号：

数据项的访问权限的含义如下（需要在默认更新栏中激活同名选项）：

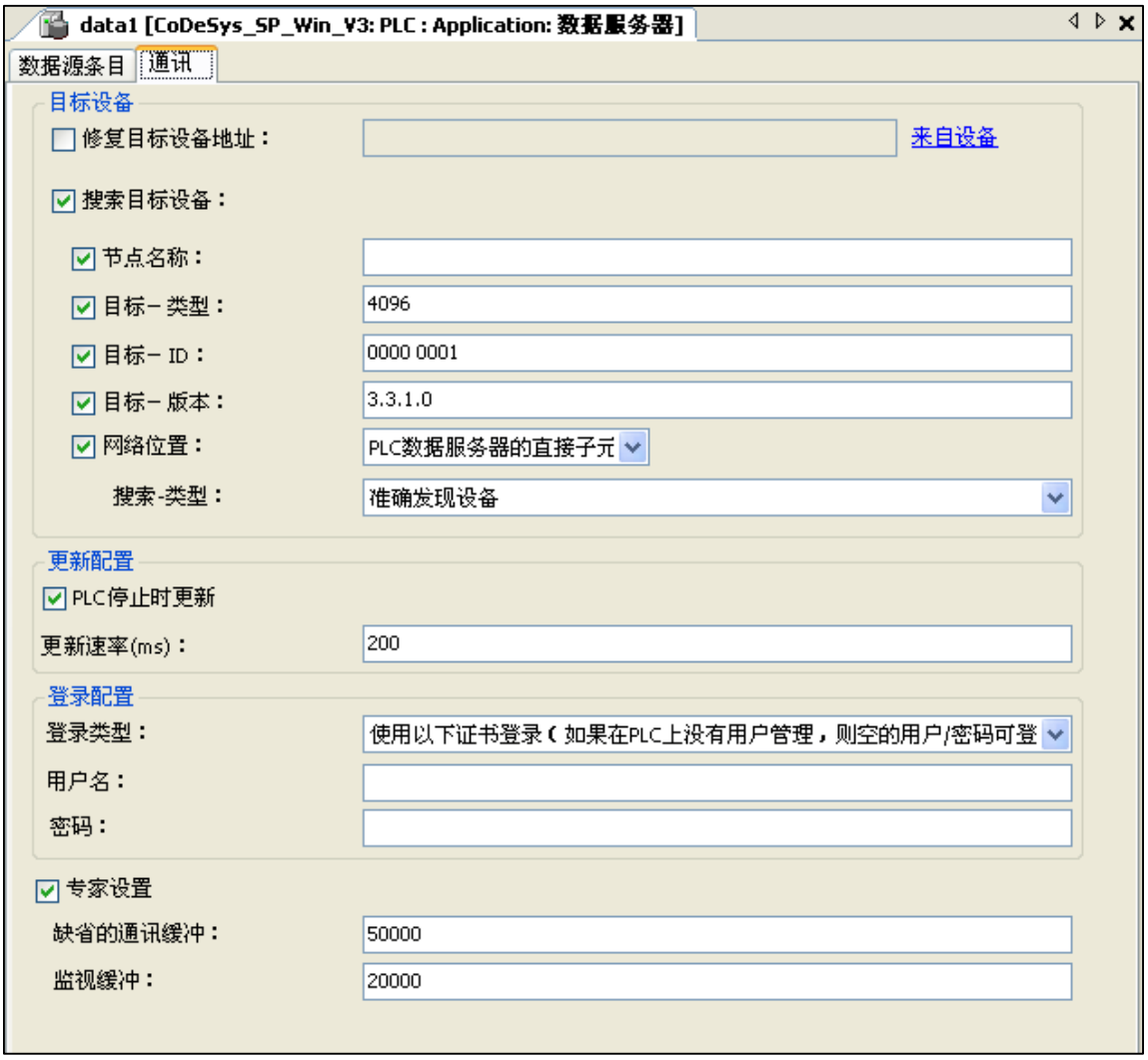
读：如果数据项在数据源中已经改变，那么此项在当前应用也将被更新。

写：如果数据项在当前应用中已经改变，那么此项在数据源中也将被更新。

注意：缺省情况下，已选数据项仅在当前活动视图中被更新。如果在其他应用的 POU 中也使用了该数据项，那么它必须使用数据服务器接口的功能激活。（这种情况请联系软件供应商）。

通信

通信对话框



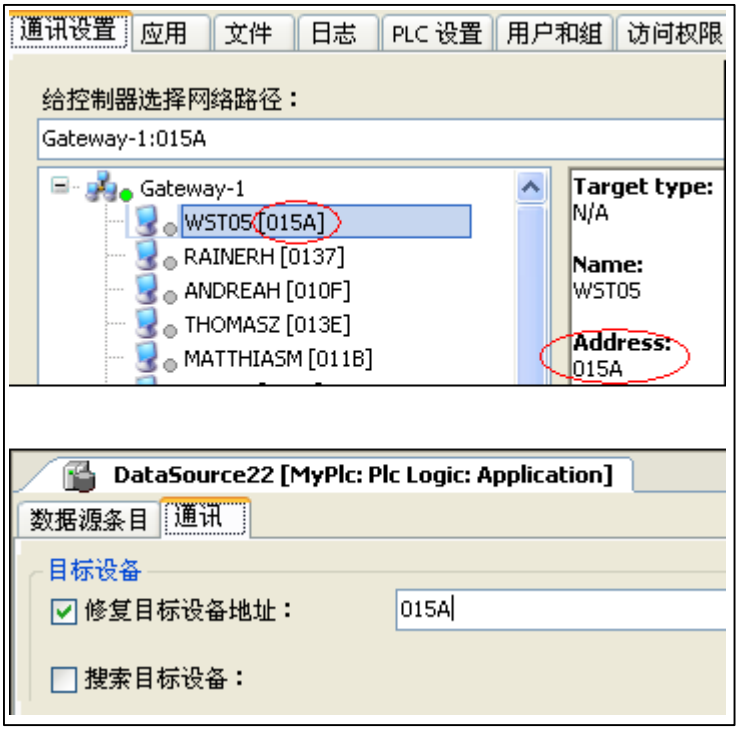
此对话框用于配置与相应远程数据源的通信参数。参考上文的注意。

1、指定数据源设备：可以是一个已定义的固定地址或是一个当前搜索到的可用设备的标准号。

目标设备

鼠标单击从设备获取，将自动输入当前配置的设备的参数，该参数与设备的通信对话框中的配置相一致。

固定目标设备地址：此选项若被激活则通过指定地址进行通信。可以通过手动方式或“从设备获取”自动设置的方式（参见上文）输入所需数据源设备的地址。例如，输入数据源设备地址



搜索目标设备：如果不使用固定目标设备地址，如果可用设备总是需要按照特定的搜索标准搜索得到，那么激活此选项并在选项框中设置一个检查标志激活需要的标准：

节点名：节点名称，例如WST06

目标类型：设备的类型号，例如"4096"

目标标识号：例如"0000 0001"

目标版本：例如"1.0.0.0"

网络位置：在可选列表中选择在哪里寻找目标设备：

指向子数据服务器 plc：在这种情况下，正在运行数据服务器的实时运行系统必须执行一个可从 IEC任务进入的路由组件。在这种情况下，数据服务器所在设备的设备描述文件中必须还要包含编译器宏定义 MPROUTER_AVAILABLE，目的是为了表示路由器库可以被使用。

用地址指向子节点：在右侧所示的编辑区中插入父节点的地址。

指向子数据服务器 plc 或用地址指向子节点：这种选择结合了上述两种方式。需在右侧所示的编辑区中插入父节点的地址，还要注意在第一种选择方式中所述的路由器必备的条件。

搜索类型：从选择列表中选择搜索类型：

查找到的首个设备：使用匹配指定标准的第一个设备。

精确查找设备：使用精确匹配指定标准的设备。

2、设置通信参数

登录配置

登录配置：从选择列表中选择下列选项中的一个，即选择到目标设备的登录应怎样被处理（或者说，哪种凭证是必须的）：

◆ 登录使用下列凭证（空用户/密码也可以，如果在 PLC 上没有帐户管理的话）：您必须在用户和密码域中输入目标设备所要求的凭证，如果目标设备没有帐户管理也可以将这些域留空。

◆ 登录使用在运行时确定的凭证：该选项应被激活，如果用户在运行时通过面板中的登录对话框被询问输入合适的凭证。

◆ 忽略登录到设备：该选项应被激活，如果您正使用一个没有任何帐户管理的老设备，无须任何登录步骤。

更新配置

在 PLC 停止后更新：如果这个选项被激活，数据源项将按照指定的频率被更新，即使当前 PLC 已停止。否则在停止状态时不更新。

更新率（毫秒）： 数据项的更新时间间隔。缺省：200 ms。

专业设置：如果此选项被激活，可以进行以下扩展设置：

默认的通信缓冲区：通信数据缓冲区的字节大小，缺省：50000。

监测缓冲区：监测数据缓冲区的字节大小，缺省：20000。

2.3.4 在视图中使用数据源

如果视图与某应用对应，那么该视图可以使用添加到这个应用的所有数据源。

数据源的数据项（变量）可以在视图的属性配置中使用。可以通过输入助手、智能化“列表组件”功能，或手动输入方式添加。如果使用手动输入方式，应按照语法键入完整的标识路径：<数据源设备名>.<POU 名>.<变量名>

例子如下：

下图是视图元素的属性对话框。变量"iCount"作为文本变量被使用，属于一个已定义的数据源，该变量的数据源设备名为"Device2"，应用名为"App1"，POU 名为"prg1"。

在可视化的元素属性配置中输入数据项：

注意：视图中的按位访问，只有在与数据服务器连接的情况下，位索引为确定常数时可用（不能是可变的变量，即使是声明的常数变量也不行）。

注意：诊断视图—因为它被唯一指定用于检验当前应用，这与目标视图截然相反，它不监视从其他数据源通过数据服务器提供的变量的当前值！因为将显示的那些仅是初始值。

注意：缺省情况下，已选数据项仅在当前活动视图中被更新。如果在其他应用的 POU 中也使用了该数据项，那么它必须使用数据服务器接口的功能激活。

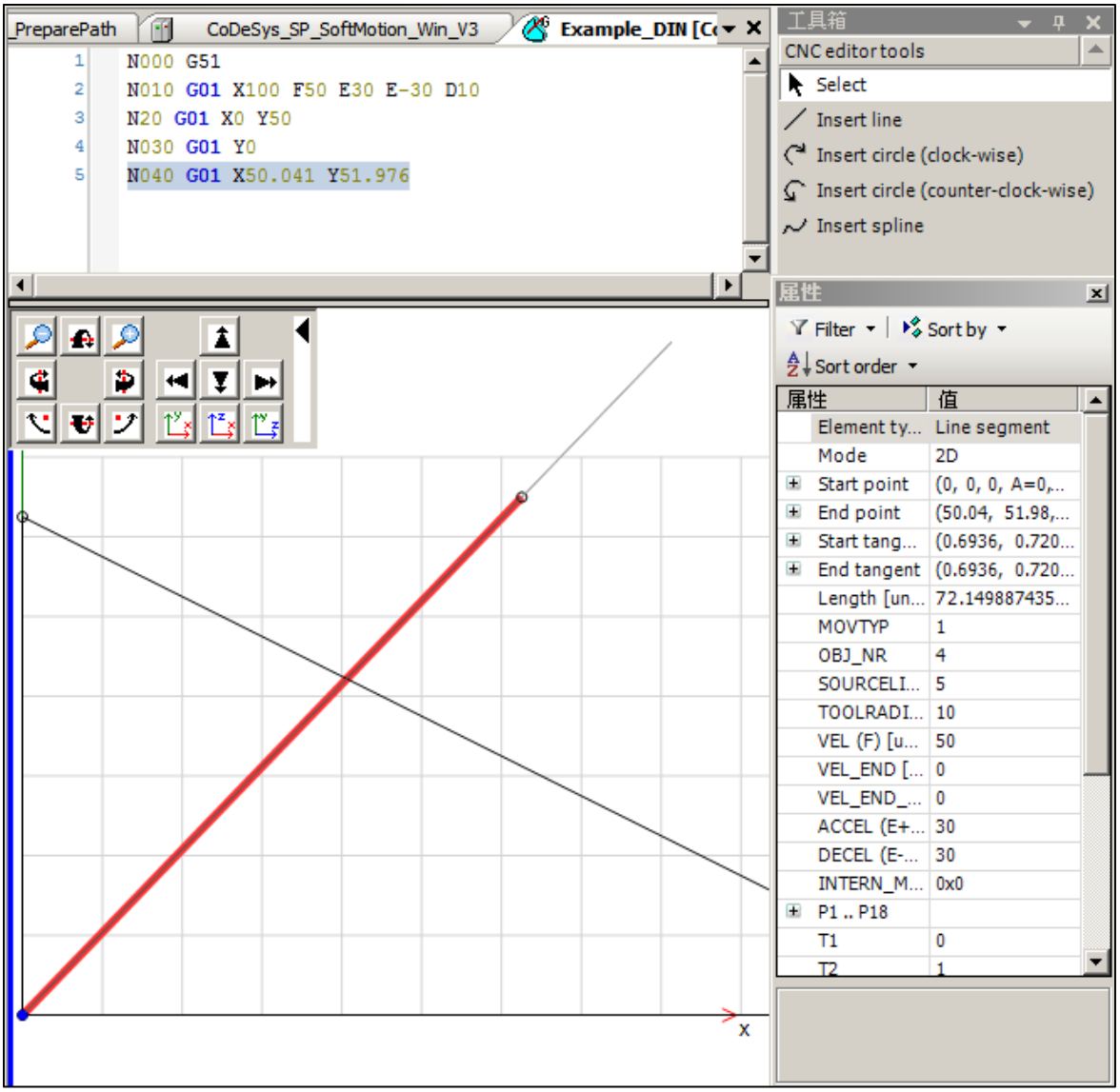
2.4 CNC 编辑器

2.4.1 文本编辑器

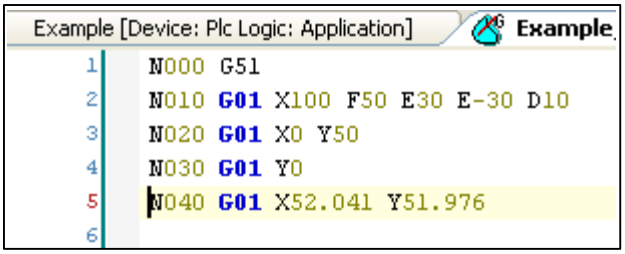
文本编辑器在'CNC' 窗口的上部。在这里，可以依据DIN66025 输入和修改CNC 程序。Din66025 的语法和被支持的路径元素相应地在图形编辑器中显示，并且可以在那里修改。在这种情况下，文本部分将被更新，反之亦然。

常规

DIN66250 的 CNC 程序



通过点击文本编辑器，语句以淡黄色突出显示，并且相关路径元素变为红色（有一个蓝色起点和一个黑边终点）。当图形路径图表中的一个路径元素被选中时，相关语句会以淡蓝色突出显示。



添加一个语句

用于插入或附加一个路径元素，生成一个新行和程序，且路径元素与 DIN66025 保持一致。

修改一个语句

通过直接在文本编辑器中编辑，全部语句都能被改变。然后，改变的元素照例在图形编辑器中被标记。通过按<F2>键，将打开输入助手，你可以选择要插入 CNC 程序中的全局变量。如果已经选择了编译模式 SMC_OutQueue，它在编译中会使 CoDeSys 创造一个准备好的 OutQueue-结构，当然，全局变量的引用在编译中也将被先前的初始值取代，由此无意的使用了全局变量。

删除一个语句

通过 Del 键删除一个选定的元素。

命令和选项

各种命令和选项可用于呈现附加的 CNC 程序数据组或对它们进行操作。它们都位于“CNC”菜单中。

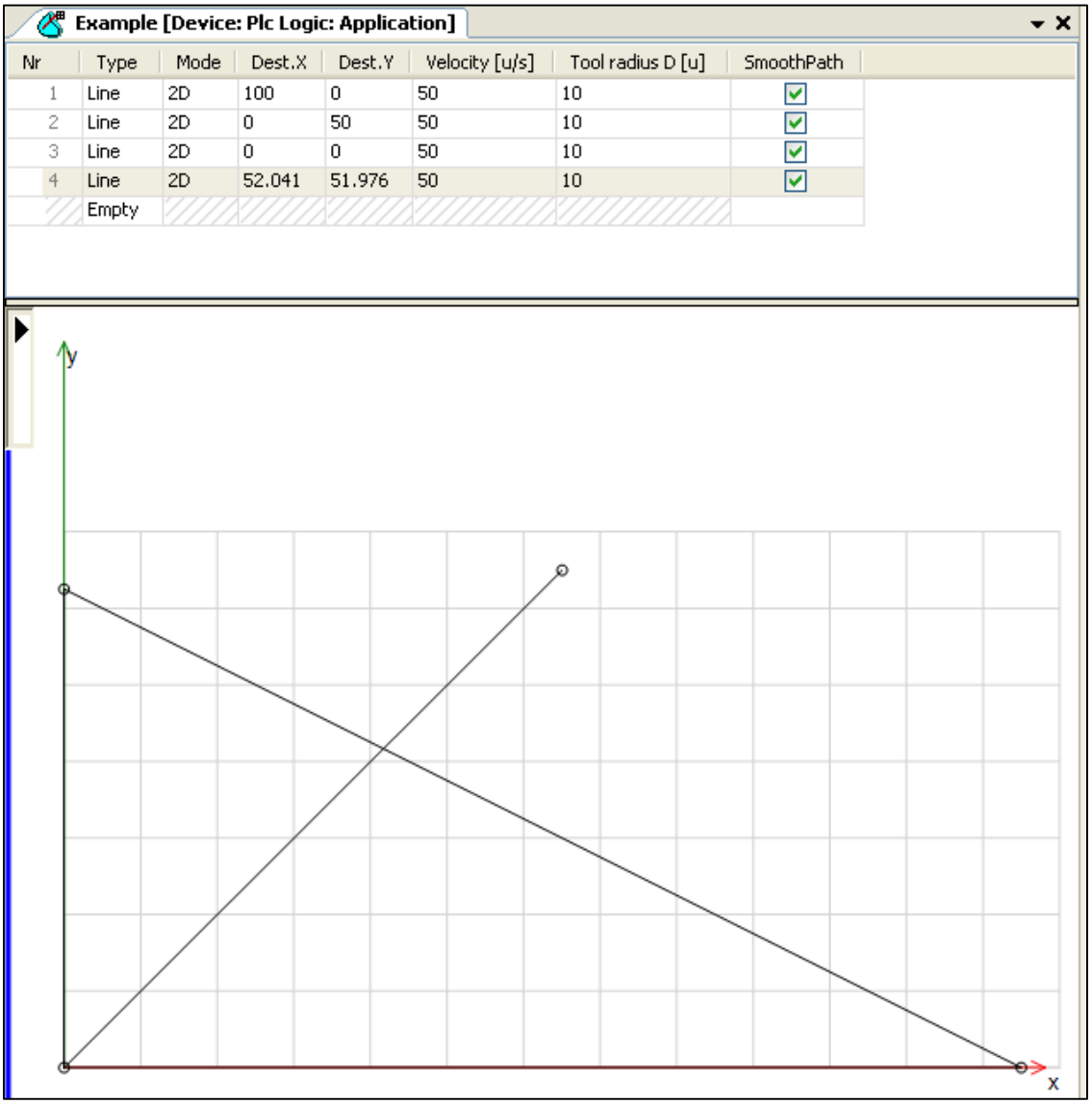
请注意，在解码器模块中提及的全局变量，当解码器处理线时将被评估， 先前可能会有一些循环传递给该对象。

示例：各种 CNC 程序都是在安装时被封装的。对它们做进一步了解，可参照全面描述。

CNC 表编辑器

在表编辑器中，路径元素被列于一个表中。表的列是按顺序配置的，并在表编辑器标签的CNC 设置中可以看到。

表视图的 CNC 程序



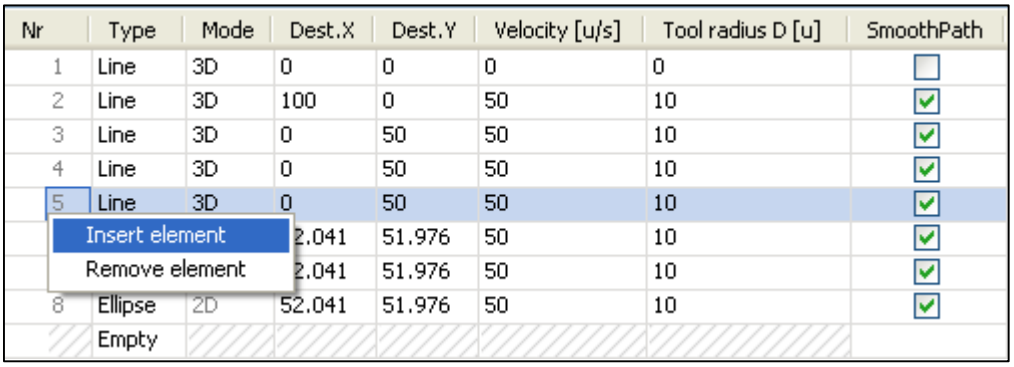
常规

通过在表编辑器中点击，标记行的焦点以浅蓝色突出显示，选中的表记录还将用蓝色细边框显示。相关路径元素变为红色，伴有一个蓝色起点。通过点击图形编辑器，当前选中元素路径显示为红色，伴有一个蓝色起点，同时，表记录中的相应行以浅灰色突出显示。可用性或列取决于元素类型，不能使用的列是空阴影的，不支持查找和替换功能。

添加一个元素

表的末行是一个空行，类型为空。当双击这条记录并从下拉菜单中选择一个类型时，就添加了一个新行。另一个获得新元素的方法是打开表记录的右键菜单，它提供了命令插入元素。这个命令用于插入一个新元素，它使用插入选中行前的行中元素一样的值。

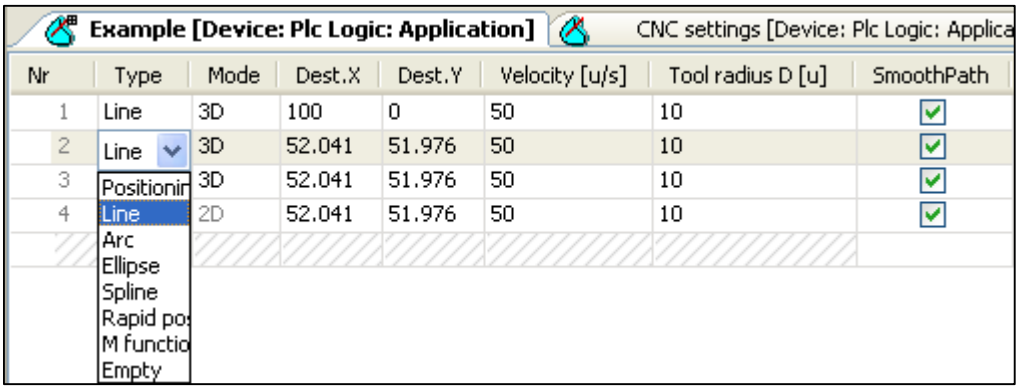
表编辑器的右键菜单



修改一个表条目

可改变列的内容以黑色字体显示。这样的记录可以通过在它们上双击鼠标或通过鼠标选中它们并按任意键来编辑。选定的记录将用一个蓝色细边框显示。然后打开在线编辑器或下拉菜单，就可以输入期望的值了。编号格式是美国式，所以不能改变行数。表记录的右键菜单提供命令为全部元素申请值，因此，当值有效时，则这个命令就复制记录的当前值到全部其他记录中或相同列中。

表编辑器中的下拉菜单



删除一个元素

用 Del 键或通过打开的右键菜单和命令删除元素，可以删除一个选定的行。

不支持的元素

与文本编辑器不同，表编辑器不支持下列元素和命令：抛物线、IEC 变量、跳转、坐标转换、步骤抑制、G4 和 G75。

2.4.2 图形编辑器

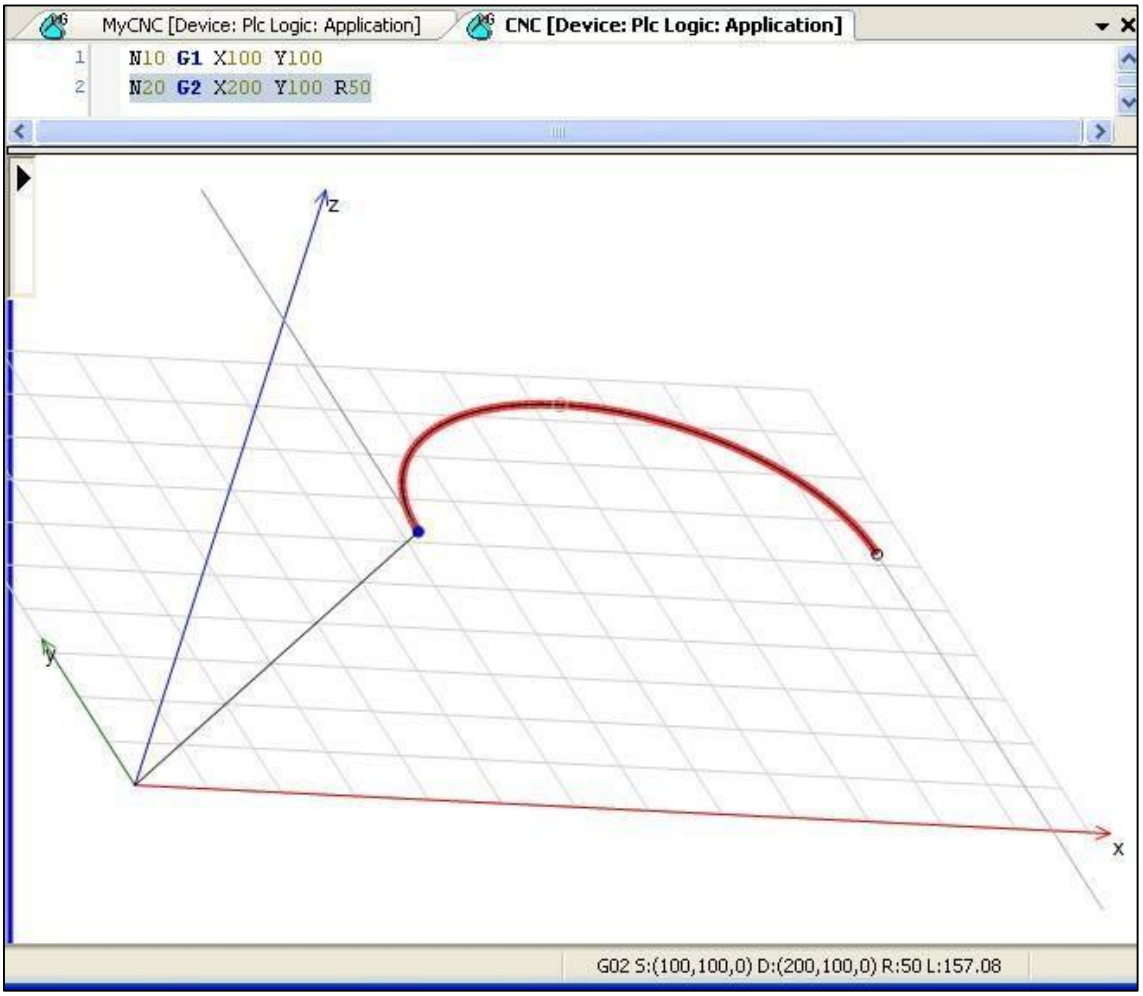
图形编辑器位于 CNC 编辑器的左下方。一方面，它显示在文本编辑器或表编辑器中描述的 CNC 程序，另一方面，可以使用工具箱修改程序。所有改变将在 Din66025 编辑器或者在窗口上面的表视图中立即自动更新。

常规

通过点击图形编辑器，可以在窗口左边看到一个蓝色边框，这意味着窗口获得了焦点。只要焦点在图形编辑器上，这个蓝色边框就会出现，并显示一个有绿色 y-轴、红色 x-轴和蓝色 z-轴的坐标系。背景是浅灰色的网格，该网格可以通过在菜单“CNC”中的命令“显示网格”打开和关闭。与所有其他元素显示为黑色不同，定位命令(G0)（包括转换位置）显示为绿色。如果终点是浮动

的，就被形象化的表示为黑边小圆。开始和结束切线都是灰色的。在一个路径元素上点击，该路径会变为红色，并有着蓝色实心起点和黑色空心终点。在文本编辑器中，相应代码语句会显示为浅蓝色。在表视图中，表行会以浅灰色突出显示。在状态行中，所选路径元素的当前位置会被显示出来。如有可能，该位置的坐标会在世界坐标轴中显示。

一个 CNC 路径的图形视图



移动 3D 程序路径

按住鼠标左键，CNC 程序会沿着x-轴、y-轴或z-轴移动。按住鼠标右键，CNC 程序会在x-轴、y-轴或z-轴上打开。在按住Ctrl 键的同时使用鼠标滚轮，可以改变缩放系数。在 CNC 左上方的控制面板中，图形显示也为移动 CNC 程序提供了不同的命令。

添加一个路径元素

工具箱为插入或附加一个路径元素提供了各种命令。

修改一个路径元素

通过在浮动点（显示为小圆）上点击，可以修改选定的元素。同时，这个圆也会变成红色并被加粗。如果切线是浮动的，在鼠标点击后也会变为红色。然后就可以在所选点的平面图上，通过移动它们的起点或终点来修改。取决于元素类型，浮动点的数量有所不同。在CNC 文本编辑器或表视图中，关联变量的当前值同时可见。

删除一个路径元素

通过 Del 键可以删除一个选定的元素。

命令和选项

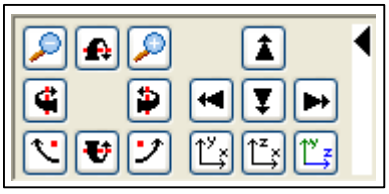
各种命令和选项都可用于呈现附加的CNC 程序数据组或对它们进行操作。它们都位于"CNC"菜单中。

示例：各种 CNC 程序都是在安装中封装的。对它们做进一步了解，可参照全面描述。

控制面板

在CNC 编辑器的左上方，有一个控制面板，它用来为改变路径外观提供不同命令。在这里，路径本身不会发生改变，但调整了运动路径的摄像头位置和观察方向。

CNC 控制面板



点击黑色箭头，控制面板消失，对路径的可视范围被扩大。可以使用小箭头进入面板，在最小化面板上点击，会将面板放到最大，可以看见整个控制面板。

最小化的 CNC 控制面板

控制面板为改变方向、位置和大小提供以下命令：

放大。

缩小。

改变摄像头的倾角。

改变摄像头的倾角。

顺时针改变摄像头的角度。

逆时针改变摄像头的角度。

顺时针转动摄像头。

逆时针转动摄像头。

向上移动摄像头。

移动摄像头至左侧。

移动摄像头至右侧。

向下移动摄像头。

设置视图方向为负 z 轴并重缩放。

设置视图方向为 x 轴并重缩放。

设置视图方向为 y 轴并重缩放。

2.4.3 工具箱

CNC 编辑器工具箱允许用户在图形编辑器中编制程序和延伸运动路径。所做的改变会同时在文本编辑器或表视图中看到。各种工具支持对鼠标的运动路径编程。缺省工具是'选择'工具，该工具提供在图形编辑器中一个路径元素的选择。要改变插入或附加模式，必须点击另一个工具，这取决于期望的路径元素。

- 选择工具是默认激活的。在激活模式下，鼠标在图形编辑器中点击将选择一个点或一个路径元素，被选中对象会变为红色，并带有蓝色的起始点。点击 ，将激活未激活的选择工具，且在图形编辑器中的光标是一个指针。

- 点击 激活直线工具。在激活模式下，鼠标在图形编辑器中点击，将插入或附加一条新线。

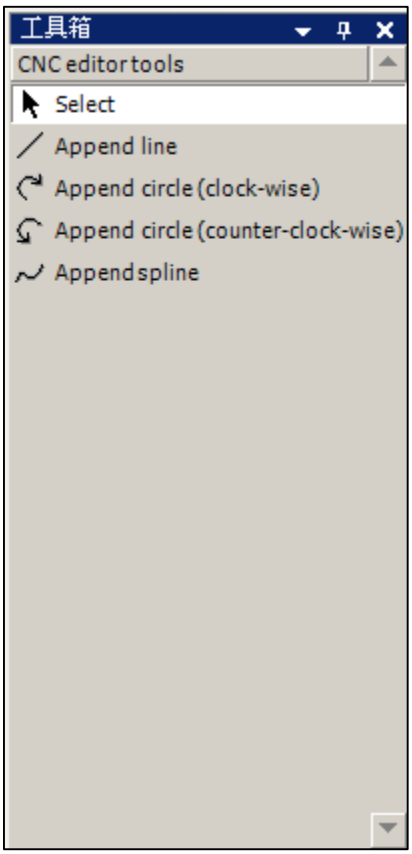
- 点击 激活圆（顺时针）工具。在激活模式下，鼠标在图形编辑器中点击，将插入或附加一个新的顺时针圆。

- 点击 激活圆（逆时针）工具。在激活模式下，鼠标在图形编辑器中点击，将插入或附加一个新的逆时针圆。

- 点击 激活曲线工具。在激活模式下，鼠标在图形编辑器中点击，将插入或附加一个新的曲线。当最后元素或选中元素是 G5 曲线时，新曲线也会是 G5 曲线；否则，该曲线是 G10 曲线。

插入一个路径元素

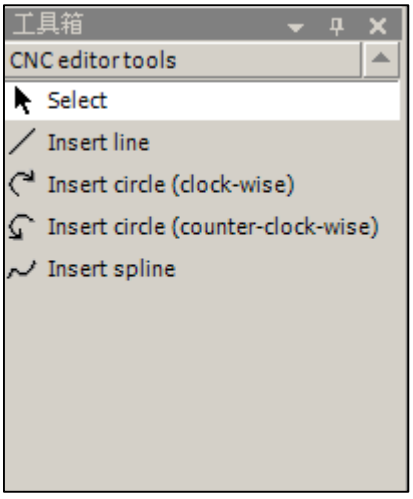
插入模式下的 CNC 工具箱



要插入一个路径元素，在这个新元素被插入前，应先选择一个元素，然后在工具箱中选择元素类型，就会立即为插入模式。在路径视图编辑器中的每个点击，标记着选中元素和新元素之间的连接点，于是新元素就被插入了。该新元素将显示为红色，带有一个蓝色起始点，而后，它就是下一个选中元素，且工具箱仍旧是插入模式。

附加一个路径元素

附加模式下的 CNC 工具箱



要附加一个路径元素，不需要在路径视图编辑器中选中元素。可以在工具箱附加模式时，从中选择期望的元素类型。在路径视图编辑器中，每个点击标记运动路径的新终点，并附加了一个新元素。这个新元素将只显示为黑色，且其起点和终点是两个小圆。

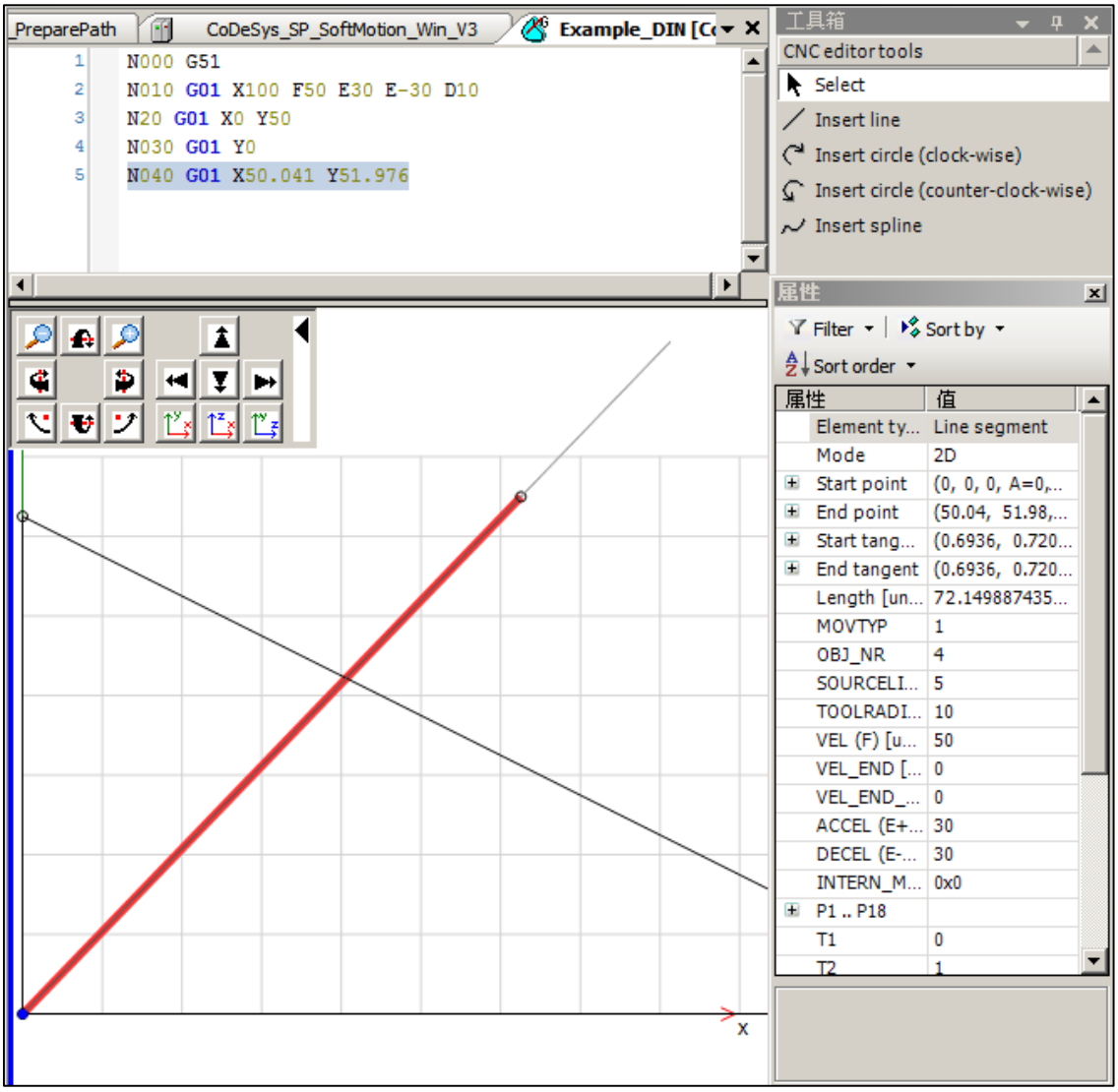
常规

在插入或附加模式下，图形编辑器中的光标由一个十字表示。通过使用 Esc 键，可以改变为选择模式，此时，光标再次变回指针。

如果选定的实现模式是文本编辑器，这个编辑器会随着下列颜色同时进行更新：

文本编辑器中的相关语句以淡蓝色突出显示。

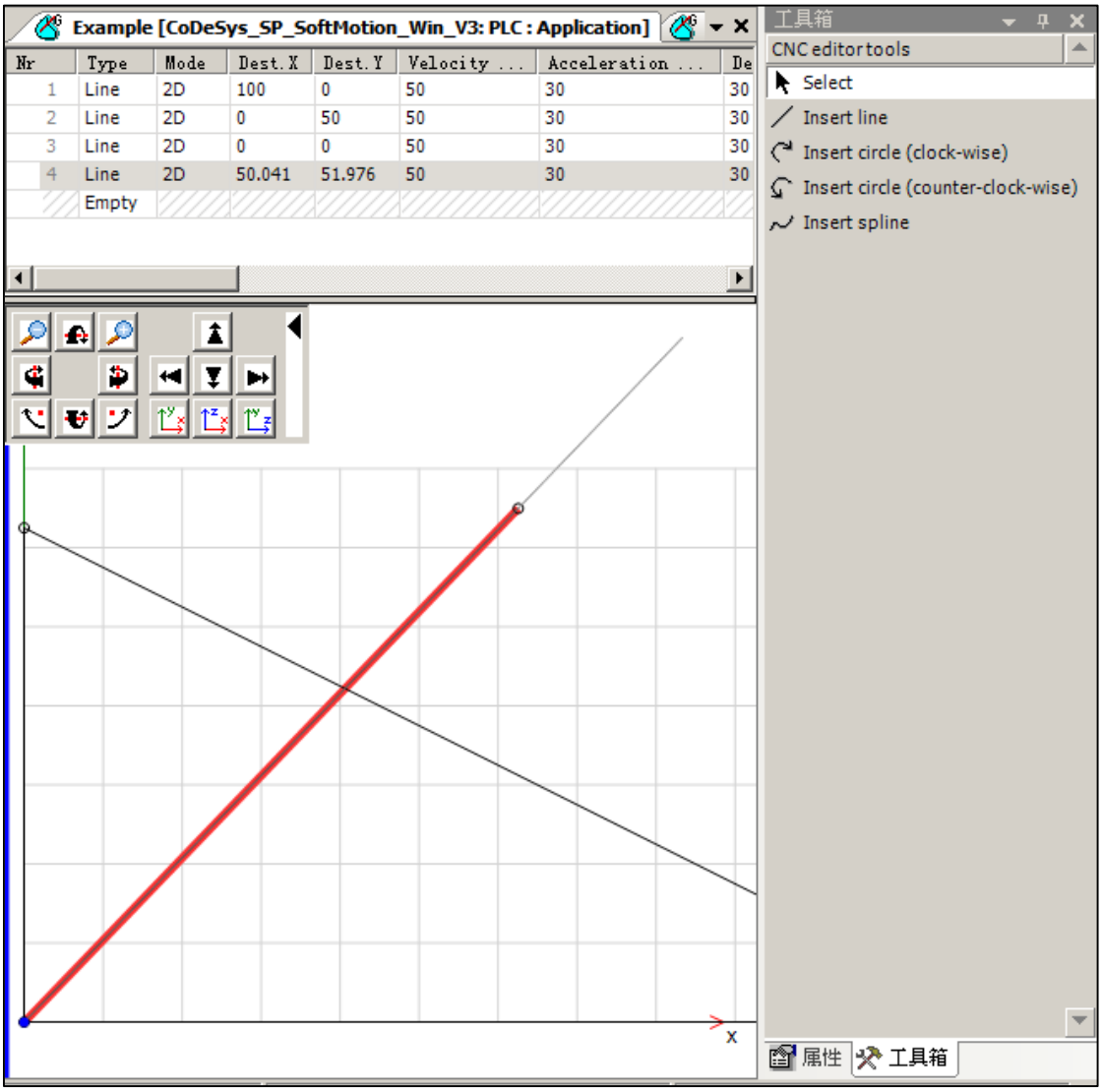
DIN66250 的 CNC 程序



如果选定的实现模式是表视图，这个表会随着下列颜色同时进行更新：

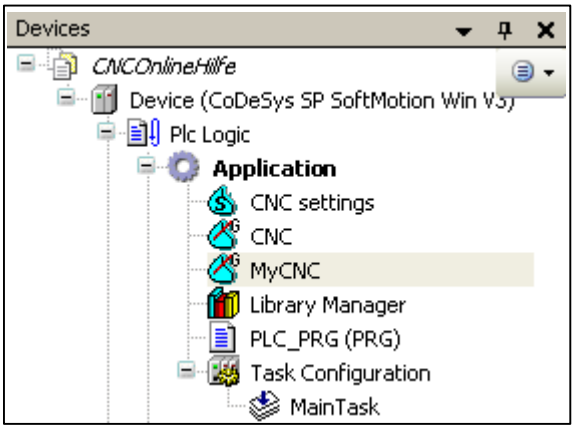
文本编辑器中的行以淡灰色突出显示。

表视图中的 CNC 程序



2.4.4 CNC 对象

CNC 对象的设备树



新建对象：

一个 CNC 对象必须位于能够执行 CoDeSys SoftMotion 的应用下。因此，选择一个应用并点击添加对象...，进入右键菜单，将打开相关对话框。在此，必须输入一个恰当的名称、预期的实施和

编译模式。点击OK 按钮后，一个新的CNC 对象就置于设备树中了。如果对象尚未存在，它会产生于” CNC 设置”旁。全部的 CNC 设置被存储在这里。相关对象的 CNC 设置会与 CNC 对象一起被存储，并且可以在"属性- CNC "中进行编辑。

删除对象：

功能与在所有其他对象中的相同，它可以删除 CNC 对象。

右键菜单：

打开 CNC 对象的右键菜单，"CNC 属性"对话框和 CNC 信息对话框也会有效。

CNC 对象的属性

在一个 CNC 对象中，命令“属性…”打开“属性-CNC”对话框，在这里可以改变 CNC 程序的具体设置。在右键菜单或‘视图’类中，都有‘属性…’命令。该对话框提供了 4 个标签：‘常规’、‘CNC’、‘访问控制’和‘编译’。

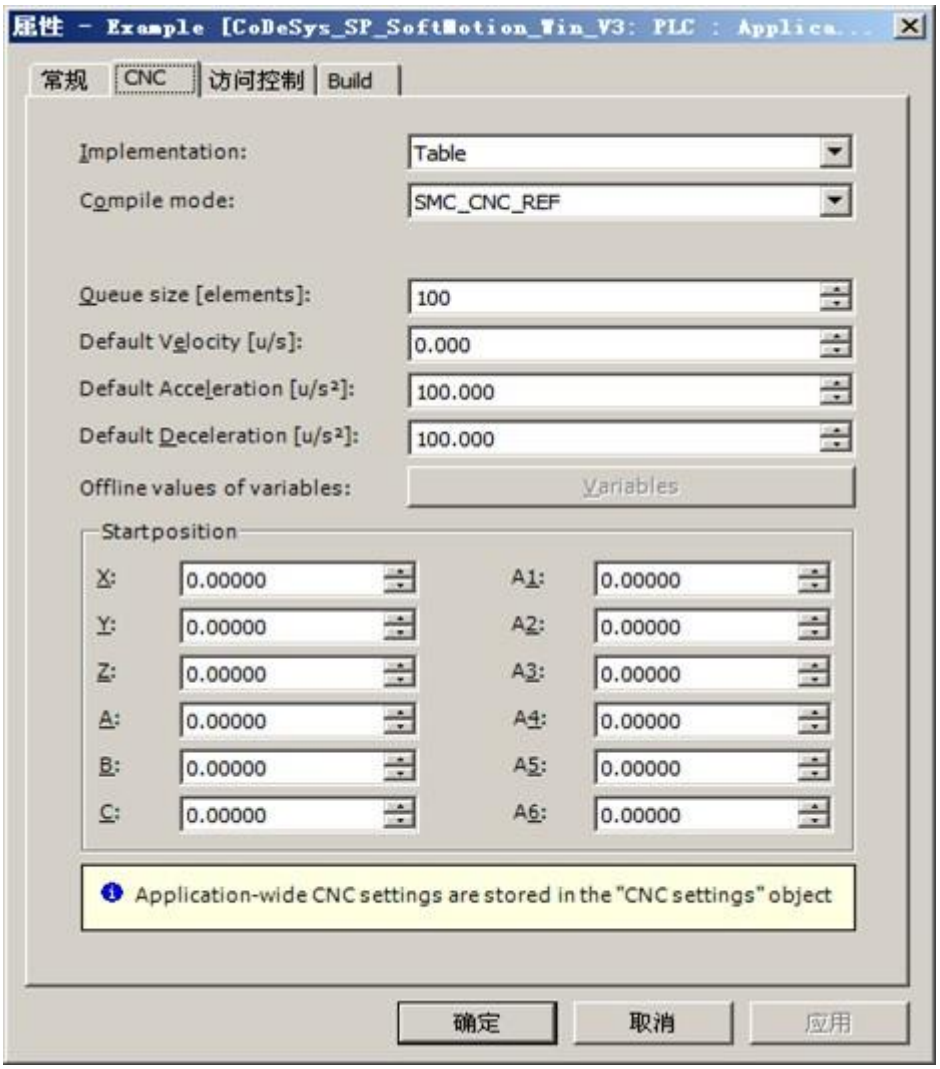
常规

属性常规

可以在这里更改 CNC 对象的名称。

CNC

属性 CNC



实现：

在这里定义实现类型。

- Din66025
- 表视图编

译模式：

编译模式取决于实现模式。

- SMC_OutQueue：在编译时创建 SMC_OutQueue 文件。
- SMC_CNC_REF：在编译时创建程序变量。

队列大小：缺省值是 100。

缺省速率：缺省值是 0。在 CNC 程序中，如果没有设置一个路径元素的速率，则使用该值。值范围：[0, 1e9]。

缺省加速度：缺省值是 100,000。在 CNC 程序中，如果没有设置一个路径元素的加速度，则使用该值。值范围：[0, 1e9]。

缺省减速度：缺省值是 1000,000。在 CNC 程序中，如果没有设置一个路径元素的减速度，则使用该值。值范围：[0, 1e9]。

起始点：缺省值是 0。

变量的离线值：如果定义了离线变量，通过点击变量按钮，打开离线值或变量对话框，可以显示和编辑该变量。通过使用 CNC 命令设置变量也可以实现该功能。

CNC 设置中的应用范围被存储在 CNC 设置对象中。

访问控制

在这里可以改变用户组权限。

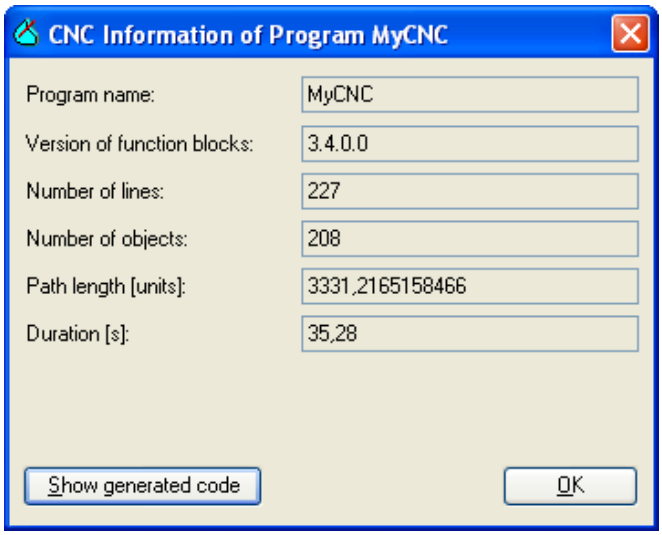
编译

在这里CNC 对象可以从编译中排除。于是没有 IEC 数据形成，可以在控制器的文件系统中以一个 ASCII 或一个 OutQueue 文件存储程序。如果想在运行时读取它，可以使用库 SM3_CNC 的一个模块。

CNC 信息

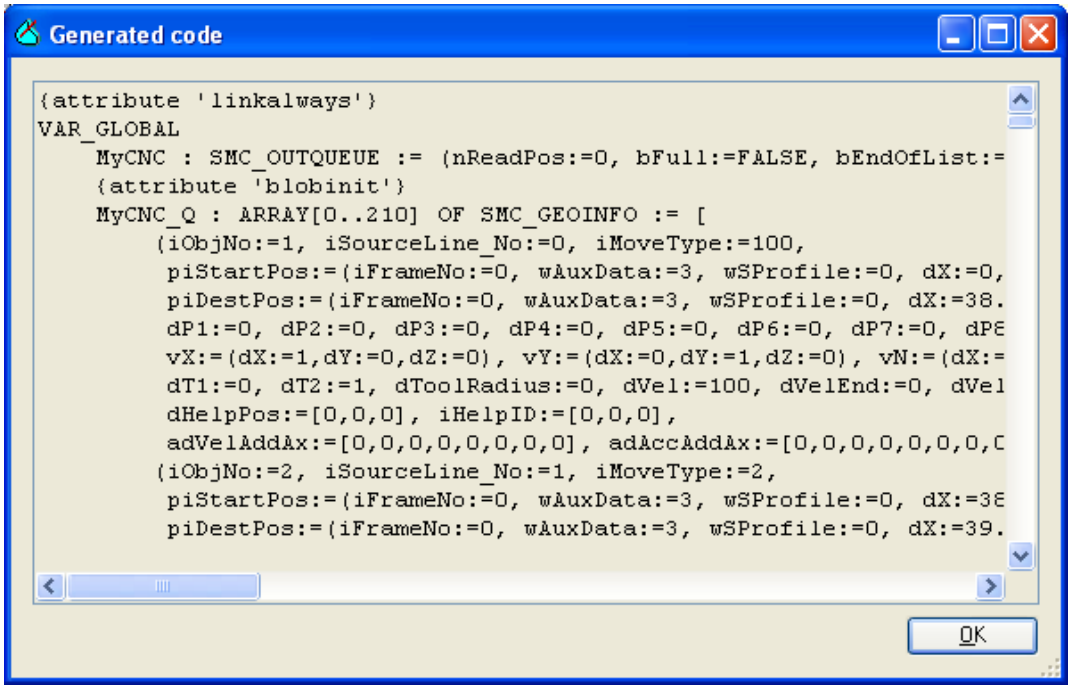
当前加载到编辑器的程序，它的程序<CNC 程序名>的 CNC 信息对话框提供以下信息：程序名、功能块版本、行数、对象数、路径长度、持续时间。此外，在成功生成路径预处理和插补点的情况下，预处理路径的计算路径持续时间也会显示。

程序 MyCNC 的 CNC 信息



显示生成的代码

生成代码对话框



当使用显示生成代码按钮时，专用 CNC 对象的程序化运动会显示一个清单，只有 CNC 程序无误时才可使用。然后，可以看到生成的 CNC 程序的 ST-代码和声明部分， 该部分分别为 SMC_CNC_REF 和 SMC_OutQueue 初始化。

2.4.5 CNC 数据结构

在编译工程的过程中，已经编辑好的CNC 数据结构在内部会转换成一个全局变量列表。如果编译模式设置为 SMC_OutQueue，每个结构变量由类型为 SMC_OUTQUEUE 的一个结构变量来表示。就是 SMC_CNC_REF，由 SMC_CNC_REF 代表的一个 CNC 元素。在IEC 程序运行期间，数据结构被实例化和全部赋值，它可以在 IEC 程序中处理，在这里提供了一个全局变量。所有结构都是 SM3_CNC 库的一部分。

全局访问 CNC 数据

决定CNC 程序中生成对象的初始化类型和数量期间，在一个隐式声明全局数据结构中，全部的对象都列在了一起。类型为 SMC_OUTQUEUE 的对象列在全局数据对象 g_CNCQueueManager 中，具有 Count 属性和 GetOutQueue(int n)方法。类型为 SMC_CNC_REF 的对象列在全局数据对象 g_CNCProgManager 中，具有 Count 属性和 GetProgram(int n)方法。用户可以在一个 IEC 程序中访问它们了。

示例：

访问 SMC_OUTQUEUE 结构的数据对象：

```
PROGRAM CNCManageQueue
VAR
    pCNC_OutQueue   : POINTER TO SMC_OUTQUEUE;
    n: INT;
    i: INT;
END_VAR
```

```
n := g_CNCQueueManager.Count;
FOR i:=0 TO n-1 DO
pCNC_OutQueue := g_CNCQueueManager.GetOutQueue(i);
 (* pCNC_OutQueue verarbeiten *)
END_FOR
```

访问 SMC_CNC_REF 结构的数据对象：

```
PROGRAM CNCManageRef
VAR
    pCNC_Ref: POINTER TO SMC_CNC_REF;
    n: INT;
    i: INT;
END_VAR

n := g_CNCProgManager.Count;
FOR i:=0 TO n-1 DO
pCNC_Ref   := g_CNCProgManager.GetProgram(i);
(* pCNC_Ref verarbeiten *)
END_FOR
```

2.4.6 CNC 在线

在线模式下，状态行不可见。

2.5 CNC 设置编辑器

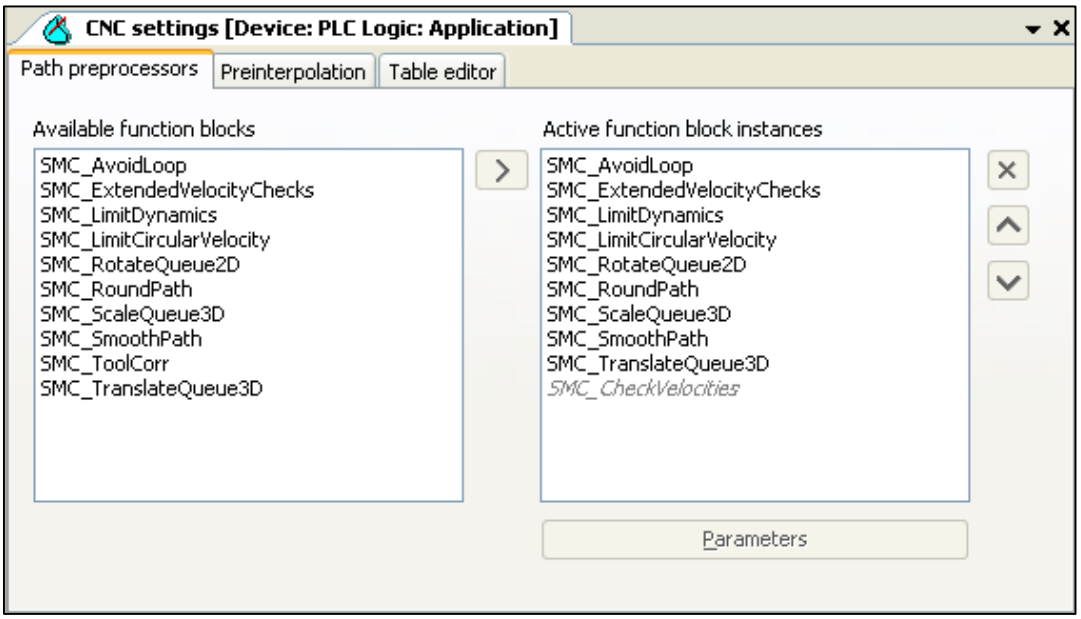
图标：

当产生一个 CNC 对象 ，并且不能被重命名或删除时，自动生成 CNC 设置对象。它位于一个应用的设备树下，那里可能仅存在一个 CNC 设置对象。在这里存储的设置对应用中全部的 CNC 对象都是有效的。而在 CNC 程序的属性 对话框中存储的设置仅对特定的 CNC 程序有效。

在CNC 设置对象上双击，打开CNC 设置对话框，在该窗口中可为路径预处理器、预插值和CNC 表编辑器定义设置。当在应用对象下没有 CNC 对象存在时，该对象只能被删除。

路径预处理器

路径预处理器的 CNC 设置



路径预处理器的调制程序，可以通过选择所需的功能块来编程。在活动的功能块实例列表中的位置，定义了调制程序的变化，并且此位置是可以改变的。在活动的功能块实例列表中，可以多次列举同一个功能块。

在左侧，列出了有效的功能块；在右侧，是活动的功能块实例。不可能删除以浅灰色斜体字显示的功能块实例。

使用该按钮或空格（Space）键，可用来复制一个被选中的可用功能块到活动的功能块实例列表中。通过双击左侧列表中所需的功能块，同样可实现此操作。

使用该按钮或删除（Del）键，可用来删除所选择的实例。

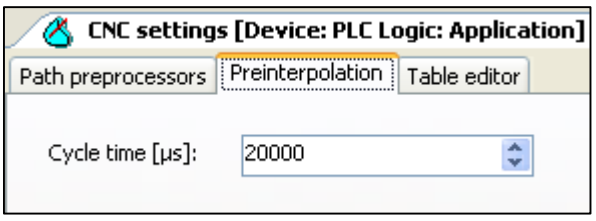
使用该按钮，可用来将选择的功能块实例向上移动一个位置。同样，在按住上挡（Shift）键的同时使用向上（↑）键也可以实现该操作。

使用该按钮，可用来将选择的功能块实例向下移动一个位置。同样，在按住上挡（Shift）键的同时使用向下（↓）键也可以实现该操作。

一些实例有参数，这时可以通过单击参数按钮或在实例上双击，打开各自的参数编辑器对话框进行设置或修改。同样，可用 Shift + P、Alt + P 或 P 键完成操作。

预插值

预插值的 CNC 设置

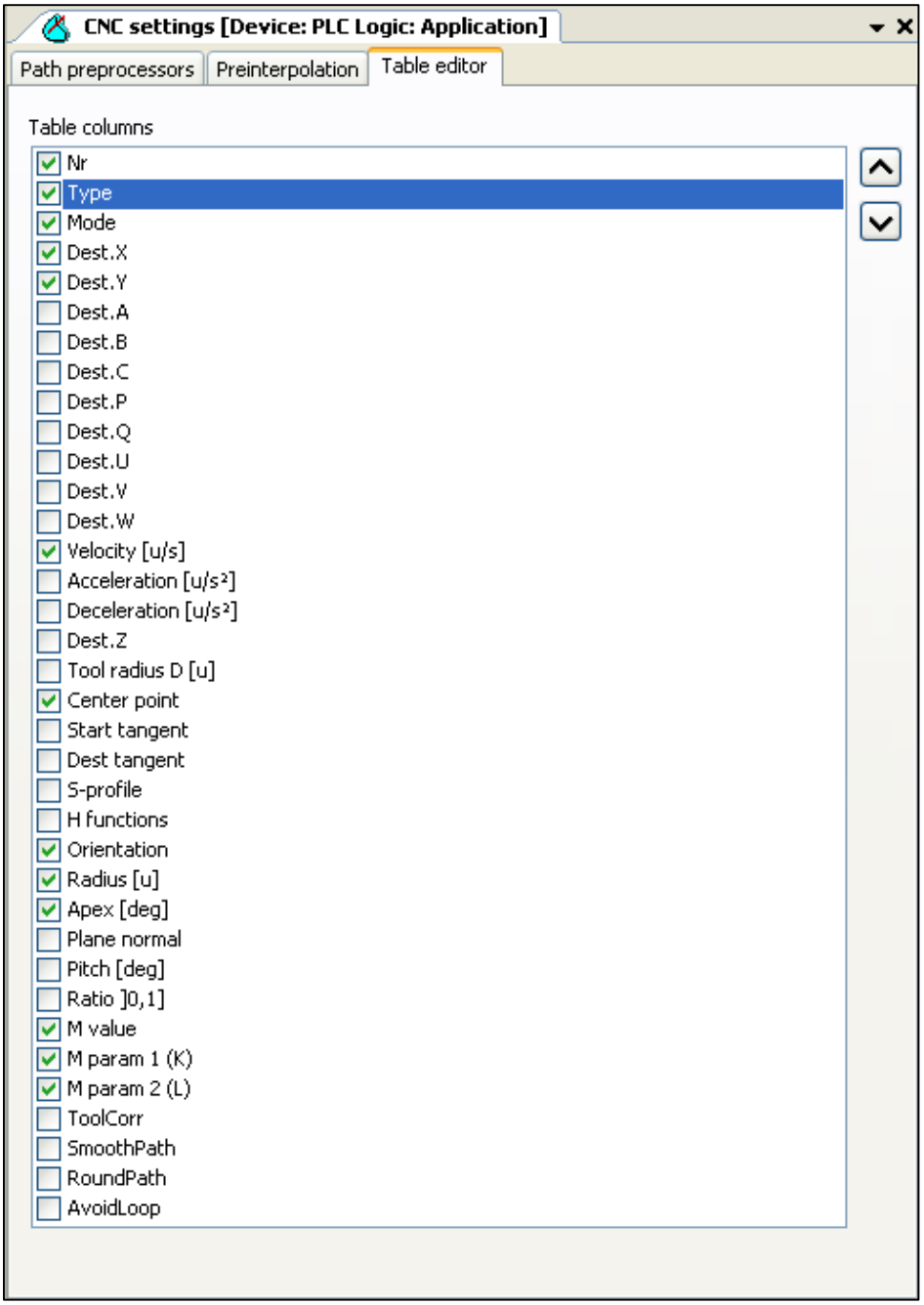


这里可以设置插值时间。缺省值是 20000µs。

注意：插值时间和相关任务的周期时间，必须与CNC 编辑器和在线应用的同一运行状况相匹配。

表编辑器

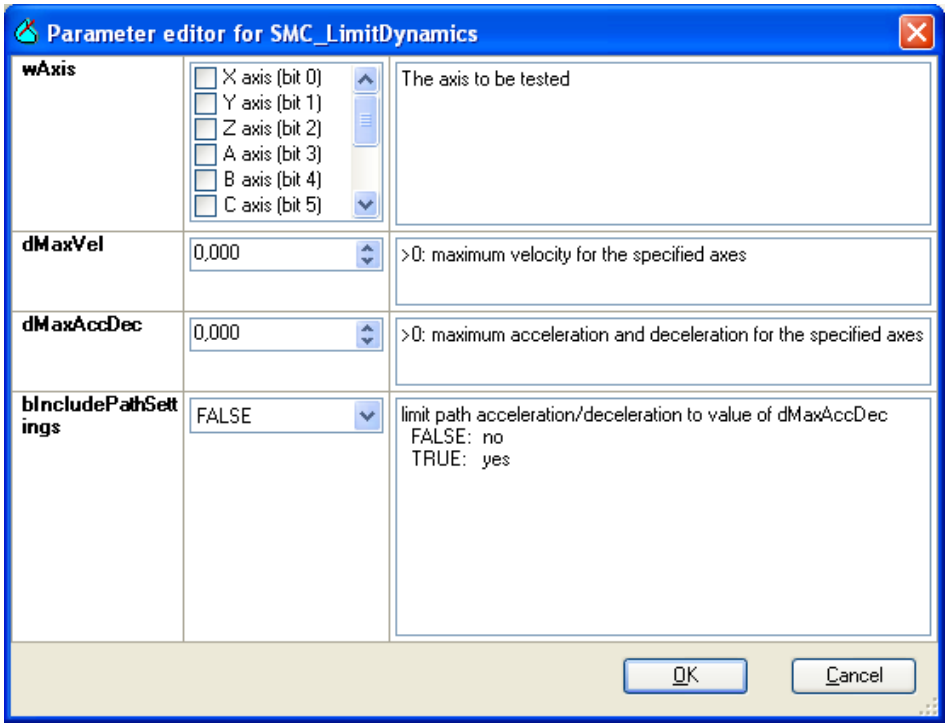
表编辑器的 CNC 设置



这里定义在表编辑器的列数和顺序。在表列数的列表中，列出了所有可用的属性。只有设置了检查标记的属性，可以在表编辑器中看见。通过单击选择框或使用空格（Space）键，可以设置或删除检查标记。定义属性的顺序，将列表属性向上移动可使用或 Shift + ↑ 组合键，将列表属性向下移动可使用或 Shift + ↓组合键。列的类型不能设置为可见。所做的修改会立即在所有的表编辑器中生效，包括打开的编辑器。

参数编辑器

参数编辑器对话框



在该对话框中，可以为路径编辑器中调制程序的功能块实例编辑输入。请注意，并非所有的输入都是可见的。（可编辑的输入在首次编辑前已具有初始值。）第一列为输入名称，第二列显示的是改变值，该值可以在下拉菜单或复选框中修改。最后一列显示的是输入注释。

2.6 声明编辑器

文本声明编辑器由“变量声明对象”插件提供，它用来创建一个POU 对象的声明部分。‘文本声明编辑器’插件可以由一个文本视图提供，在一个视图中所做的任何修改都会即刻应用到其它视图上。

基于声明编辑器选项中的当前设置，只有文本或者只有表格视图有效，或者可以通过编辑器窗口右边的按钮（文本/表格）来转换。

声明编辑器通常与编程语言编辑器联合使用。声明编辑器在窗口的上部。在线模式或离线模式下，编辑或监视对象时，可以将声明编辑器打开。声明关键字描述了 POU 类型（例如 PROGRAM、FUNCTION_BLOCK、FUNCTION ...），并且可以通过 POU-全局注释属性引申。

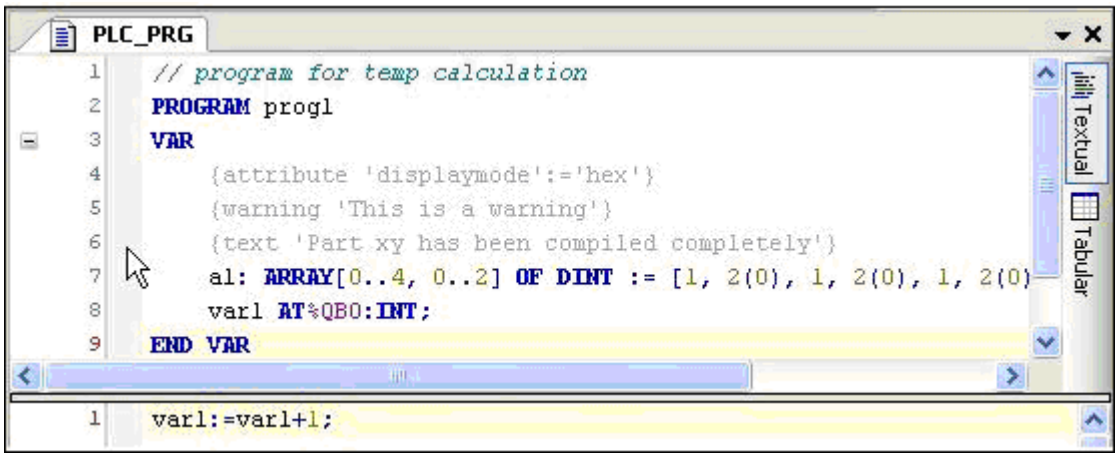
声明编辑器在线模式的结构与监视窗口类似。

全局变量列表和数据单元类型也需要进行变量声明，它们是单独的编辑器。

注意变量声明的概述。

文本声明编辑器

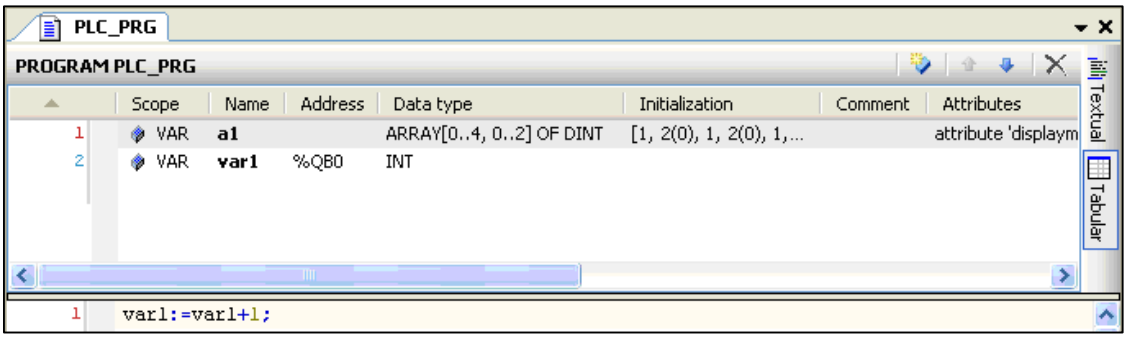
文本编辑器视图



行为和外观由各自当前文本编辑器的设置来确定，这些设置在选项和自定义对话框中定义。在这两个对话框中可以对高亮显示颜色、行号、跳格、缩进等进行缺省设置的定义。如果安装了相应的插件，则具有常规Windows 的功能，甚至具有智能鼠标的功能。

表格声明编辑器

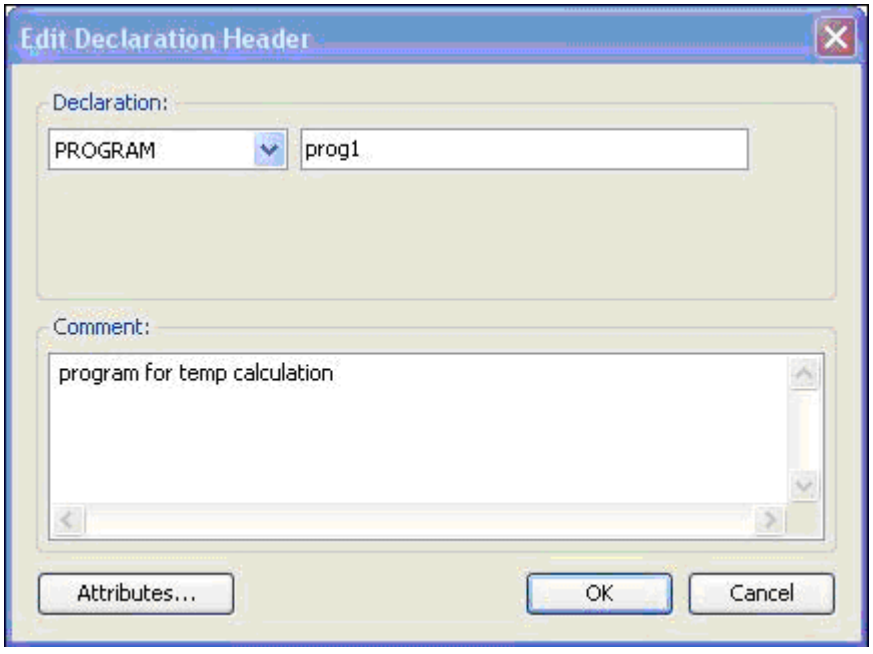
表格编辑器视图



编辑器的表格视图为普通的变量声明的定义提供需要的列：范围、名称、地址、数据类型、初始化、注释和（注解）属性。特殊声明作为编号行插入。

声明关键字可以在编辑声明关键字对话框中编辑，该对话框是在同名的命令中打开的。

编辑声明关键字对话框



在“声明：”下可以编辑 POU 对象的类型（从选择列表中选择）和名称。可以输入注释（通过 Ctrl+Enter 断行），属性按钮为指定的注解说明和属性打开属性对话框。

在现存声明上添加一个新行，先选中该行，然后从工具栏或右键菜单中使用 插入命令。要在表的末尾添加一个新的声明，先在现存最近的声明行处点击，同样使用插入命令。

在默认下，最新插入的声明首先使用范围"VAR"和最近使用的数据类型。强制变量名称的输入栏将被自动打开，在这里输入一个有效的标识符，并用<Enter>键或在视图的其它地方点击来结束。

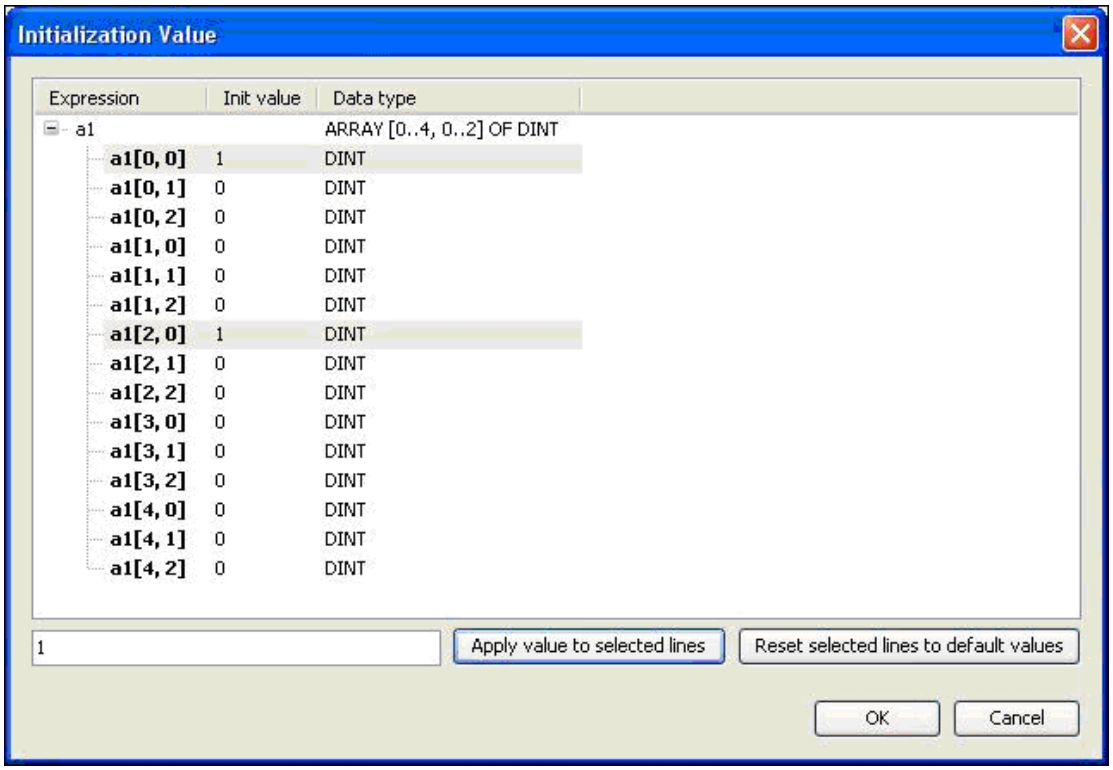
在每个表格单元上双击打开并输入值。

要编辑范围，双击打开列表，从中选择期望范围和范围属性关键字（标志）。

数据类型可以被直接键入或通过 >按钮使用辅助输入或数组向导。

初始化值可以被直接键入或通过[...]按钮使用初始化值对话框，这对结构变量很有用。

初始化值对话框

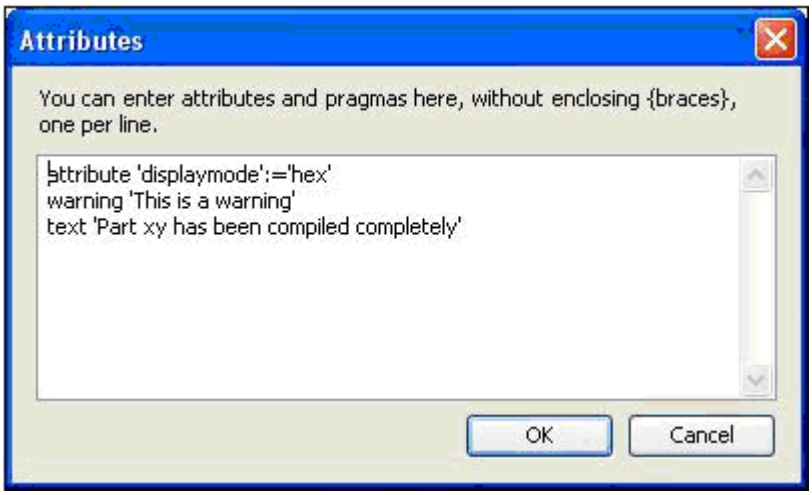


变量的所有表达式都将以当前初始化值显示。在列表下的区域中选择期望值并编辑初始化值，然后使用应用值到所选行按钮，使用重置所选行为默认值按钮恢复默认初始值。

注释中的换行符可以通过 Ctrl+Enter 插入。

在属性对话框中可以完成属性输入，在这里，多重属性和注记可以以文本格式输入。它们不需要封闭的大括号{}即可插入，每个都占一行。以下列映像中显示的为例，可见相应的文本视图在映像“文本视图窗口”中。

属性对话框，示例



每个变量是在单独的行中声明的，行是被编号的。

通过工具栏的向上移动/向下移动命令或右键菜单，可以选择一条线并向上或向下移动它，从而为行改变顺序（行号）。

声明的列表可以通过鼠标在各自列的列头处单击，根据每个列来分类。当前列的顺序，是由箭头标志指出的：（升序）或（降序）。在列头处的每次点击都会在升序和降序间改变。

要删除一个或几个声明，从右键菜单或从工具栏(✕)中选择行并使用<Del>或删除命令。

在线模式下的声明编辑器

离线模式下，在窗口中打开每个对象。登录到目标系统后，在线窗口将自动显示每个对象。

声明编辑器在线模式的表格显示与监视窗口类似。标题行显示实际对象的路径“<device name>.<application name>.<object name>”。每个监视表达式的表格，显示数据类型和当前值，还显示当前预设定的强制值或写入值。

为了对变量赋值，既可以用预设值对话框，也可以点击列中的赋值区域，直接输入值。

对于布尔变量的处理则更加容易。根据下列顺序，使用回车键或空格键，可以切换布尔变量的预设值：

如果值为 TRUE，则步骤是：FALSE -> TRUE -> nothing。

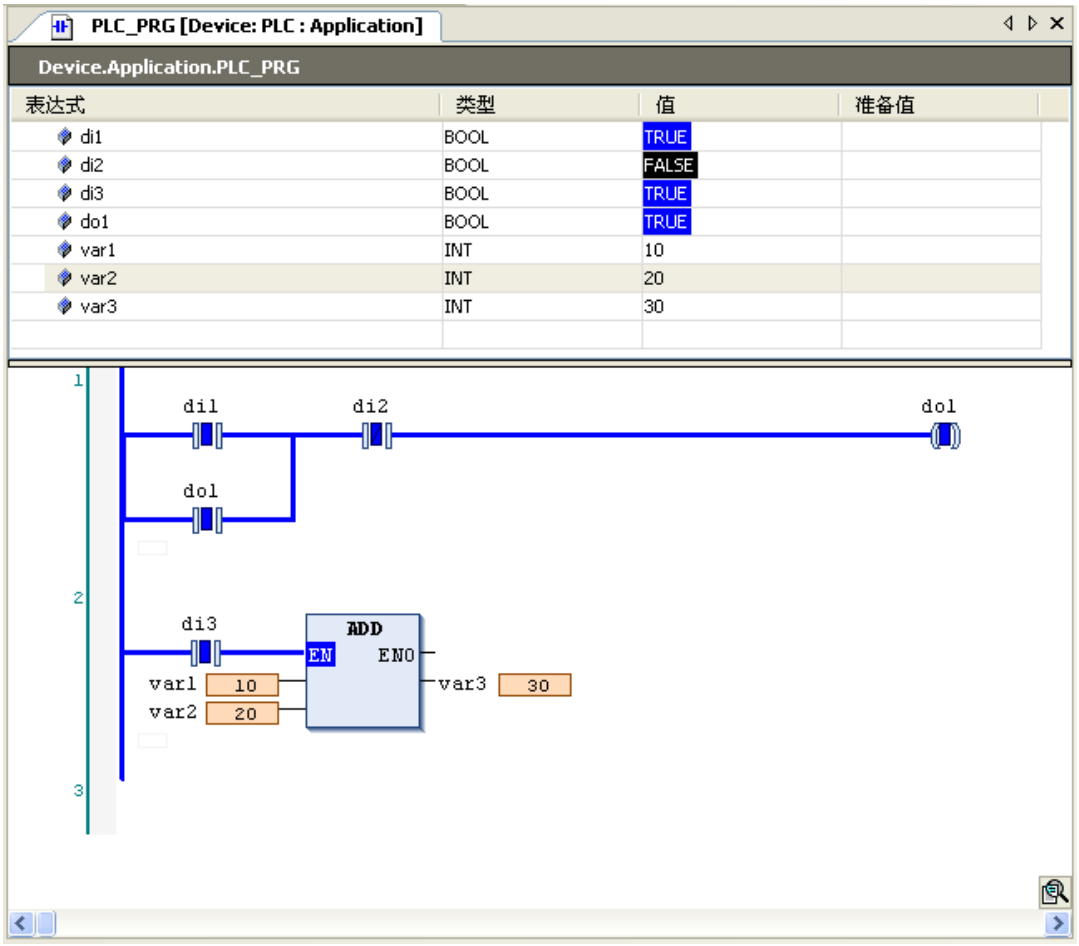
如果值为 FALSE，则步骤是：TRUE -> FALSE -> nothing。

如果监视表达式（变量）是程序实例，例如功能块，前面会标注加号和减号。通常用鼠标点击到这个标注，则实例对象的详细元素将会显示出来（如下图中的 fbinst 所示）、或者隐藏起来（如下图中的 fbinst2 所示）。图标表示变量的种类：输入变量、输出变量或普通变量。

只要默认设置取代常量（编译选项）被激活，常量在标题为值的所在列的值之前由C标志显示。

在实现部分，当鼠标位于一个变量上时，一个提示框将显示声明和变量注释。见下面的图：

举例：程序对象 PLC_PRG 上方的声明编辑器在线窗口



2.7 设备编辑器

2.7.1 通用设备编辑器

设备编辑器

设备编辑器提供了由设备视图窗口管理的，用于设备配置的对话框。

编辑器能通过“编辑对象”命令，或者通过在设备窗口中双击设备对象条目打开。

编辑器的基本设置要参照“设备编辑器选项”，例如通用对话框可以设置成不可见。

当设备条目在设备目录中被选中的时候，设备对话框默认可以通过双击或者选择“编辑对象”打开。

主对话框是根据设备类型，以设备名称来命名的

，它提供了包含以下子对话框的选项卡：

通讯设置：编程系统和可编程设备（PLC）之间连接的配置。不适用于纯 I/O 设备。

配置：分别显示设备参数的配置。

应用：显示目前正在 PLC 上运行的应用，并且允许从 PLC 中删除应用。

文件：主机和 PLC 之间的文件传输的配置。

日志：显示 PLC 的日志文件。

PLC 设置：与 I/O 操作相关的应用、停止状态下的 I/O 状态、总线周期选项的配置

I/O 映射：在工程（应用程序）变量中，I/O 设备输入和输出通道的映射。

用户和组：运行中设备访问相关的用户管理（不要与工程用户管理混同）

访问权限：特殊用户组对运行中的对象和文件访问权限的配置。

状态：设备的详细状态和诊断信息。

信息：设备的基本信息（名称、供应商、版本等等。）

总线详细配置对话框，例如：

- DP Profibus 配置

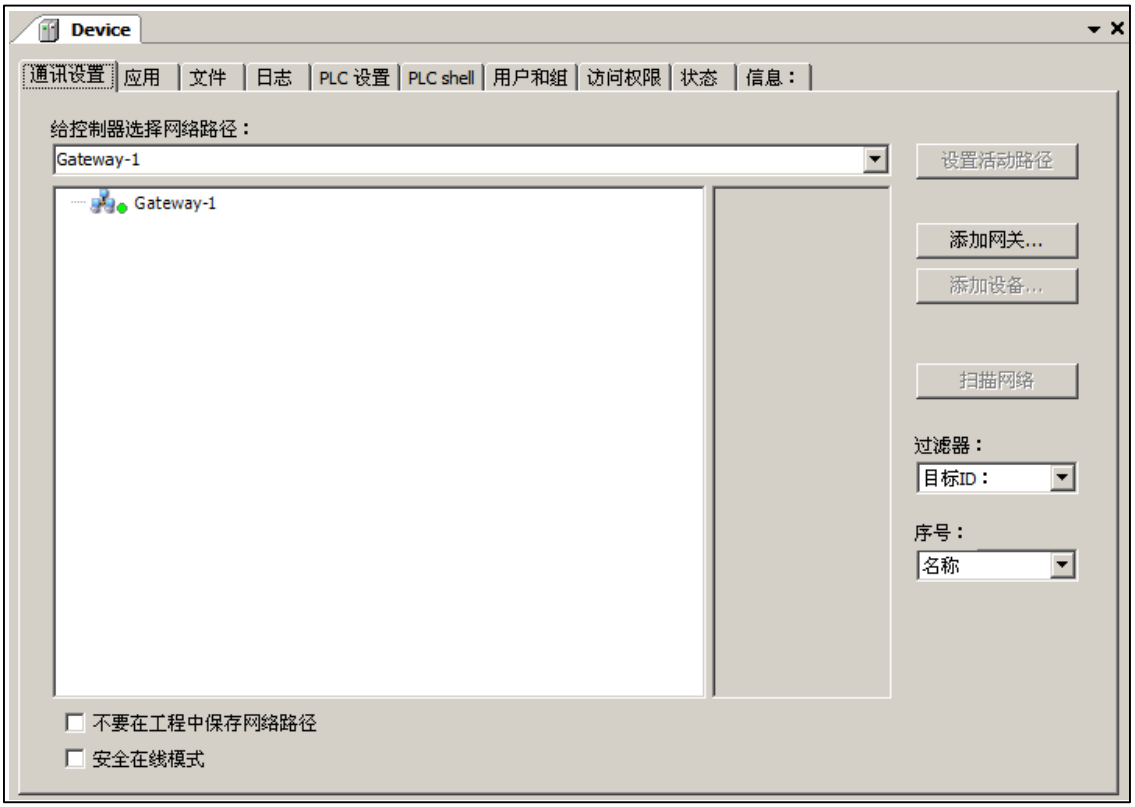
- Ethercat 配置

注意：“通讯设置”和“应用”选项卡关系到与 PLC 的通讯，并且是由“设备通讯编辑器”插件提供的。

通讯设置

此对话框由设备对话框（设备编辑器）上的一个选项卡打开，可以通过“编辑对象”命令或者在设备窗口中双击设备对象条目打开。它用来配置设备的通讯参数。

设备对话框，通讯设置

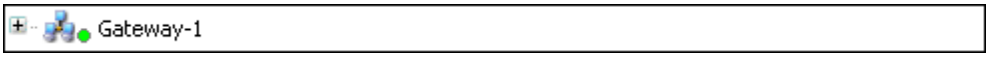


这个对话框包含一个由两部分组成的窗口，其中左边窗口为一个树状形结构，显示当前的网关通道配置，而右边窗口显示相应的通讯数据和信息。

当在你本地系统上创建第一个工程时 ，在配置目录下至少会包含一个条目：本地网关（在系统启动时自动运行）：

CoDeSys 网关服务：

CoDeSys 网关服务作为一个服务在系统启动的时候自动启动，该服务在系统栏上显示为一个控制图标：图符　表示正在运行，图符　表示停止网关。您可以停止单击该图标，通过它的菜单命令来停止或重启网关。如果需要的话，该服务也可以通过选择　的“网关”入口从程序菜单中启动。



在窗口的右边显示网关的设置（见下文，“网关”对话框）例如：

节点名称："Gateway-1"
驱动器：TCP/IP
IP 地址：127.0.0.1
端口：1217

当网关正常运行时，条目前面的粗圆点会显示绿色 ，否则显示红色 。粗圆点停留在灰色状态，则表示网关还没有被关联（根据一些通讯协议，不允许选中网关，所以它的状态不能被显示）。

在网关条目的缩进下面（通过+/-号来打开/关闭），你能看到通过这个网关可以获得的所有设备的条目：注意那些条目是被存储在系统中的，而不是存储在工程中。设备项之前有　图标。与当前配置的设备有不同目标 ID 的条目是以灰色字体显示的。要想得到最新的当前可用的设备清单，使用按钮或命令“扫描网络”（见下文）。

设备条目包括一个图标，后面是<节点名称> [节点地址]，右边窗口可以看到目标 ID，目标名称，目标类型，目标供应商和目标版本。

网关和设备节点树的显示可以被过滤器和排序函数控制。请看对话框右边的选择框：

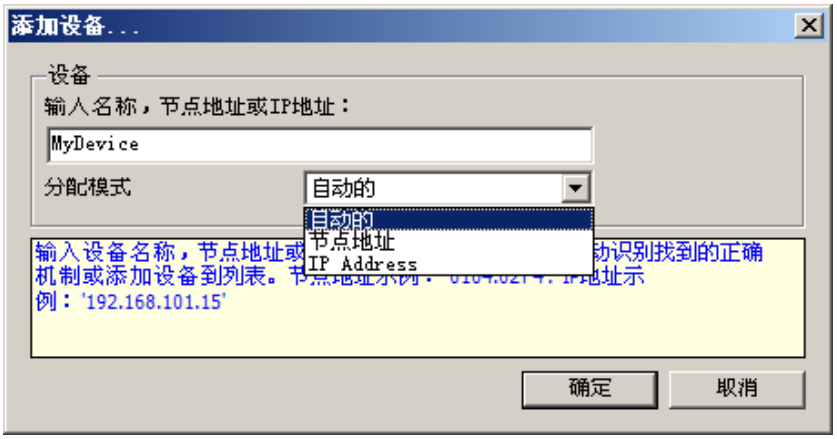
- 过滤器：列表可以被精简到只显示与工程中当前配置的设备目标 ID 相匹配的设备。
- 排序：列表可已按照名称，节点地址的字母升序排列。

当前被选中的网关通道一直显示在“选择到达控制器的网络路径”区域

使用下列按钮修改通讯配置：

当前选中的网关通道通常会显示在“请选择到控制器的网络路径”字段。使用关联菜单中的按钮来修改通信配置。

设置活动路径	此命令设置当前所选通信通道为活动状态。请查看"设置活动路径"。也可以双击通道树上的条目执行相同操作。
添加网关...	此命令打开“网关”对话框，通过此对话框你可以定义并添加一个网关到当前配置中。此命令打开“网关”对话框，在这里你可以定义一个添加到当前配置的网关。
添加设备...	此命令打开“设备”对话框，通过此对话框你能手动定义并添加一个设备到当前选中的网关条目中。（功能上相当于“扫描网络”）。

	设备对话框 
编辑网关...	此命令打开“网关”对话框进行编辑当前所选网关的设置。
删除所选设备	此命令从设备树中删除所选设备。
解析地址	此命令扫描在配置树中指定名称的设备（区分大小写的查找）。查找到的设备会在其名字后补充唯一的节点地址之后显示在网关中。扫描通常涉及在那个网关树下的条目被选中的设备。
连接网关	该命令打开一个包括两个标签的对话框，用以辅助配置本地网关，分别提供了一种手动编辑“Gateway.cfg”文件的方法。
扫描网络	此命令搜索在本地网络中可用的设备，并且因此而更新配置目录。

安全在线模式：如果用户由于安全原因需要提示确认，请激活这个选项。该选项在选择下列在线命令中的某一个时发生作用：强制值，多重下装，释放<应用>中强制列表，单循环，启动<应用>，停止<应用>，写值。

不要在工程中保存网络路径：如果当前网络路径的定义不应在工程中存储，而要存储在自己计算机的本地选项设置中，则激活这个选项。这样，当在同一个计算机中重新打开工程的时候，路径设置会被重新装载，但如果在其它计算机上使用该工程，则路径需要重新定义。

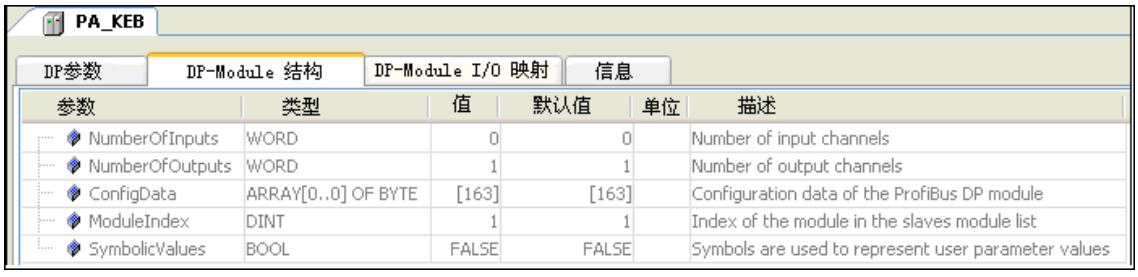
配置

如果“显示通用设备视图”选项被激活，则“设备”对话框（设备编辑器）的选项卡上的配置对话框是可用的。通过“编辑对象”命令或者双击设备条目可以打开当前在设备树下被选中设备的设备编辑器。

此选项卡以“<module type>配置”（例如：“DP-模块配置”）命名，并且显示设备详细参数属性，设备描述，也可以编辑参数值。

设备对话框，配置

Profibus 模块示例，所有参数均不可编辑：



参数	类型	值	默认值	单位	描述
NumberOfInputs	WORD	0	0		Number of input channels
NumberOfOutputs	WORD	1	1		Number of output channels
ConfigData	ARRAY[0..0] OF BYTE	[163]	[163]		Configuration data of the ProfiBus DP module
ModuleIndex	DINT	1	1		Index of the module in the slaves module list
SymbolicValues	BOOL	FALSE	FALSE		Symbols are used to represent user parameter values

以下各项用来描述一个参数：

参数：参数名称，不可编辑。

类型：参数数据类型，不可编辑。

数值：参数的默认值大多数情况下直接显示，或者通过一个助记符显示。如果此参数是可更改的（这依赖于设备的定义，不可编辑的参数颜色显示为灰色），编辑框可以通过鼠标单击表格区域打开，或者可以通过选择列表来修改参数值。如果此参数是一个文件说明，可以通过双击条目打开用于打开文件的标准对话框，并且另一个文件可以被选中。

默认值：默认的参数值，不可编辑。

单位：参数值的单位，例如“ms”毫秒，不可编辑。

描述：参数的简短描述，不可编辑。

应用

此对话框由“设备”对话框（设备编辑器）的选项卡提供，可以通过“编辑对象”命令或者通过在设备窗口中双击设备对象条目打开。它用来扫描和删除 PLC 的应用。

设备对话框，应用

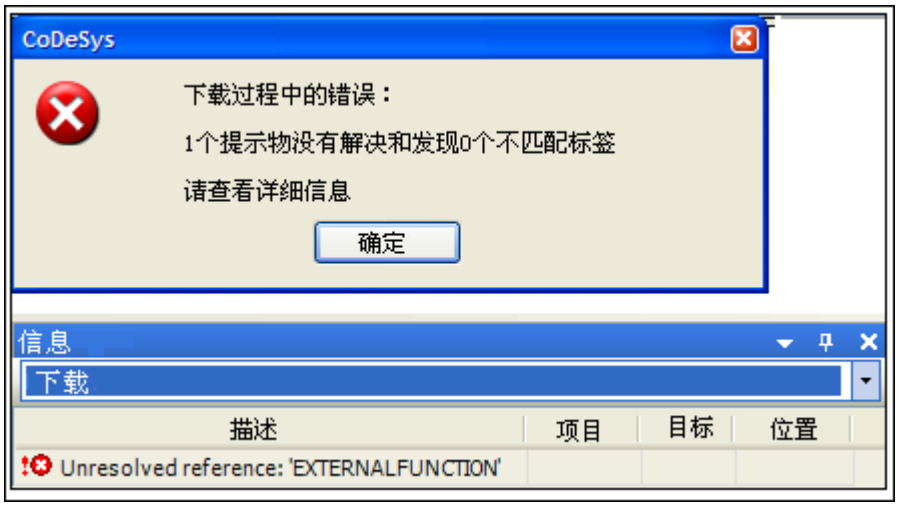
PLC 的应用： 上次扫描时在 PLC 上已经找到的应用的名称列表。

更新列表：扫描 PLC 上的应用，更新列表。

删除或者全部删除：删除 PLC 上当前被选中的应用，或者全部的应用。

当一个应用下载到 PLC 中，将检查 PLC 上的应用列表和设备目录上的应用列表是否匹配，以及 PLC 上的应用参数是否和设备配置（署名）里的应用参数匹配。如果不匹配，相应的消息（未定参考，或者署名不匹配）将会被发送到消息框和“下载”类别中的消息视图中。

未定参考消息，例如



如果多个应用直接分配到设备中，参考相应 I/O 映射的实际设置。

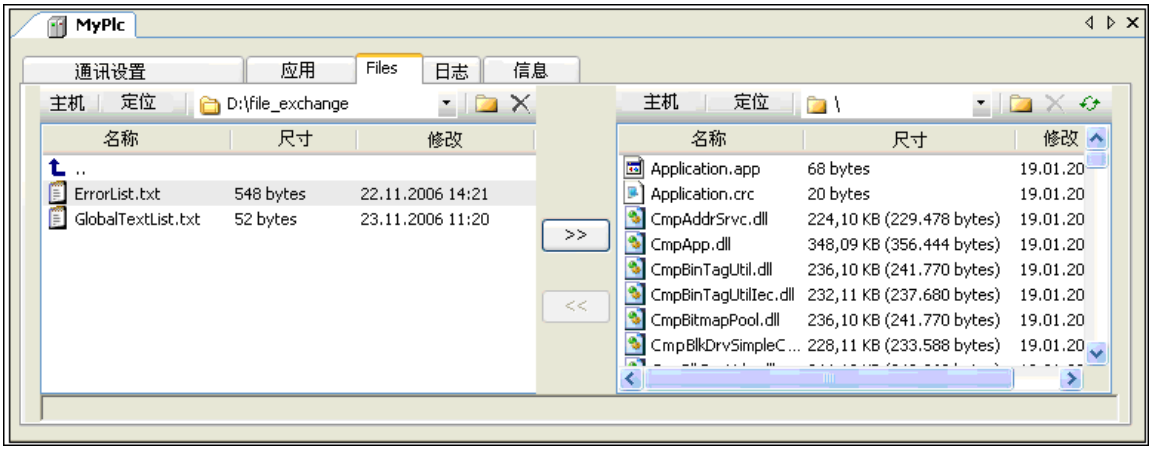
请在应用查看处理应用工程的信息。

文件

此对话框由“设备”对话框（设备编辑器）的选项卡提供，可以通过“编辑对象”命令或者在设备窗口中双击设备对象条目打开。

此对话框用来传输主机和控制器之间的文件。这意味着可以从本地网络目录中选择任何文件，把它复制到当前已连接的正在运行的系统的文件目录下，或者反向复制。

设备对话框，文件



在对话框的左边显示的是主机上的文件，右边显示的是运行系统的文件。

如果想要更新运行文件列表，用此按钮 。

如果想要创建一个新的文件夹（文件将复制到此文件夹中），用此按钮 。

如果想要删除当前选中的文件，用此按钮 。

在主机和运行系统的路径选择区中分别进入相应的目录，可以传送一个或多个文件。这可以通过条目区域的选项列表或者通过在文件系统目录里浏览的方式执行。

需要复制的文件必须在文件系统目录里被选中。允许选择多个文件，如果想要复制文件夹里面的所有文件，可以选中此文件夹。

若将选中的文件传输到定义好的主机或者运行系统目录，用按钮“>>” 或者“ <<”。“传输”在文本中相当于“复制”。因此，如果一个文件在“目标”目录里不存在，它也可以在那里被创建。如果一个相同文件名的文件已经存在而且不是写保护的，这个文件会被重新改写。万一文件被写保护了，则会产生一个相应的信息。

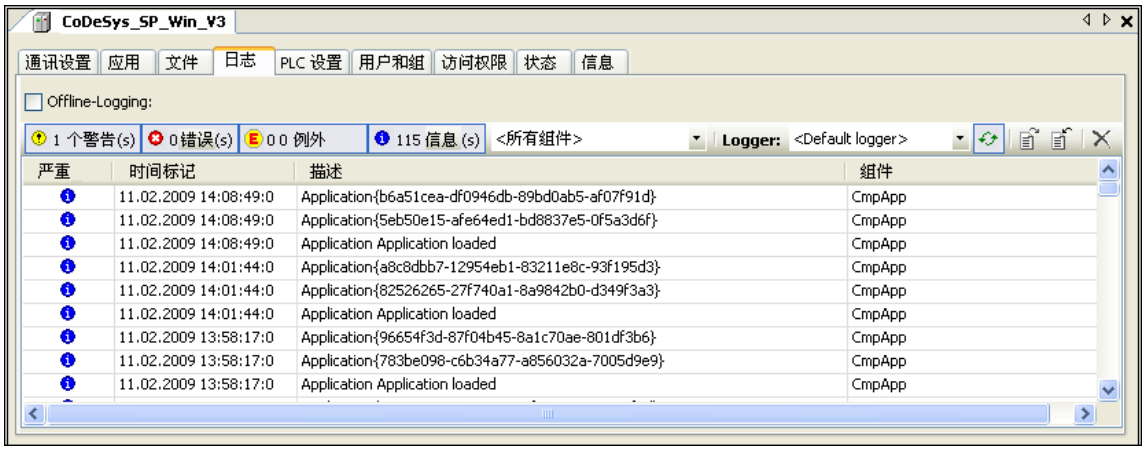
日志

此对话框可以由“设备”对话框（设备编辑器）的选项卡提供，用来观察在编程系统中已经登录的运行系统中的事件。

这包括：

- 系统启动和关闭时的事件（装载的组件和它们的版本）
- 应用下载和启动工程下载
- 客户详细条目
- I/O 驱动器的日志条目
- 数据服务器的日志条目

设备对话框，日志



日志条目行包括以下信息：

严重性：有四种类别：警告，错误，例外，信息。每一个类别条目的显示能通过列表上面条目相应的按钮切换为打开或者关闭。每个按钮都包含当前各自类别记录的数目：

时戳：数据和时间，例如"12.01.2007 09:48"

描述：事件的描述，例如 “输入<CmpFileTransfe>函数失败”

组件：组件的 ID 和名称

选择组件名称列表：如果仅仅为了显示与该组件有关的日志条目，这里可以选择一个特殊的组件，，默认的设置是"<所有组件>"。

记录器：选择列表提供了可用的记录。默认设置为“<Default Logger>”，<Default Logger>是由运

行系统定义的，而且通常在 CoDeSys SP 运行系统中和"PlcLog"是相同的。

尚未实现：日志列表由运行系统自动更新。

当前情况下用户可以通过按钮手动更新列表。

此列表可导出到一个 XML 文件中。通过按钮打开标准保存对话框，将列表保存到 XML 文件中。设置文件类型为"xml-files (*.xml)"，在选择的目录中，日志文件以指定的名称+".xml"扩展名为文件名称保存。要查看存储在这些导出 xml 文件中的日志记录，请用按钮打开浏览文件的标准对话框。这里打开文件类型被设为“xml 文件（*.xml）”。选择想要的日志文件，该记录将显示在另一个窗口中。

要删除当前日志表，也就是要删除所有显示的记录，请用按钮。

如果离线日志选项被激活，某些不需要连接到 PLC 的离线操作可以被记录。然而目前仅仅能在 CoDeSys 安全版本编程系统下实现。

PLC 设置

此对话框由“设备”对话框（设备编辑器）的选项卡提供，可以通过“编辑对象”命令或者通过在设备窗口中双击设备对象条目打开。它用于配置设备（PLC）的基本设置。

设备对话框，PLC 设置

I/O 操作的应用：在这里定义，哪个在设备目录中的设备分配的应用应该参照 I/O 映射。开始时标准工程会自动设置标准应用，然而你可以在选项列表中选择另外的应用。

PLC 设置：

在停止模式时更新 IO：如果此选项被激活（默认值），当 PLC 停止工作时，输入输出通道的值也会自动更新。为防止看门狗过期，输出被置为定义的默认值。

停止运行时输出的方式：从选项列表中选择一个下面的选项，以确定在 PLC 停止运行时输出通道的值应该怎样处理：

保持当前值：当前值保持不变。

设置所有输出量为默认值：从映射得到的默认值将被分配。

执行程序：通过工程中一个有效的程序确定输出方式。在这里输入程序名称，当工程停止运行时，该程序将被执行。通过按钮 ，使用输入助手完成该操作。

更新所有设备中的所有变量：如果该选项被激活，那么对于当前PLC 配置中的所有设备，所有 IO 变量将在总线周期任务中的每一个周期更新。这相当于选项“持续更新变量”，该选项可通过“IO 映设”对话框为每个设备单独设置。

总线周期选项：

总线周期任务：选项列表提供了当前有效工程中任务配置里已定义的任务（例如"MainTask", "EtherCAT Master" 等）。选择其中一个任务作为当前工程的总线周期，或者选择选项<未标明的>，该选项意味着将应用最短的任务周期时间，也就是最快的执行周期。您可以切换到另一种设置，但是请务必注意下列事项：

注意：在修改“<unspecified>”设置之前，您已经清楚其影响。“<unspecified>”是指，将应用由设备描述定义的缺省行为。所以请检查这方面的相关描述：缺省方式下，该任务可能被定义为具有最短周期时间，但也可能具有最长周期时间。

附加设置：

生成IO 映射强制变量：只有在设备支持的情况下此设置才有效。如果选项被激活，对于每个在IO 映射对话框被分配了变量的 IO 通道来说，应用程序建立时会为其创建两个全局变量。这些变量可以在 HMI 可视化中用于强制 I/O 值。请查看 I/O 映射 获取详细信息。

I/O 映射

设备配置对话框（设备编辑器）被命名为“<设备类型> I/O 映射”（例如，“DP-模块 I/O 映射”）。它用来在工程变量上映射 PLC，也就是为应用程序的变量分配设备输入，输出地址和存储地址。

应用目前操作的 I/O 被定义在 PLC 设置对话框中。

注意：I/O 映射是否能对当前设备进行设置，依赖于设备的类型；很可能此对话框仅仅用来显示默认设备，见下文：“IEC 对象”。

I/O 映射信息

当前设备是否能进行 I/O 映射配置取决于设备本身，有可能对话框只能显示那些隐式创建的设备实例，请看下述"IEC 对象"

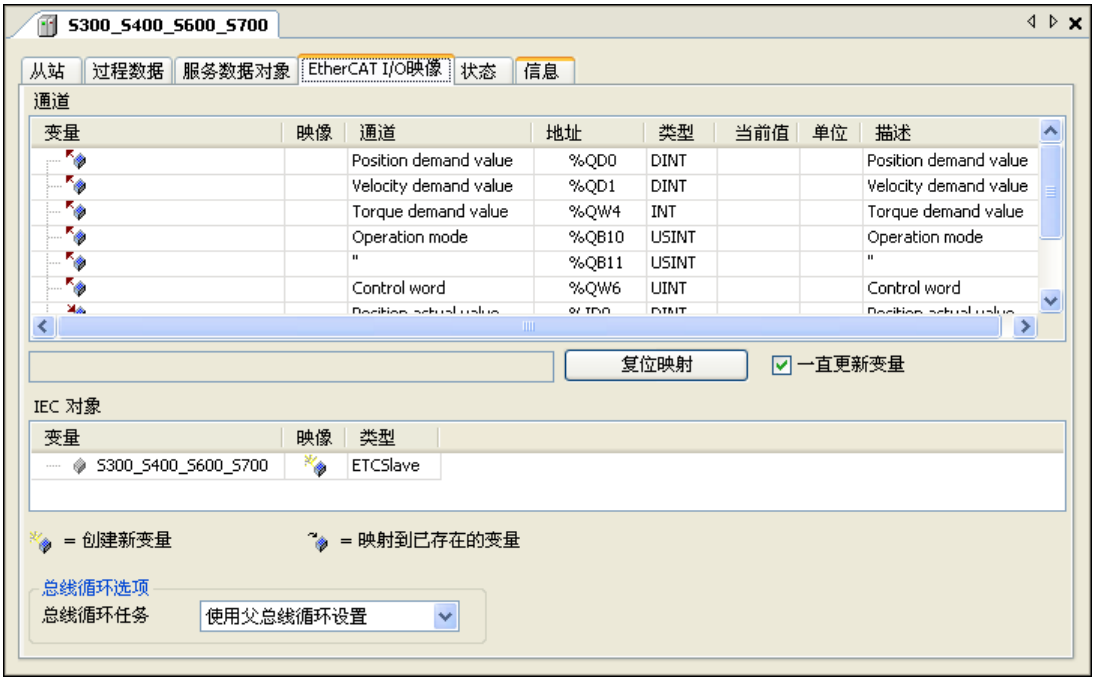
注意除了使用 I/O 映射对话框，通过“AT 声明”也可以为一个变量分配一个地址。变量进行 I/O 映射时，需要参照以下各项：

- 输入变量不能进行写访问操作。
- 请注意下列内容：1.“AT 声明”只用于局部或全局变量，不能用于 POU 输入输出变量。2.生成 I/O 强制变量（请查看强制 I/O 隐式变量）在“AT 声明”无效。3.如果利用 AT 声明一个结构体变量或者功能块成员时，所有实例将访问同一个内存地址，此内存地址类似于高级编程语言如 C 语言里面的静态变量。

◆ 结构体变量的内存分配取决于目标对象。

◆ 对于 I/O 映射对话框中每个被分配到 I/O 通道的变量，“强制变量”会在应用程序编译期间创建。这将会在机器试运行期间强制输入输出值，例如，通过一个可视化（HMI）。请查看下述内容：强制 I/O 隐式变量。

设备对话框，I/O 映射，Profibus 模块实例



使用 I/O 映射对话框

设备所提供的各项条目显示如下：

通道：设备输入或者输出通道的符号名。

地址：通道的地址，例如“%IW0”。可以修改和固定对话框当前显示的输入输出地址，以适应为给定硬件配置编址，在模块顺序有改动（每个默认模块会引起地址值的自动适应）的情况下保持地址值。假设只可以修改输入输出的整体地址，而不是修改子元素，例如：位通道。那么如果显示在映射表中的输入输出有子树，那么只有最上面条目的地址域可以被编辑（请看下面的配置：只有第一行的地址可以打开）。为了固定地址值，选择“地址”栏中的条目，然后按下空格键，打开编辑区域。一个被修改的地址将用一个符号Ⓜ作标记，并且之后的通道地址(将来可手动配　置地址)将会被相应调整。参见以下例子：

例子，手动修改地址

如果想要删除对地址的固定，重新打开地址编辑器，删除此地址条目然后按<Enter>。此地址和下面的相关地址将会设置回手动修改前的值。

类型：输入或输出通道的数据类型，例如“BOOL”型。如果数据类型不是标准类型，而是在设备描

述中定义的结构体或是位域，那么仅仅当它符合 IEC 标准或者在设备描述中用 IEC 类型表示时才会被显示，否则表格中的条目是空的。

单位：参数的单位，例如“ms”毫秒

描述：参数的简要描述。

当前值：参数的当前值，在线模式下显示。

I/O 映射 I/O 必须在“变量”栏里详细说明相应工程中的变量：

通道类型已经通过一个图标显示在变量栏里： 输入， 输出。在各自的表格单元里已经详细说明变量的路径或名称，各自的通道将被映射到表格单元上面。可以配置一个已经存在的工程变量，也可以定义一个需要声明为全局变量的新变量。在映射结构化变量时，编辑器会阻止结构体变量（例如%QB0）和特殊结构元素（例如%QB0.1 和 QB0.2 等）的同时输入。这就是说：当映射表中有位通道条目的子树的主要条目时（如下图所示），在主要条目的一行中或者其子元素（位通道）中都不能输入变量。

- 如果想配置一个存在的变量，你必须输入完整的路径，即"<application name>.<pou path>.<variable name>"；例如"app1.plc_prg.ivar"。推荐使用输入助手，可用 … 按钮。图标 （创建新变量）最初显示在映射栏里，用鼠标点击该图标，将变量映射到 IO 地址上，同时图标变成 。完成后，地址下面将出现删除线，这不意味着此内存地址不再存在，而是不直接被使用，已有变量的值由其他内存位置来管理，尤其是输出变量，其他任何已定义的变量也都不能被存放到这个地址（也就是在 I/O 映射中的%Qxx），以避免写入值时造成模糊性。

见下面例子，输出映射在已经存在的变量上：

- 如果想要映射在一个新变量上，只要输入目标变量名称；例如："bVar1"。此图标 将会在映射栏中显示。通过鼠标单击此图标变量将会被声明为全局变量，同时此图标会改变为 。从那开始此变量在应用程序中可用作全局变量。所以映射对话框是声明全局变量的另一种方式。

重置映象：使用这个按钮将重置映象为设备描述文件定义的默认值。

持续更新变量：如果此选项被激活，所有变量无论他们是否被应用或者是否映射在输入或者输出上，都将在每个总线任务周期中被更新。

提示：一个地址在程序代码中可以赋值给其余变量，也可以被赋值。以 ST 为例：

```
Var := IB100;
QX12.3 := TRUE;
```

建议在设备配置对话框里执行类似的可行的改动。

IEC 对象：

仅当存在设备对象时，I/O 配置对话框才可用，可以被应用程序访问（例如为了重启总线或者取出

信息）。此设备是否可用和如何使用，依赖于设备的类型，在详细设备配置里的在线帮助栏有描述。

总线周期选项：

该设置选项将在读输入输出之前和之后对有循环调用的设备有效。它允许设置一个设备总线周期任务：

缺省情况下，“父”总线循环被设置为有效（使用父总线循环设置），这样将在设备树中搜索下一个有效总线循环任务。

从选择列表中选择你所想要指派的总线任务。该列表提供了当前在应用程序的任务设置中定义的任务。

强制 I/O 隐式变量

在一个工厂或机器（如 HMI 可视化）试运行期间，会需要强制 I/O。可以为每个 I/O 通道生成特殊“强制变量”，它映射在设备编辑器的 I/O 映射对话框中的变量上。

设置生成IO映射强制变量必须在PLC设置对话框中激活。然后在每次应用程序编译时，每个映射的 IO 通道，将有两个变量按照下列语法生成，这里所有空的位置都将由下划线替代：

- 激活与撤销强制："设备名“_“通道名“_“IEC 地址“_“BOOL 类型强制",
- 定义通道上的强制值："设备名“_“通道名“_“IEC 地址“_通道数据类型“值"

这些变量在输入助手的"变量","IoConfig_Globals_Force_Variables"类中有效。它们可用于任何可编程系统中的编程对象、可视化和符号配置等等中。

"强制"变量的上升沿将各自的 IO 强制成由”值”变量定义的值。下降沿使得强制失效。在新值被强制之前必须将强制变量取消设置到 FALSE！

参考下列的约束。

示例：

如果定义了映射，像上图"设备对话框，I/O 映射，示例"一样，下列变量会在编译（F11）应用程序时生成，并在输入助手中有效。

- Digitax_ST_Control_word_QW0_Force：BOOL;

- Digitax_ST_Control_word_QW0_Value : UINT;

- Digitax_ST_Target_position_QD1_Force : BOOL;

- Digitax_ST_Target_position_QD1_Value : DINT;

- Digitax_ST_Status_word_IW0_Force：BOOL;

- Digitax_ST_Status_word_IW0_Value : UINT;

- Digitax_ST_Position_actual_value_ID1_Force：BOOL;

- Digitax_ST_Position_actual_value ID1_Value : DINT

约束

- 只有在 I/O 映射对话框中映射到变量的通道（例如：“变量”栏中必须定义变量，无论是新建的还是已存在的）才能被上述隐式变量强制。

- 未使用的输入/输出以及应用程序中通过 AT 声明映射的不能被强制。
- 每个 IO 通道至少要在一个任务中被使用。
- 强制 I/O 不在监视器（监视视图，I/O 映射对话框等）中显示。这些值只在 IO 驱动写入设备时被隐式使用。
- 强制输入由红色强制符号F表示，不是强制输入-/输出。

用户和组

设备支持“在线用户管理”，此对话框可以通过“设备”对话框（设备编辑器）的一个选项卡打开。它允许设置用户帐户和用户组，和访问权限管理一起控制在线模式下 PLC 对象的访问权限。

普通

为了安全起见控制器的某些功能仅能被授权用户执行。“在线用户管理”提供了设置用户帐户，配置用户组的访问权限和登陆时强制用户身份验证。

设备用户管理能通过设备描述预先定义，而且它也取决于在编程系统配置对话框里定义的描述。

和工程用户管理一样，用户必须是组的成员，而且只有用户组能获得访问权限。

配置对话框的使用

基本上用户管理对话框的操作和工程用户管理对话框的操作很相似。同样也可能从工程用户管理中“导入”用户帐户定义。

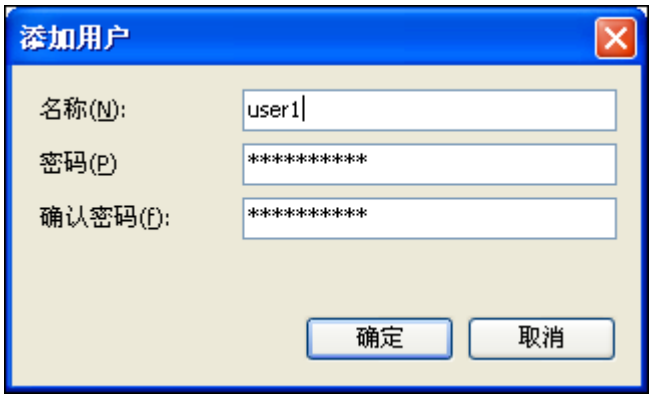
设备对话框，用户和用户组

用户：

以下按钮用来设置用户帐户：

添加...你可以通过“添加用户”对话框定义一个用户的名称和密码。密码必须在确认密码区域再次输入确认。

添加用户对话框



导入...“导入用户”对话框显示所有当前在工程用户管理中定义了的用户名称。为了获得设备描述用户管理列表中的信息， 可以选择一个或者多个条目，点击确认按钮，在打开的输入密码对话框中输入工程的用户管理中对应的用户密码。鉴于安全原因用户的密码不会被导入，需要重新定义。

注意：每一个被导入的用户帐户基本上密码都是空的！

编辑...可以修改当前被选中的用户帐户的用户名和密码。这个“编辑用户<user name>”对话框是和“添加用户”对话框（见上文）相对应的。

删除删除当前选中的用户帐户

用户组：

添加...可以通过“添加用户组”对话框定义一个新组名称并且从当前定义了的用户中选择加入该组的成员。

添加用户组对话框

“导入用户组” 对话框列出了当前在工程用户管理中定义了的用户组。你可以选择一个或者几个用户组，然后使用“确认“按钮确认来得到它们在设备详细用户管理用户组列表中的完整信息。

可以修改当前被选中的用户组的组名和相关用户。为此可以打开“编辑用户组 <group name>”对话框进行修改，与“添加用户组”对话框相对应。

删除当前被选中的用户组

应用并且存储当前配置：

见对话框顶栏各自的按钮：

：当前用户管理配置必须下载到设备里才能生效。设备当前应用的配置能上传到配置对话框。

：当前配置能以“xml”文件形式（*.dum）存储，然后从文件中重新装载，导出的文件用来在多重系统中设置相同用户的配置。为此将提供用于浏览文件系统的对话框。文件过滤器会自动设置成"*.dum"文件格式，"*.dum"意思是“设备用户管理”文件。

通过使用"打印..." （文件菜单中的命令）或者"打印文档..."（工程菜单中的命令），也可以将这些设置打印成文档输出。

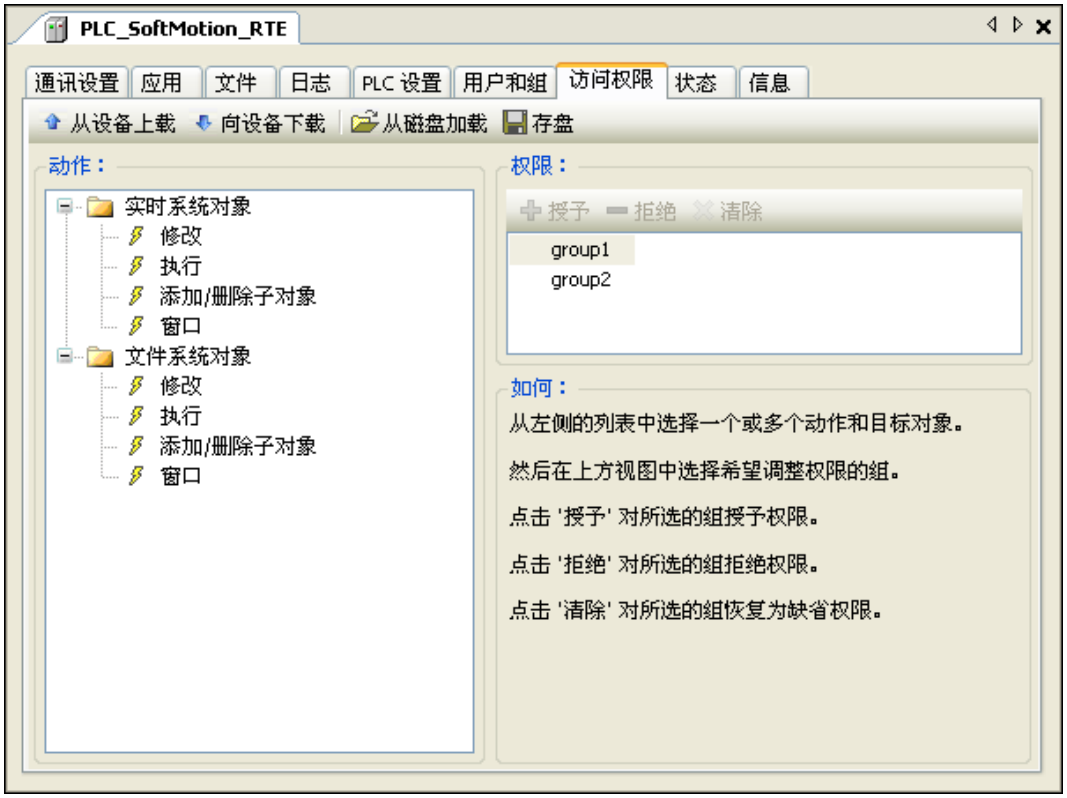
访问权限

此对话框可通过“设备”对话框（设备编辑器）的一个选项卡打开。它是“在线用户管理”的一部分，用于授权或禁止当前定义了的用户组的权限，这样定义在 PLC 运行时用户文件或对象（比如一个应用）的访问权限。

注意权限只能分配到用户组，而不是特定用户。因此用户必须是一个用户组的成员。用户和用户组的配置是通过设备编辑器的“用户和组”选项卡实现的。

参见下图：从 PLCLogic 对象中将添加或删除子对象的权限授予用户组

设备对话框，访问权限



下文介绍如何定义访问权限和如何将它们下载到设备 上，或者存储在一个可重载的文件里。

定义访问权限

要定义在一个或多个对象上执行动作的权限，必须在 “动作”窗口中的动作类型下选择对象条目，然后在“权限”窗口中选择想要的用户组，然后点击授权或禁止按钮（同样在权限窗口中）。

参见帮助窗口中的说明：

动作：

此部分列出了运行时在 PLC 文件系统或运行对象（例如应用）可执行的操作。该目录的结构如下：

对象类别：

在第一级，有两个文件夹，分别是文件系统对象文件夹和运行对象文件夹。

动作类型：

总是有一些节点用于四种类型的动作，这些动作可以在特定的对象上执行。节点的设置，也仅仅是为了结构化的目的。

添加/删除子对象（在现存的对象上添加或者删除“子”对象）

执行（例如开始/结束应用，设置断点等。）

修改（例如下载应用...）

视图（监视）

对象（动作“目标”）：

在每个动作类型节点下面找到动作的“目标”，即 PLC 文件或者运行对象，例如设备。这些对象条目在设备目录或者文件系统结构下以树形结构显示。

注意：当在对象目录中分配给“父”对象一个访问权限时，通常意味着它的“子”对象也继承了此权限，只要它们没有明确自己的定义。不过这可能依据设备不同而以不同方式处理，所以最好先核实一下。总之，继承在此对话框中是不可视的。

权限：

此区域显示了当前定义了的用户组。在各个组前面下列每个图标显示了分配给目前在动作窗口被选择的对象的权限：

：该组具有在当前“动作”窗口选定的操作权限。

： 该组不具有在当前“动作”窗口选定的操作权限。

：当前对于“动作”窗口中被选中的动作没有明确的访问权限定义。

如果多个动作在“动作”窗口里被选中，它们相对当前被选中的组来说没有独特的设置，将无图标显示。

按钮栏：在“动作”窗口里选中了想要的对象、想要进行的动作，以及在权限窗口中选中了想要的组之后，可以使用下列按钮：

Grant：明确授予访问权限。

Deny：明确禁止访问权限。

Clear：对当前在“动作”窗口选中的动作授予的访问权限会被删除，即设为默认值。

应用和存储当前配置：

见对话框的顶端各自的按钮：

向设备下载 从设备上载：当前配置的访问权限必须下载到设备里才能生效。当前应用在设备上的配置能上传到配置对话框。

存盘 从磁盘加载：当前的配置能以“xml”文件形式（*.dum）存储，然后从此文件中重载，导出的文件用在多重系统中设置相同用户的配置。为此将提供用于浏览文件系统的标准对话框。文件过滤器会自动设置成"*.dum"文件格式，"*.dum"意思是“设备访问权限”文件。

通过使用"打印..." （文件菜单中的命令）或者"打印文档..."（工程菜单中的命令），也可以将这些设置打印成文档输出。

状态

此对话框由“设备”对话框（设备编辑器）里的一个选项卡打开。它显示各自设备的状态信息（例如“运行”、“停止”）和详细的诊断信息；同时也用在主卡和内部总线系统。

信息

此对话框可由“设备”对话框（设备编辑器)）的一个选项卡打开，它能通过“编辑对象”命令，或者双击设备窗口上的设备条目打开。

如果当前设备可用，将会显示下列基本信息：名称、供应商、类型、版本号，订单号、描述、映像。

2.7.2 CAN 总线配置编辑器

根据 CANopenDS301 协议，CAN 总线设备编辑器插件提供对 CANopen 设备的组态配置。专用 CAN 总线要求兼容支持标准 CAN 总线。总线配置对话框要添加到标准设备编辑器对话框中。为支持不同的 CAN 总线硬件，需要在设备树中添加合适的对象。

CAN 总线设备在交换信息的过程中获得描述它们的时间错误依赖行为的配置。为特定模块生成过程数据对象（PDO）的输入输出映射，并发送和接收过程数据的服务。然后生成一个服务数据对象（SDO）的列表用来传输请求和响应服务，也就是 CAN 总线目标系统的完整配置数据。通信和过程数据对象是在“对象库”中进行管理的。

CANopen 设备的初始配置由当前主从设备描述文件决定，通过配置编辑对话框对其修改。对于标准的应用程序和设备，推荐使用自动配置模式。

因此，构造一个 CAN 总线配置的设备树，需要做以下工作...

1、安装相应的描述文件。

2、如果 CANbus 和 SJA 芯片集成在一起，CoDeSYs 配置文件（...\GatewayPLC\CoDeSysSP.cfg）需要进行如下扩充：Component.<subsequent number>=CmpSJACanDrv

3、在设备树中插入不同设备并分别映射 到总线结构中。

4、在设备编辑器中设置合适的配置参数。

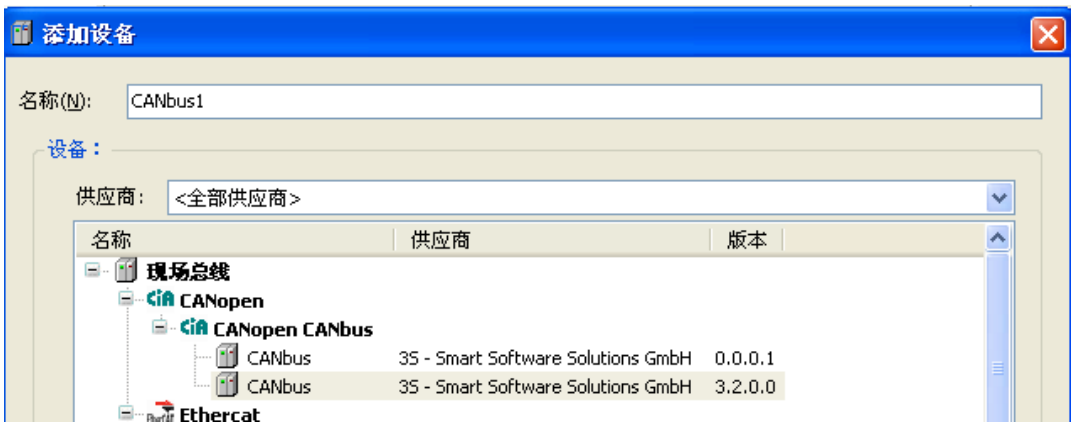
安装 CANopen 设备：

首先，通过设备库在本机系统中安装相应的设备描述，然后将CAN 总线硬件映射到设备树上。这关系到 CAN 总线和 CANopen 管理器的基本模块描述文件(*.devdesc.xml)，和总线系统中 CANopen 设备的出厂 EDS 文件（电子数据表），这些描述文件的所有默认值都是由 CAN 总线配置插件提供的。这里的每个文件都包含了模块的参数定义和配置选项。

CoDeSys 中 CAN 总线基础配置/插入 CAN 总线设备：

◆ CAN 总线配置树的根节点必须是“CAN 总线”对象。CAN 总线对象只能在应用程序的设备节点处插入。在配置树中选择设备节点，弹出“添加设备”对话框，选中 “CAN 总线”，添加 CAN 总线对象。

添加 CAN 总线对象，如下图所示：

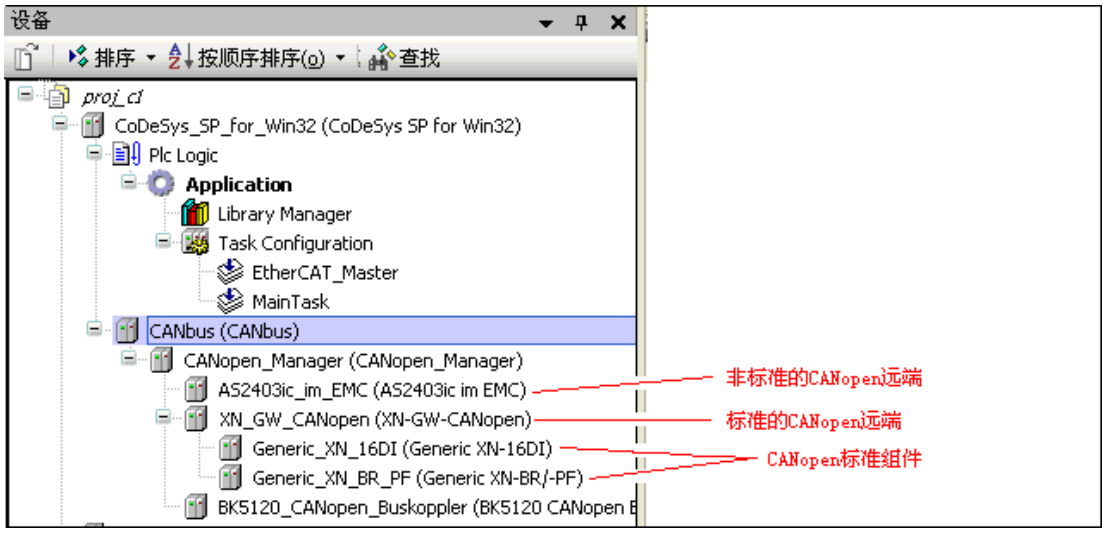


◆ 在 CANbus 节点上，只能配置总线传输数据波特率。

◆ 在配置树的 CANbus 节点下必须插入有效的“CANopen 管理器”。它相当于 CAN 总线的主站。通过内部的函数支持 CAN 总线的配置，用户无需再定义配置。CANopen 管理器简化了内部网络连接，特别是简化了专用模块间的 PDO 连接。

◆ CANopen 管理器下可添加特定 CANopen 设备硬件（如“CAN 总线子站”）。有两种类型的设备可以接入：非模块化设备和模块化设备。模块化设备可接入 CAN 总线模块，并自动完成 I/O 映射。

CAN 总线配置如下图所示：



配置

完成设备树与 CAN 总线硬件的映射后，使用配置对话框选中或修改默认设置：

选中设备树的设备节点，双击或者使用 “编辑对象”命令，在设备编辑器的窗口弹出应用程序配置对话框。

设备或者模块的配置通常依赖设备描述文件的定义。

需要注意设备编辑器选项的常规设置，例如，通用对话框被设置为不可见的。

CANopen 配置对话框通常由模块来命名，它包含如下子对话框：

CAN 总线：

- CAN 总线（常规设置，目前只支持波特率设置）
- CANbus 总线配置（由*.devdesc-file 提供）
- CANbus 总线 I/O 映射
- 状态
- 信息

CANopen 管理器：

- 设备编辑器的标准对话框是可用的，但通常并不配置它们，因为目前 CANopen 管理器的设备描述不包含任何参数（CAN 总线常规设置，CANopen 管理器，CANopenI/O 映射，状态，信息）

CANopen 设备：

CANopen 远程设备（主模块设置）

- PDO 映射
- 接收 PDO 映射
- 发送 PDO 映射
- 服务数据对象
- CANopen 设置（参数由 EDS 文件定义）
- CANopen I/O 映射
- 状态
- 信息

CANopen 模块：

- CANopen 模块设置（参数由 EDS 文件定义）
- CANopen 模块 I/O 映射
- 状态
- 信息

CANopen 设备接口

从 IEC 应用访问 CANopen 设备，可以通过 IIoDrvBusControl.Library 库提供的接口实现。

CAN 总线节点

CAN 总线

CAN 总线的节点对象必须是 CAN 总线配置设备树中的根节点。

在相应的配置对话框中定义了以下参数：

波特率（bit/s）：总线上传输数据的波特率。它的单位在节点配置对话框中被定义为位/秒。波特率的默认值是由 CAN 总线设备的描述文件（*.devdesc）设置的。

波特率可设为以下值：10000，20000，50000，100000，125000，250000，500000，800000 和 1000000，单位是位/秒，在配置数据时被存储为千位/秒。

网络个数：通过 CAN 总线接口连接的 CAN 网络个数。可输入值 0 到 100。

CAN 总线配置

CAN 总线节点的配置对话框目前是无用的，因为除波特率和网络个数外，设备描述文件没有提供其他配置参数。

CAN 总线 I/O 映射

I/O 映射的总体描述和使用。

状态

CAN 总线设备或模块的状态配置编辑器可显示状态信息（如：“运行”，“停止”）、设备专用的诊断信息以及卡的使用情况和内部总线系统的状态。

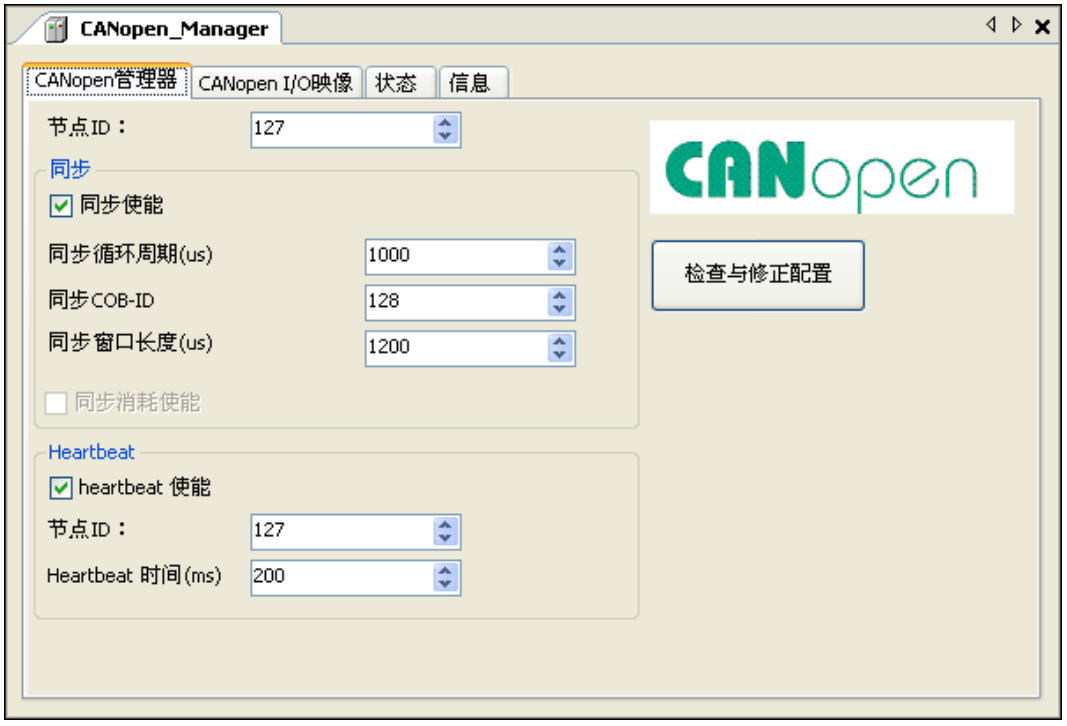
信息

CANopen 设备或模块的信息对话框显示设备或在设备树中被选中模块的信息。若当前选中模块是可用的，将显示以下信息：名称，供应商，类别，版本号，顺序号，描述，图像。

CANopen 管理器

在 CAN 总线配置中，CANopen 管理器是 CAN 总线节点的子节点。它通过内部函数支持 CAN 总线配置，通常被用作 CAN 总线的主站。

CANopen 管理器对话框



CANopen 管理器对话框目前可以为总线上的 PDO 传输启用同步模式，并设置一些默认参数，这些参数可以被常规 CAN 总线配置修改。

同步：

同步使能：若该模式被启用（默认为禁止），总线上的数据传输是同步的，以下的默认值可以修改：

同步循环周期（单位：微秒）：同步信息以同步循环周期定义的时间间隔发送，同步循环周期的单位为微秒。范围：[100, 232-1]。

同步 COB-ID：通信对象标识符，标示同步信息。范围：[1,2047]。

同步窗长度(单位：微秒)：用于同步 PDO，以微秒为单位的时间窗长度。范围：[1,232-1] 或者 0（如果不使用的话）。

同步 PDO 在同步信息传输完毕后立即传输。

同步接收使能：若该模式被启用（默认为禁止），CANopen 管理器将接收到另一个设备发送的同步信息。

注意：CANopen 管理器启用同步传输模式将使所有其他总线客户端的周期选项失效。

脉冲：

脉冲工作方式是除了节点保护功能外的另一种保护机制。这种方式通过主从站模块来执行。主站通常用来发送请求脉冲到从站。新增加的从站将被自动配置合适的脉冲时间。

Heartbeat 使能：启用此选项，主站将根据内部定义的“脉冲时间”发送脉冲请求。如果增加了有脉冲功能的新从站，它们的脉冲行为会被自动地使能和组态，即节点 ID 和管理器组态中的设置一致，但是脉冲间隔变为 1 倍或2 倍。如果没有使能CANopen 管理器的脉冲，那么会使能节点守护来代替(生命时间因子为 10，守护时间为 100ms)。注意，CANopen (从)设备可以被组态成脉冲发生器。

节点 ID：总线中的每个脉冲发生器必须有唯一的编号。范围（1-127）。

Heartbeat 时间（毫秒）：脉冲发送间隔，单位为毫秒。

当在 CANopen 管理器中增加了多种设备时，用户可能会得到报错信息，这些信息包含被多次重复分配的节点 ID 或者来自设备描述文件的错误 COB-ID。点击按钮“组态检查与修改”，可在打开的对话框内解决这些问题：

对话框 – 组态检查与修改

重复的节点 ID：该框中包含具有相同 ID 的所有设备的列表。用户可以通过编辑节点 ID 一栏来重新分配新的节点。

错误的 COB-ID(只显示已被使能的 PDO)。不需考虑公式中第 31Bit：从设备描述文件中自动创建的 COB-ID 不一定正确。这里列出所有错误条目，每个条目包含相关的设备名称，节点 ID 和索引。有三种方式可用来修改这些 COB-ID：

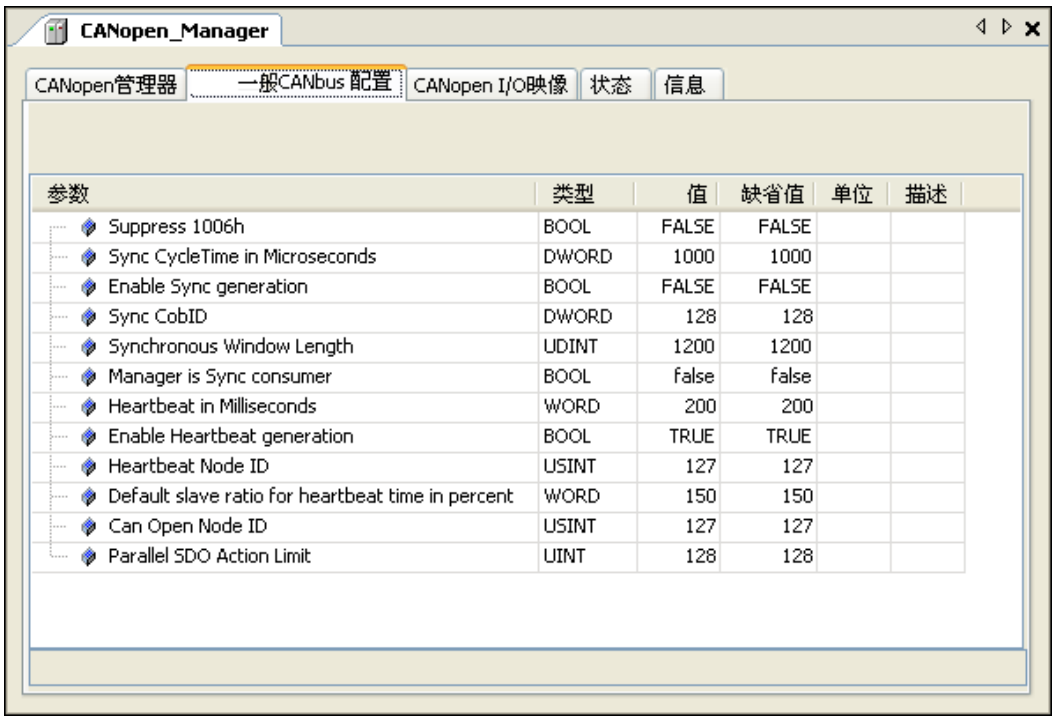
- 编辑所列公式得到有效的 COB-ID。可以在相应的表中直接编辑。
- 或者点击列表最后一栏给出的“自动建议”。
- 或者通过点击“使用全部建议”按钮，立即使用所有自动建议(在最后一栏中显示)。

修正项将自动地从列表中被删除。

CAN 总线常规配置

只有在“显示常规设备配置视图”选项被激活的情况下，CANopen 管理器的CAN 总线常规配置对话框才会在设备编辑器中显示。设备编辑器提供的相应配置对话框也适用其他设备。

CAN 总线常规配置对话框如下图所示：



设备的专用参数作为表格列出，这些参数是由制造商提供的设备描述文件定义的，如果描述文件允许，这些参数可修改。多数参数可以在 CANopen 管理器对话框中设置。下面的参数只能在“一般 CANbus 结构”配置对话框中获取：

Suppress 1006h：如果为 TRUE 时，即使同步使能被激活，对象 0x1006 也不可写。如果为 FALSE 或不可用， 对象 0x1006 可写，就像之前的版本中那样。缺省：FALSE。

缺省从设备 heartbeat 时间比率 (参数 ID 275)： 如果 CANOpenManager 的使能 heartbeat 产生被激活，该值会被用于计算从设备缺省的

Heartbeat 消耗时间。缺省：150

可选择的设备缺省(参数 ID 276)：指定从设备配置对话框中可选择设备复选框的缺省值。缺省：FALSE

ParallelSDOActionLimit：可同时被处理的 SDO 动作个数，也就是CANopen 管理器 SDO 通道个

数。

它关系到配置用到的 SDO 以及应用要求的 SDO。缺省：128。

CANopen I/O 映射

I/O 映射的描述和使用。

状态

CAN 总线设备或模块的状态配置编辑器可显示状态信息（如：“运行”，“停止”）、设备专用的诊断信息以及卡的使用情况和内部总线系统的状态。

信息

CANopen 设备或模块的信息对话框显示设备或在设备树中被选中模块的信息。若当前选中模块是可用的，将显示以下信息：名称，供应商，类别，版本号，顺序号，描述，图像。

CANopen 设备

非模块化和模块化设备

在 CAN 总线配置中，CANopen 管理器节点可接入两种类型的 CAN 总线设备：

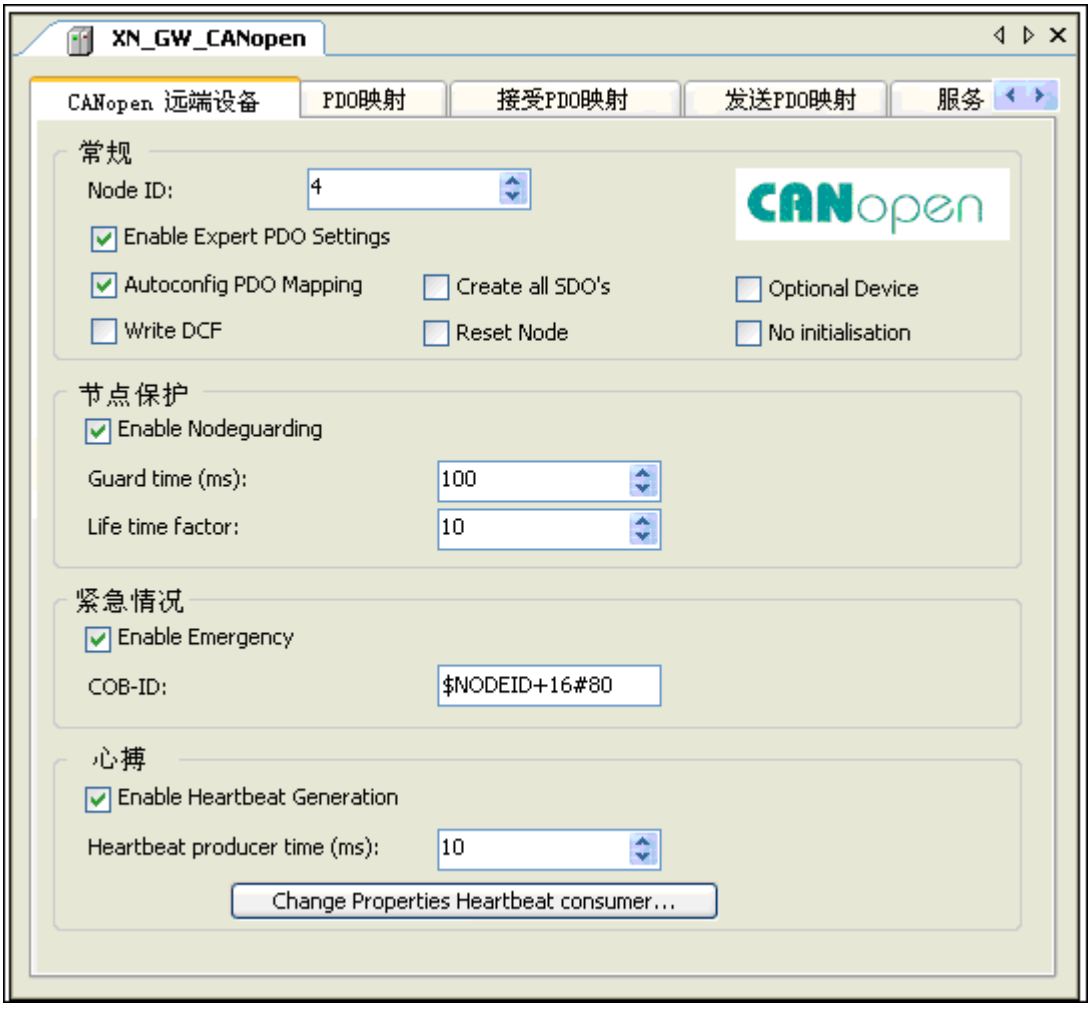
模块化设备允许接入 CAN 总线模块。专用从站模块自带映射列表，不需要使用 CANopen I/O 映射对话框。标准应用程序推荐使用模块化设备的 CANopen 远程设备对话框的“自动配置 PDO 映射”选项激活自动映射配置。

非模块化设备不能接入远程模块。因此，与模块化设备相比，它的配置对话框包含 I/O 映射对话框 且没有自动映射配置的选项。

CANopen 远程设备

在 CANopen 设备配置对话框中，可设定设备的节点编号（1-127）。选中“启用专业 PDO 设置”选项后，该对话框将按照设备描述文件（EDS 文件）提供的参数放大显示 CANopen 模块各主要默认参数。用户可修改这些参数。

CANopen 远程设备配置如下图所示：



常规：

自动配置 PDO 映射：启用这个选项（强烈推荐标准应用程序使用），PDO 映射（接收 PDO 映射和发送 PDO 映射对话框），将在设备描述文件定义的基础上自动生成，且不能在映射对话框中修改。非模块化设备没有此选项。

节点编号：CAN 总线模块在总线上的唯一标识，需与模块本身数字对应。设定范围（1-127）。输入的编号只能是十进制数。

编辑 DCF：目前无效。

创建所有 SDO：所有的对象，无论默认值是否变化，都要下装到 PLC 中。

重置节点：由设备描述文件决定是否激活。激活此选项，从站下装配置前将恢复默认设置。这导致了从站的 CANopen 通讯参数重置为他们在下装配置前的默认值。这依赖于参数可被完全重置的设备，个别配置通过从选项框后面的选择列表中选择子索引来完成。例如，默认 CANopen 通讯设置是由子索引 2（1000h to 1FFFh, object 1011h）来编址的。

可选设备：激活此选项（对话框的实用性由目标对象决定），主站将只尝试读取节点一次。如果节点没有响应，主站将忽略此节点不再读取，也就是说，主站将回到正常的操作模式。

未初始化：激活此选项（对话框中的实用性由目标对象决定），主站将不发送配置 SDO 而直接激活节点。（不过，SDO 数据将被创建并保存在控制器上。）

节点防护：

启用节点防护：激活此选项，信息根据防护时间间隔（默认 100 毫秒）发送到模块 。如果模块没有给出包含特定防护 COB-ID（通信对象标识）的响应，它将被置为“超时”状态 。超时一定次数以后（运行超时次数，如果设备配置文件中没有其它的默认设置，或默认设置为 0，则一般默认为 10 次），模块将被置为“不正确”状态。这时模块的状态将被保存在诊断报告中。如果没有定义防护时间和运行超时次数（0），则不检测模块。

紧急情况：

当内部出现错误的时候，模块将发送包含特定 COB-ID 的紧急信息。每个模块的紧急信息都不一样，这些信息被保存在诊断报告中。

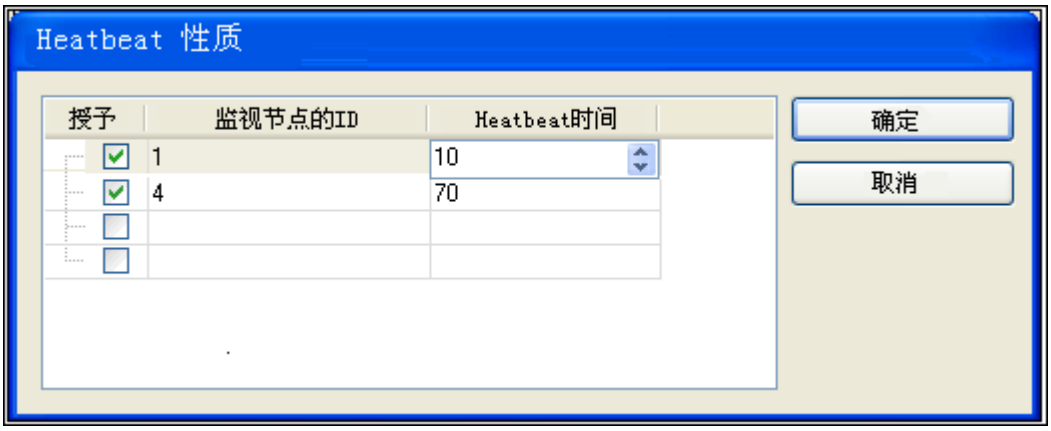
脉冲：

脉冲协议是错误监测协议。脉冲发生器周期发送低优先级信息。信息被各种脉冲放大器接收并放大。如果放大器没有接收到脉冲，它将发送一条特定信息。

启动脉冲发生器：激活此选项，模块将根据脉冲发生器时间间隔（如果设备配置文件中没有其他的默认设置或默认设置为 0，则一般默认为 10 毫秒，）发送脉冲。

变频脉冲放大器：此按钮打开的对话框中列出由 EDS 文件定义的节点，并可用于为节点启用防护。勾中 Enable 列启用防护，在脉冲时间列可输入合适的时间，以毫秒为单位（双击此区域，可打开时域选择框）。激活脉冲放大器选项后，各模块将接收各自主站发送的脉冲。若接收不到脉冲，模块将关闭 I/0 通道。

脉冲属性对话框如下图所示：



注意：主站（CANopen 管理器）将为新接入的具有脉冲功能的设备自动调整脉冲设置。

PDO 映射

CANopen 设备配置的 PDO 映射对话框是所有目前被定义的发送和接收 PDO 的一种简化显示。

若 PDO 有写权限（设备描述文件中的定义，默认：写权限），则可选择是否激活 PDO。模块化的设备只有在自动配置模式禁用时才有此选项。

双击 PDO 节点，弹出 PDO 属性对话框，可修改 PDO 的传输模式。

在接收 PDO 映射和发送 PDO 映射对话框中可修改更多 PDO 配置。

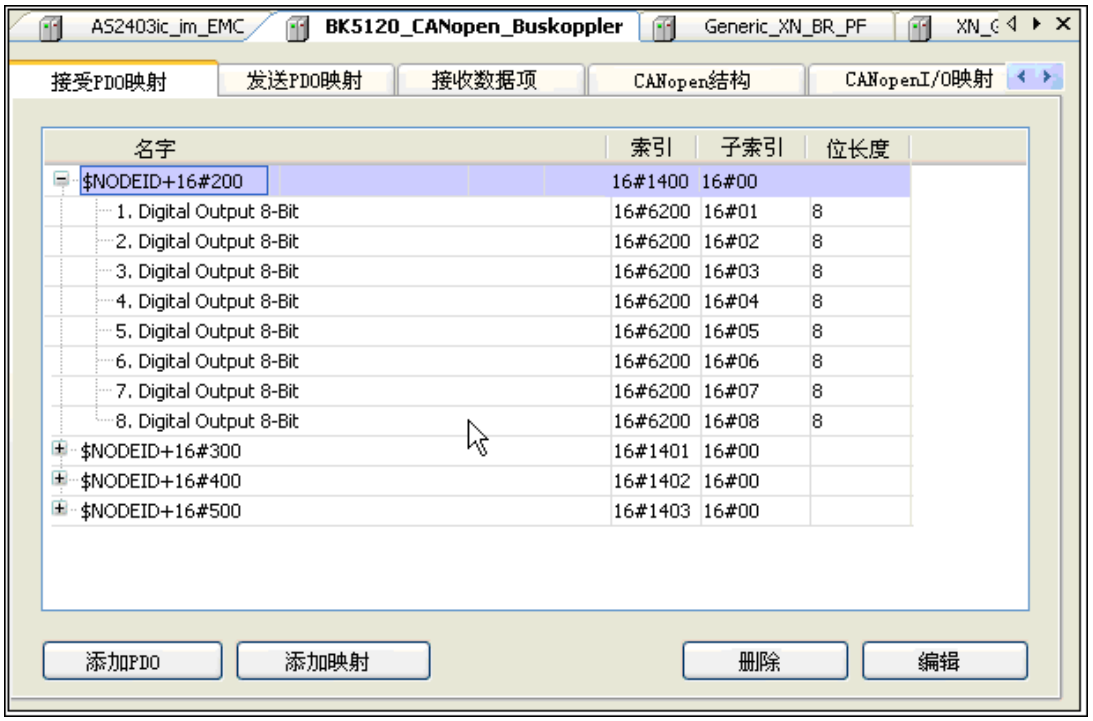
接收 PDO 映射

只有在选项“启用专业 PDO 设置 ”被激活的情况下这个对话框才有效。

接收 PDO 映射，即用于 CAN 总线数据交换的 PDO 输出（对象库）在 CANopen 设备配置 的接收 PDO 映射对话框中定义。与 IEC 地址对应的配置在 I/O 映射表中显示。

该对话框显示 PDO 并缩进显示 PDO 各自的映射。在名称列中同时显示由设备描述文件定义的 PDO 的 COB-ID 和在对象库中使用的映射对象。映射对象的索引、子索引和字节长度分别在相应列中显示。

接收 PDO 映射对话框如下图所示：



双击 PDO 节点，弹出 PDO 属性对话框，显示下面描述的参数。

如果选项 自动配置 PDO 映射 被激活 ...

标准应用程序强烈推荐激活此选项，激活后，对话框中显示的PDO 映射是根据设备描述文件的定义自动生成的，且不能被修改。这样所有相关参数（例如数字量输入和输出）将根据定义好的优先顺序被映射。

如果选项 自动配置 PDO 映射 未被激活 ...

可手动完成映射配置。可修改现有 PDO 或定义新的 PDO，并建立合适的映射。选中 PDO，点击删除按钮，相应映射将从配置中移除。

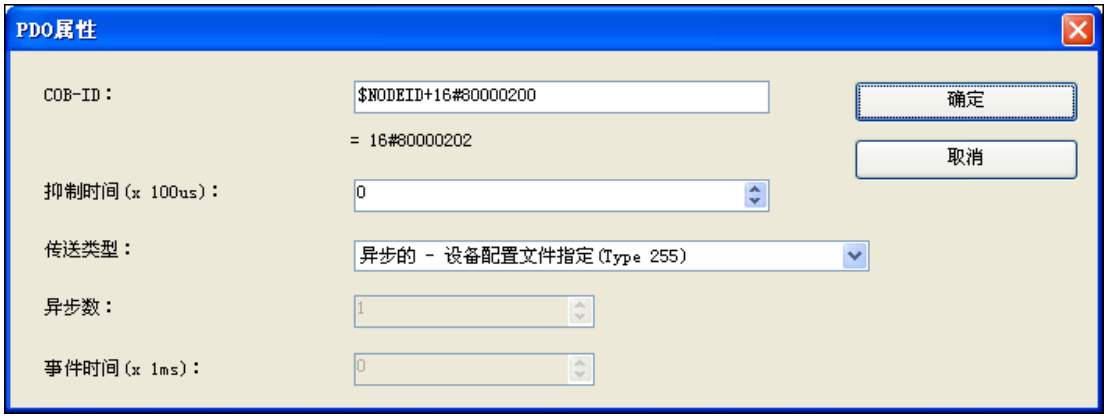
修改现有的 PDO 映射

选中表中的 PDO，单击“编辑”，弹出对话框，可增加新映射或修改现有映射。

添加新的 PDOs 和映射

单击“添加 PDO”，弹出 PDO 属性对话框，可新增 PDO 并修改 PDO 的节点。

PDO 属性对话框如下图所示：



COB-Id：每条 PDO 信息都必须拥有一个唯一的 COB-ID（通信对象标识）。COB-ID 区域显示由 EDS 文件定义的 COB-ID 的当前值。COB-ID 通常在模块的节点编号基础上产生，用“$NodeID”代替实际的节点编号。例如：“$NodeID+0x200”。用户可修改当前节点。

禁止时间（x100μs）：只有当设备支持以下功能：禁止时间是同一个 PDO 传输两条信息之间的最小间隔时间与 100μs 的乘积，该值域才可修改。例如：这个设置可避免当一个值改变的时候PDO 发送太频繁。默认值为 0，范围 0-65535。

传输类型：可设置传输模式。可能的设置如下：

非循环—同步传输：PDO 同步传输，但不是周期循环传输（默认设置）。

周期—同步传输：PDO 同步传输，位于 PDO 两次传输之间的同步信息编号由同步编码器提供。

同步—RTR 传输：PDO 在传送同步信息后更新，但并不传输。只在有明确传输请求时传输（远程传输请求）。

异步—RTR 传输：PDO 只在有明确传输请求时更新并传输（远程传输请求）。

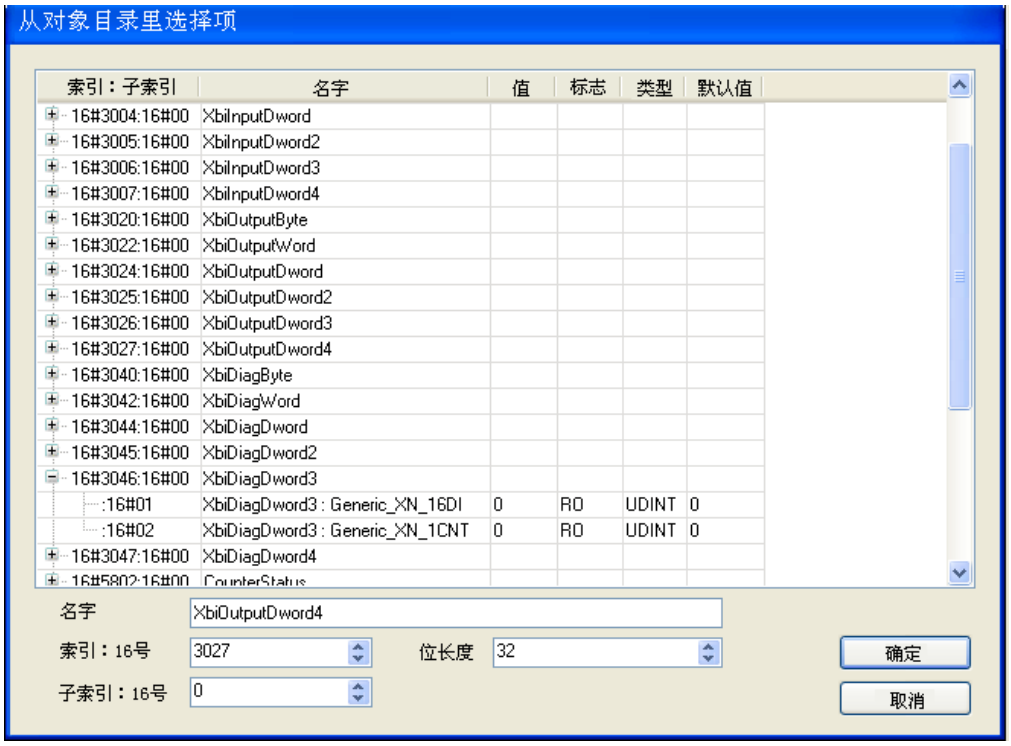
异步—设备详细信息和异步—详细制造商：PDO 只在特殊事件发生时传输。

同步编码器：启用周期传输模式时，在此设定在两次PDO 传输之间被传送的同步传输信息的编号(见 CANopen 远程设备对话框中的‘常规通信周期 ’区域)。默认值为“1”。范围：1-240。

事件—时间（x1ms）：只有当设备支持以下功能：合适的传输模式下，可设定两个传输之间的时间间隔（单位：毫秒），该值域才可修改。默认值为“0”，范围：0-65535。

选中 PDO 节点，单击“添加映射”，弹出从对象库中选择项对话框，可为 PDO 添加合适的对象。

从对象库中选择项对话框如下图所示：



该对话框列出设备的所有由 EDS 文件定义的映射节点。在节点处单击“+”可展开下拉菜单。接收 PDO 中只写权限的对象（只写）和发送 PDOs 中只读权限的对象（只读）不能通过单击“+”展开下拉菜单来列出。每个对象显示以下信息：索引，子索引，名称，值，访问权限（RW=可读写，RO=只读，WO=只写，CONST=不变且只读，等等），数据类型和默认值。

通常在对话框下面的区域显示当前选中节点的名称、索引、子索引和字节长度，并可修改。

发送 PDO 映射

该对话框只有当设备的“启用专业 PDO 设置”选项被激活时才可用。

“CANopen 设备配置 ”的发送PDO 映射对话框定义PDO 发送的映射。即：用于CAN 总线数据交换的 PDO 输入（对象库入口）。

该对话框的描述与接收 PDO 映射对话框的描述相对应。

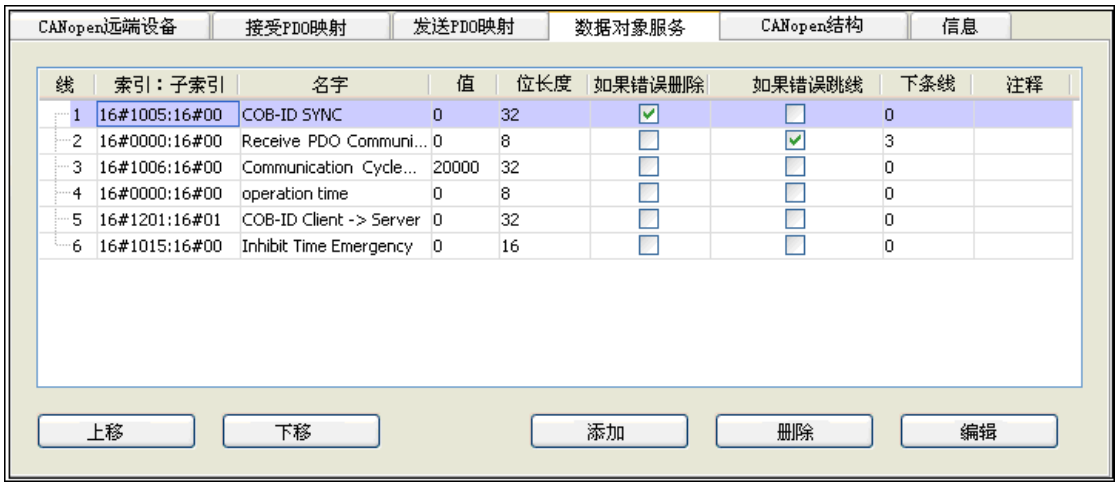
服务数据对象

在 CAN 总线初始化过程中，当前配置值通过服务数据对象传输到 CAN 模块。

服务数据对象配置对话框可配置选中的SDO，修改 SDO 的传输顺序，并定义SDO 在传输过程中出现错误时的处理方式。

注意：若当前设备的“启用专业 PDO 设置”选项未被激活，则该对话框只列出用户定义的 SDO。

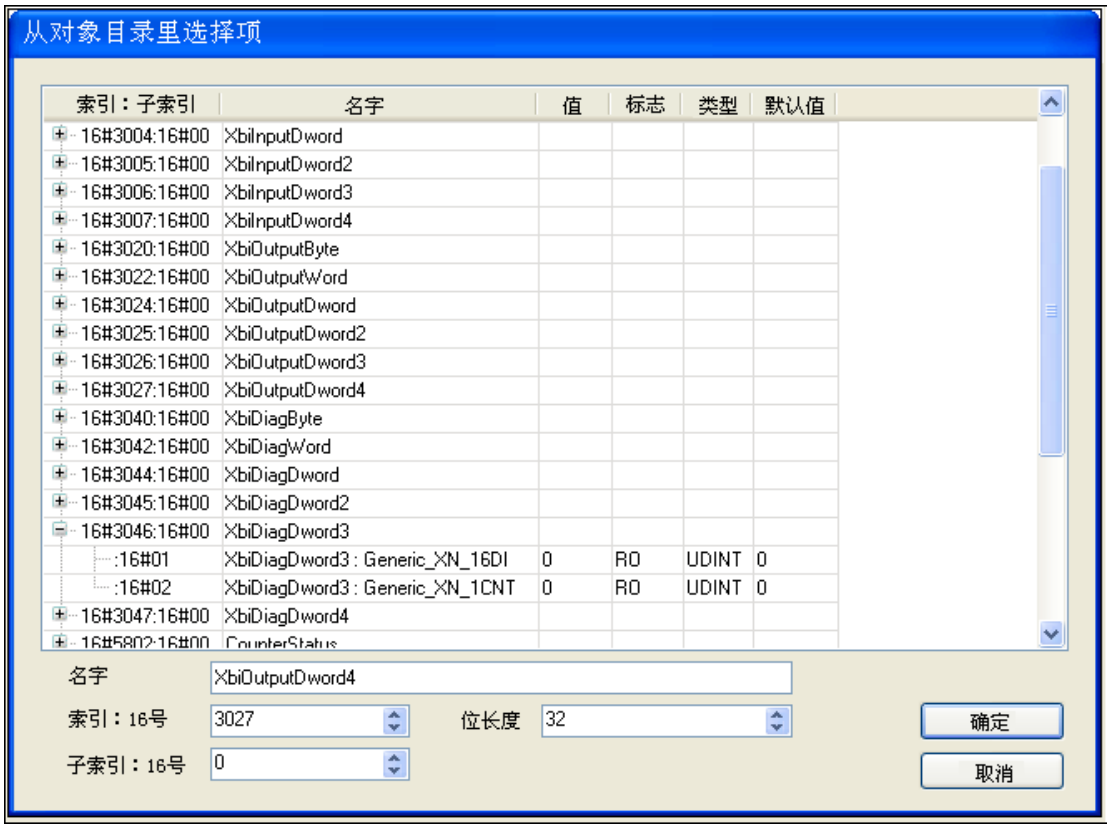
服务数据对象对话框如下图所示



要添加 SDO 节点 ，可点击“新建...

将弹出从对象库选择项对话框，该对话框和映射配置一样，可提供由 EDS 文件定义的对象库入口。

从对象库选择项对话框如下图所示：



对话框中只显示有写权限（访问类型＝ *W*）的对象。添加 SDO 前，可在对话框下面的区域修改相应的参数。通过定义新的索引和子索引，可为 SDO 添加新的对象，该对象不是在 EDS 文件中定义的。

SDO 列表中的顺序（上—下）是 SDO 被传输到模块的顺序。单击“上移”、“下移”，可修改选中节点在列表中的顺序。

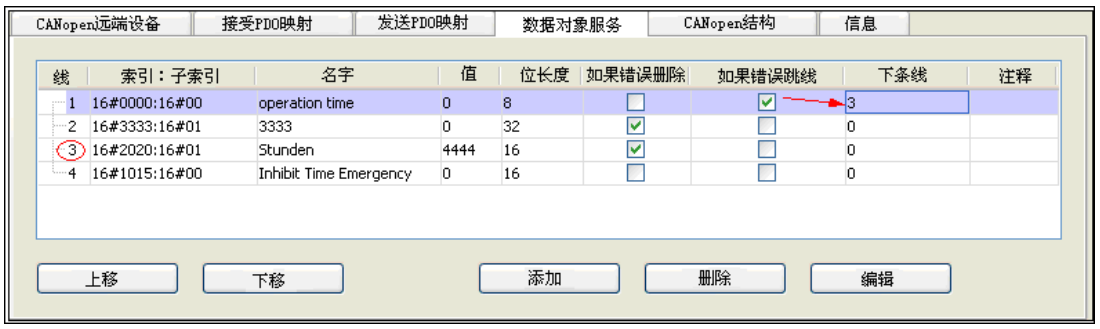
选中值域的空白处，单击空格键或者鼠标右键，可设定当前值。

激活下面的设置之一，可选择不同的 SDO 传输过程中出现错误时的处理方法：

出现错误时中止：传输出现错误时自动终止。

出现错误时跳转到指定行：出现错误时，传输将跳转到在该 SDO“下一行”区域中指定的行（行编号，行编号在行列中显示）继续传输。

配置错误处理的例子如下图所示：



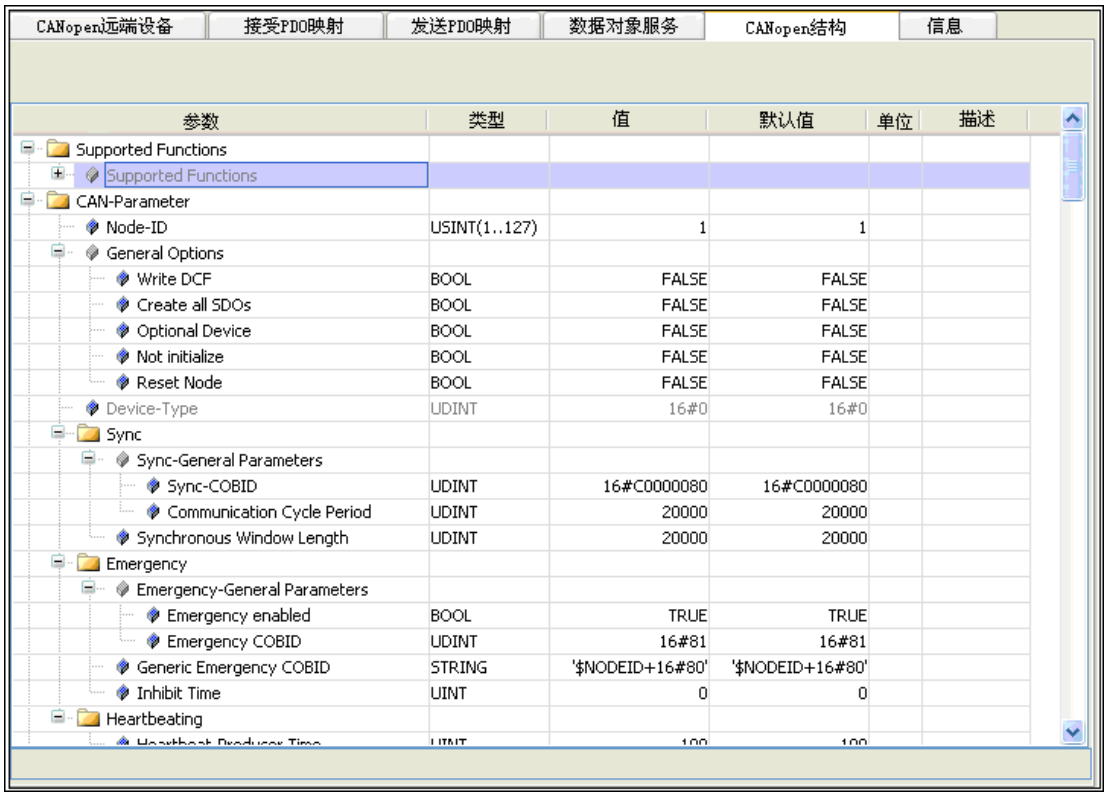
单击“编辑”...，弹出从对象库中选择项对话框，可参照上文中添加 SDO 的方法修改已在配置列表中列出的 SDO 的参数。

单击“删除”...，可删除当前选中节点。

CANopen 配置

CANopen 设备配置的CANopen 配置对话框列出由EDS 文件定义的参数，编辑相应区域可修改相应值。CANopen 配置对话框是编辑设备参数的标准设备编辑器对话框

CANopen 配置对话框如下图所示：



CANopenI/O 映射

非模块化设备的 CANopen 设备 配置才需使用此对话框，它是配置模块 I/O 映射的标准设备编辑

器对话框，即：硬件的输入输出通道与 IEC 变量相对应。

状态

CAN 总线设备或模块的状态配置编辑器可显示状态信息（如：“运行”，“停止”）、设备专用的诊断信息以及卡的使用情况和内部总线系统的状态。

信息

CANopen 设备或模块的信息对话框显示设备或在设备树中被选中模块的信息。若当前选中模块是可用的，将显示以下信息：名称，供应商，类别，版本号，顺序号，描述，图像。

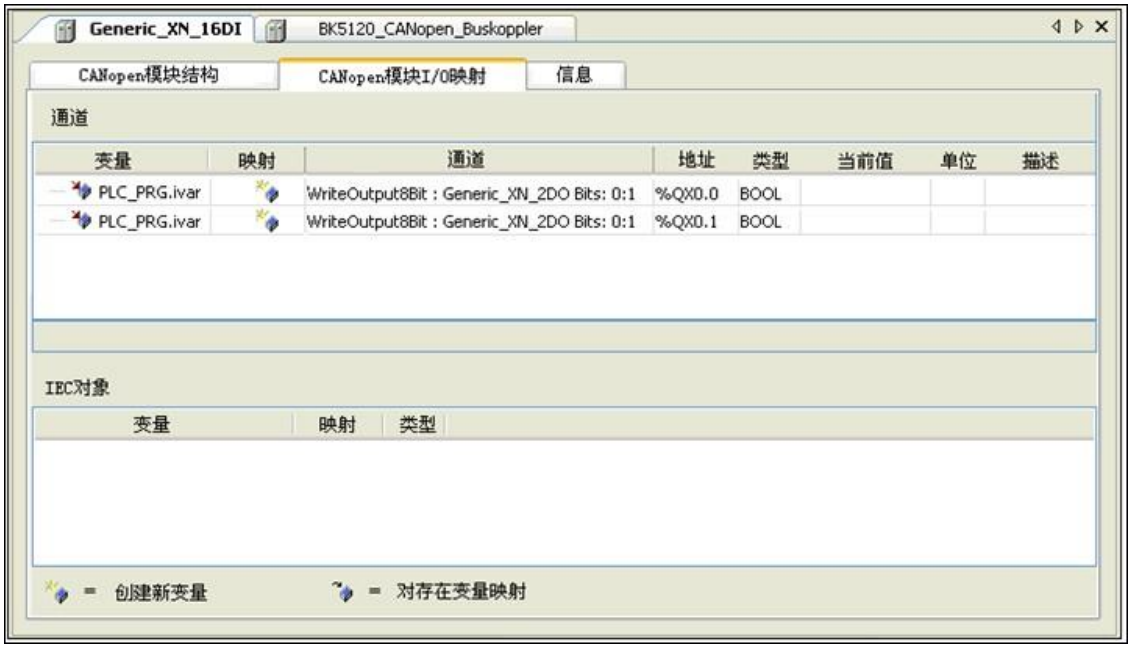
CANopen 模块

CANopen 模块配置

该对话框是 CAN 总线配置 的一部分，是设备编辑器的标准对话框，用于配置当前选中的 CANopen 模块的参数。

CANopen-Module I/O Mapping CANopen 模块 I/O 映射

该对话框是设备编辑器的标准对话框，用于配置模块的 I/O 映射。I/O 映射对话框如下图所示：



模块的该对话框能被激活时，远程模块（CANopen 设备模块）相对于没有 I/O 映射配置对话框的非模块化设备可认为是“模块化设备”。

激活“自动配置 PDO 映射 ”选项，当在标准 CANopen 设备节点下插入或者移除模块时，所有 CANopen 模块的 I/O 映射将自动更新。

状态

CAN 总线设备或模块的状态配置编辑器可显示状态信息（如：“运行”，“停止”）、设备专用的诊断信息以及卡的使用情况和内部总线系统的状态。

信息

CANopen 设备或模块的信息对话框显示设备或在设备树中被选中模块的信息。若当前选中模块是可用的，将显示以下信息：名称，供应商，类别，版本号，顺序号，描述，图像。

2.7.3 EtherCAT 设备编辑器

“EtherCAT”插件提供了对话框，该对话框用于配置插入到“设备视图 ”窗口里的工程设备目录中的 EtherCAT 主站和从站模块。这些对话框被添加到“设备编辑器”对话框中。

EtherCAT 模块配置起初是由主/从设备的设备描述文件决定，并可以被用户在配置编辑对话框中修改。然而，对于标准的应用以及尽可能操作简单，可以选择“自动配置”模式来进行最大程度的自动配置。

请注意，访问主站和从站并修改配置参数，也可以在操作模式下，通过使用主站和从站实例，或者 EtherCAT 库中的模块来实现。

EtherCAT 概述：

EtherCAT,是一种开放式的实时以太网，它实现了实时性能，因为使用双绞线或光纤，可以达到更新每 1000 个分布式 I/O 只需 30 μs，或者每 100 轴只需 100 μs。支持的拓扑类型包括线形、树形、星形。总线上有主站和从站。

Ethernet 数据包可以被处理 on the fly，而不必在每个连接点都接收，然后进行解码并复制为过程数据。每一个从节点的 FMMU（现场总线内存管理单元）读取地址匹配的数据，同时报文被转寄给下一个设备。同样，输入数据可以在报文通过时插入至报文中。帧的传递仅被延迟几个纳秒。由于数据链独立于EtherCAT 端子物理顺序，因此可以对EtherCAT 端子进行任意编址。从站之间可进行广播、多点传送和通讯。

精确同步在广泛要求同时动作的分布过程中显得尤为重要，如几个伺服轴在执行同时联动任务时。分布时钟的精确校准是同步的最有效解决方案。在通讯系统中，和完全同步通讯相比，分步式校准时钟在某种程度上具备错误延迟的容错性。

EtherCAT 设备的安装和插入：

CoDeSys SP Win V3 中加入 EtherCAT 设备必须首先安装 WinPCap 库（免费下载请登录 winpcap.org）。另外，还需要在 CoDeSys 配置文件（..\GatewayPLC\CoDeSysSP.cfg）中添加一项：

为了添加 EtherCAT 设备，CoDeSys 配置文件(...\GatewayPLC\CoDeSysSP.cfg)需要扩展如下配置项：

Component.<subsequent number>=SysEthernet

Component.<subsequent number>=CmpEt100Drv (使用 IntelPro100)

Component.<subsequent number>=CmpRTL81x9Mpd (使用 RTL8139)

Component.<subsequent number>=CmpRTL8169Mpd (使用 Realtek RTL8169 或者 RTL8168(PCIe 版本))

为了在设备目录中插入和配置 EtherCAT 设备，主站和从站必须使用硬件提供的设备描述文件，通过“设备库”对话框安装（ 标准的 CoDeSys 设置自动地完成） 。主站的设备描述文件 (*.devdesc.xml) i.a.，定义了可以插入的从站。从站的描述可以是 " xml " 文件格式（文件类型：EtherCAT XML 设备描述配置文件）。

通过“添加对象”或者“添加设备“命令，能将一个 EtherCAT 主站添加到设备目录中，在主站下可以插入一个或多个从站。

添加一个主站装置后，IoDrvEtherCAT.library 库文件包含的指令自动地添加至库管理器。主站设备被插入到设备目录后，库 IoDrvEtherCAT.library 自动的包含到库管理器中。

❑ 支持可选设备

如果缺少应用中配置了的 EtherCAT 设备，启动现场总线时将引发一个错误，最终将导致堆栈失败。在第一个 PLC 周期开始时设置<Ethercat 主站实例名称>.StartConfigWithLessDevice := TRUE;，缺少的设备会被当作可选设备对待，不再对现场总线的启动产生负面影响。

❑ 禁止输出额外消息

为了尽快的输出，EtherCAT 主站将向每个任务分别发送消息。如果一个已连接的驱动器同步实时输出数据，总线周期任务是唯一允许设置输出的，因为多个消息会扰乱同步。为了禁止其他任务的消息，可以设置<Ethercat 主站实例的名称>.EnableTaskOutputMessage := FALSE;

配置对话框：

在设备目录中选中主站或者从站条目后，可以通过双击或者使用“编辑对象”命令，打开用于配置 EtherCAT 的设备编辑器。

注意“设备编辑选项”中的编辑设置，例如通用对话框可以设置为不显示。

也要注意总线周期任务是在 PLC 设置中定义的。

EtherCAT 配置对话框总是以主站或从站名称命名，并提供包含下列子对话框的选项卡：

对于 EtherCAT 主站：

- 主站
- EtherCAT 配置/主站
- EtherCAT I/O 映射/主站
- 状态
- 信息

对于 EtherCAT 从站：

- 从站
- FMMU/同步执行
- 高级过程数据
- 过程数据
- 启动参数
- EtherCAT 配置/从站
- EtherCAT I/O 映射/从站
- 状态
- 信息

通过应用访问 EtherCAT 配置：

主站和从站的实例自动生成，并且能够应用在应用程序，例如开始、停止、状况检查等

- 主站的隐式实例

- 从站的隐式实例
- 检查所有从设备的组合列表

此外 EtherCAT 库提供功能块，实现在总线操作过程中特殊参数的读取和设置。

- "CAN over EtherCAT" 功能块

EtherCAT 主站

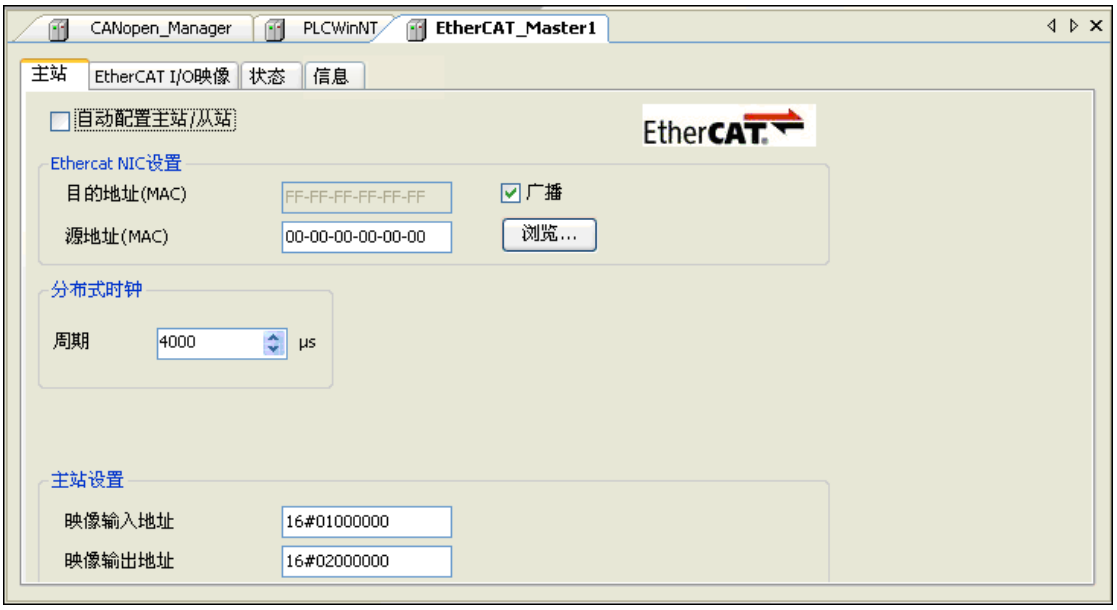
主站

EtherCAT 主站配置对话框上的“Master”选项卡，提供了设备描述文件中定义的主站设置。

EtherCAT 无需使用专用通讯处理机，只需使用无源的 NIC 卡或主板集成的以太网 MAC 即可实现。

插入 EtherCAT 主站设备时，将在当前有效应用程序的任务配置中自动插入 EtherCAT_Master 任务。

EtherCAT 主站配置对话框



自动配置主站/从站： 如果选中该选项，基于描述文件以及隐含的计算，主站和从站的主要配置将自动完成。在这种情况下 FMMU/Sync 设置对话框将不显示。

但是尽管主站处于自动配置模式，从站依然可以选中“高级模式”，允许手动修改自动产生的过程数据配置。

注意：自动配置模式是默认执行的，对于标准的应用程序通常是足够的，强烈推荐使用。如果取消此选项,主站和从站所有配置都必须手动进行，要求具备专业知识。要配置从站到从站的通讯，自动配置选项必须被设为无效。

EtherCAT NIC 设置

目的地址(MAC)：EtherCAT 网络成员应该接受报文的 MAC 地址。如果选中广播选项，则无需指定地址。

源地址(MAC)：网络名称：MAC 地址，网卡名称，例如：PLC (目标系统)

选择以下选项之一：

- 通过 MAC 选择网络
- 通过名称选择网络

设置之前请考虑以下内容：每个网络适配器有一个唯一的 MAC-ID. 所以，如果通过MAC-ID指定网络，您将在其他设备上不能使用此工程. 要使工程与设备无关，最好通过名称指定网络。

分布式时钟

主循环[μs]：指一个时间段，在此之后可以传送一个新数据报文。如果“分布式时钟”功能被执行，主站循环时间将传至从站。因此数据交换可以实现精确同步,在分布式过程中要求同步动作时(例如几个伺服轴执行同时联动任务)，这是特别重要的。可以在网络范围内提供信号抖动小于 1 微秒的、非常精确的时钟基。

诊断信息：在在线模式中，开始运行后将会显示诊断信息。如果"All slaves done"信息出现，表明配置已结束，总线开始运行。

选项

使用 LRW 代替 LRW/LRD：激活此项会使能从设备-从设备的通信。组合读写命令（LRW）会替代单独的读（LR）和写命令（LRW）。

使能每个任务的信息：激活此项, 处理输入输出信息的读写命令可以由不同任务完成。

自动重启从设备： 激活此项,主机会在通信终止后立即尝试重启从机。

主站设置

只有自动配置模式无效时，才可以进行主站设置（见上文）。否则，这些设置将自动完成，而且在这个对话框中是不可见的。

输入映射地址：是用于输入数据的第一个从站的第一个逻辑地址 。

输出映射地址：是用于输出数据的第一个从站的第一个逻辑地址。

EtherCAT 配置/主站

EtherCAT 主站配置对话框上的这个选项卡，提供了设备描述文件中定义的所有主站参数。

如果选中自动配置模式，这些配置将基于给定的设备描述和网络拓扑结构自动生成。通常对于标准的应用程序，不需要修改。

参数列表中包括了参数名称，数据类型，当前设定值，默认值，单位和简单描述。为了配置，数值区可能是可编辑的。但是如果设置了自动配置模式，任何参数的修改均无效！

这里可看到设备编辑器配置选项卡的概述。

EtherCAT 映射/主站

EtherCAT 主站配置编辑器的这个标签，对 EtherCAT 输出或者输入指定了工程变量。因此当 PLC 连接到 EtherCAT 主站后，可由应用程序控制。

自动生成的用于应用程序中的主站实例，被显示在这个对话框的下面部分（IEC Objects)。

状态

EtherCAT 主站配置编辑器的这个选项卡，提供显示状态信息（如："运行"，"停止"）和详细设备诊断信息，同样在当前使用的卡和内部总线系统上。

信息

此对话框提供一个 EtherCAT 主或从配置对话框。如果配置当前模块，将显示以下信息，包括：名称，销售商，类型，版本编号，分类，订单编号，描述，形象等。

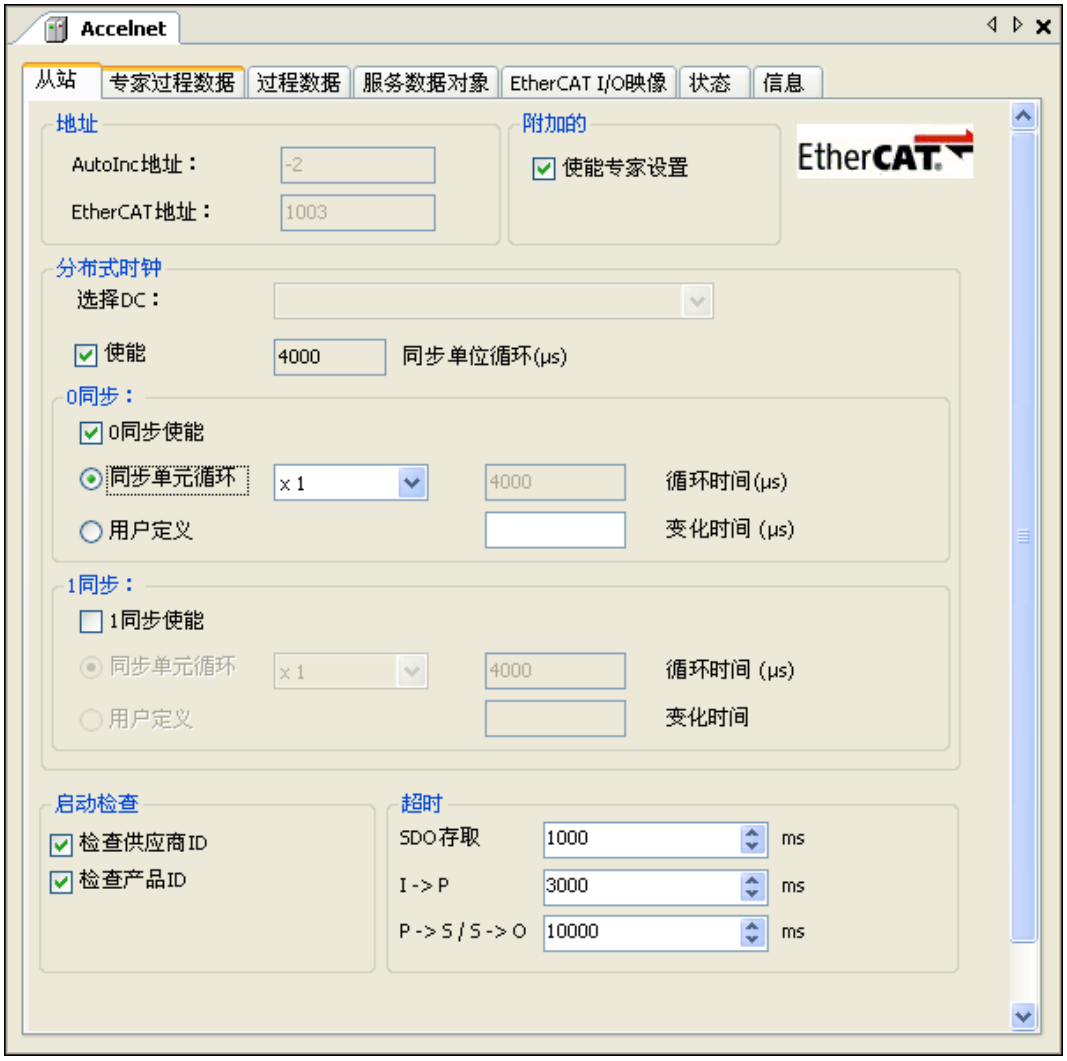
EtherCAT 主站或者从站配置对话框中的一个选项卡，提供了信息对话框。如果可获得当前模块的信息，将显示：名称、销售商、类型、版本号、分类、订单号、描述、映象等。

EtherCAT 从站

从站

EtherCAT 从站配置编辑器的"从站"选项卡，提供了在设备描述文件中定义的从站主要设置。

EtherCAT 从站配置对话框



地址

如果自动配置主站/从站选项无效，则以下域可设：

自动增量地址：自动增量地址（16 位），由从站在网络中的位置确定。这个地址仅仅在启动时使

用，此时主站需要向从站分配 EtherCAT 地址。当用于此目的的首个报文通过从站时，每一个经过的从站将自身的自动增量地址加 1。地址为 0 的从站将最终接收数据。可能的输入值，例如" -8 "。

EtherCAT 地址：从站的最终地址，由主站在启动时分配。这个地址和网络中的位置无关。

分布式时钟

选择分布时钟：下拉菜单提供了由设备描述文件提供的所有关于分布时钟的设置。

使能：如果"分布式时钟"的功能启动，在同步单元周期（µs）域显示的数据交换周期时间，将由主站周期时间确定。因而，主站时钟能够同步网络内部的数据交换。

同步单元操作设置，与从站相关：

同步 0：

使能同步 0：如果该选项被选中，使用 " sync0 "同步单元（ beckhoff ）运用。一个同步单元描述了一套同步交换的过程数据。

同步单元周期：如果该选项被选中，主站周期的时间乘以所选择的系数，将被用作从站的同步周期时间。循环时间（µs ）栏显示当前设置的周期时间。

用户自定义：如果该选项被选中，可以在周期时间栏输入以微秒（µs）为单位的期望时间。

同步 1：

使能同步 1：如果该选项被选中，使用 "sync1"同步单元（beckhoff）运用。一个同步单元描述了一套同步交换的过程数据。

同步单元周期：如果该选项被选中，主站周期的时间乘以所选择的系数，将被用作从站的同步周期时间。循环时间（µs）栏显示当前设置的周期时间。

用户自定义：如果该选项被选中，可以在周期时间栏输入以微秒（µs）为单位的期望时间。

诊断

这部分仅在在线模式时可见。

当前状态　显示从站的当前状态，可能的值包括：初始化，运行前，安全运行，运行。

如果状态为"运行"，从站配置已经正确结束。

附加的

使能高级设置： 如果该选项被选中，用于启动检查和超时（见下文）的附加高级设置将生效，高级过程数据对话框被激活。然而对于标准的应用，主站缺省有效的自动配置模式是足够的，因而没有必要使用"高级设置"。

启动检查

默认在系统启动时，根据当前的配置设置，检查供应商 ID 和/或者产品 ID 。如果检测到不匹配，总线将将停止运行，并且不执行进一步的操作。这是为了避免下载一个错误配置。

在这里你可以使该选项无效，以关掉检查。

超时

默认对于以下动作，不定义超时。如果依然想观察他们是否超过了一个确定时间，在这里指定时间（以微秒为单位）。

SDO 存取：在系统启动时，发送 SDO 列表。

I -> P: 从"初始化"模式切换到"运行前"模式。

P -> S / S-> O: 从“运行前”切换到“安全运行”模式，或者从“安全运行”到“运行”模式。

DC 周期单元控制：分配到本地 µC

选择关于分布式时钟功能定义的选项，这些选项将被分配到本地微处理器。控制功能已经在 EtherCAT 从设备的寄存器 0x980 中完成。可能的设置：周期单元，锁存器单元 0,锁存器单元 1.

站别名

此设置只有被从设备描述显式支持才能被激活。允许直接命令地址别名以获取与总线中位置无关的从设备地址。

FMMU/同步

如果主站的自动配置 模式无效，此对话框仅可通过 EtherCAT 从站配置编辑器的选项卡打开。它将显示 设备描述文件定义的从站的 FMMUs（现场总线内存管理单元）和同步管理器。可以修改这些设置，例如用来配置从站到从站的通讯。

注意：这些都是高级设置，对于标准应用来说通常不需要！

FMMU：这个表显示了用于处理过程数据的从站的现场总线内存管理。定义了每个物理地址（Ph. 开始地址）上的逻辑地址（globstartaddr）的映射。通过添加…或者编辑…按钮，可以添加新的单元，或者通过编辑 FMMU 对话框编辑现有单元。

同步管理器：此表显示从站的同步管理。对于每个可用的同步管理类型（邮箱输入，邮箱输出，输入、输出）i.a.，定义了物理开始地址，访问类型，缓冲区和物理地址，中断信号送达地址。通过添加或者编辑按钮，可以添加新的同步管理器，或者通过编辑同步管理器对话框编辑现有同步管理器。

高级过程数据

如果从站的“高级设置”模式有效，则此对话框可以通过 EtherCAT 从站配置编辑器的选项卡打开。它提供了另外一种更详细的过程数据视图，过程数据同时也显示在“过程数据”选项卡中。

另外，还可以选择 PDO 分配和 PDO 配置的下载。

过程数据

此对话框可以通过 EtherCAT 从站配置编辑器的选项卡打开。它显示了设备描述文件所描述的从站输入和输出过程数据。

设备中被选择的输入（可读）和输出（可写），将在 I/O 映射对话框中作为 PLC 的输出和输入获取，工程变量可能被映射。

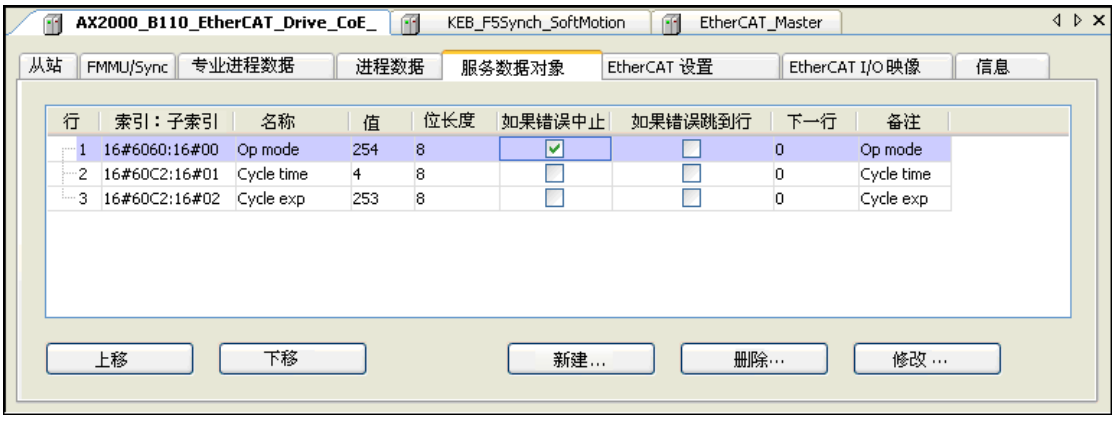
为了更改当前的选择，必须先用鼠标点击当前选择的数据前面的检验栏，以取消选择。此后可以设置另一个。

服务数据对象

如果从站的“高级设置”模式有效，此对话框可以通过 EtherCAT 从站配置编辑器的选项卡打开。

对于特殊的从站设备定义 SDO，将在系统启动时，将指定的的参数传给设备。仅当设备支持"CAN over EtherCAT"时，显示此对话框。目标字典中包含的必要的数据对象，包括在 EtherCAT XML 描述文件中，或者包括在 XML 描述文件所引用的一个 EDS 文件中。

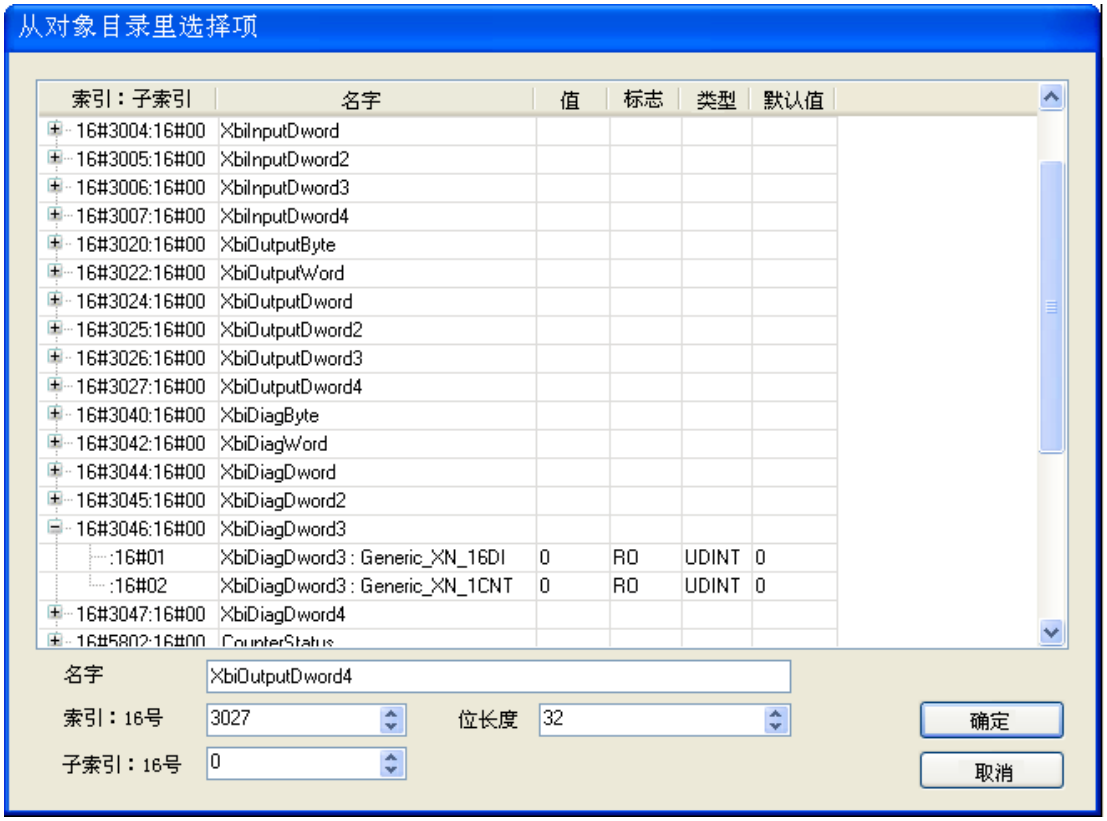
服务数据对象对话框



在配置表中添加一个 SDO 条目，选择“新建”...按钮。

“从目标字典中选择条目”对话框将打开，提供 EDS 文件定义的可用的目标字典条目。

对话框－从对象库中选择条目



添加 SDO 之前，可以通过在对象表下面的区域编辑，来修改其参数。通过定义新的索引/子索引，可以给 SDO 选择和添加一个 EDS 文件中没有描述的新对象。如果对象字典不全或者根本不存在时，这是有用的。

在 SDO 表中的顺序（自上而下）代表了传送到模块的顺序。使用按钮“上移”和“下移”，可以在列表中移动当前选择的条目。

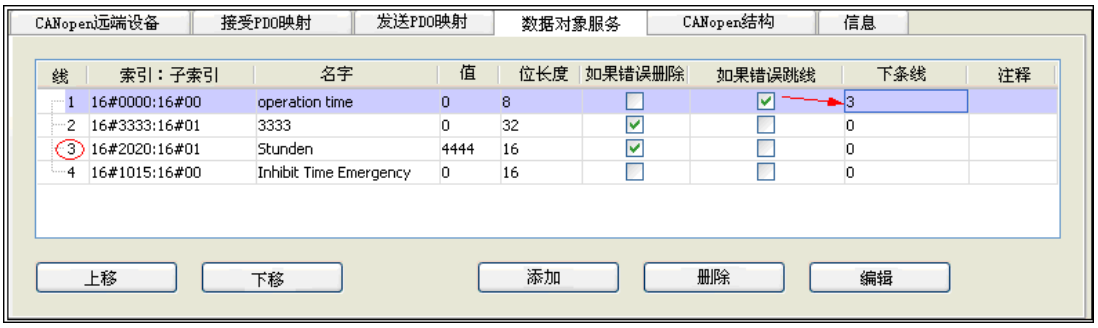
值和注释字段也可以直接编辑。选定区域后,按“空格”键或用鼠标点击，可实现编辑。

定义 SDO 传送错误时的操作，可以通过激活以下设置之一：

如果错误则中止：传送将终止，并给出错误状态。

“如果错误跳转到行”和“下一行”：将继续传送 SDO 指定的行，该行在栏

配置错误操作，例子



为了修改一个已经被列入 SDO 配置表中的 SDO 参数，可以使用“编辑...”按钮，打开“从对象库中选择条目”对话框。可以如上文所述描述的增加 SDO 一样，进行编辑。

“删除...” 按钮将删除当前选定的条目。

EtherCAT 配置/从站

EtherCAT 从站配置对话框上的这个选项卡，提供了 设备描述文件中定义的所有从站参数。

如果主站的自动配置 模式被激活，这些配置将基于给定的设备描述和网络拓扑结构自动生成。通常对于标准应用，不需要修改。

参数列表中包括了参数名称、数据类型、当前设定值、默认值、单位和简要描述。为了配置，数值区可能是可编辑的。但是如果设置了自动配置模式，任何参数的修改均无效！

这里可看到设备编辑器配置选项卡的概述。

Ethercat I/O 映射

EtherCAT 从站配置编辑器的选项卡，为 EtherCAT 输出或者输入指定工程变量。因此连接到 EtherCAT 从站的 PLC，可由应用控制。

状态

EtherCAT 主站配置编辑器的这个选项卡，提供显示状态信息（如："运行"，"停止"）和详细设备诊断信息，同样在当前使用的卡和内部总线系统上。

信息

此对话框提供一个 EtherCAT 主或从配置对话框。如果配置当前模块，将显示以下信息，包括：名称，销售商，类型，版本编号，分类，订单编号，描述，形象等。

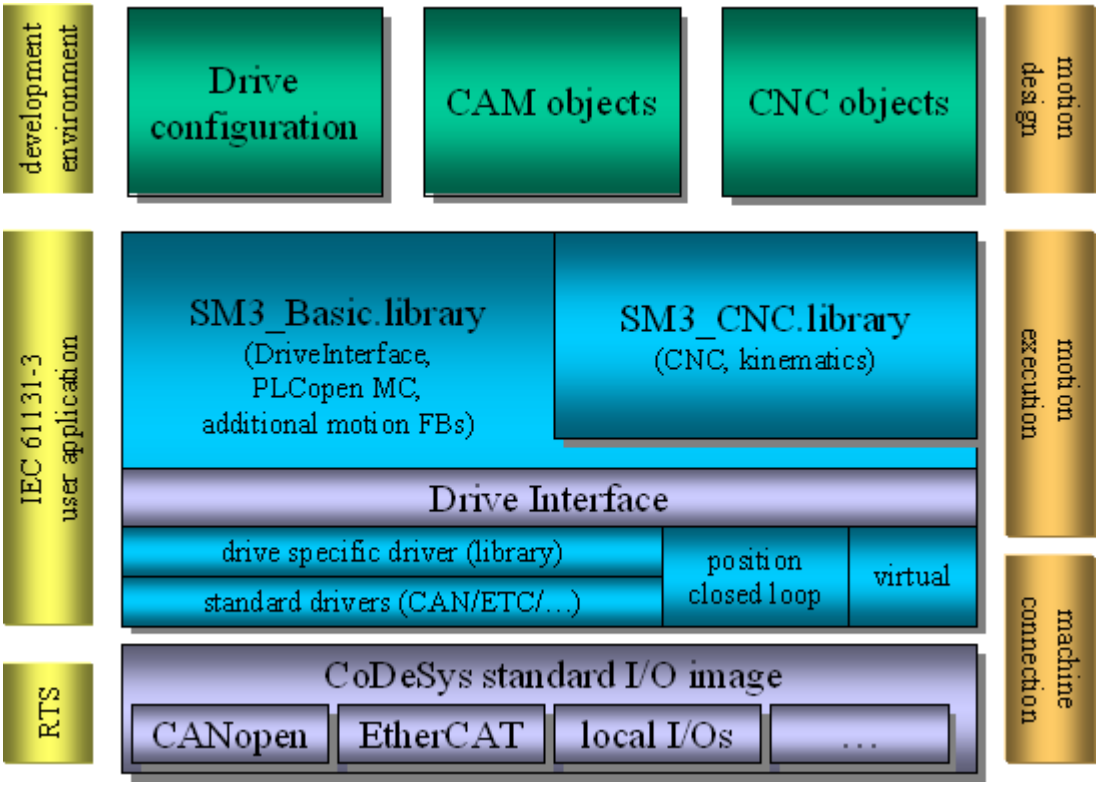
EtherCAT 主站或者从站配置对话框中的一个选项卡，提供了信息对话框。如果可获得当前模块的信息，将显示：名称、销售商、类型、版本号、分类、订单号、描述、映象等。

2.7.4 Softmotion 设备编辑器

关于组件的 SoftMotion 概念及回顾

SoftMotion 能够使用户有效地实现运动控制而无需了解其细节：从初级的单轴运动或电子凸轮到多维的复杂运动。主要的应用不仅集中在运动的功能特性，而且还集中在序列和过程控制以及相关的功能，使得软件运动应用与 CoDeSys 开发环境融为一体。程序逻辑完全由 PLC 程序处理，其中纯粹的运动则由调用库函数执行。

CoDeSys SoftMotion 是一个软件包， 它是运动控制器的开发及实时运行环境。除了包含 CoDeSys 和 CoDeSys SP 标准特性外，其还包含运动控制器所特有的部分：



❑ 驱动配置提供一个编辑器用来导入结构和驱动-硬件的配置到 CoDeSys 用户界面。因此，驱动界面库中的函数将会创建 IEC 数据结构作为该驱动的抽象表示。例如，IEC 程序员不需要做任何的额外工作，驱动界面都会自动地与驱动器设备通讯，并且处理该驱动器数据结构及传输更新数据。为了控制驱动器，IEC 程序可以有两种方式访问其抽象的数据结构，通过使用 SoftMotion 库的标准模块（SM3_Basic.library, SM3_CNC.library）或者通过 使用由 IEC 程序员为此目的而创建的模块。由此，目标值会被周期性地输出；这就意味着，在每个 IEC 任务中会在每次循环中都计算一次目标值（如位置，速度，加速度等），并将目标值从驱动界面传输给驱动器。去“教”驱动（例如只给定一个最终位置，驱动将会主动的运动并报告它的到达）是不可能的。原因是当一个指令被执行时，中央控制器对于驱动是没有任何影响的。因此，它不太可能实现例如对多轴的协调动作控制。

❑ 电子凸轮编辑器是一个独立的 CoDeSys 插件，其可以被集成到程序界面内。电子凸轮被用来控制多个驱动轴。在该编辑器内，一个电子凸轮可以通过图形或表格的方式来实现。至此，一个包含对象描述的全局数据结构实例将会被隐含创建，并且被发送给该应用，在此处，它可以通过一个对应的 POU 来访问。

❑ CNC 编辑器是一个独立的 CoDeSys 插件。它容许多维运动的编程，其可以通过与驱动-硬件

相连接的驱动界面进行调度和控制。该编辑器工作类似与 CNC 语言 DIN66025，是由一个图形编辑器和一个与之同步的文本编辑器构成。除此之外，表编辑器可用相同的功能范围替代文本编辑器。基本上，其可以实现 9 维运动，但这里只能实现 3 维的非线性插补。因此，在 3 维系统内，直线、圆弧、平行线、椭圆及样条曲线都可以被编程，而另外维度方向上则只能实现线性插补。对于每个已经被设计好了的轨迹，CoDeSys 都会自动地为其创建一个数据结构，其可以在IEC 编程时使用。

❑ 库 SM3_Basic.library 是一个基本元素，其必须被包含在 SoftMotion 应用中。由以下元素组成：

◆ PLCopen 功能块依照PLCopen 标准，其可以简单地实现一个单轴运动的控制，也可以使得两个轴的同步运动。除了库元素状态检测、参数化及一般操作外，还有些功能块用来设定轴相关的速度定义及加速度参数等。如果两个轴需要同步，一个轴做为主轴并按照一定的规则控制第二个轴（从轴）。这个规则既可以是一个使用相关 POU 的电子凸轮，其使从轴连接到主轴。此外，有些功能块支持电子齿轮功能和相位移动功能。

◆ 几个特有的运动控制功能块。

◆ 驱动界面基本驱动器功能块 (AXIS_REF, AXIS_REF_VIRTUAL, AXIS_REF_MAPPING)

◆ 公用功能：例如文件复位或错误报告

◆ 实现特殊功能驱动器的功能块：基本功能块可以被更多的特殊驱动器扩展，其可以与一个特定的驱动类型进行通讯。

库 SM3_CNC.library 基于 SM3_Basic.library。除了一些功能块用于运动学的转换外，它还提供所有 POU 所必需的创建、执行以及可视化的 CNC 运动。

驱动界面是SM3_Basic.library 库的一部分，其负责IEC 程序与驱动之间的通讯。它还为扩展基本驱动功能块到特殊驱动器类型提供了可能性。这些特殊驱动功能块（以 AXIS_REF_*命名）被特意地组合为一个分离的库，该库可以做为相关设备的设备描述参考。还有，驱动界面包括总线特定的库，其用来处理 I/O 映射。

注意一些应用程序示例的描述。

可移植性

除了这些直接复位硬件元件的驱动器外，所有的 SoftMotion 实时元件可以用 IEC61131-3 编程。因此，其有很大的平台可移植性。

注意：“通讯设定”和“应用”标签涉及到与 PLC 的通讯，并且其可以由“设备通讯编辑器”插件提供。

SoftMotion 驱动界面

SoftMotion 驱动界面是一个标准化的界面，允许在 IEC 程序内包括、配置以及定位一个驱动硬件的抽象影像。在一个通用界面内的各种硬件映射不仅可以简单地进行驱动更改及 IEC 程序复用，还可以使得 PLC 和各种驱动间或在驱动设备间进行通讯。这个界面在应用中（AXIS_REF）是连接驱动与其表达式的界面，对于驱动控制来说其负责更新及传递必须的运动数据。

驱动界面是由下面几项组成：

◆ SoftMotion 设备的设备描述在设备树中进行表示。

◆ 设备描述所参考的库其依照特殊设备类型的需要，来扩展或重载基本的 AXIS_REF 功能块。

◆ 包括功能块的库，其用于一个非循环数据的读/写，还用于对现场总线栈的基本函数进行包装。

基本库导入到库管理器也可以通过 SoftMotion PLC 提供一个 SoftMotion 一般驱动池的应用来实现。接着可以有两种方式添加驱动设备：

- 作为自由驱动设备的配置

在目标树中，这些驱动设备不能被确定到另一个设备中，但是可以收集在每个 SoftMotion PLC 所配备的 SoftMotion 一般驱动池中。

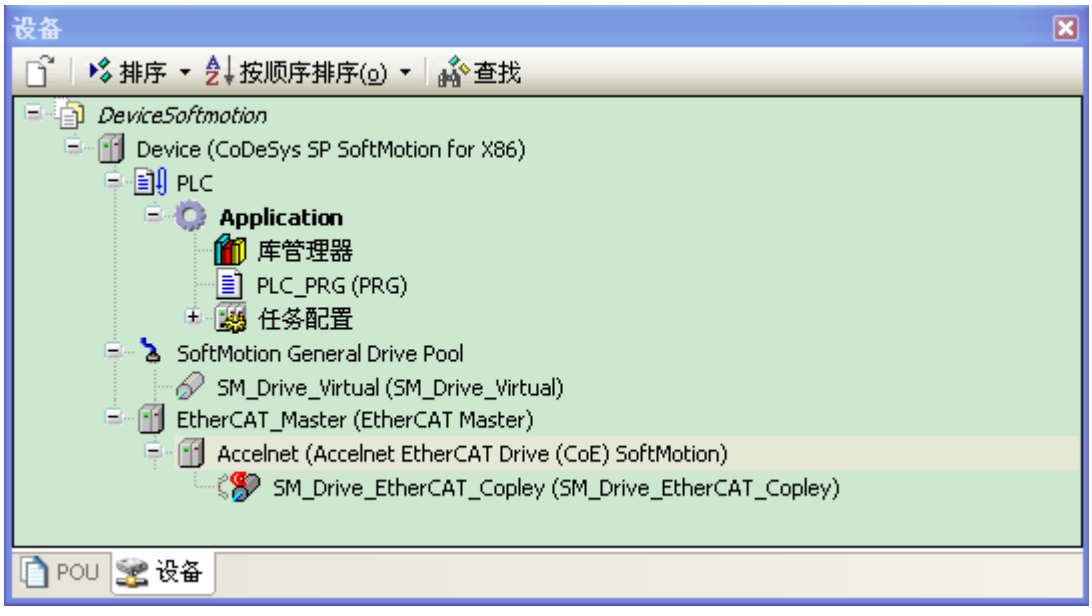
- 例如虚拟驱动 是自由驱动设备，因此它们不能被连接到一个别的现场总线设备；还有位置控制的设备可以做为自由设备，所以对于该现场总线设备给它们提供输入/输出并不是必须的。

自由驱动设备的典型配置

- 已连接的驱动设备配置

这些驱动设备在现场总线拓扑图内，采用另外一种设备表示驱动的方式代替一对一方式表示的相互关联关系。与这些驱动设备相关联的图标由一些小符号来扩充，这些符号可以表示其相连接现场总线类型。例如，任何标准的伺服驱动设备可以通过一个现场总线连接到控制器。

已连接驱动设备的典型配置



配置逻辑驱动

SoftMotion 逻辑驱动是主设备或编码器上的虚拟驱动。

逻辑驱动的坐标轴反映了主坐标轴作为单独实例时的运动情况。允许在不影响主坐标轴的情况下执行下列指令：MC_SetPosition, MC_TouchProbe（可能用于在触摸式探测通道未被主坐标轴和主坐标轴的其他逻辑坐标轴使用的前提下的逻辑坐标轴）和 MC_AbortTrigger。

坐标轴类型 AXIS_REF_LOGICAL_SM3 是库 SM3_Basic 的一部分。此类型的实例在 SoftMotion 逻辑驱动编辑器下配置：

- SoftMotion 逻辑驱动
- SM_Drive_Logical: 配置
- SM_Drive_Logical: I/O 映射
- 状态
- 信息

SoftMotion 驱动：逻辑

SoftMotion 驱动编辑器的第一个选项卡给出了一些配置框。

SoftMotion 驱动：逻辑，取模坐标轴

SoftMotion 驱动：逻辑，有穷坐标轴

坐标轴类型：

如果坐标轴类型被设置为取模，那么取模设置可见并且可以设置取模值.

如果坐标轴类型被设置为有限，那么可以设置正负坐标限。

主设备参考值：

可以在此单选框中选择设置和实际的主设备参考值。根据此输入来确定主设备的设置值或实际值作为逻辑坐标轴的输入和偏移补偿。

时间偏移：

此处可以设置推测的周期数。周期数定义时间间隔，其设置值必须被推测。

输入数据过滤：

过滤器计算移动平均值。上限为所选的周期数，相关值都被用于计算平均的位置/速度。计算结果是平滑值。范围是 0-9。如果周期数数设为 0，最新的一个值用于过滤位置或速度。不然就是用计算出来的值。

SM_Drive_Logical：配置

逻辑坐标轴的数据结构实例列在所有已配置设置的表格视图里。

SM_Drive_Logical：I/O 映射

不能进行映射。

状态

可看到逻辑驱动的状态信息。

信息

此页显示逻辑驱动从设备描述文件读出的信息。

控制器上的位置控制 (SM_Drive_PosControl)

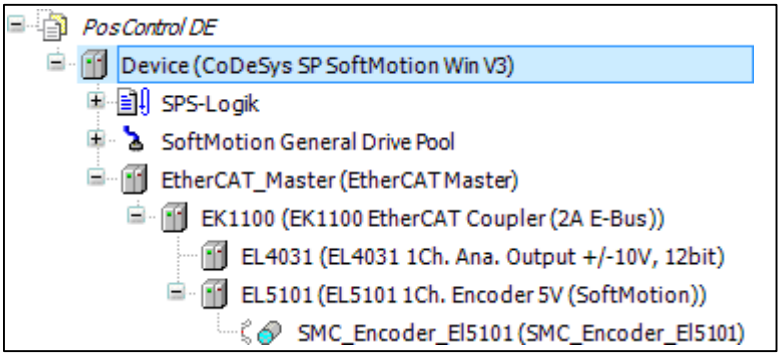
此例中没有使用伺服控制器接管当前调节器，转速控制和位置控制。取而代之的是，它展示了怎样用CoDeSys 对具有位置反馈的速度控制设备（例如，一个频率转换器）进行位置控制。前提是，受设定速度控制和具有实际位置返回的设备有效。

此例是典型的，使用 10V 模拟输出夹钳（Beckhoff EL4031，其信号作为频率转换器转速的设定位置）和位置反馈的解码终端（Beckhoff EL5101）。

要求

必须把终端添加到 SoftMotion 通用驱动库的已有的设备树和位置控制坐标轴。

设备树

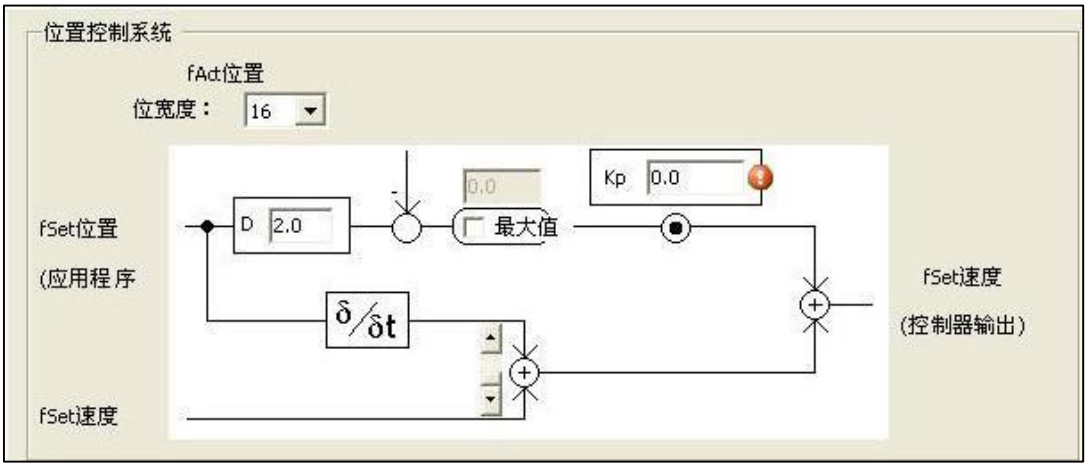


通过双击打开SM_Drive_PosControl 的设备编辑器，在基本和缩放/映射选项卡里进行常用设置。要进行正确的缩放配置，你必须知道在电动机一个完整的旋转里解码终端增量的多少。在此例中，一个完整的电动机旋转力有 4096 个增量，用弧度来表示就是 360 度。

SoftMotion 驱动：缩放/映射

然后请仔细查看位置控制回路选项卡，进行下列设置。

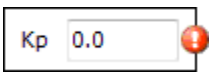
SoftMotion 驱动：位置控制回路



D

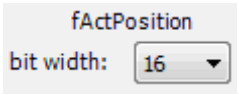
静止时间：按照周期数定义，接收的实际位置（解码器）是与坐标轴的设定位置有相位偏移的。它取决于使用的组件并必须用实验的方法来确定（请看下面内容）。

Kp



比例常量：它是位置误差必须乘以的误差因子（设定位置和实际位置之间的差别）。然后把它加上设定速度。首先这个因子被设定为 0 然后进行试验确定。

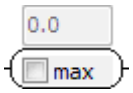
实际位置的位宽



实际值可以以 16，24，32 位接收。此处必须为控制功能块设置一个正确值。因为使用的组件返

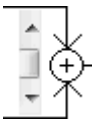
回 UINT 型的位置数据，因而必须设为 16。

位置误差监控.



可关闭此选项，如果需要的话可通过复选框激活。另外必须设置最大偏差。如果在操作时偏差越界，坐标轴进入错误状态。

fSetVelocity



使用设定速度/设定位置：通常设定速度和设定位置的计算是一致的，所以这块区域不太重要并且只在特定情况下使用。所以要把滚动条翻到最下面才能使用速度作为来源。

现在要缩放转移到执行机构的速度值。为此需要知道应用单元的最大速度值和相应的已转移数据的原始值。在此例中，通过 6#7FFF 输出使速度达到最高点，这和按照 3600 度/秒设置的 10 转/秒相符合。

下一步是设置连接坐标轴和设备数据的现场总线。有效的坐标轴周期数据存储在数据结构 in 和 out 里。可以通过程序链接数据结构或直接映射到 I/O 设备的设备编辑器。要把输出（转速的设定值）映射到夹钳，必须打开 EL4031 设备，然后把输出映射到选项卡 EtherCAT I/O 映射坐标轴的变量 out.iSetVelocity。如果输出是 32 位的，必须使用变量 out.diSetVelocity。

设备 EL4031：EtherCAT I/O 映射

对位置控制输入做相同的操作。打开设备 EL5101，把位置输入值映射到 in.wActPosition(如果是 32 bit ， 就选择 in.dwActPosition)。

设备 EL5101： EtherCAT I/O 映射

对坐标轴的操作，必须设置下列输入值。如果已使用的组件提供这些数据（使用现场总线连接中频率转换器），可以直接映射它们。此处，数值的简单接口是无效的，所以必须在应用程序中手动设置。

SM_Drive_PosControl.in.bDriveStartRealState := TRUE;

SM_Drive_PosControl.in.bRegulatorRealState := TRUE;

下列输入 ...

bLimitPos	BOOL	正向限制开关(只针对有限坐标轴， 激活 bHW)
bLimitNeg	BOOL	负向限制开关

wActPosition/ dwActPosition	WORD/ DWORD	实际位置
bExternalError	BOOL	外部错误
bRegulatorRealState	BOOL	TRUE：控制的坐标轴
bDriveStartRealState	BOOL	FALSE：快停的坐标轴

...和输出是有效的

bRegulatorOn	BOOL	TRUE：控制的坐标轴转换
bDriveStart	BOOL	FALSE：快停的坐标轴转换
diSetVelocity / iSetVelocity	DINT/INT	设置速度

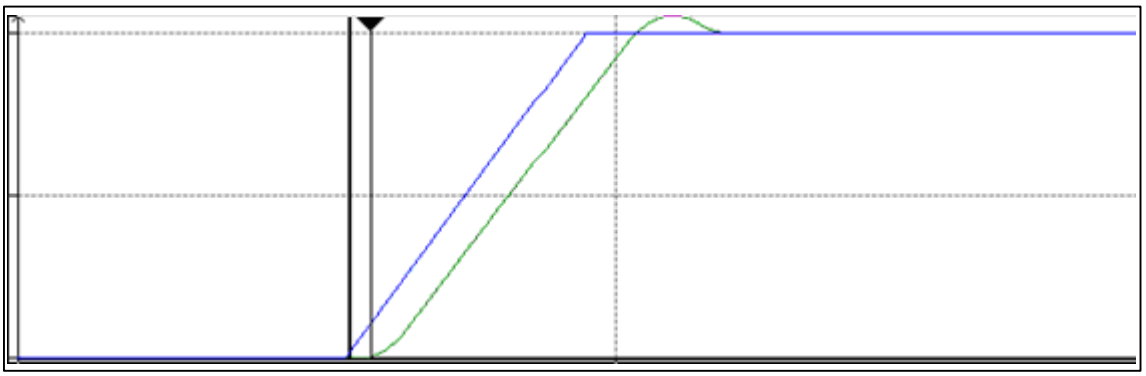
现在应用程序可以在线，也可以设置控制参数。

请注意，坐标轴可变成不可控的，所以，必须做好适当的安全预防。

首先，必须尝试在没有位置控制的情况下操作坐标轴（设置 fKp 为 0.0）并验证缩放设置。为此，打开 MC_Power 轴，启动 MC_MoveVelocity。此时请注意，坐标轴以程序设定的速度移动（1 转/秒），为防止偏差，必须调整相应的缩放比例。

现在停止移动（例如，MC_MC_MoveRealtive）并启动跟踪函数。

要确定系统的停止时间，必须测量设置位置和实际位置之间的瞬时偏移。为此设置 MC_MoveRelative 为最大速度和最大加速度（例如 108）.然后启动跟踪采样和 MC_MoveRelative。在测量运动的开始时间和实际位置的首次反应的时间偏差时，就获取了系统停止时间。



用绝对时间除以周期时间，可以得到偏移周期数。

现在可以尝试查找 fKP 设置。为此设置 fKp 为很小的值（例如 0.0001）并逐渐增大。用跟踪采样检查在每一次变化时的行为。一检查到有振幅，已经达到上限，应该减小 fKP 设置的 10%。此值可以放在设备描述里，此时坐标轴可以使用。

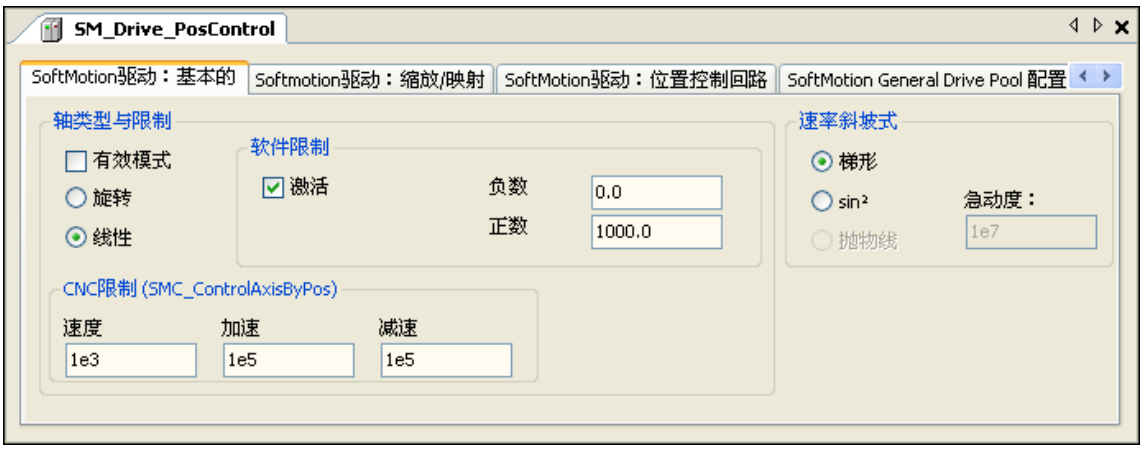
设备 SoftMotion 编辑器的配置

依据特定设备类型，相关的设备 SoftMotion 编辑器将按照其配置包含不同的选项卡。用户可以在 SoftMotion 编辑器的基本配置中找到其详细的描述。

SoftMotion 驱动：基本

设备 SoftMotion 编辑器的第一个选项卡提供对插入设备的基本设定配置。

设备 Softmotion 编辑器 - SoftMotion 驱动：基本



轴类型与限制：

虚拟模式：如果勾选了复选框中的“虚拟模式”项，该设备将会由一个类似于一个虚拟驱动设备来进行仿真。因此，对于虚拟驱动器来说，该复选项将会一直被选中。对于实际被连接的设备，万一该选择了该选项，也不会对于现场总线上的设备造成影响，其还会在通常状态下工作，但其将不会对一个真正的设备进行信息的发送或接收。

有两种轴类型可以选择：线性轴或者旋转轴。根据这个选择，其右侧的小框将会以此而命名。

软件限定 （对于线性驱动器可能的设定）：如果想设定整个行程的最小值（反向限定值）或最大值（正向限定值）时，激活该复选项.

模数设定（对于旋转驱动器所必须的设定）：必须定义模数的范围。

CNC 限定(SMC_ControlAxisBy)：

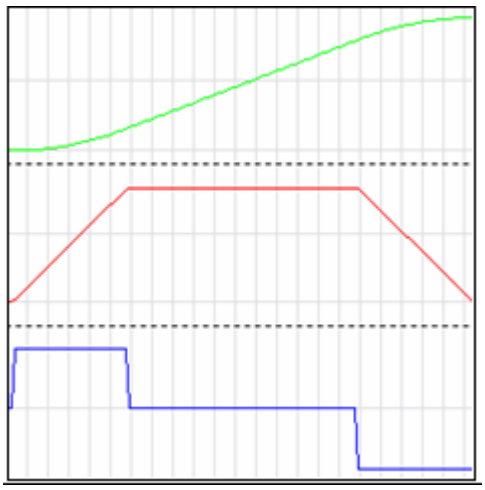
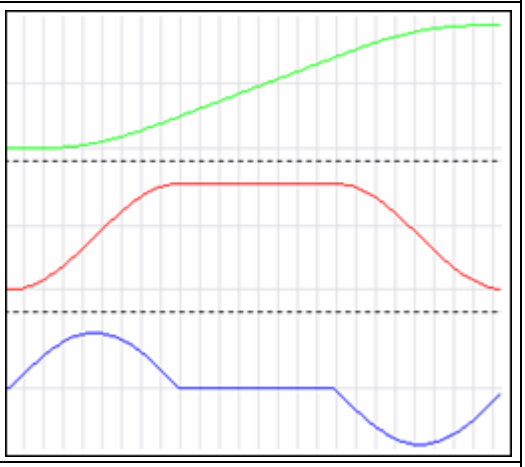
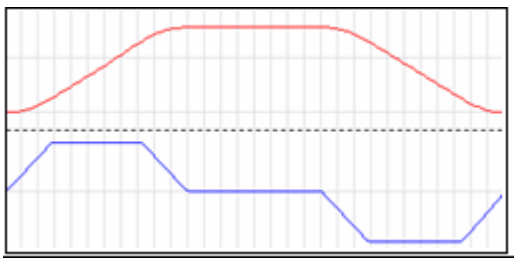
在这个地方可以通过库模块 SMC_ControlAxisBy 来定义一些数值的限定值（例如：速度、加速度和减速度），以此来决定跳转的次数。（例如 SMC_ControlAxisByPos）

速度斜坡类型：

速度斜坡类型定义一个轴及主/从轴运动过程中速度产生的曲线模块。其关联的库中将会提供多达三种不同的选择：梯形将产生一个梯形的速度曲线（既在每一段上都有着一样的加速度），sin2 将产生一个 sin2 的速度曲线（其有着连续的加速度），抛物线将产生一个抛物线的速度曲线（其有着连续的分段地加速度曲线）。

对于后两种模式，还必须额外定义跃度。

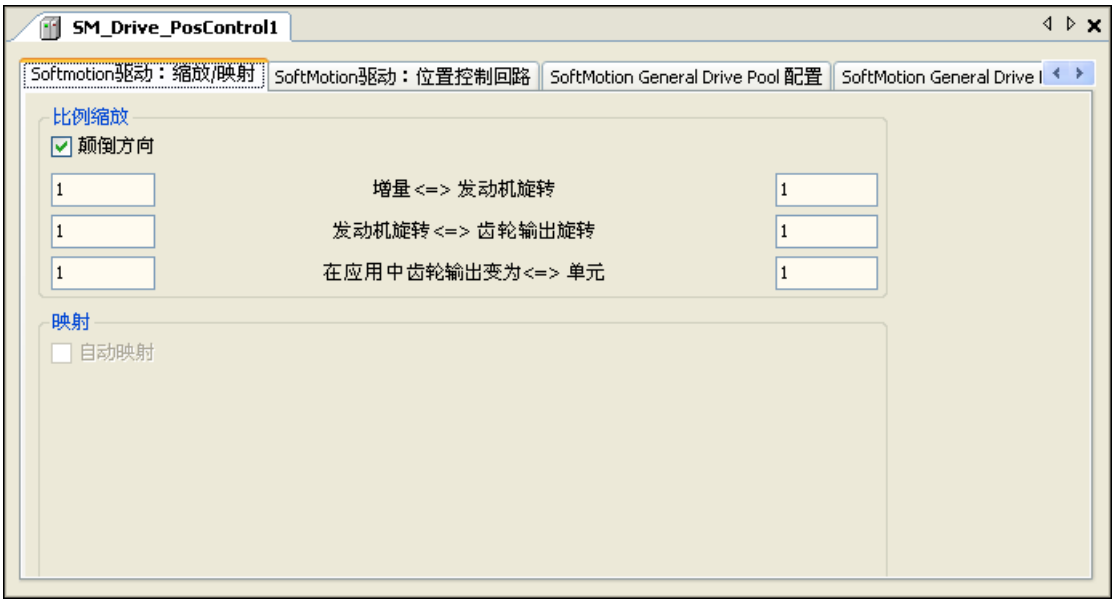
接下来的图像中展示了不同斜坡类型的效果；位置曲线用绿色表示，速度用红色表示，加速度为蓝色。

梯形 速度是分段线性的，但其还是连续的，由此可知其加速度是分段的，每一段都为一个常数，其段与段之间会有跳变，并不连续。	
sin2 速度函数的 变化边沿是平滑的（其由 sin2函数来实现的，而不是一条直线函数），由此不会产生加速度的跳变。 由于运动序列已经被清晰地定义，一般情况下，此时的跃度不能由用户来限定。只有在驱动器在开始时的加速度不为0时，对于跃度的限定才有效。然后对于跃度的限定，会使驱动将在一个实际运动开始之前将其加速度返回到0。相比梯形曲线来说，此时的减速时间将会变长。	
抛物线（尚未实现） 减速度有一个连续的，分段线性化的曲线，该曲线的斜度由跃度来限定。所以其速度会是一个连续的、分段的抛物线曲线。 实际上，这是仅有的跃度可以限定的曲线类型。	

Softmotion 设备：缩放/映射

这一页提供在其应用中，对于物理设备到其表现形式之间的调整。

设备 Softmotion 编辑器 - Softmotion 设备：缩放/映射



缩放：

必须输入乘数因子参数，其用来定义从设备接收到的整数位置值（在左侧第一行输入）与 IEC 程序内使用的技术单位（右侧第三行输入）之间的转换关系。此外，通过编辑更多的参数值，可以将一个设备纳入其中。

一种典型的配置例如，一个驱动器一转产生3600 个计数值，而其需要在实际应用中被缩放转换成采用弧度为技术单位。

映射：

使用自动映射，IEC 程序所涉及的驱动将会自动地被映射到该驱动恰当的输入/输出处。在勾掉该复选项后，你在显示的参数列表（其会根据设备描述文件来进行不同的编辑项显示）内通过修改地址或输入/输出的类型，从而实现对映射的手动编辑。

状态

尚未实现

信息

这一页中显示的信息是针对该特殊设备的设备描述文件中读出的信息。

特殊使用情况摘要

库 SM3_Basic.library 包括一些高级别的功能块，其并不是通过标准的设定/激活位置界面的方式相互作用，而是通过给定命令或通过 AXIS_REF 的读写参数来修改驱动的设定。使用这些功能块可以使得下面的使用情况得到正确的处理：

一个驱动的重新初始化

在驱动的重新初始化的运行过程中必须进行的驱动参数修改（像缩放因子，旋转驱动的位置周期等等…）。在一个输入信号的上升沿触发调用功能块 SMC3_ReinitDrive，其将复位驱动参数 wCommuncationState 并且重复完成该驱动的启动过程。

一个驱动的使能/取消使能

AXIS_REF 中的参数化 bRegulatorOn 和 bDriveStart 控制一个驱动的上电阶段及刹车；通过使

用功能块 MC_Power 可以对其产生影响。

为了使能上电阶段，参数 bRegulatorOn 必须被设定；驱动通过参数 bRegulatorRealState 报告这个进程的当前状态。如果其被驱动设备所支持，运动过程中bRegulatorOn 参数的一个复位将会使得驱动设备立刻终止力矩的输出。

变量 bDriveStart 决定驱动是否被允许移动；驱动通过参数 bDriveStartRealState 报告当前状态。如果其被驱动设备所支持，在运动过程中的，参数 bDriveStart 的一个复位将会使驱动按照配置的斜坡类型进行减速；之后，只要 bRegulatorOn 参数保持设定，则驱动将保持其位置不变。

机械刹车的处理

如果被驱动所支持，变量eBrakeControl 可以通过功能块SMC3_BrakeControl 进行设定（复位）。三种不同的模式是可能的（SMC3_BrakeSetState: SMC_BRAKE_AUTO, SMC_BRAKE_OPEN, SMC_BRAKE_CLOSE）

变量 bBrakeClosedRealState 报告刹车的当前状态。

从 SoftMotion 的视角来看，对机械刹车的控制和其状态既不依赖驱动状态也不依赖当前正在进行的运动。

控制模式的切换

功能块SMC_SetControllerMode 可以使用户在不同控制模式下进行切换。参数byControllerMode 和 byRealControllerMode 分别用来表示期望的控制器状态和实际的控制器状态。其可能的值为：

1	力矩/功率控制
2	速度控制
3,0	位置控制
4	电流控制

在变量 byControllerMode 被写入后，实际状态将会被检测直到它与期望的控制模式一致为止。

归零（由驱动控制的）

SoftMotion 提供两种可能性来使一个驱动归零：一种由驱动控制，一种由控制器控制。

后一种通过调用功能块 SMC_Homing 来实现。因而，其归零的参考信号必须是控制器 IO 镜像的一部分。在归零操作中，驱动保持在位置控制模式。

一般，这种方法的精度没有由驱动控制的归零方式好，其原因是控制器的位置锁存器是基于来源于驱动的实际位置，因此它会受到通讯周期的限制。

相对应的，驱动器控制的归零所用的参考开关要接到驱动本身，它可以获得当前位置更为迅速，因此会有比较高的精度。在驱动控制的归零过程中，驱动被设定到一个不同的模式，此时驱动不会在去根据控制器设定值来运动，而是由其内部进程来控制。一旦归零过程完成，驱动就会立刻复位到其原先的控制模式下。

通过设定 fReference and bStartReference，可以使功能块 MC_Home 做为驱动控制的归零进程启动。此外，AXIS_REF (wState)的状态会被设定为“归零中”。一旦驱动进入到归零进程中（其意味着驱动将不再遵守控制器的设定值），驱动器将设定参数 bStartReferenceRealState 为 TRUE。为了表示归零过程的结束，驱动器将会把参数 bStartReference 和 bStartReferenceRealState 复

位到 FALSE．．

锁存功能（驱动控制的）

SoftMotion 提供两种可能的方式用来锁存一个驱动的实际位置：高精度的方式是直接将信号接入到驱动内，由驱动来直接锁存它的位置。第二种可能的方式是使用控制器输入用以测量实际位置。第一种方式更加精确（因为其对于信号的传递有着更少的延迟），但是其需要驱动支持这种特性。而且，一般的窗口捕捉功能只有第二种方式才支持。尽管如此，捕捉过程内置于驱动还是标准方式。

功能块 MC_TouchProbe 设计用来锁存功能，其将通过数据结构 aCaptDesc（AXIS_REF 参数的一部分）与驱动器通讯；这个数据结构包含全部关于捕捉位置的信息，捕捉的启动和停止，以及有效的捕捉窗口位置。

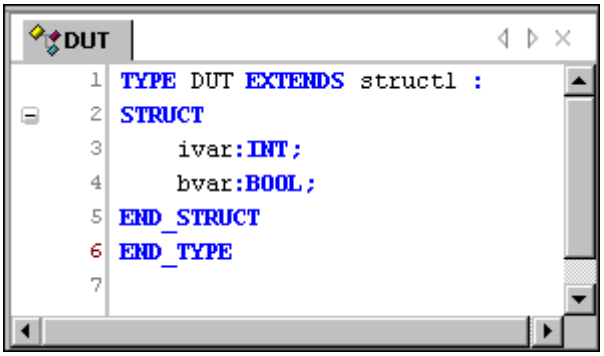
2.8 DUT 编辑器

数据单元类型编辑器

用户可以通过数据单元类型编辑器（DUT 编辑器）自行定义数据类型。DUT 编辑器是一种文本编辑器，其操作取决于编辑器的当前设置。

在通过添加对象对话框，添加 DUT 对象时，DUT 编辑器将被自动打开。此时 DUT 将缺省提供扩展结构体格式的声明，用户可以根据需要将其变为一种简单结构体的声明或其它数据类型单元的声明，如，枚举类型。

当在 POU 视图中中打开当前选中的已存在的 DUT 对象时，DUT 编辑器也会被打开。



2.9 FBD/LD/IL 编辑器

本编辑器是一个综合编辑器，用于 FBD（功能块图），LD（梯形图）和 IL（指令表）语言编写程序组织单元（POUs）。

在此编辑器内可以对指令和单元进行的统一设置，以及实现三种语言模式的自动转换。在离线模式下，编程者都可以切换到其它的编辑器视图（视图...）。

当然，有些特殊的单元不能转换，只能用于可显示它的语言中。同样，有一些结构体不能准确地在 IL 和 FBD 编译器之间转换。这涉及到：表达的取反，及显式/隐式输出。

通过“节列表编辑器”插件我们可使用 FBD/LD/IL 编辑器。行为，外观和菜单在自定义和选项对话框的设置中定义，特别是涉及到注释和地址显示的选项。

当编辑一个用 FBD/LD/IL 语言编写的对象时，此编辑器在双向窗口中打开，此编辑器上部分为一个声明编辑器。

在通过“添加对象”命令创建一个新对象时指定该对象的编程语言。

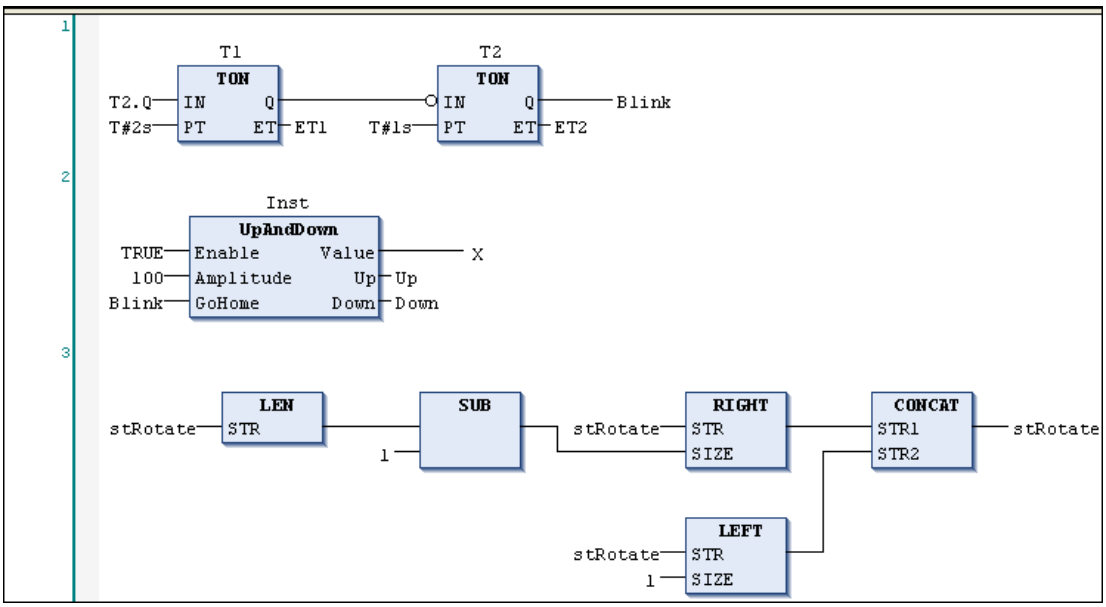
2.9.1 语言

功能块图-FBD

功能块图是一种图形化的编程语言。它由一系列的节组成，每一个节又由图形结构的运算块和连接线的组成，实现逻辑、运算、调用、跳转或返回功能。

FBD/LD/IL 编辑器由“节列表编辑器”插件提供。

功能模块图中节的例子



梯形图-LD

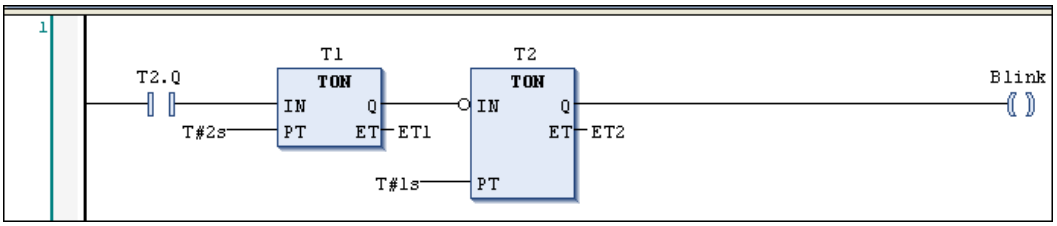
梯形图是一种图形化的编程语言，它类似于电路的结构。

梯形图不但很适用于逻辑的转换，并且它也能创建类似于 FBD 中的节，所以用梯形图调用程序组织单元也是是很方便的。

梯形图包含了一系列的节，左右两边各有一个垂直的电流线（能量线）限制其范围，在中间是由触点、线圈、连接线组成的电路图。每一个节的左边有一系列触点，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，通过连接线从左到右传递状态。否则传递“关“的状态。在节最右边的线圈，根据左边的状态获得一个开或关的值，并相应地赋给一个布尔变量真或假值。

FBD/LD/IL 编辑器由“节列表编辑器”插件提供

梯形图节的例子



指令表-IL

指令表是遵循 IEC 61131-3 标准的编程语言。

此语言是基于累加器的编程语言。支持所有 IEC 61131-3 标准的操作符、多输入/多输出、取反、注释、输出的置位/复位和无条件/条件跳转。

每条指令首先使用 LD 操作符将值装载到累加器中。然后累加器将被执行。执行的结果存储在累加器中，使用 ST 指令可以将该结果赋给其它变量。

为了实现条件执行或循环，IL 支持如 EQ, GT, LT, GE, LE, NE 比较操作符和跳转指令。跳转指令可以分为无条件(JMP)或条件的(JMPC / JMPCN)跳转。条件跳转时需检查累加器的值是真或假。

语法：

指令表由一系列的指令组成。

每条指令占据一行，包含一个操作符和一个或多个用逗号隔开的操作数。操作数之间用逗号分隔。

每行的指令前面可以有标题，标题后要有冒号。例如：在下文中的例子"ml:" 标题可以是跳转指令的目的地，在下文中的例子“跳转到下一个”。

每行结束处可以加入注释。

每行指令之间可以插入空行。

IL 编辑器中的 IL 程序例子

```
1
    LD          ivar            boolean input
    ST          inst1.IN        starts timer…
    JMPC        m1
    CAL         inst1           (
                PT:= tl         ,   start time
                ET=> tout1      )
    LD          inst1.Q             is TRUE, PT s…
    ST          inst2.IN            starts timer…
2   m1:
    LD          dwvar
    ADD         230
    ST          dwres
```

2.9.2 IL 语言中的修饰符和操作符

以下的修饰符可以在 指令表中使用：

C	结合 JMP，CAL，RET 使用	本指令仅当前面的表达式结果为真时执行
N	结合 JMPC，CALC，RETC 使用	本指令仅当前面的表达式结果为假时执行
N	其他	操作数（而非累加器中的数）取反

下表中列出了哪些操作符可结合特定的修饰符一起使用。

累加器通常储存当前值，该值是执行前一指令得到的结果。

操作数	修饰符	含义	举例
LD	N	将操作数（取反）载入累加器	LD iVar
ST	N	将累加器中的数（取反）存入操作数变量中	ST iErg
S		当累加器中的数为真，将操作数（布尔型）设为真	S bVar1
R		当累加器中的数为假，将操作数（布尔型）设为真	R bVar1
AND	N,(	累加器中的数和（取反的）操作数逐位进行与运算	AND bVar2
OR	N,(	累加器中的数和（取反的）操作数逐位进行或运算	OR xVar
XOR	N,(	累加器中的数和（取反的）操作数逐位进行异或运算	XOR N,(bVar1,bVar2)
NOT		累加器中的数逐位取反	
ADD	(	将累加器中的数和操作数相加，其结果复制到累加器中	ADD (iVar1,iVar2)
SUB	(	累加器中的数减去操作数，其结果复制到累加器中	SUB iVar2
MUL	(	累加器中的数和操作数相乘，其结果复制到累加器中	MUL iVar2
DIV	(	累加器中的数除以操作数，其结果复制到累加器中	DIV 44
GT	(	检查累加器中的数是否大于操作数，其结果（布尔型）复制到累加器中 ； >	GT 23
GE	(	检查累加器中的数是否大于或等于操作数，其结果（布尔型）复制到累加器中 >=	GE iVar2
EQ	(	检查累加器中的数是否等于操作数，其结果（布尔型）复制到累加器中; =	EQ iVar2
NE	(	检查累加器中的数是否不等于操作数，其结果（布尔型）复制到累加器中; <>	NE iVar1
LE	(	检查累加器中的数是否小于或等于操作数，其结果（布尔型）复制到累加器中; <=	LE 5
LT	(	检查累加器中的数是否小于操作数，其结果（布尔型）复制到累加器中 无条件（有条件）跳转到标签; <	LT cVar1
JMP	CN	无条件（有条件）跳转到标签	JMPN next
CAL	CN	（有条件）调用一个程序或功能块（当累加器中的数为正时）	CAL prog1
RET		从当前 POU 中返回，跳回到调用的 POU 中	RET
RET	C	有条件-仅当累加器中的数为真时，从当前 POU 中返回，跳回到调用的 POU 中	RETC
RET	CN	有条件-仅当累加器中的数为假时，从当前 POU 中返回，跳回到调用的 POU 中	RETCN
)		评估延迟的操作数	

使用修饰符的 IL 程序例子：

```
1
    AND     TRUE    load TRUE to accumulator
    ANDN    bVar1   execute AND with negated value of bVar1
    JMPC    m1      if accum. is TRUE, jump to label "m1"

    LDN     bVar2   store negated value of bVar2...
    ST      bRes    ... in bRes

2   m1:

    LD      bVar2   store value of bVar2...
    ST      bRes    ... in bRes
```

2.9.3 FBD 和 LD 编辑器视图

“节”是 FBD 和 LD 程序的基本组成单元。每个节包含逻辑式或运算表达式，POU（函数，程序，功能块调用等），跳转，返回指令。

当创建一个新的对象时，编辑窗口将会自动添加一个空节。

注意：一般编辑器的设置参看选项对话框中'FBD, LD and IL'选项卡。

工具小贴士显示变量或运算块参数的信息

放置在变量名或运算块参数上的光标处会显示一个工具小贴士，它显示运算块参数或变量的类型。经过重新定义，地址和注释符号以及操作数注释也将显示（在第二行的引用标号）

插入和排列：

- 此编辑器视图命令默认在FBD/LD/IL 菜单 中有效的，其中最重要部分通常在context 菜单中。哪些元素可通过菜单命令插入取决于光标的当前位置或当前所选(可能有多种选择)。
- 元素也可以直接用鼠标从工具箱拖动到编辑器窗口或从编辑器中的一个位置到另外一个位置（“拖&放”）。为此单击鼠标选中一个元素，然后按住鼠标左键，把元素拖到编辑器视图的节中。到了节区域中，所有元素类型的可能插入位置都会由灰色位置标记标识。当把光标放到这些位置时，位置标识将变成绿色，此时释放鼠标，把元素放到该位置。
- 在编辑菜单中，剪切，复制，粘贴和删除都是缺省有效的。同样可以通过拖放复制一个元素：点击鼠标选中一个节中的元素，按住鼠标左键，拖动元素到目标位置。只要达到目标位置（绿色位置标识），一个加号将被加进光标图标。释放鼠标，插入元素。
- 光标的位置见 FBD/LD/IL 光标位置。

操作：

- 箭头用于跳转到下一个或前一个光标位置；也可能在节之间。
- <Tab>键用于跳转到一个节中的下一个或前一个光标位置。
- <Ctrl>+<Home> 滚动到文档开始且标记首节。
- <Ctrl+End> 滚动到文档的结尾且标记后节。
- <PageUp> 滚动到上一个屏幕且标记最上面的节。
- <PageDown>滚动到下一个屏幕且标记最上面的节。

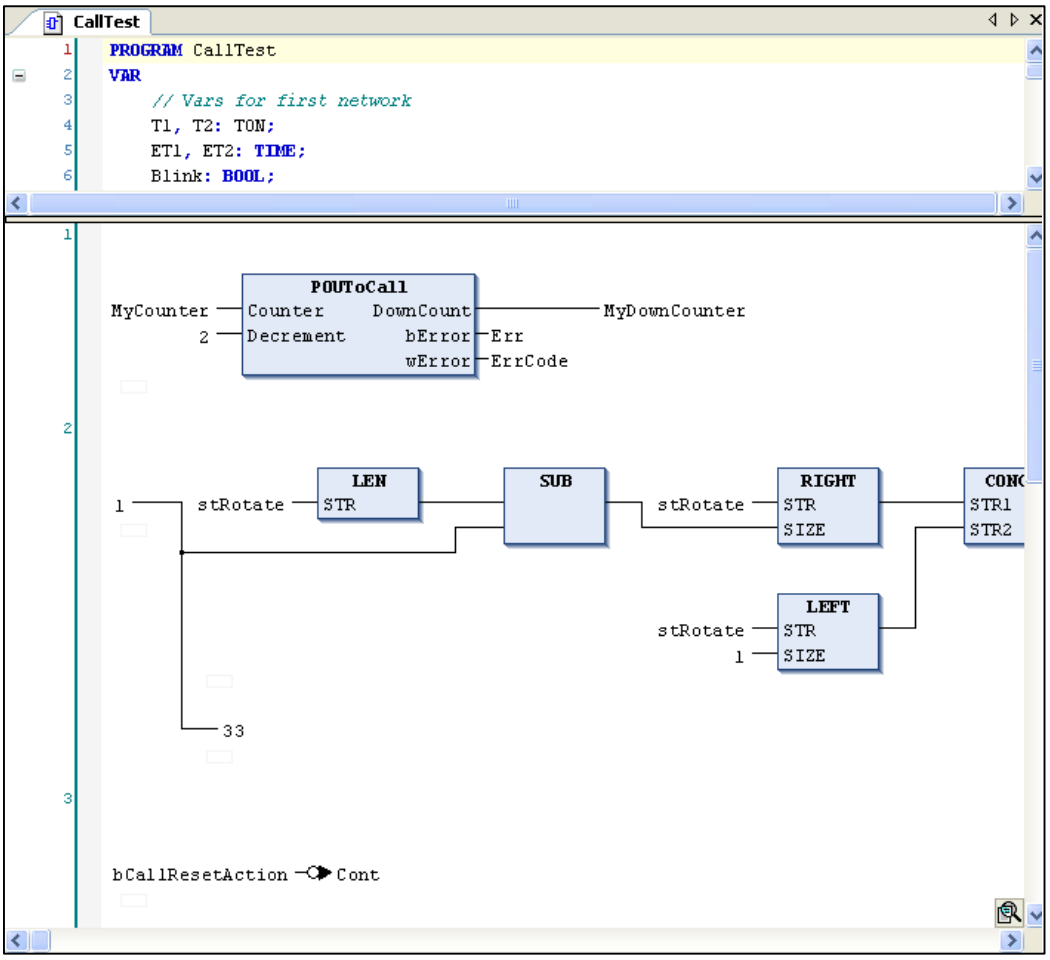
选择：

- 可以通过在光标位置点击鼠标或使用箭头或制表键选择一个单元，或节。

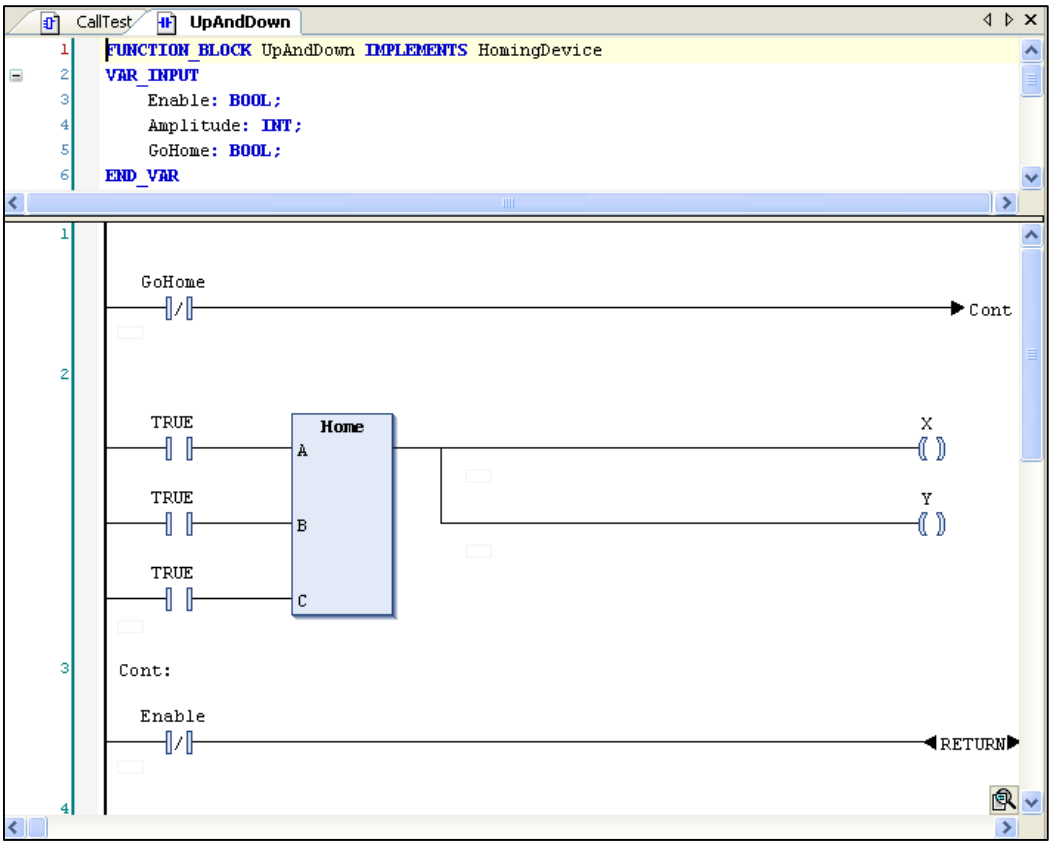
◆ 在选择一个接一个的所需元素时，通过保持选择，可实现不相邻元素或节的多选。

◆ 摁住<shift>键的同时选中决定节的开始和结束的两个触点，这样 LD 中多个邻近元素或节可以被选中。如果你想剪切（复制）和粘贴节的一部分，按下<Ctrl>键，同时选中该部分边缘的两个触点，中间的元素将会自动选中。

FBD 编辑窗口的例子



LD 编辑窗口的例子



2.9.4 IL 编辑器视图

相比在 CoDeSys 版本 V2.3.中使用的纯文本编辑器来说，IL（指令列表）编辑器，是一个表格编辑器。FBD 或 LD 程序中的节概念在 IL 程序中有体现。对于 IL 程序来说，基本上一个节 就足够，但是考虑到需在 FBD, LD 和 IL 之间转换，我们可以有意识地使用节构建一个 IL 程序。

注意：编辑器的设置参看选项对话框中 FBD, LD and IL 选项卡。

插入和排列：

此编辑器工作命令默认在 FBD/LD/IL 菜单中，大部分重要命令在右键菜单中可以找到。

编程单元，也就是元素，可以通过 FBD/LD/IL 菜单中插入命令被插入到当前光标位置。

在编辑菜单中剪切，复制，粘贴和删除菜单能用于编辑程序，

如何构建表格编辑器，在编辑器中如何使用向导及如何使用复杂操作数，如何调用和跳转见下文。

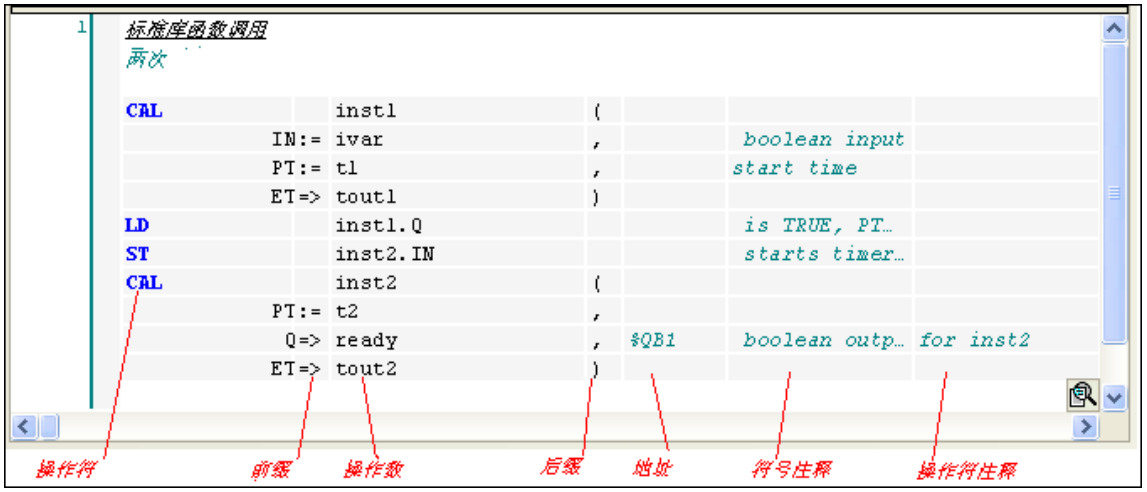
表格编辑器的结构：

每条程序都是以表格行的形式写出，通过下面表格栏构建：

列	内容	说明
1	（操作符）	该栏目包括指令列表操作符（LD，ST，CAL，AND,OR 等等）或是函数名在调用功能块的同时也指定各自的参数。在这种情况下，在前缀字段必须输入“:=”或是“=>”
2	操作数	该字段仅能包括一个操作数或是一个跳转标签。如果需用多于一个操作数（多操作数/扩展操作数 如“AND A, B,C”或是调用带有多个参数的函数），那些操作数必须在后面各行中写入并且操作数字段要为空。在这种情况下，加入参数分隔逗号。在调用功能块，程序或动作时，必须加上相应的打开或关

		闭括号。
3	地址	该字段包括操作数在声明部分定义的地址。该字段不能编辑，并且可以通过“显示符号地址”来切换打开和关闭。
4	符号注释	该字段包括在声明部分为操作数定义的注释。该字段不能编辑，并且能通过“显示符号注释”选项来切换打开或是关闭
5	操作符注释	该字段包括当前行的注释。是可编辑的，并且可以通过“显示操作符注释”选项.来切换打开或是关闭。

IL 表编辑器



操作表格：

Up- and Down 箭头键：向前或后移动

<Tab>:在一行内移动到右边

<Shift>+<Tab>:在一行内移动到左边

<Space>:打开当前所选编辑域，或用鼠标点击该区域也可。如果是可编辑的，可经由此按钮打开输入助手，当前打开的编辑可由<Enter>关闭，确定当前输入。或由<Esc>键删除已键入的输入。

<Ctrl>+<Enter>: 在当前行下输入一个新行

<Delete>: 删除当前行，也就是当前选中的所有域。

Cut, Copy, Paste: 剪切，复制，粘贴：至少选中此行的一个区，使用复制命令可以复制一行或几行。使用剪切命令剪切一行。粘贴将会在当前行被选中前插入到复制剪切行，如果没有任何域被选中，将会插入到节的结尾。

<Ctrl>+<Home> 滚动到文档开始且标记首节。

<Ctrl+End> 滚动到文档的结尾且标记后节。

<PageUp> 滚动到上一个屏幕且标记最上面的矩形。

<PageDown> 滚动到下一个屏幕且标记最上面的矩形。

多重运算（可扩展的操作数）：

如果同样的操作用于多重运算，有两种编程方法：

1、运算数依行排列且由逗号隔开，例如:

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

2、指令依行重复，例如:

```
LD      7
ADD     2
ADD     4
ADD     7
ST      iVar
```

复杂运算：

进行复杂运算时，以一个开括号开始，然后在下行中完成运算，最后单起一行输入闭括号。

例子：每个循环字符串循环移动一个字符。

(相应的 ST 代码: stRotate := CONCAT(RIGHT(stRotate, (LEN(stRotate) - 1)), (LEFT(stRotate, 1)));)

```
LD      stRotate
RIGHT   (   stRotate
LEN
SUB     1
        )
CONCAT  (   stRotate
LEFT    1
        )
ST      stRotate
```

函数调用：

在操作符区输入函数名。(第一个) 输入参数作为 LD 的操作数。如果有更多的参数，下一个必须是在同一行作为函数名输入的， 之后的参数可以添加到这一行并用逗号隔开， 或添加到下面的行里。 函数返回值将被保存在累加器中，但是 ...

注意，根据 IEC 标准的规定，多返回值的函数调用是不可能的，仅有一个返回值操作才是合理的。

例如：函数 GeomAverage, 该函数调用三个输入参数。第一个参数是预操作数 X7，第二个参数"25"在函数名后给出。 第三个由变量 tvar 给出， 或在同一行或下一行。分配返回值给变量'Ave':

(在结构文本中相应的 ST 代码为：Ave := GeomAverage(X7, 25);)

```
LD          X7
GeomAverage 25
ST          Ave
```

功能块调用，程序调用:

使用 CAL- 或 CALC 操作符 。在开括号前的操作域（第二栏）中输入功能块实例名或程序名。在开括号后写入输入参数：

首栏：为参数名，操作符?=?表示输入参数，?=>?表示输出参数。

第二栏：为实参，在最后参数的后面要以闭括号结尾。

例如：带有两个输入和两个输出参数的 POUToCAll 调用

(相应的 ST 代码为: POUToCall(Counter := MyCounter, Decrement:=2, bError=>Err, wError= > ErrCode);)

```
CAL              POUToCall          (
      Counter:=  MyCounter          ,
    Decrement:=  2                  ,
       bError=>  Err                ,
       wError=>  ErrCode            )
```

调用功能块或程序时，没必要使用所有参数。

注意：根据 IEC 标准限制，不能使用复杂表达式，调用功能块或程序的必须分配一个输入参数。

动作调用：

如函数或程序调用一样。动作名附加到实例名或程序名上。

例如：动作调用 (相应的 ST 代码： Inst.ResetAction();)

```
CAL              Inst.ResetAction    ()
```

方法调用：

如函数调用一样。实例名与所附的方法名输入到第一栏（操作符）。

例如：方法 Home 调用（相应的 ST 代码: Z := IHome.Home(TRUE, TRUE, TRUE);)

```
LD               TRUE
IHome.Home       TRUE                ,
                 TRUE
ST               Z
```

跳转：

跳转 指令由在第一栏（操作符）输入"JMP"、在第二栏（操作数）输入标签调用。标签定义在目标节的标签 区域。注意，无条件跳转前面的声明列表必须以以下命令结尾：ST, STN, S, R, CAL, RET 或 另一个 JMP。 由第一个栏(操作符)中的“JMPC”而不是 “JMP”编辑的条件跳转，不适用上述情况。跳转的执行情况依赖于加载的值。

例如：条件跳转指令：一旦 bCallRestAction 为真，此程序应该跳转到标题为"Cont"节

```
LDN              bCallResetAction
JMPC             Cont
```

2.9.5 菜单

当光标放在 FBD/LD/IL 编辑器窗口时，FBD/LD/IL 菜单在菜单栏中默认是有效的，它为在当前编辑器视图中编程提供了命令。

例子，在 FBD 编辑器的 FBD/LD/IL 菜单

2.9.6 FBD/LD/IL 元素

工具栏

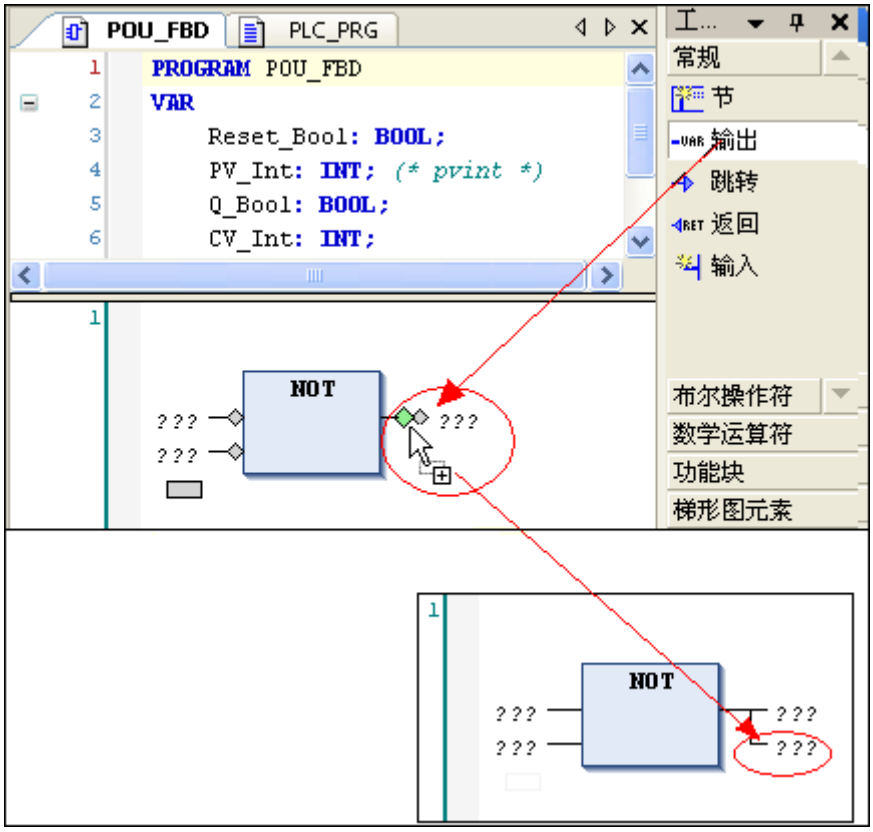
FBD/LD/IL 编辑器提供了工具栏，工具栏中指令支持拖拽的插入方式。工具栏可以通过视图菜单下'工具栏' 子菜单打开。

根据当前激活编译器窗口的不同，可以插入的指令也有所不同（具体见插入指令 插入命令描述部分）。指令分类存放在不同文件夹中：通用（通用指令，如节，输出等），布尔操作符，数学运算符和其他操作符（例如：SEL, MUX, LIMIT 和 MOVE），功能块（例如：R_TRIG, F_TRIG, RS, SR, TON, TOF, CTD, CTU），梯形图单元和 POU（自定义）。

“POUs”类别列出了所有的 POU，这些 POU 由用户定义并和编辑器中当前打开的FBD/LD/IL 对象在同一工程下。如果一个 POU 在其属性中被分配了位图，那么将显示在 POU 名字之前，否则，将有 POU 类型的标准图标。当在应用程序中添加或删除 POU 时，列表会自动更新。

文件夹可以通过鼠标点击打开。在下图中通用类文件是展开的，其它文件夹是折叠起来的。这个图向我们展示了如何从工具箱中拖动并插入一个输出。当前只有通用类是展开的。

从工具栏插入，例如：



想在编辑器中插入一个元素，要用鼠标点击在工具栏中选中它，然后拖动到编辑器窗口。当按住鼠标拖动指令到编辑器窗口中时，在可以的插入位置以特殊符号（菱形）指示。最近的插入位置变成绿色，当放开鼠标，则这个指令将插入到绿色位置。

节

节是FBD 或 LD 程序的基本实体。在 FBD/LD 编辑器中，节垂直排列。每个节通过左侧的一系列节号标示，由逻辑式，算术表达式，程序，功能或功能块调用指令，跳转或返回指令构成。

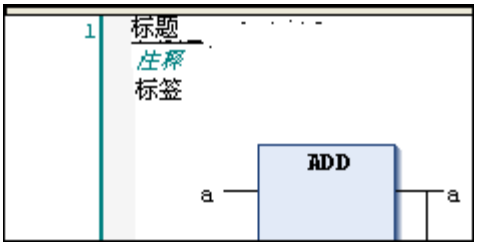
IL 编辑器， 基于 FBD 和 LD 编辑器的通用编辑器， 也使用节元素。如果一个对象最初是在 FBD 或 LD 编程，然后 转换成 IL，此节将在 IL 程序中显示。相反如果一个对象最初是在 IL 中编程，你至少需要一个含有所有指令的节元素，但你也可使用更高级的节来构建程序。例如，转换成 FBD 或 LD。

节可选择性的分配到一个标题，注释和标签：

在“FBD，LD，IL 编辑器选项 ”中打开和关闭标题和注释框.如果此选项被激活，直接点击节的上沿，可以打开一个编辑文本编辑标题。若要加入评论，直接在标题栏下打开一个编辑框。此注释可能为多线形。分支通过<enter> 插入，注释文本的输入由<ctrl> + <enter>终止。

使用插入标签命令添加标签，该标签通过跳转分配地址。如果一个标签已被定义，它会显示在标题和注释框下；或当没有标题和注释框时-直接在节上沿的下面。

标题，注释和标签在节中的位置



节可以在“注释状态”设置，影响的是，此节不能执行但可以像注释一样显示和处理。

就目前选定的（节光标位置 6），默认可以使用复制，剪切，插入和删除命令。

注意：用鼠标右键在标题、注释或者标签上点击(光标位置 6)，仅仅是选择了条目而不是整个节。因此该默认命令对节本身没有影响。

要插入一个节，可以使用命令插入节或从工具箱拖动它。一个节所有的元素都可以通过在编辑器中拖&放来进行复制或移动。

请注意可以通过插入“分支”来创建"子节"。

RET 网络：

在线模式时，一个附加的空网络将被自动显示在已有网络下。它将被定义为“RET”而不是网络号。它反映了执行在什么位置返回调用 POU，并提供一个可能的停止位置。

FDB/LD/IL 的输出

取决于在 FBD 或 LD 的所选位置，输出将直接插入到所选择输入的前边（光标位置 2），在所选输出的后面（光标位置 4）或在一节的结尾（光标位置 6）之后。在 LD 节中输出将由线圈显示。或者从工具箱拖动输出元素或通过在编辑视图中通过拖&拽复制或移动之。

在插入文本之后，字符"???" 由被分配的变量的名字取代。为此目的可通过按钮 ... 使用输入助手。

在 IL 中，输出是通过 LD 和 ST 指令调用。点击这里 ，看操作符和修改的描述。

跳转

取决于在 FBD 或 LD 中所选的位置 ，跳转将被直接插入到所选输入的前部（光标位置 2），在所选输出的后面（光标位置 4）或在一节的结尾（光标位置 6）之后。

或者从工具箱拖动跳转元素或在编辑视图中通过拖&拽复制或移动它。

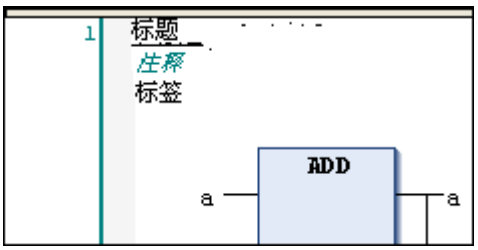
插入之后，自动的输入的"???" 将由跳转分配的标签所取代。

在 IL 中，跳转通过一个 JMP 指令被插入。

标签

在每个 FBD / LD 或 IL 节 的注释栏下方有一个输入文本，来定义一个标签（见插入标签命令）。此标签是一个可选的标识符且当定义 跳转 时可以确定其地址。它可以包含任何字符。

标签在节中的位置



见“FBD, LD, IL 编辑器选项对话框”，来定义注释和标题的显示。

FBD/LD/IL 运算块

插入 FBD, LD 或 IL 节中的运算块是一个复杂的元素，它可以是函数，如定时器、计数器，算术运算或程序，IEC 函数和 IEC 功能块。运算块可以有输入和输出，并可以由用户系统库提供或用户自行编写。不管怎样至少要有一个布尔输入和输出。

在 FBD, IL 中的用法

通过插入运算块 命令或插入空运算块命令，一个运算块可以插入到一个 LD 节左部（像一个触点）或 FBD 节中。可以从工具箱插入或通过拖拽在编辑器中复制或移动。请在插入运算块查看详细描述。

在 IL 中的用法：

在一个 IL 程序中，插入带有参数的 CAL 指令来代表运算块元素。

万一元素块界面被改变时，在当前工具箱中，不必通过“参数更新”命令重新插入运算块中，运算块参数（输入，输出）就可以更新。

FDB/LD/IL 中的返回指令

使用返回指令可退出 FBD, LD 或 IL 程序组织单元。

在 FBD 或 LD 节中，返回指令可以平行放置或放在前元素的结尾。如果一个返回的输入为真，此 POU 的进程将会立即停止。

或者从工具箱拖动元素或在编辑器内从其他位置进行复制或移动。

返回元素

在 IL 中，使用复位指令。

FDB/LD/IL 中的分支/线圈

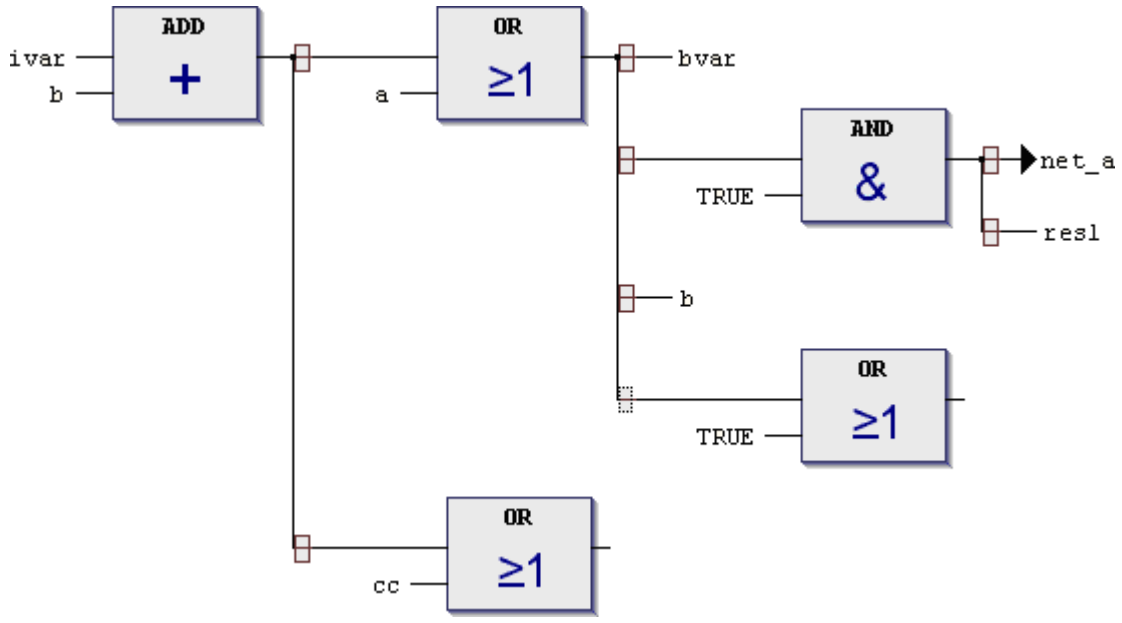
FBD, LD

在 FBD 或 LD 节中，分支或线圈从当前光标的位置分裂此进程。该进程将会由两个子节继续，从上而下的分别执行。每个子节将再有分支，在节中允许多分支。

每个子节有一个自己的"标志"（一个直立的矩形符号），它可被选定（光标位置 11），以在此分支

上执行动作，如剪切和粘贴操作。

分支标识的例子

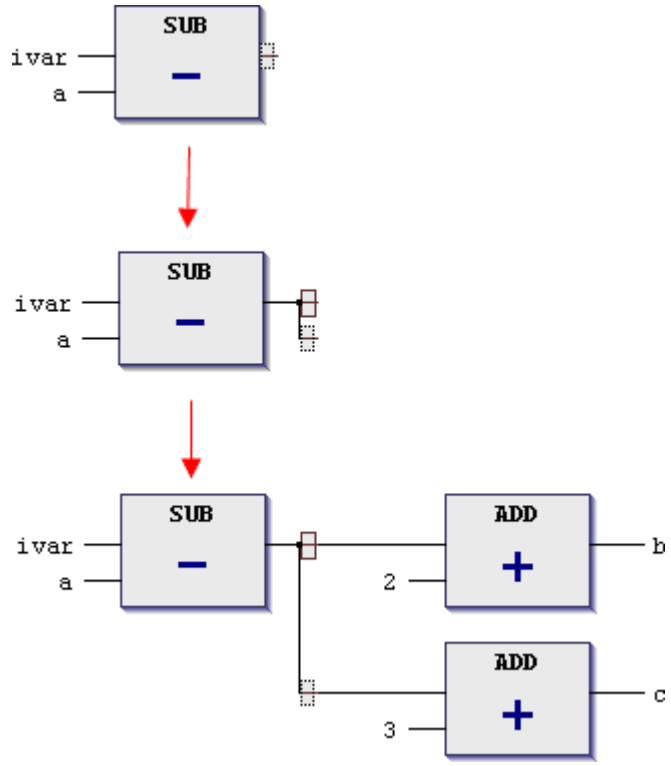


在 FBD 中通过 “插入分支” 命令插入分支。也可以从工具箱中拖拽元素。在'插入分支"中查看可能的插入位置。

注意：目前在子节中还不能进行剪切&粘贴。

请看下图的例子：一个分支已经插入到了 SUB 运算块输出。这创建了两个子节，每个可以由其子网标记选择。之后，在每个子节上都添加了一个 ADD 运算块。

在 FBD 中的例子，插入分支

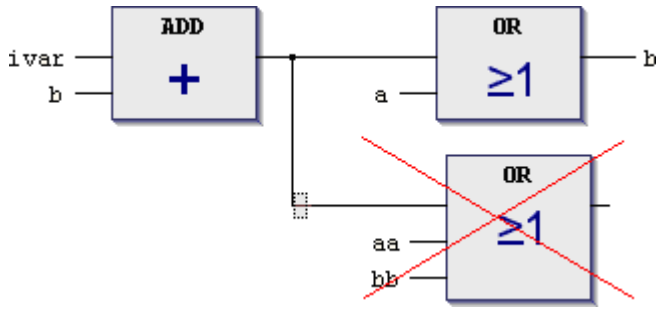


通过断开分支的两个支的仅有联系，可以创建各自的子节。当分支的一支被选中时，通过断开命

令可以断开。

要删除一个子节，首先把子节上的所有元素都删除，也就是那些被定位在子节标记右边的元素，然后选择标记并使用标准命令删除或<Del>。请看下图。3 输入 OR 元素必须在删除下一级的子节之前被删除。

删除分支和子节



反之亦然，一个分支的两支或两个子节可以通过联结命令再次连接。为此目的，在其他支或子节的一个或几个输入中的输出必须被选中，同时需一直按下<Ctrl>键和执行命令。

在线模式下的执行：

分支将从左到右然后从上到下的执行：

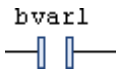
IL (指令列表)

在 IL 中一个分支或线圈由相应的指令顺序代表。

触点

这是 LD 元素。

在 LD 中左部分的每节包含一个或几个触点。触点由两行平行线代表。



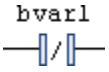
触点从左到右传递"ON" (TRUE) 或 "OFF" (FALSE)状态。

一个布尔型的变量被分配到每个触点。如果此变量为真，此条件将从左到右传递，且最终到达此节的右部，否则右部为假。

多个触点可串联也可并联，两个并联触点中只需一个触点为真，则平行线传输真，多个串联的触点要都为真，最后一个触点才能传输真，

所以此触点的排列或与并联电路，或与线圈相符和。

触点也能在触点符号取反,，即触点符号中的斜线。



在这种情况下，输入信号取反值将会被复制到一个合适的布尔变量，因此：一个取反的触点只能联结那些合适的布尔变量为假的情况，

触点能通过 FBD/LD/IL 菜单中的下列命令插入一个 LD 节：插入触点 ，或插入触点（右），以及插入平行触点（上)， 插入平行触电（下），或直接从工具栏中拖出。

FBD, IL

前在FBD 或 IL 中工作，此命令将无效，但是插入到一个 LD 节中触点线圈将由相应的FBD 元素或 IL 指令替代。

线圈

这是一个 LD 元素。

在 LD 网络的右部，有一些由括号代表的所谓线圈。

他们只能平行的排列。线圈从左到右的传送连接值，并且赋值给一个合适的布尔变量。它的值为开（真）或关（假）可以显示出来

触点和线圈也能在线圈符号取反，即线圈符号中的斜线。

在这种情况下，输入信号取反 值将会被复制到一个合适的布尔变量，因此：一个取反的触点只能联结那些合适的布尔变量为假的情况，

线圈能通过插入任务 命令被插入到一个节中，此任务默认为FBD/LD 菜单的一部分，或者是从工具栏，梯形元素类 中拖出。

FBD, IL:

前在 FBD 或 IL 中工作，此命令将无效，但是插入到一个 LD 节中触点线圈将由相应的 FBD 元素或 IL 指令代表。

FDB/LD/IL 中置位/复位

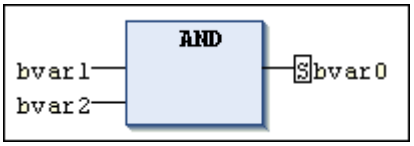
FBD 和 LD

在 FBD 或相应的 LD 线圈中，一个布尔型输出能被置位或复位。要在设置状态之间改变，请在选中输出时，从右键菜单选择命令置位/复位。此输出和线圈将有 S 或 R 标识。

置位：如果真值已经到达一个置位的输出或线圈，此输出/线圈将为真且保持真。在这个位置上只要已经运行此值将不能被覆盖。

复位：如果真值已经到达一个复位的输出或线圈，此输出/线圈将为假且保持假。在这个位置上只要已经运行此值将不能被覆盖。

在 FBD 中置位，例子



IL

在指令列表中，此 S 和 R 操作符被用于置位或复位一个操作。

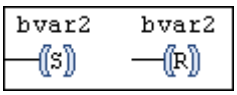
置位/复位线圈

线圈 能被定义为置位或复位线圈。在线圈符号：(S)中，由" S "置位的一个线圈。置位线圈的布尔变量被覆盖真值。这就是说，变量一旦设置为真值，将一直是真。

在线圈符号：(R)中，由" R "复位的一个线圈。复位线圈的布尔变量被覆盖假值：此变量一旦设置为假，将一直为假。

在编辑器中，置位和复位线圈，能从工具箱（梯形元素类： ）通过拖动直接插入。

置位线圈，复位线圈，例子



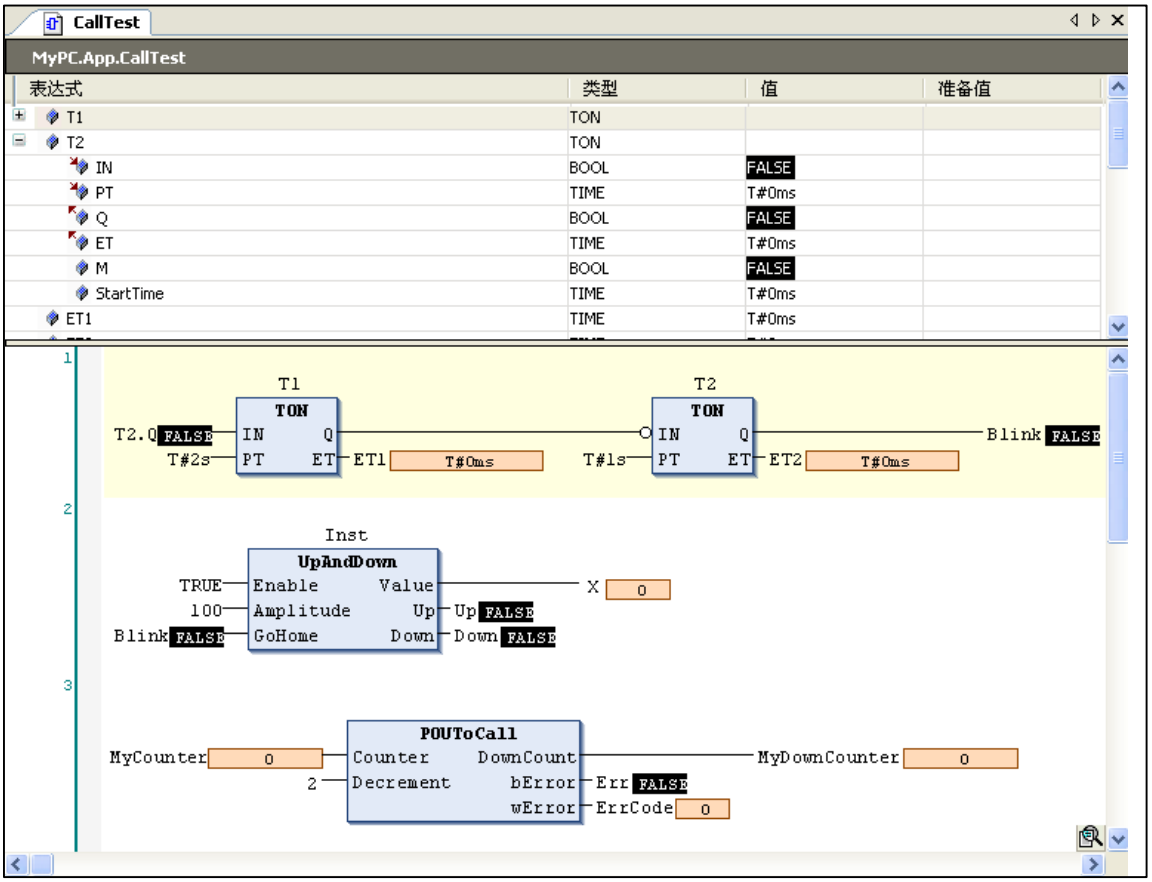
2.9.7 在线模式下的 FBD/LD/IL 编辑器

在线模式下，FBD/LD/IL 编辑器为监视，写和强制变量，及控制器中的表达式提供视图。调试功能（断点，步进等）也是有效的。

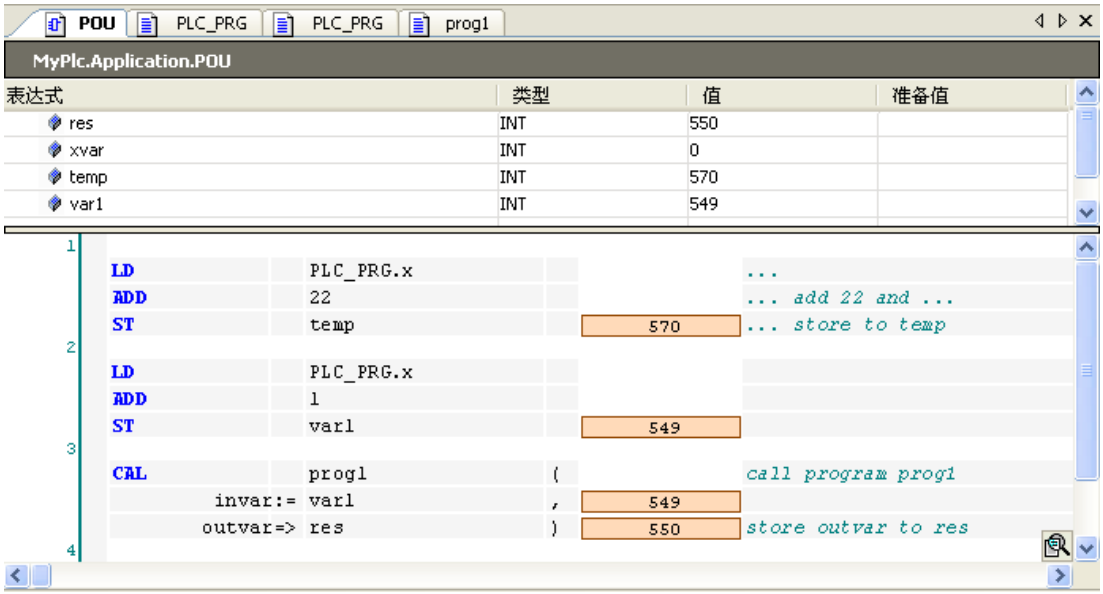
监视

如果离线监视在选项对话框中不能被激活，通过每个变量后的小监视窗口，或另外的监视栏显示实际值（离线监视），它在 FBD 或 LD 编辑器中被补充。

FBD 程序在线视图的例子

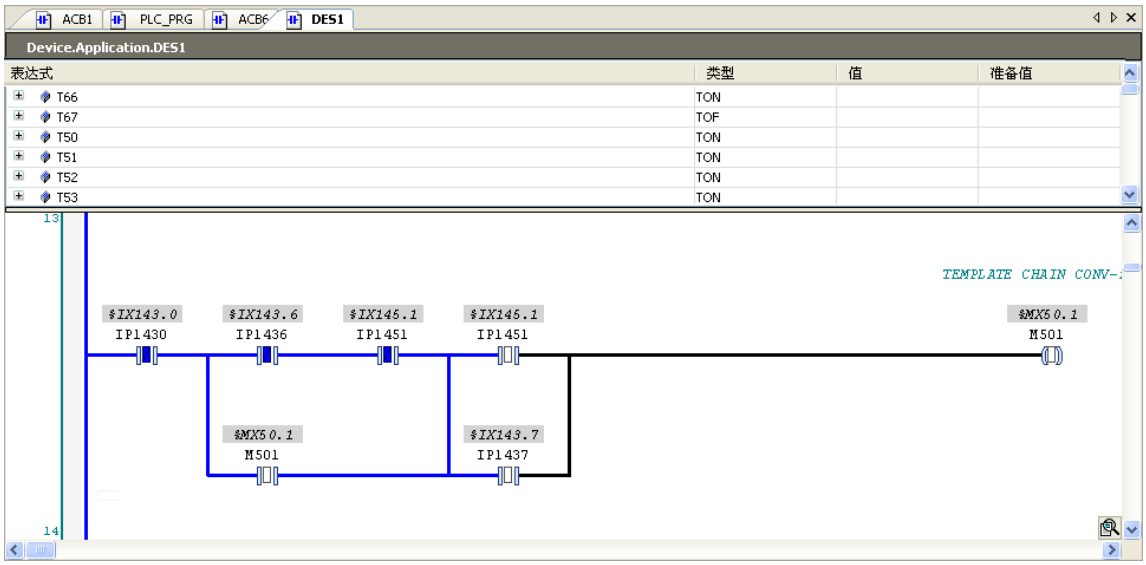


程序在线视图的例子



在在线视图中，梯形网络可以动画连接：为TRUE 值的连接显示为蓝色粗体，为FALSE 值的连接显示为黑色粗体，无值的连接或无模拟量值的连接显示为标准样式（未加粗的黑色字体）。（提示：连接值是由监视值计算得来的，不是真的能流。）

LD 程序在线视图的例子

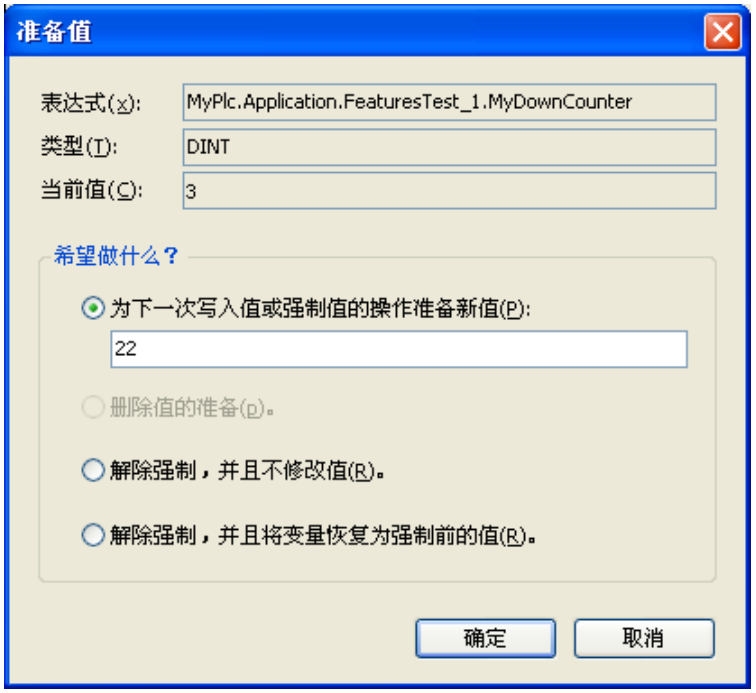


功能块POU 的在线视图：执行部分没有值显示在监视窗口，但是表达式的值和离线监视文本将会显示三个问号。

变量的强制/写入

在线模式时，您可以在声明编辑器或者实现部分为强制或写入变量准备一个值。在实现部分鼠标点击变量，会打开下面的对话框：

对话框－准备值



你会发现变量名已经由设备树内的路径，类型和当前值完成。通过激活相应的条目你可以选择是否想：

- 准备一个新值，且输入到文本编辑器中
- 删除一个准备值
- 释放实际强制值
- 释放实际强制值且复位到强制前的分配值

当执行菜单命令强制值（在线条目 ）或按下 F7 时，选择的动作将被执行。

断点或者停止位置：

被定义为用于调试的 断点的可能位置，基本上是那些变量值可改变的地方（说明）、程序流分支的地方、或者其它 POU 被调用的地方。就是下列位置：

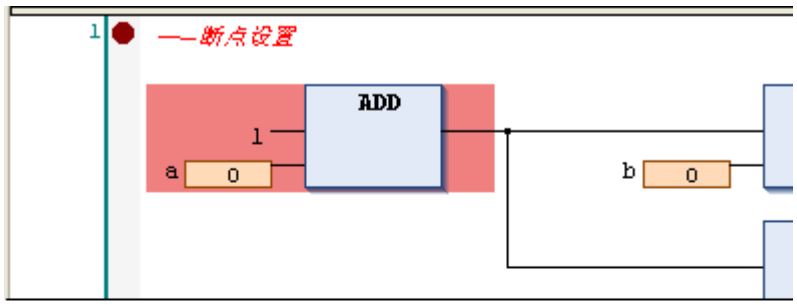
- 断点将被用于网络中第一个可能的位置
- 在块上，如果它包含了说明；但不是象 ADD、DIV 那样的运算块；请看下面插入的注意
- 在赋值时
- 在 POU 结束时返回到调用者的地方；在线模式时一个空网络将自动显示，它被定义为“RET”而不是网络号

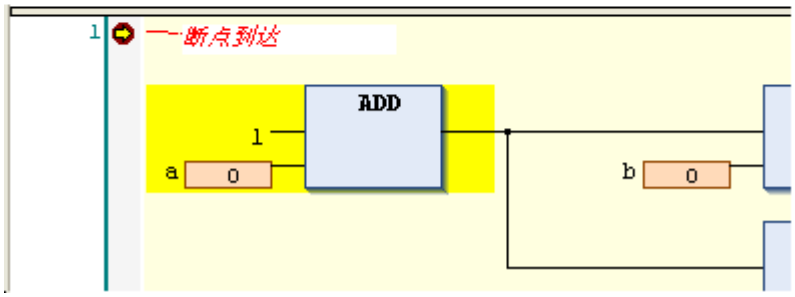
注意：当前您不能在网络的第一个块上设置断点。但是如果一个断点被设置在整个网络上，停止位置将自动应用于第一个块。

您可以在断点对话框中的选择列表中查看所有当前可能的位置。

一个包含任意活动断点位置的网络将用断点图标（红色的实心圆）标识在网络号右边，网络中第一个可能的断点位置用红色阴影矩形背景标识。无效的断点位置用红色空心圆或红色矩形框表示。

例如 - 断点设置或到达的位置

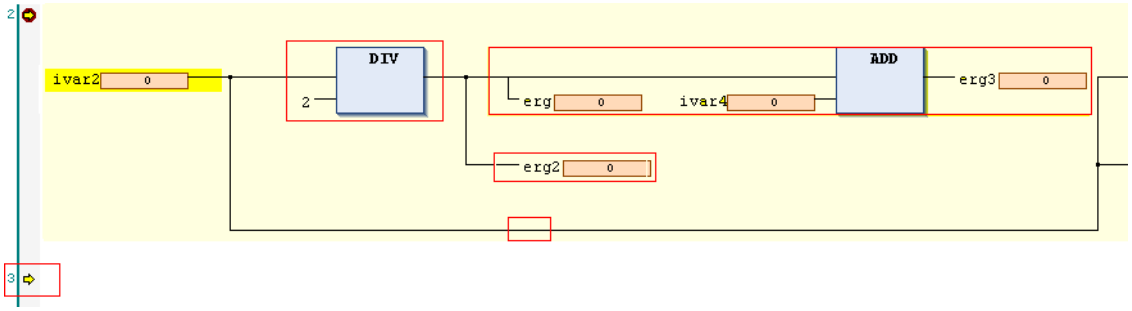




下面是在跳转中有可能停止的位置：在一节的开始，在运算符中，在最后一节之后。

在 FBD 中的停止位置

下图中显示了可能停止的位置：当前停止位置由黄色阴影显示。红色矩形框内也为可以停止的位置。

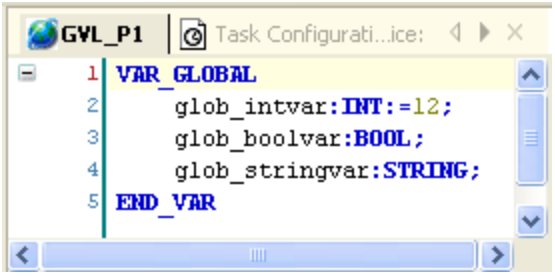


注意：如果是在一个方法中设置了断点，那么在所有被调用到的相关方法中都会自动设上断点。例如一个接口管理方法被调用到了，那么在实现这个接口的功能块、和使用了这个方法的功能块中，都会自动设上断点。如果一个方法是通过指向功能块的指针来调用的，那么在这个功能块、和所有使用这个方法的功能块中，都会自动设上断点。

2.10 全局变量列表编辑器

全局变量列表编辑器是一个用于编辑全局变量列表的声明编辑器。它根据文本编辑器的当前设置工作。在线模式下也以声明编辑器的形式出现。声明语句必须以 "VAR_GLOBAL" 开头且以 "END_VAR" 结束。这些关键字自动生成，关键字之间输入全局变量的有效声明。

全局变量列表编辑器



2.11 库管理编辑器

工程中可以通过库管理器来添加库并对库进行操作。用户可以通过库对话框安装库和定义库文件夹。库对话框也是库管理器的一部分，可以在编辑器窗口或者菜单栏（默认在“工具”菜单）中通过相应命令打开。

由“库管理编辑器”插件提供库管理编辑器。用户可以通过添加对象命令将这个编辑器加入工程。这一操作与在 POU 视图或设备视图窗口中向设备和应用添加对象类似。这些位置只可插入一个库管理器对象，其名称与“添加对象”对话框中定义的相同。

POU 视图中的库管理器对象

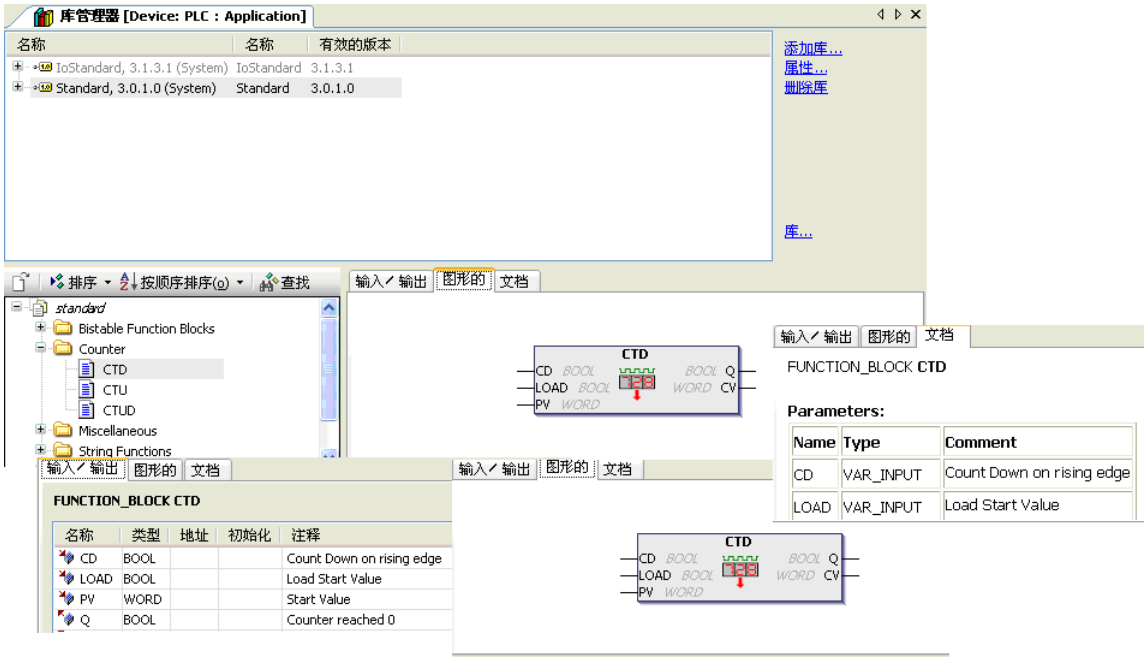
在编辑器窗口点击编辑对象命令或者双击对象，都可以打开库管理器。

库管理器的相关编译信息会在消息窗口单独列表显示。

库管理编辑器窗口

通过编辑对象命令或双击对象可以在编辑器视图窗口中打开库管理编辑器。

库管理编辑器



编辑器窗口的结构

对话框的上部显示当前包含在工程中的库。其中包括：

名称：建立一个库时在工程信息摘要对话框里指定的标题、版本和公司名称。

命名空间：如果没有在一个库的工程信息中明确地指定其命名空间，那么默认的命名空间为“<库名>”。库的命名空间可以作为访问某个指定模块的标识符前缀，而工程中可能还有其它模块与这个模块同名。您可以在“属性”对话框中修改一个被调用的库的标准命名空间，以方便在（工程内部的）局部使用。请看下面的相应命令。您也可以点击这里查看有关库的命名空间的信息。

有效版本。“有效版本”是在库属性中定义的、当前使用的库版本。

由插件自动包含到工程中的库以灰色显示，而手动添加的库则显示为黑色。

库名之前的图标表示库的类型：

	CoDeSys V3.x 库（包含版本信息）
	无法找到引用信息或者引用了无效的库（请查看消息视图中“库管理器”列表里的相应信息）。参见“重新装载库”命令。

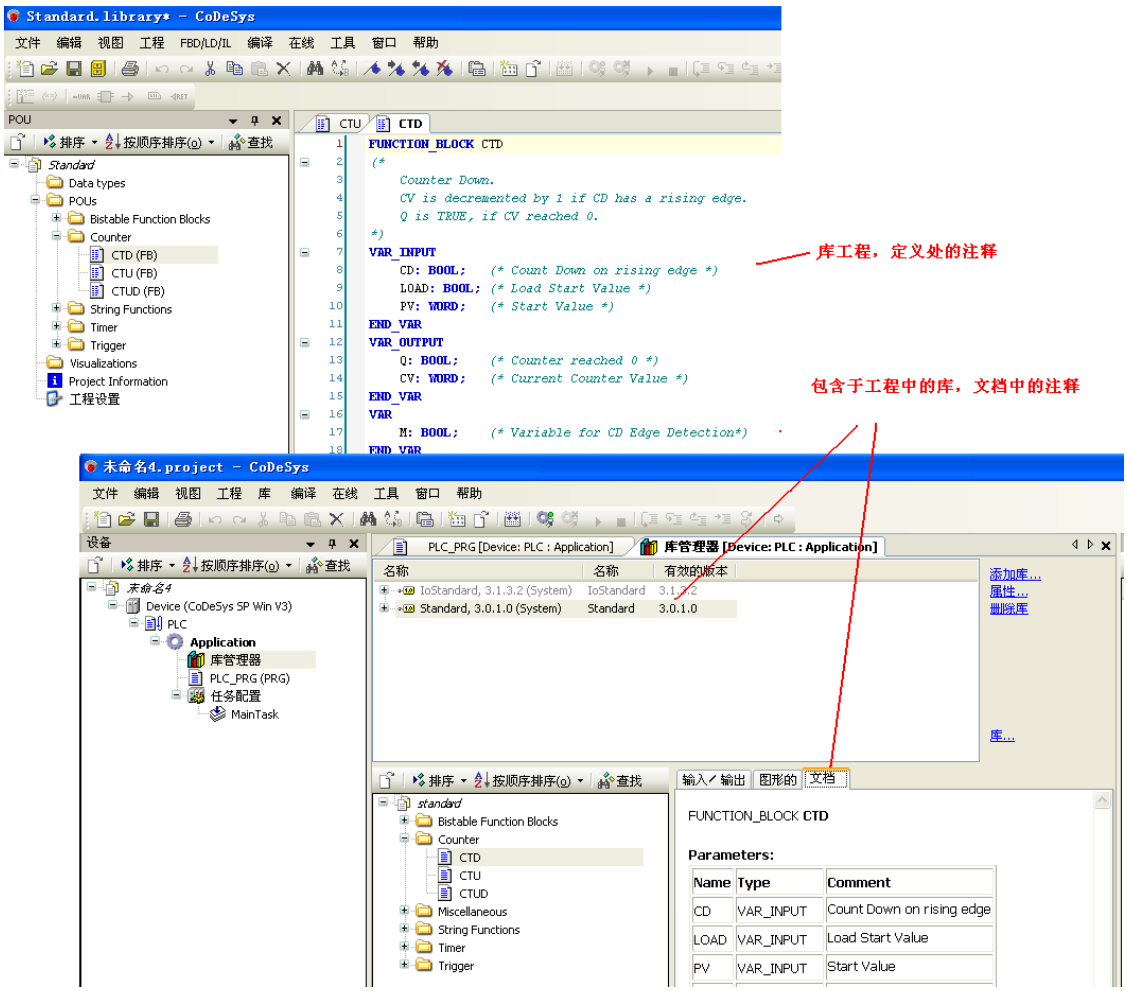
如果一个库依存于其它库（引用库），那些库——如果有效——也将被自动包含进来，并和图标一起显示于子树中。用户可以使用相应的加（或减）符号展开或折叠子树。请参见“库管理器对话框”（上文）中“VisuInputs”库的例子。

在编辑器的左下角，显示了当前选中的库中的模块。您可以使用分类、查找等常见按钮来处理模块树状图。

用户可以在右下角获取下列信息：

文档：左边区域中选中的模块组件在这里以表格形式显示了变量名、数据类型和可能在建立库时添加到声明组件中的注释。由此可见，插入这样的模块有利于自动地为用户提供模块文档。

添加了注释的库模块举例



输入/输出标签：表格中列出了当前选中的库模块组件的变量名、（数据）类型、地址、初始值和注释。这些数据和在库中定义的一致。

图片：IEC 模块程序在此处以图形形式表示出来。

编辑器窗口中的按钮和命令

选中一个或几个库时，窗口中会出现以下的命令选项。这些命令与“库”菜单中的命令相对应。在默认设置下，只要打开库管理编辑器，菜单栏中就会显示这些命令：

添加库... 向当前工程添加库；前提是系统中已安装了这个库。

属性 用于设置库的命名空间和版本（当这个库在其它工程中作为“引用库”时需要用到）、是否可见以及访问方式。

移除库： 列表里选中的库将被从工程中删除。

库... 用于定义库的位置以及安装、卸载库。

如果用户试图插入一个已包含在工程中的库，系统将给出提示。

库菜单

打开库管理器，菜单栏默认显示“库”菜单，其中包含库管理器对话框命令。

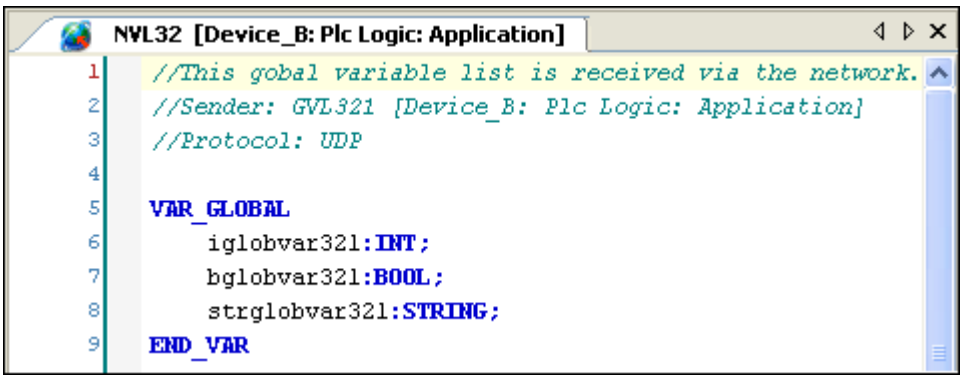
请注意在默认设置下，库...命令位于工具菜单中。

点击这里可以查看关于 CoDeSys 库管理的常用信息。

2.12 网络变量列表编辑器

网络变量列表编辑器（NVL 编辑器）是一个用于编辑网络变量列表的声明编辑器 。此编辑器工作在文本编辑器 的当前设置状态下，在线状态下，此编辑器的布局和变量声明编辑器的布局相似。在网络变量列表编辑器中， 变量声明必须以关键字"VAR_GLOBAL" 开始， 以关键字"END_VAR"结束。这两个关键字会在打开编辑器的时候自动加入。在这两个关键字之间，用户可以声明合法的全局变量。

NVL 编辑器

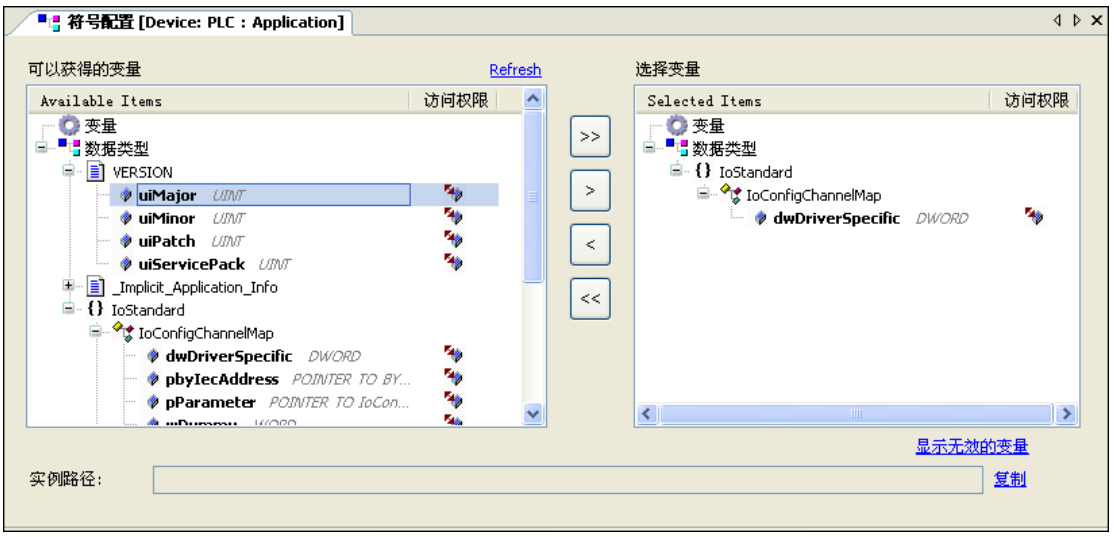


2.13 符号表配置编辑器

符号配置编辑器可以创建符号描述，用于配置外部资源（如 OPC Server）访问工程中的变量时的权限控制。关于符号描述的介绍请见符号配置。

双击设备树中的某个符号配置对象，在打开的符号配置编辑窗口中可以对某应用进行符号配置。

符号配置编辑器



在编辑窗口的左边，当前应用中的可用变量以树形结构显示在“变量”和“数据类型”节点下，通过缩进的方式下一级显示 POU、GVL 及数据类型名称，再下一级中显示变量名称、数据类型和访问权

限。点击刷新，可更新当前可用变量的信息。使用编译器选项总是链接，一个未被编译的对象能被重新解释和下装到 PLC 中。然后所有在这个对象中声明的变量将有效。此外，编译指示{attribute 'linkalways'}可用于符号配置中，使未编译的变量有效。

在编辑窗口右边，同样以树形结构显示当前选中的变量。这些选中的变量可以被导出到符号文件中。点击显示非法变量按钮，将以红色字体显示出当前应用中所有的非法变量，如变量声明发生变动。

通过点击编辑窗口中心的单箭头按钮，可以将左边可用变量树中选择的项（或节点）添加到右边已选变量树中，反方向的单箭头按钮对应的是相反操作。而双箭头按钮是对整个树进行操作。

在编辑右边的窗口中，可以通过鼠标点击符号的访问权限列修改选中项的访问权限。每次点击鼠标依次设定：读写、只写、只读。在对话框的底部，显示当前选择项的实例路径，可以用“复制实例路径”按钮将其复制到剪贴板，这对在文本编辑区插入一个很长的路径是很有用的。

2.14 任务编辑器

任务配置为应用程序的过程控制定义一个或者多个任务。因此任务配置是应用程序的基本资源对象，而且在设备窗口上必须可用。

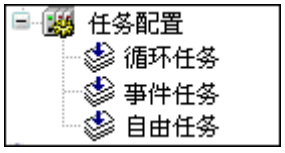
与 CoDeSys 2.3 不同，此版本任务配置的树结构是不可见的，并且不能在编辑器中处理。但是它被集成在了设备窗口的资源树中。任务配置是每个应用程序必需的资源对象，可以通过添加对象命令来添加。

在任务配置树的最高位置总能够找到“任务配置”目录。在任务配置下面是当前定义的一些任务，每个任务通过名字来区分。每个任务调用的 POU 没有显示在任务配置树中。

任务树可以通过设备树的相关命令进行编辑（添加、复制、粘贴或者删除任务）。例如，用‘添加对象’来添加一个新的任务。

每个任务都在任务编辑对话框中被配置，该任务编辑对话框还提供在线监视视图。任务配置的可用参数由配置文件设定。

在设备树的应用下进行任务配置



任务是 IEC 程序处理过程中的一个时间单元。任务通过名字，优先级以及触发条件类型来定义。这个条件可以定义为时间（周期、非周期）或者触发该任务的内部或者外部事件； 例如，某工程全局变量的上升沿或者控制器的中断事件。

每个任务都可以指定一系列由该任务触发的 POU。如果该任务在当前周期被执行，那么这些 POU 将在一个周期时间内被处理。

优先级和条件综合决定任务的执行顺序。

每个任务都可以配置一个看门狗（时间控制）；可能的设置参数由配置文件决定。

另外，还存在系统事件（例如，开始、停止、复位）与工程 POU 执行的连接的可能性。

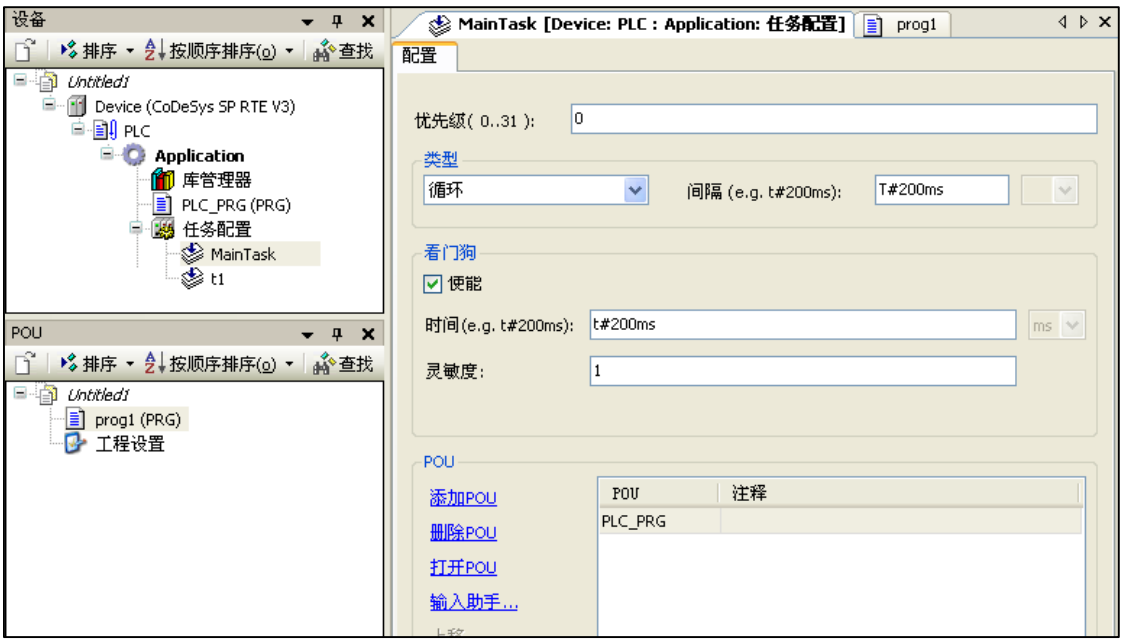
在线模式下可以监视任务的处理过程。

2.14.1 任务编辑器用法

“任务配置编辑”插件提供了处理任务配置（应用的一种基本的资源对象）的功能。

双击设备视图窗口中的 任务配置对象或者执行‘编辑对象’命令可以打开该编辑器。

任务编辑器，任务配置对话框



与 CoDeSys 2.3 不同，本版本的任务配置的树结构是集成在设备视图窗口的资源树中的。在这里可以通过添加对象命令添加任务配置对象和特定的任务。

双击设备树上的任务配置对象，将打开任务编辑器，其中的对话框可进行如下配置：

属性对话框：任务基本配置设定信息。

配置对话框：对每个任务进行配置，任务特定参数可以通过参数对话框进行任意扩展。

系统事件对话框：设置 POU 的调用与系统事件之间的连接。

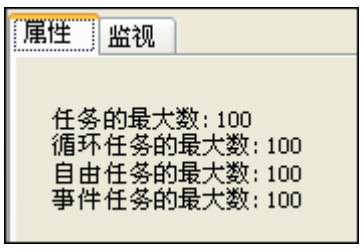
对话框中的配置参数依赖于当前使用的设备（目标）。尚未实现：如果基本描述通过用户特别定义被扩展，那么这些参数也将显示在附加的参数对话框中。

注意：请不要在多个任务中使用相同字符串名的函数（参见 standard.library），这将因程序重名而引起错误。

属性对话框

当在任务配置树中选择最上面的对象，将在任务编辑器窗口中打开属性对话框。

任务配置属性对话框

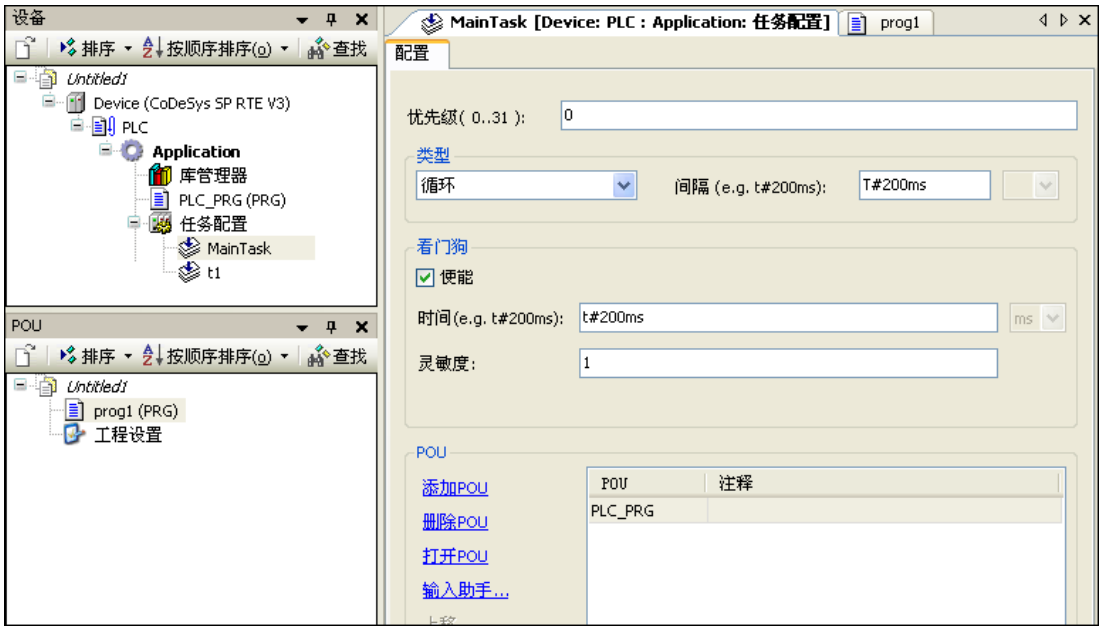


属性对话框将显示配置文件提供的当前任务的配置信息，例如：每种任务类型允许的最大任务数。

配置对话框

当在设备视图的任务配置中插入一个任务（通过'添加对象'命令）时，将打开设置任务参数的任务编辑器对话框：

任务配置对话框



插入希望的属性：

注意：可以在设备树中通过编辑目录来修改任务的名字。

优先级（0-31）：（0 到 31 之间的一个数字，0 是最高优先级，31 是最低优先级），

类型：有下列可供选择的任务类型列表：

循环：任务将按照'间隔'（参看下面的内容）项给出的时间定义被循环执行。

自由运行：程序一开始运行任务就被处理，一个运行周期结束后任务将在下一个循环中被自动重新启动。

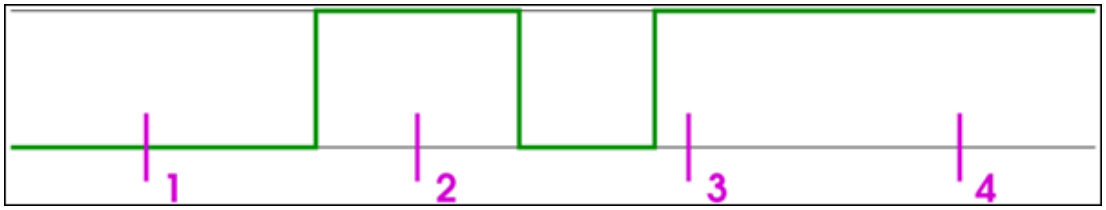
状态：如果在事件项定义的变量为真时，任务将开始执行。

外部事件：一旦事件区定义的系统事件产生，任务将被执行。目标系统决定支持或者提供选择列表中的哪些事件。（不要和系统事件混淆）

状态和事件之间的区别：

状态驱动任务开始的条件只要指定事件为 TRUE 时就可以了，而事件驱动任务时需要让事件从 FALSE 变到 TRUE。如果任务调度的采样率太低, 事件的上升沿可能不会被发现。

下列示例说明了状态和事件对任务驱动的不同结果：



在示例的 1-4 点(紫色)说明了任务对不同类型所做出的不同响应：

行为点	1	2	3	4
状态	不开始	开始	开始	开始
事件	不开始	开始	不开始	不开始

依赖于任务类型选择的必要项：

间隔(如果事件需要一个时间作为入口条件,那么'周期'类型与'外部事件触发'类型相区别是必要的)：时间阶段,这个时间阶段后任务应该重新开始执行。如果输入一个数字，就可以在编辑区后面的下拉框中选择希望的单位：毫秒[ms]或者微秒[µs]。窗口重绘后[ms]-格式型的输入将以时间格式显示 (例如, "t#200ms")；但是也可以直接输入时间格式的值。[ms]型的输入总是以单纯的数字显示(例如, "300")。

事件(对于'事件'或者'外部事件触发'类型的任务是必要的)：一个全局变量，该全局变量一旦产生上升沿，将触发任务开始执行。使用…按钮或者输入助手<F2>可以打开所有可用的全局事件变量列表。

注意：如果驱动任务的事件来源于一个条目,那么至少存在一个不被事件驱动的任务.否则 I/O 将不会被更新,并且任务也将从不会执行。

看门狗

每个任务都可以配置一个时间控制装置(看门狗)。如果配置文件支持扩展的看门狗配置，可以预定义上下限和定义一个默认的看门狗时间，以及一个以百分比定义的时间。

使能：当这个参数被激活(☑)，那么一旦任务处理时间超过'时间'项(参看下列描述)定义的时间，任务就会以错误状态结束。如果选项"中断时升级 IO"可用，输出项设置为预定义的缺省值。

下列情况是可能的：

1、连续超出时间，则有下表：

灵敏度	周期内的异常个数
0, 1, 2	1
3	2
...	...
n	n-1

2、一次超出时间：如果当前循环时间大于（时间*灵敏度）时的异常。例如：Time=t#10ms, Sensitivity=5 -> 一旦任务运行超过 50 毫秒就出现异常。这用来在首次循环时检测无限循环。

时间(例如：t#200ms)： 看门狗时间；描述见上：使能。这个阶段有效后看门狗就被激活。 就是说，如果超过当前设置的看门狗时间（见下），任务将在错误状态（“异常”）下被终止，这里考虑了当前设置的灵敏度（见下）。如果在 PLC 设置对话框中的选项“在停止时更新 IO”是可用的，输出将被设置为预定义的默认值。

灵敏度：个数；描述见上：使能

注意，对于使用 CmpIecTask.library 库函数提供的特殊 SPS 循环，可以禁用看门狗。通常由于初始化过程中 SPS 循环需要更多的时间，因此这是有用的。。

为任务句柄声明一个适当的变量（类型为 RTS_IEC_HANDLE）

hIecTask : RTS_IEC_HANDLE;

按照下面的格式使用接口函数可以执行禁用（和重启）：

hIecTask := IecTaskGetCurrent(0);

IecTaskDisableWatchdog(hIecTask);

... // Code that is protected against watchdog

IecTaskEnableWatchdog(hIecTask);

POUs:

被当前任务控制的 POUs 的 POU 名字和随意的注释被列在这里的表格中。 表格的左面有一些编辑命令：

通过添加 POU 命令打开输入助手对话框来定义一个新的 POU。 在这里可以选择工程中的任意一个程序变量。

通过输入助手命令打开输入助手，并且在列表条目中选择另外一个程序来代替一个程序的调用。

选择表格中的一个 POU，使用删除命令可以删除一个 POU 调用。

使用打开 POU 命令可以在对应的编辑器中打开当前选中的程序。

POU 调用在列表中从上到下的顺序决定了在线模式下的执行顺序。可以通过向上和向下命令来移动当前选中条目在列表中的位置。

参数对话框

参数对话框只有被目标定义了后，才在任务编辑器对话框的标签页中可用。该对话框提供了任务的创建者详细属性（姓名、类型、值和描述）。

2.14.2 在线模式下的任务编辑器

对于任务配置中定义的任务的执行应该遵循以下原则：

一旦任务的条件被满足，该任务就执行；例如，指定的时间到或者该任务的条件（事件）变量产生了一个上升沿。

如果多个任务的条件都有效，那么将执行优先级高的任务。

如果多个任务的条件有效并且优先级相等，那么将优先执行等待时间较长的任务。

程序将按照它们在任务编辑器中的顺序（上下）被调用。如果一个程序用相同的名字被指派给设备树和一个库中的应用以及 POUs 窗口中的全局工程，那么直接指派给应用的程序将被执行。

监视，任务编辑器的在线视图

当任务配置树的最高节点被选择， 在进一步的标签页中除了属性对话外还有监视对话框可用。在线模式下，该表格视图中将显示任务状态，以及周期和周期时间的目前统计。这些值的刷新周期和 PLC 值的监视周期相同。

任务配置，监视

对于每个任务，每行显示下列信息：

任务	任务配置中定义的任务名字
状态	可能条目： 未创建：最后一次更新后还未开始；尤其是事件型任务 创建：任务已经被运行时间系统确认， 但是还未被设置为操作状态。 有效：任务处于正常操作状态 异常：任务出现异常
IEC-周期计数	启动应用程序后运行周期的个数；如果目标系统不支持此功能则为“0”。
周期计数	已经运行的时间周期数。取决于目标系统，它可以等于 IEC-Cycle Count，或者更大值，此时即使应用程序没运行周期也一样被计数。
最后一个周期(µs)	最后一个运行时间，单位 µs
平均周期 (µs)	所有周期的平均运行时间，单位 µs
最大周期 (µs)	所有周期的最大运行时间，单位 µs
最小周期 (µs)	所有周期的最小运行时间，单位 µs
抖动时间(µs)	最后一次抖动时间，单位 µs
最小抖动时间 (µs)	最小抖动时间，单位 µs
最大抖动时间 (µs)	最大抖动时间，单位 µs

*抖动：任务开始运行到操作系统标识该任务已经运行之间的时间

当把光标放在一个任务名字上，可以通过上下文菜单的复位命令来将该任务的值复位成 0。

2.15 跟踪编辑器

跟踪功能由“跟踪编辑器”插件提供。该功能可记录并读取一段时间内变量在 PLC 上值的变化。待跟踪的变量值周期性地被写入运行系统中特定大小的 Codesys 缓冲区中，用户可以以时间为轴，以图表的形式查看。用户还可以定义跟踪的起始时间，以及显示的时间范围。

跟踪编辑器对话框中可以定义跟踪采样的配置以及显示。用户可以同时跟踪和显示多个变量。

在线模式下，为了根据当前的跟踪配置开始跟踪采样，必须将配置下载到运行系统中。跟踪编辑器窗口中将显示跟踪变量的曲线图形，用户可以将它们保存到一个外部文件中，以后可以重新加载到编辑器中。

在编辑器窗口和“跟踪”菜单下可以使用打开配置对话框的命令，以及所有修改当前显示的跟踪曲线的命令。

使用特殊的视图元素“跟踪”，跟踪的输出可以被集成到视图中。

跟踪编辑器概述

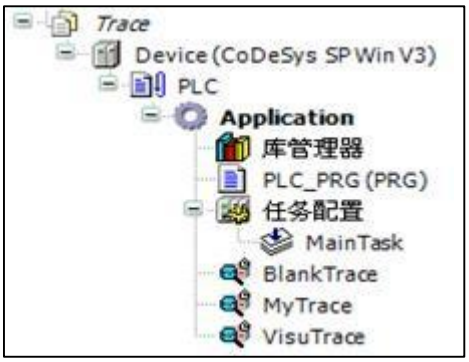
“跟踪编辑器”插件提供的跟踪编辑器，是用来配置并显示“跟踪曲线”的。“跟踪曲线”，是指变量的一组采样值，也就是某一段时间内从 PLC 读上来的连续的值。另外可以设置一个触发器。如果已经设置了触发器，那就只保存和显示触发值。采样值存放在 CoDeSys 缓冲区，可以在跟踪编辑器窗口里的坐标系中以图表形式显示出来。

用户可以定义一个或多个变量显示在跟踪窗口的变量坐标系中。

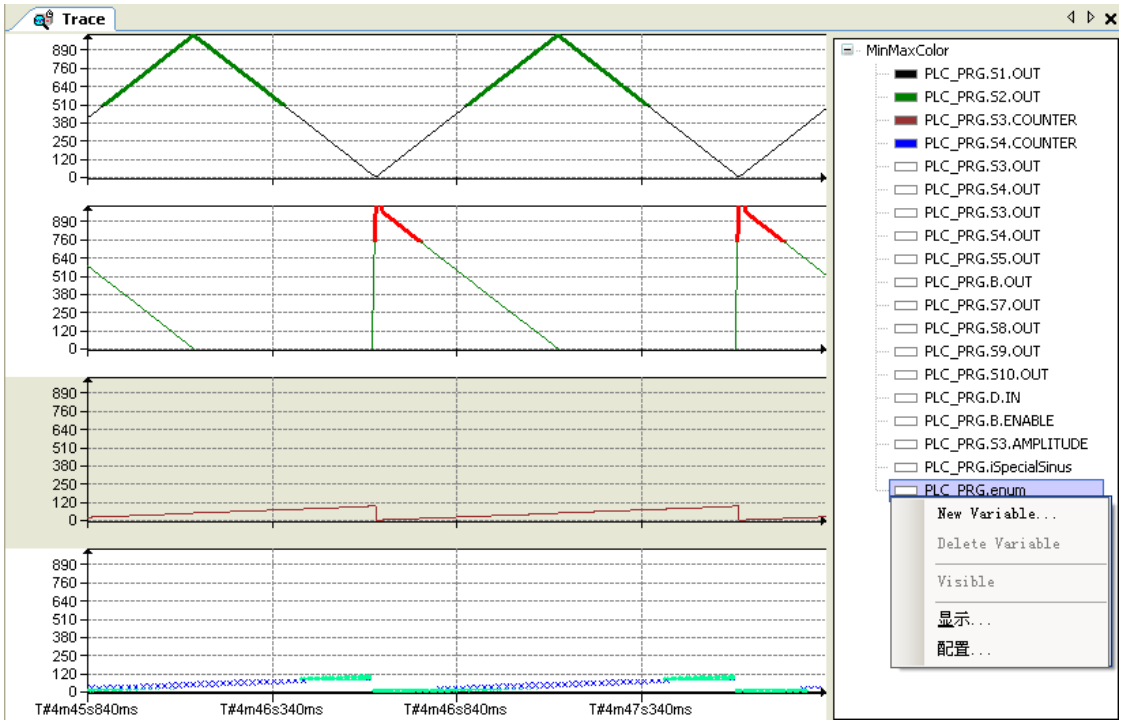
如果运行时系统有附加组件’跟踪管理器’，那么就可以在此进行跟踪配置管理和数据存储。也可以通过跟踪管理器访问设备跟踪如当前值的记录等。如果没有可用的跟踪管理器，用 IEC-代码下载功能。这种情况下菜单命令在线列表...和上传跟踪将不可用。

同时触发的变量可以被记录在一个跟踪对象中。跟踪对象里可以显示变量坐标系，可能是在多通道模式下。有不同触发的变量的跟踪必须记录在它们各自的跟踪对象中。你可以创建任意个数的跟踪对象。

举例：带有跟踪的工程

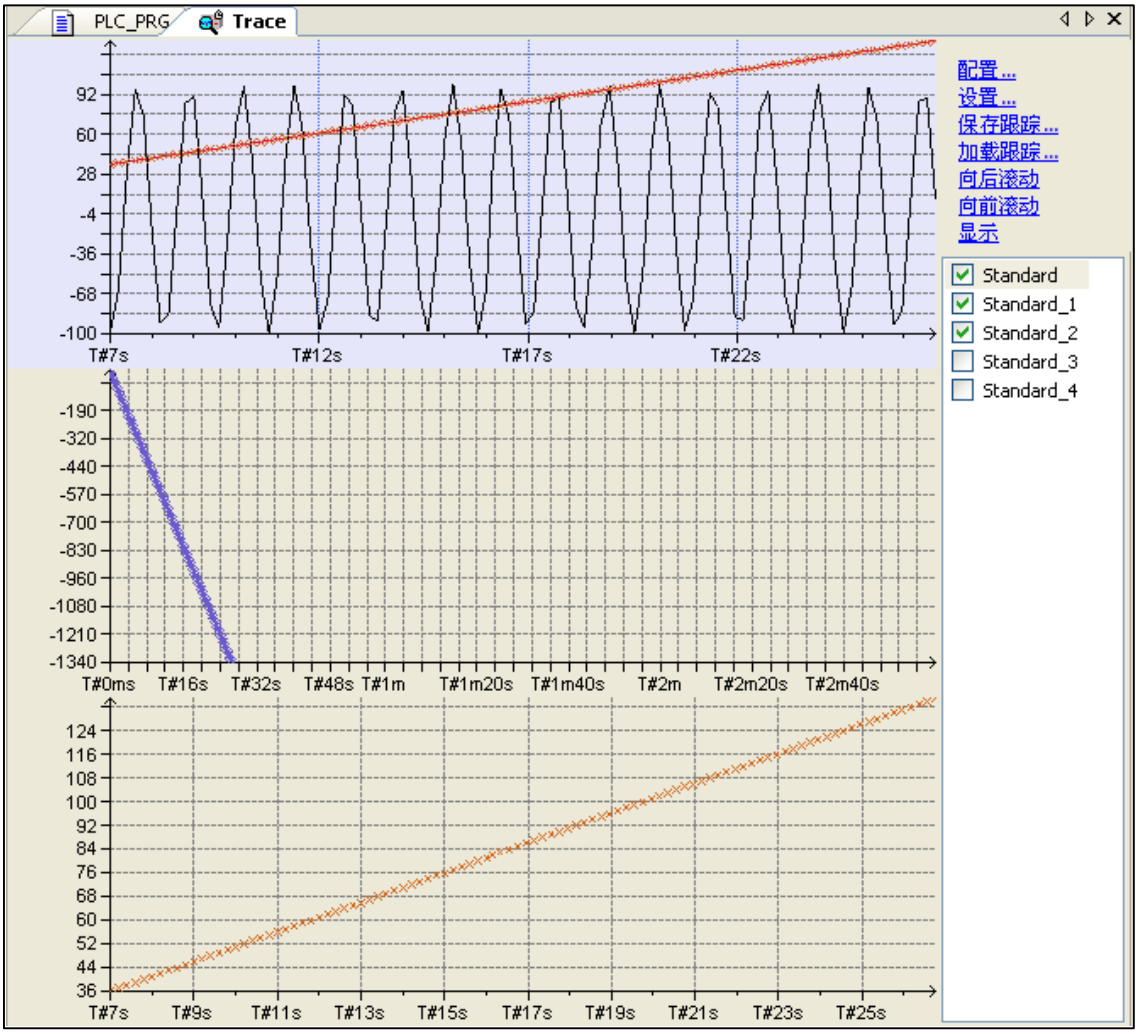


举例：跟踪编辑器窗口，显示在线模式下对四个变量的跟踪



此图为在线模式下跟踪的例子。在对话框右下角的记录树中，选择了三条记录进行显示。在左上角，

是配置、加载/保存跟踪配置和滚动当前显示等命令。



命令与缩放功能

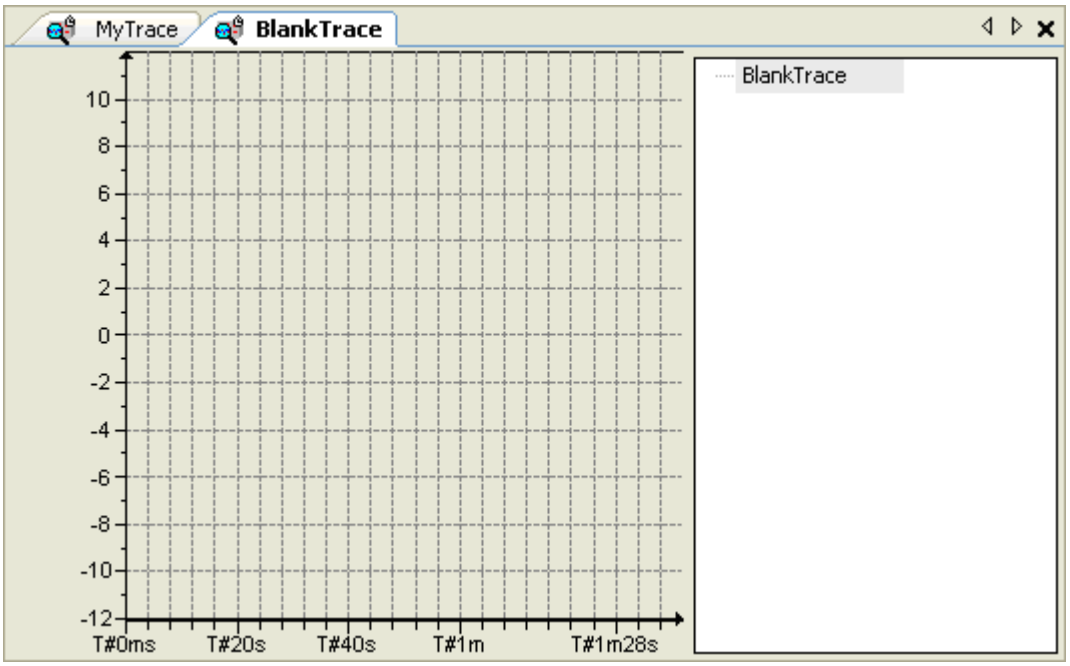
跟踪编辑器分别“工作”在离线或在线模式下时，跟踪菜单和跟踪窗口都有可用的命令。跟踪菜单提供对跟踪图的操作。跟踪窗口图表的右键菜单也有跟踪菜单命令。有触发设置的跟踪配置和变量选择只能在显示跟踪树的跟踪窗口右边的右键菜单中操作。跟踪树中不同的选择对应不同的可用命令。

跟踪配置

在设备视图窗口中，通过“添加对象”命令可以添加一个“跟踪”对象；双击该对象或通过“编辑对象”命令可以打开跟踪编辑器。

在标题为“ <跟踪对象名>”的编辑窗口中，左边为一片空白区域，用来显示跟踪图和在右边区域的配置...中配置的跟踪树。

跟踪编辑器窗口，未配置



跟踪定义了一个或多个跟踪变量。在此普通设置中可以定义相关的跟踪开始，采样，触发条件，缓冲大小，响应任务等。另外每个跟踪变量的“外观”（显示曲线的坐标系的显示变量）也必须设置。

在跟踪窗口右边区域的右键菜单中设置跟踪。

新建变量...用来添加一个变量并设置显示参数（颜色，图形类型等）。

删除变量：删除所选的变量。

可见：设置所选的变量为可见。

外观...对话框用来确定图的外观（没有加载配置时，该命令是灰色的）。

配置...对话框用来定义跟踪（外观，条件，曲线类型和图的外观等）和跟踪变量。

跟踪配置可以保存为*.trace 文件，也可以重新加载到跟踪窗口。

在线模式下的跟踪编辑器

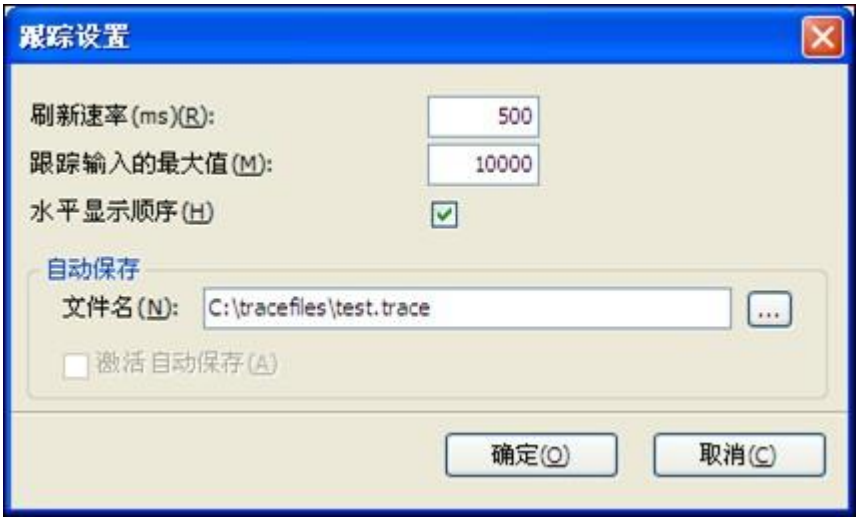
为了激活在线模式下的跟踪，生成的目标代码必须要下载到目标系统中。

跟踪菜单为当前显示的跟踪图表提供了额外的命令。

设置...

在跟踪编辑器对话框中，点击设置...，将打开跟踪设置对话框。这些设置对于当前加载的跟踪配置中的所有“记录”都有效。

跟踪设置对话框



刷新频率（毫秒）：毫秒级的时间间隔，在线模式下，跟踪窗口在此时间内更新一次。缺省值：500毫秒。

跟踪的最大点数：运行系统的环形缓冲区中，能够保存的值的最大数目。缺省值：10000。当缓冲区填满时，从“最旧”的变量值处开始覆盖。

水平外观顺序：若本选项被激活，并且在线模式下要显示不止一条记录，视图窗口会被水平隔开，即这些记录会从上往下有次序地排下去。若本选项未被激活，该窗口会被垂直隔开，一行最多显示三条记录。

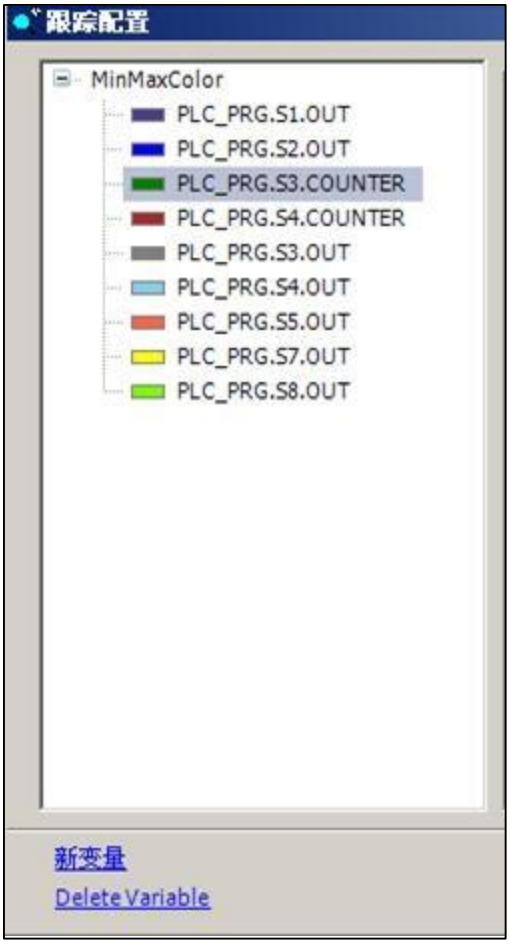
配置…

点击跟踪编辑器窗口右边区域中跟踪树右键菜单的配置...命令，将打开跟踪配置对话框。双击跟踪名或变量名，也可以打开此对话框。用户可以配置跟踪哪些变量，以及跟踪和定义变量时需要设置的参数。

注意：设置对话框中的常规设置，总是对当前加载的跟踪配置有效。

跟踪配置由一条或多条记录的配置（记录设置）和在记录中所跟踪的特定变量的特殊设置（变量设置）组成。当前所配置的记录清单和跟踪变量清单，在跟踪配置对话框的左部以树型显示。

跟踪配置对话框中的跟踪树

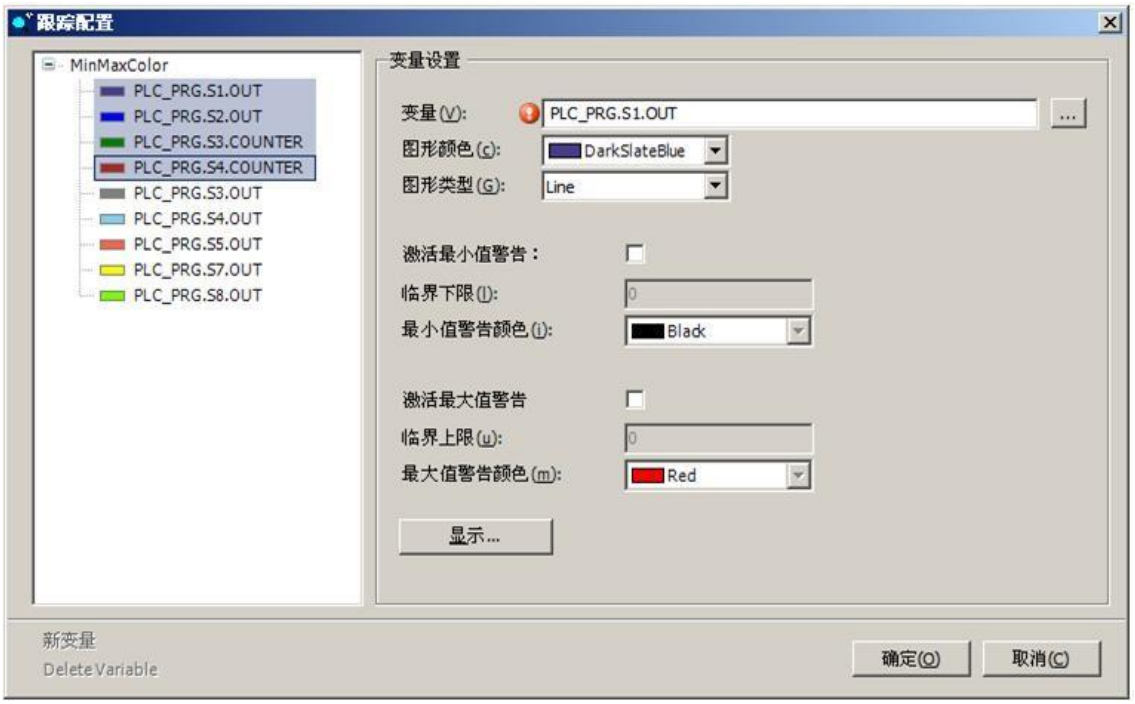


请看第一级已配置跟踪的名字和下面缩进显示的跟踪变量名。选择跟踪树的第一级跟踪记录名称修改跟踪设置。记录设置对话框被打开。

记录树下的命令：新建变量，删除变量，可以添加变量或删除。

要修改变量设置，请选择相关变量名，然后变量设置对话框将打开。也可以用 Shift + 鼠标点击或 Ctrl + 鼠标点击在变量列表选中多项。这样就可以修改所选项的变量设置。

选择多个变量

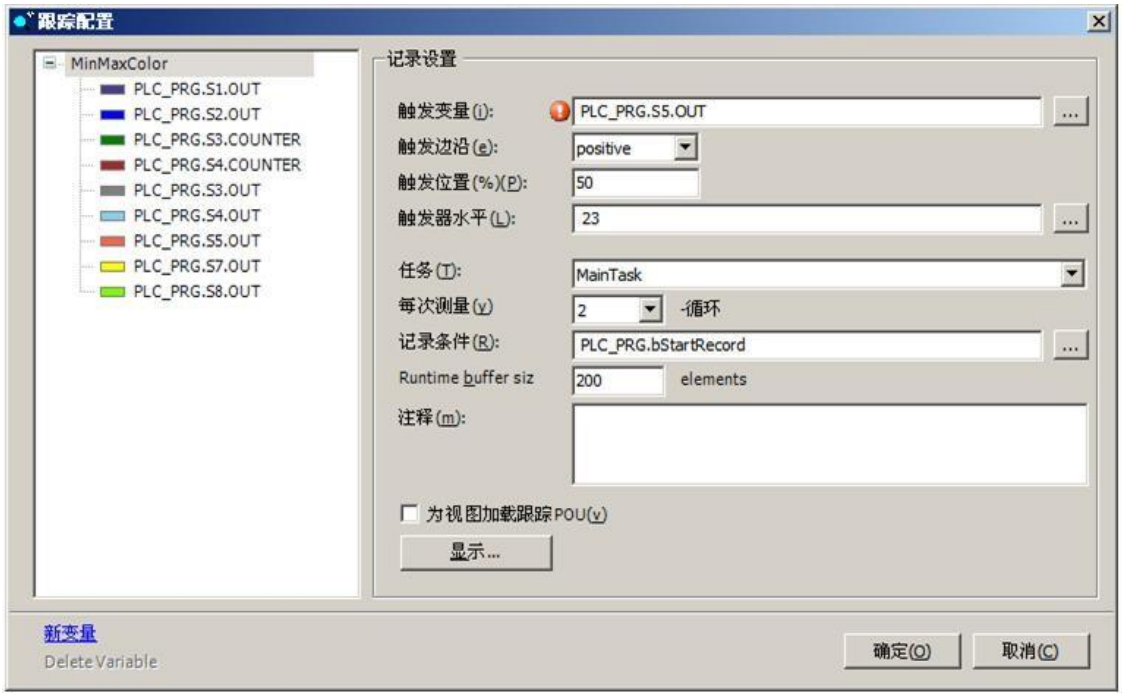


下面描述了跟踪设置和专用跟踪变量：

记录设置：

请看跟踪配置对话框中的记录设置的描述。

跟踪配置，记录设置



触发变量：配置触发变量是可选的，它与其它一些条件共同决定了跟踪的时间范围。下面进行详细解释：

该变量可以是一个布尔变量、一个表达式或一个模拟变量，也可以输入枚举变量或属性变量。。当该变量满足了定义的值——该值根据“触发边沿”（定义见下文）类型来决定，跟踪将在采样一段时间后停止——该采样时间段由“位置”（定义见下文）的百分比来决定。也就是说，一旦触发变量变为真或满足某一特定值，跟踪将继续一段定义好的周期。

点击按钮[...]使用输入助手来选择一个合适的触发变量。

注意，通过“记录条件”（定义见下文）也可以控制跟踪的开始。

触发边沿：

无	无触发（缺省值）
上升沿触发	布尔型触发变量的上升沿，或模拟触发变量增大到为它所定义的“水平”值，触发事件发生
下降沿触发	布尔型触发变量的下降沿，或模拟触发变量减小到为它所定义的“水平”值，触发事件发生
上升下降沿触发	在上升沿触发或下降沿触发中描述的条件达到时（如上所述），触发事件发生

触发位置（%）：触发事件发生后，要记录的跟踪变量的测量值百分比。即定义，触发事件发生前所要记录的跟踪变量测量值的百分比，以及触发事件发生后要记录的百分比。例如：填入 25，则在跟踪曲线中，75%是触发事件发生前的测量值，25%是触发事件发生后的测量值。如果希望一发生触发事件，就开始跟踪，那么填入 100。

触发水平：若使用模拟量作为触发变量，在此处定义该变量为多少时产生触发事件。可直接输入一个数值，或用变量来定义该数值（允许使用枚举型常量）。缺省值：空。

任务：在可用的任务列表中选择，该任务被执行后从中读取出跟踪变量的值。缺省值：任务配置树中的第一个任务。

第 x 个周期处测量：此处定义再次读取变量前我们需要等待的周期数，通过（缓冲区大小*第 x 个周期处测量*任务间隔）计算公式，可以计算出最小的监视时间段。

记录条件：此处可输入一个布尔变量、一个数值或一个布尔表达式。该条件为真，则启动跟踪采样。若此处没有任何输入，则在下装跟踪配置并且应用开始运行后，立即开始跟踪记录。

缓冲区大小：字节数，运行系统中分配给当前记录用以保存跟踪变量值的缓冲区大小。监视跟踪时会显示此缓冲区中的内容。缺省值：100。

注释：在此输入有关当前记录的注释文本。

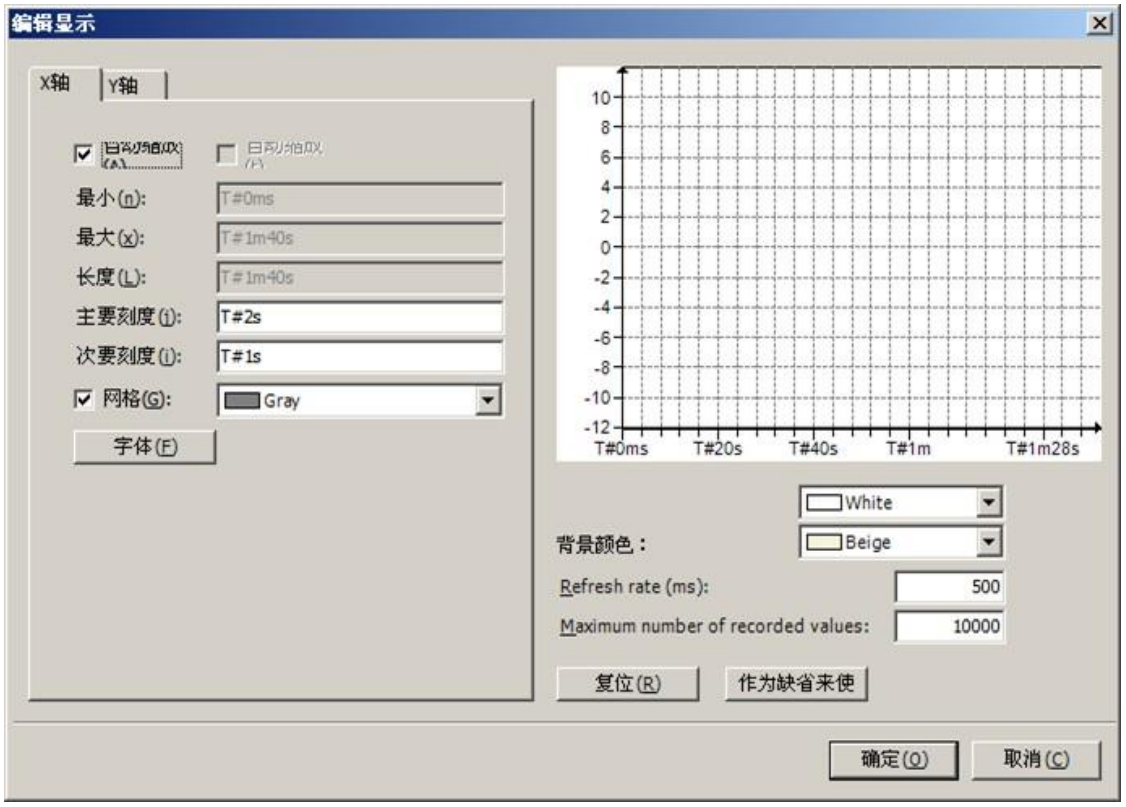
外观：点击此按钮将打开“编辑外观<记录名>”对话框，在该对话框中可以为当前配置的记录设置其跟踪窗口的显示（坐标轴，颜色，滚动动作）。

生成跟踪 POU：勾选其复选框，引发对组件<跟踪名>_<任务名>_VISU 的隐式编译。如果想在视图中集成跟踪的输出，可使用该选项。

注意：如果想同时以两个不同的外观显示同一跟踪变量的跟踪曲线，应将此变量添加到外观不同的跟踪对象中去。

下面的选项用来设置跟踪图的 X 轴，Y 轴，当跟踪图显示在单通道视图时，它们是可见的。

编辑外观对话框



自动缩放：如果此选项被激活，跟踪的时间范围——即跟踪窗口的 X 轴所显示的时间范围会根据运行系统中的当前记录所定义的缓冲区大小，自动缩放。

如果此选项未被激活，可以通过下列设置来定义跟踪窗口中所显示的时间范围：

如果选项“自动缩放”未被激活：

最小值，最大值：此处输入的时间（标准时间格式，例如“T#200s”）精确地定义了要显示的跟踪的时间范围。这意味着，X 轴从最小值时间点开始，在最大值时间点结束。如果想查看其他的时间范围，需要手动滚动界面。

如果选项“自动滚动”被激活（只有当“自动缩放”选项未被激活时才有效）：

长度：在此处输入的时间范围（标准时间格式，例如“T#2m”）会被显示。坐标轴以显示变量最新值的方式自动滚动。

主要刻度：坐标轴上用长记号指示的值间隔。如果网格被激活了，它们延长为虚线。这个值要匹配坐标轴上的值类型，并且要符合 IEC 句法；举一个 X 轴时间值的例子：例如“T#2s”代表 2 秒。

次要刻度：坐标轴上用短记号指示的值间隔（即使网格被激活了也不会延长）。有效值见“主要刻度”。

反向：如果激活该选项，坐标轴的方向将会翻转。例如，对于 X 轴来说，坐标轴的左右端会被对调。

描述：如果激活该选项，在编辑框中输入的文本将在 Y 轴的上端显示。X 轴上无任何显示。

网格：如果激活该选项，坐标系中将显示网格。主要刻度将延长为虚线。网格线的颜色可以从颜色列表中选择。

画笔箭头：如果激活该选项，坐标轴的端点处会画上一个箭头，它指明坐标轴的方向。

填充箭头：如果激活该选项，画笔箭头会被填充为实心箭头。

位置：滑块的位置定义了坐标轴在坐标系中的位置：底部、中间、顶部或是左边、中间、右边。

字体：点击此按钮将打开标准对话框来定义跟踪显示的字体。

背景色：从给出的颜色列表中选择坐标系的背景色，只要没有在跟踪窗口中选择相同的颜色，它将一直有效。

被选择的后背景色：从给出的颜色列表中选择坐标系的背景色，只有跟踪窗口中选择了相同的颜色后它才有效。

刷新频率（毫秒）：毫秒级的时间间隔，在线模式下，跟踪窗口在此时间内更新一次。缺省值：500毫秒。

记录的最大点数：运行系统的循环队列缓冲区中，能够保存的值的最大数目。缺省值：10000。当缓冲区填满时，从“最旧”的变量值处开始覆盖。

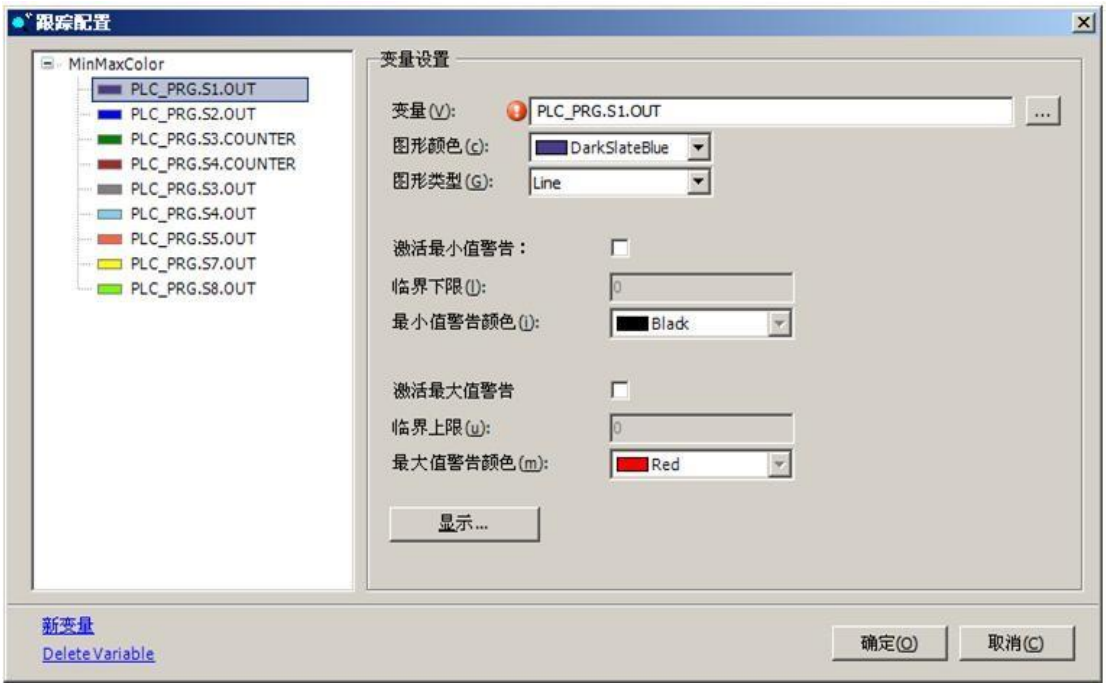
复位：将外观复位为缺省值。

缺省使用：将当前外观设置为缺省。配置新跟踪或变量时将使用它。

变量设置

请看跟踪配置对话框对象设置的描述。

跟踪配置，变量设置



在每条跟踪记录中必须定义要“跟踪”并在记录坐标系中要显示的变量。若要在一条记录中添加一个变量，在跟踪窗口的左侧的记录树中选中该记录条目，点击新变量按钮（窗口的底部），此时窗口的右侧会显示变量设置对话框，同时，记录树中会添加一个空的变量条目，定义此变量：

变量：在此处输入所跟踪的变量的名称。除了数据变量，还可以跟踪指针，属性，引用或数组元素的内容。使用输入助手（...）以保证选择一个有效的变量。检查并编辑当前配置（详见下文）。

该变量会立即加入到记录树中。以后想要修改变量的设置，只需在记录树中选中该变量，将再次打开变量设置对话框。

图形颜色：从给出的颜色列表中选择变量的跟踪曲线的显示颜色。

图形类型：在此处选择数据点在变量的跟踪曲线中应如何显示。对于大量数据推荐使用“线条”类型。

无	不显示图形
线条	采样点连成一条线.
交叉	采样点用十字交叉点表示
齿形	显示记录值所连成的齿形图形。
梯形	两个采样点值用梯形来连接：垂直线延长到下一个点的 y 值，水平线延长到下一个点的 x 值。
点	用小点表示点。

激活最小值警告：若激活该选项，一旦变量超过了“下限”中定义的值，跟踪图表将显示“最小值警告色”中定义的颜色。

下限：详见上文，激活最小值警告

最小值警告色：详见上文，激活最小值警告。

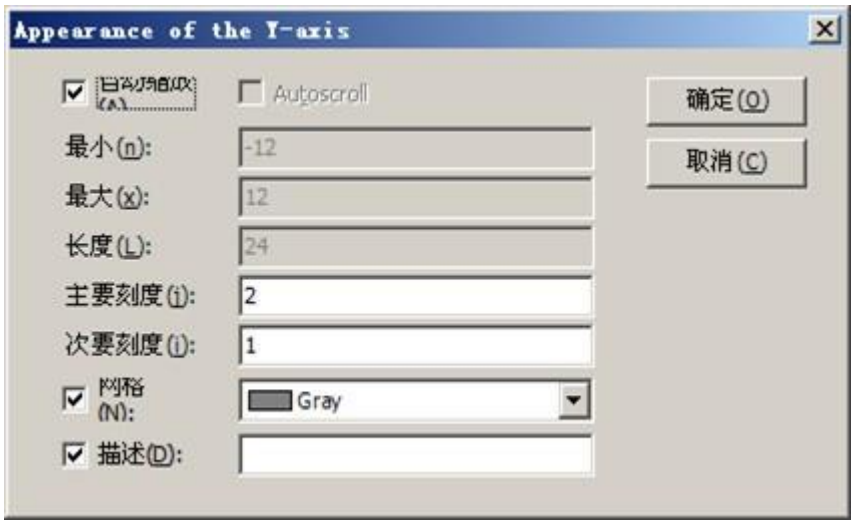
激活最大值警告：若激活该选项，一旦变量超过了“上限”中定义的值，跟踪图表将显示“最大值警告色”中定义的颜色。

上限：详见上文，激活最大值警告

最大值警告色：详见上文，激活最大值警告

外观：该按钮打开 Y 轴外观对话框，你可以设置跟踪窗口当前配置的 Y 轴（颜色和滚动行为）每个变量各自的显示风格。当跟踪图显示在多通道视图的时候可以使用此设置。

编辑 Y 轴外观对话框



自动滚动：如果此选项被激活，自动显示从最小 Y 值到最大 Y 值的跟踪图。

如果此选项未被激活，可以通过下列设置来定义跟踪窗口中所显示的范围：

如果选项“自动滚动”未被激活：

最小值，最大值：此处输入的 Y 值，精确地定义了要显示的跟踪范围。 这意味着，范围从 Y 轴最小值开始，在最大值结束。如果想查看其它的范围， 需要手动滚动界面。

如果选项“自动滚动”被激活（只有当“自动缩放”选项未被激活时才有效）：

长度：在此处输入的值用来定义显示范围的长度，坐标轴以显示 Y 轴最大值的方式自动滚动。

主要刻度：坐标轴上用长记号指示的值间隔。如果网格被激活了，它们延长为虚线。这个值要匹配

坐标轴上的值类型，并且要符合 IEC 句法；举一个 X 轴时间值的例子：例如“T#2s”代表 2 秒。

次要刻度：坐标轴上用短记号指示的值间隔（即使网格被激活了也不会延长）。有效值见“主要刻度”。

网格：如果激活该选项，坐标系中将显示网格。主要刻度将延长为虚线。网格线的颜色可以从颜色列表中选择。

描述：如果激活该选项，在编辑框中输入的文本将在 X 轴的右端显示。

字体：点击此按钮将打开标准对话框来定义跟踪显示的字体。

保存跟踪...

此命令在跟踪编辑器窗口中，用来保存当前的跟踪配置到文件中。打开保存跟踪对话框（保存文件的标准对话框），可以定义文件名和存放路径。Codesys 跟踪文件的默认扩展名为*.trace。

使用加加载跟踪…命令，可以重新加载跟踪配置。

加载跟踪…

此命令在跟踪编辑器窗口中，用来加载已保存在文件中的跟踪配置到编辑器中。该文件由保存跟踪…命令创建。

此命令将打开加载跟踪对话框，该对话框是一个标准的文件浏览对话框。缺省文件类型是“Codesys跟踪文件（*.trace）”。被选中的跟踪文件会在编辑器窗口中加载并显示。

向后滚动

此命令在跟踪编辑器窗口中，用来移动跟踪窗口中的当前可视区域。图表小步地沿着 X 轴往右移动，以查看旧值。

在按住 Alt 键的同时按下向左箭头，同样可以执行此功能。

注意也可通过滚动鼠标来完成此功能。

向前滚动

此命令在跟踪编辑器窗口中，用来沿着 X 轴小步地移动跟踪窗口中的图表。所有当前所显示的记录都会滚向右侧。

在按住 Alt 键的同时按下向右箭头，同样可以执行此功能。

注意也可通过滚动鼠标来完成此功能。

外观. . .

此命令在跟踪编辑器窗口中，用来打开记录列表中选中的记录的外观编辑对话框。该对话框是可用的，当选中跟踪树中最顶层的跟踪名时，跟踪编辑器的外观编辑对话框被打开。如果跟踪树中的变量被选中，打开 Y-轴外观对话框。

在跟踪的配置对话框时，该对话框也是可用的。

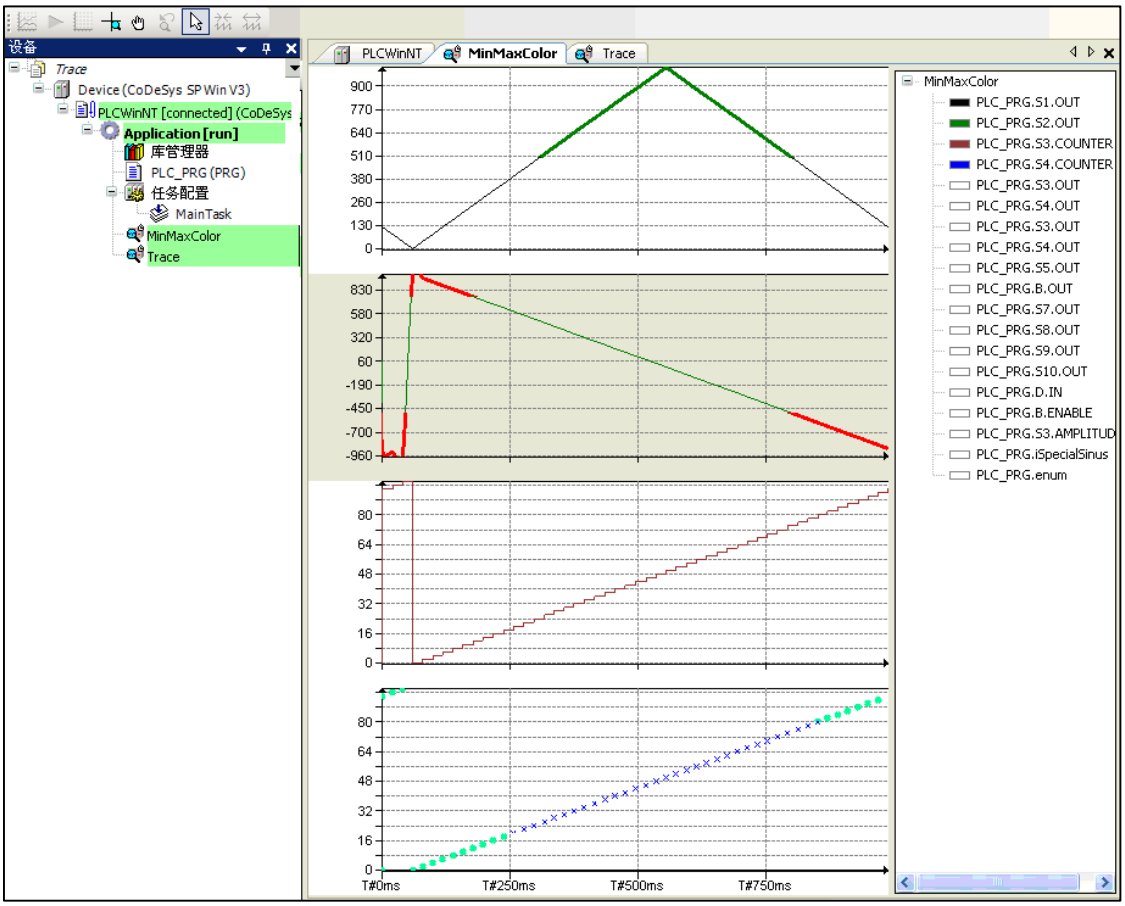
在两个对话框中定义记录要如何在跟踪窗口中显示，即用户可以定义坐标系的样式（坐标轴配置，颜色，网格等），该记录的跟踪变量值图表将在此坐标系中显示。

跟踪的外观设置（外观编辑对话框）缺省是在单通道模式下使用，但是变量的外观设置（Y-轴外观对话框）在多通道模式中可见。

在线模式跟踪编辑器

“跟踪”与应用一样，都运行在当前设备上。用户可以通过查看设备树中的跟踪对象是否为绿色来判别跟踪采样是否被激活。

设备树中激活的跟踪



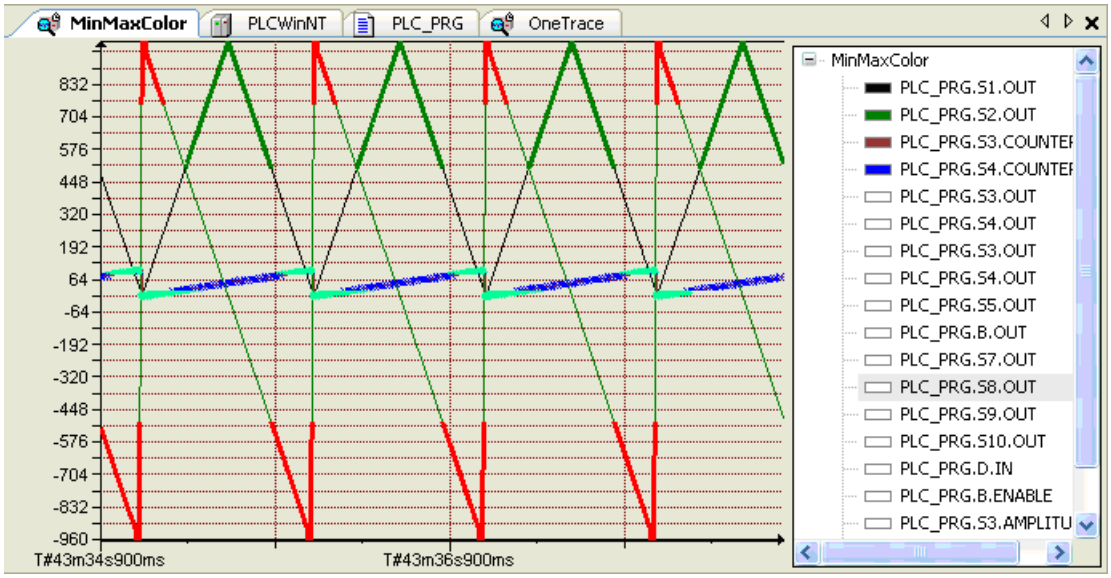
下载跟踪

对于第一次使用或之后对跟踪配置的任何修改，必须明确的下载跟踪“应用”来激活对当前运行应用的跟踪。修改应用之后将自动结束跟踪，必须重新下载它。在不做任何改动登录或退出应用时，跟踪将继续，不重新下载。

跟踪图的配置

变量的配置...对话框在在线模式下也可用，并可以在线修改。可以双击编辑器右边区域的配置树中的变量名，或打开右键菜单，选择配置命令。

在线模式的跟踪窗口，单通道模式的例子



控制跟踪显示

跟踪菜单命令是缺省有效的，可以控制当前显示的跟踪图（启动/停止跟踪，复位触发）并修改跟踪曲线当前显示的段（鼠标滚动，鼠标缩放，等）。

跟踪图的值范围取决于跟踪设置。但是可以通过跟踪菜单的滚动和缩放功能手动调整。也可以通过快捷键。对于沿 Y 轴的垂直滚动，可以在跟踪的当前视图中用向上箭头和向下箭头来实现。对于沿 X 轴的水平滚动，可以在 ALT 键的同时使用向左箭头或向右箭头。

如果多通道选项启用，每个图都有自己的图表，选中的图可以通过上述快捷键进行滚动。要使多张图同时滚动，可以按住 Ctrl 键同时使用箭头键移动 Y 轴。所有的跟踪快捷键都列在跟踪快捷键中。

所有的跟踪快捷键都列在跟踪快捷键中。

注意，也可用滚动鼠标来实现缩放。

当鼠标光标放在跟踪窗口上时，X 轴的值将显示在状态行中。

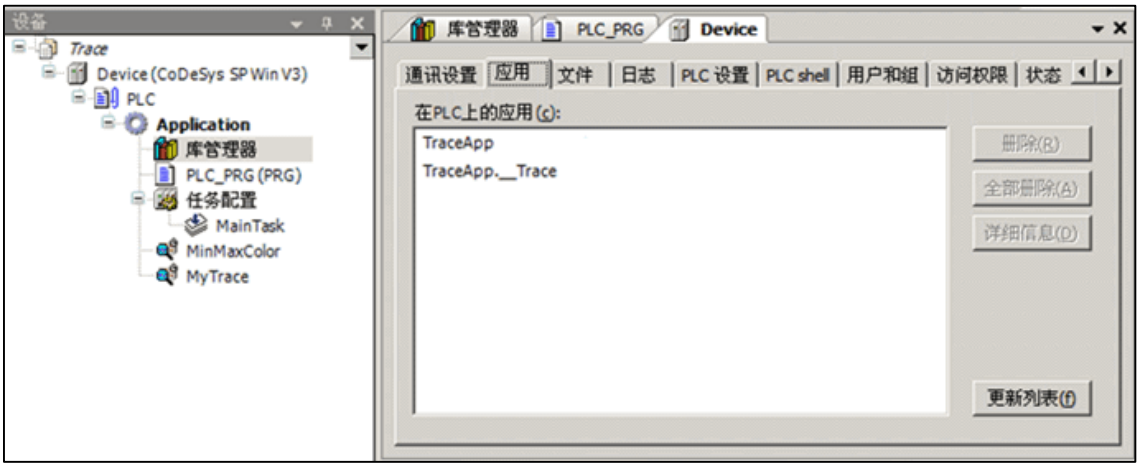
跟踪子应用

如果运行时系统有附加组件'跟踪管理器'，那么跟踪管理器将下载到其子应用中。然后列在 PLC 设备编辑器的应用对话框中，并可以扫描和删除子应用。

跟踪子应用的语法

<应用名>.__<跟踪名>

设备编辑器，应用选项卡：跟踪应用



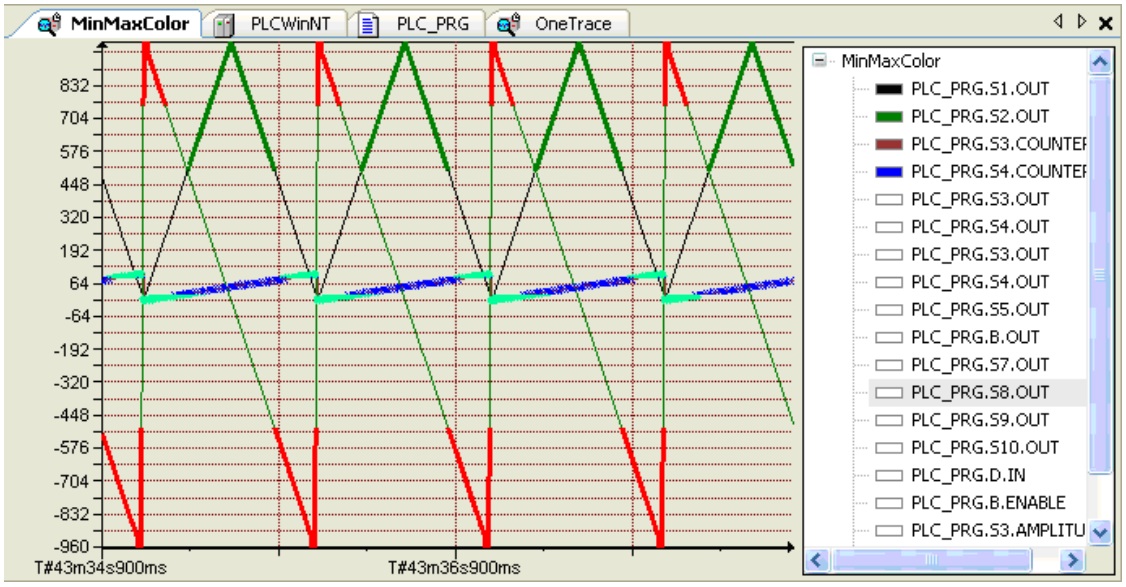
跟踪显示

在线模式下，跟踪编辑器窗口根据记录和变量配置以及常规设置，来显示跟踪的采样值。

跟踪及其所有可视变量都以图的形式显示在坐标系中。坐标系按照记录设置中定义的样式来显示。对于特定变量来说，它的颜色和类型按照变量设置来显示。X 轴表示记录的时间，Y 轴表示相应的变量值。

在跟踪窗口右侧的跟踪树中，可以设置要显示值的变量为可见，激活他们在在图标中的图。另外，可以在线修改离线设置的配置。

在线模式下的跟踪窗口，举例



每条记录所显示的值范围由该记录的设置所决定。不过，用户也可以通过手动的向前滚动、向后滚动以及缩放功能来控制。这些命令可从跟踪菜单中获得，也可用快捷键完成操作。对于垂直滚动，可以在跟踪图表当前视图中用向上箭头和向下箭头来实现。对于沿 X 轴的水平滚动，可以在 ALT 键的同时使用向左箭头或向右箭头。

如果多通道选项启用，每个图都有自己的图表，选中的图可以通过上述快捷键进行滚动。要使多张图同时滚动，可以按住 Ctrl 键同时使用箭头键移动 Y 轴。所有的跟踪快捷键都列在跟踪快捷键中。

注意，也可用滚动鼠标来实现缩放。

当鼠标光标放在跟踪窗口上时，X 轴的值将显示在状态行中。

2.16 视图编辑器

视图编辑器用于编辑视图对象，它由“视图编辑器”插件提供。请点击这里，可以查看 CoDeSys V3.x 中使用视图的前提条件和对视图的概述。

通过“添加对象”命令，可以将视图对象添加到 POU 或者“设备”窗口中。如果编辑视图对象，则会打开视图编辑器。如果激活界面编辑器，编辑器窗口将显示为两个部分，上部为声明编辑器，下部为实际的视图编辑器。

编辑器窗口的标题为 “<视图名>[<对象路径>]”。

还有更多的工具和编辑器可以与视图编辑器联合使用，从而创建视图：

视图工具箱，包含所有当前可用的视图元素。

视图元素属性编辑器，包含当前在编辑窗口中选定的视图元素的属性。

如果下列编辑器中的至少一个被各自命令激活（默认在视图菜单中），视图编辑器将分为两部分，各个标签在上半部：

通过界面编辑器可以定义占位符，将视图插入到其它视图中。

热键配置编辑器可配置单键与组合键对应的动作。但是请注意设用该视图的设备必须支持各个键。

元素表编辑器，提供了所有当前视图元素的总览，并允许选择、删除和从后到前地改变元素位置，反过来也一样。

请注意：在编程系统中使用可视化工作，选项对话框, 类视图的当前设置是有效的。例如，网格可被应用到编辑器窗口中。

注意：在设备相关的视图中，可能会有关于配置可能的依赖设备的限制（如：有效元素列表、字体、颜色、图像格式、视图中视图元素的最大数量、设备下视图的最大数量）。

当元素被插入到视图编辑器中时，可以将其选中。通过拖拽鼠标的动作修改位置和尺寸 。通过视图编辑器所提供的相应命令对元素进行对齐和排序。

更多的配置 选项取决于元素的类型。在视图元素属性窗口中可显示与编辑所有的元素属性。

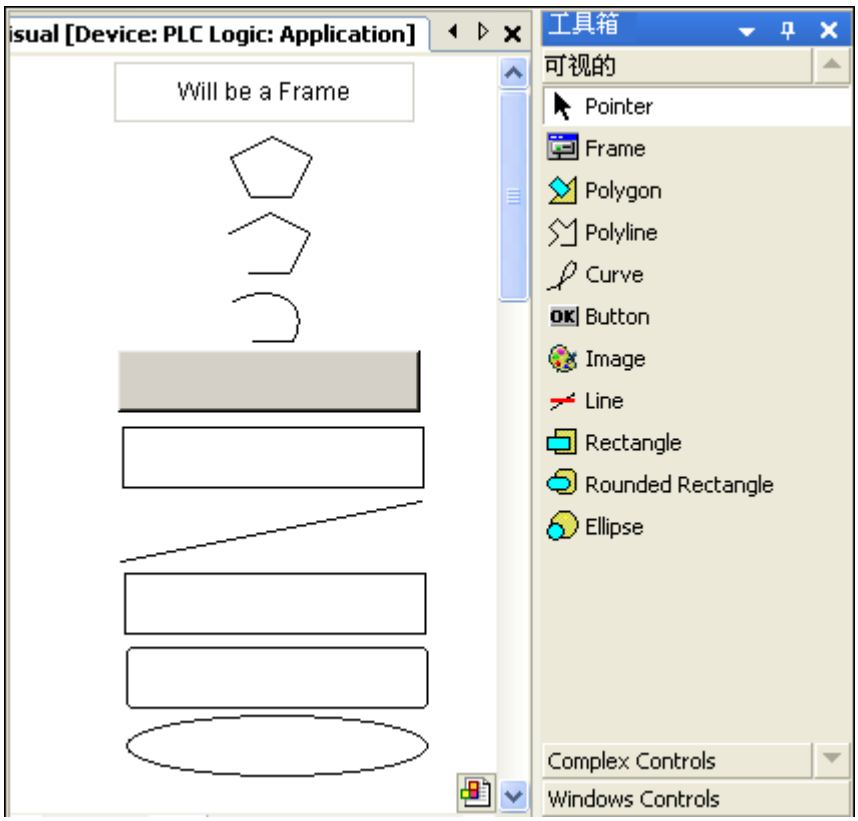
插入视图元素，工具箱

与视图编辑器结合使用的工具箱提供了所有当前可用的视图元素。这些视图元素由库提供，哪些元素有效取决于视图元素库的配置，以及与设备描述相关的设备相关视图。

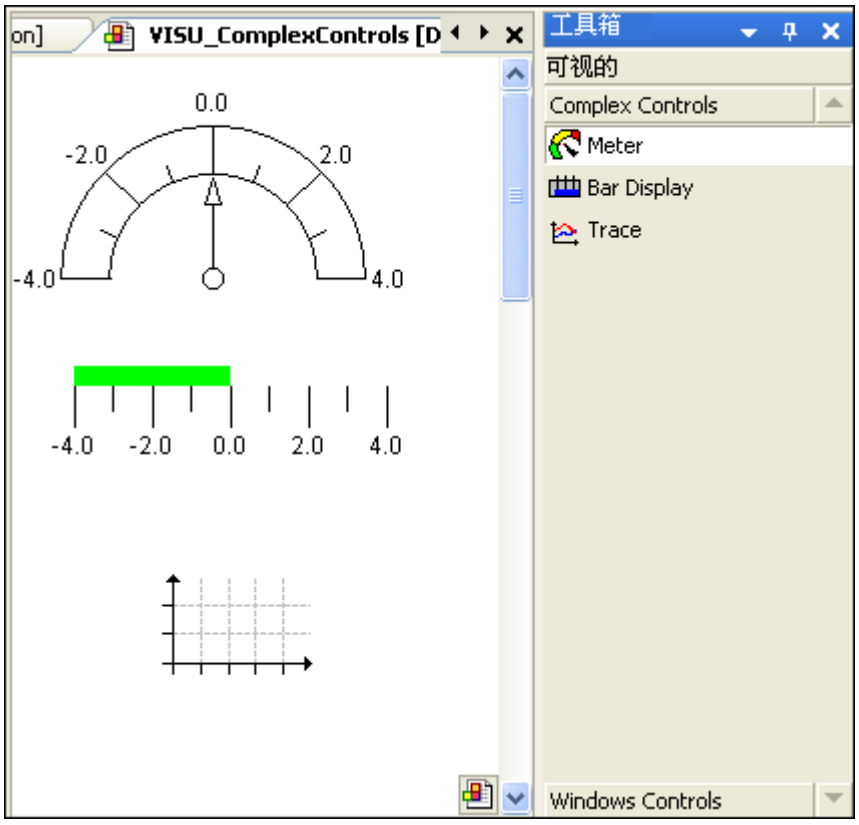
在缺省情况下，工具箱 窗口可通过“视图（View）”菜单打开。所列出的视图元素显示名称和图标，通过鼠标拖放，视图元素可以被插入到当前打开的视图编辑器窗口。

工具箱窗口可通过“视图（View）”菜单打开。它包括三个类别：可视化、复杂控制和窗口控制，其中列出的视图元素显示名称和图标，通过鼠标拖放，元素可以被插入到当前打开的视图编辑器窗口。

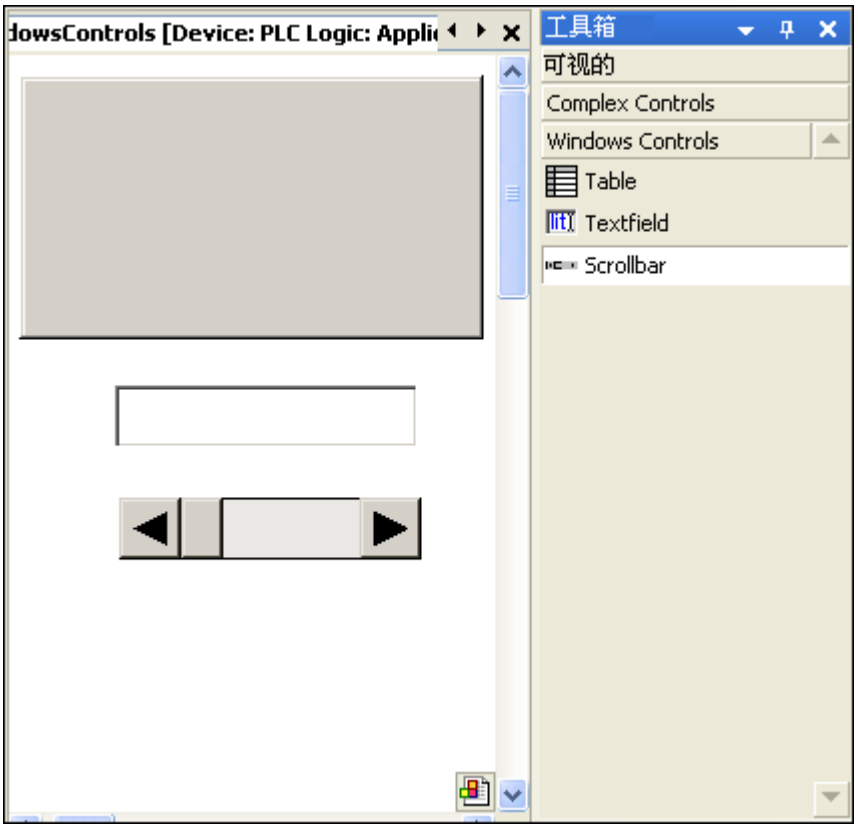
举例：工具箱“可视化”类和被插入的元素



举例：工具箱“复杂控制”类和被插入的元素



举例：工具箱“窗口控制”类和被插入的元素



选中元素拖动鼠标时，指针处的“+”号表示将元素放入编辑器窗口。松开鼠标按键，该元素就会显示在编辑器中。

另一种插入元素的方法是采用添加视图元素命令。在缺省情况下，此命令在“视图”菜单中。选择该命令会打开一个子菜单，提供了与工具箱中相同的可用元素。必要时可用使用自定义对话框来对“视图命令”相关的菜单和命令进行配置。

使用视图菜单中的框架选择命令，可以在框架元素中插入当前视图的其他元素。

插入元素后，可以直接在视图编辑器里选定元素，并修改其位置、尺寸、顺序和对齐，或者在元素属性窗口中配置更多的参数。

注意，使用视图元素所需的标准库是：VisuElems.library.

选择元素

对于每个选定的元素，可以通过拖拉其轮廓方框上的点来修改该元素的位置或尺寸。

在元素上移动鼠标可以选择视图元素 。当光标符号变成手形符号时，单击鼠标，即可选中该视图元素。

如果希望一次选择多个元素，可以按住<SHIFT>键然后用鼠标逐个选择，或者用鼠标左键拉出包围多个元素的矩形框。还可以参考“视图命令”中的“全选”和“不选”。

相对元素可以在元素列表中选中。也可以选中一个组中单个的子元素。

键盘的用法：

选中一个元素后，通过< Tab >键的插入命令（z-命令）移动选中下一个元素，通过<Shift> + <Tab>移动选中前一个元素。

选中一个元素后，通过使用<空格>键，打开文本输入区。输入一个字符串后，通过<Enter>键就可

以关闭编辑区域，该字符串将被应用到元素（元素属性“文本”）中。

当前选定元素的属性将显示在视图元素属性窗口,并且可以在此进行编辑。如果当前选择了多个元素，那么属性的修改将应用到所有元素中。

位置、尺寸、对齐和顺序

修改视图元素的位置：

位置

既可以选择元素并按住鼠标将其拖放到目标位置，也可以用方向键代替鼠标来移动该元素。

另一种编辑元素“位置”的方法是在视图元素属性窗口中修改。

尺寸

选择元素，并点击拖动元素轮廓上的小方点，就可以修改该元素的尺寸。拖动时，会有垂直、水平或对角线方向的双向箭头来指示尺寸变化的方向。

另一种方法是在视图元素属性窗口中编辑“位置”中的尺寸值。

顺序

可以使用视图/顺序子菜单中的命令，来定义元素在视图 Z 轴上的位置（即位于前景、背景或二者之间）。

对齐

使用视图/对齐子菜单中的命令，可以对齐两个或多个元素。

在缺省情况下，“视图”菜单提供全部视图命令。必要时可打开自定义对话框对视图命令相关的菜单和命令进行配置。

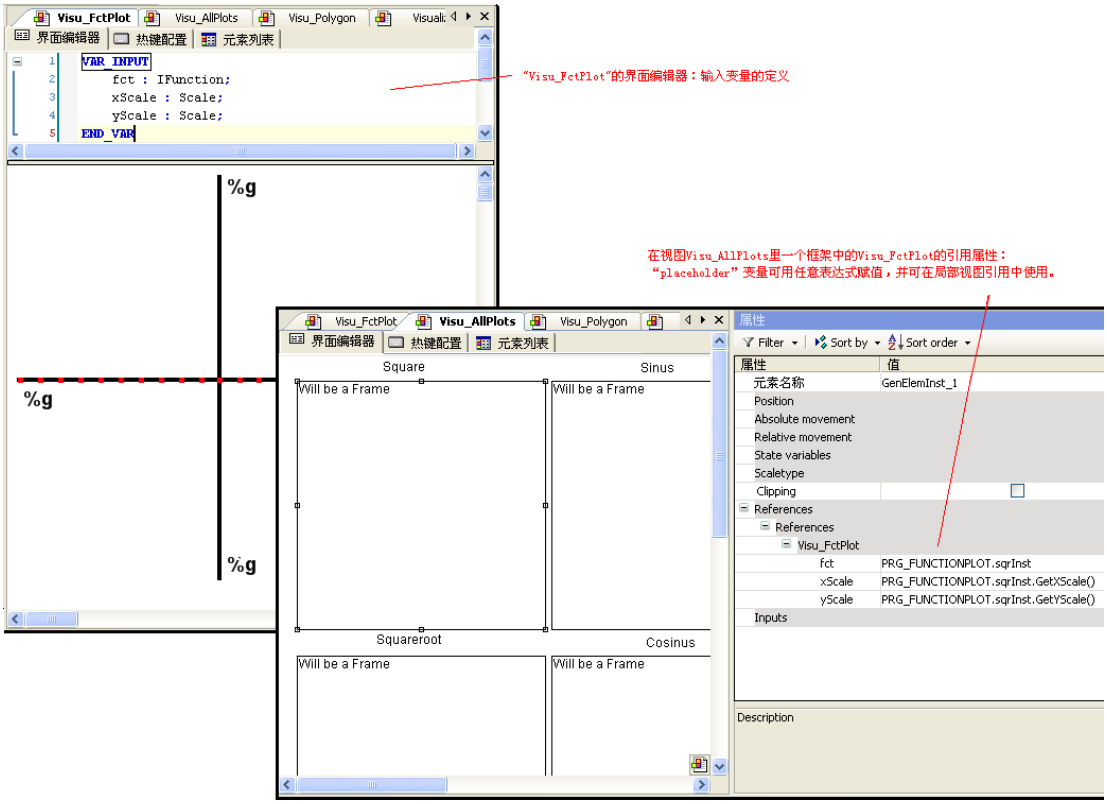
元素成组

多个元素可被选择和成组。这方面内容请参见视图命令“成组”和“取消组”的描述。一个组的子元素仅可通过元素列表被选择。

视图界面编辑器

视图编辑器 的这部分可通过命令界面编辑器 （默认在“视图”菜单中）被打开或关闭。它将做为独立标签在视图编辑器的上半部分和其他编辑器一起被提供，就像热键配置编辑器和元素列表编辑器一样。

视图界面编辑器，示例



该编辑器是用来定义视图中"定位符" 变量的，这个视图被别的视图引用到一个框架中。

更多引用和框架的信息见框架，引用，定位符...。

先决条件以及 CoDeSys V3.x 中的视图概览，见视图。

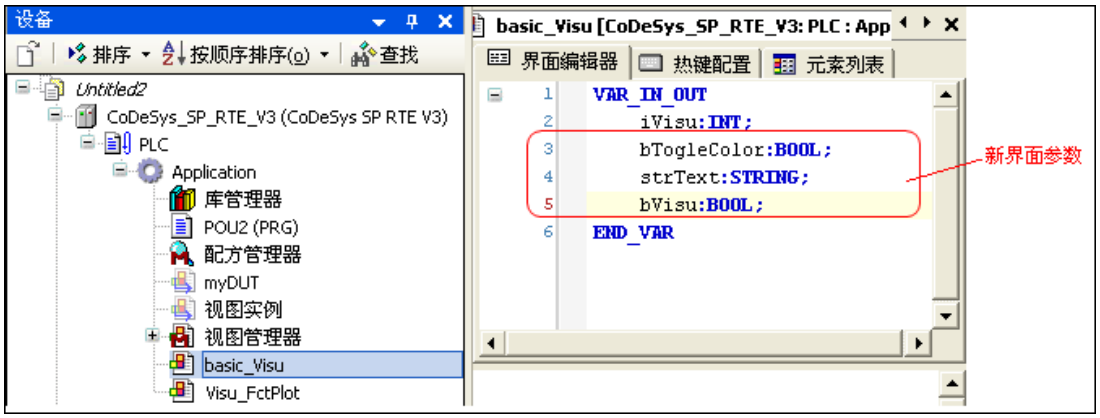
因为视图是被当做功能块处理的，界面编辑器也将被做为普通声明编辑器添加到视图编辑器窗口的上半部分。这里您可以定义输入变量(VAR_INPUT)，当以后在另一个视图的框架元素中包括视图对象时，它可被局部使用的表达式替换。这种替换是在框架元素对话框的视图元素属性中完成的。被包括（实例化）的视图和它们的输入变量在这里列出并可被任何表达式赋值，然后这些表达式又可用于某个实例的配置。

如果一个视图（它被另一个界面的框架元素所引用）的界面被修改，更新框架参数对话框将在您试图保存、编译工程或打开相关对象的时候出现。在这个对话框里您必须（重新）配置界面参数：

怎样使用该对话框，见下面的例子：

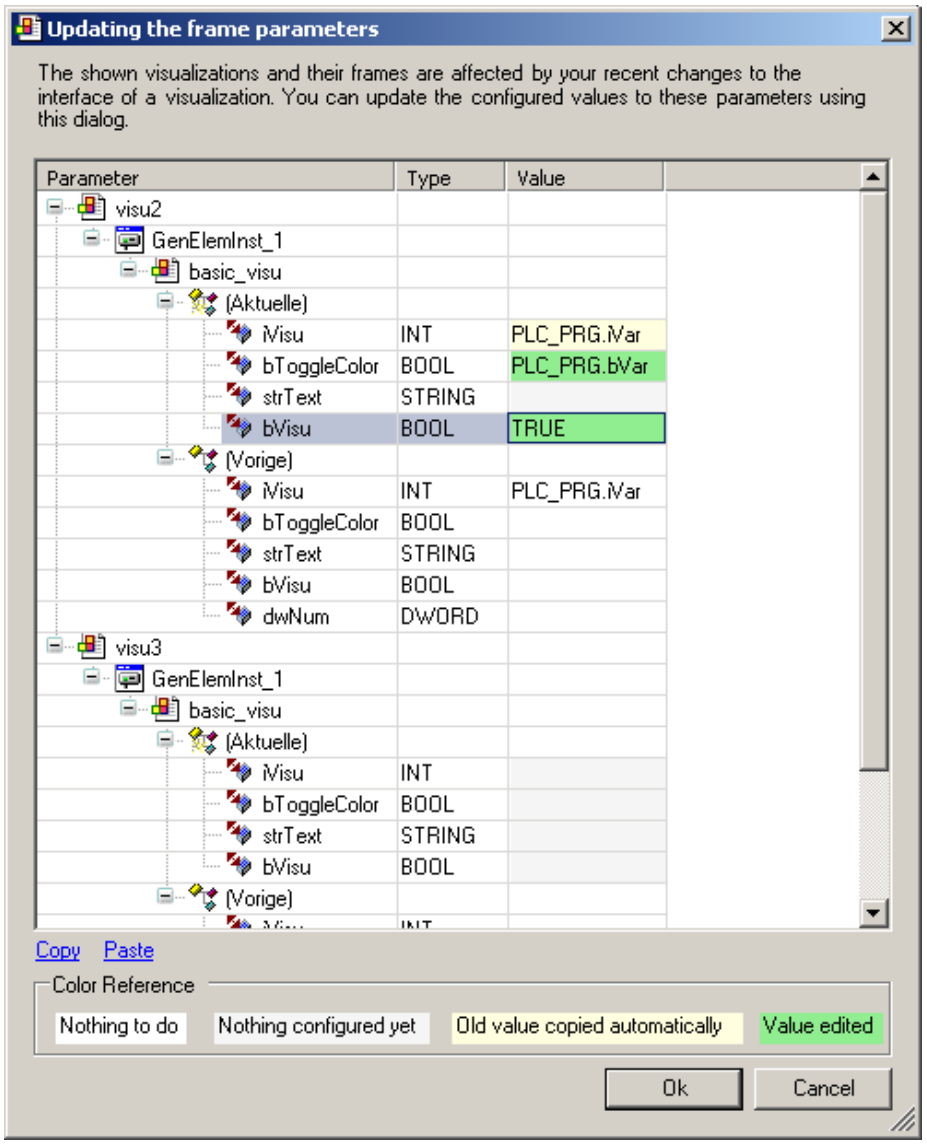
basic_visu 视图的界面最初只定义了 iVisu 参数。basic_visu 分别被插入到 visu2 和 visu3 框架中，在 visu2 中参数 iVisu 被赋给工程变量 PLC_PRG.ivar。工程初始化时使用了这个配置。然后 basic_visu 的界面被参数 bToggleColor, strText 和 bVisu 扩展：

在界面编辑器中修改



现在，如果您想编译或存储工程，那么首先会出现新参数配置对话框：

窗口参数更新对话框



对所有引用已修改的 basic_visu 的视图，即 visu2 和 visu3，每个当前的和以前的界面参数的值都被显示出来。当前值可在这里被（重新）配置。表中“值”域的阴影表示还需要输入或者已经完成（见对话框底部的文字说明）：

米黄色阴影：该参数自动从以前的配置中复制（例如，给 visu2 的 PLC_PRG.iVar）。

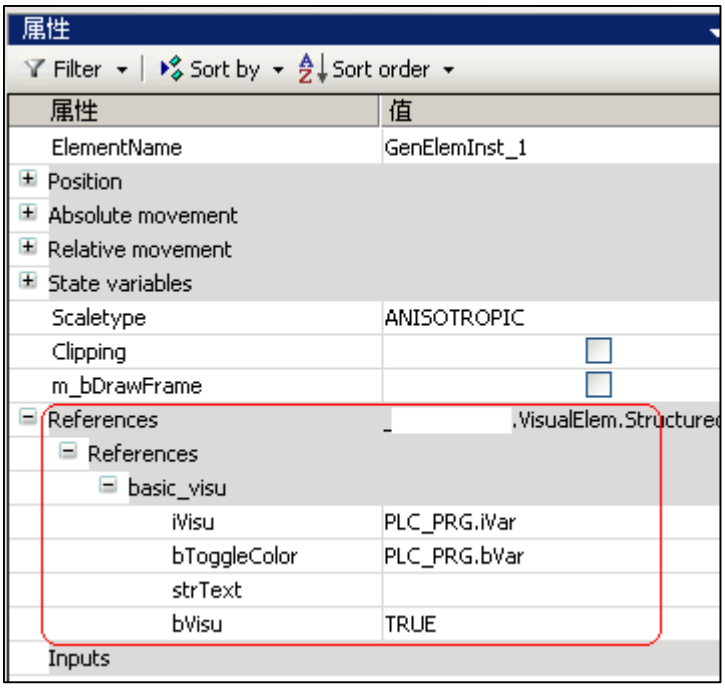
灰色阴影：还没指定任何值给这个新参数（例如，给 visu2 和 visu3 的 strText）。

绿色阴影：已经有值赋给了这个新参数（例如，给 visu2 的 bToggleColor 和 bVisu。

白色阴影：这里不应输入任何值。

可点击值域或按住<Space>键来编辑它。已有的值记录可被复制到另一个域中。选择包含要复制记录的域，并点击按钮复制，然后选择要复制到的域并点击按钮粘贴。

确认设置后点确定更新对话框，新参数设置将显示在相应框架元素的属性对话框（“引用”类）里，比如显示在 visu2 的框架中：



如果，在引用视图（这里是 "basic_visu"）中，您想使用输入变量传递的实例名，您必须用预编译字段“parameterstringof”提供相应的字符串。要显示所有细节，见“字段”'parameterstringof'。

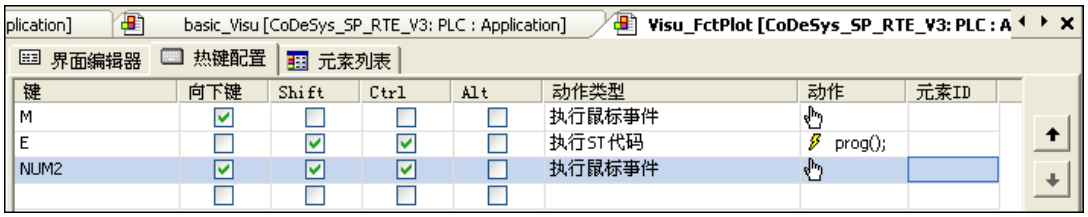
热键配置编辑器

除标准键盘使用外，视图编辑器的这部分还允许为在在线模式时用键盘操作视图配置特殊热键。就是说，一个动作可被指定到一个特定的键或组合键上。这个设置只对当前视图有效。在应用程序中对所有视图都有效的键配置，应在视图管理器中完成。

注意，可被另外设置的某个视图元素的“热键”属性，也在元素属性（输入）中。所以这种元素特性的键定义将被在热键配置编辑器中被描述。它既可在任何修改将被更新的地方，也可在别的编辑器中被编辑。

热键配置编辑器（通过热键配置命令打开）除了位于视图编辑器上方的界面编辑器和其它编辑器外，有效的独立标签。

热键配置编辑器



在表编辑器的每一行，一个键或者键和一个/几个辅助键的组合可关联一个动作。具体如下：

键：键的名称。键名可被手工输入或者双击这个字段在弹出的列表中选择。这个列表提供了所有在设备描述中定义了的键，或者在没有直接指定给一个设备的视图中，是指每一个设备支持的所有键。

按键：如果该选项是激活的，当键被按下时动作会被执行。否则当键被放开时动作会被执行。如果动作应在键按下和抬起时被被执行，表中必须有两个合适的定义。

Shift， Ctrl，Alt：如果该选项被激活，这三个键必须和其它键一起按下才会执行动作。

动作类型：动作类型可双击这个字段，在弹出的列表中选择。动作类型对应于那些在视图元素中的输入配置对话框中所用的类型。

动作：要执行动作的详细配置，基于动作类型和视图元素的输入配置对话框对应的鼠标动作配置。

元素 ID：键上被指定的视图元素的 ID（通过“热键”属性）。在当前视图中是唯一的。

如果一个键与几个动作相关联，他们将按在配置表中的顺序自上而下的执行。您可以通过选择一个键定义（点击表中的某行），并使用表右边的箭头按钮上下移动记录，来改变这个顺序。

注意：这是在设备描述中定义的，键也是在该设备上运行的视图所支持的！

请注意在处理键事件时下面的调用顺序：

1、应用程序的事件句柄，如果被激活（可选）；例如 g_VisuEventManager

2、在视图管理器中，键配置对程序的所有视图都有效

3、标准键盘使用定义

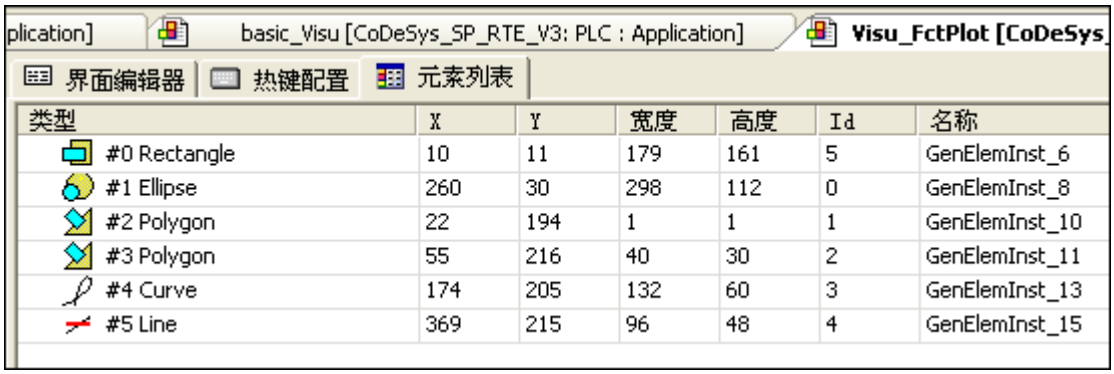
4、各个视图的特殊键配置（由热键配置编辑器处理），主视图要在其他包含在窗口中的视图之前被考虑。

元素列表编辑器

视图编辑器的这部分提供了当前打开的视图的视图元素列表。

元素列表可通过 Element List 命令打开（默认在视图菜单中），它是除了位于视图编辑器上方的界面编辑器和其它编辑器外有效的独立标签。

元素列表



类型	X	Y	宽度	高度	Id	名称
#0 Rectangle	10	11	179	161	5	GenElemInst_6
#1 Ellipse	260	30	298	112	0	GenElemInst_8
#2 Polygon	22	194	1	1	1	GenElemInst_10
#3 Polygon	55	216	40	30	2	GenElemInst_11
#4 Curve	174	205	132	60	3	GenElemInst_13
#5 Line	369	215	96	48	4	GenElemInst_15

元素根据它们在视图区 Z 轴上的位置自上而下的列出，最后面的在第一行。下列值将被显示，但是不能被编辑：

类型：在工具箱中使用的元素类型和图标，从插入序列中得到的初始元素个数。

X，Y：元素左上角位置；（0,0=视图区的左上角）。

ID：内部指定的元素标识。

名称：做为在属性元素中定义的元素名

元素可通过选择相应的行被选中，这与主视图编辑器中的选择同步。需要注意，组中的子元素仅能在元素列表中被选中。

选中元素在 Z 轴上-即在背景-前景轴上-的位置，可通过使用下列命令来修改：

- 至顶层
- 向上一层
- 至底层
- 向下一层

下列命令可被用于在元素列表中存储或删除任何编辑动作：

- 撤销
- 重作
- 删除

2.17 视图管理编辑器

视图管理器用于处理应用中所有视图的常规设置。设置修改后立即生效。

在设备树中，当向应用中插入视图时，视图管理器对象 将自动被添加。

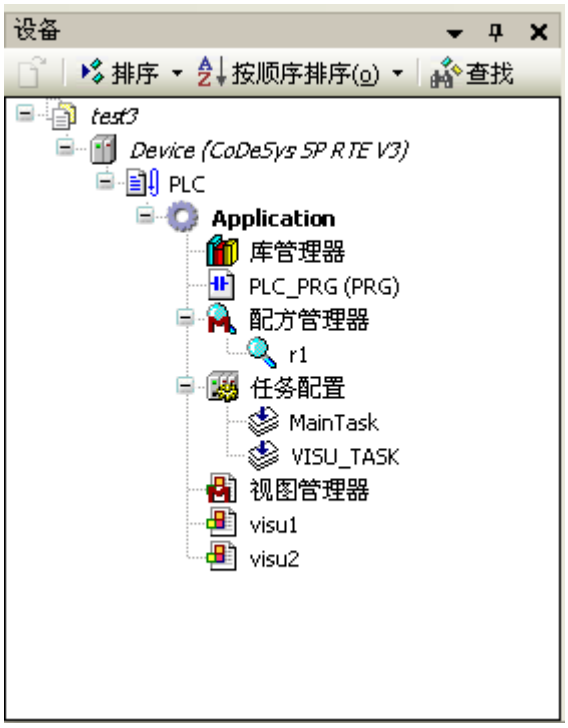
主编辑对话框包括独立的跳格键设置、默认热键和视图下载配置标签。

只有对目标视图和网络视图的设置在独立对话框中处理。可通过视图管理器下设备树中的目标视图或网络视图对象打开。 注意，从编程系统的 V3.3.2.0 版本开始，如果在视图管理器下没有有效的客户对象，那么会自动使用“诊断视图”模式。详情请参见：编程系统中视图运行。

注意：如果在一个工程中创建了一个版本<V3.3.2，相当于设置了一个已定义的编译器版本，工程将依据该版本中的有效视图特征来处理。

下图中，“Application”下的视图管理器主要对视图 PLC_VISU，PLC_VISU_1，Vis1 和 Vis2 进行管理。

设备树中的视图管理器



双击设备树中的条目，或者选中条目并使用右键菜单中的“编辑对象”命令，可以打开视图管理器编辑器。此编辑器在标题为“视图管理器<应用的路径名>”的窗口里打开，包含子对话框的设置和默认热键。

设置

视图管理器编辑器，设置

这个视图管理器对话框的标签包含对所有指定到应用程序的视图有效的设置。

注意：如果使用了“诊断视图”模式，则当前无效设置将置为灰色。

文件传输模式：

传输视图文件到 PLC：如果该选项被激活，用于视图的文件（比如图像和文件列表文件）将在程序下装时从本地文件系统（见后面的选项描述）被拷贝到 PLC 上。比如说，这对于目标视图、以及对于在视图元素的输入配置中可能会用到的默认数字小键盘和视图键盘都是需要的。

使用本地视图文件：文件列表和被视图使用的图像文件从本地文件目录中读取，这些目录在工程设置，视图类中定义。

使用 Unicode 字符串：如果此选项被激活，所有在视图中使用的字符串将被处理为 Unicode 格式。

使用 CurrentVisu 变量：如果激活此选项，全局字符串变量“CurrentVisu”将有效。因此，通过在视图中设置此变量，可以在程序中控制视图的显示。

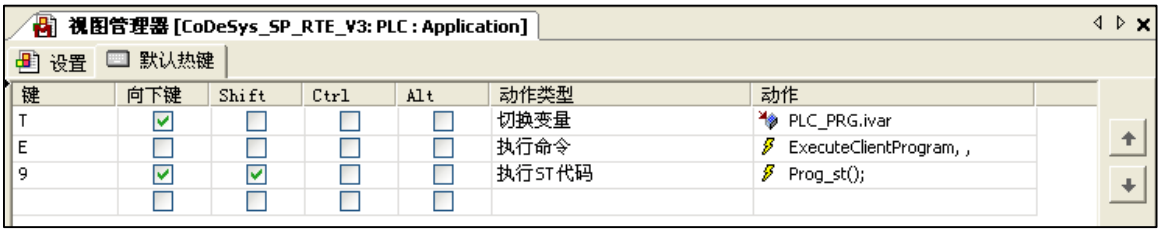
扩展设置：如果必须修改下列设置，则激活此选项。对于标准视图应用，不需要激活此选项。

视图的存储空间：以字节为单位的存储空间，分配给视图使用，默认值为 400000

绘图缓冲区的大小（每个客户端）：以字节为单位的缓冲区大小，分配给客户端绘图动作使用，默认值为 50000。

默认热键

视图管理器编辑器，默认热键

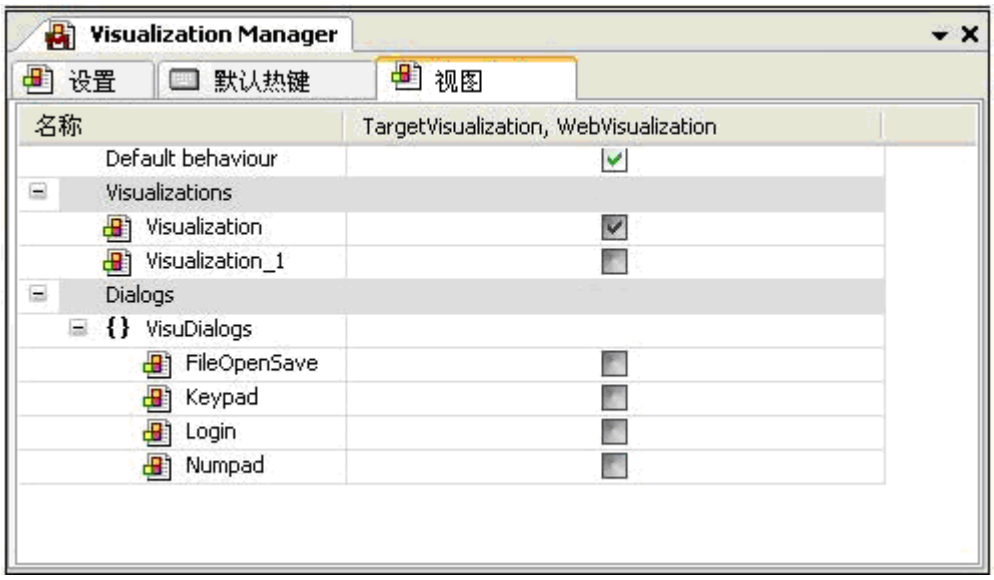


这个视图管理器对话框的标签包含对于所有指定到应用程序的视图有效的键设置。就是说定义的快捷键-如果被相应的设备支持-可被用于用户在在线模式时通过键盘向视图输入。配置器的使用和在视图编辑器中集成的配置器的使用时一样的，所以请参见热键配置编辑器的描述。

此外还有一些与设备无关的、在视图内被支持的、用于导航的标准快捷键。

视图

视图管理器编辑器，视图



“视图管理器”的这个标签显示了当前的有效视图。默认下，只有这些位于当前应用下或被直接引用的视图，可以自动下载到各自的目标系统中。如果打开默认行为选项，在各自的“目标视图”或“网络视图”栏中，它们将会以检查标志标记在表中。如果想要给下载包进一步添加视图，这些视图默认下是不包括的，因为它们只能被间接使用/引用（如：通过任何变量申请），然后添加适当的检查标志。

注意：这个特点替代了先前使用的有效的“视图引用”对象。该对象不再有效，无论下一步怎样工作，已经存在一个对象了。

目标视图编辑器

在视图管理器下面插入的目标视图对象，包括了目标视图的设置。他们可在编辑窗口中被修改，该窗口可通过在对象上双击打开：

目标视图编辑器

注意，运行目标视图的先决条件在目标视图帮助页中有所描述。

启动视图：当目标系统启动时，由目标视图自动显示的视图名。默认时这里输入的是“视图”。要设置另一个视图，您可以使用“输入助手”，它提供了当前可用的视图。

最佳适合：

在线模式最佳适合：如果该选项没被选中，视图将完全按照在对象属性中定义的大小显示在目标视图屏幕上。如果选中了该选项，则目标将根据以下设置调整：

使用自动测得的客户区大小：目标视图将填满整个客户区。

使用指定的客户区大小：视图将填满由下列参数定义的屏幕区域：

客户区宽度：以像素表示的宽度

客户区高度：以像素表示的高度

网络视图编辑器

插入到视图管理器下的网络视图对象包括对目标视图的设置。这些设置可在编辑器窗口（在对象上双击鼠标打开）中被修改：

网络视图编辑器

对运行网络视图的先决条件和基本信息见网络视图帮助页。

启动视图：当网络视图启动时，要自动显示的视图名称。默认填入“视图”。为设置另一个视图，您可以使用输入助手，它提供当前可用视图。

.htm 文件名： 视图基本 HTML 页的名称。在网页浏览器里也必须作为地址输入； 例子：http://localhost:8080/webvisu.htm

更新率（ms）：在网络浏览器中的刷新率，以毫秒表示

默认通讯缓冲大小：以字节表示，定义了用于在网络客户和服务器间的数据传输的可用缓冲区的最大值。

最佳配合：

在线模式最佳配合：如果该选项被激活，视图大小将与网络浏览器的大小相配合。

使用指定的客户区大小：如果该项被激活，视图大小将由如下设置定义：

客户区宽度：以像素表示的宽度

客户区高度：以像素表示的高度

2.18 监控列表编辑器

监视窗口/监视列表编辑器

“监视列表”插件提供监视列表功能。

通过监视命令子菜单（默认在“查看”菜单中）打开监视窗口。监视窗口提供了用于创建监视列表的编辑器。

监视列表可以用来定义和监视不同对象的表达式。在线模式下，通过监视列表，可向控制器中的表达式写入值或强制值。监视窗口中默认设置了 4 个独立的监视列表：监视 1、监视 2、监视 3 和监视 4。在线模式下，被激活应用中的所有当前强制值会自动添加到监视所有强制值列表中。

创建监视列表

在“表达式”列点击鼠标，打开编辑窗口，可在监视窗口中创建监视列表监视<n>。输入所需的监视表达式的完整路径：

监视表达式的语法：<设备名称>.<应用名称>.<对象名称>.<变量名称>

举例： "Dev1.App1.PLC_PRG.ivar" (如下图所示)

变量类型用图标表示：输入变量，输出变量，普通变量。如果注释被添加到了变量的声明部分，它将被移到注释列中。

关闭编辑窗口，变量的类型自动添加到“类型”列。

在线模式下，“值”列用于显示表达式的当前值。

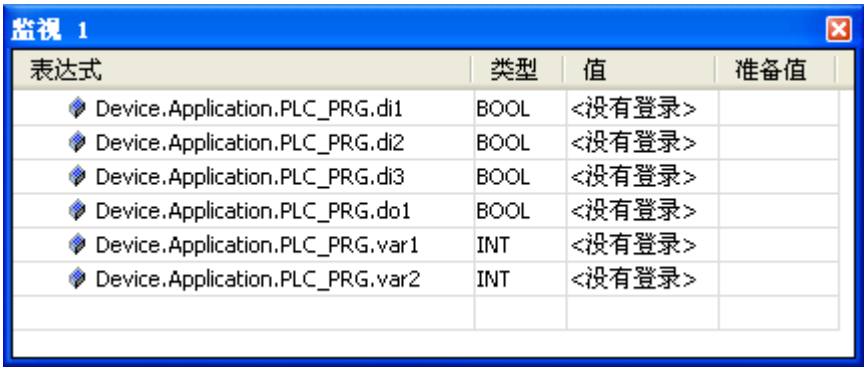
为了对变量设定预设值，可以点击预设值对话框列中的赋值区域，直接输入所需设定值。

对于布尔变量的处理则更加容易。根据下列顺序，使用回车键或空格键，可切换布尔变量的预设值：如果值为 TRUE，则步骤是：FALSE -> TRUE -> 空。

如果值为 FALSE，则步骤是：TRUE -> FALSE -> 空。

在其它行中对其他的表达式/变量进行同样的操作。离线模式下的监视窗口如下图所示：它包括对象 PLC_PRG 和 Prog_St 的表达式。注意，对于结构变量，例如功能块实例，输入实例名（见例中：“Dev1.App1.PLC_PRG.fbinst”）后，将自动添加实例的各个组成部分。通过鼠标点击加/减符号，可以将这些组成部分显示或隐藏在一个折叠中。

离线模式下的监视窗口举例



在线模式下，此窗口用于监视。

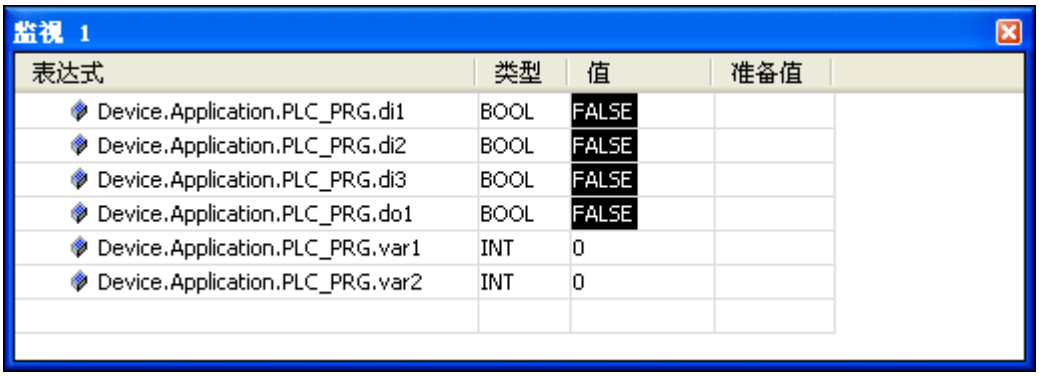
在线模式下的监视列表

监视：

在线模式下，监视列表“监视<n>”在“值”列中显示变量的当前值。

有关如何建立监视列表，以及如何处理结构变量，请点击这里获得更多信息。

示例，在线模式下的监视窗口



写入值：

通过“写入值”或“强制值”命令——这些命令在其它监视窗口中也可用（例如，声明编辑器），将在“预设值”列中输入的值，写入或强制给控制器的表达式中。

监视所有强制：

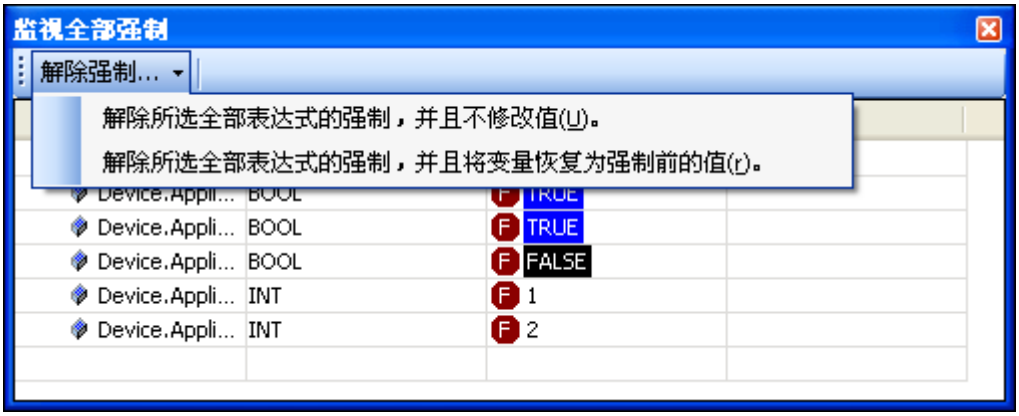
这是特殊的监视列表窗口，在线模式下，被激活应用中的所有当前强制值会自动添加到该窗口中。与在线模式 “监视<n>” 列表窗口类似，表达式、类型、值和预设值会显示出来。

通过“解除强制...”按钮，选用下面命令之一来释放强制值。

解除所选全部表达式的强制，并且不修改值。

解除所选全部表达式的强制，并且将变量恢复围墙之前的值。

举例，监视所有强制



3 编程参考

3.1 声明

3.1.1 变量声明

用户可以在 POU 的声明部分或者通过自动声明对话框、DUT 和 GVL 编辑器声明变量。

要声明的变量的“类型”（在声明对话框中指”范围”）可由“关键字”指定，每个带有关键字的组都可以包含若干个变量的声明。常见的声明以关键字“VAR”开头、“END_VAR”结束。点击 这里可以查看变量的类型。

VAR_INPUT
VAR_OUTPUT
VAR_IN_OUT
VAR_GLOBAL
VAR_TEMP
VAR_STAT
VAR_EXTERNAL
VAR_CONFIG

变量的类型关键字之后可以添加属性。属性也是一种关键字， 例如"RETAIN" (VAR_INPUT RETAIN)。

变量的声明必须符合以下规则：

语法：< 标识符> {AT <地址>}:<类型> {:=<初始化值>};

{}中为可选部分。

标识符：标识符就是变量的名称。定义标识符时必须注意下列事项，并请参考这里给出的命名建议。

- 不得包含空格和特殊字符。
- 标识符中不区分大小写字母。例如“VAR1”、“Var1”和“var1”都是同一个标识符。
- 标识符中可以识别下划线（例如“A_BCD”和“AB_CD”会被识别为两个不同的标识符），但标识符中不能有连续的两个下划线。
- 对于整个标识符和其中有意义的字符部分的长度没有限制
- 必须遵守下表中列出的关于重复使用标识符的规则：

标识符的重复使用（命名空间）：

- 不能在局部范围内重复使用同一个标识符。
- 标识符名称不能与关键字相同。
- 可以在全局范围内多次使用一个标识符，因此一个局部变量的名称可以与全局变量相同。此时

局部变量在其所属 POU 内拥有优先权。

◆ 一个全局变量列表中定义的变量名称可以与另一个全局变量列表中定义的变量相同。关于这一规则，请注意下述的 IEC61131-3 扩展标准的特性。CoDeSys V2.3 版本中没有用到这些特性：

1、全局操作符：以“.”开头的实例路径通常代表全局量。因此，若某个局部变量如“ivar”与全局变量的名称相同，那么“.ivar”将指向全局变量。

2、全局变量列表名可以作为它所包含变量的命名空间。因此可以在不同全局变量列表中以相同的名称声明不同变量。您只需在各个变量名前冠以列表名，就可以访问需要的变量。例如：
globlist1.ivar := globlist2.ivar; (* 将 globlist2 的变量 ivar 复制给 globlist1 的 ivar *)

3、访问定义在某个库的全局变量列表中的变量，可以使用这条语法：<库的命名空间>.<全局变量列表名>.<变量>。关于库命名空间的介绍，请看下文。例如：
globlist1.ivar := lib1.globlist1.ivar (*将库 lib1 中 globlist1 的变量 ivar 复制给 globlist1 的变量 ivar *)

◆ 当您通过库管理器将一个库包含到另一个库中时，还可以给前者定义一个命名空间，这样就可以通过<库的命名空间>.<模块名|变量名> 访问库模块或变量。请注意在嵌套的库中，必须连续地声明所有相关库的命名空间。例如：如果lib1 被 lib0 引用，那么可以用 Lib0.Lib1.fun 来访问 Lib1 的模块fun:ivar := Lib1.fun(4, 5); (* 将 fun 的返回值复制给工程中的变量ivar *) 在引用库 Lib 的属性中激活“将所有的 IEC 符号发布给工程，引用将被直接包含”复选框，模块 fun 也可以通过"Lib0.fun"来直接访问。

AT<地址>：可以使用关键字 AT 把变量直接联结到确定地址。

您也可以在功能块中为变量指定一个不完整的地址描述。为了能在局部实例中使用这个变量，必须在“变量设置”中输入相应设置。

类型：有效的数据类型。也可以在数据类型之后选择添加“:=<初始化值>”

可以选择将 pragma 指令添加到对象的声明部分，从而满足不同的代码生成方式。

提示

请注意，您可以使用自动声明。
可以使用快捷方式以提高声明的输入速度。
请注意，您可以在对象的声明部分使用影响代码生成的 pragma 命令。

3.1.2 标识符命名建议

在声明变量（变量名）、用户自定义数据类型和创建 POU（函数、功能块、程序）时定义标识符。为了使标识符的名称尽量不与其他名称重复，除了必须遵守的事项（请看这里）之外，您可能还需要参考以下一些建议。

查看以下标识符的命名：

变量名

CoDeSys V3.x 库中的变量名

用户自定义数据类型（DUT）

CoDeSys V3.x 库中的用户自定义数据类型（DUT）

函数、功能块、程序（POU）、动作

CoDeSys V3.x 库中的 POU

视图名称

（1）变量名

给应用程序和库中的变量命名时应当尽可能地遵循匈牙利命名法：

每一个变量的基本名字中应该包含一个有意义的简短描述。

基本名字中每一个单词的首字母应当大写，其它字母则为小写（例如：FileSize）

依据变量的数据类型，在基本名字之前加上小写字母前缀。

请看下表列出的一些特定数据类型的推荐前缀和其它相关信息：

数据类型	下限	上限	数据长度	前缀	备注
BOOL	FALSE	TRUE	1 Bit	x*	
				b	保留
BYTE			8 Bit	by	不能进行算术运算
WORD			16 Bit	w	不能进行算术运算
DWORD			32 Bit	dw	不能进行算术运算
LWORD			64 Bit	lw	不能进行算术运算
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	
DATE_AND_TIME				dt	
DATE				date	
ENUM			16 Bit	e	
POINTER				p	
ARRAY				a	

* 为了区别于 BYTE 类型并与 IEC 编程习惯相符，选择 x 作为布尔变量的前缀。

例如：

bySubIndex: BYTE;

sFileName: STRING;

udiCounter: UDINT;

在嵌套声明中，按照声明顺序连接前缀：

例如：

pabyTelegramData: POINTER TO ARRAY [0..7] OF BYTE;

将功能块实例和用户自定义数据类型变量作为前缀时使用 FB-或者数据类型名称的快捷方式。（如：sdo）

例如：

cansdoReceivedTelegram: CAN_SDOTelegram;

```
TYPE CAN_SDOTelegram :      (* sdo 为前缀 *)
STRUCT
    wIndex:WORD;
    bySubIndex:BYTE;
    byLen:BYTE;
    aby: ARRAY [0..3] OF BYTE;
END_STRUCT
END_TYPE
```

局部常数（c）以前缀 c 和下划线开头，后面接类型前缀和变量名。

例如：

```
VAR CONSTANT
    c_uiSyncID: UINT := 16#80;
END_VAR
```

对于全局变量(g)和全局常数(gc)，需要在库前缀后接附加前缀+下划线

例如：

```
VAR_GLOBAL
    CAN_g_iTest: INT;
END_VAR
VAR_GLOBAL CONSTANT
    CAN_gc_dwExample: DWORD;
END_VAR
```

(2) CoDeSys V3.x 库中的变量名

基本信息请参见上文的（1）变量名，以下情况除外：由于使用命名空间取代了前缀的功能，所以全局变量和常数不需要添加库前缀。

例如：

g_iTest: INT; (声明)

CAN.g_iTest (实现，在应用程序中调用)

(3) 用户自定义数据类型（DUT）

结构：每个结构数据类型名称都由一个库前缀（如 CAN）、下划线和此结构的适当简短描述（如 SDOTelegram）组成。这个结构的变量使用的组合前缀应紧接在冒号之后。

例如：

```
TYPE CAN_SDOTelegram :        (* sdo 为前缀 *)
STRUCT
    wIndex:WORD;
    bySubIndex:BYTE;
    byLen:BYTE;
    abyData: ARRAY [0..3] OF BYTE;
END_STRUCT
END_TYPE
```

枚举由库前缀开头（如：CAL），后面接下划线和大写的标识符。

请注意在以前版本的 CoDeSys 中，由于没有自动转化为 INT 类型值，ENUM 值> 16#7FFF 引起了错误。因此应当把 ENUM 值定义成正确的 INT 类型。

例如：

```
TYPE
CAL_Day :( CAL_M
ONDAY,
CAL_TUESDAY,
CAL_WEDNESDAY,
CAL_THIRSDAY,
CAL_FRIDAY,
CAL_SATURDAY,
CAL_SUNDAY);
```

声明：

```
eToday: CAL_Day;
```

(4) CoDeSys V3.x 库中的用户自定义数据类型（DUT）

由于使用命名空间代替了前缀的功能，因此 CoDeSys V3.x 库中的 DUT 名称没有库前缀。定义枚举成员时也没有库前缀。

例如（在命名空间为 CAL 的库中）：

TYPE

Day :(MONDAY,

TUESDAY,

WEDNESDAY,

THIRSDAY,

FRIDAY,

SATURDAY,

SUNDAY);

声明：

eToday: CAL.Day;

在应用中使用这个变量：

IF eToday = CAL.Day.MONDAY THEN

(5) 函数、功能块、程序（POU）、动作

函数、功能块和程序的名称由库前缀（如：CAN）、下划线和一个 POU 的简短表意名称（如：SendTelegram）组成。与变量名相同的是，POU 名称中单词的第一个字母应该大写，其它字母则为小写。建议用一个动词和一个名词组成 POU 的名称。

例如：

FUNCTION_BLOCK CAN_SendTelegram (* canst 为前缀 *)

声明部分中POU 的简短描述应当是一种注释。进一步来说，所有的输入和输出都应当带有注释。而对于功能块而言，已安装实例的附加前缀应紧跟在名称之后。

没有前缀的动作：这种动作以 pry 开头，只能在内部调用（如被 POU 本身调用）。

每一个函数——为了与以前版本CoDeSys 兼容——至少需要一个参数。外部函数不能将结构体作为返回值。

(6) CoDeSys V3.x 库中的 POU

由于使用命名空间取代了前缀的功能，所以 CoDeSys V3.x 库中的POU 名称没有库前缀。创建方法名称的规定与创建动作相同。方法的输入应当带有英文注释，并且应在方法的声明中加入简短的描述。接口名称应以字母“I”开头，例如：ICANDevice

(7) 视图名称

注意：现在请不要像对工程中的其它对象一样对视图进行命名，因为此类操作会在改变视图时导致一些问题。

3.1.3 变量初始化

所有变量声明的默认初始化值均为 0。 但用户可以在各个变量和数据类型的声明中添加自定义的初始值。

使用赋值操作符":="指定用户自定义初始值。这个值可以是任意有效的 ST 表达式。因此常量、其它变量和函数都可以作为初始值，而程序只检查给其他变量初始化的变量自身是否已被初始化。

正确的变量初始化举例：

```
VAR
 var1:INT := 12;                    (* 整数变量的初始化值为 12 *)
 x : INT := 13 + 8;                 (* 使用常量表达式定义初始化值 *)
 y : INT := x + fun(4);             (* 使用含有函数调用的表达式定义初始化值；请注意此处的顺序！ *)
 z : POINTER TO INT :=              (* IEC61131-3 中未描述的情况：使用地址函数定义初始化值；请注意此处：在线修改时指针不会被初始化！ *)
 ADR(y);
END_VAR
```

参见：变量任意初始化

注意：从编译器版本 3.3.2.0 开始，全局变量列表的变量总在 POU 的局部变量之前初始化。

3.1.4 变量任意初始化

变量可以初始化为任意有效的 ST 表达式。初始化时可以访问本范围之外的其他变量，也可以调用函数。但程序需检查给其他变量初始化的变量自身是否已被初始化。

正确的变量初始化举例：

```
VAR
x : INT := 13 + 8;
y : INT := x + fun(4);
z : POINTER TO INT := ADR(y); (* be careful: the pointer will not be initialized during online change! *) (* 请注意：在线修改时指针不会被初始化！*)
END_VAR
```

3.1.5 声明编辑器

声明编辑器是一个用于变量声明的文本或表编辑器。它通常与语言编辑器一起出现。

3.1.6 自动声明对话框

可以在选项对话框的“文本编辑器”类别 ->“编辑”中定义自动声明对话框。当编辑器的实现部分尚未输入完声明字符串但已经按下<回车>键时，应立刻弹出自动声明对话框。用户可以在这个对话框中进行变量声明。

如果配置文件支持，可以使用“自动声明”命令（默认在编辑菜单）打开对话框。

该对话框可被命令’自动声明’打开，在编辑菜单是缺省的，也可以通过快捷键<Shift>+<F2>打开。如果选择了已经声明的变量，在该对话框中可以编辑其声明。

3.1.7 快捷方式

在声明变量时使用的声明编辑器和其它文本编辑器中可以使用“快捷方式”。

当您输入完一行声明代码并按下<Ctrl>+<Enter>键时，就可以激活此方式。此时可以使用快捷方式来代替完整的声明语句。

用户可以使用下列快捷方式：

- 每行的最后一个标识符作为需声明变量标的识符。
- 声明的类型取决于本行最末一个标识符。也就是说将有以下情况：

B 或 BOOL	代表	BOOL
I 或 INT	代表	INT
R 或 REAL	代表	REAL
S 或 string	代表	STRING

- 如果没有通过以上规则确定声明的类型，则将默认为布尔类型，不会把最后一个标识符作为声明类型（见下文：例 1）。
- 所有常数都将根据声明的类型转换成一个初始值或字符串（见下文：例 2 和例 3）。
- 根据“AT”属性扩展地址（例如%MD12）（见下文：例 4）。
- 分号（;）后面的文本被视为注释（见下文：例 4）。
- 忽略本行的所有其它字符（见下文，如例 5 中的感叹号）。

例如：

	快捷方式	声明
（1）	A	A: BOOL;
（2）	A B I 2	A, B: INT := 2;
（3）	ST S 2; A string	ST:STRING(2); (* 一个字符串 *)
（4）	X %MD12 R 5 Real Number	X AT %MD12: REAL := 5.0;(* 实数 *)
（5）	B !	B: BOOL;

3.1.8 AT 声明

若需要将工程变量指定到一个确定的地址，您可以在变量的声明里键入此地址。注意，将变量赋值到一个地址，也可以在 PLC 配置（设备树）里的设备的映射对话框中完成。

语法：<标识符> AT <地址> : <数据类型>;

关键字AT 之后必须跟随一个有效地址，查看地址帮助页面获取更多信息并考虑字节地址模式下的可能重叠。此声明允许为地址分配有意义的名字。进出信号的任何改变只能在一个地方进行（比如声明）。

给变量指定地址时，请注意以下几点：

- 需要某个输入的变量不能以写方式访问。编译器以 error 截断操作。
- 只能对局部或者全局变量进行 AT 声明，而 POU 的输入和输出变量不能进行此类声明。
- AT 声明不能和 VAR RETAIN 或 VAR PERSISTENT 一起使用。
- 如果结构或功能块的成员进行了 AT 声明，那么每个实例都将访问同一个内存地址。这与传统的编程语言，例如 C 中的“静态变量”相同。
- 目标设置也决定了结构体的内存布局。

例如：

counter_heat7 AT %QX0.0: BOOL;

lightcabinetimpulse AT %IX7.2: BOOL;

download AT %MX2.2: BOOL;

注意：如果布尔型变量被指定到一个字节、字或者双字地址，那么它们将占用一个字节而不是偏移量之后的一个二进制位！这个字节的值为 TRUE 或者 FALSE。

请注意，您也可以在 PLC 设置（设备树）的设备映射对话框中完成把一个变量指定到地址的操作。

3.1.9 关键字

所有编辑器中的关键字均为大写。

以下的字符串保留为关键字，不能作为变量或 POU 的标识符：

ABS

ACOS

ACTION (仅用于导出格式)

ADD

ADR

AND

ANDN

ARRAY

ASIN

AT

ATAN

BITADR

BOOL

BY

BYTE

CAL

CALC

CALCN

CASE

CONSTANT

COS

DATE

DINT

DIV

DO

DT

DWoRD

ELSE

ELSIF

END_ACTION (仅用于导出格式)

END_CASE

END_FOR

END_FUNCTION (仅用于导出格式)

END_FUNCTION_BLOCK (仅用于导出格式)

END_IF

END_PROGRAM (仅用于导出格式)

END_REPEAT

END_STRUCT

END_TYPE

END_VAR

END_WHILE

EQ

EXIT

EXP

EXPT

FALSE

FOR

FUNCTION

FUNCTION_BLOCK

GE

GT

IF

INDEXOF

INT

JMP

JMPC

JMPCN

LD

LDN

LE

LINT

LN

LOG

LREAL

LT

LTIME

LWORD

MAX

MIN

MOD

MOVE

MUL

MUX

NE

NOT

OF

OR

ORN

PARAMS

PERSISTENT

POINTER

PROGRAM

R

REFERENCE

READ_ONLY

READ_WRITE

REAL

REPEAT

RET

RETAIN

RETC

RETCN

RETURN

ROL

ROR

S

SEL

SHL

SHR

SIN

SINT

SIZEOF

SUPER

SQRT

ST

STN

STRING

STRUCT

SUPER

SUB

TAN

THEN

TIME

TO

TOD

TRUE

TRUNC

TYPE

UDINT

UINT

ULINT

UNTIL

USINT

VAR

VAR_ACCESS (仅用于指定情况，取决于硬件)

VAR_CONFIG

VAR_EXTERNAL

VAR_GLOBAL

VAR_IN_OUT

VAR_INPUT

VAR_OUTPUT

VAR_STAT

VAR_TEMP

WHILE

WORD

WSTRING

XOR

XORN

此外，输入助手中所列的所有转换操作符都被视为关键字。

3.1.10 局部变量-VAR

使用关键字 VAR 开头并以 END_VAR 结束，可以声明 POU 的所有局部变量。这些变量没有外部连接；也就是说不能从外部写入这些变量。

请注意可以对“VAR”添加属性。

例如：

```
VAR
iLoc1:INT; (* 1. Local Variable 局部变量 *)
END_VAR
```

3.1.11 输入变量-VAR_INPUT

使用关键字VAR_INPUT 开头并以END_VAR 结束，声明POU 的所有输入变量。这意味着调用输入变量时将给出变量的值。

请注意可以对“VAR_INPUT”添加属性。

例如：

```
VAR_INPUT
iIn1:INT (* 1. Inputvariable 输入变量*)
END_VAR
```

3.1.12 输出变量-VAR_OUTPUT

使用关键字 VAR_ OUTPUT 开头并以 END_VAR 结束，声明 POU 的所有输出变量。这些变量将被传回调用这个函数的 POU，以备他用。

请注意可以对“VAR_OUTPUT”添加属性。

例如：

```
VAR_OUTPUT
 iOut1:INT; (* 1. Outputvariable 输出变量*)
END_VAR
```

函数和方法中的输出变量：

根据 IEC61131-3 草案 2，函数（和方法）可以有附加输出。但这些输出必须在函数调用中指明，例如：

```
fun(iIn1 := 1, iIn2 := 2, iOut1 => iLoc1, iOut2 => iLoc2);
```

如上所示，fun 函数的返回值将写入局部声明变量 loc1 和 loc2。

3.1.13 输入和输出变量-VAR_IN_OUT

使用关键字 VAR_IN_OUT 开头并以 END_VAR 结束，声明 POU 的所有输入和输出变量。

提示：此种类型变量所传递的变量值已经发生改变（“作为指针传递”，通过引用调用）。也就是说这类变量的输入值不能为常数。因此，即使对于一个功能块的VAR_IN_OUT 变量而言，也不能通过<功能块程序><输入/输出变量>从外界直接对其进行读写。

例如：

```
VAR_IN_OUT
 iInOut1:INT; (* 1. 输入输出变量 *)
END_VAR
```

3.1.14 全局变量-VAR_GLOBAL

可以把应用于整个工程中的“普通”变量、常数，外部变量或保持型变量声明为全局变量。

注意：如果在 POU 中定义的局部变量与全局变量的名字相同，那么这个局部变量在 POU 内部拥有优先权。

注意：从编译器版本 3.2.0.0 开始， 全局变量将始终在 POU 的局部变量之前被初始化。.

声明本地变量须使用关键字 VAR_GLOBAL 开头、并以 END_VAR 结束。请注意可以对“VAR_GLOBAL”添加属性。

名称前带有点号的变量将被作为全局变量：例如“.iGlobVar1”。

全局变量列表用来在工程中结构化和处理全局变量。用户可以通过添加对象命令添加“GVL”。

3.1.15 临时变量-VAR_TEMP

此特性是 IEC 61131-3 标准的扩展。

只能在程序和功能块的内部进行 VAR_TEMP 声明，也只能在程序体和功能块内部访问此类变量。每当调用 POU 时，声明为 VAR_TEMP 的变量都会（重新）初始化。

必须使用关键字 VAR_TEMP 开头、并以 END_VAR 结束在本地声明临时变量。

3.1.16 静态变量-VAR_STAT

此特性是 IEC 61131-3 标准的扩展。

静态变量用于功能块、方法和函数。可以使用关键字VAR_STAT 开头、并以END_VAR 结束在本地声明此类变量，它们在被各自 POU 第一次调用时初始化。

只有在静态变量声明的范围内才可以访问这些变量（与 C 中的静态变量相似），但与全局变量的相同之处在于，退出POU 之后它依然保持原来的值不变。例如，可以在函数中用它们来计算这个函数被调用的次数。

请注意可以对“VAR_STAT”添加属性。

3.1.17 外部变量-VAR_EXTERNAL

这是一种被“导入”到 POU 中的全局变量。

在全局变量列表中使用关键字 VAR_EXTERNAL 开头并以 END_VAR 结束，本地声明必须完全符合全局声明。若全局声明不存在，将会出现错误信息提示。

注意：CoDeSys 中没必要把变量定义为外部变量。 提供这些关键字是为了与 IEC 61131-3.兼容

注意：不能把”AT 声明"和 "VAR RETAIN" 或者 "VAR PERSISTENT 一起使用

例如：

```
VAR_EXTERNAL
  iVarExt1:INT:=12; (* 1st external variable 第一个外部变量 *)
END_VAR
```

3.1.18 变量类型的属性关键字

变量类型的属性关键字

声明变量类型时可以添加下列属性关键字以确定范围：

RETAIN：参见 RETAIN 类型的保持型变量

PERSISTENT：参见 PERSISTENT 类型的 r 保持型变量

CONSTANT：参见常数

3.1.19 保持型变量-RETAIN、PERSISTANT

保持型变量可以在整个程序运行期间保持它们的值。

此类变量可以声明为为“retain 变量”或者更加严格的“Persistent 变量”。每一种类型都使用独立的存储空间。

不同的声明类型使保持型变量在复位、下装和 PLC 重新启动时具有不同的“抗拒”能力。在应用中主要使用两种保持性标志的联合类型（见下面 Persistent 变量）。

注意：打开 V2.3 版本的工程时，Retain 变量的声明依然有效，但必须重新进行 persistent 变量的声明，请参见下文：必须建立一个专属 persistent 的全局变量列表！

Retain 类型变量

保持型变量可以在 POU 或全局变量列表中使用关键字“RETAIN”来声明此类变量。

例如：

```
VAR RETAIN
  iRem1 : INT; (* 1.Retain 变量*)
END_VAR
```

Retain 属性：以关键字RETAIN 声明Retain 类型变量。Retain 型变量在控制器正常关闭、打开（或收到在线命令“热复位”），甚至意外关闭之后这类变量仍然能保持原来的值。随着程序重新开始运行，存储的值能继续发挥作用。生产线上的计件器便是一个典型的例子：电源被切断之后，它仍然可以在再次启动时继续计数。
而其它所有变量此时都将被重新初始化，变为指定初始值或标准初始化的值。

与 Persistent 变量不同的是，Retain 变量将会在新的程序下装之后被重新初始化。

在控制器正常关闭、打开或收到“热复位”的在线命令、甚至在控制器意外关闭之后，Retain 变量仍可以保持原来的值。随着程序的重新运行，存储的值能继续发挥作用。而所有其它值将被重新初始化，变为指定初始化值或默认值。

应用举例：生产机器中的计件器，在掉电之后可以继续计数。

但 Retain 变量在“初始化复位”、“冷复位”和程序下装之后将会重新初始化；而相对地，persistent 变量只在“初始化复位”之后重新初始化。

内存位置: 依赖于设备

(1) Retain 型变量存储于标准数据存储区，并且被循环拷贝到一个 Retain 内存中，或者

(2) Retain 型变量仅仅被存储在一个单独的内存区中。

Retain 属性可以与 Persistent 属性联合。请看下文，一览表。

注意：
如果将一个程序中的局部变量声明为“VAR RETAIN”，那么这个变量将存储在 retain 区域（与全局 retain 变量相似）。

如果功能块中的一个局部变量被声明为 VAR RETAIN，

-见上，内存位置，选项（1）：只有功能块中的 Retain 型变量被存储于 Retain 区内，或者
-见上，内存位置，选项（2）：完整的实例将被存储在 Retain 区中，这时只有被声明的 Retain 型变量被做为保留值被处理。

将函数中的一个局部变量声明为“VAR RETAIN”的操作是无效的。这个变量也不会被存储到 retain 区域！把函数中的一个局部变量声明为 PERSISTENT 的操作也将无效！

Persistent 变量

由关键字“PERSISTENT”识别 Persistent 变量（VAR_GLOBAL PERSISTENT）。它们仅在（应用）重设起点时被重新初始化。与 retain 变量相比，它们在下装后仍保持原有的值。一个 Persistent变量的应用例子是操作时间计数器，即使在断电和下装以后它仍然应该继续计数。见下面的一览表。

Persistent 变量的处理情况与 CoDeSys V2.3 中有所不同，如下所述：

Persistent 变量只能在特定的“Persistent 变量”对象类型全局变量列表中声明，这一类型是指定一个应用的。每一个应用之中可能只有一个这种列表。

注意 ： 从 V3.3.0.1 版开始用“VAR_GLOBAL PERSISTENT”声明，这与 “VAR_GLOBAL PERSISTENT RETAIN”或“VAR_GLOBAL RETAIN PERSISTENT”声明方法等效。

与 retain 变量一样，Persistent 变量也存储在一个独立的内存区中。

例如：

```
VAR GLOBAL PERSISTENT RETAIN
  iVarPers1 : DINT; (* 1. App1 中的 Persistent+Retain 变量 *)
bVarPers : BOOL;  (* 2. App1 中的 Persistent+Retain 变量 *)
END_VAR
```

注意：目前只有全局 persistent 变量可以这样。

目标系统必须提供一个独立的存储空间来保存每一个应用的 persistent 变量列表。

每一次重新装载应用时，PLC 中的 persistent 变量列表将与工程中的列表核对。此时通过应用的名称识别 PLC i.a.中的列表。一旦两者不一致，将会提示用户重新初始化应用中所有的 persistent 变量。这种错误可能是由于对已经存在于列表中的声明进行重命名、移除或其它操作造成的。

用户只能在列表的末尾添加新的声明。在下装过程中，这些声明将会被识别为“新”声明，而不会造成整个列表的重新初始化。

保持型变量的行为概述表

x =保持原值　　- =被初始化

after Online commandl 收到在线命令之后	VAR	VAR RETAIN	VAR PERSISTENT	VAR PERSISTENT VAR RETAIN PERSISTENT VAR PERSISTENT RETAIN
Reset Warm <application> 热复位	-	x	x	x

Reset Kalt <application>	-	-	x	x
Reset Ursprung <application>	-	-	-	-
Download <application> 下装	-	-	x	x
Online Change <application> 在线修改	x	x	x	x
Reboot PLC 重新启动 PLC	-	x	-	x

3.1.20 常量 CONSTANT，文字

用关键字 CONSTANT 标识常量。它可以声明为局部常量或全局常量。

语法：

```
VAR CONSTANT
<标识符>:<类型> := <初始化值>;
END_VAR
```

例如：

```
VAR CONSTANT
  c_iCon1:INT:=12; (* 1. 常量*)
END_VAR
```

参见对可能常量列表的操作数描述。并请注意其中可以使用类型常量：

文字

一般说来，使用 IEC 常量时会尽可能地使用占用空间最小的数据类型。如果需要使用其它数据类型，可以借助文字实现，而不需再进行明确的常量声明。为此常量之前需要添加一个定义其类型的前缀。

书写形式如下：

语法：<类型>#<文字>

<类型>指定需要的数据类型；可以键入各种简单数据类型。需写成大写字母。

<文字>指定常量。此处键入的数据必须与<类型>指定的数据类型相匹配。

例如：

```
iVar1:=DINT#34;
```

如果常量转换为目标类型时有数据损失，将会出现错误提示：

在可以使用普通常量的地方均可以使用文字。

在线模式的常量

只要缺省设置替代常量(编译选项)被激活，在线模式下声明或监视窗口的值一栏的常量前会出现标识。这种情况下，它们不能被强制或写入访问。

3.1.21 变量配置-VAR_CONFIG

“变量配置”可以用来将功能块变量映射到过程映像，例如 I/O 设备之中，而不需指定已经在功能块变量声明中的确定地址。此处确定地址的指定是围绕所有在全局 VAR_CONFIG 列表中的功能块实例进行的。

为此您可以把“不完整的”地址指定给在关键字 VAR 和 END_VAR 之间声明的功能块变量。这些不完整的地址用星号标志。

语法：<标识符> AT %<I|Q>* : <数据类型>;

不完整定义的地址的使用举例：

```
FUNCTION_BLOCK locio
VAR
  xLocIn AT %I*: BOOL := TRUE;
  xLocOut AT %Q*: BOOL;
END_VAR
```

此处定义了两个局部 I/O 变量，一个局部输入（%I*）和一个局部输出（%Q*）。

地址的最终定义在全局变量列表的“变量配置”中进行：

为此通过添加对象命令向设备窗口中添加一个“全局变量列表”对象（GVL）。这里在关键字 VAR_CONFIG 和 END_VAR 之间键入带有确定地址的实例变量的声明。

必须通过完整的实例路径指定实例的变量，在这里各个POU 和实例名称按照时期分开。声明必须包含一个地址，这个地址的种类（输入/输出） 对应于那个在功能块中不完整地指定的地址（%I*, %Q*）。数据类型也必须与功能块中的声明相符。

语法：<实例变量路径> AT %<I|Q><位置> : <数据类型>;

当实例不存在从而导致实例路径无效时，将显示配置变量错误。另一方面，当指定给不完整的地址的实例变量的确定地址配置不存在时，也会提示出错。

变量配置举例

假设程序中给出了如下对功能块 locio 的定义（参见上文的例子）：

```
PROGRAM PLC_PRG
VAR
```

locioVar1: locio;

locioVar2: locio;

END_VAR

那么以下为正确的变量配置的形式：

VAR_CONFIG

PLC_PRG.locioVar1.xLocIn AT %IX1.0 : BOOL;

PLC_PRG.locioVar1.xLocOut AT %QX0.0 : BOOL;

PLC_PRG.locioVar2.xLocIn AT %IX1.0 : BOOL;

PLC_PRG.locioVar2.xLocOut AT %QX0.3 : BOOL;

END_VAR

注意：对直接映射 I/O 的改变将即刻显示在过程映像中，而对通过VAR_CONFIG 映射的变量的改变则不会在相应的任务结束之前显示出来。

3.1.22 用户自定义数据类型的声明和初始化

除标准数据类型之外，还可以使用用户自定义数据类型。

关于声明和初始化的信息请参见 DUT（数据单元类型）页面。

3.1.23 可扩展函数-PARAMS

（此功能尚未实现）

作为对 IEC61131-3 标准的扩展，可以用相同类型的输入参数的变量数量来声明函数或方法。

语法：<可扩展_声明> : = <函数名> : PARAMS OF <数据类型>

例如：

FUNCTION funHelp : INT

VAR_INPUT

pint : PARAMS OF <基本数据类型>;

END_VAR

隐含的 pint 指向一个包含 INT 类型成员 Count 和 POINTER TO <基本数据类型>的参数的结构。

调用 funHelp 时，也可以指向一个<基本数据类型>的任意长度的数组，或者一个<基本数据类型>表达式的变量数量，或同时指向这两者。

例如：

x := funHelp(1, 2, 3);

y := funHelp(intArray);

z := funHelp(1, intArray, 2);

一个函数可能只包含一个 PARAMS-输入，并且这个输入一定是这个函数的最后一个输入。没有

在函数调用内明确指定 PARAMS-输入的语法。

3.1.24 FB_init，FB_reinit 方法

FB_init

“FB_init”方法取代了 CoDeSys V2.3 中的 INI 操作符。

“FB_init”的方法是一个用于功能块的特殊方法。可以明确声明，也可以在任意情况下隐含使用。因此所有功能块可以在任何需要的时候使用这一方法。

初始化方法包含功能块的初始化代码，这些代码已经在功能块的声明部分进行了声明。如果明确声明初始化方法，那么隐含的初始化代码将会嵌入到这个方法之中。编程用户也可以添加其他的初始化代码。

注意：执行到用户定义的初始化代码时，功能块已经通过隐含的初始化代码完全初始化了。

每一个声明实例下装之后就会调用初始化方法。注意：在线修改时，初始化值将会被前一个值覆盖！

继承条件下的调用顺序请查看：FB_exit, 执行顺序

请注意通过 FB_init: call_after_init 属性初始化之后被自动调用的功能块实例方法的定义。

初始化方法的接口：

```
METHOD fb_init : BOOL
VAR_INPUT
   bInitRetains : BOOL; // 若值为 TRUE，初始化 retain 变量(热启动 / 冷启动)
   bInCopyCode : BOOL;   // 若值为 TRUE，后面的实例将转移到副本代码 (在线修改)
END_VAR
```

没有定义返回值。

注意：请注意可以使用“exit”方法及其导致的执行顺序。

用户自定义输入：

初始化方法中可以声明附加的输入，并且必须在功能块实例的声明中对这些输入赋值。

功能块“serialdevice”的初始化方法举例：

```
METHOD fb_init : BOOL
VAR_INPUT
   bInitRetains : BOOL; (* retains 的初始化 *)
   bInCopyCode : BOOL;   (* 转移到副本代码的实例 *)
   nCOMnum : INT;          (* 附加的输入：监听的 COM 接口的数量*)
END_VAR
```

功能块“serialdevice”的声明：

COM1 : serialdevice(nCOMnum:=1);

COM0 : serialdevice(nCOMnum:=0);

FB_reinit

如果把一个名为“FB_reinit”的方法声明成一个功能块实例，那么将会在复制这个实例时调用此方法（例如，在功能块声明修改后，又发生了在线改变）。这个方法将使由副本代码创建的新的实例模块重新初始化。由于复制之后原实例的数据将写入新的实例，而您却可能需要原来的初始值，因此重新初始化是很有必要的。FB_reinit 方法没有输入。注意，与FB_init 相比，这个方法必须被显式地声明。如果想要实现基本功能块的重初始化，POU 必须显式调用 FB_reinit

继承条件下的调用顺序请查看：FB_exit, 执行顺序

3.1.25 FB_Exit 方法

“FB_Exit”方法是一个用于功能块的特殊方法。使用这个方法必须进行明确的声明，而没有隐含的声明。如果已经声明，那么将在新的下装之前对所有功能块的声明实例调用此方法。也可能在在线修改时对所有实例的移动和删除调用此退出方法。

退出方法的界面：只有一个强制参数：

METHOD fb_exit : BOOL

VAR_INPUT

bInCopyCode : BOOL; // if TRUE, the exit method is called for exiting an instance that is copied afterwards (online change).如果值为 TRUE，将调用退出方法来退出后面所复制的实例（在线修改）。

END_VAR

请参见“FB_init”方法，并注意以下执行顺序：

退出方法：退出旧的实例：old_inst.fb_exit(bInCopyCode := TRUE);

初始化方法：初始化新的实例：new_inst.fb_init(bInitRetains := FALSE, bInCopyCode := TRUE);

复制功能块的值（复制代码）：copy_fub(&old_inst, &new_inst);

除此之外，在继承情况下是下列的调用顺序（此列表中使用的 POU 假设为："SubFB EXTENDS MainFB" 和 "SubSubFB EXTENDS SubFB"）

bSubSubFb.FB_Exit(...);

fbSubFb.FB_Exit(...);

fbMainFb.FB_Exit(...);

fbMainFb.FB_Init(...);

fbSubFb.FB_Init(...);

fbSubSubFb.FB_Init(...);

für FB_reinit:

fbMainFb.FB_reinit(...);

fbSubFb.FB_reinit(...);

fbSubSubFb.FB_Init(...);

3.1.26 pragma 命令

Pragma 介绍

通过 pragma 命令可以改变一个或几个变量的字段，而这些字段影响着编译和预编译过程。这就意味着 pragma 命令可以影响代码的生成。例如，它可以决定是否对某个变量进行初始化和监控、是否将其加入参数列表或符号表设置，也可以决定是否令其在库管理器中可见。用户可以令系统在编译过程中输出信息，也可以选择条件 pragmas。这些条件 pragma 定义了如何根据各种特定条件来处理变量。用户还可以把这些 pragma 作为“定义”输入到特定对象的编译属性中。

pragma 命令可以单独成行，也可以与其后的参数一起写在实现或声明编辑器的代码行中。在 FBD/LD/IL 编辑器下使用命令“Insert label”,然后使用 pragma 在文本区替换缺省的文本“标签：”。如果希望想使用 pragma 设置标签，先插入 pragma，后面紧跟该标签。

pragma 命令位于花括号中，不区分字母的大小写：

{ <指令文本> }

命令语句可以紧接于变量名之后。其中左、右括号必须在同一行。

根据 pragma 的类型和内容的不同，pragma 的作用范围可以仅限于自己所在的代码行，也可以作用到由另一个适当的 pragma 结束的地方，或是到由不同的参数执行这个 pragma，甚至还可以一直作用到文件结束。此处所说的“文件”可以指一个声明部分、实现部分、全局变量列表或者类型声明。

如果编译器无法正确地解释命令文本，就会把整个 pragma 作为一个注释。此时会提出警告信息。

请看以下的 pragma 类型：

信息 Pragma

字段 Pragma

条件 Pragma

信息 Pragma

信息 Pragma 可以使信息窗口在编译工程时输出信息。

可以把 pragma 命令插入 POU 的文本编辑器中的某条代码行之内，也可以单独列为一行。编译工程时会将此命令编译到您插入的位置。关于这些请参见“difined (identifier)”条件 Pragma 描述提供的例子。

有四种信息 pragma：

Pragma	信息类型
{text 字符串文本}	文本：显示指定的文本。
{info 字符串文本}	消息：显示指定的文本。
{warning digit 字符串文本}	警告：显示指定的文本。尚未实现以数字1至4显示警告级别的功能。与“全局类型数据”的 Obsolete Pragma 不同的是，这一警告是在局部范围内明确定义的。
{error 字符串文本}	错误：显示指定的文本。

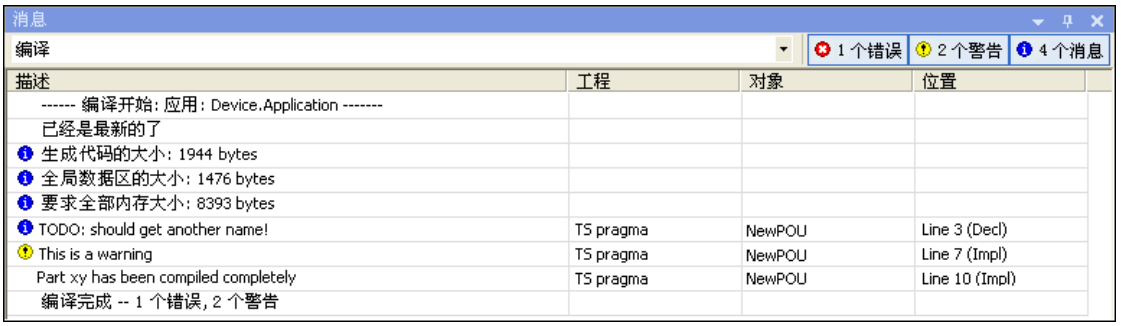
注意：可以通过“下一条信息”命令跳转到造成消息、警告或错误类型信息的源代码源位置（例如当 pragma 位于 POU 中时），但不能对文本类型 Pragma 进行此操作。

例如：

ST 编辑器中的声明和实现：

```
VAR
ivar : INT; {info 'TODO: should get another name'}
bvar : BOOL;
arrTest : ARRAY [0..10] OF INT;
i : INT;
END_VAR
arrTest[i] := arrTest[i]+1;
ivar:=ivar+1;
{warning 'This is a warning'}
{text 'Part xy has been compiled completely'}
```

信息窗口输出：



字段 Pragma

用户自定义属性

任何用户（或应用）定义的 pragmas 属性都可以指定给 POU、类型声明或者变量。编译前，可以通过条件 pragma 测试这些属性。

语法：{attribute '属性'}

通常这条 pragma 指令对于当前代码行有效。如果该指令单独成行，那么它对后续的代码行有效。请点击这里查看条件 pragma 的使用方法。

可以把用户定义属性指定给下列对象：

POU，动作　　例如

为函数 fun1 指定“vision”属性：

```
{attribute 'vision'}
FUNCTION fun1 : INT
VAR_INPUT
i : INT;
END_VAR
VAR
END_VAR
```

变量　　例如

对变量 ivar 添加“DoCount”属性：

```
PROGRAM PLC_PRG
VAR
ivar:INT; {attribute 'DoCount'};
bvar:BOOL;
END_VAR
```

类型　　例如

对数据类型 DUT_1 添加“aType”属性：

```
{attribute 'aType'}
TYPE DUT_1 :
STRUCT
a:INT;
b:BOOL;
END_STRUCT
END_TYPE
```

Displaymode 属性

单个变量的显式模式可以通过 pragma{attribute 'displaymode'}来定义。该设置将覆盖所有监视变量显式模式的全局设置，这通过 显示模式 类中的命令来完成（在在线菜单中默认可用）。

这个编译标识必须位于变量声明行上面的行中。

语法：{attribute 'displaymode':= <displaymode>}

如下定义是可能的：

- 二进制格式显示:

{attribute 'displaymode':='bin'}

{attribute 'displaymode':='binary'}

- 十进制格式显示:

{attribute 'displaymode':='dec'}

{attribute 'displaymode':='decimal'}

- 十六进制格式显示:

{attribute 'displaymode':='hex'}

{attribute 'displaymode':='hexadecimal'}

示例:

```
VAR
    {attribute 'displaymode':='hex'}
    dwVar1: DWORD;
END_VAR
```

扩展属性

在被引用的视图中将数组所有的元素作为输入变量，借助 pragma {attribute 'ExpandFully'}，在视图属性对话框中可以访问数组的所有元素。

语法：{attribute 'ExpandFully'}

例如：在视图 visu_main 的结构中插入视图 visu。

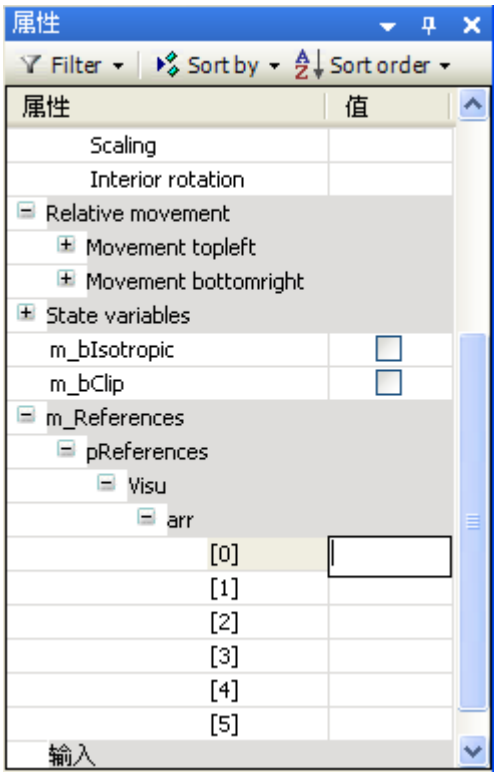
因为输入变量 arr 是在visu 的编辑界面中定义的，所以在visu_main 的视图属性对话框中可以进行赋值。

为了获得属性对话框中数组的元素， 在 visu 的编辑界面中， 必须直接在 arr 的前面添加'ExpandFully'属性。

在 visu 编辑界面中的声明：

```
VAR_INPUT
    {attribute 'ExpandFully'}
arr : ARRAY[0..5] OF INT;
END_VAR
```

视图 visu_main 的结果属性对话框



Global_init_slot 属性

只能对签名使用 pragma {attribute 'global_init_slot'}。默认设置下全局变量列表中变量的初始化顺序是随机的，但某些情况下（例如当一个全局列表的变量与另一个列表中的变量相互关联时）却有必要规定初始化的顺序。此时用户便可以使用 pragma 来实现这一操作。

语法：{attribute 'global_init_slot':= <值>}

上面语句中的“<值>”是一个描述签名初始化顺序的整数值。默认为 50000。数值越小表示越早被初始化。当签名具有相同的 attribute 'global_init_slot'的值时，将无法确定它们初始化的顺序。

例如：

假设工程中包含两个全局变量列表 GVL_1 和 GVL_2：

全局变量“A”属于上面全局变量列表 GVL_1：

{attribute 'global_init_slot':= 300}

VAR_GLOBAL

A : INT:=1000;

END_VAR

GVL_2 中变量“B”和“C”的初始化值取决于变量“A”。

{attribute 'global_init_slot':= 350}

```
VAR_GLOBAL
B : INT:=A+1;
C : INT:=A-1;
END_VAR
```

若将 GVL_1 的 attribute 'global_initial_slot'设为 300（即上例中初始化顺序值的最小值），请确保在初始化“B”之前已经定义表达式“A+1”。

Hide 属性

您可以使用 pragma {attribute 'hide'}来隐藏“列表组件”功能、输入助手和在线模块的声明部分的变量或整个签名。本命令只对 pragma 语句之后的变量有效。

语法：{attribute 'hide'}

使用这个属性隐藏签名的所有局部变量。

例如：将这个属性用于功能块 myPOU：

```
FUNCTION_BLOCK myPOU
VAR_INPUT
a:INT;
{attribute 'hide'}
a_invisible: BOOL;
a_visible: BOOL;
END_VAR
VAR_OUTPUT
b:INT;
END_VAR
VAR
END_VAR
```

在主程序中定义了两个 myPOU 功能块的实例：

```
PROGRAM PLC_PRG
VAR
POU1, POU2: myPOU;
END_VAR
```

将输入值输入POU1 时，列表组件功能会出现文字“POU1”，在 PLC_PRG 的实现部分会显示变量“a”、“a_visible”和“b”，但不会显示隐藏变量“a_invisible”。

Hide_all_locals 属性

您可以使用 pragma {attribute 'hide_all_locals'}在“列表组件”功能和输入助手中隐藏所有签名的本地变量。设置这一属性的操作和对每个局部变量设置“hide”属性相同。

语法：{attribute 'hide_all_locals'}

例如：

将这个属性用于功能块 myPOU：

```
{attribute 'hide_all_locals'}
FUNCTION_BLOCK myPOU
VAR_INPUT
a:INT;
END_VAR
VAR_OUTPUT
b:BOOL;
END_VAR
VAR
c,d:INT;
END_VAR
```

主程序中定义了两个 myPOU 功能块的实例：

```
PROGRAM PLC_PRG
VAR
POU1, POU2: myPOU;
END_VAR
```

将输入值输入 POU1 时，列表组件会出现文字“POU1”，在 PLC_PRG 的实现部分会显示变量“a”和“b”，但不会显示隐藏变量“c”和“d”。

Initialize_on_call 属性

可以向输入变量添加 pragma {attribute 'initialize_on_call'}。添加了此属性的功能块输入将在每次调用功能块时初始化。如果输入原为指针，而此指针在线修改时被删除，那么这个输入将被设为 NULL。

语法：{attribute 'initialize_on_call'}

Init_Namespace 属性

pragma {attribute 'init_namespace'}包含的 STRING 或 WSTRING 类型，将在当前命名空间被初始化，这是该变量所在功能块实例的路径。应用 pragma 需要先对string 类型的变量指定'noinit'属性，以及对相应的功能块指定 'reflection' 属性。

语法：{attribute 'init_namespace 捉

举例：

功能块 POU 提供了所有可能的属性:

{attribute 'reflection'}

FUNCTION_BLOCK POU

VAR_OUTPUT

{attribute 'no_init'}

{attribute 'instance-path'}

myStr: STRING;

END_VAR

在 PLC_PRG 主函数中，定义一个功能块 POU 的实例 fb。:

PROGRAM PLC_PRG

VAR

fb:POU;

newString: STRING;

END_VAR

newString:=fb.myStr;

变量 myStr 在当前命名空间被初始化，如. "PLCWinNT.Application.PLC_PRG.fb". 它的值被赋给主函数中的字符串变量 newString。

Init_On_Onlchange 属性

给变量添加 pragma {attribute 'init_on_onlchange'}，可以使其在每一次在线修改之后都被初始化。

语法：{attribute 'init_on_onlchange'}

Instance-path 属性

向本地字符串变量添加 pragma {attribute 'instance-path'}，可以使这个变量和其所属 POU 的设备树状路径一起初始化。这是一个在处理错误信息时很有用的特性。使用这个 pragma 之前，需要对相应的 POU 使用“reflection”属性，并对字符串变量使用“noinit”属性。

语法：{attribute 'instance-path'}

例如：

设以下功能块已具有'reflection'属性：

{attribute 'reflection'}

FUNCTION_BLOCK POU

VAR

{attribute 'instance-path'}

{attribute 'noinit'}

str: STRING;

END_VAR

在 PLC_PRG 程序中调用功能块实例：

PROGRAM PLC_PRG

VAR

myPOU:POU;

myString: STRING;

END_VAR

myPOU();

myString:=myPOU.str;

myPOU 实例已经初始化，POU 实例 myPOU 的路径被赋给字符串变量“str”（如“PLCWinNT.Application.PLC_PRG.myPOU”）。而这一路径在主程序中被传递给变量“myString”。

提示：字符串变量的长度可能是任意定义的（甚至大于 255），但是请当心如果字符串被分配给一个短数据类型，它可能会被截断（从尾部）。

linkalways 属性

编译指示{attribute 'linkalways'}允许以某种方式为编译器标注 POUs，以使他们总是被包括到编译信息中。结果，具有这个选项的对象将总是被编译并下装到 PLC 上。该选项仅影响位于应用下面的或者被插入到应用下面的库中的 POUs 和 GVLs。编译器选项 Link always 效果一样。

语法：{attribute 'linkalways'}

当使用符号配置编辑器时，已标注的 POUs 被用做符号配置的可选变量的基础。

示例：

全局变量列表 GVLMoreSymbols 是通过使用字段来实现的：

{attribute'linkalways'}

VAR_GLOBAS

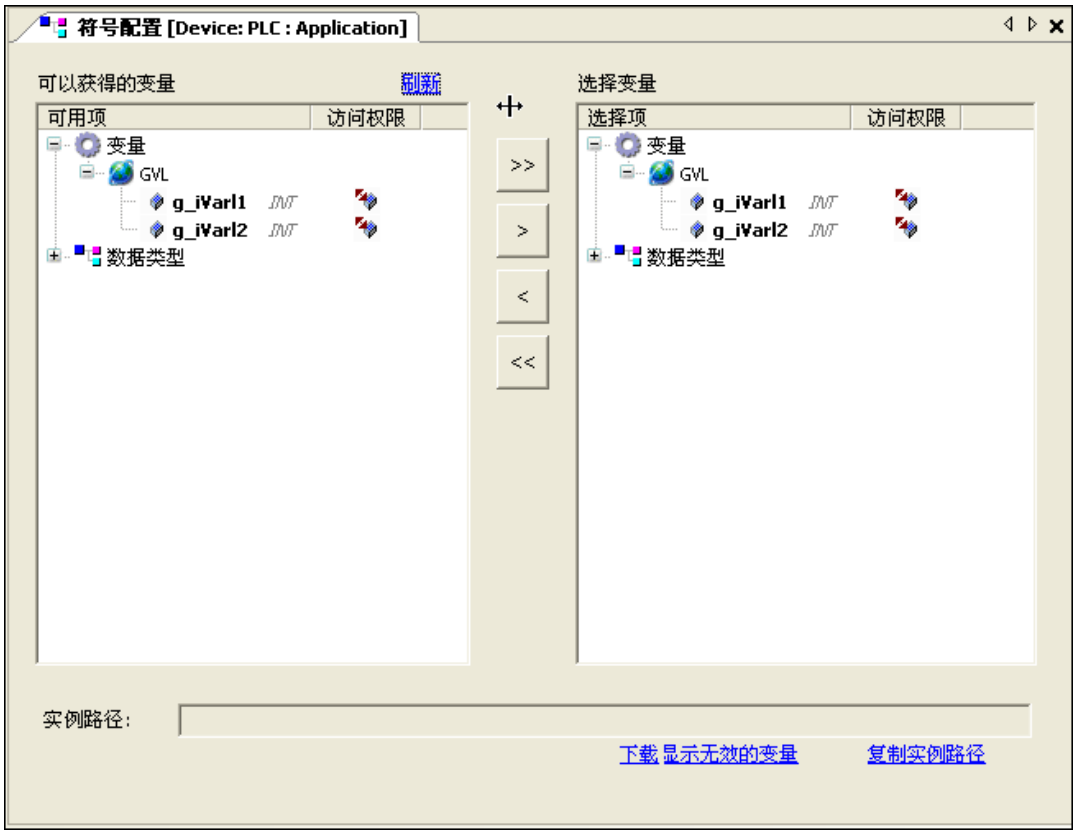
g_iVar1: INT;

g_iVar2: INT;

END_VAR

用以上代码使 GVLMoreSymbols 的符号在符号配置中可选。

符号配置编辑器



call_after_init 属性

使用pragma（属性call_after_init）可以定义一个方法使之在功能块实例初始化之后被调用。为此，功能块本身和要调用的实例方法（性能原因）都必须设置该属性，方法必须在 FB_Init 以及实例声明的初始化表达式被赋值后被调用。

此功能在编译器版本 >= 3.4.1.0.后被支持。

语法：{属性 ' call_after_init'}

示例:

有如下定义:

{属性 'call_after_init'}

FUNCTION_BLOCK FB

... <functionblock definition>

{属性'call_after_init'}

METHOD FB_AfterInit

... <方法声明>

... 有一个这样的声明:

inst : FB := (in1 := 99);

... 会导致下列顺序的代码处理:

inst.FB_Init();

```
inst.in1 := 99;
inst.FB_AfterInit();
```

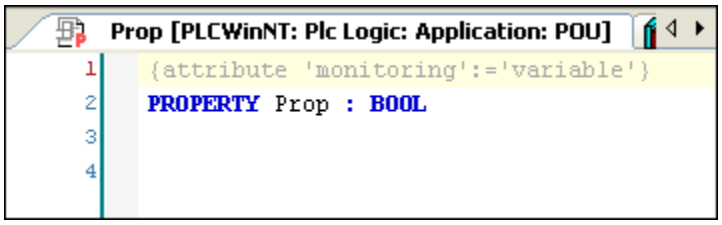
这样在 FB_Afterinit 你可以根据用户的初始化定义进行相应的处理。

Monitoring 属性

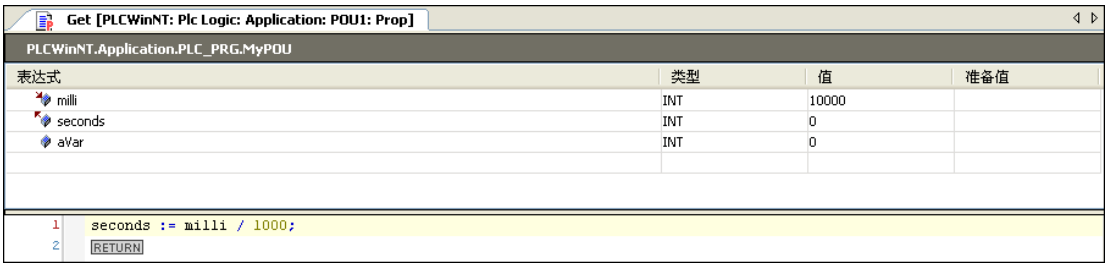
通过行内监控或监视列表可以在在线模式下监控属性。

通过在属性定义上面那一行添加“监视”字段编译指示，可以激活监视。那么在使用该属性的 POU 的在线查看中，或者在一个监视列表中，具有这个属性的变量的名称、类型和值将被显示。在那里也可以输入预定的值，来强制变量具有这个属性。

用于监视变量的属性示例



监视视图示例



有两种方法来得到被显示的属性变量的当前值。在某些使用情况下请仔细考虑，哪种字段编译指示适合于显示期望的值。这取决于在属性内部是否实现了对变量的操作：

1. Pragma '{attribute 'monitoring':='variable'}'

此时将创建一个隐式变量。当应用调用 Set 或 Get 方法时，此变量总是获得当前值。这个隐式变量的值将被监视。

语法:{attribute 'monitoring':='variable'}

2. Pragma '{attribute 'monitoring':='call'}'

该字段仅能用于返回简单数据类型或指针的属性，而不能用于返回结构类型的：

要监视的值可通过调用属性直接得到，例如实时系统调用 Get 方法的监视服务。注意，如果在变量上的操作是在属性内部实现的，值仍有可能被改变！

语法:{attribute 'monitoring':='call'}

No_check 属性

向 POU 中添加 pragma {attribute 'no_check'}可以阻止在本 POU 中调用任何校验函数。由于校验函数可能对性能造成影响，所以在频繁调用校验函数或已经得到验证的POU 中应用这一属性是有

好处的。

语法：{attribute 'no_check'}

No_Copy 属性

在线模式中通常需要重新分配实例（例如 POU），因此系统会复制这些实例中的变量值。

如果向一个变量添加 pragma {attribute 'no_copy'}，那么这个变量的在线修改副本不会运行，相反这个变量将被初始化。如果在线模式中修改了某个变量，这一属性对于指向这个变量的地址的局部指针变量是很有用的。

语法：{attribute 'no_copy'}

No-exit 属性

如果功能块提供了退出方法，那么您就可以通过向功能块实例赋予 pragma {attribute 'no-exit'}来阻止在特定实例中调用这个方法。

语法：{attribute 'symbol' := 'no-exit'}

例如：

假设功能块“POU”中添加了退出方法 FB_Exit。

以程序 PLC_PRG 中两个 POU 类型的变量为例：

```
PROGRAM PLC_PRG
VAR
POU1 : POU;
{attribute 'symbol' := 'no-exit'}
POU2 : POU;
END_VAR
```

当 POU1 中的 bInCopyCode 变量变为 TRUE 时，尽管这个变量值对 POU2 没有影响，但还是将调用退出方法 FB_Exit 来退出实例，事后（在线修改）将会复制这个实例。

Noinit 属性

带有 pragma {attribute 'no_init'}的变量不会被隐含初始化。属于这个变量的 pragma 将在随后声明。

语法：{attribute 'no_init'}

也可以是：

{attribute 'no-init'}

{attribute 'noinit'}

例如：

```
PROGRAM PLC_PRG
VAR
A : INT;
{attribute 'no_init'}
B : INT;
END_VAR
```

如果重置联合的应用程序，那么整数变量“A”将会再次被隐含地初始化为 0，而变量“B”将维持当前的值。

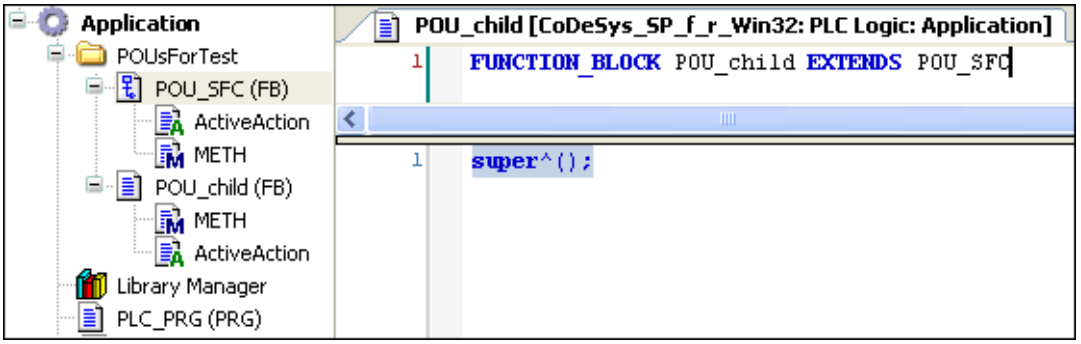
NO_virtual_actions 属性

假设以SFC 实现的由基本功能块派生的功能块，它们使用基类的主SFC 流程。其中被调用的动作显示了与方法相同的虚拟行为，就是说基类动作可通过相关派生类的特定实现来重载。

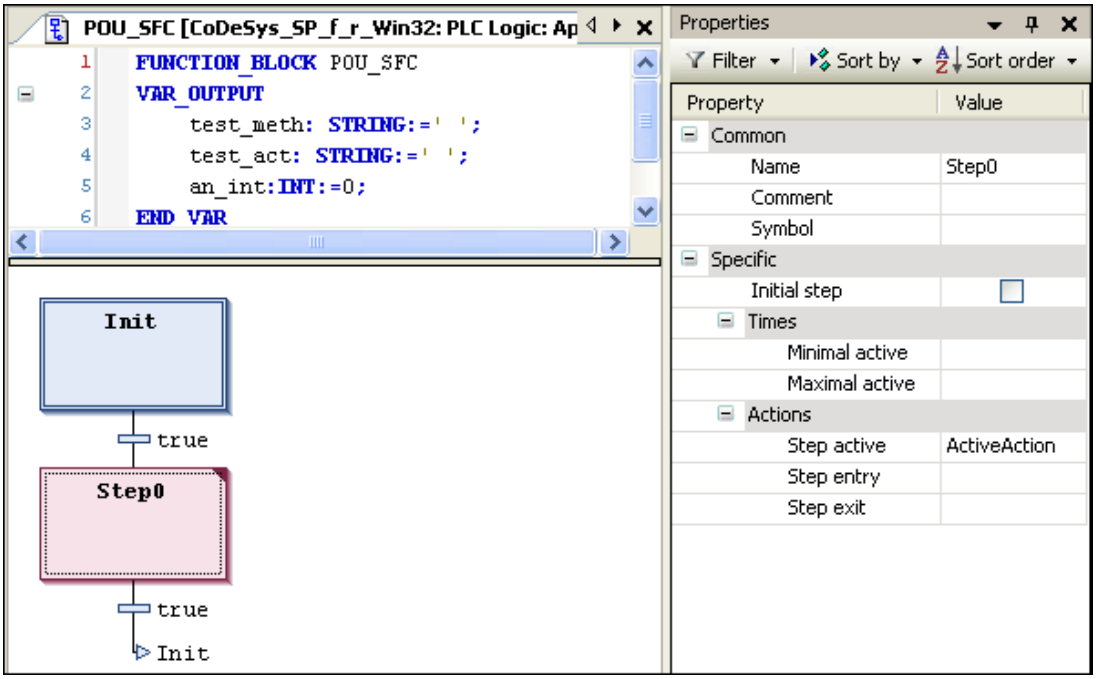
如果基类动作应被防止重载，则基类中应加入 pragma {attribute 'no_virtual_actions'}.

语法：{aattribute 'no_virtual_actions’}

示例：在下面的示例中功能块 POU_SFC 提供了将由功能块 POU_child 继承的基类.



通过使用关键字 SUPER，派生类 POU_child 调用基类用 SFC 实现的流程。



这种流程的典型实现，它具有关联的步动作 ActiveAction（它参与输出变量的赋值），局限于后面是单步执行的初始步骤：

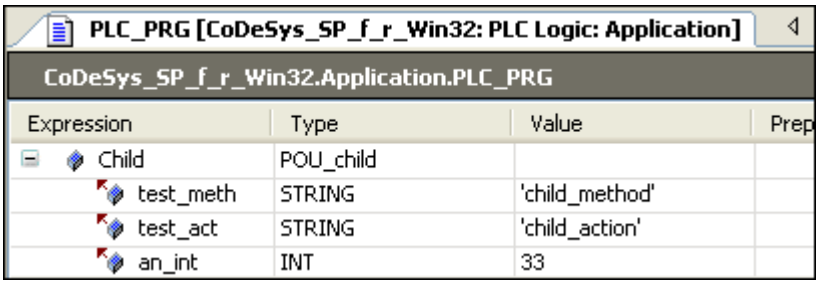
```
an_int:=an_int+1;                    // 整型变量计算动作调用次数
test_act:='father_action';           // 字符串变量 test_act 的赋值
METH();                              // 调用给字符串变量 test_meth 赋值的方法
```

当有派生类 POU_child 时，步动作将被指定的 ActiveAction 的实现重载，它不同于使用字符串“child_action”赋值的原始动作，而用字符串“ather_action”给变量 test_act 赋值。

同样地，方法 METH 在基类内把字符串“father_method”赋给变量 test_meth，它将这样被重载：test_meth 将被赋值为“child_method”。

主程序 PLC_PRG 将执行“Child”的重复调用（一个 POU_child 的实例）。

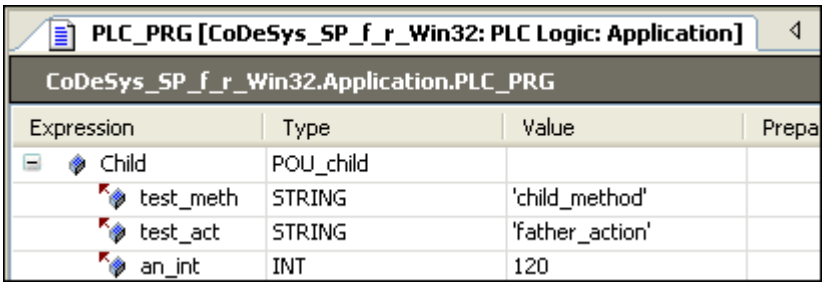
如预料的一样，输出字符串的实际值反映了派生类动作和方法的调用：



但是，如果基类被字段“no_virtual_actions”保护

```
{attribute 'no_virtual_actions'}
FUNCTION_BLOCK POU_SFC...
```

就可观察到不同的行为：与上述相反地，方法 METH 仍将被它在派生类中的实现重载，步动作的调用现在将调用基类的 ActiveAction 动作。因此，test_act 将被赋值为字符串“father_action：



Obsolete 属性

如果各个数据类型（结构、功能块等等）都用于工程内部，那么用户就可以在定义数据类型时添加 "obsolete" pragma，以便在编译时使用用户自定义的警告。比如，用户可以申明以后不得再次使用某种数据类型。

与局部使用的信息 Pragma 不同的是，这个警告是在定义内部定义的，因此对于数据类型的所有实例“全局”有效。

通常这条 pragma 指令度对于当前行或者后续代码行（如果该指令位于一个独立行的话）都有效。

语法：{attribute 'obsolete' := 'user defined text'}

例如：

在定义功能块 fb1 时插入 obsolete pragma：

{attribute 'obsolete' := 'datatype fb1 not valid!'}

FUNCTION_BLOCK fb1

VAR_INPUT

i:INT;

END_VAR

...

如果声明中把 fb1 作为一种数据类型，例如“fbinst:=fb1;”，那么编译这个工程时将出现警告：“datatype fb1 not valid”

Pack_mode 属性

pragma {attribute 'pack_mode'}用于在数据结构重新定位时定义它的封装模式。必须在数据结构的顶部设置这个属性，它影响整个结构的封装。

语法：{attribute 'pack_mode':= '<Value>'}

这条语法中的单引号内的 <Value>必须是下表中的某一个值：

值	封装模式
0	紧密排列。例如，在存储空间中不留间隙
1	1字节排列（与紧密排列相同）
2	2字节排列。例如存储空间的最大间隙为1字节
4	4字节排列，例如存储空间的最大间隙为3字节
8	8字节排列，例如存储空间的最大间隙为7字节。

例如：

```
{attribute 'pack_mode' := '1'}
TYPE myStruct:
STRUCT
  Enable: BOOL;
  Counter: INT;
  MaxSize: BOOL;
  MaxSizeReached: BOOL;
END_STRUCT
END_TYPE
```

myStruct 类型的变量将按照以下例子对齐：例如，如果它的成员 Enable 的地址是 0x0100，那么成员 Counter 的地址就为 0x0101，MaxSize 在 0x0103，而 MaxSizeReached 位于 0x0104。'pack_mode'=2 时，Counter 将位于 0x0102，MaxSize 位于 0x0104，而 MaxSizeReached 则在 0x0105。

注意： 这一属性也可以用于 POU。但由于已经存在 POU 的内部指针，用户必须小心使用。

Parameterstringof 属性

利用 pragma{attribute 'parameterstringof'}，变量实例名可以通过视图变得容易访问。

语法：{attribute 'parameterstringof? := ' <variable>'}

举例：在主函数中使用用户定义的数据结构 DUT 定义一个实例 myDUT:

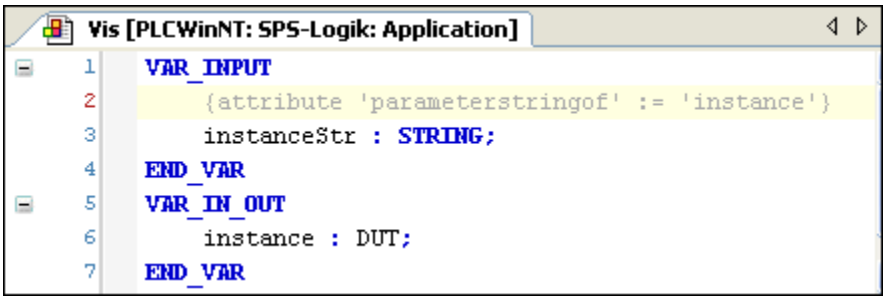
```
PROGRAM PLC_PRG
VAR
myDUT: DUT;
END_VAR
```

这个实例将被传递给视图功能块"Vis" (通过参数 "instance")，该视图功能块又被另一个视图功能块"MainVisu"引用:

"MainVisu"中包含的视图元素的详细属性：

Elementname	GenElemInst_2
Clipping	☐
Show frame	☐
Scaletype	ANISOTROPIC
⊟ References	_3S.CoDeSys.VisualElem.StructuredTypeNode
⊟ References	
⊟ PLCWinNT.Application.Vis	
instance	PLC_PRG.myDut
⊟ Position	
Y	104

当然，在 Vis 相关的界面编辑器中可以找到输入/输出变量"instance",实例，以及名为“instanceStr”的另一个输入变量。:



尽管在引用输入列表中没有列出输入变量“instanceStr”(见上面的图) 。但是，因为变量“instanceStr” 具有 “parameterstringof” 属性， 因此将该属性中涉及的变量名自动赋给“instanceStr”。在本文例子中，属性所涉及变量是一个实例；因此字符串变量“instanceStr”被赋值为"PLC_PRG.myDUT"，现在可以被视图“Vis”使用了，如 text variable defining a placeholder %s 提到的。

Qualified_Only 属性

在一个全局变量列表的顶部写入 pragma {attribute 'qualified_only'}之后，用户就只能通过使用全局变量名访问其中的变量，例如 gvl.g_var。这一功能也可用于枚举类型变量，并可避免与局部变量名称的匹配问题。

语法：{attribute 'qualified_only'}

例如：假设下列全局变量列表 GVL 具有属性“qualified_only”：

{attribute 'qualified_only'}

VAR_GLOBAL

iVar:INT;

END_VAR

在 POU（例如 program PLC_PRG 程序）中，调用这个全局变量必须带有前缀 GVL，例如：

GVL.iVar:=5;

如下不完整的变量调用将导致错误：

iVar:=5;

Reflection 属性

pragma {attribute 'reflection'}用于签名。为了优化性能，POU 必须具有 instance-path 属性。

语法：{attribute 'reflection'}

例如：查看 instance-pat 属性的例子。

Symbol 属性

pragma {attribute 'symbol'}定义了将在图标设置中处理哪些变量，也就是说这些变量将作为“图标”导出到图标列表、工程目录的XML-文件，也可以导出到对用户不可见、但在目标系统中对外部访问有效（如通过 OPC-Server）的文件。即使没有被配置或者在图标设置编辑框中也是不可见的，拥有该属性的变量也会被下装到 PLC。

请注意在设备树的各个应用程序下，必须把图标设置作为一个对象。

语法：

{attribute 'symbol':= 'none'| 'read'| 'write'| 'readwrite'}

请注意只允许访问来自程序或全局变量列表的图标。访问图标时必须完整地写出图标名。

pragma 定义可以指定给某个特定的变量或程序中声明的所有变量：

指定给单个变量时， pragma 应当位于变量声明语句代码行之前。

指定给程序声明部分的所有变量时，pragma应当位于声明编辑器的第一行。但此时仍然可以通过增加 pragma 来改变某个特定变量的设置。

可以通过定义 pragma 参数“none”，“read”、“write”或者“readwrite” 来访问图标。如果没有定义参数，将会使用默认的“readwrite”。

例如：

在下面的设置中，将会以读写形式导出变量 A 和 B，以读形式导出变量 D。

```
{attribute 'symbol' := 'readwrite'}
PROGRAM PLC_PRG
VAR
A : INT;
B : INT;
{attribute 'symbol' := 'none'}
C : INT;
{attribute 'symbol' := 'read'}
D : INT;
END_VAR
```

条件 pragma

ExST （扩展 ST）语言支持几种条件 pragma 命令，这些命令在预编译和编译过程中影响代码的生成。

例如编译中生成实现部分代码的情况可能取决于是否声明了某个特定的数据类型或变量、数据类型或变量是否具有某个特定属性、变量是否具有某个特定的数据类型、某个指定的POU 或任务是否有效或是否是这个调用树的一部分等等。

注意： 对在POUs 树中声明的POU 或 GVL，使用在应用程序中声明了的“{define…}”，是不可能的。在应用程序中的定义将只对插入到相应程序下的接口有效。

{define 标识符字	如果 token 字符串非空（这是允许并明确定义的），那么预处理过程中所有

符串}	标识符的后续实例将会被给定的 token 序列代替。直到对象末尾或被{ undefine }控制命令取消定义，标识符仍然有效。可以参见下文在“条件编译”中使用此命令的例子。
{undefine 标识符} .	取消标识符的预处理器定义（使用{ define}进行预处理定义，参见上文），标识符变为“未定义”。如果此时没有定义指定的标识符， 将忽略这个 pragma。
{IF 表达式} ... {ELSIF 表达式} ... {ELSE} ... {END_IF}	这是**条件**编译选项。在编译时，指定的“表达式”必须为常数；按顺序计算这些表达式的值，然后对非零表达式下的文本进行预处理和编译，并忽略其它部分的代码。必需按照左边的顺序排列各段代码，但 elsif 段和 else 段为可选部分，也可以在没有 else 部分的情况下使用 elseif。 在常数表达式中，可以使用几种“条件编译操作符”，详见下文。
{include 文件名}	这个 pragma 使用完整的*文件名*目录来取代 pragma。用户指定路径时可以使用绝对路径或相对路径。若指定相对路径，将按照如下顺序寻找文件：(1) 工程的工作目录。(2)编程系统设置中定义的标准位置。

条件编译操作符

在条件编译 pragma 的常数表达式（{if}和{elsif}，参见上文）中可以使用几种操作符。这些操作符有时不能通过{undefine}和{define}来取消定义和重新定义。请注意这些表达式和通过{define}定义的表达式一样，可以用于一个对像（“编译”类别）的属性对话框的“编译定义”。

defined (标识符)	用于标识符，如果已经由{define}指令定义了此标识符并且后面没有使用{undefine}取消定义，那么值为 TRUE；否则为 FALSE。
	例如
	预处理：两个应用 App1和 App2，App1中的{define}命令定义了变量“hallo”，而 App2中没有定义此变量。
	{IF defined (hallo)}
	(*应用“App1” 中执行以下代码*)
	hugo := hugo + SINT#1;
	{ELSE}
	(*应用“App2” 中执行以下代码*)
	hugo := hugo - SINT#1;

defined (variable:变量)	用于变量，如果在当前范围内声明了指定的变量，那么值为 TRUE；否则为 FALSE。 例如 预处理：两个应用 App1和 App2，App2中的{define}命令定义了变量“hallo”，而 App1中没有定义此变量。 {IF defined (variable:g_bTest)} (*仅在“App2” 中执行以下代码*) g_bTest := x > 300; {END_IF}

defined (type:标识符)	用于类型标识符，如果声明了指定名称的类型，那么值为 TRUE；否则为 FALSE。 例如 预处理：两个应用 App1和 App2，App1中定义了数据类型“DUT”，而 App2中没有定义此数据类型。 {IF defined (type:DUT)} (*仅在“App1”中执行以下代码*) bDutDefined := TRUE; {END_IF}

defined (pou:pou 名称)	用于 pou 名称，如果定义了指定名称的 POU 或动作，那么值为 TRUE；否则为 FALSE。
	例如
	预处理：两个应用 App1和 App2，App1中有 POU“CheckBounds”，而 App2中没有此 POU。
	{IF defined (pou:CheckBounds)}
	(*仅在“App1”中执行以下代码*)
	arrTest[CheckBounds(0,i,10)] := arrTest[CheckBounds(0,i,10)] + 1;
	{ELSE}
	(*仅在“App2”中执行以下代码*)
	arrTest[i] := arrTest[i]+1;
	{END_IF}

此功能尚未实现：defined (task:标识符)	用于任务标识符，如果定义了指定名称的任务，那么值为 TRUE，否则为 FALSE。
	例如
	预处理：两个应用 App1和 App2，App1中有任务“PLC_PRG_Task”，而 App2中没有此任务。
	{IF defined (task:PLC_PRG_Task)}
	(*仅在“App1”中执行以下代码*)
	erg := plc_prg.x;
	{ELSE}
	(*仅在“App2”中执行以下代码*)
	erg := prog.x;
	{END_IF}

此功能尚未实现：defined (resource:标识符)	用于资源标识符，如果定义了指定名称的资源，那么值为 TRUE；否则为 FALSE。
	例如
	预处理：两个应用 App1和 App2，App1中有资源对象“glob_var1”（全局变量列表），而 App2中没有这个资源对象。
	{IF defined (resource:glob_var1)}
	(*仅在“App1”中执行以下代码*)
	gvar_x := gvar_x + ivar;
	{ELSE}
	(*仅在“App2”中执行以下代码*)
	x := x + ivar;
	{END_IF}

hasattribute (pou: pou 名称, '属性')	用于 POU，如果在 POU 声明部分的第一行指明了指定属性，那么值为 TRUE。 例如 预处理：两个应用 App1和 App2，App1和 App2中均定义了函数“fun1”，但 App1中 fun1还具有属性“vision”： App1中定义 fun1： {attribute 'vision'} FUNCTION fun1 : INT VAR_INPUT i : INT; END_VAR VAR END_VAR App2中定义 fun1： FUNCTION fun1 : INT VAR_INPUT i : INT; END_VAR VAR END_VAR {IF hasattribute (pou: fun1, 'vision')} (*仅在应用“App1”中执行以下代码行*) ergvar := fun1(ivar); {END_IF}
hasattribute (variable: 变量, '属性')	用于变量，如果在变量声明之前的代码行中通过{attribute}命令指明了指定的属性，那么值为 TRUE。 例如 预处理：两个应用 App1和 App2，App1和 App2中均使用了变量“g_globalInt”，但 App1中变量还具有“DoCount”属性： 在 App1中声明 g_globalInt： VAR_GLOBAL {attribute 'DoCount'} g_globalInt : INT; g_multiType : STRING; END_VAR 在 App2中声明 g_globalInt： VAR_GLOBAL g_globalInt : INT; g_multiType : STRING; END_VAR {IF hasattribute (variable: g_globalInt, 'DoCount')} (*由于App1中变量 g_globalInt 具有'DoCount'属性，因此仅在“App1”

	中执行以下代码*) g_globalInt := g_globalInt + 1; {END_IF}
hastype（variable:变量，类型空间)	用于变量，如果指定变量具有指明的 类型空间，那么值为 TRUE；否则为 FALSE。 ANY ANY_DERIVED ANY_ELEMENTARY ANY_MAGNITUDE ANY_BIT ANY_STRING ANY_DATE ANY_NUM ANY_REAL ANY_INT LREAL REAL LINT DINT INT SINT ULINT UDINT UINT USINT TIME LWORD DWORD WORD BYTE BOOL STRING WSTRING DATE_AND_TIME DATE TIME_OF_DAY 例如 预处理：两个应用 App1和 App2，在 App1中以 LREAL 类型声明变量“g_multitype”，在 App2中则声明为 STRING 类型： {IF (hastype (variable: g_multitype, LREAL))} (*仅在“App1”中执行以下代码*) g_multitype := (0.9 + g_multitype) * 1.1; {ELSIF (hastype (variable: g_multitype, STRING))} (*仅在“App2”中执行以下代码*) g_multitype := 'dies ist ein multitalent'; {END_IF}

hasvalue (定义标识符, 字符串)	如果定义了(*定义标识符*)，并且它的值为(*字符串*)，那么值为 TRUE，否则为 FALSE。
	例如
	预处理：应用 App1和 App2中均使用了变量“test”，在 App1中变量的值为“1”，在 App2中为“2”：
	{IF hasvalue(test,'1')}
	(*由于 App1中变量 test 的值为1，因此在“App1”中执行以下代码*)
	x := x + 1;
	{ELSIF hasvalue(test,'2')}
	(*由于 App2中变量 test 的值为2，因此在“App2”中执行以下代行*)
	x := x + 2;
	{END_IF}

NOT 操作符	转化操作符的值。 例如 预处理： 两个应用 App1 和 App2，App1 和 App2 中均使用了 POU“PLC_PRG1”，但只有 App1中有 POU CheckBounds： {IF defined (pou: PLC_PRG1) AND NOT (defined (pou: CheckBounds))} (*仅在“App2” 中执行以下代码*) bANDNotTest := TRUE; {END_IF}
操作符 AND 操作符	如果两个操作符均为 TRUE，那么值为 TRUE。 例如 预处理：App1和 App2中均使用了 POU“PLC_PRG1”，但只有 App1中有 POU CheckBounds： {IF defined (pou: PLC_PRG1) AND (defined (pou: CheckBounds))} (*由于只有 App1中定义了“PLC_PRG1”和“CheckBounds”，因此仅在 App1中执行以下代码*) bORTest := TRUE; {END_IF}
操作符 OR 操作符	如果其中一个操作符为 TRUE，那么值为 TRUE。 例如 预处理：App1和 App2中均使用了 POU“PLC_PRG1”，但只有 App1中有 POU CheckBounds： {IF defined (pou: PLC_PRG1) OR (defined (pou: CheckBounds))} (*由于 App1和 App2至少定义了“PLC_PRG1”和“CheckBounds”中的一个 POU，因此将在 App1和 App2中执行以下代码*) bORTest := TRUE; {END_IF}
(操作符)	给操作符加上括号。

3.1.27 列表组件功能

输入的文本需符合 IEC 61131-3 标准。“IEC61131-3 文本编辑器”插件提供了"列表组件"功能。这个功能可以在工程中通过文本编辑器选项激活或者关闭。

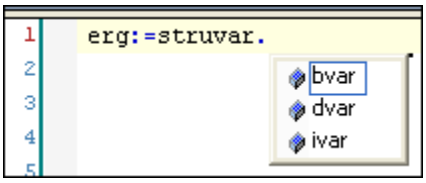
“列表组件”可以帮助您插入正确的标识符：

在任何一个可插入全局变量标识符的地方输入一个句点“.”时，将出现一个列出所有当前可用全局变量的选择框。您可以选择其中一个变量并敲击<回车>键，或者通过双击鼠标将需要的变量插入到句点之后。

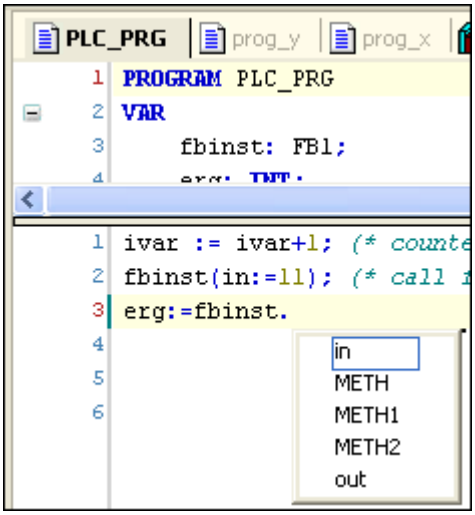
在一个功能块实例或者结构体变量后输入一个句点时，也将出现一个选择框。其中列出了所有相应功能块的输入和输出变量或者结构成员。您可以通过敲击<回车>键或双击鼠标从列表中选择需要的元素。

示例

“列表组件”列出了结构体成员:



“列表组件”列出功能块的元素:



在输入任意字符串后并按下<Ctrl> + <空格>键，将出现一个列出了工程中所有POU 和全局变量的选择框。第一个以输入字符开头的列表项将被选中，敲击<回车>键可以将之输入到程序当中。

3.2 数据类型

编写程序时可以使用标准数据类型和用户定义数据类型。每一个标识符都被指定为一种数据类型，这个数据类型决定了它将占用多大的存储空间以及将存储何种类型的值。

3.2.1 标准数据类型

CoDeSys 支持所有 IEC1131-3 标准中的数据类型。如下所列：

- 布尔类型
- 整数类型
- 实数类型/长实数类型
- 字符串类型
- 双字节字符串类型
- 时间数据类型

请注意存在一些标准的扩展数据类型，所以您也可以自己定义数据类型（用户自定义数据类型）。

布尔类型(布尔类型(bool))

布尔型变量的值可以为 TRUE (1)和 FALSE (0)。占 8 bits 内存空间。

参见：布尔常量（操作数）

整数类型(整数类型(Integer Data Types))

下表列出了所有可用的整数类型变量。不同数字类型的取值范围不同，整数类型的取值范围如下表所示：

类型	数据下限	数据上限	存储空间
BYTE	0	255	8 Bit
WORD	0	65535	16 Bit
DWORD	0	4294967295	32 Bit
LWORD	0	264-1	64 Bit
SINT	-128	127	8 Bit
USINT	0	255	8 Bit
INT	-32768	32767	16 Bit

从较大的数据类型转换为较小的数据类型时，有可能丢失信息。

参见：数字常量（操作数）

实数类型/长实数类型(实数类型/长实数类型(REAL / LREAL))

实数和长实数类型也称为浮点型，用于表示有理数。实数类型数据占用 32 位存储空间，而长实数类型占用 64 位存储空间。

实数类型的取值范围：1.175494351e-38 to 3.402823466e+38

长实数类型的取值范围：2.2250738585072014e-308 to 1.7976931348623158e+308

注意：数据类型 LREAL 的支持取决于目标设备。在编译时 64 位类型的 LREAL 是被转换为 REAL（可能有信息丢失），还是保持不变，请见相应文档。

注意：如果 REAL 或 LREAL 类型转换成 SINT, USINT, INT, UINT, DINT, UDINT, LINT 或 ULINT 类型，且实型数据的值超出了整形的范围，结果将会是不确定的并且该值取决于目标系统。这种情况即使是产生一个异常也是可能的! 为了获取与目标无关的代码，应由应用程序处理所有值域越界问题。如果real/lreal 型数据在整形的值域范围内，他们之间的转换在所有系统上都可以进行。

参见：实数/长实数常量（操作数）

字符串类型(字符串类型(STRING))

字符串类型变量可以包含任意一串字符。声明时的大小决定了存储变量所需要的存储空间。这里的存储空间它是指字符串中字符的数量，用圆括号或方括号括起来。如果没有指名存储空间的大小，则默认为 80 个字符。

CoDeSys 中一般不限制字符串的长度，但是字符串函数只能处理长度在 1-255 个字符之间的字符串！ 如果一个变量用字符串初始化，而字符串对于变量的数据类型来说又太长，字符串将会从右至左相应的截断。

例如，声明一个长度为 35 个字符的字符串：

str:STRING(35):='This is a String';

参见：

双字节字符串

字符串常量（操作数）

时间数据类型(时间数据类型(Time Data Types))

时间数据类型包括 TIME、TIME_OF_DAY（缩写为 TOD）、DATE 和 DATE_AND_TIME（缩写为 DT）。内部处理这些数据的方式与双字（DWORD）类型相似。

TIME 和 TOD 的单位为毫秒，TOD 的初始值是 12:00 A.M.

DATE 和 DT 的单位为秒，DT 的初始值是 1970 年 1 月 1 日 12:00 A.M.

请参见以下文档：

- 长时间类型。在 IEC 61131-3 标准的扩展中这是一个 32 位的时间数据类型。
- 时间常量
- 日期常量
- 日期时间常量
- 时刻常量

3.2.2 IEC1131-3 标准的扩展数据类型

作为对 IEC1131-3 标准中数据类型的补充，CodeSys 隐含一些标准的扩展数据类型：

- 联合
- 任意数据类型
- 长时间类型
- 双字节字符串
- 引用
- 指针

联合(UNION)

作为 IEC 61131-3 标准的扩展，在用户定义数据类型中可以声明联合。

联合中的所有的成员都有相同的偏移，即所有成员都占用同一个存储空间。因此，假设有如下所定义的一个联合，那么对 name.a 所做的赋值操作同样作用于 name.b。

例如：

TYPE name: UNION

a : LREAL;

b : LINT;

END_UNION

END_TYPE

任意数据类型

尚未实现的功能：作为 IEC 61131-3 标准的扩展，可以把函数或者方法的输入声明为任意数据类型。

长时间类型(LTIME)

作为 IEC 61131-3 标准的扩展，提供长时间类型数据作为高精度计时器的时间基量。

语法：LTIME#<时间声明>

时间声明可以包含用于 TIME 类型常量的时间单位以及：

"us" :微秒

"ns" :纳秒

例如：

LTIME1 := LTIME#1000d15h23m12s34ms2us44ns

与 TIME 类型不同的是：TIME 的长度为 32 位且精度为毫秒。

双字节字符串(WSTRING)

这个字符串数据类型是 IEC 61131-3 标准的扩展。

与字符串类型数据（ASCII）不同的是，这一数据类型由 Unicode 解码。

例如：

wstr:WSTRING:='This is a WString';

参见：

字符串

字符串常量 (操作数)

引用(References)

这一数据类型是 IEC 61131-3 标准的扩展。

引用是一个对象的别名。这个别名可以通过标识符读写。与指针不同的是，引用所指向的数据将被直接改变，因此引用的赋值和所指向的数据是相同的。设置引用的地址用一个特定的赋值操作完成。一个引用是否指向一个有效的数据（不等于 0），可以使用一个专门的操作符来检查，如下所示。

用以下语法声明引用：

语法

<标识符>：REFERENCE TO <数据类型>

声明示例：

```
ref_int : REFERENCE TO INT;
a : INT;
b : INT;
```

此时 ref_int 可以作为整型变量的别名使用

使用示例：

```
ref_int REF= a;     (* 此时 ref_int 指向 a *)
ref_int := 12;        (* 此时 a 的值为 12 *)
b := ref_int * 2;    (* 此时 b 的值为 24 *)
ref_int REF= b;     (* 此时 ref_int 指向 b *)
ref_int := a / 2;    (* 此时 b 的值为 6 *)
ref_int REF= 0;    ( *引用的显式初始化 *)
```

注意：不可声明诸如 REFERENCE、ARRAY 和 POINTER 的引用。

注意：当编译器版本>=V3.3.0.0 时，引用将被初始化（为 0）。

有效引用的检查：

操作符“__ISVALIDREF”可用来检查引用是否指向一个不等于 0 的有效值。

语法：

<boolean variable> := __ISVALIDREF(<datatype>声明为 REFERENCE 类型);

如果引用指向一个有效值，则<boolean variable>为真（TRUE），否则为假（FALSE）。

示例：

声明:

```
ivar : INT;
ref_int : REFERENCE TO INT;
ref_int0: REFERENCE TO INT;
testref: BOOL := FALSE;
```

实现:

```
ivar := ivar +1;
ref_int REF= hugo;
ref_int0 REF= 0;
testref := __ISVALIDREF(ref_int);    (* 为 TRUE，因为 ref_int 指向 ivar，它不等于 0 *)
testref0 := __ISVALIDREF(ref_int0);      (* 为 FALSE，因为 ref_int 被设为 0 *)
```

指针(Pointers)

作为 IEC 61131-3 标准的扩展，您可以使用指针。

指针用来在应用程序运行时存储变量、程序、功能块、方法和函数的地址。它可以指向上述的任何一个对象以及任意数据类型，包括用户定义数据类型。请注意，您可以使用隐含的指针校验功能。

声明指针的语法如下：<标识符>: POINTER TO <数据类型 | 功能块 | 程序 | 方法 | 函数>;

取指针地址内容即意味着读取指针当前所指地址中存储的数据。通过在指针标识符后添加内容操作符“^”，可以取得指针所指地址的内容： 请看下面“pt^”的使用示例。

通过地址操作符 ADR 可以将变量的地址赋给指针。

示例：

```
VAR
   pt:POINTER TO INT;   (* 声明一个指针 pt *)
   var_int1:INT := 5;   (* 声明变量 var_int1 和 var_int2 *)
   var_int2:INT;
END_VAR
pt := ADR(var_int1); (* 将 varint1 的地址赋给指针 pt *)
var_int2:= pt^;          (* 通过取指针 pt 的地址内容，将 var_int1 的值 5 赋给 var_int2 *)
```

函数指针

与CoDeSys V2.3 不同的是，本版本用函数指针取代了INDEXOF 操作符。这类指针可以指向外部库，但不可以在编程系统的应用程序内部调用函数指针！注册回调函数（系统库函数）的运行时函数需要函数指针，之后根据注册所需的回调函数的不同，各自的函数将由运行时系统隐式调用（例如 STOP）。若需要使能这样的系统调用（运行时系统），必须为函数对象设置各自的属性 (类别"编译")。

ADR 运算符可以用于函数名、程序名、功能块名和方法名。由于函数在在线修改之后有可能被移

动，因此运算结果不是函数的地址，而是指向函数的指针的地址。只要函数存在，这一地址就将一直有效。

指针的索引访问

作为 IEC 61131-3 标准的扩展，允许对POINTER、STRING 和WSTRING 等类型的变量进行索引访问“[]”。

pint[i]返回被指向的数据类型。

指针的索引访问的算法：如果对指针变量进行索引访问的操作，则偏移量将按照下式计算：pint[i] = (pint + i * 被指向的数据类型的长度)^。索引访问也隐含了获取指针地址内容的操作。所得结果的数据类型是指针所指的数据类型。请注意：pint[7] != (pint + 7)^!

如果对 STRING 类型的变量进行索引访问的操作，则会得到索引表达式偏移量处的字符。所得结果是 BYTE 类型。str[i]将会以 SINT 的形式（ASCII 编码）返回字符串中的第 i 个字符。

如果对 WSTRING 类型变量进行索引访问的操作，则会得到索引表达式偏移量处的字符。所得结果是WORD 类型。wstr[i]将会以 INT 形式（Unicode 编码）返回字符串中的第 i 个字符。

注意：请注意引用与指针不同，它们将直接修改所指的数据。

指针校验函数(CheckPointer function)

为了在程序运行时检查指针的指向，您可以在每次访问指针的地址之前使用隐含的“指针校验”功能。您可以通过添加对象对话框向应用程序中添加“用于隐含检查的 POU”对象。点选指针校验的复选框，选择一种实现语言，确认无误后点击打开。校验功能将会在编辑器中以您选择的语言打开。声明部分是预先设置的，与前述选项的选择无关，并且只有添加了其它局部变量之后才可以更改！与其它校验函数不同的是，没有提供指针校验函数的默认实现，这一部分需要由用户编写！

指针校验函数需要检查指针指向的地址是否在有效的存储范围之类，另外还需要检查引用的连续内存空间与指针所指的变量的数据类型是否匹配。若满足上述两个条件，指针校验应当返回这个输入指针。出现错误时则交由用户进行适当的处理。

模板：

声明部分：

```
// 隐含生成的代码：请勿修改
FUNCTION CheckPointer : POINTER TO BYTE
VAR_INPUT
ptToTest : POINTER TO BYTE;
iSize : DINT;
iGran : DINT;
bWrite: BOOL;
END_VAR
```

实现部分:

//没有标准的实现。请在此处添加您编写的代码

CheckPointer := ptToTest;

调用本函数时需要以下输入参数：

ptToTest：指针的目标地址

iSize：引用的变量的长度；iSize 的数据类型必须与整型兼容且能涵盖存储在指针地址中的可能的最大数据长度。

iGran: 存取的单位长度，如用于被引用变量的最大的未结构化的数据类型；iGran 的数据类型必须与整型兼容。

bWrite：读取方式（TRUE 为写方式，FALSE 为读方式）；bWrite 必须是布尔类型

返回值：取指针地址内容时使用的地址，因此最好是第一个输入参数的地址(ptToTest)。

3.2.3 自定义数据类型

除标准数据类型之外，用户还可以在工程中定义一些特殊的数据类型。

这些数据类型可以通过在 POU 窗口中建立 DUT 对象，或者在 POU 的声明部分进行定义。

请您参考对象的命名建议，以确保其命名尽量不与其它名称重复。

参见下列用户自定义数据类型：

- 数组
- 结构
- 枚举
- 子范围
- 引用
- 指针

数组(数组(Arrays))

一维、二维和三维数组属于基本的数据类型。您可以在POU 的声明部分或者全局变量表中定义数组。请注意可以使用隐含的边界检查。

语法：<数组名>:ARRAY [<ll1>..<ul1>,<ll2>..<ul2>] OF <基本数据类型>

ll1, ll2, ll3 表示字段范围的最小值，ul1, ul2 and ul3 表示字段范围的最大值。字段范围必须是整数。

例如：Card_game: ARRAY [1..13, 1..4] OF INT;

数组的初始化：

注意：与 CoDeSys V2.3 不同的是，在本版本中初始化部分必须加上方括号！

数组的完全初始化举例：

arr1 : ARRAY [1..5] OF INT := [1,2,3,4,5];

arr2 : ARRAY [1..2,3..4] OF INT := [1,3(7)]; (* 即 1,7,7,7 的缩写形式 *)

arr3 : ARRAY [1..2,2..3,3..4] OF INT := [2(0),4(4),2,3];

(* 即 0,0,4,4,4,4,2,3 的缩写形式 *)

结构数组的初始化举例：

结构定义：

```
TYPE STRUCT1
STRUCT
  p1:int;
  p2:int;
  p3:dword;
END_STRUCT
END_TYPE
```

数组初始化：

```
ARRAY[1..3] OF STRUCT1:= [(p1:=1,p2:=10,p3:=4723),(p1:=2,p2:=0,p3:=299),
(p1:=14,p2:=5,p3:=112)];
```

数组的部分初始化举例：

```
arr1 : ARRAY [1..10] OF INT := [1,2];
```

对于那些没有预先赋值的元素，按照基本数据类型的缺省初始值进行初始化。在上例中，元素[3]到[10]被初始化为 0。

数组元素的访问：

在二维数组中访问数组元素，使用下面的语法：

<数组名>[Index1,Index2]

例如:Card_game [9,2]

校验验函数(Check Functions)

为了能够在程序运行时正确地访问数组元素，必须使用边界校验（CheckBounds）函数。您可以通过添加对象对话框向应用程序中添加“用于隐含检查的 POU”对象。点选边界校验的复选框，选择一种实现语言，确认无误后点击打开。边界校验函数将在编辑器中以您选择的语言打开。声明部分是预先设置的，与前述选项的选择无关，并且只有添加了其它局部变量之后才可以更改！可以用 ST 语言修改该函数。

此校验函数会以适当的方式处理越界错误（例如设置错误标记或改变索引值）。每当向数组类型变量赋值时，本函数就会自动运行。

注意：为了保证校验函数有效，请勿改变隐含的校验函数的声明部分！

校验函数使用举例：

默认的校验函数如下所示：

声明部分：

```
// 隐含生成的代码：请勿修改
FUNCTION CheckBounds : DINT
VAR_INPUT
index, lower, upper: DINT;
END_VAR
```

实现部分:

```
// 隐含生成的代码：仅作为使用建议
IF    index < lower THEN
CheckBounds := lower;
ELSIF    index > upper THEN
CheckBounds := upper;
ELSE
CheckBounds := index;
END_IF
```

调用本函数时需要以下输入参数：

index: 数组元素的索引

lower: 字段范围的最小值

upper: 字段范围的最大值

如果索引在字段范围之内，返回值就是索引本身。否则将根据越界的情况返回字段范围的上限或者下限值。

以下程序中超出了数组“a”的上限值：

```
PROGRAM PLC_PRG
VAR
  a: ARRAY[0..7] OF BOOL;
  b: INT:=10;
END_VAR
a[b]:=TRUE;
```

在上面的程序中，隐含调用的 CheckBounds 函数把索引值从“10”修改为上限值“7”，随后布尔量 TRUE 被赋给数组元素 a[7]。这就是 CheckBounds 函数修正超出范围的访问的过程。

结构(Structures)

用户可以通过添加对象对话框建立“DUT”（数据类型单元）来创建一个结构。

声明结构使用关键字 TYPE 和 STRUCT 开始，并以 END_STRUCT 和 END_TYPE 结束。

注意：与 CoDeSys V2.3 不同的是，结构声明中的 TYPE 后必须添加一个“:”！

结构声明的语法如下：

```
TYPE <结构名>:
STRUCT
 <变量的声明 1>
 .
 .
 <变量声明 n>
END_STRUCT
END_TYPE
```

<结构名>是一种可以在整个工程中被识别的数据类型，可以像标准数据类型一样使用。

您可以使用互锁结构。唯一的限制是不可以指定结构中变量的地址（不允许使用 AT 来指定变量的地址）

定义名为 Polygonline 的结构举例：

```
TYPE Polygonline:
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
```

结构的初始化：

结构初始化举例：

```
Poly_1:polygonline := ( Start:=[3,3], Point1 =[5,2], Point2:=[7,3], Point3:=[8,5], Point4:=[5,7], End := [3,5]);
```

不能以变量作为初始化的值。参见“数组”中结构数组的初始化举例。

结构成员的访问：

您可以使用下述的语法访问结构的成员：

<结构名>.<结构成员名>

因此对于上述的结构“polygonline”，您可以用 Poly_1.Start 访问结构成员“start”。

枚举(Enumerations)

枚举是由很多字符串常量组成的用户定义数据类型。这些常量称为枚举值。

即使在 POU 之内声明枚举值，枚举值仍然可以在整个工程范围内被识别。

用户可以通过添加对象对话框建立“DUT”对象来创建一个枚举。

注意：与 CoDeSys V2.3 不同的是，必须使用“TYPE”才能进行局部枚举声明。

语法：

TYPE <标识符>:(<Enum_0> ,<Enum_1>, ...,<Enum_n>) |<base data type>;
END_TYPE

<标识符>变量可以代表枚举值 <Enum_..> 并被初始化为第一个值。这些值与所有的数字类型兼容，也就是说可以像使用整形变量一样对其进行操作。您也可以把数 x 赋给这个值。如果在声明中枚举值没有被初始化为指定的值，将从 0 开始递增依次进行初始化。请确保初始化时的初始值在内部元素中递增。运行时将会检查这些值的有效性。

例如：

TYPE TRAFFIC_SIGNAL: (red, yellow, green:=10); (* 每个颜色的初始值是 Red=0，Yellow=1，Green=10 *)

END_TYPE

TRAFFIC_SIGNAL1 : TRAFFIC_SIGNAL;

TRAFFIC_SIGNAL1:=0; (* 交通信号的值是 Red *)

FOR i:= red TO green DO

i := i + 1;

END_FOR;

IEC 61131-3 标准的扩展：

枚举类型的名称可以（作为范围操作符）用来确定需要访问的枚举常量。

因此，您可以在不同枚举中使用相同的常量。

例如：

定义两个枚举：

TYPE COLORS_1: (red, blue);
END_TYPE

TYPE COLORS_2: (green, blue, yellow);

END_TYPE

在 POU 中使用枚举值 Blue：

声明：

colorvar1 : COLORS_1;

colorvar2 : COLORS_2;

实现：

(* 可以这样使用：*)

colorvar1 := colors_1.blue;

colorvar2 := colors_2.blue;

(* 不能这样使用：*)

colorvar1 := blue;

colorvar2 := blue;

可以明确指定枚举所指向的数据类型（默认类型是 INT）。

例如：

需要将枚举 BigEnum 的数据类型指定为 DINT:

TYPE BigEnum : (yellow, blue, green:=16#8000) DINT;

END_TYPE

子范围(子范围(Subrange types))

子范围是一种用户自定义类型，其值的范围是基本数据类型的一个子集。注意可以使用隐含的范围边界校验（Range boundary checks）。

在 DUT 中声明子范围。您也可以把一个变量直接声明为子范围：

声明为一个 DUT 对象的语法如下:

TYPE <名称> : <Inttype> (<ug>..<og>) END_TYPE;

<名称>	必须是一个有效的 IEC 标识符。
<Inttype>	是以下数据类型中的一个：SINT、USINT、INT、UINT、DINT、UDINT、BYTE、WORD、DWORD (LINT、ULINT、LWORD)。
<ug>	是一个与基本数据类型相兼容的常量，它设定了范围数据类型的下界。下界本身也属于这个范围。
<og>	是一个与基本数据类型相兼容的常量，它设定了范围数据类型的上界。上界本身也属于这个范围。

例如：

TYPE

SubInt : INT (-4095..4095);

```
END_TYPE
```

把变量直接声明为子范围：

```
VAR
 i : INT (-4095..4095);
 ui : UINT (0..10000);
END_VAR
```

如果一个（在声明或实现中）被指定为子范围的值不在子范围的界限之内（如令以上声明中的 i:=5000），系统将报错。

校验函数（Check Functions）

为了在运行中检查值的范围，应用中必须使用 CheckRangeSigned 或 CheckRangeUnsigned 函数。您可以通过添加对象对话框向应用程序中添加“用于隐含检查的 POU”对象。点选 CheckRangeSigned 或 CheckRangeUnsigned 的复选框，选择一种实现语言，确认无误后点击打开。所选的函数将在编辑器中以您选择的语言打开。声明部分是预先设置的，与前述选项的选择无关，并且只有添加了其它局部变量之后才可以更改！可以用 ST 语言修改该函数。

设置这个校验函数的目的是为了合理处置子范围的错误（例如设置错误标记或改变值）。每当为子范围类型变量赋值时，本函数就会自动运行。

注意：为了保证校验函数有效，请勿改变隐含的校验函数的声明部分！

举例：对一个属于有符号数子范围的变量赋值时，就已自动调用了 CheckRangeSigned 函数。本函数的默认实现将把数值调整到允许的范围之内，如下所示：

声明部分：

```
//隐含生成的代码：请勿修改
FUNCTION CheckRangeSigned : DINT
VAR_INPUT
value, lower, upper: DINT;
END_VAR
```

实现部分：

```
// 隐含生成的代码：仅作为使用建议
IF (value < lower) THEN
CheckRangeSigned := lower;
ELSIF(value > upper) THEN
CheckRangeSigned := upper;
ELSE
CheckRangeSigned := value;
```

```
END_IF
```

调用本函数时需要以下输入参数：

value：指定给范围类型数据的数值

lower：范围的下限

upper：范围的上限

如果指定的值在范围之内，那么函数就输出这个值。否则将根据超出范围的情况返回范围的上限或者下限值。

赋值 i:=10*y 将会自动被替换为

```
i := CheckRangeSigned(10*y, -4095, 4095);
```

例如，如果y 的值为 1000，那么在默认实现中将不会使变量 i 为“10*1000=10000”，而是会令 i 等于范围的上限值，如“4095”

使用 CheckRangeUnsigned 函数的情况与此相同。

注意：如果不使用CheckRangeSigned 函数和CheckRangeUnsigned 函数，那么在运行时将不会对子范围进行任何类型的检查！在这种情况下，变量 i 随时有可能变为介于-32768 和 32767 之间的任何值。

注意：CheckRangeSigned 函数和CheckRangeUnsigned 函数的使用有可能导致死循环。例如，当您把子范围作为 FOR 循环的增量，而这一循环的终止条件在子范围的允许取值之外时。

导致死循环的例子：

```
VAR
  ui : UINT (0..10000);
END_VAR

FOR ui:=0 TO 10000 DO
...
END_FOR
```

由于校验函数使变量 ui 的值不会超过 10000，故程序永远无法跳出这个 FOR 循环。

引用(References)

这一数据类型是 IEC 61131-3 标准的扩展。

引用是一个对象的别名。这个别名可以通过标识符读写。与指针不同的是，引用所指向的数据将被直接改变，因此引用的赋值和所指向的数据是相同的。设置引用的地址用一个特定的赋值操作完成。一个引用是否指向一个有效的数据（不等于 0），可以使用一个专门的操作符来检查，如下所示。

用以下语法声明引用：

语法

<标识符> : REFERENCE TO <数据类型>

声明示例：

```
ref_int : REFERENCE TO INT;
a : INT;
b : INT;
```

此时 ref_int 可以作为整型变量的别名使用

使用示例：

```
ref_int REF= a;     (* 此时 ref_int 指向 a *)
ref_int := 12;        (* 此时 a 的值为 12 *)
b := ref_int * 2;   (* 此时 b 的值为 24 *)
ref_int REF= b;     (* 此时 ref_int 指向 b *)
ref_int := a / 2;   (* 此时 b 的值为 6 *)
ref_int REF= 0;   ( *引用的显式初始化 *)
```

注意：不可声明诸如 REFERENCE、ARRAY 和 POINTER 的引用。

注意：当编译器版本>=V3.3.0.0 时，引用将被初始化（为 0）。

有效引用的检查：

操作符“__ISVALIDREF”可用来检查引用是否指向一个不等于 0 的有效值。

语法：

<boolean variable> := __ISVALIDREF(<datatype>声明为 REFERENCE 类型);

如果引用指向一个有效值，则<boolean variable>为真（TRUE），否则为假（FALSE）。

示例：

声明:

```
ivar : INT;
ref_int : REFERENCE TO INT;
ref_int0: REFERENCE TO INT;
testref: BOOL := FALSE;
```

实现:

```
ivar := ivar +1;
ref_int REF= hugo;
ref_int0 REF= 0;
```

testref := __ISVALIDREF(ref_int); (* 为 TRUE，因为 ref_int 指向 ivar，它不等于 0 *)

testref0 := __ISVALIDREF(ref_int0); (* 为 FALSE，因为 ref_int 被设为 0 *)

指针(Pointers)

作为 IEC 61131-3 标准的扩展，您可以使用指针。

指针用来在应用程序运行时存储变量、程序、功能块、方法和函数的地址。它可以指向上述的任何一个对象以及任意数据类型，包括用户定义数据类型。请注意，您可以使用隐含的指针校验功能。

声明指针的语法如下:

<标识符>: POINTER TO <数据类型 | 功能块 | 程序 | 方法 | 函数>;

取指针地址内容即意味着读取指针当前所指地址中存储的数据。通过在指针标识符后添加内容操作符“^”，可以取得指针所指地址的内容： 请看下面“pt^”的使用示例。

通过地址操作符 ADR 可以将变量的地址赋给指针。

示例：

```
VAR
   pt:POINTER TO INT;  (* 声明一个指针 pt *)
   var_int1:INT := 5;  (* 声明变量 var_int1 和 var_int2 *)
   var_int2:INT;
END_VAR
pt := ADR(var_int1); (* 将 varint1 的地址赋给指针 pt *)
var_int2:= pt^;          (* 通过取指针 pt 的地址内容，将 var_int1 的值 5 赋给 var_int2 *)
```

函数指针

与CoDeSys V2.3 不同的是，本版本用函数指针取代了INDEXOF 操作符。这类指针可以指向外部库，但不可以在编程系统的应用程序内部调用函数指针！注册回调函数（系统库函数）的运行时函数需要函数指针，之后根据注册所需的回调函数的不同，各自的函数将由运行时系统隐式调用（例如 STOP）。若需要使能这样的系统调用（运行时系统），必须为函数对象设置各自的属性 (类别"编译")。

ADR 运算符可以用于函数名、程序名、功能块名和方法名。由于函数在在线修改之后有可能被移动，因此运算结果不是函数的地址，而是指向函数的指针的地址。只要函数存在，这一地址就将一直有效。

指针的索引访问

作为 IEC 61131-3 标准的扩展，允许对POINTER、STRING 和WSTRING 等类型的变量进行索引访问“[]”。

pint[i]返回被指向的数据类型。

指针的索引访问的算法：如果对指针变量进行索引访问的操作，则偏移量将按照下式计算：pint[i] = (pint + i * 被指向的数据类型的长度)^。索引访问也隐含了获取指针地址内容的操作。所得结果的数据类型是指针所指的数据类型。请注意：pint[7] != (pint + 7)^!

如果对 STRING 类型的变量进行索引访问的操作，则会得到索引表达式偏移量处的字符。所得结果是 BYTE 类型。str[i]将会以 SINT 的形式（ASCII 编码）返回字符串中的第 i 个字符。

如果对 WSTRING 类型变量进行索引访问的操作，则会得到索引表达式偏移量处的字符。所得结果是WORD 类型。wstr[i]将会以 INT 形式（Unicode 编码）返回字符串中的第 i 个字符。

注意：请注意引用与指针不同，它们将直接修改所指的数据。

指针校验函数(CheckPointer function)

为了在程序运行时检查指针的指向，您可以在每次访问指针的地址之前使用隐含的“指针校验”功能。您可以通过添加对象对话框向应用程序中添加“用于隐含检查的 POU”对象。点选指针校验的复选框，选择一种实现语言，确认无误后点击打开。校验功能将会在编辑器中以您选择的语言打开。声明部分是预先设置的，与前述选项的选择无关，并且只有添加了其它局部变量之后才可以更改！与其它校验函数不同的是，没有提供指针校验函数的默认实现，这一部分需要由用户编写！

指针校验函数需要检查指针指向的地址是否在有效的存储范围之类，另外还需要检查引用的连续内存空间与指针所指的变量的数据类型是否匹配。若满足上述两个条件，指针校验应当返回这个输入指针。出现错误时则交由用户进行适当的处理。

模板：

声明部分：

```
// 隐含生成的代码：请勿修改
FUNCTION CheckPointer : POINTER TO BYTE
VAR_INPUT
ptToTest : POINTER TO BYTE;
iSize : DINT;
iGran : DINT;
bWrite: BOOL;
END_VAR
```

实现部分:

```
//没有标准的实现。请在此处添加您编写的代码
CheckPointer := ptToTest;
```

调用本函数时需要以下输入参数：

ptToTest：指针的目标地址

iSize：引用的变量的长度；iSize 的数据类型必须与整型兼容且能涵盖存储在指针地址中的可能的

最大数据长度。

iGran: 存取的单位长度，如用于被引用变量的最大的未结构化的数据类型；iGran 的数据类型必须与整型兼容。

bWrite：读取方式（TRUE 为写方式，FALSE 为读方式）；bWrite 必须是布尔类型

返回值：取指针地址内容时使用的地址，因此最好是第一个输入参数的地址(ptToTest)。

3.3 操作符

3.3.1 IEC 操作符和标准扩展函数

CoDeSys V3.x 支持所有 IEC 操作符。与标准函数不同的是，系统将自动识别工程中的此类操作符。

除 IEC 操作符之外，还可以使用下列 IEC 标准未规定的操作符： ANDN、ORN、XORN、INDEXOF 和 SIZEOF（参见算术操作符）、ADR、BITADR 和内容运算符（参见地址操作符）、以及一些“范围操作符”。

使用操作符的方法和在 POU 中使用函数的方法相似。

注意：使用浮点类型变量进行运算的结果取决于当前使用的目标系统！

请看下列各类操作符：

- 赋值操作符：:=, MOVE
- 算术操作符
- 位字符串操作符
- 移位操作符
- 选择操作符
- 比较操作符
- 地址操作符
- 调用操作符
- 类型转换操作符
- IEC 扩展操作符
- 数学函数
- IEC 扩展“范围操作符”

3.3.2 算术操作符

可以使用下列 IEC1131-3 标准规定的操作符：ADD、MUL、SUB、DIV、MOD、MOVE

还可以使用两种普通扩展操作符：INDEXOF、SIZEOF

加法

IEC 操作符：变量相加

允许的变量类型： BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL 和 LREAL。

可以将两个时间类型变量相加，得到另一个时间类型变量。
（例如：t#45s + t#50s = t#1m35s）

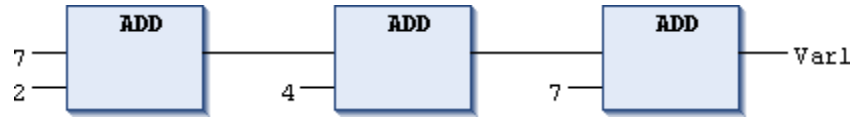
IL 举例：

```
LD    7
ADD   2
ADD   4
ADD   7
ST    iVar
```

ST 举例：

var1 := 7+2+4+7;

FBD 举例：



乘法

IEC 操作符：变量相乘。

允许的变量类型：BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL 和 LREAL。

IL 举例：

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

ST 举例：

var1 := 7*2*4*7;

FBD 举例：

减法

IEC 操作符： 从某个变量中减去一个变量。

允许的变量类型：BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL 和 LREAL。

可以从某个时间变量中减去一个时间变量，结果仍为时间变量。但是请您注意，此处不支持负的时间值。

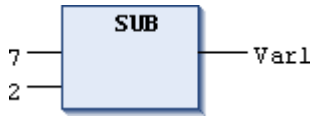
IL 举例：

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

ST 举例：

var1 := 7-2;

FBD 举例：



除法

IEC 操作符：用一个变量除另一个变量。

允许的变量类型：BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL 和 LREAL。

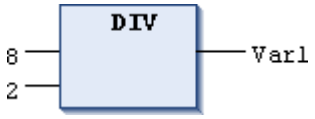
IL 举例：（Var 的结果为 4）

```
LD    8
DIV   2
ST    Var1
```

ST 举例：

var1 := 8/2;

FBD 举例：



注意：请注意在不同的目标系统下，用零除变量可能得到不同的结果！

校验验函数

如果需要检查除数的值（例如为了避免除数为零）您可以使用系统提供的校验函数 CheckDivInt、CheckDivLint、CheckDivReal 以及 CheckDivLReal。一旦把这些函数包含到应用中，每一个进行除法的地方都会预先调用它们。用户可以通过添加对象对话框来添加这些函数。您只需在对话框中选择“用于隐含效验的 POU”，然后点选相应校验函数的复选框，并选择一种实现语言，确认无误后点击“打开”。您选择的函数就会在编辑器中以所选实现语言的形式打开。函数的声明部分是预先设置的，与前述选项的选择无关，并且只有在添加局部变量后才可以更改。可以用 ST 语言修改该函数。

请看下面实现部分中 CheckDivReal 函数的例子：

CheckDivReal 函数的默认实现部分：

声明部分：

//隐含生成的代码：请勿修改

```
FUNCTION CheckDivReal : REAL
VAR_INPUT
  divisor:REAL;
END_VAR
```

实现部分：

//隐含生成的代码：仅作为使用建议

```
IF divisor = 0 THEN
CheckDivReal:=1;
ELSE
  CheckDivReal:=divisor;
END_IF;
```

DIV 操作符将 CheckDivReal 函数的输出作为除数。下面的例子中止了一个以 0 为除数的除法。也就是说在 CheckDivReal 函数对除法的预先干预下，除数“d”的初始值从 0 变成了“1”。因此，除法运算结果为 799。

```
PROGRAM PLC_PRG
VAR
  erg:REAL;
v1:REAL:=799;
d:REAL;
END_VAR
```

```
erg:= v1 / d;
```

取余

IEC 操作符：一个变量与另一个变量相除取余。

允许的变量类型：BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT。结果为除法运算的余数，是一个整数。

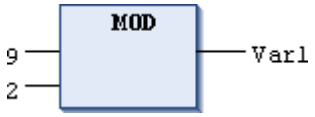
IL 举例：(Var1 的结果为 1)

```
LD    9
MOD   2
ST    Var1
```

ST 举例：

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

FBD 举例：



赋值

IEC 操作符：将一个变量的值赋给另一个适当类型的变量。

在图形编辑器 FBD、LD、CFC 中，MOVE 是一个方框。在这个方框里（未锁定的）EN/ENO 功能也可以用于变量赋值。

在 CFC 中的连接处使用 EN/ENO 功能的例子：

只有当 en_i 的值为 TRUE 时，var1 的值才能赋给 var2。

IL 举例：(结果：var2 得到 var1 的值)

```
LD      var1
MOVE
ST      var2
```

(!您同样可以使用：

```
LD      var1
ST      var2
```

)

ST 举例：

ivar2 := MOVE(ivar1);

(！您同样可以使用 ivar2 := ivar1)

字节长度

IEC 61131-3 标准中没有规定此算术操作符。

这个操作符用来确定给定变量 x 需要占用多少个字节。

SIZEOF 操作符通常返回一个无符号数。返回值的类型与变量 x 的大小相匹配。

SIZEOF(x)的返回值	返回值的类型
0 <= x 的值 < 256	USINT
256 <= x 的值 < 65536	UINT
65536 <= x 的值 < 4294967296	UDINT
4294967296 <= x 的值	ULINT

ST 举例：

var1 := SIZEOF(arr1); (* var1:=USINT#10; *)

IL 举例： (结果为 10)

arr1:ARRAY[0..4] OF INT;

Var1:INT;

```
LD      arr1
SIZEOF
ST      Var1
```

索引

IEC 61131-3 标准中没有规定此算术操作符。

此函数用来查找 POU 的内部索引。

ST 举例：

var1 := INDEXOF(POU2);

3.3.3 位操作符

根据 IEC1131-3 标准，可以使用下列位字符串操作符：AND、OR、XOR、NOT

尚未实现的功能：标准中没有规定的操作符：ANDN、ORN、XORN

字符串操作符比较两个或多个操作数的相应二进制位。

与

IEC 位字符串操作符：对位操作数按位进行与运算。若每个输入位都为 1，那么结果位也为 1，否则为 0。

允许的数据类型：BOOL、BYTE、WORD、DWORD。

IL 举例：（Var1 的结果为 2#1000_0010）

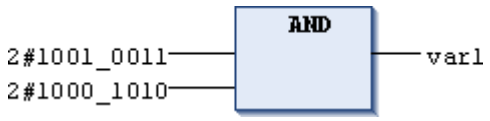
Var1:BYTE;

```
LD      2#1001_0011
AND     2#1000_1010
ST      var1
```

ST 举例：

var1 := 2#1001_0011 AND 2#1000_1010

FBD 举例：



与非

IEC 61131-3 标准中没有规定此 IEC 位字符串操作符。

此功能尚未实现。

或

IEC 位字符串操作符：对位操作数按位进行或运算。若至少有一个输位为 1，结果就为 1，否则为 0。

允许的数据类型: BOOL、BYTE、WORD 和 DWORD。

IL 举例：(var1 的结果为 2#1001_1011)

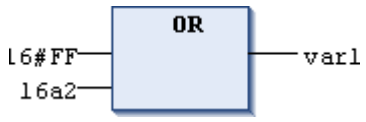
var1 :BYTE;

```
LD      16#FF
OR      16a2
ST      var1
```

ST 举例：

Var1 := 2#1001_0011 OR 2#1000_1010

FBD 举例：



或非

IEC 61131-3 标准中没有规定此 IEC 位字符串操作符。

此功能尚未实现。

异或

IEC 位字符串操作符：对位操作数按位进行异或运算。如果只有一个输入位为 1，那么结果也为 1；如果两个输入都为 1 或者都为 0，那么结果位为 0。

允许的数据类型：BOOL、BYTE、WORD、DWORD。

注意：请注意XOR 函数的扩展运算方式，即超过两个输入的情况。这些输入将两两进行运算，然后再将所得结果两两运算（这样的运算方式符合标准的规定，但可能与用户所预计的情况不同）。

IL 举例：(结果为 2#0001_1001)
Var1 :BYTE;

```
LD      2#1001_0011
XOR     2#1000_1010
ST      var1
```

ST 举例：

Var1 := 2#1001_0011 XOR 2#1000_1010

FBD 举例:

异或非

IEC 61131-3 标准中没有规定这个 IEC 位字符串操作符。

此功能尚未实现。

非

IEC 位字符串操作符：将位操作数按位进行非运算。如果相应输入位为 0，那么结果为 1，反之亦然。

允许的数据类型：BOOL, BYTE, WORD, DWORD。

IL 举例：（Var1 的结果为 2#0110_1100）

Var1 :BYTE;

```
LD      2#1001_0011
NOT
ST      varl
```

ST 举例：

Var1 := NOT 2#1001_0011

FBD 举例：

3.3.4 移位操作符

根据 IEC1131-e 标准，可以使用下列移位操作符：SHL、SHR、ROL、ROR

左移

IEC 操作符： 将操作数按位左移。

erg:= SHL (in, n)

in：需要左移的操作数。

n：操作数左移的位数。

如果 n 超出了数据本身的位数，BYTE、WORD 和 DWORD 类型的操作数将会补 0，而有符号类型的操作数（例如 INT）将会进行算术移位。也就是说会将这些数的最高位的值补在空出的二进制位上。

注意：请注意，算术运算的总的位数受到输入变量的数据类型的影响！如果输入变量是常量，将会取长度最小的数据类型。而输出变量的数据类型则不会对算术运算产生影响。

注意：请看下面十六进制数运算的例子，尽管 byte 和 word 形式的输入变量值相等，但根据输入变量的数据类型的不同（BYTE 或 WORD），erg_byte 和 erg_word 将得到不同的结果。

ST 举例：

```
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); (* 结果为 16#14 *)

erg_word:=SHL(in_word;n); (* 结果为 16#0114 *)
```

FBD 举例：

IL 举例：

```
LD    in_byte
SHL   2
ST    erg_byte
```

右移

IEC 操作符： 将操作数按位右移。

erg:= SHR (in, n)

in: 需要右移的操作数。

n: 操作数右移的位数。

如果 n 超出了数据本身的位数，BYTE、WORD 和 DWORD 类型的操作数将会补 0，而有符号类型的操作数（例如 INT）将会进行算术移位。也就是说会将这些数的最高位的值补在空出的二进制位上。

注意：请注意，算术运算的总的位数受到输入变量的数据类型的影响！如果输入变量是常量，将会取长度最小的数据类型。而输出变量的数据类型则不会对算术运算产生影响。

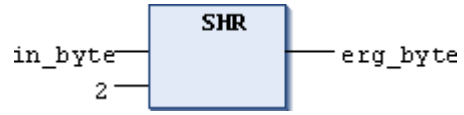
请看下面十六进制数运算的例子。请注意这里算术运算的结果取决于输入变量的类型（BYTE 或 WORD）。

ST 举例：

```
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); (* 结果为 11 *)

erg_word:=SHR(in_word;n); (* 结果为 0011 *)
```

FBD 举例：



IL 举例：

```
LD      in_byte
SHR     2
ST      erg_byte
```

循环左移

IEC 操作符：将操作数按位循环左移。

erg:= ROL (in, n)

允许的数据类型：BYTE、WORD、DWORD。

in 将会左移二进制位 n 次，同时左边移出的位重新补充到右边。

注意：请注意，算术运算的总的位数受到输入变量的数据类型的影响！如果输入变量是常量，将取长度的最小的数据类型。输出变量的数据类型则不会对算术运算产生影响。

请看下面十六进制数运算的例子，尽管 byte 和 word 形式的输入变量值相等，但根据输入变量数据类型的不同（BYTE 或 WORD），erg_byte 和 erg_word 将得到不同的结果。

ST 举例：

```
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); (* 结果为 16#15 *)

erg_word:=ROL(in_word,n); (* 结果为 16#0114 *)
```

FBD 举例：

IL 举例：

```
LD      in_byte
ROL     n
ST      erg_byte
```

循环右移

IEC 操作符：将操作数按位循环右移。

erg = ROR (in, n)

允许的数据类型：BYTE、WORD、DWORD。

in 将会右移二进制位 n 次，同时右边移出的位将会重新补充到左边。

注意：请注意，算术运算的总的位数受到输入变量数据类型的影响！如果输入变量是常量，将取长度最小的数据类型。输出变量的数据类型则不会对算术运算产生影响。

请看下面十六进制数运算的例子，尽管 byte 和 word 形式的输入变量值相等，但根据输入变量的数据类型的不同（BYTE 或 WORD），erg_byte 和 erg_word 将得到不同的结果。

ST 举例：

```
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); (* 结果为 16#51 *)

erg_word:=ROR(in_word;n); (* 结果为 16#4011 *)
```

FBD 举例：

IL 举例：

```
LD      in_byte
ROR     n
ST      erg_byte
```

3.3.5 选择操作符

所有选择操作符后均可带有变量。为了表述清晰，仅介绍以下使用常量操作符的情况： SEL、MAX、MIN、LIMIT、MUX

二选一

IEC 选择操作符： 从两个操作数中选择一个。由 G 决定 IN0 还是 IN1 为输出。

OUT := SEL(G, IN0, IN1) 的含义：

OUT := IN0; 若 G=FALSE

OUT := IN1; 若 G=TRUE.

允许的数据类型：

IN0, IN1, OUT: 任意类型

G: BOOL.

IL 举例：

```
LD   TRUE
SEL 3,4     (* IN0 = 3, IN1 =4 *)
ST  Var1    (* 结果为 4 *)
LD   FALSE
SEL 3,4
ST  Var1    (* 结果为 3 *)
```

ST 举例：

```
Var1:=SEL(TRUE,3,4); (* 结果为 4 *)
```

FBD 举例：

注意：请注意如果 IN0 的值为 TRUE，那么不会处理在 IN1 或者 IN2 之前的表达式。

取最大值

IEC 选择操作符：取最大值函数。返回两个值中较大的那一个。

OUT := MAX(IN0, IN1)

IN0, IN1 和 OUT 可以为任意类型的变量。

IL 举例：(结果为 90)

```
LD    90
MAX   30
MAX   40
MAX   77
ST    Var1
```

ST 举例：

```
Var1:=MAX(30,40); (* 结果为 40 *)
Var1:=MAX(40,MAX(90,30)); (* 结果为 90 *)
```

FBD 举例：

取最小值

IEC 选择操作符：取最小值函数。返回两个值中较小的那一个。

OUT := MIN(IN0, IN1)

IN0, IN1 和 OUT 可以是任意类型的变量。

IL 举例：(结果为 30)

```
LD      90
MIN     30
MIN     40
MIN     77
ST      Var1
```

ST 举例：

Var1:=MIN(90,30); (* 结果为 30 *);

Var1:=MIN(MIN(90,30),40); (* 结果为 30 *);

FBD 举例：

取极限值

IEC 选择操作符：取极限。

OUT := LIMIT(Min, IN, Max)即为：

OUT := MIN (MAX (IN, Min), Max)

Max 是结果的上限值，Min 是结果的下限值。如果 IN 值大于上限值 Max，LIMIT 将返回 Max，而如果 IN 小于 Min，那么结果为 Min。

IN 和 OUT 可以是任意类型的变量。.

IL 举例：(结果为 80)

```
LD      90
LIMIT   30
        80
ST      Var1
```

ST 举例

Var1:=LIMIT(30,90,80); (* 结果为 80 *);

多选一

IEC 选择操作符：多项选择操作符。

OUT := MUX(K, IN0,...,INn)即为：

OUT := INK.

IN0、...,INn 以及 OUT 可以是任意类型的变量。K 必须为 BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT 或 UDINT 类型。MUX 从这一组值中选择第 K 个值。

IL 举例：(结果为 30)

```
LD      0
MUX     30  ,
        40  ,
        50  ,
        60  ,
        70  ,
        80
ST      Var1
```

ST 举例：

Var1:=MUX(0,30,40,50,60,70,80); (* 结果为 30 *);

注意：请注意，除了 INK 之外，不会为了缩短运行时间而处理输入量之前的表达式！仅在仿真模式下才会执行所有的表达式。

3.3.6 比较操作符

根据 IEC1131-3 标准，可以使用下列操作符：GT、LT、LE、GE、EQ、NE

这些是布尔操作符，用来比较两个输入（第一和第二个操作数）的大小。

大于

IEC 比较操作符 ：大于

这是一个布尔量操作符。当第一个操作数比第二个大时，返回值为 TRUE。操作数可以是任何基本数据类型。

IL 举例：(结果为 FALSE)

```
LD      20
GT      30
ST      Var1
```

ST 举例：

VAR1 := 20 > 30 > 40 > 50 > 60 > 70;

FBD 举例：

小于

IEC 比较操作符：小于

这是一个布尔量操作符。当第一个操作数比第二个小时，返回值为 TRUE。操作数可以是任何基本数据类型。

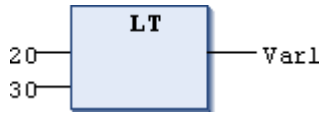
IL 举例：(结果为 TRUE)

```
LD    20
LT    30
ST    Var1
```

ST 举例：

VAR1 := 20 < 30;

FBD 举例：



小于等于

IEC 比较操作符 ：小于或等于

这是一个布尔量操作符。当第一个操作数小于或者等于第二个操作数时，返回值为 TRUE。操作数可以是任何基本数据类型。

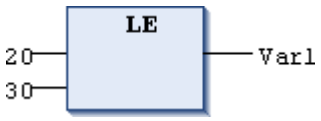
IL 举例：(结果为 TRUE)

```
LD    20
LE    30
ST    Var1
```

ST 举例：

VAR1 := 20 <= 30;

FBD 举例：



大于等于

IEC 比较操作符 ：大于或等于

这是一个布尔量操作符。当第一个操作数大于或者等于第二个操作数时，返回值为 TRUE。操作数可以是任何基本数据类型。

IL 举例：(结果为 TRUE)

```
LD    60
GE    40
ST    Var1
```

ST 举例：

VAR1 := 60 >= 40;

FBD 举例：

等于

IEC 比较操作符 ：相等

这是一个布尔量操作符。当两个操作数相等时，返回值为 TRUE。操作数可以是任何基本数据类型。

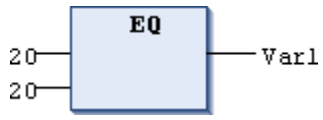
IL 举例：(结果为 TRUE)

```
LD    40
EQ    40
ST    Var1
```

ST 举例：

VAR1 := 40 = 40;

FBD 举例：



不等于

IEC 比较操作符 ：不相等。

这是一个布尔量操作符。当两个操作数不相等时，返回值为 TRUE。操作数可以是任何基本数据类型。

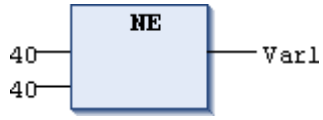
IL 举例：(结果为 FALSE)

```
LD    40
NE    40
ST    Var1
```

ST 举例：

VAR1 := 40 <> 40;

FBD 举例：



3.3.7 地址操作符

ADR 和 BITADR 以及内容运算符 "^"是 CoDeSys 中有效的标准扩展地址运算符。

取地址

IEC 61131-3 标准中没有规定这个地址操作符。

ADR 返回变量自身的地址，数据类型为 DWORD。这个地址可以作为指针传递给操作函数，也可以赋给工程内的某个指针。

注意：与 CoDeSys V2.3 不同的是，现在版本中的ADR 操作符可以与函数名、程序名、功能块名和方法名一起使用，因此它可以取代 INDEXOF 操作符。请参见“函数指针”。但需要注意的是，函数指针可以指向外部库，但不能在 CoDeSys 内部调用函数指针！若需要使用系统调用（系统运行时），您必须为函数对象设置各自的属性（位于“编译”菜单）。

ST 举例：

```
dwVar:=ADR(bVAR);
```

IL 举例：

```
LD      bVar
ADR
ST      dwVar
```

注意：在线修改之后，某些地址中的数据可能会被改变。请在使用地址指针时务必注意。

取位地址

IEC 操作符： 地址函数，IEC 61131-3 标准中没有规定此操作符。

BITADR 的返回值为 DWORD 中的位偏移量。请注意偏移量的值取决于目标设置中的字节编址是否被激活。

```
VAR
  var1 AT %IX2.3:BOOL;
  bitoffset: DWORD;
END_VAR
```

ST 举例：

```
bitoffset:=BITADR(var1); (* 如果字节编址＝TRUE，则结果为19，如果字节编址为FALSE，则结果为 35 *)
```

IL 举例：

```
LD      Varl
BITADR
ST      bitoffset
```

注意：在线修改之后，某些地址中的数据可能会被改变。请在使用地址指针时务必注意。

取内容

标准 IEC61131-3.IEC 中没有规定这个地址操作符。在指针标识符之后添加取内容操作符“^”可以取得指针的内容。

ST 举例：

```
pt:POINTER TO INT;
var_int1:INT;
var_int2:INT;
pt := ADR(var_int1);
var_int2:=pt^;
```

注意：在线修改之后，某些地址中的数据可能会被改变。请在使用地址指针时务必注意。

3.3.8 调用操作符

IEC 操作符：调用一个功能块或程序。

IL 中使用 CAL 来调用一个功能块实例。作为输入的变量位于功能块实例名称之后的括号里。

例如：

从功能块中调用 Inst 实例，这个功能块的输入变量 Par1 和 Par2 分别为 0 和 TRUE。

```
CAL INST(PAR1 := 0, PAR2 := TRUE)
```

3.3.9 类型转换操作符

不允许将“较大的”数据类型隐含地转换为“较小的”数据类型（例如隐含地把 INT 转换为 BYTE 或把 DINT 转换为WORD）。用户想要进行此类转换时需要特别指名。一般来说，可以从任意一种基本类型转换为另外任意一种基本类型。

语法：<基本数据类型 1>_TO_<基本数据类型 2>

请注意...TO_STRING 的转换中，字符串是从左边开始生成的。如果定义的字符串长度小于原来的的长度，右边部分会被截去。

可以进行以下的类型转换：

- 布尔类型转换
- 转换为布尔类型
- 整数类型间转换
- 实数/长实数类型转换
- 时间/时刻类型转换
- 日期/日期时间类型转换
- 字符串类型转换
- 新增部分与 2.3 版本的对照
- TRUNC (转为 DINT)
- TRUNC_INT

- ANY NUM_TO_<数字数据类型>
- ANY_TO_<任意数据类型>

布尔类型转换

IEC 操作符：从布尔类型转换为其它任意类型。

布尔类型转换操作符的语法：BOOL_TO_<数据类型>

转换为数字类型时，若操作数为 TRUE，结果为 1；若操作数为 FALSE，结果为 0。

转换为字符串类型时，若操作数为 TRUE，结果为“TRUE”，若操作数为 FALSE，则结果为“FALSE”。

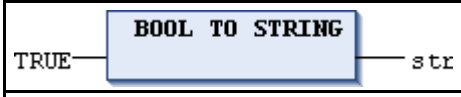
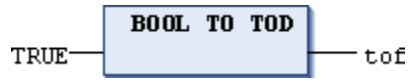
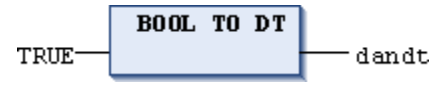
IL 举例：

LD TRUE BOOL_TO_INT ST i	(* 结果为 1 *)
LD TRUE BOOL_TO_STRI... ST str	(* 结果为“TRUE” *)
LD TRUE BOOL_TO_TIME ST t	(* 结果为 T#1ms *)
LD TRUE BOOL_TO_TOD ST tof	(* 结果为 TOD#00:00:00.001 *)
LD FALSE BOOL_TO_DATE ST dandt	(* 结果为 D#1970-01-01 *)
LD TRUE BOOL_TO_DT ST dandt	(* 结果为 DT#1970-01-01-00:00:01 *)

ST 举例：

i:=BOOL_TO_INT(TRUE);	(* 结果为1 *)
str:=BOOL_TO_STRING(TRUE);	(* 结果为“TRUE” *)
t:=BOOL_TO_TIME(TRUE);	(* 结果为 T#1ms *)
tof:=BOOL_TO_TOD(TRUE);	(* 结果为 TOD#00:00:00.001 *)
dat:=BOOL_TO_DATE(FALSE);	(* 结果为 D#1970 *)
dandt:=BOOL_TO_DT(TRUE);	(* 结果为 DT#1970-01-01-00:00:01 *)

FBD 举例：

	(* 结果为1 *)
	(* 结果为“TRUE” *)
	(* 结果为 T#1ms *)
	(* 结果为 TOD#00:00:00.001 *)
	(* 结果为 D#1970-01-01 *)
	(* 结果为 DT#1970-01-01-00:00:01 *)

转换为布尔类型

IEC 操作符：从其它变量类型转换为布尔类型。

转换为布尔类型操作符的语法：<数据类型>_TO_BOOL

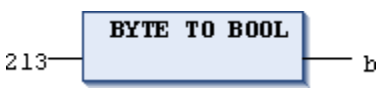
当操作数不为 0 时，结果为 TRUE。当操作数为 0 时，结果为 FALSE。

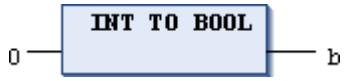
当操作数为“TRUE”，转换为字符串类型变量的结果为 TRUE，否则结果为 FALSE。

IL 举例：

LD 213 BYTE TO BOOL ST b	(* 结果为 TRUE *)
LD 0 INT TO BOOL ST b	(* 结果为 FALSE *)
LD T#5ms TIME TO BOOL ST b	(* 结果为 TRUE *)
LD 'TRUE' STRING TO BOOL ST b	(* 结果为 TRUE *)

FBD 举例：

	(*结果为 TRUE *)

	(* 结果为 FALSE *)
	(* 结果为 TRUE *)
	(* 结果为 TRUE *)

ST 举例：

b := BYTE_TO_BOOL(2#11010101);	(* 结果为 TRUE *)
b := INT_TO_BOOL(0);	(* 结果为 FALSE *)
b := TIME_TO_BOOL(T#5ms);	(* 结果为 TRUE *)
b := STRING_TO_BOOL('TRUE');	(* 结果为 TRUE *)

整数类型之间的转换

从一种整数类型转换为为其它数字类型。

转换操作符的语法：

<INT 数据类型>_TO_<INT 数据类型>

从较大的数据类型转换为较小的数据类型时，有可能丢失部分信息。如果需转换的数超过了数值范围，则忽略这个数的第一字节。

ST 举例：

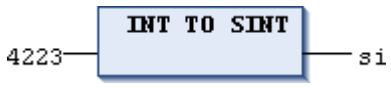
si := INT_TO_SINT(4223); (* 结果为 127 *)

若将整数 4223 保存为 SINT 变量，它将变为 127（十六进制的 16#7f）。

IL 举例：

```
LD          4223
INT TO SINT
ST          si
```

FBD 举例：



实数/长实数类型转换

IEC 操作符：从实数/长实数变量类型转换为其它类型：

数值将被四舍五入为近似的整数值，然后转换成新的变量类型。但转换为 STRING、BOOL、REAL 和 LREAL 类型变量时例外。

注意: 如果 REAL 或 LREAL 类型转换成SINT, USINT, INT, UINT, DINT, UDINT, LINT 或 ULINT 类型，且实型数据的值超出了整形的范围，结果将会是不确定的并且该值取决于目标系统。这种

情况即使产生一个异常也是可能的! 为了获取与目标无关的代码，应由应用程序进行值域越界处理。如果 real/lreal 型数据在整形的值域范围内，他们之间的转换在所有系统上都可以进行。

请注意转换为字符串类型时，（长）实数的位数不能超过 16 位。如果位数太多，那么第十六位将被四舍五入。如果定义的字符串长度比较短，那么数字的右侧部分将被截去。

从较大的数据类型转换为较小的数据类型时，有可能丢失部分信息。

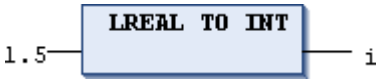
ST 举例：

```
i := REAL_TO_INT(1.5); (* 结果为 2 *)
j := REAL_TO_INT(1.4); (* 结果为 1 *)
i := REAL_TO_INT(-1.5); (* 结果为-2 *)
j := REAL_TO_INT(-1.4); (* 结果为-1 *)
```

IL 举例：

```
LD          2.7
REAL TO INT
ST          i
```

FBD 举例：



时间/时刻类型转换

IEC 操作符：把时间（TIME）或时刻（TIME_OF_DAY）类型变量转换为其他类型变量。

转换操作符的语法：

<时间数据类型>_TO_<其它数据类型>

时间将以毫秒为单位以 DEWOD 类型（TIME_OF_DAY 变量从 12:00 A.M 开始）存储在内部，然后再进行转化。

从较大的数据类型转换为较小的数据类型时，有可能丢失部分信息。

转换为字符串类型变量时，转换结果为时间常量。

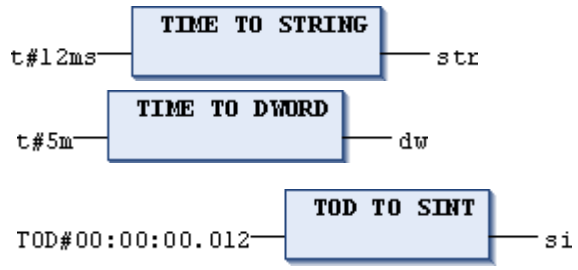
IL 举例：

LD T#12ms TIME TO STTI... ST str	(*结果为“T#12ms”*)
LD T#300000ms TIME TO DWORD ST dw	(*结果为300000 *)
LD TOD#00:00:00.012 TIME TO SINT ST si	(*结果为12 *)

ST 举例：

str :=TIME_TO_STRING(T#12ms);	(* 结果为 T#12ms *)
dw:=TIME_TO_DWORD(T#5m);	(* 结果为300000 *)
si:=TOD_TO_SINT(TOD#00:00:00.012);	(* 结果为12 *)

FBD 举例：



日期/日期时间类型转换

IEC 操作符：从日期和日期时间类型变量转换为其他类型变量。

转换操作符的语法：

日期以秒为单位，用 DWORD 数据类型从 1970 年 1 月 1 日起存储在内部。然后再进行转化。

从较大的数据类型转换为较小的数据类型时，有可能丢失部分信息。

转换为字符串类型变量时，转换结果为日期常量。

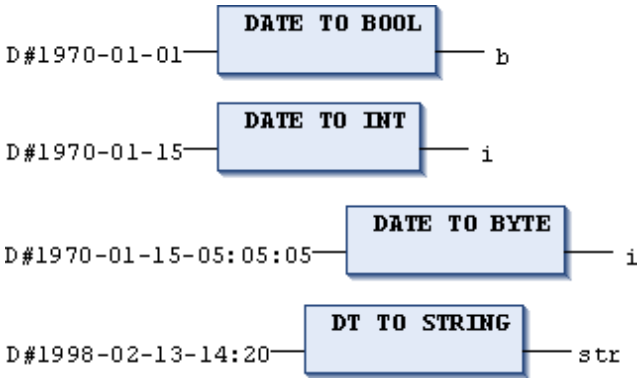
IL 举例：

LD D#1970-01-01 DATE_TO_BOOL ST b	(* 结果为 FALSE *)
LD D#1970-01-01 DATE_TO_INT ST i	(* 结果为 29952 *)
LD D#1970-01-15-05:05:. DATE_TO_BYTE ST byt	(* 结果为 129 *)
LD D#1998-02-13-14:20 DATE_TO_STRI... ST str	(* 结果为“DT#1998-02-13-14:20” *)

ST 举例：

b :=DATE_TO_BOOL(D#1970-01-01);	(* 结果为 FALSE *)
i :=DATE_TO_INT(D#1970-01-15);	(* 结果为29952 *)
byt :=DT_TO_BYTE(DT#1970-01-15-05:05:05);	(* 结果为129 *)
str:=DT_TO_STRING(DT#1998-02-13-14:20);	(* 结果为“DT#1998-02-13-14:20” *)

FBD 举例：



字符串类型转换命令

IEC 操作符：把字符串类型变量转换为其它类型。

转换的过程依据C 语言标准的编译机制进行：先把STRING 转换为 INT 类型变量，然后把 INT 转换为 BYTE 类型。由于高字节将被截去，因此结果将介于 0-255 之间。

由于一条单独的机器指令就可以完成这一转换，因此在绝大多数处理中都可以由这种方法生成最优的代码。

转换操作符的语法：STRING_TO_<数据类型>

STRING 类型变量的操作数中必须包含一个在目标类型变量里有效的值，否则转换的结果为 0。

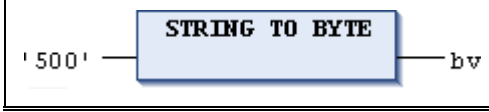
IL 举例：

LD 'TRUE' STRING TO BOOL ST b	(* 结果为 TRUE *)
LD 'abc34' STRING TO WORD ST w	(* 结果为0 *)
LD 't#117ms' STRING TO TIME ST t	(* 结果为 T#117ms *)
LD '500' STRING TO BYTE ST bv	(* 结果为244 *)

ST 举例：

b :=STRING_TO_BOOL('TRUE');	(* 结果为 TRUE *)
w :=STRING_TO_WORD('abc34');	(* 结果为0 *)
t :=STRING_TO_TIME('T#127ms');	(* 结果为 T#127ms *)
bv :=STRING:TO_BYTE('500');	(* 结果为244 *)

FBD 举例：

	(* 结果为 TRUE *)
	(* 结果为0 *)
	(* 结果为 T#127ms *)
	(* 结果为244 *)

取整

IEC 运算符: 转换将 REAL 类型转化为 INT 类型，将使用该值的整数部分。.

注意：TRUNC_INT 对应于 CoDeSys V2.3 版本中的TRUNC，所以当导入工程是 2.3 版本时会自动替换。须注意 v3 版本中转换函数 TRUNC 的变化。

当执行 IL 语言举例:

```
LD      1.9
TRUNC
ST      iVar
```

ST 语言举例：

iVar:=TRUNC_INT(1.9); (* Result is 1 *)

iVar:=TRUNC_INT(-1.4); (* Result is -1 *)

截尾取整

IEC 操作符：把实数类型（REAL）转换成 DINT 类型。取被转换值的整数部分。

注意：在 2.3 版本中TRUNC 运算符将REAL 类型转化为 INT 类型。因此当导入 2.3 版本工程时，TRUNC 将自动被替换为 TRUNC_INT。

当执行一个较大类型向较小类型的转化时，有丢失信息的风险。

IL 举例：

```
LD      1.9
TRUNC
ST      iVar
```

ST 举例：

diVar:=TRUNC(1.9); (* 结果为 1 *)

diVar:=TRUNC(-1.4); (* 结果为-1 *)

任意类型转换

IEC 操作符：把任意类型、或任意数字类型的数据转化为另一种类型。

语法：

ANY_NUM_TO_<数字数据类型>

ANY_TO_<任意数据类型>

举例：

把实数类型的变量转化为整数类型：

re : REAL := 1.234;

i : INT := ANY_TO_INT(re)

3.3.10 数学函数

可以使用下列 IEC 数学运算操作符：ABS、SQRT、LN、LOG、EXP、SIN、COS、TAN、ASIN、ACOS、ATAN、EXPT

绝对值

IEC 操作符：返回一个数的绝对值。例如 ABS(-2)的返回值为 2。

输入输出可以是任何数字基本数据类型。

IL 举例： (结果为 2)

```
LD      -2
ABS
ST      i
```

ST 举例：

i:=ABS(-2);

FBD 举例：

平方根

IEC 操作符：返回一个数的平方根。

输入变量可以是任何数字基本数据类型，输出变量必须是 REAL 或 LREAL 类型。

IL 举例：(q 的结果为 4)

```
LD      16
SQRT
ST      q
```

ST 举例：

q:=SQRT(16);

FBD 举例：

自然对数

IEC 操作符：返回一个数的自然对数。

输入变量可以是任何数字基本数据类型，输出变量必须是 REAL 或 LREAL 类型。

IL 举例：(q 的结果为 3.80666)

```
LD    45
LN
ST    q
```

ST 举例：

q:=LN(45);

FBD 举例：

常用对数

IEC 操作符：返回值是以 10 为底的对数。

输入变量可以是任何数字基本数据类型，输出变量必须是 REAL 或 LREAL 类型。

IL 举例：(q 的结果为 2.49762 *)

```
LD    314.5
LOG
ST    q
```

ST 举例：

q:=LOG(314.5);

FBD 举例：

指数

IEC 操作符：返回指数函数。

输入变量可以是任何数字基本数据类型，输出变量必须是 REAL 或 LREAL 类型。

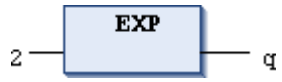
IL 举例：(q 的结果为 7.389056099)

```
LD    2
EXP
ST    q
```

ST 举例：

q:=EXP(2);

FBD 举例：



正弦

IEC 操作符：返回一个数的正弦值。

输入变量可以是任何数字基本数据类型，输出变量必须是 REAL 或 LREAL 类型。

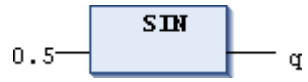
IL 举例：(q 的结果为 0.479426 *)

```
LD    0.5
SIN
ST    q
```

ST 举例：

q:=SIN(0.5);

FBD 举例：



余弦

IEC 操作符：返回一个数的余弦值。数据以弧度计算。

输入变量可以是任何数字基本数据类型，输出变量必须是 REAL 或 LREAL 类型。

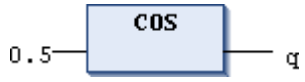
IL 举例：(* q 的结果为 0.877583 *)

```
LD    0.5
COS
ST    q
```

ST 举例：

q:=COS(0.5);

FBD 举例：



正切

IEC 操作符：返回一个数的正切值。数据以弧度计算。

输入变量可以是任何数字基本数据类型，输出变量必须是 REAL 或 LREAL 类型。

IL 举例：(q 的结果为 0.546302 *)

```
LD      0.5
TAN
ST      q
```

ST 举例：

q:=TAN(0.5);

FBD 举例：

反正弦

IEC 操作符：返回一个数的反正弦（正弦的反函数）值。

输入变量可以是任何数字基本数据类型，输出变量必须是 REAL 或 LREAL 类型。

IL 举例：(q 的结果为 0.523599 *)

```
LD      0.5
ASIN
ST      q
```

ST 举例：

q:=ASIN(0.5);

FBD 举例：

反余弦

IEC 操作符：返回一个数的反余弦（余弦的反函数）值。数据以弧度计算。

输入变量可以是任何数字基本数据类型，输出变量必须是 REAL 或 LREAL 类型。

IL 举例：(q 的结果为 1.0472)

```
LD      0.5
ACOS
ST      q
```

ST 举例：

q:=ACOS(0.5);

FBD 举例：

反正切

IEC 操作符：返回一个数的反正切（正切的反函数）值。数据以弧度计算。

输入变量可以是任何数字基本数据类型，输出变量必须是 REAL 或 LREAL 类型。

IL 举例：(q 的结果为 0.463648)

```
LD      0.5
ATAN
ST      q
```

ST 举例：

q:=ATAN(0.5);

FBD 举例：

幂

IEC 操作符：求一个变量关于另一个变量的幂：

OUT = IN1IN2.

输入变量可以是任何数字基本数据类型，输出变量必须是 REAL 或 LREAL 类型。

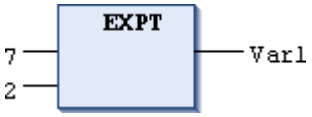
IL 举例：(结果为 49)

```
LD      7
EXPT    2
ST      Var1
```

ST 举例：

var1 := EXPT(7,2);

FBD 举例：



3.3.11 IEC 扩展操作符

CoDeSys 支持的 IEC 扩展操作符如下:

- ADR
- BITADR
- INDEXOF

- SIZEOF
- ANDN
- ORN
- XORN
- _DELETE
- _ISVALIDREF
- _NEW
- _QUERYINTERFACE
- _QUERYPOINTER

取地址

IEC 61131-3 标准中没有规定这个地址操作符。

ADR 返回变量自身的地址，数据类型为 DWORD。这个地址可以作为指针传递给操作函数，也可以赋给工程内的某个指针。

注意：与 CoDeSys V2.3 不同的是，现在版本中的ADR 操作符可以与函数名、程序名、功能块名和方法名一起使用，因此它可以取代 INDEXOF 操作符。请参见“函数指针”。但需要注意的是，函数指针可以指向外部库，但不能在 CoDeSys 内部调用函数指针！若需要使用系统调用（系统运行时），您必须为函数对象设置各自的属性（位于“编译”菜单）。

ST 举例：

```
dwVar:=ADR(bVAR);
```

IL 举例：

```
LD      bVar
ADR
ST      dwVar
```

注意：在线修改之后，某些地址中的数据可能会被改变。请在使用地址指针时务必注意。

取位地址

IEC 操作符： 地址函数，IEC 61131-3 标准中没有规定此操作符。

BITADR 的返回值为 DWORD 中的位偏移量。请注意偏移量的值取决于目标设置中的字节编址是否被激活。

```
VAR
  var1 AT %IX2.3:BOOL;
  bitoffset: DWORD;

END_VAR
```

ST 举例：

```
bitoffset:=BITADR(var1); (* 如果字节编址＝TRUE，则结果为19，如果字节编址为FALSE，则结果为 35 *)
```

IL 举例：

```
LD      Var1
BITADR
ST      bitoffset
```

注意：在线修改之后，某些地址中的数据可能会被改变。请在使用地址指针时务必注意。

索引

IEC 61131-3 标准中没有规定此算术操作符。

此函数用来查找 POU 的内部索引。

ST 举例：

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

字节长度

IEC 61131-3 标准中没有规定此算术操作符。

这个操作符用来确定给定变量 x 需要占用多少个字节。

SIZEOF 操作符通常返回一个无符号数。返回值的类型与变量 x 的大小相匹配。

SIZEOF(x)的返回值	返回值的类型
0 <= x 的值 < 256	USINT
256 <= x 的值 < 65536	UINT
65536 <= x 的值 < 4294967296	UDINT
4294967296 <= x 的值	ULINT

ST 举例：

```
var1 := SIZEOF(arr1);    (* var1:=USINT#10; *)
```

IL 举例： (结果为 10)

```
arr1:ARRAY[0..4] OF INT;
Var1:INT;
LD      arr1
SIZEOF
ST      Var1
```

与非

IEC 61131-3 标准中没有规定此 IEC 位字符串操作符。

此功能尚未实现。

或非

IEC 61131-3 标准中没有规定此 IEC 位字符串操作符。

此功能尚未实现。

异或非

IEC 61131-3 标准中没有规定这个 IEC 位字符串操作符。

此功能尚未实现。

__New

此操作符不在 IEC 61131-3 标准中规定.

注意: 出于兼容需要，编译器版本必须>= 3.3.2.0.

操作符 __NEW 为功能块实例或标准数据类型数组分配内存。此操作符返回一个合适的类型，即一个指向对象的非零指针。如果在分配时未使用该操作符，将会弹出错误信息。

如果__NEW 操作不成功，将返回 0.

释放内存使用 __DELETE.

语法:

__NEW (<类型>, [<大小>])

该操作符将创建一个指定类型的<类型>新对象并返回该<类型>的指针. 在创建之后调用对象的初始化。 如果返回 0，表示创建失败。

如果 <类型>是标量，必须设置可选操作数<长度>，操作符以该长度创建标量类型的数组。

示例: pScalarType := __New(ScalarType, length);

注意:在线修改动态创建的对象时不能复制代码!

因此不含库(因为它们不能改变)的功能块 和具有'enable_dynamic_creation' 属性的功能块允许使用 __New-操作符

如果有此标志的功能块进行修改，拷贝代码将是必须的，此时将产生错误信息。

注意: 内存分配的代码应该是不可重入的。

信号量 (SysSemEnter) 用于避免两个任务同时分配内存. 因此, __New 的扩展使用将会引起较高的抖动。

标量类型的示例:

```
TYPE DUT :
STRUCT
a,b,c,d,e,f : INT;
END_STRUCT
END_TYPE
```

```
PROGRAM PLC_PRG
VAR
	pDut : POINTER TO DUT;
	bInit: BOOL := TRUE;
	bDelete: BOOL;
END_VAR

IF (bInit) THEN
	pDut := __NEW(DUT);
	bInit := FALSE;
END_IF
IF (bDelete) THEN
	__DELETE(pDut);
END_IF
```

功能块示例:

```
FBDynamic(FP)
{attribute 'enable_dynamic_creation'}
FUNCTION_BLOCK FBDynamic
VAR_INPUT
	in1, in2 : INT;
END_VAR
VAR_OUTPUT
	out : INT;
END_VAR
VAR
	test1 : INT := 1234;
	_inc : INT := 0;
	_dut : POINTER TO DUT;
	neu : BOOL;
END_VAR
out := in1 + in2;
```

```
PLC_PRG(PRG)
VAR
        pFB : POINTER TO FBDynamic;
        loc : INT;
        bInit: BOOL := TRUE;
        bDelete: BOOL;
END_VAR
IF (pFB <> 0) THEN
        pFB^(in1 := 1, in2 := loc, out => loc);
        pFB^.INC();
END_IF
IF (bDelete) THEN
        __DELETE(pFB);
END_IF
```

数组示例:

```
PLC_PRG(PRG)
VAR
        bInit: BOOL := TRUE;
        bDelete: BOOL;
       pArrayBytes : POINTER TO BYTE;
       pArrayDuts : POINTER TO BYTE;
       test: INT;
       parr : POINTER TO BYTE;
END_VAR
IF (bInit) THEN
       pArrayBytes := __NEW(BYTE, 25);
       bInit := FALSE;
END_IF
IF (pArrayBytes <> 0) THEN
       pArrayBytes[24] := 125;
       test := pArrayBytes[24];
END_IF
IF (bDelete) THEN
```

```
    __DELETE(pArrayBytes);
END_IF
```

__DELETE

该操作符不在 IEC 61131-3 标准中规定.

注意: 出于兼容需要，编译器版本必须>= 3.3.2.0.

操作符_DELETE 释放由__NEW 分配的内存。 __DELETE 没有返回值，其操作数执行之后将被设置为 0.

分配内存请使用__NEW.

语法：__DELETE(<指针>)

如果指针是指向功能块的，在指针被设置为 NULL 之前将调用专用方法 FB_Exit.

功能块示例：

```
FUNCTION_BLOCK FBDynamic
VAR_INPUT
in1, in2 : INT;
END_VAR
VAR_OUTPUT
out : INT;
END_VAR
VAR
test1 : INT := 1234;
_inc : INT := 0;
_dut : POINTER TO DUT;
neu : BOOL;
END_VAR
out := in1 + in2;

METHOD FB_Exit : BOOL
VAR_INPUT
bInCopyCode : BOOL;
END_VAR
__Delete(_dut);

METHOD FB_Init : BOOL
```

```
VAR_INPUT
bInitRetains : BOOL;
bInCopyCode : BOOL;
END_VAR
_dut := __NEW(DUT);

METHOD INC : INT
VAR_INPUT
END_VAR
_inc := _inc + 1;
INC := _inc;

PLC_PRG(PRG)
VAR
pFB : POINTER TO FBDynamic;
bInit: BOOL := TRUE;
bDelete: BOOL;
loc : INT;
END_VAR
IF (bInit) THEN
pFB := __NEW(FBDynamic);
bInit := FALSE;
END_IF
IF (pFB <> 0) THEN
pFB^(in1 := 1, in2 := loc, out => loc);
pFB^.INC();
END_IF
IF (bDelete) THEN
__DELETE(pFB);
END_IF
```

__ISVALIDREF

该操作符不是由 IEC 61131-3 标准规定的。它可用于检查一个引用是否指向有效值。关于怎样使用以及示例，请见引用数据类型的描述。

__QUERYINTERFACE

该运算符不是由 IEC 61131-3 标准规定的。

在运行时__QUERYINTERFACE 允许接口引用的类型转换。该运算符返回一个 BOOL 类型的结果。TRUE 意味着转换成功执行。

注意：由于兼容性的原因，要转换的引用定义必须是基接口__SYSTEM.IQueryInterface 的扩展，并且编译器版本必须大于 3.3.0.20。

语法：

__QUERYINTERFACE(<ITF_Source>, < ITF_Dest>)

这个运算符的第一个参数是一个接口引用或者预期类型的功能块实例，第二个参数是一个接口引用。如果从 ITF_Source 引用的对象实现了这个接口，那么在执行___QUERYINTERFACE 之后，ITF_Dest 保留了一个期望接口的引用。此时转换成功，运算符返回结果 TRUE。否则返回 FALSE。

显式转换的先决条件是，ITF_Source 和 ITF_Dest 都必须是接口__System.IQueryInterface 的扩展。该接口是隐式提供的，不需任何库。

ST 中的示例:

```
INTERFACE ItfBase EXTENDS __System.IQueryInterface
METHOD mbase : BOOL
END_METHOD

INTERFACE ItfDerived1 EXTENDS  __System.IQueryInterface
METHOD mderived1 : BOOL
END_METHOD

INTERFACE ItfDerived2 EXTENDS  __System.IQueryInterface
METHOD mderived2 : BOOL
END_METHOD

PROGRAMM POU
VAR
itfderived1 : ItfDerived1;
itfderived2 : ItfDerived2;
bTest1, bTest2, xResult1, xResult2: BOOL;
END_VAR
```

xResult1 := __QUERYINTERFACE(itfbase, itfderived1); // ItfBase 类型的变量， 也就是 ItfDerived1

xResult2 := __QUERYINTERFACE(itfbase, itfderived2); // ItfBase 类型的变量， 也就是 ItfDerived2

IF (xResult1 = TRUE) THEN

bTest1 := itfderived1.mderived1();

ELSIF xResult2 THEN

bTest2 := itfderived2.mderived2();

END_IF

范围操作符

如果一个工程内多个变量或模块的名称相互重复，可以使用几种 IEC 扩展操作符来区分欲访问的对象。下列范围操作符可以定义它们各自的命名空间：

"." (全局操作符)

"<全局变量列表名>."

"<库名>."

"<枚举名>."

全局操作符

全局操作符：IEC 61131-3 标准的扩展操作符。

以“.”开头的实例路径通常指向全局变量（名字空间）。因此，当局部变量与全局变量的名称同为“<varname>”时，“.<varname>”指的是全局变量。

全局变量列表名

全局操作符：IEC 61131-3 标准的扩展操作符。

全局变量列表的名称可以作为这个列表所包含变量的名字空间，这样就可以在不同的全局变量列表中使用相同的名字来声明变量。用户只需在变量名之前加上“<全局变量列表名>”就可以访问需要的变量。

例如：

全局变量列表“globlist1”和“globlist2”都包含有名为“varx”的变量。下面的语句把“globlist2”的“varx”赋给“globlist1”的“varx”：

globlist1.varx := globlist2.varx; (* 在本代码行 *)

如果两个以上的全局变量列表中声明了同一个变量名，而引用这个变量时却没有在前面加上全局变量列表名，那么系统将报错。

枚举名

范围操作符：IEC 61131-3 标准的扩展操作符。

类型名称枚举可以用来确定需要访问的枚举常量，用户只需将“<枚举名>”置于常量名之前即可。因此，您可以在不同的枚举中使用相同的常量。

举例：

常量“blue”是枚举“Colors”的成员，同时也是枚举“Feelings”的成员。

```
color := Colors.Blue;   // 访问 Color 类型中的枚举值 Blue
feeling := Feelings.Blue; // 访问 Feeling 类型中的枚举值 Blue
```

3.4 操作数

以下类型均可作为操作数：

- 常量(布尔常量、时间常量、日期常量、时刻常量、 日期时间常量、数字常量、实数/长实数常量、字符串常量、文字)
- 变量
- 地址
- 函数

3.4.1 常量

布尔常量

布尔常量只有 TRUE 和 FALSE 两个逻辑值。

参见：布尔数据类型

时间常量

时间常量通常用来操作标准定时器模块。除了符合IEC61131-3标准的32位时间常量TIME，也可以扩展支持 LTIME，并将其作为高精度定时器的时间基量。LTIME 长度为 64 位，精度为纳秒。

声明时间常量的语法：

t#<时间声明>

除了“t#”，也可以使用“T#”、“time”、“TIME”。

时间声明可以包含以下时间单位。这些单位必须按照以下的顺序出现，但无需同时使用。

“d”：天

“h”：小时

“m”：分

“s”：秒

“ms”：毫秒

ST 赋值中正确的时间常量举例:

TIME1 := T#14ms;	
TIME1 := T#100S12ms;	（*最高位可以超限*）
TIME1 := t#12h34m15s;	

以下的时间常量是不正确的：

TIME1 := t#5m68s;	（*低位不允许超限*）
TIME1 := 15ms;	（*缺少 T# *）
TIME1 := t#4ms13d;	（*顺序错误*）

声明长时间常量（LTIME）的语法：

LTIME#<时间声明>

时间声明可以包含 TIME 常量中使用的时间单位（见上文），还可以使用：

“us”：微妙

“ns”：纳秒

例如：

ST 赋值中正确的 LTIME 常量举例：

LTIME1 := LTIME#1000d15h23m12s34ms2us44ns

LTIME1 := LTIME#3445343m3424732874823ns

参见：时间数据类型

日期常量

本常量用于输入日期：

语法：

dt#<日期声明>

除了“d#”也可以使用：“D#”、“date”、“DATE”。

日期声明使用“<年-月-日>”的格式。

日期值在内部以 DWORD 值形式处理，以秒为单位从 1970 年 01 月 01 日 00:00 开始。

举例：

DATE#1996-05-06

d#1972-03-29

参见：时间数据类型

时刻常量

本常量用来存储一天中的时刻。

语法：tod#<时间声明>

除了“tod#”，也可以使用“TOD#”、“time_of_day”、“TIME_OF_DAY”。

时间声明使用“<时:分:秒>”的格式。

秒值可以为实数，也就是说可以指定带小数的秒值。

时刻变量在内部以 DWORD 值形式处理，以秒为单位从 00:00 开始。

举例：

TIME_OF_DAY#15:36:30.123

tod#00:00:00

参见：时间数据类型

日期时间常量

日期常量和时刻常量可以连接起来，构成所谓的日期时间常量。

语法：dt#<日期和时间声明>

除了“dt#”，也可以使用：“DT#”、“date_and_time”、“DATE_AND_TIME”。

日期和时间的声明使用“<年-月-日-时:分:秒>”的格式。

秒值可以为实数，也就是说可以指定带小数的秒值。

日期时间常量在内部以 DWORD 值形式处理，以秒为单位从 1970 年 01 月 01 日 00:00 开始。

举例：

DATE_AND_TIME#1996-05-06-15:36:30

dt#1972-03-29-00:00:00

参见：时间数据类型

数字常量

数字常量可以是二进制数、八进制数、十进制数和十六进制数。

如果一个整数不是十进制数，那么它的前方必须加上进制数字和符号（#）。

十六进制数中的数值 10-15 将和其他情况下一样采用字母 A-F 表示。

您可以在数字中使用带下划线的字符。

举例：

14	（十进制数）
2#1001_0011	（二进制数）
8#67	（八进制数）
16#A	（十六进制数）

这些数值可以是 BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL 或者 LREAL 类型。.

无法将“较大”的变量类型自动转换为“较小”的变量类型，这意味着不能简单地把 DINT 变量作为 INT 变量使用，而必须使用类型转换。

实数/长实数常量

实数和长实数常量可以以十进制小数或者指数的形式表示，使用标准美式格式的小数点。

举例：

7.4	表示7,4
1.64e+009	表示1,64e+009

字符串常量

字符串就是若干个字符的序列。

字符串常量在一组单引号之间。您也可以在其中插入空格和特殊字符（例如变音符号）。它们将和其它字符一样被处理。请注意下面的例子中，在字符串常量里使用的美元符号“$”：

输入的字符串	含义
$<两个十六进制数>	八位字符代码的十六进制表示
$$	美元符号
$'	单引号
$L 或 $l	输入行
$N 或 $n	新行
$P 或 $p	输入页
$R 或 $r	换行
$T 或 $t	Tab 键

举例：

'w1W ?'

' Abby and Craig '

':-)'

'costs ($$)'

文字

一般说来，在使用 IEC 常量的时候，会尽可能地使用最短的数据类型。若要需要使用其他的数据类型，则可以借助于文字实现，而无需做出明确的常量声明。

为此，这个常量需要加上一个定义了其类型的前缀。

语法：<类型>#<文字>

<类型> 指定需要的数据类型；可以使用的类型包括：BOOL、SINT、USINT、BYTE、INT、UINT、WORD、DINT、UDINT、 DWORD、REAL、LREAL。类型名称必须使用大写形式。

<文字> 指定常量。此处的数据必须符合<类型>指定的数据类型。

举例：var1:=DINT#34;

若常量的数据在转化为目标形式的过程中有损失，会出现错误信息提示。

一般常量可以使用的地方都可以使用文字。

3.4.2 变量

用户可以在 POU 的声明部分或全局变量列表中声明全局变量。

见变量声明的相关信息，其中包括变量标识符和多重使用的规则。

在声明的数据类型允许的任何地方都可以使用变量。

您可以通过输入助手访问有效的变量。

访问数组、结构和 POU 的变量

使用下面的语法访问变量……

二维数组元素：<数组名>[索引 1,索引 2]

结构变量：<结构名>.<变量名>

功能块和程序变量：<功能块名>.<变量名>

按位寻址

在整型变量中可以访问每个独立的数据位。为此在变量后面附加需要寻址的位的索引，并以“圆点”分隔。位索引可以由任意常量给出，从 0 开始编号。

注意：在视图中明确地定义了位索引之后（并且不是通过已定义的常量），还需要通过数据服务器连接才能访问数据位。

语法：<变量名>.<位索引>

举例：

```
a : INT;
b : BOOL;
...
a.2 := b;
```

变量 a 的第三位将等于变量 b。

如果索引大于变量的数据位的宽度，将会出现下列错误提示：“Index '<n>' outside the valid range for variable '<var>'!”

可以访问数据位的数据类型包括： SINT、INT、DINT、USINT、UINT、UDINT、BYTE、WORD、DWORD。

如果变量类型不允许访问数据位，将会出现下列错误提示：“Invalid data type '<type>' for direct indexing”

VAR_IN_OUT 变量不能访问数据位！

通过全局常量访问数据位：

如果您声明了一个定义了位索引的全局常量，您就可以使用这个常量访问数据位。

通过全局常量访问数据位举例：

例 1 访问变量的数据位；例 2 访问结构类型变量的数据位：

两个例子共用的在全局变量列表中的声明:

变量的使能位（enable）决定了可以访问哪一个数据位：

```
VAR_GLOBAL CONSTANT
enable:int:=2;
END_VAR
```

例 1，访问整型变量的数据位：

在 POU 中的声明：

```
VAR
xxx:int;
END_VAR
```

访问数据位：

```
xxx.enable:=true;      (* -> 变量 xxx 的第三位被设置为 TRUE *)
```

例 2，访问结构体的整型成员的数据位：

声明结构 stru1：

```
TYPE stru1 :
STRUCT
bvar:BOOL;
rvar:REAL;
wvar:WORD;
{bitaccess enable 42 'Start drive'}
END_STRUCT
END_TYPE
```

在 POU 中的声明：

```
VAR
x:stru1;
END_VAR
```

访问数据位：

```
x.wvar.enable:=true;
```

以上的操作会把变量x 的第 42 位设置为TRUE。由于 bvar 长度为 8 位，而rvar 长度为 32 位，于是将会访问 wvar 变量的第二位，并得到结果数值 4。

3.4.3 地址

注意地址

注意：请您在使用指向地址的指针时注意，在线修改可能会改变地址中的内容！

内存

您可以使用任意允许的数据长度访问内存。

例如，地址%MD48 代表了内存区域中的字节地址 192、193、194 和 195（48 * 4 = 192）。第一个字节的数字是 0。下表列出了 IEC 寻址时，根据数据类型的大小（X:bit,B:byte,W:word:D:dword），相应的内存位置。

Ex 示例:

%MX	96.0 - 96.7	96.8 - 192.15	97.0 - 97.7	97.8 - 97.15
%MB	192	193	194	195
%MW	96		97	
%MD	48			

您可以使用相同的方式访问字、字节甚至二进制位：可以使用地址%MX96.0，访问第 96 个字的第一个二进制位（二进制位通常以字的长度存储）。

更多关于寻址的信息见地址 .

注意： 请您在使用指向地址的指针时注意，在线修改可能会改变地址中的内容！

地址

当指定一个地址时，内存位置和大小是由特殊字符序列表示的。

语法：%<范围前缀><长度前缀><数字|.数字|.数字.....>

可以使用以下内存区域范围的前缀：

I	输入区（通过输入驱动的物理输入，“传感器”）
Q	输出区（通过输出驱动的物理输出，“执行器”）
M	内存区

可以使用以下的长度前缀：

X	单个二进制位
None	单个二进制位
B	字节（8位）
W	字（16位）
D	双字（32位）

例如：

%QX7.5 and %Q7.5	输出区的地址7，第5位
%IW215	输入区的地址215，1个字
%QB7	输出区的地址7，1个字节
%MD48	内存区的地址48，双字。
%IW2.5.7.1	取决于 PLC 的设置

ivar AT %IW0 : WORD;	t 带有指定地址的变量声明的例子

请确定使用有效的地址：

要在应用中指定一个有效地址，首先您必须知道进程映像中的适当位置，也就是所使用的内存区：Input(I)，Output(Q)或者 Memory(M)区域，见上。然后关于指定想要的大小：

当前设备配置和设定（硬件结构、设备描述、IO 设置）扮演了决定性的角色。尤其要考虑到，使用“字节寻址模式”的设备和使用面向字模式的“IEC 寻址模式”的设备在地址解释之间差异。

所以依赖于大小和寻址模式，不同的内存单元可被相同的地址定义寻址。

见下表中字节寻址和面向字的 IEC 寻址对于 bit，bytes，words 和 dwords 的比较。这个表画出了字节寻址模式时重叠的内存区（见表后示例）。

关于写法，请注意对于 bit 地址，IEC 寻址模式总是面向字的，就是说在点号之前的区域对应于字的个数，名称之后的区域对应于 bit 的个数。

对于地址大小 D，W 和 X，面向字节和字的寻址的比较：

双字/字				字节	X （位）					
字节寻址		面向字的IEC寻址			字节寻址			面向字的IEC寻址		
D0	W0	D0	W0	B0	X0.7	...	X0.0	X0.7	...	X0.0
D1	W1			B1	X1.7	...	X1.0	X0.15	...	X0.8
...	W2		W1	B2	...			X1.7	...	X1.0
	W3			B3				X1.15	...	X1.8
	W4	D1	W2	B4						
	...			B5						
			W3	B6						
				B7						
		D2	W4	B8						
		...		...						
		...		...						
		...		...						
D(n-3)		D(n/4)	...							
	W(n-1)		W(n/2)							
				Bn	Xn.7	...	Xn.0	X(n/2).15	...	X(n/2).8

n =字节数

在字节寻址模式时，内存重叠是变化的，例如:

D0 包括 B0-B3，W0 包括 B0 和 B1，W1 包括 B1 和 B2，W2 包括 B2 和 B3->。为了避免重叠，W1 或者 D1,D2,D3 都不能用于寻址！

注意：如果没有指定明确的单个二进制位的地址，布尔值的长度将为一个字节。例如：改变 varbool1 AT %QW0 的值将影响从 QX0.0 到 QX0.7 的内容。

注意：请您在使用指向地址的指针时注意，在线修改可能会改变地址中的内容！

3.4.4 函数

在 ST 中，也可以把函数调用作为一个操作数

例如：Result := Fct(7) + 3;

TIME()-函数

这个函数的返回值是系统启动后所经过的时间（单位为毫秒）。

数据类型为 TIME。

例如在 IL 中：

```
TIME
ST systime   (* 将会得到例如：T#35m11s342ms 的结果 *)
```

ST 中的例子：

```
systime:=TIME();
```

4 Standard.lib 库和 Util.lib 库使用说明

4.1 CODESYS 中的操作符总结

下表为操作符总结，这些操作符分别都存在于 CODESYS 的 Standard.lib 和 Util.lib 程序库中。

注意‘IL 符’栏：只有使用此符号的行才被显示。一个必要条件是：第一组必要的操作数已被顺利载入前一行（如，LD in）。

‘Mod.IL’栏显示 IL 中有可能出现的修饰符：

C	前行表达式为 TRUE 的结果的修饰符。
N	对于 JMPC，CALC，RETC：如前行的结果为 FALSE，则执行此命令。
N	否则：否定操作数（非累积器）
(	括弧中操作符：只有括号出现后操作才开始执行。

表 3-1 操作符总结表

在 ST 中	在 AWL 中	Mod.AWL	描述
'			字符串分隔符
.. [ ]			Array 值域大小
:			声明中操作数与类型间的分隔符。
;			说明终止。
^			指示符废止。
	LD var1	N	缓冲器中变量 1 值的载入。
:=	**ST var1**	N	将实际结果存入变量 1。
	S boolvar		当实际结果为 TRUE，准确设置操作数布尔变量为 TRUE。
	R boolvar		当实际结果为FALSE，准确设置操作数布尔变量为 TRUE。
	JMP label	**CN**	直接跳至标签。
<Program name>	**CAL** prog1	**CN**	调用程序 prog1。
<Instance name>	**CAL** inst1	**CN**	调用功能程序 inst1。
<Fctname>(vx, vy,..)	**<Fctname> vx, vy**	**CN**	调用功能 fctname 和 transmit 变量 vx，vy。
RETURN	**RET**	CN	Leave POU and go back to caller
	(		括号后的值作操作数处理，括号前的运算在括号内的表达式之前不执行。
	)		现执行已被设置运算
AND	**AND**	N,(	位 AND
OR	**OR**	N,(	位 OR
XOR	**XOR**	N,(	位专用 OR
NOT	**NOT**		位 NOT

+	**ADD**	(	加法
-	**SUB**	(	减法
*****	**MUL**	(	乘法
/	**DIV**	(	除法
>	**GT**	(	大于
>=	**GE**	(	大于等于
=	**EQ**	(	等于
<>	**NE**	(	不等于
<=	**LE**	(	小于等于
<	**LT**	(	小于
MOD(in)	**MOD**		模相除
INDEXOF(in)	**INDEXOF**		POU in1 的内部指数
SIZEOF(in)	**SIZEOF**		已知 in 数据类型所需字节数
SHL(K,in)	**SHL**		操作符 in 位左移 k
SHR(K,in)	**SHR**		操作符 in 位右移 k
ROL(K,in)	**ROL**		操作符 in 位旋转 k 到左
ROR(K,in)	**ROR**		操作符 in 位旋转 k 到右
SEL(G,in0,in1)	**SEL**		in0（G 为 FALSE）和 in1（G 为 TRUE）两个操作数之间的两进制选择。
MAX(in0,in1)	**MAX**		2 个值中较大的一个的复原。
MIN(in0,in1)	**MIN**		in0 和 in1 两值之中较小一个的复原。
LIMIT(MIN,in,Max)	**LIMIT**		限制值域（in 重置为 MIN，或 MAX，以免超出范围）。
MUX(K,in0,...in_n)	**MUX**		一组值中（in 至 In_n）选择第 k 个值。
ADR(in)	**ADR**		DWORD 中操作数的地址。
ADRINST()	**ADRINST**()		调用操作符即功能块操作符的地址。
BITADR(in)	**BITADR**		DWORD 中操作数的位平衡。
BOOL_TO_<type>(in)	**BOOL_TO_<type>**		布尔操作数的类型转换。
<type>_TO_BOOL(in)	**<type>_TO_BOOL**		类型转换成 BOOL。
INT_TO_<type>(in)	**INT_TO_<type>**		INT 操作数类型转换位另一基本类型。
REAL_TO_<type>(in)	**REAL_TO_<type>**		REAL 操作数类型转换位另一基本类型。
LREAL_TO_<type>(in)	**LREAL_TO_<type>**		LREAL 操作数类型转换位另一基本类型。
TIME_TO_<type>(in)	**TIME_TO_<type>**		TIME 操作数类型转换位另一基本类型。
TOD_TO_<type>(in)	**TOD_TO__<type>**		TOD 操作数类型转换位另一基本类型。
DATE_TO_<type>(in)	**DATE_TO_<type>**		DATE 操作数类型转换位另一基本类型。
DT_TO_<type>(in)	**DT_TO_<type>**		DT 操作数类型转换位另一基本类型。

STRING_TO_<type>(in)	**STRING_TO_<type>**		字符串操作数类型转换位另一基本类型，in 必须含所求类型的有效值。
TRUNC(in)	**TRUNC**		从 REAL 转换为 INT。
ABS(in)	**ABS**		操作数 in 的绝对值。
SQRT(in)	**SQRT**		操作数 in 的平方根。
LN(in)	**LN**		操作数 in 的自然对数。
LOG(in)	**LOG**		操作数 in 的以 10 为底的对数。
EXP(in)	**EXP**		操作数 in 的幂方。
SIN(in)	**SIN**		操作数 in 的正弦。
COS(in)	**COS**		操作数 in 的余弦。
TAN(in)	**TAN**		操作数 in 的正切。
ASIN(in)	**ASIN**		操作数 in 的反正弦。
ACOS(in)	**ACOS**		操作数 in 的反余弦。
ATAN(in)	**ATAN**		操作数 in 的反正切。
EXPT(in,expt)	**EXPT** expt		操作数 in with expt 的幂乘方。

<备注> 请于每个 CODESYS 程序库中的相关附录里获取详细的使用信息。

4.2 Standard.lib 库指令

标准库 Standard.lib 由 CoDeSys 编程系统默认提供。

所有符合IEC61131-3 的函数和功能块被包含在标准库中，并作为IEC 编程系统中的标准程序组织单元。标准函数与“操作符”之间的区别在于，操作符能直接被编程系统所识别，而标准程序组织单元需要添加到工程中才能使用（通过添加标准库）。

表 3-2 Standard.lib 库指令一览表

在 ST 中	在 AWL 中	描述
LEN(in)	**LEN**	操作数 in 的字符串长度
LEFT(str,size)	**LEFT**	已知字符串 str 大小的左边第一个字符串。
RIGHT(str,size)	**RIGHT**	已知字符串 str 大小的右边第一个字符串。
MID(str,size,pos)	**MID**	POS 处已知大小的部分 str 字符串。
CONCAT('str1','str2')	**CONCAT** 'str2'	两个后继字符串的合并。
INSERT('str1','str2',pos)	**INSERT** 'str2',p	在 pos 处把字符串 str1 插入字符串 str2
DELETE('str1',len,pos)	**DELETE** len,pos	删除部分字符串（长度 len），启动位置为 str1 的 pos 处。
REPLACE('str1','str2',len,pos)	**REPLACE** 'str2',len,pos	替换长度字符串为str2，起始位置为 str1 的 pos 处。
FIND('str1','str2')	**FIND** 'str2'	在 str1 中寻找部分字符串 str2。
SR	**SR**	双稳态的 FB 设置为显性。
RS	**RS**	双稳态的 FB 重设。
SEMA	**SEMA**	FB：臂板软件（可中断）
R_TRIG	**R_TRIG**	FB：上升边界被监测。
F_TRIG	**F_TRIG**	FB：下降边界被监测。

CTU	**CTU**	FB：升值计数。
CTD	**CTD**	FB：下降计数。
CTUD	**CTUD**	FB：上下计数。
TP	**TP**	FB：触发
TON	**TON**	FB：On-Delay 时钟。
TOF	**TOF**	FB：Off-Delay 时钟。
RTC	**RTC**	FB：现实时钟。

4.2.1 字符串功能

LEN

由标准库 standard.library 提供。

字符串（STRING）类型的函数。其功能是返回字符串的长度。

输入：

STR：字符串（STRING）；被分析的字符串

返回值：整型（INT）；字符串的长度（字符的个数）

IL 语言示例：（结果是“4”）

```
LD      'SUSI'
LEN
ST      VarINT1
```

FBD 语言示例：

ST 语言示例：

VarINT1 := LEN ('SUSI');

注意：字符串函数并不是“多线程安全”的：当调用任务时，字符串函数只能在单个任务中被调用；如果同一个函数在不同的任务中被调用，存在着字符串被多个任务相互覆盖的危险。

LEFT

由标准库 standard.library 提供。

字符串（STRING）类型的函数。其功能是，从左边数起，返回特定字符。

输入：

STR：字符串（STRING）；被分析的字符串

SIZE：整型（INT）；从左边数起，所选字符串的长度（字符的个数）

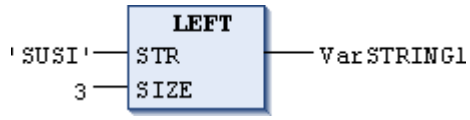
返回值：字符串（STRING）；所选字符

LEFT（STR，SIZE）的含义是：从左边数起，返回 STR 字符串中长度为 SIZE 的字符串。.

IL 语言示例：（结果是“SUS”）

```
LD      'SUSI'
LEFT    3
ST      VarSTRING1
```

FBD 语言示例：



ST 语言示例：

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

注意：字符串函数并不是“多线程安全”的：当调用任务时，字符串函数只能在单个任务中被调用；如果同一个函数在不同的任务中被调用，存在着字符串被多个任务相互覆盖的危险。

RIGHT

由标准库 standard.library 提供。

字符串（STRING）类型的函数，其功能是，从右边数起，返回部分字符串。

输入：

STR：字符串（STRING）；被分析的字符串

SIZE：整型（INT）；从右边数起，所选字符串的长度（字符的个数）

返回值：字符串（STRING）；所选字符

RIGHT（STR，SIZE）的含义是：从右边数起，返回 STR 字符串中长度为 SIZE 的字符。

IL 语言示例：（结果是“USI”）

```
LD      'SUSI'
RIGHT   3
ST      VarSTRING1
```

FBD 语言示例：

ST 语言示例：

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

注意：字符串函数并不是“多线程安全”的：当调用任务时，字符串函数只能在单个任务中被调用；如果同一个函数在不同的任务中被调用，存在着字符串被多个任务相互覆盖的危险。

MID

由标准库 standard.library 提供。

字符串（STRING）类型的函数，其功能是取字符串中的一部分字符。

输入：

STR：字符串（STRING）；被分析的字符串

LEN：整型（INT）；字符串中所选部分字符的长度（字符的个数）

POS：整型（INT）；所选部分字符的起始位置，从 STR 左边开始数。

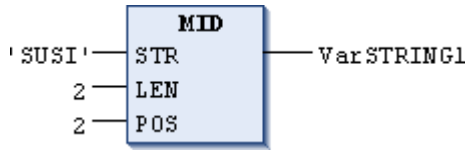
返回值：字符串（STRING）；所选的部分字符串

MID（STR，LEN，POS）的含义是：从 STR 字符串的左边第 POS 个字符开始，返回 LEN 长度的字符。

IL 语言示例：（结果是“US”）

```
LD      'SUSI'
MID     2,
        2
ST      VarSTRING1
```

FBD 语言示例：



ST 语言示例：

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

注意：字符串函数并不是“多线程安全”的：当调用任务时，字符串函数只能在单个任务中被调用；如果同一个函数在不同的任务中被调用，存在着字符串被多个任务相互覆盖的危险。

CONCAT

由标准库 standard.library 提供。

字符串（STRING）类型的函数，其功能是串联（合并）两个字符串。

输入：

STR1，STR2：字符串（STRING）；待合并的两个字符串

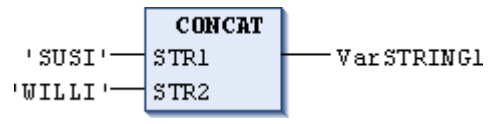
返回值：字符串（STRING）；合并后的字符串

CONCAT（STR1，STR2）的含义是：将 STR1 和 STR2 字符串合并为一个字符串 STR1STR2。

IL 语言示例：（结果是“SUSIWILLI”）

```
LD      'SUSI'
CONCAT  'WILLI'
ST      VarSTRING1
```

FBD 语言示例：



ST 语言示例：

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

注意：字符串函数并不是“多线程安全”的：当调用任务时，字符串函数只能在单个任务中被调用；如果同一个函数在不同的任务中被调用，存在着字符串被多个任务相互覆盖的危险。

INSERT

由标准库 standard.library 提供。

字符串（STRING）类型的函数，其功能是将一个字符串插入另一个字符串的特定位置中。

输入：

STR1：字符串（STRING）；原字符串

STR2：字符串（STRING）；插入到 STR1 中去的字符串

POS：整型（INT）；在 STR1 中插入 STR2 的位置，从 STR1 左边开始数的字符个数

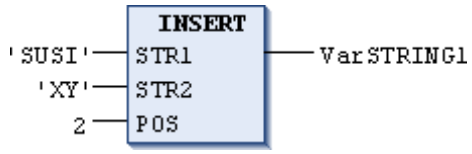
返回值：字符串（STRING）；插入后的新字符串

INSERT（STR1，STR2，POS）的含义是：在 STR1 中的第 POS 个位置，将 STR2 插入到 STR1 中。

IL 语言示例：（结果是“SUXYSI”）

```
LD      'SUSI'
INSERT  'XY'
        2
ST      VarSTRING1
```

FBD 语言示例：



ST 语言示例：

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

注意：字符串函数并不是“多线程安全”的：当调用任务时，字符串函数只能在单个任务中被调用；如果同一个函数在不同的任务中被调用，存在着字符串被多个任务相互覆盖的危险。

DELETE

由标准库 standard.library 提供。

字符串（STRING）类型的函数，其功能是删除一个大的字符串中从指定位置开始的部分字符串。

输入：

STR：字符串（STRING）；待删除部分字符的原字符串

LEN：整型（INT）；被删除的部分字符的长度，字符的个数

POS：整型（INT）；所删除的部分字符的起始位置，从 STR 左边开始数。

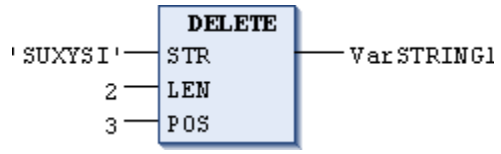
返回值：字符串（STRING）；删除后的新字符串

DELETE（STR，L，P）的含义是：从第 POS 个位置开始，在 STR 中删除 L 个字符。

IL 语言示例：（结果是“SUSI”）

```
LD      'SUXYSI'
DELETE  2       ,
        3
ST      VarSTRING1
```

FBD 语言示例：



ST 语言示例：

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

注意：字符串函数并不是“多线程安全”的：当调用任务时，字符串函数只能在单个任务中被调用；如果同一个函数在不同的任务中被调用，存在着字符串被多个任务相互覆盖的危险。

REPLACE

由标准库 standard.library 提供。

字符串（STRING）类型的函数，其功能是用一个字符串替换一个大的字符串中从指定位置开始的部分字符。

输入：

STR1：字符串（STRING）；原字符串

STR2：字符串（STRING）；替换 STR1 中部分字符的字符串

L：整型（INT）；STR1 中，被替换的部分字符串的长度

P：整型（INT）；插入 STR2 的位置，从 STR1 左边开始数，替换现有的 L 个字符。

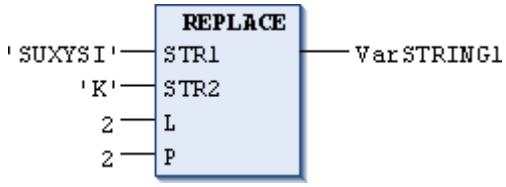
返回值：字符串（STRING）；替换后的新字符串

REPLACE（STR1，STR2，L，P）的含义是：从 STR1 左侧数起的第 P 个位置开始，用 STR2 替换 STR1 中的 L 个字符。

IL 语言示例：（结果是“SKYSI”）

```
LD      'SUXYSI'
REPLACE 'K'     ,
        2       ,
        2
ST      VarSTRING1
```

FBD 语言示例：



ST 语言示例：

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

注意：字符串函数并不是“多线程安全”的：当调用任务时，字符串函数只能在单个任务中被调用；如果同一个函数在不同的任务中被调用，存在着字符串被多个任务相互覆盖的危险。

FIND

由标准库 standard.library 提供。

字符串（STRING）类型的函数 ，其功能是在一个字符串中找到另一个较小的字符串的位置。

输入：

STR1：字符串（STRING）；大字符串

STR2：字符串（STRING）；在 STR1 中查找其位置所在的字符串

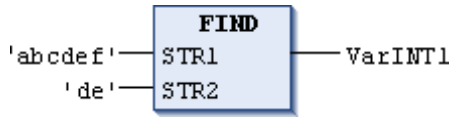
返回值：整型（INT）；在 STR1 中 STR2 的起始位置；如果在 STR1 中没有找到 STR2，则输出 0。

FIND（STR1，STR2）的含义是：在 STR1 中找到 STR2 第一次出现的位置。如果在 STR1 中没有找到 STR2，则输出 0。

IL 语言示例：（结果是“4”）

```
LD      'abcdef'
FIND    'de'
ST      VarSTRING1
```

FBD 语言示例：



ST 语言示例：

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

注意：字符串函数并不是“多线程安全”的：当调用任务时，字符串函数只能在单个任务中被调用；如果同一个函数在不同的任务中被调用，存在着字符串被多个任务相互覆盖的危险。

4.2.2 双稳态功能块

SR

由标准库 standard.library 提供。

功能块，置位优先功能块。

输入：

SET1：布尔型（BOOL）

RESET：布尔型（BOOL）

输出：

Q1：布尔型（BOOL） Q1=SR

（SET1，RESET）的含义是： Q1 =

(NOT RESET AND Q1) OR SET1

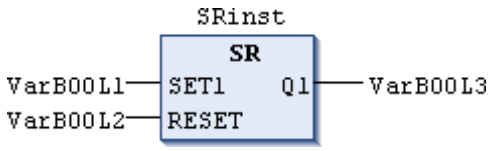
变量声明示例：

SRInst : SR ;

IL 语言示例：

```
CAL     SRinst  (
    SET1:= VarBool1  ,
  RESET:= VarBool2  )
LD      SRinst.Q1
ST      VarBool3
```

FBD 语言示例：



ST 语言示例：

SRInst(SET1:= VarBOOL1 , RESET:=VarBOOL2);

VarBOOL3 := SRInst.Q1 ;

RS

由标准库 standard.library 提供。

功能块，复位优先功能块。

输入：

SET：布尔型（BOOL）

RESET1：布尔型（BOOL）

输出：

Q1：布尔型（BOOL） Q1=RS（SET，

RESET1）的含义是： Q1 = NOT

RESET1 AND (Q1 OR SET)

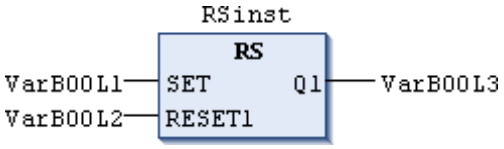
变量声明示例：

RSInst : RS ;

IL 语言示例：

```
CAL     RSinst  (
    SET:= VarBool1  ,
RESET1:= VarBool2  )
LD      RSinst.Q1
ST      VarBool3
```

FBD 语言示例：



Example in ST：

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

VarBOOL3 := RSInst.Q1 ;

SEMA

FB：臂板软件（可中断）

4.2.3 触发器

R_TRIG

由标准库 standard.library 提供。

该功能块检测一个上升沿。

输入：

CLK：布尔型（BOOL）；被检测上升沿的布尔型输入信号

输出：

Q：布尔型（BOOL）；当 CLK 上检测到一个上升沿时，其值为 TRUE

只要输入变量 CLK 为 FALSE，输出 Q 和中间变量 M 都保持为 FALSE。一旦 CLK 为 TRUE，Q 会先返回TRUE，然后 M 也会被置为TRUE。这意味着每次调用这个功能块时，Q 会返回FALSE 直到 CLK 在下降沿后有一个上升沿

(Q := CLK AND NOT M;

M := NOT CLK;)

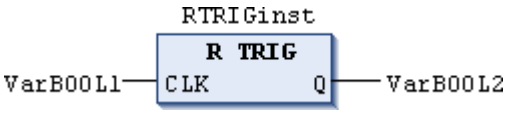
变量声明示例：

RTRIGInst : R_TRIG ;

IL 语言示例：

```
CAL     RTRIGinst   (
    CLK:= VarBool1  )
LD      RTRIGinst.Q
ST      VarBool2
```

FBD 语言示例：



ST 语言示例：

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

F_TRIG

由标准库 standard.library 提供。

该功能块检测一个下降沿。

输入：

CLK：布尔型（BOOL）；被检测其下降沿的布尔型输入信号

输出：

Q：布尔型（BOOL）；当 CLK 上检测到一个下降沿时，其值为 TRUE

只要输入变量 CLK 为 TRUE，输出 Q 和中间变量 M 都保持为 FALSE。一旦 CLK 为 FALSE，Q 会先返回TRUE，然后 M 也会被置为TRUE。这意味着每次调用这个功能块时，Q 会返回FALSE 直到 CLK 在上升沿后有一个下降沿

(Q := NOT CLK AND NOT M;

M := NOT CLK;)

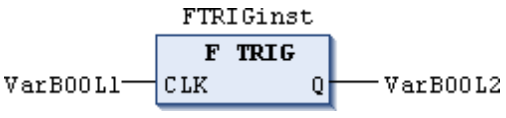
变量声明示例：

```
FTRIGInst : F_TRIG ;
```

IL 语言示例：

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

FBD 语言示例：



ST 语言示例：

```
FTRIGInst(CLK:= VarBOOL1);
```

VarBOOL2 := FTRIGInst.Q;

4.2.4 计数器

CTU

由标准库 standard.library 提供。

该功能块用作一个递增计数器。

输入：

CU：布尔型（BOOL）；该输入端的上升沿触发 CV 的递增计数

RESET：布尔型（BOOL）；当其为 TRUE 时，CV 被复位为 0

PV：字型（WORD）；CV 计数的上限

输出：

Q：布尔型（BOOL）；一旦 CV 达到其上限 PV 时，其值为 TRUE

CV：字型（WORD）；不断加 1 的值，直至其达到 PV

当 RESET 为 TRUE 时，计数变量 CV 被初始化为 0。当 CU 端有一个从 FALSE 变为 TRUE 的上升沿时，CV 将加 1。当 CV 大于或等于上限 PV 时，Q 返回 TRUE。

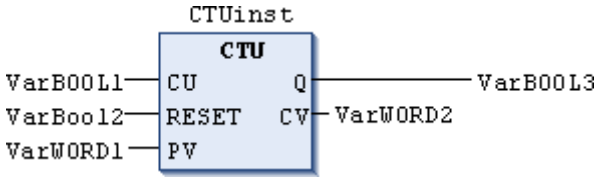
变量声明示例：

CTUInst : CTU ;

IL 语言示例：

```
CAL     CTDinst(
    CD:= VarBOOL1,
  LOAD:= VarBOOL2,
    PV:= VarWORD1,
   CV=> VarWORD2)
LD      CTDinst.Q
ST      VarBOOL3
```

FBD 语言示例：



ST 语言示例：

CTUInst(CU:= VarBOOL1, RESET:=VarBOOL2 , PV:= VarINT1);

VarBOOL3 := CTUInst.Q ;

VarWORD2 := CTUInst.CV;

CTD

由标准库 standard.library 提供。

该功能块用作一个递减计数器。

输入：

CD：布尔型（BOOL）；该输入端的上升沿触发 CV 的递减计数

LOAD：布尔型（BOOL）；当其为 TRUE 时，CV 被置为上限值 PV

PV：字型（WORD）；上限值，也就是 CV 开始递减时的初始值

输出：

Q：布尔型（BOOL）；一旦 CV 达到 0 时，其值为 TRUE

CV：字型（WORD）；不断减 1 的值，从 PV 开始直至其达到 0

当LOAD 为TRUE 时，计数变量CV 被初始化为上限值 PV。当CD 端有一个从FALSE 变为TRUE 的上升沿时，若CV 大于 0 时，它将减 1（也就是说，它不会输出小于 0 的值）。当 CV 等于 0 时，Q 返回 TRUE。

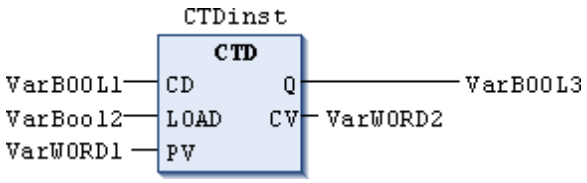
变量声明示例：

CTDInst : CTD ;

IL 语言示例：

```
CAL     CTDinst(
          CD:= VarBOOL1,
        LOAD:= VarBOOL2,
          PV:= VarWORD1,
          CV=> VarWORD2)
LD      CTDinst.Q
ST      VarBOOL3
```

FBD 语言示例：



ST 语言示例：

CTDInst(CD:= VarBOOL1, LOAD:=VarBOOL2 , PV:= VarINT1);

VarBOOL3 := CTDInst.Q ;

VarWORD2 := CTDInst.CV;

CTUD

由标准库 standard.library 提供。

该功能块用作一个递增和递减计数器。输入：

CU：布尔型（BOOL）；当 CU 端有上升沿时，触发 CV 的递增计数

CD：布尔型（BOOL）；当 CD 端有上升沿时，触发 CV 的递减计数

RESET：布尔型（BOOL）；当其为 TRUE 时，CV 被复位为 0

LOAD：布尔型（BOOL）；当其为 TRUE 时，CV 被置为 PV

PV：字型（WORD）；CV 递增时的上限值，或 CV 开始递减时的初始值

输出：

QU：布尔型（BOOL）；一旦 CV 达到 PV 时，其值为 TRUE

QD：布尔型（BOOL）；一旦 CV 达到 0 时，其值为 TRUE

CV：字型（WORD）；不断减 1 的值，从 PV 开始直至其达到 0

当 RESET 为 TRUE 时，计数变量 CV 被初始化为 0。当 LOAD 为 TRUE 时，计数变量 CV 被初始化为上限值 PV。当 CU 端有一个从FALSE 变为TRUE 的上升沿时，CV 将加 1。当CD 端有一个从 FALSE 变为 TRUE 的上升沿时，若 CV 不会降到 0 以下时，它将减 1。当 CV 大于或等于上限 PV 时，QU 返回 TRUE。当 CV 等于 0 时，QD 返回 TRUE。

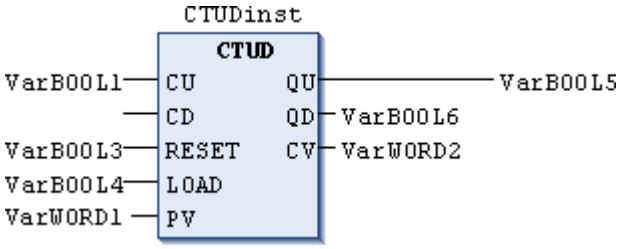
变量声明示例：

CTUDInst : CUTD ;

IL 语言示例：

```
CAL     CTUDinst(
          CU:= VarBOOL1,
       RESET:= VarBOOL3,
        LOAD:= VarBOOL4,
          PV:= VarWORD1,
          QD=> VarBOOL6,
          CV=> VarWORD2)
LD      CTUDinst.QU
ST      VarBOOL5
```

FBD 语言示例：



ST 语言示例：

CTUDInst(CU := VarBOOL1, CD:= VarBOOL2, RESET := VarBOOL3, LOAD:=VarBOOL4 , PV:= VarINT1);

VarBOOL5 := CTUDInst.QU ;

VarBOOL6 := CTUDInst.QD ;

VarINT1 := CTUDInst.CV;

4.2.5 定时器

TP

由标准库 standard.library 提供。

触发定时器功能块。定时器的输出值不断增加，直至其达到限值。在计时期间，“脉冲”变量为 TRUE，其他时候为 FALSE。

输入：

IN：布尔型（BOOL）；该输入端的上升沿触发 ET 端的计时

PT：时间型（TIME）；计时时间的上限值

输出：

Q：布尔型（BOOL）；当 ET 端在计时的时候，其值为 TRUE

ET：时间型（TIME）；时间的当前状态

TP(IN,PT,Q,ET)的含义是：

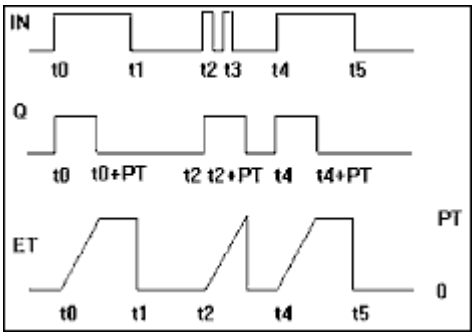
当 IN 为 FALSE 时,Q 为 FALSE，ET 为 0。

一旦 IN 变为TRUE，定时器的输出端 ET 以毫秒精度开始计时，直到它等于 PT，随后它会维持不变。

当 IN 变为 TRUE 且 ET 小于或等于 PT 时，Q 为 TRUE。否则它为 FALSE。

在由 PT 值指定的时间到达时，Q 返回了一个信号

TP 时序图



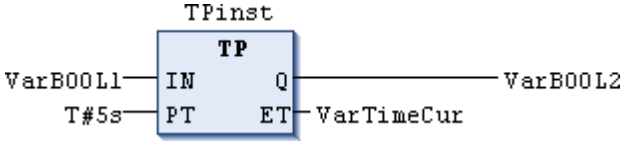
变量声明示例：

TPInst : TP ;

IL 语言示例：

```
CAL     TPinst  (
        IN:= VarBOOL1   ,
        PT:= T#5s       ,
        ET=> VarTimeCur )
LD      TPinst.Q
ST      VarBOOL2
```

FBD 语言示例：



ST 语言示例：

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

VarBOOL2 :=TPInst.Q;

TON

由标准库 standard.library 提供。

定时器功能块，完成开延时的功能。当定时器的输入端变为 TRUE 时，等过了一段时间后，定时器的输出端才变为 TRUE。

输入：

IN：布尔型（BOOL）；该输入端的上升沿触发 ET 端的计时

PT：时间型（TIME）；ET 计时时间的上限值（延时时间）

输出：

Q：布尔型（BOOL）；一旦 ET 端计时达到上限值 PT 时，输出一个上升沿（延时时间过去了）

ET：时间型（TIME）；时间的当前状态

TON(IN,PT,Q,ET)的含义是：

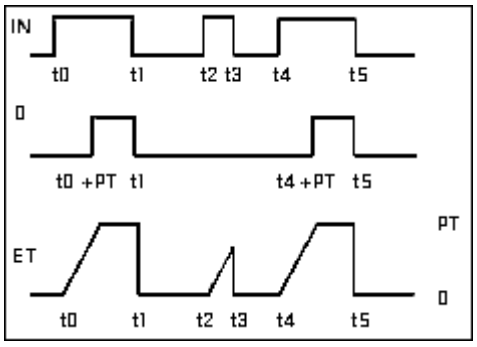
当 IN 为 FALSE 时,Q 为 FALSE，ET 为 0。

一旦IN 变为TRUE，定时器的输出端 ET 以精确到毫秒级别开始计时，直到它等于 PT，随后它会维持不变。

当 IN 变为 TRUE 且 ET 等于 PT 时，Q 为 TRUE。否则它为 FALSE。

由上可知，在等待了精确到毫秒级别的 PT 值决定的时间段后，Q 返回了一个上升沿

TP 时序图



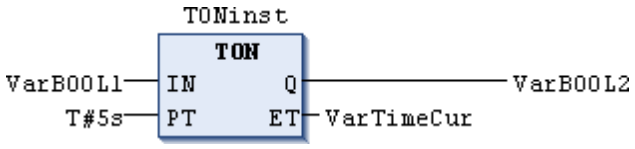
变量声明示例：

TONInst : TON ;

IL 语言示例：

```
CAL     TONinst  (
    IN:= VarBOOL1 ,
    PT:= T#5s     ,
    ET=> VarTimeCur )
LD      TONinst.Q
ST      VarBOOL2
```

FBD 语言示例：



ST 语言示例：

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

TOF

由标准库 standard.library 提供。

定时器功能块，完成关延时的功能。当定时器的输入端由 TRUE 变为 FALSE 时（下降沿），等过了一段时间后，定时器的输出端才变为 FALSE。

输入：

IN：布尔型（BOOL）；该输入端的下降沿触发 ET 端的计时

PT：时间型（TIME）；ET 计时时间的上限值（延时时间）

输出：

Q：布尔型（BOOL）；一旦 ET 端计时达到上限值 PT 时，输出一个下降沿（延时时间过去了）

ET：时间型（TIME）；时间的当前状态

TOFIN,PT,Q,ET)的含义是：

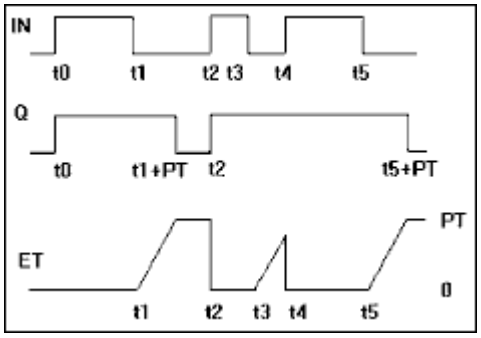
当 IN 为 TRUE 时,Q 为 TRUE，ET 为 0。

一旦 IN 变为 FALSE，定时器的输出端 ET 以精确到毫秒级别开始计时，直到它等于 PT，随后它会维持不变。

当 IN 变为 FALSE 且 ET 等于 PT 时，Q 为 FALSE。否则它为 TRUE。

由上可知，在等待了精确到毫秒级别的 PT 值决定的时间段后，Q 返回了一个下降沿

TP 时序图



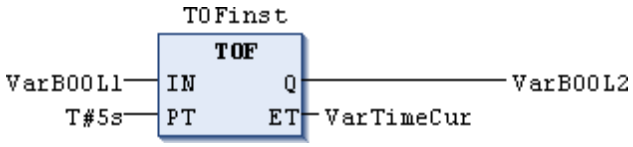
变量声明示例：

TOFInst : TOF ;

IL 语言示例：

```
CAL     TOFinst (
        IN:= VarBOOL1 ,
        PT:= T#5s ,
        ET=> VarTimeCur )
LD      TOFinst.Q
ST      VarBOOL2
```

FBD 语言示例：



ST 语言示例：

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

VarBOOL2 :=TOFInst.Q;

RTC

由标准库 standard.library 提供。

实时时钟定时器功能块，返回从给定时间开始计时的当前日期和时间。

输入：

EN：布尔型（BOOL）；该输入端的上升沿触发 CDT 端的计时

PDT：日期时间型（DATE_AND_TIME）；计时开始的日期和时间

输出：

Q：布尔型（BOOL）；一旦 CDT 端开始计时，输出 TRUE

CDT：日期时间型（DATE_AND_TIME）；计时的当前日期和时间

VarBOOL2:=RTC(EN, PDT, Q, CDT)的含义是：

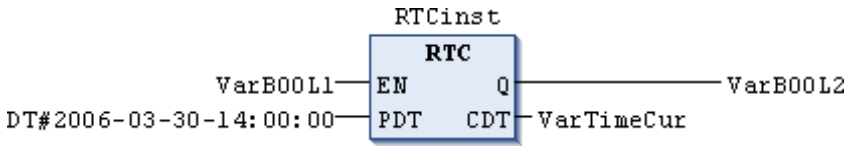
当 EN 为 FALSE 时,输出变量 Q 为 FASLE，CDT 为 DT#1970-01-01-00:00:00。

一旦 EN 变为 TRUE（上升沿），只要 EN 一直保持为 TRUE，CDT 端以 PDT 值作为初始值，开始以秒为精度递增（见上图中的例子）。只要 EN 重新变为 FALSE，CDT 被复位为初始值 DT#1970-01-01-00:00:00。

IL 语言示例：

```
CAL     RTCinst     (
    EN:= VarBOOL1   ,
   PDT:= DT#2006-03-30-14:00… ,
   CDT=> VarTimeCur )
LD      RTCinst.Q
ST      VarBOOL2
```

FBD 语言示例：



ST 语言示例：

RTC(EN:=VarBOOL1, PDT:=DT#2006-03-30-14:00:00, Q=>VarBOOL2, CDT=>VarTimeCur);

4.3 Util.lib 库指令

Util.lib 应用库包含多种功能块的集合，这些功能块用于BCD 码转换，位/字节函数，数学辅助函数（例如控制器、信号发生器、函数操作器）和模拟值处理。

表 1-3 Util.lib 库指令一览表

指令名称	描述
BCD_TO_INT	字节转换：BCD 转为 INT 格式。
INT_TO_BCD	字节转换：INT 转为 BCD 格式。
EXTRACT(in,n)	DWORD in 的第 n 位以 BOOL 类型还原。
PACK	多达 8 位被压入一个字节。
PUTBIT	DWORD in 一个位被设位一定的值。
UNPACK	字节被作为单个的位被复原。
DERIVATIVE	局部派生。
INTEGRAL	整数。
LIN_TRAFO	REAL 值的转换。
STATISTICS_INT	INT 格式的最小，最大和平均值。
STATISTICS_REAL	REAL 格式的最小，最大和平均值。
VARIANCE	变异。
PD	PD 控制器。
PID	PID 控制器。
PID_FIXCYCLE	PID_FIXCYCLE 控制器。
BLINK	脉冲信号。
FREQ_MEASURE	布尔输入信号的测量频率。

GEN	周期性功能。
CHARCURVE	线性功能
RAMP_INT	限制供应功能（INT）减弱的上升。
RAMP_REAL	限制供应功能（REAL）减弱的上升。
HYSTERESIS	滞后作用。
LIMITALARM	注意输入值是否超出定义范围的限制。

4.3.1 BCD 转换

BCD_TO_INT

由库 util.library 提供。

该函数将 BCD 码转换成 INT 值。

函数的输入值是 BYTE 类型，输出值是 INT 类型。

如果输入值不是 BCD 码，输出值是-1。

ST 语言举例：

```
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，因为输入值不是 BCD 码格式 *)
```

INT_TO_BCD

由库 util.library 提供。

该函数将整数值转换成 BCD 码格式：

函数的输入值是 INT 类型，输出值是 BYTE 类型。

如果一个整数值不能转换为 BCD 码字节时，则输出 255。

ST 语言举例：

```
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 *)
```

4.3.2 位/字节算法功能块

EXTRACT

由库 util.library 提供。

该函数的输入 X 是 DWORD 类型，N 是 BYTE 类型。输出是 BOOL 类型的值，它包含输入 X 的第 N 位，函数从 X 的第 0 位开始算起。

ST 语言举例：

```
FLAG:=EXTRACT(X:=81, N:=4); (* 结果：TRUE，因为 81 的二进制数是 1010001，所以第 4 位是 1 *)
```

FLAG:=EXTRACT(X:=33, N:=0); (* 结果：TRUE，因为 33 的二进制数是 100001，所以第 0 位是 1 *)

PACK

由库 util.library 提供。

该函数是将 8 个布尔类型的输入位 B0, B1, ..., B7 整合为 1 个字节 。

UNPACK 功能块与该功能块密切相关。

PUTBIT

由库 util.library 提供。

该函数的输入由 DOWRD 类型的 X，BYTE 类型的 N，BOOL 类型的 B 组成。

PUTBIT 设置 X 中的第 N 位为 B，从 X 的第 0 位开始算起。

ST 语言举例：

A:=38;　　(* 二进制数 100110 *)

B:=PUTBIT(A,4,TRUE); (* 结果：54 = 2#110110 *)

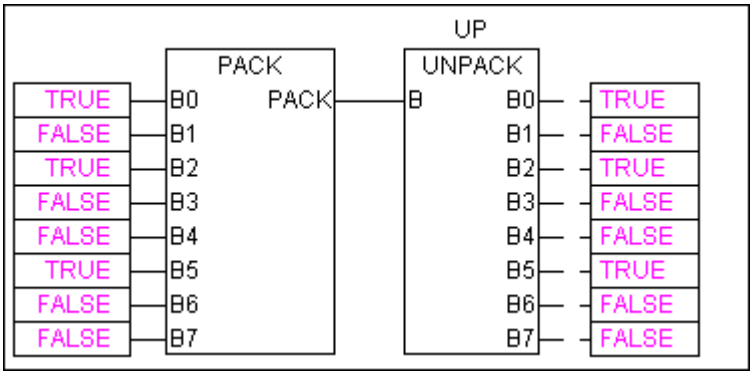
C:=PUTBIT(A,1,FALSE); (* 结果：36 = 2#100100 *)

UNPACK

由库 util.library 提供。

UNPACK 将输入 B 由 BYTE 类型转换为 8 个 BOOL 类型的输出变量 B0,...,B7，是 PACK 的逆操作。

FDB 语言举例：Output：



4.3.3 数学辅助功能块

DERIVATIVE

由库 util.library 提供。

该功能块近似求解微分。

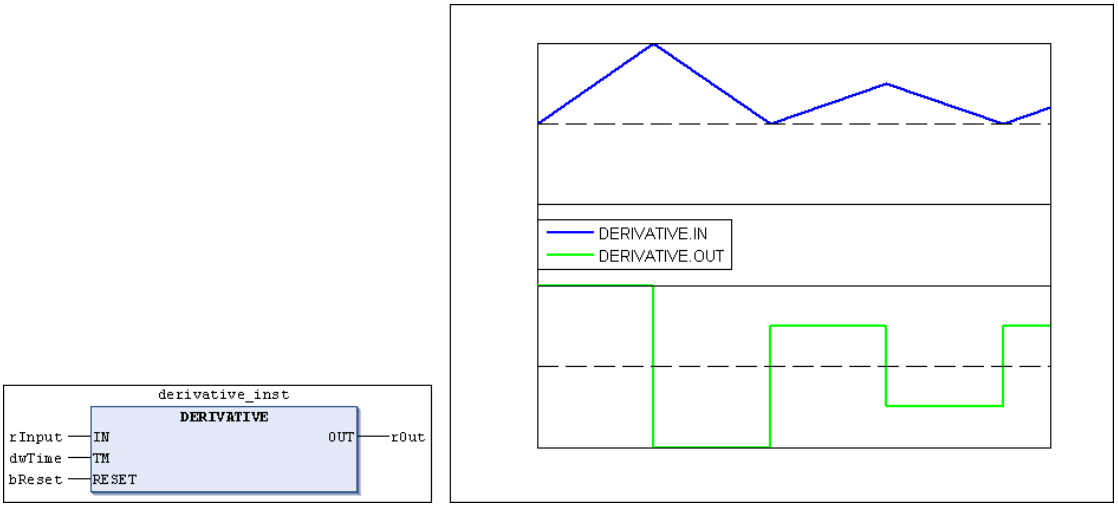
功能块 REAL 型输入变量 IN 是函数的值。TM 是 DWORD 类型，包含以毫秒为单位的时间，

RESET 是 BOOL 类型，其值为 TRUE 时，允许功能块重新启动。

输出 OUT 是 REAL 类型。

为了获得最好的结果，DERIVATIVE 近似使用最近的四个值，这样使输入参数不精确产生的误差尽可能小。

FBD 语言功能块：



INTEGRAL

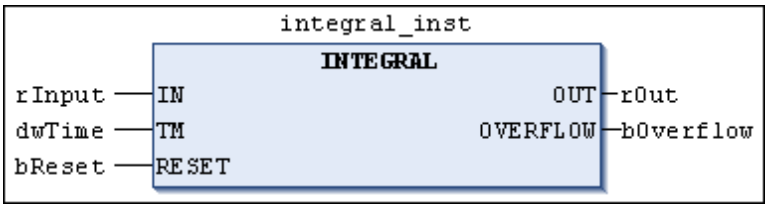
由库 util.library 提供。该功能块近似求解函数的积分。

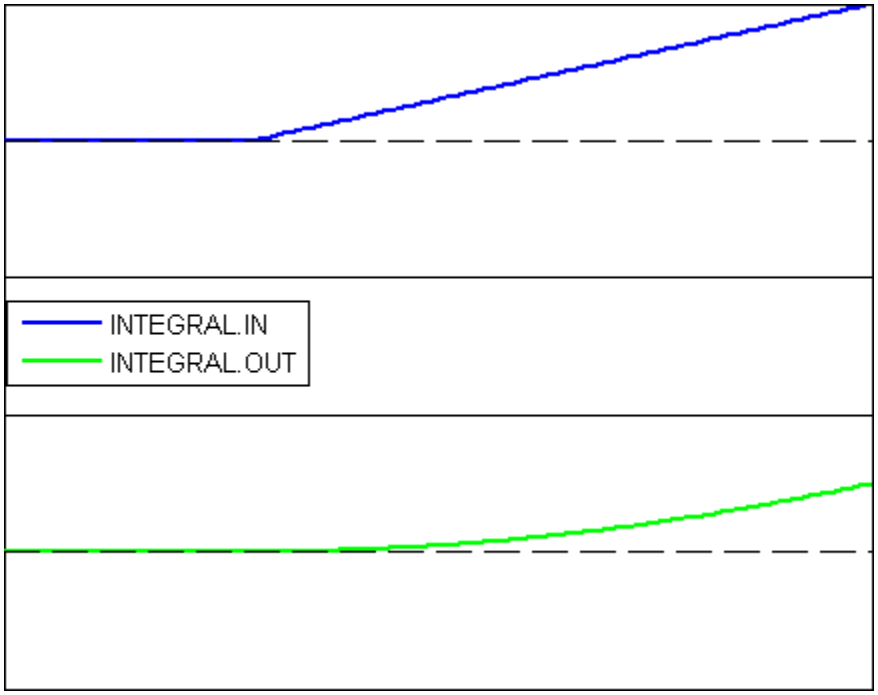
这是一个模拟量运算，REAL 型输入 IN 是被积函数的值。DWORD 类型的输入 TM 是以毫秒为单位的时间。RESET 是 BOOL 类型，其值为 TRUE 时，允许功能块重新启动。

输出 OUT 是 REAL 类型。

积分由两个函数近似求解。平均值作为近似积分提供。

FBD 语言功能块：例如：线性函数积分：

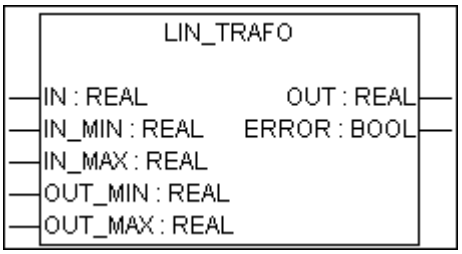




LIN_TRAFO

该功能块（util.library）将由下限和上限值确定的范围内的实数，转换为由另外下限和上限值确定的范围内的实数。下面的公式是转换的基础。

(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	输入值范围下限
IN_MAX	REAL	输入值范围上限
OUT_MIN	REAL	输出值范围下限
OUT_MAX	REAL	输出值范围上限

输出值：

变量	数据类型	描述
OUT	REAL	输出值
ERROR	BOOL	如果 IN_MIN=IN_MAX，或者输入值 IN 超出了设定的输入值范围，则发生错误，输出为 TRUE。

应用示例：

一个温度传感器提供电压值（输入值 IN）。这里需要转换为以摄氏度表示的温度值（输出值 OUT）。

输入（电压）值的范围由 IN_MIN=0 和 IN_MAX=10 来确定。输出（摄氏度）值范围由 OUT_MIN=-20 和 OUT_MAX=40 来确定。

因此输入 5V 电压，将得到温度值为 10 摄氏度。

STATISTICS_INT

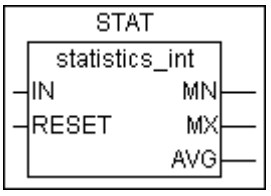
由库 util.library 提供。

该功能块计算一些标准的统计值：

输入 IN 是 INT 类型。RESET 是 BOOL 类型，其值为 TRUE 时，所有值重新初始化。

输出 MN、MX、AVG 分别代表 IN 的最小值、最大值和平均值（即 IN 的期望值）。三个输出都是 INT 类型。

FBD 语言功能块：



STATISTICS_REAL

由库 util.library 提供。

该功能块和 STATISTICS_INT 相似，只是输入 IN 和输出 MN, MX, AVG 都是 REAL 类型。

VARIANCE

由库 util.library 提供。

VARIANCE 用于计算输入值的方差。

输入 IN 是 REAL 类型，RESET 是 BOOL 类型，输出 OUT 也是 REAL 类型。

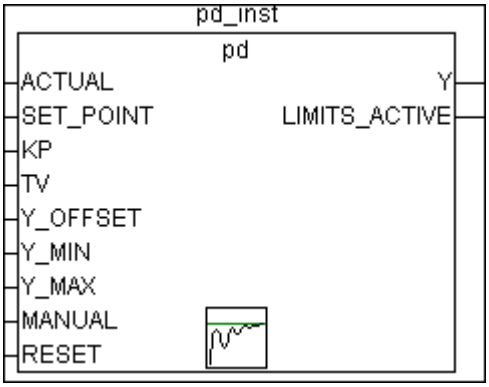
该功能块计算输入值的方差。VARIANCE 可以通过 RESET=TRUE 来复位。

标准偏差是 VARIANCE 的平方根。

4.3.4 调节器

PD

库 util.library 提供以下 PD 控制器功能块：



功能块输入：

变量	数据类型	描述
ACTUAL	REAL	被控变量的当前值
SET_POINT	REAL	期望值，命令变量
KP	REAL	比例系数，P 部分的单位增益
TV	REAL	微分时间，D 部分单位增益，以秒为单位，例如，“0.5” 是500毫秒
Y_MANUAL	REAL	定义 MANUAL = TRUE（手动模式下）的 Y 输出值
Y_OFFSET	REAL	操作变量 Y 的偏移量
Y_MIN, Y_MAX	REAL	操作变量 Y 的最小值和最大值。如果 Y 的值超限，输出 LIMITS_ACTIVE 将设置为 TRUE，同时 Y 将保持在规定的范围内。这个控制只有在 Y_MIN<Y_MAX 的情况下起作用。
MANUAL	BOOL	如果为 TRUE，手动操作将被激活，也就是说，操作值将由 Y_MANUAL 定义。
RESET		RESET 为 TRUE 时复位控制器； 在重新初始化时， Y = Y_OFFSET。

功能块输出：

变量	数据类型	描述
Y	REAL	操作值，由功能块计算得到（参见下面）
LIMITS_ACTIVE	BOOL	此值为 TRUE，表示 Y 已经超出了给定的范围(Y_MIN, Y_MAX)。

Y_OFFSET, Y_MIN 和 Y_MAX 用于在规定的范围之内转换操作值。

MANUAL 可以用于切换打开和关闭手动操作；RESET 用于重新初始化控制器。

在正常操作时（MANUAL = RESET = LIMITS_ACTIVE = FALSE），控制器计算控制器偏差，也就是 SET_POINT – ACTUAL 的差值，得到相对于时间的导数 de/dt ，并且在内部保存这些值。

输出值，也就是操作值 Y，按照以下公式计算：

Y = KP × (D + TV dD/dt) + Y_OFFSET 这里 D=SET_POINT-ACTUAL

所以除了 P 部分以外，当前控制器误差的改变（D 部分）也影响操作值。

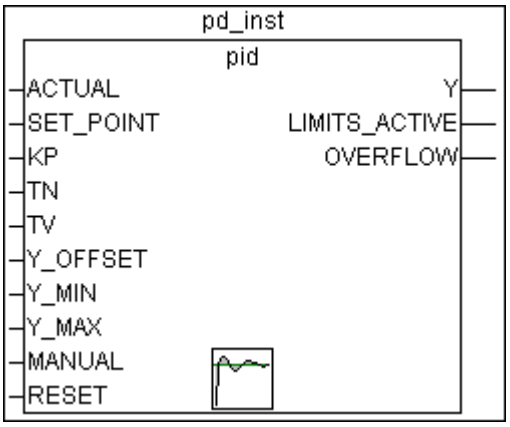
另外，Y 被由 Y_MIN 和 Y_MAX.规定的范围所限制。如果 Y 超出范围，LIMITS_ACTIVE 将为 TRUE。如果不希望限制操作值， Y_MIN 和 Y_MAX 必须设置为 0。

一旦 MANUAL=TRUE，Y_MANUAL 将赋给 Y。

通过将 TV 设置为 0，可以容易的创建一个 P 控制器。

PID

库 util.library 提供以下 PID 控制器功能块：



和PD控制器不同，该功能块包含一个REAL 类型输入TN，用于重新调整时间，单位是秒（例如，“0.5”是 500 毫秒）。

功能块输入：

变量	数据类型	描述
ACTUAL	REAL	被控变量的当前值
SET_POINT	REAL	期望值，命令变量
KP	REAL	比例系数，P 部分的单位增益
TN	REAL	复位时间，I 部分单位增益的倒数，以秒为单位，例如，“0.5”是500毫秒
TV	REAL	微分时间，D 部分单位增益，以秒为单位，例如，“0.5”是500毫秒
Y_MANUAL	REAL	定义 MANUAL = TRUE（手动模式下）的 Y 输出值
Y_OFFSET	REAL	操作变量 Y 的偏移量
Y_MIN, Y_MAX	REAL	操作变量 Y 的最小值和最大值。如果 Y 的值超限，输出 LIMITS_ACTIVE 将设置为 TRUE，同时 Y 将保持在规定的范围内。这个控制只有在 Y_MIN<Y_MAX 的情况下起作用。
MANUAL	BOOL	如果为 TRUE，手动操作将被激活，也就是说，操作值将由 Y_MANUAL 定义。
RESET	BOOL	RESET 为 TRUE 时复位控制器；在重新初始化时，Y = Y_OFFSET。

功能块输出：

变量	数据类型	描述
Y	REAL	操作值，由功能块计算得到（参见下面）
LIMITS_ACTIVE	BOOL	此值为 TRUE，表示 Y 已经超出了给定的范围 (Y_MIN, Y_MAX)。
OVERFLOW	BOOL	此值为 TRUE，表示溢出（参见下面）

Y_OFFSET, Y_MIN 和 Y_MAX 用于在规定的范围之内转换操作值。

MANUAL 可以用于切换到手动操作；RESET 用于重新初始化控制器。

在正常操作时（MANUAL = RESET = LIMITS_ACTIVE = FALSE），控制器计算控制器偏差，也就是 SET_POINT – ACTUAL 的差值，得到关于时间的导数 de/dt，并且在内部保存这些值。

和 PD 控制器不同的是，输出操作值 Y 中包含了积分部分，按照以下公式计算：

Y = KP × (D + 1/TN ?edt + TV dD/dt) + Y_OFFSET

所以除了 P 部分以外，当前控制器误差的改变（D 部分）和历史控制器误差（I 部分）都影响操作值。

通过将 TV 设置为 0，PID 控制器可以容易的转换为一个 PI 控制器。

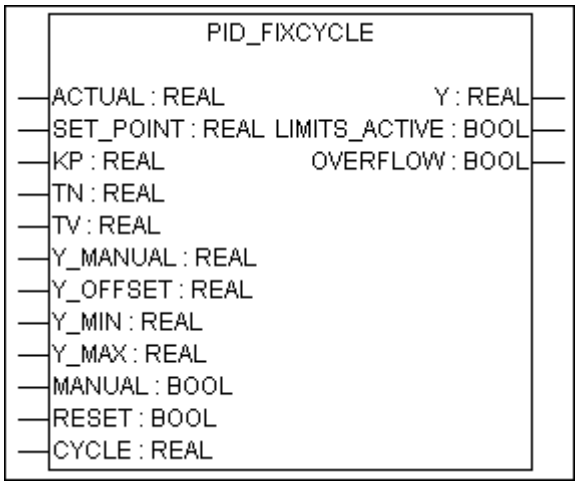
因为存在积分部分，如果误差 D 的积分变的过大，则可能由于控制器参数不正确而产生溢出。因此为了安全，提供一个 BOOL 型输出 OVERFLOW，在溢出时该值为 TRUE。这个仅仅发生在控制系统由于不正确的参数而不稳定时。此时，控制器将暂停，并且只有重新初始化才能再次运行。

注意：只要对操作量(Y_MIN u. Y_MAX)的限制有效，就会适应主要的部分，如同输入值的历史记录会自动影响对输出值的限制。如果不想使用该方式，可以采用下面的方案：关掉在 PID 控制器上的限制(Y_MIN>=Y_MAX)，并在输出值 Y 上取代 LIMIT 运算符（ICE 标准）的应用。

PID_FIXCYCLE

由库 util.library 提供。

PID_FIXCYCLE 控制器功能模块：



该功能模块的功能和 PID 控制器一致，区别是其周期时间由输入 CYCLE（以秒为单位）设置，而不是通过一个内部函数自动测量。

4.3.5 信号发生器

BLINK

由库 util.library 提供。

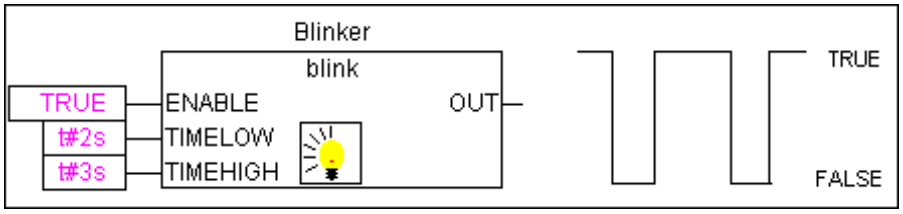
功能块 BLINK 产生脉冲信号。输入由 BOOL 类型 ENABLE，以及 TIME 类型 TIMELOW 和

TIMEHIGH 组成。输出 OUT 是 BOOL 类型。

如果 ENABLE 为 TRUE，在时间周期 TIMEHIGH，BLINK 设置输出为 TRUE；然后在时间周期 TIMELOW，设置输出为 FALSE。

当 ENABLE 复位为FALSE，输出 OUT 将保持不变，也就是说不再产生脉冲。如果在ENABLE 复位为FALSE 时，需要准确的使OUT 输出FALSE，可以使用“OUT 和ENABLE 相与”（也就是说，还要再增加一个“AND”函数）。

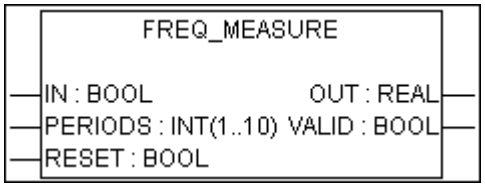
CFC 语言举例：



FREQ_MEASURE

由库 util.library 提供。

该功能块测量一个布尔类型输入信号的（平均）频率（单位 Hz）。可以设定对多少个周期进行平均。一个周期是指输入信号的两个上升沿之间的时间间隔。



输入变量：

变量	数据类型	描述
IN	BOOL	输入信号
PERIODS	INT	周期个数，一个周期是指需要计算平均频率的输入信号的两个上升沿之间的间隔。取值范围： 1~10。
RESET	BOOL	复位所有参数到0。

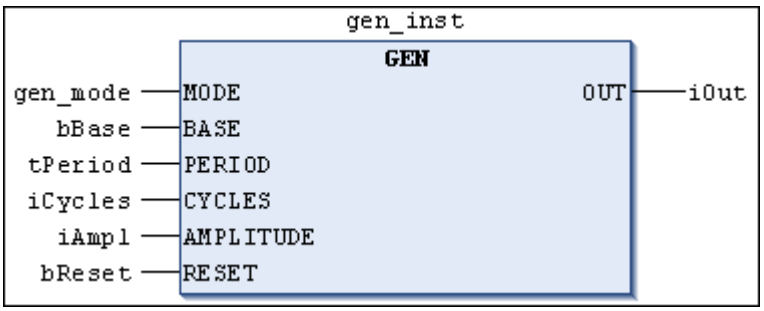
约定：

变量	数据类型	描述
OUT	REAL	结果频率，单位 Hz
VALID	BOOL	保持 FALSE 直到第一个测量结束，或者时间大于3倍的 OUT（表示输入有错误）。

GEN

由库 util.library 提供。

CFC 中的例子



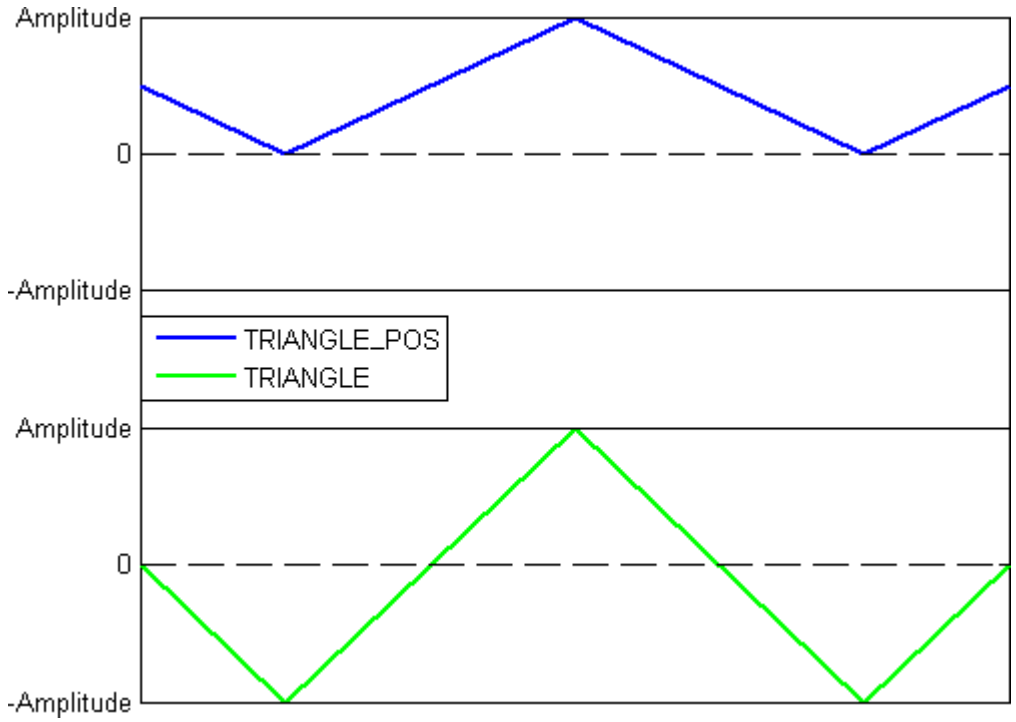
函数发生器产生典型的周期函数：

输入包括：预定义的 GEN_MODE 类型 MODE，BOOL 类型 BASE，TIME 类型 PERIOD，两个 INT 类型 CYCLES 和 AMPLITUDE，BOOL 类型 RESET。

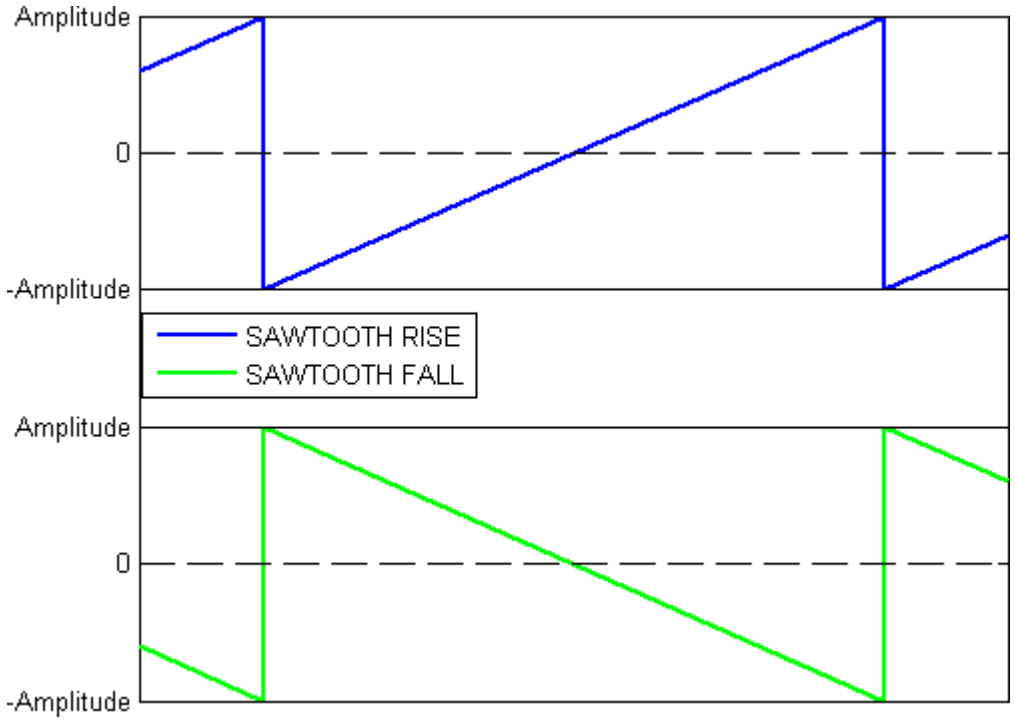
MODE 指定要产生的函数类型，可取的枚举值分别为：TRIANGLE 和 TRIANGLE_POS 生成三角函数， SAWTOOTH_RISE 生成上升锯齿函数， SAWTOOTH_FALL 生成下降锯齿函数，RECTANGLE 生成矩形函数，SINE 和 COSINE 分别生成正弦和余弦函数：

不同 MODE 类型的输出图示

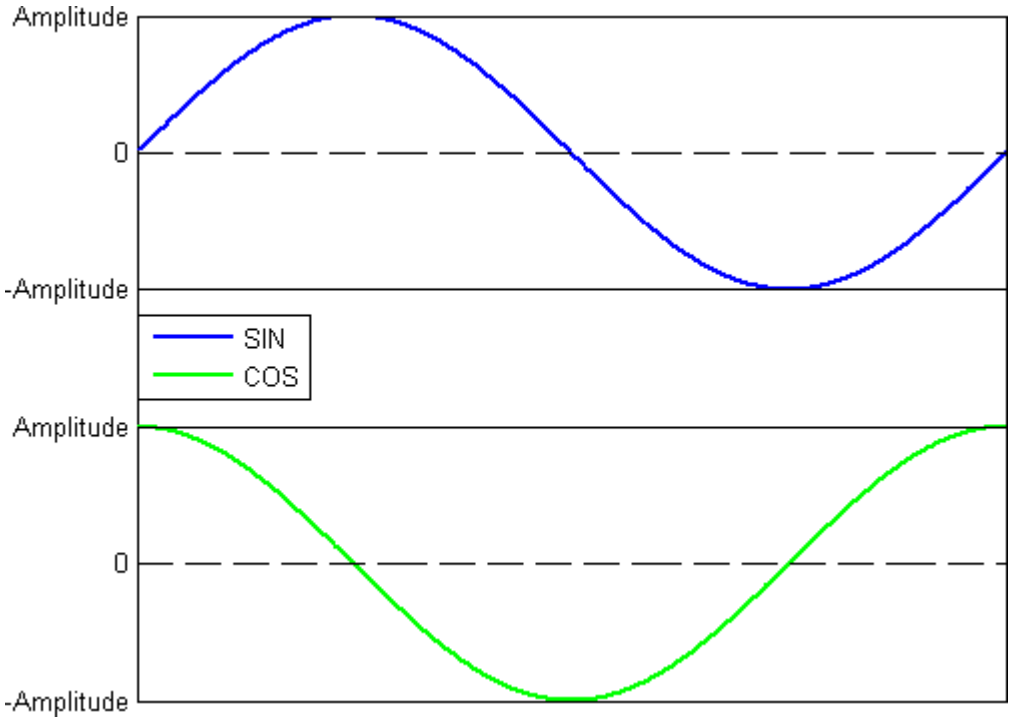
TRIANGLE / TRIANGLE_POS：



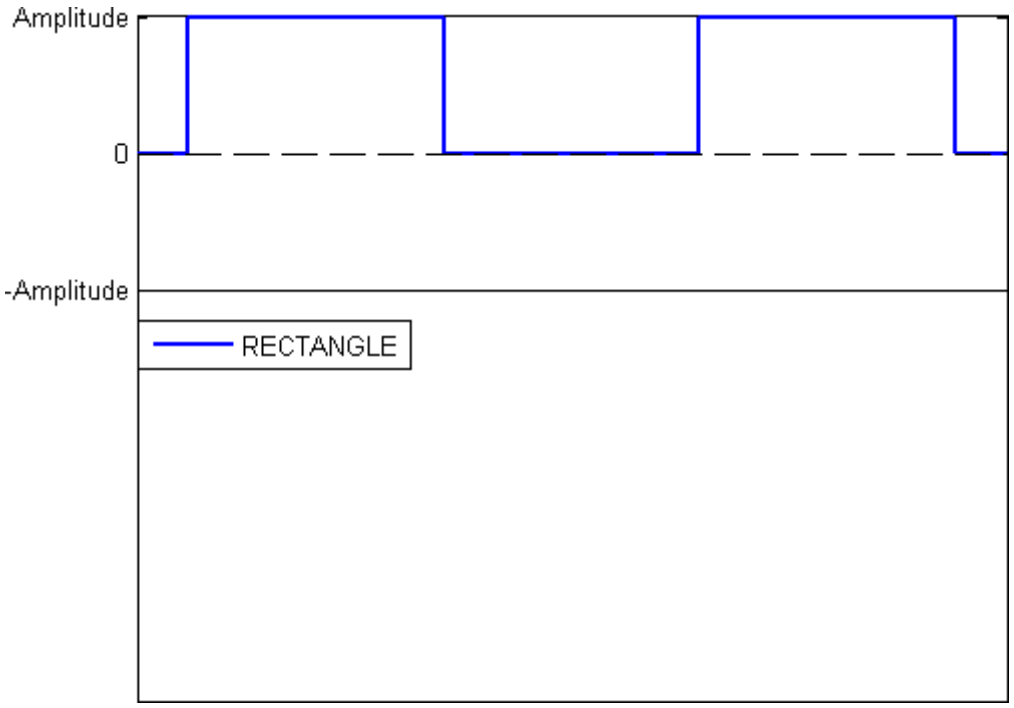
SAWTOOTH_RISE / SAWTOOTH FALL：



SINUS / COSINUS：



RECTANGLE：



BASE 定义循环周期是否真正和定义时间相关（BASE=TRUE），或者是否和指定的循环次数相关，即功能块调用次数（BASE=FALSE 时）。

PERIOD 或者 CYCLES 定义相应的循环周期。

AMPLITUDE 定义产生函数的振幅。

一旦 RESET=TRUE，函数发生器重新设置为 0。

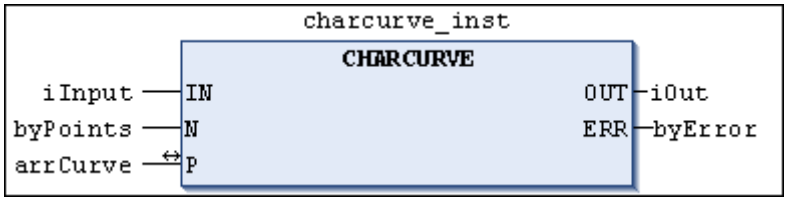
4.3.6 函数操纵功能块

CHARCURVE

由库 util.library 提供。

该功能块用于将给定的点列分段线性化。

FBD 中的例子



操作值 IN 是 INT 类型。字节 N 指定分段线性函数的点数。各个线性段由 POINT 型数组 P[0..10] 所代表的点列形成，POINT 类型变量是由两个 INT 值(X 和 Y)组成的一个结构，表示一个点。

输出由 INT 类型数据 OUT 和 BYTE 类型数据 ERR 组成，其中 OUT 是操作值，ERR 在必要时指出错误。

数组中的点 P[0]..P[N-1] 必须依照 X 值大小顺序存储，否则 ERR 为 1。如果输入 IN 不在 P[0].X 和 P[N-1].X 之内，ERR=2，OUT 包含相应的限制值 P[0]. Y 或者 P[N-1].Y。

如果 N 超出了 2~11 之间的允许值，ERR=4。

ST 语言举例：

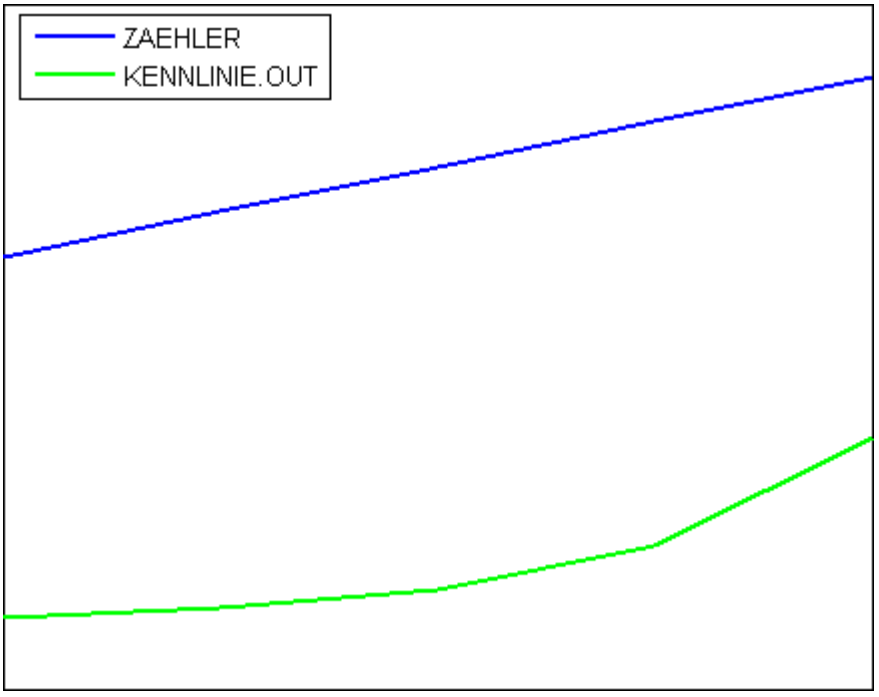
首先必须在头部定义数组 P：

```
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
```

然后用连续增加的值作为 CHARCURVE 的输入：

```
COUNTER:=COUNTER+10;
CHARACTERISTIC_LINE(IN:=COUNTER,N:=5,P:=KL);
```

随后的跟踪显示效果：



RAMP_INT

由库 util.library 提供。

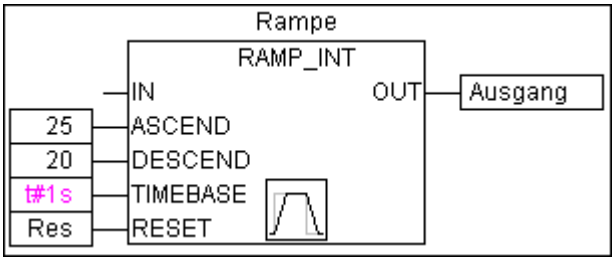
RAMP_INT 用于限制给定函数上升或者下降。

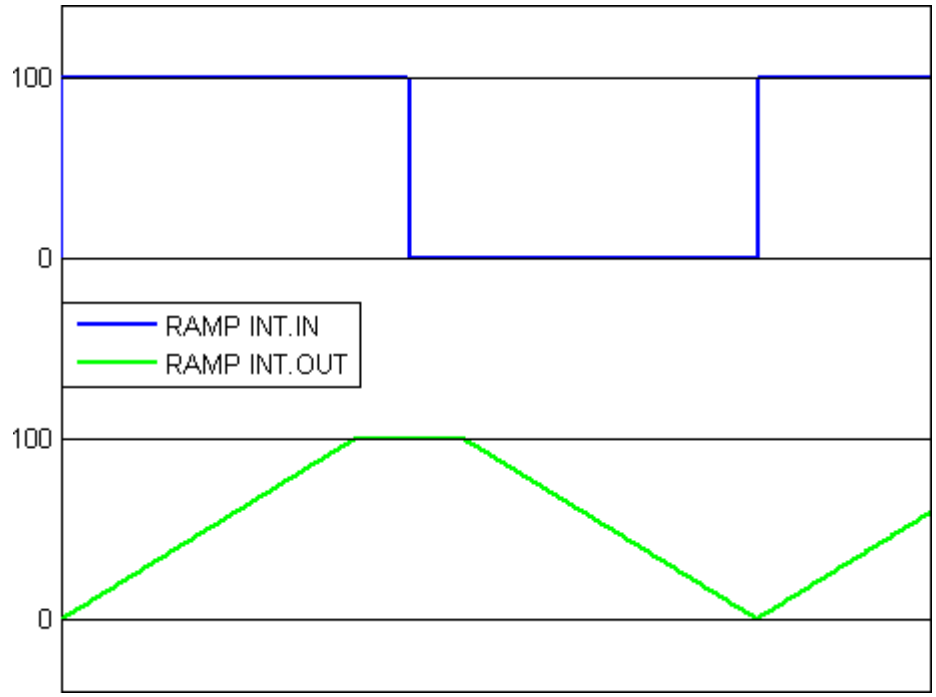
输入由以下三个 INT 类型数据组成：输入 IN、在给定时间间隔内的最大增加值 ASCEND 和最大减小值 DESCEND，该时间间隔由 TIME 类型的 TIMEBASE 来定义。RESET 设置为 TRUE，将使 RAMP_INT 重新初始化。

输出 OUT 是 INT 类型，包含上升和下降限制后的函数值。

当 TIMEBASE 设置为 t#0s 时，ASCEND 和 DESCEND 与时间间隔无关，而是保持不变。

CFC 语言举例：





RAMP_REAL

由库 util.library 提供。

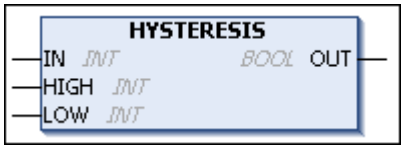
RAMP_REAL 函数和 RAMP_INT 相似，区别就是输入 IN、ASCEND、DESCEND 和输出 OUT 都是 REAL 类型。

4.3.7 模拟量监视功能块

HYSTERESIS

由库 util.library 提供。

FUP 中的 HYSTERESIS

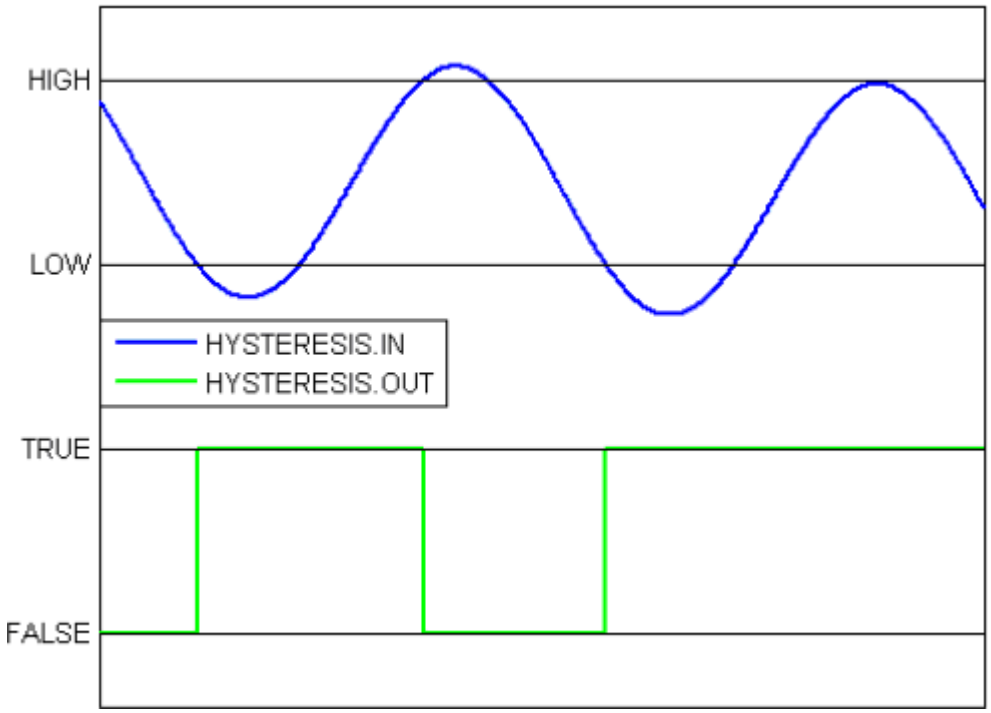


该功能块包括三个 INT 类型的输入 IN、HIGH 和 LOW。输出 OUT 是 BOOL 类型。

如果 IN 低于限值 LOW，OUT 变为TRUE。如果 IN 高于上限 HIGH，OUT 变为为 FALSE。

如果IN 降到下限LOW,OUT 将变为TRUE。下次运行IN 未超过上限前，输出将再次变为FALSE，直到 IN 再次降到 LOW，因此，OUT 再次获得 TRUE。

时间的函数 Hysteresis.IN 和 Hysteresis.OUT 的图解比较



LIMITALARM

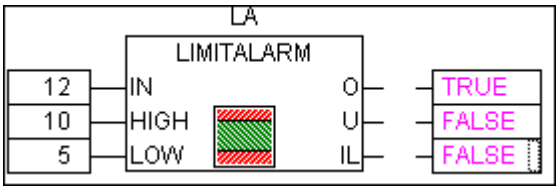
由库 util.library 提供。

该功能块规定当输入值超过给定范围时，输出为真。

输入 IN、HIGH 和 LOW 都是 INT 类型，输出 O、U 和 IL 都是 BOOL 类型。

如果 IN 高于 HIGH，O 为TRUE。当 IN 低于 LOW，U 为TRUE。IN 位于 LOW 和HIGH 之间时，IL 为 TRUE。

FBD 语言举例：Result:



5 SM3_Basic Library 库使用说明

SM3_Basic Library 库是 CODESYS SoftMotion 应用程序的基本库，因此它必须在 SoftMotion 工程中被包括。这将在一个 SoftMotion 设备插入时自动进行。

该库提供了以下功能块和函数：

- PLCopen 功能块既允许实现单轴运动控制，又允许实现两轴同步运动。除了状态检查、参数化和一般操作的库元素以外，还有以定义的速度、加速度参数来驱动轴的功能块。此外还有同步主、从轴和传动轴的模块。
- DriveInterface 基本驱动程序功能块（AXIS_REF_SM3, AXIS_REF_VIRTUAL_SM3, AXIS_REF_MAPPING_SM3）。
- 工具函数，比如文件服务和错误报告。
- 特定驱动器驱动程序的功能块或实现，基本驱动功能块的扩展。这些特殊驱动功能块（名为 AXIS_REF_*）通常被包括在一个单独的库中，该库被这个设备的相应设备描述所参考。

注意：所有运动生成功能块，或者一般来说所有具有 AXIS_REF 类型 VAR_IN_OUT 的功能块，在总线周期中都要被由 AXIS_REF_SM3 表示的驱动设备的任务所调用。

表 4-1 SM3_Basic.Library 库指令一览表

驱动接口
AXIS_REF_SM3
AXIS_REF_LOGICAL_SM3
SMC3_ReinitDrive
数据类型
基本驱动
SMC3_BrakeSetState
MC_CAM_REF
SMC_CAMTable_<variable type>_<number of elements>_1
SMC_CAMTable_<variable-type>_<number of elements>_2
SMC_CAMTappet
SMC_CAMTAPPETACTION
SMC_CAMTAPPETTYPE
SMC_CAMXYVA
MC_DIRECTION
SMC_CONTROLLER_MODE
SMC_HOMING_MODE
TRIGGER_REF
错误
SMC_ERROR
SMC_LANGUAGE_TYPE
PLCopen
MC_CAMSWITCH_REF
MC_CAMSWITCH_TR

MC_CAM_ID
MC_INT_STATUS
MC_OUTPUT_REF
MC_TAPPETMODE
MC_TA_REF
MC_TP_REF
MC_TRACK_REF
MC_TV_REF
SMC_TA
SMC_TAPPETDATA
SMC_TP
SMC_TV
Pous
基本驱动
SMC3_BrakeControl
SMC3_BrakeStatus
SMC_SetControllerMode
SMC_GetMaxSetAccDec
SMC_GetMaxSetVelocity
SMC_GetTrackingError
SMC_InPosition
SMC_MeasureDistance
SMC_CheckLimits
SMC_FollowPosition
SMC_FollowPositionVelocity
SMC_FollowSetValues
SMC_FollowVelocity
SMC_ClearFBError
SMC_ReadFBError
SMC_Homing
SMC_ChangeGearingRatio
错误
SMC_ErrorString
文件
SMC_ReadCAM
SMC_WriteCAM
SMC_AxisDiagnosticLog
PLCopen
SMC_BacklashCompensation
SMC_ReadSetPosition
SMC_SetTorque
SMC_CAMBounds
SMC_CAMBounds_Pos
SMC_CamRegister
SMC_CamEditor
SMC_GetCamSlaveSetPosition

SMC_GetTappetValue
MC_CamIn
MC_CamOut
MC_CamTableSelect
MC_GearIn
MC_GearInPos
MC_GearOut
MC_Phasing
MC_AccelerationProfile
MC_Halt
MC_Home
MC_MoveAbsolute
MC_MoveAdditive
MC_MoveRelative
MC_MoveSuperImposed
MC_MoveVelocity
MC_PositionProfile
MC_Power
MC_ReadActualPosition
MC_ReadAxisError
MC_ReadBoolParameter
MC_ReadStatus
MC_ReadParameter
MC_Reset
MC_Stop
MC_VelocityProfile
MC_WriteBoolParameter
MC_WriteParameter
MC_AbortTrigger
MC_DigitalCamSwitch
MC_ReadActualTorque
MC_ReadActualVelocity
MC_SetPosition
MC_TouchProbe
SMC_MoveContinuousAbsolute
SMC_MoveContinuousRelative
MC_Jog
SMC_Inch
Simple test
SMC_StartupDrive

5.1 驱动接口

AXIS_REF_SM3

这个驱动接口（由 SM3_Basic.library 提供）相对于 CoDeSys V2.3 不再是一个数据结构而是一个功能块。它由“SM3_Basic.library”库提供，并包括作为驱动器的工作。每一个 SoftMotion 轴都是 AXIS_REF_SM3 实例的一个扩展。

由于它们不需要被一般用户显示地使用，并不是 AXIS_REF_SM3 and AXIS_REF_*功能块的所有组件都在这里详细描述。

编号	名称	类型	初始值	描述
1000	nAxisState	SMC_AXIS_STATE (INT)	standstill	根据 PLCopen 状态图，轴的状态： 0：关机 1：错误停止 2：正在停止 3：待机 4：离散运动 5：连续运动 6：同步运动 7：自动导引
1012	bCommunication	BOOL	FALSE	TRUE：通讯正常
1013	wCommunicationState	WORD	16#FFFF	
1014	uiDriveInterfaceError	UINT	0	驱动接口错误号
1036	bDisableErrorLogging	BOOL	FALSE	不将错误记录到 fbeFBError
1040	bVirtual	BOOL	FALSE	TRUE：虚拟驱动；应用 AXIS_REF_VIRTUAL_SM3方法
1112,9	fMaxVelocity	LREAL	100	速度最大值，以“技术单位/秒”表示
1113	fSWMaxVelocity	LREAL	100	隐式动作的速度最大值，以“技术单位/秒”表示
1122,13	fMaxAcceleration	LREAL	100	加速度最大值，以“技术单位/秒²”表示
1123	fSWMaxAcceleration	LREAL	100	隐式运动的加速度最大值，以“技术单位/秒”表示
1132,15	fMaxDeceleration	LREAL	100	减加速度最大值，以“技术单位/秒²”表示
1133	fSWMaxDeceleration	LREAL	100	隐式运动的减加速度最大值，以“技术单位/秒”表示
1142,16	fMaxJerk	LREAL	100000	跃度最大值，以“技术单位/秒³”表示
1143	fSWMaxJerk	LREAL	100000	隐式运动的跃度最大值，以“技术单位/秒”表示
1152	fMaxCurrent	LREAL	100	当前（A）的最大值

1153	fSWMaxCurrent	LREAL	0	用户定义的当前（A）的最大值
1162	fMaxTorque	LREAL	0	最大转矩值，以“Nm”或“N（单位长度）”表示
1200,2	fSWLimitPositive	LREAL	0	在正方向上的位置限制，以“技术单位”表示
1201,3	fSWLimitNegative	LREAL	0	在负方向上的位置限制，以“技术单位”表示
1203	fSWLimitDeceleration	LREAL	0	减加速度[技术单位/秒²]，如果有软件限制开关，系统将以该值刹车。
1205	bSWLimitEnable	BOOL	FALSE	激活软件限制开关
1250	fSWErrorMaxDistance	LREAL	0	错误发生后，驱动到达停滞状态的距离
无参数编号的变量				
	strDriveInterfaceError	STRING	‘’	驱动接口错误的文字描述

AXIS_REF_LOGICAL_SM3

轴类型 AXIS_REF_LOGICAL_SM3 是主轴或编码器下的虚拟轴。此轴反映了主轴的动作，通过作为一个单独实例，在不影响主轴位置的情况下提供可执行的指令。如果 MC_SetPosition 在逻辑轴上执行，它不影响主轴的位置。

MC_TouchProbe 可能用作逻辑轴的前提条件是，接触式探头通道不能在主轴或其他主轴的逻辑轴上使用。逻辑轴未按照运动生成的 FBs(例如 MoveAbsolute)移动或其它除 MC_SetPosition, MC_TouchProbe or MC_AbortTrigger 之外的管理 FBs。

逻辑轴的状态是 synchronized_motion, 与主轴的状态独立。

	类型	初始值	描述
eValueReference	SMC_ValueSelect	按照配置程序的值进行设置。	依靠输入值(SMC_SET_VALUE)或主轴的实际值（SMC_ACT_VALUE）作为逻辑轴和移位补偿。
fDeadTimeCycle	LREAL/REAL	按照配置程序的值进行设置。	输入fDeadTimeCycles 定义了推测的周期数。
usiFilterDepthPosition	USINT	按照配置程序的值进行设置。	输入 usiFilterDepthPosition [0..9] 定义了用于低通滤波输入的位置值的数量。滤波器计算了最后位置移动的平均值，还计算了一个平滑位置。 usiFilterDepthVelocity=0：使用新位置的值作为 FilteredPosition.
usiFilterDepthVelocity	USINT	按照配置程序的值进行设置。	输入 usiFilterDepthVelocity [0..9] 定义了用于低通滤波输入的位置值的数量。滤波器计算了最后位置移动的平均值 [1..9]，还计算了一个平滑速度。 usiFilteusiFilterDepthVelocity=0：(位置-最后位置) / 滤波速度的周期

fModulo	LREAL	按照配置程序的值进行设置。	
iMovementType	INT	按照配置程序的值进行设置。	
wDriveID	WORD	按照配置程序的值进行设置。	
bManualShiftMode	BOOL	按照配置程序的值进行设置。	
fManualShiftDistance	LREAL	按照配置程序的值进行设置。	若 bManualShiftMode 为 TRUE，将要设置 fManualShiftDistance 的值。然后，按照轴计算位置。位置 = 主轴.滤波位置 + fManualShiftDistance

SMC3_ReinitDrive

意思是通过重新运行启动相，直到功能块将 bDone 设为 TRUE 应用才能控制驱动。

输入 bVirtual 决定了一个轴是否真的存在或是否是模拟的。如果将输入 bVirtual 设置为 TRUE，轴将被设置为虚拟模式。然后，它才将被一个仿真器（类似于虚拟驱动的设备）所代替。这对现场总线设备没有影响，将照常工作但不从实际设备那里接受或发送信息。这些对应 “SoftMotion Device Editor”中的设置。

假如连接了驱动设备，则认为 SMC3_ReinitDrive 不自动启动相关的现场总线。实际上，功能块在现场总线重启或在使用“SMC_ChangeGearingRatio”激活设置后可以被执行。

FUNCTION_BLOCK SMC3_ReinitDrive

组件	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF_SM3		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
bExecute	BOOL	FALSE	若为 TRUE，则将执行功能块。
bVirtual	BOOL	FALSE	若为 TRUE，轴将被设为虚拟模式。
VAR_OUTPUT			
bDone	BOOL	FALSE	若执行完成，则为 TRUE。
bBusy	BOOL	FALSE	若功能块未执行完成，则为 TRUE。
bError	BOOL	FALSE	在功能块内部发生错误的信号。
nErrorID	SMC_ERROR	0	错误代码

5.2 数据类型

5.2.1 基本驱动

SMC3_BrakeSetState

用于功能块 SMC3_BrakeControl，并决定了驱动如何设置机械制动器：

枚举值	类型	初始值	描述
SMC_BRAKE_AUTO	SMC3_BrakeSet	0	制动器的打开和关闭由驱动器自身处理。

	State		
SMC_BRAKE_OPEN	SMC3_BrakeSet State	1	制动器打开
SMC_BRAKE_CLOSE	SMC3_BrakeSet State	2	制动器关闭

MC_CAM_REF

这个数据类型是库“SM3_Basic.library”的一部分。它代表一个普通 CAM 并包含如下元素：

组件	类型	初始值	描述
wCamStructID	WORD	16#DC34	
byType	BYTE	0	0=多项式的，1=等距的，2=元素最优化的，非等距的，3= XYVA（由主设备位置，从设备位置，速度，加速度组成的特定点的多项式描述）
byVarType	BYTE	0	1：INT 2：UINT 3：DINT 4：UDINT 5：REAL 6：LREAL
xStart	LREAL		CAM 域。主设备的起始位置。
xEnd	LREAL		CAM 域。主设备的终止位置。
nElements	INT		元素个数，它依赖于从设备位置、主设备/从设备位置或 XYAV 点的类型编号。
nTappets	INT		开关动作的个数。
pce	POINTER TO BYTE	0	指向SMC_CAMElement 或SMC_CamTable 的指针。
pt	POINTER TO SMC_ CAMTappet	0	指向SMC_CAMElement 或SMC_CamTable 的指针。
dwTappetActiveBits	DWORD		内部变量。
strCAMName	STRING	''	CAM 名称。
byInterpolationQuality	BYTE	1	1 = 线性插补。 3 = 三次方插补。
bChangedOnline	BOOL	FALSE	内部变量。

SMC_CAMTable_<变量类型>_<元素数>_1

该数据结构描述等距曲线表，成员如下：

STRUCT SMC_CAMTable_<变量类型>_<元素数>_1

组件	类型	初始值	描述
Table	数组 [0..< 元素数>-1] OF 数组 [0..1] OF <变量类型>		从轴位置变量的数组，必须包含与主轴起始位置和终止位置对应的从轴位置

fEditorMasterMin	<变量类型>		存储于 SoftMotion 单元中的主轴位置变量，值域为[fEditorMasterMin, fEditorMasterMax]
fEditorMasterMax	<变量类型>		
fEditorSlaveMin	<变量类型>		存储于 SoftMotion 单元中的从轴位置变量，值域为[fEditorSlaveMin, fEditorSLaverMax]
fEditorSlaveMax	<变量类型>		
fTableMasterMin	<变量类型>		存储于 Table 单元中的主轴位置变量，值域为[fTableMasterMin, fTableMasterMax]
fTableMasterMax	<变量类型>		
fTableSlaveMin	<变量类型>		存储于 Table 单元中的从轴位置变量，值域为[fTableSlaveMin, fTableSlaveMax]
fTableSlaveMax:	<变量类型>		

SMC_CAMTable_<变量类型>_<元素数>_2

该数据结构描述非等距曲线表，成员如下：

STRUCT SMC_CAMTable_<变量类型>_<元素数>_2

组件	类型	初始值	描述
Table	数组 [0..< 元素数>-1] OF <变量类型>		主轴/从轴位置变量的数组，必须包含起始位置和终止位置。索引0参考主轴，索引1参考从轴。
fEditorMasterMin	<变量类型>		存储于 SoftMotion 单元中的主轴位置变量，值域为[fEditorMasterMin, fEditorMasterMax]。
fEditorMasterMax	<变量类型>		
fEditorSlaveMin	<变量类型>		存储于 SoftMotion 单元中的从轴位置变量，值域为[fEditorSlaveMin, fEditorSLaverMax]。
fEditorSlaveMax	<变量类型>		
fTableMasterMin	<变量类型>		存储于 Table 单元中的主轴位置变量，值域为[fTableMasterMin, fTableMasterMax]。
fTableMasterMax	<变量类型>		
fTableSlaveMin	<变量类型>		存储于 Table 单元中的从轴位置变量，值域为[fTableSlaveMin, fTableSlaveMax]。
fTableSlaveMax	<变量类型>		

SMC_CAMTappet

描述参考了 MC_CAM_REF 的 cam 表的挺杆。

组件	类型	初始值	描述
ctt	SMC_CAMTAPPETTYPE	TAPPET_pos	“SMC_CAMTAPPETTYPE”
cta	SMC_CAMTAPPETACTION	TAPPETACTION_on	“SMC_CAMTAPPETACTION”
dwDelay	DWORD	0	假设 cta = TAPPETACTION，此值将决定延迟时间（单位：μs）
dwDuration	DWORD	0	假设 cta = TAPPETACTION，此值将决定挺杆切换成 ON 的时间（单位：μs）。
iGroupID	INT	0	挺杆输出切换的组或轨道 ID
x	LREAL	0	挺杆切换时的主站位置
dwActive	DWORD	16#FFFFFFFF	内部变量

SMC_CAMTAPPETACTION

它确定了当 tappet 已被经过时，哪些开关动作被执行：

组件	类型	初始值	描述

TAPPETACTION_on	SMC_CAMTAPPETACTION	0	开启
TAPPETACTION_off	SMC_CAMTAPPETACTION	1	断开
TAPPETACTION_inv	SMC_CAMTAPPETACTION	2	相反
TAPPETACTION_time	SMC_CAMTAPPETACTION	3	在特定时间内开启

SMC_CAMTAPPETTYPE

该指令决定了挺杆动作激活时通过的方向。

组件	类型	初始值	描述
TAPPET_pos	SMC_CAMTAPPETTYPE	0	当主站经过正方向的位置时，挺杆动作激活。
TAPPET_all	SMC_CAMTAPPETTYPE	1	当主站经过两个方向（正/负方向）的位置时，挺杆动作激活。
TAPPET_neg	SMC_CAMTAPPETTYPE	2	当主站经过负方向的位置时，挺杆动作激活。

SMC_CAMXYVA

一个 XYVA-CAM 用 SMC_CAMXYVA 类型的数组变量代表，描述了每个 cam 点：

枚举值	类型	初始值	描述
dX	LREAL		主站位置
dY	LREAL		从站位置
dV	LREAL		一阶导数 dY/dX; 从站速度
dA	LREAL		二阶倒数 d^2Y/dX^2; 从站加速度

在最小值时，cam 的开始和结束点必须包括在数组中。

MC_DIRECTION

这个全局变量是库“SM3_Basic.library”的一部分。它用作几个功能块的输入并指定运动方向。请注意，不是所有的模式都能够应用于所有的功能块和轴类型（模数/有限）。

枚举值	类型	初始值	描述
fastest	MC_DIRECTION	3	自动选择方向以尽可能快地到达目标位置（仅模数轴）。
current	MC_DIRECTION	2	保持能够到达目标的实际方向（仅模数轴）
positive	MC_DIRECTION	1	在正方向上移动
shortest	MC_DIRECTION	0	按最短距离选择方向（仅模数轴）
negative	MC_DIRECTION	-1	在负方向上移动

SMC_CONTROLLER_MODE

这个数据类型是库“SM3_Basic.library”的一部分。它描述了电机的哪个量被控制。

枚举值	类型	初始值	描述
SMC_torque	SMC_CONTROLLER_MODE	1	力矩
SMC_velocity	SMC_CONTROLLER_MODE	2	速度
SMC_position	SMC_CONTROLLER_MODE	3	位置
SMC_current	SMC_CONTROLLER_MODE	4	当前模式

SMC_HOMING_MODE

这个全局变量是库“SM3_Basic.library”的一部分。在 SMC_Homing 使用时它定义了归航序列：

枚举值	类型	初始值	描述
FAST_BSLOW_S_STOP	SMC_HOMING_MODE	0	快速移动到参考开关；反向并慢速离开参考开关；执行“设置位置”；停止
FAST_BSLOW_STOP_S	SMC_HOMING_MODE	1	快速移动到参考开关；反向并慢速离开参考开关；停止；执行“设置位置”
FAST_BSLOW_I_S_STOP	SMC_HOMING_MODE	2	快速移动到参考开关；反向并慢速离开参考开关；等待标志脉冲；执行“设置位置”；停止
FAST_SLOW_S_STOP	SMC_HOMING_MODE	4	快速移动到参考开关；慢速离开参考开关；执行“设置位置”；停止
FAST_SLOW_STOP_S	SMC_HOMING_MODE	5	快速移动到参考开关；慢速离开参考开关；停止；执行“设置位置”
FAST_SLOW_I_S_STOP	SMC_HOMING_MODE	6	快速移动到参考开关；慢速离开参考开关；等待标志脉冲；执行“设置位置”；停止

TRIGGER_REF

这个数据类型是库“SM3_Basic.library”的一部分。它描述了一个触发输入。所使用的轴驱动程序决定了，哪个触发编号对应哪个硬件触发。

组件	类型	初始值	描述
iTriggerNumber	INT	-1	触发频道；由驱动程序定义（仅用 qhen bFastLatching=TRUE）
bFastLatching	BOOL	TRUE	TRUE：驱动设备闭锁已实行（精确） FALSE：闭锁在运动任务周期中已完成（不精确）
bInput	BOOL		当 bFastLatching=FALSE 时的触发信号
bActive	BOOL		内部变量

5.2.2 错误

SMC_ERROR

这个全局变量是库“SM3_Basic.library”的一部分。它包括 SoftMotion 功能块可能返回的所有错误号。

见：SMC_ErrorString 功能块生成错误字符串输出量。

错误编号	模块	枚举量	描述
0	所有	SMC_NO_ERROR	无错误
1	DriveInterface	SMC_DI_GENERAL_COMMUNICATION_ERROR	通讯错误(例如 e.g. Sercos 环断开)
2	DriveInterface	SMC_DI_AXIS_ERROR	轴错误
10	DriveInterface	SMC_DI_SWLIMITS_EXCEEDED	位置在允许值之外(SWLimit)
11	DriveInterface	SMC_DI_HWLIMITS_EX	硬件结束开关量激活

		CEEDED	
12	DriveInterface	SMC_DI_LINEAR_AXIS_OUTOFRANGE	轴的位置限制已被超过
13	DriveInterface	SMC_DI_HALT_OR_QUICKSTOP_NOT_SUPPORTED	不支持驱动设备状态 “Halt” 或 “Quickstop”
14	DriveInterface	SMC_DI_VOLTAGE_DISABLED	驱动设备没通电
15	DriveInterface	SMC_DI_IRREGULAR_ACTPOSITION	驱动给出的实际位置看起来是不合规则的，请检查通信。
16	DriveInterface	SMC_DI_POSITIONLAGERROR	位置延迟错误。设置位置和实际位置之间的误差超过了给定的限制。
20	all motion generating modules	SMC_REGULATOR_OR_START_NOT_SET	控制器不能执行或者正在刹车
21	All motion generating modules	SMC_WRONG_CONTROLLER_MODE	轴没有位于正确的控制模式。
30	DriveInterface	SMC_FB_WASNT_CALLED_DURING_MOTION	在运动结束前运动创建模块没有被再次调用。
31	All modules	SMC_AXIS_IS_NO_AXIS_REF	所给的 AXIS_REF 变量不示 AXIS_REF 类型。
32	All motion generating modules	SMC_AXIS_REF_CHANGED_DURING_OPERATION	在模块活动时，输入的 AXIS_REF 变量没有被交换。
33	DriveInterface	SMC_FB_ACTIVE_AXIS_DIABLED	在被移动时，轴失效 (MC_Power.bRegulatorOn)
34	All motion generating modules	SMC_AXIS_NOT_READY_FOR_MOTION	轴在当前状态不能执行运动命令。
40	VirtualDrive	SMC_VD_MAX_VELOCITY_EXCEEDED	超过最大速度(fMaxVelocity)
41	VirtualDrive	SMC_VD_MAX_ACCELERATION_EXCEEDED	超过最大加速度 (fMaxAcceleration)
42	VirtualDrive	SMC_VD_MAX_DECELERATION_EXCEEDED	超过最大减加速度 (fMaxDeceleration)
50	SMC_Homing	SMC_3SH_INVALID_VELACC_VALUES	无效的速度或加速度值
51	SMC_Homing	SMC_3SH_MODE_NEEDS_HWLIMIT	模式要求（出于安全原因）使用终止开关
70	SMC_SetControllerMode	SMC_SCM_NOT_SUPPORTED	模式不支持
71	SMC_SetControllerMode	SMC_SCM_AXIS_IN_WRONG_STATE	在当前模式下，不能改变控制器模式。
75	SMC_SetTorqu	SMC_ST_WRONG_CON	轴不在正确的模式下

	e	TROLLER_MODE	
80	SMC_ResetAxisGroup	SMC_RAG_ERROR_DURING_STARTUP	轴组启动错误
90	SMC_ChangeGearingRatio	SMC_CGR_ZERO_VALUES	无效值
91	SMC_ChangeGearingRatio	SMC_CGR_DRIVE_POWERED	驱动装置受控时传动参数可能被修改
92	SMC_ChangeGearingRatio	SMC_CGR_INVALID_POSPERIOD	无效位置周期(<=0)
110	MC_Power	SMC_P_FTASKCYCLE_EMPTY	轴不包含任何循环时间信息(fTaskCycle = 0)
120	MC_Reset	SMC_R_NO_ERROR_TO_RESET	无错误的轴
121	MC_Reset	SMC_R_DRIVE_DOESNT_ANSWER	轴不执行错误重置
122	MC_Reset	SMC_R_ERROR_NOT_RESETTABLE	错误不能被重置
123	MC_Reset	SMC_R_DRIVE_DOESNT_ANSWER_IN_TIME	与轴的通讯不起作用
130	MC_ReadParameter, MC_ReadBoolParameter	SMC_RP_PARAM_UNKNOWN	参数个数不知道
131	MC_ReadParameter, MC_ReadBoolParameter	SMC_RP_REQUESTING_ERROR	向驱动设备传递时出错；见功能块实例 ReadDriveParameter (SM_DriveBasic.lib)的错误号 。
140	MC_WriteParameter, MC_WriteBoolParameter	SMC_WP_PARAM_INVALID	参数个数未知或不允许写。
141	MC_WriteParameter, MC_WriteBoolParameter	SMC_WP_SENDING_ERROR	见模块实例 WriteDriveParameter (Drive_Basic.lib)的错误号 。
170	MC_Home	SMC_H_AXIS_WASNT_STANDSTILL	轴没有处于 standstill 状态
171	MC_Home	SMC_H_AXIS_DIDNT_START_HOMING	启动自动导引动作时出错。
172	MC_Home	SMC_H_AXIS_DIDNT_ANSWER	通讯错误。
173	MC_Home	SMC_H_ERROR_WHEN_STOPPING	自动导引停止后出错。减速设置？
180	MC_Stop	SMC_MS_UNKNOWN_STOPPING_ERROR	停止时未知错误
181	MC_Stop	SMC_MS_INVALID_ACC	无效速度或加速度值

		DEC_VALUES	
182	MC_Stop	SMC_MS_DIRECTION_NOT_APPLICABLE	direction=shortest 不适用
183	MC_Stop	SMC_MS_AXIS_IN_ERRORSTOP	驱动装置处于 errorstop 状态。停止不能被执行。
184	MC_Stop	SMC_BLOCKING_MC_STOP_WASNT_CALLED	阻塞轴(Execute=TRUE)MC_Stop 的一个实例，还未调用。请调用 MC_Stop(Execute=FALSE)。
201	MC_MoveAbsolute	SMC_MA_INVALID_VELACC_VALUES	速度或加速度值无效
202	MC_MoveAbsolute	SMC_MA_INVALID_DIRECTION	方向错误
226	MC_MoveRelative	SMC_MR_INVALID_VELACC_VALUES	速度或加速度值无效
227	MC_MoveRelative	SMC_MR_INVALID_DIRECTION	方向错误
251	MC_MoveAdditive	SMC_MAD_INVALID_VELACC_VALUES	速度或加速度值无效
252	MC_MoveAdditive	SMC_MAD_INVALID_DIRECTION	方向错误
276	MC_MoveSuperImposed	SMC_MSI_INVALID_VELACC_VALUES	速度或加速度值无效
277	MC_MoveSuperImposed	SMC_MSI_INVALID_DIRECTION	方向错误
301	MC_MoveVelocity	SMC_MV_INVALID_ACCDEC_VALUES	速度或加速度值无效
302	MC_MoveVelocity	SMC_MV_DIRECTION_NOT_APPLICABLE	Direction=shortest/fastest 不适用
325	MC_PositionProfile	SMC_PP_ARRAYSIZE	错误的数组大小
326	MC_PositionProfile	SMC_PP_STEP0MS	时间步长 = t#0s
350	MC_VelocityProfile	SMC_VP_ARRAYSIZE	错误的数组大小
351	MC_VelocityProfile	SMC_VP_STEP0MS	时间步长 = t#0s
375	MC_AccelerationProfile	SMC_AP_ARRAYSIZE	错误的数组大小
376	MC_AccelerationProfile	SMC_AP_STEP0MS	时间步长 = t#0s
400	MC_TouchProbe	SMC_TP_TRIGGEROCCUPIED	触发已经激活
401	MC_TouchProbe	SMC_TP_COULDNT_SET_WINDOW	DriveInterface 不支持时间函数
402	MC_TouchPro	SMC_TP_COMM_ERRO	通讯错误

	be	R	
410	MC_AbortTrigger	SMC_AT_TRIGGERNOTOCCUPIED	触发已分配
426	SMC_MoveContinuousRelative	SMC_MCR_INVALID_VELACC_VALUES	速度或加速度值无效
427	SMC_MoveContinuousRelative	SMC_MCR_INVALID_DIRECTION	无效的方向
451	SMC_MoveContinuousAbsolute	SMC_MCA_INVALID_VELACC_VALUES	速度或加速度值无效
452	SMC_MoveContinuousAbsolute	SMC_MCA_INVALID_DIRECTION	无效的方向
453	SMC_MoveContinuousAbsolute	SMC_MCA_DIRECTION_NOT_APPLICABLE	Direction= fastest 不适用
600	SMC_CamRegister	SMC_CR_NO_TAPPETS_IN_CAM	CAM 不包含任何 tappets
601	SMC_CamRegister	SMC_CR_TOO_MANY_TAPPETS	Tappet 组 ID 超过 MAX_ NUM_ TAPPETS
602	SMC_CamRegister	SMC_CR_MORE_THAN_32_ACCESSES	在一 CAM_REF 上有超过32个连接
625	MC_CamIN	SMC_CI_NO_CAM_SELECTED	没有 CAM 被选择
626	MC_CamIN	SMC_CI_MASTER_OUT_OF_SCALE	主轴在有效区域之外
627	MC_CamIN	SMC_CI_RAMPIN_NEEDS_VELACC_VALUES	对 ramp_in 函数，速度和加速度值必须被指定。
628	MC_CamIN	SMC_CI_SCALING_INCORRECT	缩放变量 fEditor/ TableMasterMin/ Max 不正确
640	SMC_CAMBounds, SMC_CamBounds_Pos	SMC_CB_NOT_IMPLEMENTED	以 CAM 格式给出的功能块没有实现
675	MC_GearIn	SMC_GI_RATIO_DENOM	比值分母 = 0
676	MC_GearIn	SMC_GI_INVALID_ACC	加速度无效
677	MC_GearIn	SMC_GI_INVALID_DEC	减加速度无效
725	MC_Phase	SMC_PH_INVALID_VELACCDEC	速度、减加速度或加速度无效
726	MC_Phase	SMC_PH_ROTARYAXIS_PERIOD0	旋转轴 fPositionPeriod = 0
750	All modules using	SMC_NO_CAM_REF_TYPE	所给的CAM 不是 MC_CAM_REF 类型的

	MC_CAM_REF as input		
775	MC_GearInPos	SMC_GIP_MASTER_DIRECTION_CHANGE	在与从轴连接期间，主轴改变了旋转方向
800	SMC_Backlash Compensation	SMC_BC_BL_TOO_BIG	齿轮间隙（fBacklash）太大（>位置回合/2）
1001	SMC_Interpolator	SMC_INT_VEL_ZERO	轨迹未被处理，因为设置速度＝0
1002	SMC_Interpolator	SMC_INT_NO_STOP_AT_END	上一个轨迹对象具有 Vel_End > 0。
1003	SMC_Interpolator	SMC_INT_DATA_UNDERRUN	警告：在 DataIn 中处理了GEOINFO 表，但是表尾没有设置。原因：忘记设置在 DataIn 中的队列的 EndOfList，或者 SMC_Interpolator 比轨迹生成模块要快。
1004	SMC_Interpolator	SMC_INT_VEL_NONZERO_AT_STOP	停止时速度>0。
1005	SMC_Interpolator	SMC_INT_TOO_MANY_RECURSIONS	太多 SMC_Interpolator 递归。SoftMotion 错误。
1006	SMC_Interpolator	SMC_INT_NO_CHECKVELOCITIES	Input-OutQueue DataIn 没有最后处理 SMC_CHeckVelocities 模块。
1007	SMC_Interpolator	SMC_INT_PATH_EXCEEDED	内部/数字错误。
1008	SMC_Interpolator	SMC_INT_VEL_ACC_DEC_ZERO	速度、加速度或减加速度值为空或太低。
1009	SMC_Interpolator	SMC_INT_DWIPOTIME_ZERO	以 dwIpoTime = 0调用 FB
1050	SMC_Interpolator2Dir	SMC_INT2DIR_BUFFER_TOO_SMALL	数据缓冲区太小
1051	SMC_Interpolator2Dir	SMC_INT2DIR_PATH_FITS_NOT_IN_QUEUE	轨迹在队列中没有运行完。
1080	SMC_Interpolator	SMC_WAR_INT_OUTQUEUE_TOO_SMALL	警告：OutQueueDataIn 容量太小无法保证不停止。
1081	SMC_Interpolator	SMC_WAR_END_VELOCITIES_INCORRECT	警告：终止速度不一致
1100	SMC_CheckVelocities	SMC_CV_ACC_DEC_VEL_NONPOSITIVE	不允许的速度、加速度或减加速度值
1200	SMC_NCDecoder	SMC_DEC_ACC_TOO_LITTLE	不允许的加速度值
1201	SMC_NCDecoder	SMC_DEC_RET_TOO_LITTLE	不允许的加速度值
1202	SMC_NCDecoder	SMC_DEC_OUTQUEUE_RAN_EMPTY	数据低载运行。队列已读但为空。
1203	SMC_NCDecoder	SMC_DEC_JUMP_TO_UNKNOWN_LINE	跳过一行没有执行因为行号未知。

1204	SMC_NCDecoder	SMC_DEC_INVALID_SYNTAX	无效的语法。
1205	SMC_NCDecoder	SMC_DEC_3DMODE_OBJECT_NOT_SUPPORTED	对象不应在3D 模式。
1300	SMC_GCodeViewer	SMC_GCV_BUFFER_TOO_SMALL	缓冲区过小
1301	SMC_GCodeViewer	SMC_GCV_BUFFER_WRONG_TYPE	缓冲元素类型错误
1302	SMC_GCodeViewer	SMC_GCV_UNKNOWN_IPO_LINE	插补器的实际行找不到
1500	All function blocks using SMC_CNC_REF	SMC_NO_CNC_REF_TYPE	给出的 CNC 程序不是 SMC_CNC_REF 类型的
1501	All function blocks using SMC_OUTQUEUE	SMC_NO_OUTQUEUE_TYPE	给出的 OutQueue 不是 SMC_OUTQUEUE 类型的
1600	CNC function blocks	SMC_3D_MODE_NOT_SUPPORTED	这些功能块只能在2D 路径下工作。
2000	SMC_ReadNCFile	SMC_RNCF_FILE_DOESNT_EXIST	文件不存在
2001	SMC_ReadNCFile	SMC_RNCF_NO_BUFFER	无缓冲区被分配
2002	SMC_ReadNCFile	SMC_RNCF_BUFFER_TOO_SMALL	缓冲区太小
2003	SMC_ReadNCFile	SMC_RNCF_DATA_UNDERRUN	数据低载运行，缓冲区已读但为空。
2004	SMC_ReadNCFile	SMC_RNCF_VAR_COULDNT_BE_REPLACED	占位符变量不能被替换。
2005	SMC_ReadNCFile	SMC_RNCF_NOT_VARLIST	输入 pvl 不指向 SMC_VARLIST 对象。
2050	SMC_ReadNCQueue	SMC_RNCQ_FILE_DOESNT_EXIST	文件不能打开。
2051	SMC_ReadNCQueue	SMC_RNCQ_NO_BUFFER	无缓冲区被定义
2052	SMC_ReadNCQueue	SMC_RNCQ_BUFFER_TOO_SMALL	缓冲区太小
2053	SMC_ReadNCQueue	SMC_RNCQ_UNEXPECTED_EOF	文件异常结束
2100	SMC_AxisDiagnosticLog	SMC_ADL_FILE_CANNOT_BE_OPENED	文件不能打开
2101	SMC_AxisDiagnosticLog	SMC_ADL_BUFFER_OVERRUN	缓冲区过载；WriteToFile 一定是被过于频繁地调用

2200	SMC_ReadCAM	SMC_RCAM_FILE_DOESNT_EXIST	文件不能打开
2201	SMC_ReadCAM	SMC_RCAM_TOO_MUCH_DATA	要存储地 CAM 过大
2202	SMC_ReadCAM	SMC_RCAM_WRONG_COMPILE_TYPE	错误地编译模式
2203	SMC_ReadCAM	SMC_RCAM_WRONG_VERSION	文件版本错误
2204	SMC_ReadCAM	SMC_RCAM_UNEXPECTED_EOF	文件异常结束
3001	SMC_WriteDriveParamsToFile	SMC_WDPF_CHANNEL_OCCUPIED	SMC_WDPF_TIMEOUT_PREPARING_LIST
3002	SMC_WriteDriveParamsToFile	SMC_WDPF_CANNOT_CREATE_FILE	文件不能被创建
3003	SMC_WriteDriveParamsToFile	SMC_WDPF_ERROR_WHEN_READING_PARAMS	读取参数出错
3004	SMC_WriteDriveParamsToFile	SMC_WDPF_TIMEOUT_PREPARING_LIST	准备参数表时超时
5000	SMC_Encoder	SMC_ENC_DENOM_ZERO	编码参考的转变因数(dwRatioTechUnitsDenom)分母为0。
5001	SMC_Encoder	SMC_ENC_AXISUSEDBYOTHERFB	其他模块尝试在编码器轴上处理运动。
5002	DriveInterface	SMC_ENC_FILTER_DEPTH_INVALID	无效的过滤器深度。

SMC_LANGUAGE_TYPE

这个全局变量是库“SM3_Basic.library”的一部分。它用于函数 SMC_ErrorString 以确定语言。

枚举值	类型	初始值	描述
english	SMC_LANGUAGE_TYPE	0	英语
deutsch	SMC_LANGUAGE_TYPE	1	德语

5.2.3 PLCopen

MC_CAMSWITCH_REF

这个数据类型是库“SM3_Basic.library”的一部分。它描述了 CAMSwitches 表并用作 MC_DigitalCAMSwitch 的输入。

组件	类型	初始值	描述
NoOfSwitches	BYTE		切换位置的编号。
Periodic	POINTER TO MC_CAMSWITCH_TR	FALSE	见 MC_CAMSWITCH_TR

MC_CAMSWITCH_TR

这个数据结构是库“SM3_Basic.library”的一部分。它描述了随行件的切换位置。

组件	类型	初始值	描述
TrackNumber	INT		输出编号；每次输出几个位置范围是可能的。
FirstOnPosition	LREAL		输出的接通位置
LastOnPosition	LREAL		输出的断开位置
AxisDirection	INT		0＝输出将被转变为两个方向，1＝输出只被转变为正方向，-1＝输出只被转变为负方向
CamSwitchMode	INT		0＝随行件被基于位置计算；1＝随行件被基于时间计算，这时只有 FirstOnPosition 的值将被使用，而且对于给定的时间（Duration）输出一直为 TRUE。
Duration	TIME		时间周期，在 CAMSwitchMode=1时随行件输出将保持为 TRUE。

MC_CAM_ID

这个数据类型是库“SM3_Basic.library”的一部分。它是一个描述 CAM 工作台的内部数据结构。它由 MC_CAMTableSelect 生成并做为 MC_CamIn 的输入。

SMC_INT_STATUS

这个枚举是库“SM3_Basic.library”的一部分。它描述插值器实例的状态。

枚举值	类型	初始值	描述
IPO_UNKNOWN	SMC_INT_STATUS	0	内部状态
IPO_INIT	SMC_INT_STATUS	1	初始状态；运动尚未开始
IPO_ACCEL	SMC_INT_STATUS	2	进行插补；当前正在加速
IPO_CONSTANT	SMC_INT_STATUS	3	进行插补；当前速度恒定
IPO_DECEL	SMC_INT_STATUS	4	进行插补；当前正在减速
IPO_FINISHED	SMC_INT_STATUS	5	插补完成
IPO_WAIT	SMC_INT_STATUS	6	进行插补；当前正在等待

MC_OUTPUT_REF

这个数据类型是库“SM3_Basic.library”的一部分。它定义了 32 个布尔数组的形式的挺杆输出。

TYPE MC_OUTPUT_REF :

ARRAY [0..31] OF BOOL;

END_TYPE

MC_TAPPETMODE

这个数据类型是库“SM3_Basic.library”的一部分。它用于功能块 MC_DigitalCAMSwitch 以确定 tappet 值应参考主轴设定点（1），还是实际位置（2）。在自动检测（0）情况下，功能块的判断依赖于：驱动的控制状态是使用了设定值（bRegulatorOn = TRUE）还是实际值（bRegulatorOn = FALSE）。

组件	类型	初始值	描述

tp_mode_auto	MC_TAPPETMODE	0	自动模式
tp_mode_demandposition	MC_TAPPETMODE	1	使用设定值
tp_mode_actualposition	MC_TAPPETMODE	2	使用实际值

MC_TA_REF

这个数据类型是库“SM3_Basic.library”的一部分。它描述了一个以后将被 MC_AccelerationProfile 执行的轨迹。

组件	类型	初始值	描述
Number_of_pairs	INT	0	分布点编号
IsAbsolute	BOOL	TRUE	加速度必须被解释为绝对值或相对值
MC_TA_Array	ARRAY[1..0] OF SMC_TA		时间/加速度点

MC_TP_REF

这个数据类型是库“SM3_Basic.library”的一部分。它描述了一个以后将被 MC_PositionProfile 执行的轨迹。

组件	类型	初始值	描述
Number_of_pairs	INT	0	分布点编号
IsAbsolute	BOOL	TRUE	位置值必须被解释为绝对值或相对值
MC_TP_Array	ARRAY[1..100] OF SMC_TP		时间/位置点

MC_TRACK_REF

这个数据类型是库“SM3_Basic.library”的一部分。

组件	类型	初始值	描述
OnCompensation	LREAL	0	延迟（正值）或者提前接通（负值）可被设置。时间用秒给出。
OffCompensation	LREAL	0	延迟（正值）或者提前接通（负值）可被设置。时间用秒给出。
Hysteresis	LREAL	0	可设置一个附加迟滞，这将避免输出的持续开关，这种情况是可能发生的，比如如果伺服机控制器由于位置控制正好在开关位置稍稍回转了一点。

MC_TV_REF

这个数据类型是库“SM3_Basic.library”的一部分。它描述了一个以后将被 MC_VelocityProfile 执行的轨迹。

组件	类型	初始值	描述
Number_of_pairs	INT	0	分布速度点的编号。
IsAbsolute	BOOL	TRUE	速度值是绝对值或相对值。
MC_TV_Array	ARRAY[1..0] OF MC_TV		时间/速度点。

SMC_TA

这个数据类型是库“SM3_Basic.library”的一部分。它定义了轨迹上的一个点，该点由 MC_TA_REF

定义并由一对时间/加速度值组成。

组件	类型	初始值	描述
delta_time	TIME	TIME#0ms	到达上一个和当前点之间的时间周期。
acceleration	LREAL	0	在这点上的（绝对-/相对-）加速度

SMC_TAPPETDATA

这个数据类型是库“SM3_Basic.library”的一部分。它是一个内部数据结构并做为 MC_CAMIn 和 SMC_GetTappetValue 之间的接口。

SMC_TP

这个结构是库“SM3_Basic.library”的一部分。它定义了轨迹上的一个点，该点由 MC_TP_REF 定义并由一对时间/位置值组成。

组件	类型	初始值	描述
delta_time	TIME	TIME#0ms	到达上一个点和当前点的时间周期。
position	LREAL	0	在这点上的（绝对-/相对-）位置

SMC_TV

这个数据类型是库“SM3_Basic.library”的一部分它定义了轨迹上的一个点，该点由 MC_TP_REF 定义并由一对时间/速度值组成。

组件	类型	初始值	描述
delta_time	TIME	TIME#0ms	到达上一个点和当前点的时间周期。
velocity	LREAL	0	在这点上的（绝对-/相对-）速度

5.3 POUs

5.3.1 基本驱动

SMC3_BRAKECONTROL

这个功能块是库“SM3_Basic.library”的一部分。如果被驱动装置和它的 SoftMotion 驱动程序支持，该功能块确定了机械刹车制动器的行为。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
bExecute	BOOL	FALSE	当为 TRUE 时，功能块是活动的。
eSetBrake	SMC3_BrakeSetState	SMC_BRAKE_AUTO	希望的刹车模式，见 SMC_BrakeSetState。
VAR_OUTPUT			
bDone	BOOL	FALSE	如果参数可用则为 TRUE。
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
bError	BOOL	FALSE	在功能块内部发生错误的信号。
nErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

SMC3_BRAKESTATUS

这个功能块是库“SM3_Basic.library”的一部分。它读取机械制动器的状态。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
bEnable	BOOL	FALSE	必须设为 TRUE 以使得功能块能够执行。
eSetBrake	SMC3_BrakeSetState	SMC_BRAKE_AUTO	希望的刹车模式，见 SMC_BrakeSetState
VAR_OUTPUT			
bValid	BOOL	FALSE	如果参数可用则为 TRUE。
bBusy	BOOL	FALSE	如果参数可用则为 TRUE。
bError	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
nErrorID	SMC_ERROR	0	在功能块内部发生错误的信号。
bBrakeClosed	BOOL		错误指示，见 SMC_Error。

SMC_SETCONTROLLERMODE

这个功能块是库“SM3_Basic.library”的一部分。如果驱动设备支持，该模块可用来切换到其他控制器模式。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		控制器模式必须为其改变的驱动设备，见 AXIS_REF。
VAR_INPUT			
bExecute	BOOL	FALSE	当为 TRUE 时，功能块是活动的。
nControllerMode	SMC_CONTROLLER_MODE	SMC_position	想要的控制器模式见 SMC_ControllerMode
VAR_OUTPUT			
bDone	BOOL	FALSE	如果参数可用则为 TRUE。
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
bError	BOOL	FALSE	在功能块内部发生错误的信号。
nErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

SMC_GETMAXSETACCDEC

这个功能块是库“SM3_Basic.library”的一部分。它用来测量一个轴的加速度（或减加速度）绝对值的最大值。如果 bEnable 为 TRUE 测量将进行，当 bResetMax 保持 TRUE 时测量值重置为 0。用 dwTimeStamp 您可以读取任何 DWORD (例如，调用计数器)， 这些 DWORD 值接受并输出新的最大值。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。

VAR_INPUT			
bEnable	BOOL	FALSE	必须设为 TRUE，以使得功能块能够执行。
dwTimeStamp	DWORD		可选的时间戳输入；可用来找出最大值什么时候发生。
VAR_OUTPUT			
bValid	BOOL	FALSE	如果参数可用则为 TRUE
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
fMaxAcceleration	LREAL	0	以技术单位/秒²表示的最大加速度值
dwTimeAtMax	DWORD	0	当最大值发生时 dwTimeStamp 的值

SMC_GETMAXSETVELOCITY

这个功能块是库“SM3_Basic.library”的一部分。它用来测量一个轴速度最大值。如果 bEnable 为 TRUE 测量将进行，当 bResetMax 保持 TRUE 时测量值重置为 0。用 dwTimeStamp 您可以读取任何 DWORD (例如，调用计数器)， 这些 DWORD 值接受并输出新的最大值。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
bEnable	BOOL	FALSE	必须设为 TRUE，以使得功能块能够执行。
dwTimeStamp	DWORD		可选的时间戳输入；可用来找出最大值什么时候发生。
VAR_OUTPUT			
bValid	BOOL	FALSE	如果参数可用则为 TRUE
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
fMaxVelocity	LREAL	0	以技术单位/秒表示的最大速度值
dwTimeAtMax	DWORD	0	当最大值发生时 dwTimeStamp 的值

SMC_GETTRACKINGERROR

这个功能块是库“SM3_Basic.library”的一部分。它测量用来补偿死区时间的实际和最大迟滞误差，这种误差可能在现场总线通讯时发生并将持续很多个周期 (byDeadTimeCycles) 。与 SMC_GetMaxSetVelocity 相似，时间戳(dwTimeStamp)可被用来测量在最大值时的时间。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
bEnable	BOOL	FALSE	必须设为 TRUE，以使得功能块能够执行。
byDeadTimeCycles	BYTE	2	死区时间周期个数。
dwTimeStamp	DWORD		可选的时间戳输入；可用来找出最大值什么时候发生。
VAR_OUTPUT			
bValid	BOOL	FALSE	如果参数可用则为 TRUE
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
fActTrackingError	LREAL	0	当前延迟错误
fMaxTrackingError	LREAL	0	最大延迟错误

dwTimeAtMax	DWORD	0	当最大值发生时 dwTimeStamp 的值

SMC_INPOSITION

这个功能块是库“SM3_Basic.library”的一部分。它检查轴的实际位置是否在一段确定的时间内在某个位置值以内。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
bEnable	BOOL	FALSE	必须设为 TRUE，以使得功能块能够执行。
fPosWindow	LREAL		如果 fPosWindow >=Distance(实际位置，设置位置)，驱动设备在位置窗口内。
fPosTime	LREAL		在 bInPosition 被设为 TRUE 之前，该功能块的时间跨度位置窗口内。
fTimeOut	LREAL	SMC_BRAKE_AUTO	如果功能块激活后经过的时间超过了 fTimeOut，并且 bInPosition 还没有被设置，那么 bTimeOut 将被设置为 TRUE。这种机制可通过将 dTimeOut 设置为0来使其失效。
VAR_OUTPUT			
bInPosition	BOOL	FALSE	如果参数可用则为 TRUE。
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
bTimeOut	BOOL	FALSE	功能块内部发生错误的信号。

SMC_MEASUREDISTANCE

这个功能块是库“SM3_Basic.library”的一部分。它用于一个旋转轴，以测量走过的距离（外壳也被计算在内）。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
bExecute	BOOL	FALSE	上升沿启动测量
VAR_OUTPUT			
fDistance	LREAL	0	测量开时后走过的距离
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
bError	BOOL	FALSE	在功能块内部发生错误的信号。
nErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

SMC_CHECKLIMITS

这个功能块是库“SM3_Basic.library”的一部分。它可用来检查驱动装置的实际设置点是否超过了在控制器中配置的最大值。检查的结果将被 bLimitsExceeded 输出指示。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。

VAR_INPUT			
bEnable	BOOL	FALSE	必须设为 TRUE，以使得功能块能够执行。
bCheckVel	BOOL	TRUE	如果为 TRUE，设置的速度将被检查。
bCheckAccDec	BOOL	FALSE	如果为 TRUE，设置的加速度和减加速度将被检查。
VAR_OUTPUT			
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
bError	BOOL		在功能块内部发生错误的信号。
iErrorId	SMC_ERROR		错误指示，见 SMC_Error。
bLimitsExceeded	BOOL		如果实际设置点超过了在 PLC 配置中定义的、并且存储在 AXIS_REF（fSWMaxVelocity, fSWMaxAcceleration 或 fSWMaxDeceleration）中的限制，则该值为 TRUE。

SMC_FOLLOWPOSITION

这个功能块是库“SM3_Basic.library”的一部分。它将不作任何检查地把位置设置点写入轴。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
bExecute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
fSetPosition	LREAL		以技术单位表示的设置位置
VAR_OUTPUT			
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
bCommandAborted	BOOL		如果该命令已被其他命令终止则为 TRUE。
bError	BOOL		在功能块内部发生错误的信号。
iErrorID	SMC_ERROR		错误指示，见 SMC_Error。

SMC_FOLLOWPOSITIONVELOCITY

这个功能块是库“SM3_Basic.library”的一部分。除了可能额外定义的速度以外， 它的使用和 SMC_FollowPosition 等效。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
bExecute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
fSetPosition	LREAL		以技术单位表示的设置位置
fSetVelocity	LREAL		以技术单位/秒表示的设置速度
VAR_OUTPUT			
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
bCommandAborted	BOOL		如果该命令已被其他命令终止则为 TRUE。
bError	BOOL		在功能块内部发生错误的信号。
iErrorID	SMC_ERROR		错误指示，见 SMC_Error。

SMC_FOLLOWSETVALUES

这个功能块是库“SM3_Basic.library”的一部分。它将不作任何检查地把设置值写入轴。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
bExecute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
dwValueMask	DWORD		这些值的比特位决定了，下列哪个输入值将被写入轴中（TRUE）或者哪些量将被忽略（FALSE）：Bit0： fSetPosition Bit1：fSetVelocity Bit2：fSetAcceleration Bit3： fSetTorque Bit4：fSetCurrent
fSetPosition	LREAL		以技术单位表示的设置位置
fSetVelocity	LREAL		以技术单位/秒表示的设置速度
fSetAcceleartion	LREAL		以技术单位/秒^{2}表示的设置加速度
fSetTorque	LREAL		以 Nm 或 N（线性）表示的设置力矩
fSetCurrent	LREAL		设置电流强度(A)
VAR_OUTPUT			
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
bCommandAborted	BOOL		如果该命令已被其他命令终止则为 TRUE。
bError	BOOL		在功能块内部发生错误的信号。
iErrorID	SMC_ERROR		错误指示，见 SMC_Error。

SMC_FOLLOWVELOCITY

这个功能块是库“SM3_Basic.library”的一部分。它将不作任何检查地将设置速度写入轴。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
bExecute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
fSetVelocity	LREAL		以技术单位/秒表示的设置速度。
VAR_OUTPUT			
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
bCommandAborted	BOOL		如果该命令已被其他命令终止则为 TRUE。
bError	BOOL		在功能块内部发生错误的信号。
iErrorID	SMC_ERROR		错误指示，见 SMC_Error。

SMC_CLEARFBERROR

这个功能块是库“SM3_Basic.library”的一部分。它用来删除最老的功能块错误信息。

	类型	初始值	描述
VAR_INPUT			
pDrive	POINTER TO AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_OUTPUT			
SMC_ClearFBError	BOOL		FALSE（伪输出）

SMC_READFBERROR

这个功能块是库“SM3_Basic.library”的一部分。它用来读取功能块错误的最老信息。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
bEnable	BOOL	FALSE	必须被设为 TRUE 以使功能块能够执行。
VAR_OUTPUT			
bValid	BOOL	FALSE	如果参数有效则为 TRUE。
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
bFBError	BOOL	FALSE	在功能块内部发生错误的信号。
nFBErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。
pbyErrorInstance	POINTER TO BYTE		该输出指向报告此错误的功能块。
tTimeStamp	TIME		错误的时间戳。

SMC_HOMING

这个功能块是库“SM3_Basic.library”的一部分。它执行轴的参考运动。做为参考开关，一个布尔值被使用，典型地该值是由一个硬件输入的。

在模块被 bExecute 上升沿启动之后，它以 fVelocityFast 速度沿着 nDirection 定义的方向移动轴，直到 bReferenceSwitch = FALSE，这时参考开关将被关闭。然后轴将减速并按照 fVelocitySlow 被反向驱动。 参考位置将被设置， 驱动装置将被精确停止在参考开关打开的位置（bReferenceSwitch = TRUE）。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
bExecute	BOOL	FALSE	
fHomePosition	LREAL	0	在自动导引后的静止位置 (技术单位[u])
fVelocitySlow	LREAL	0	使用高速度直到参考开关被找到[u/s]
fVelocityFast	LREAL	0	使用低速度以去除参考开关[u/s]
fAcceleration	LREAL	0	加速度值[u/s2]
fDeceleration	LREAL	0	减加速度值[u/s2]
nDirection	MC_DIRECTION	negative	请查看 MC_Direction； 可能值：正，负，当前值；不能应用的！：最短，最快

bReferenceSwitch	BOOL	TRUE	参考开关
fSignalDelay	LREAL	0	为死区补偿所传递的 bReferenceSwitch 的时间[s]
nHomingMode	SMC_HOMING_MODE	FAST_BSLOW_S_STOP	自动导引模式，见 smc_homing_mode
bReturnToZero	BOOL	FALSE	如果为 TRUE，轴将在自动导引后被移动到零位置。
bIndexOccured	BOOL	FALSE	标志脉冲，对 bStartLatchingIndex 的反应
fIndexPosition	LREAL	0	出现标志的位置
bIgnoreHWLimit	BOOL	FALSE	如果被激活，当自动导引时设置 bHWLimitEnable = FALSE
VAR_OUTPUT			
bDone	BOOL	FALSE	如果自动导引已完成则为 TRUE
bBusy	BOOL		当功能块执行还没结束时为 TRUE。
bCommandAborted	BOOL	FALSE	如果该命令已被其他命令终止则为 TRUE。
bError	BOOL	FALSE	在功能块内部发生错误的信号。
nErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。
bStartLatchingIndex	BOOL		对于几个模式，标志脉冲应被评估。一旦标志脉冲封闭被启动（这必须由应用程序来处理），功能块就设置这个输出。当出现标志时，结果被设置在 bIndexOccured 和 fIndexPosition 中。

SMC_CHANGEGEARINGRATIO

这个功能块是库“SM3_Basic.library”的一部分。在该模块的帮助下，传动比和驱动设备类型可以被修改。

当该模块执行完毕后轴必须被 SMC3_ReinitDrive 通过（bKeepRatioSettings=TRUE）重启，以保证所有变量被正确初始化！

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
bExecute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
dwRatioTechUnitsDenom	DWORD		SoftMotionUnit–转换比增量（分母）
iRatioTechUnitsNum	INT		SoftMotionUnit–转换比增量（分子）
fPositionPeriod	LREAL		位置周期，模值（仅对回转驱动设备）
iMovementType	INT		0：回转轴，1：线性轴。
VAR_OUTPUT			
bDone	BOOL	FALSE	如果自动导引完成则为 TRUE。
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
bError	BOOL	FALSE	在功能块内部发生错误的信号。
nErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

5.3.2 错误

SMC_ERRORSTRING

这个功能块是库“SM3_Basic.library”的一部分。依赖于输入 ErrorID 和语言，函数 SMC_ErrorString 将返回一个字符串表达的错误。

	类型	初始值	描述
VAR_INPUT			
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。
Language	SMC_LANGUAGE_TYPE	english	希望的语言，见 SMC_LANGUAGE_TYPE
VAR_OUTPUT			
SMC_ErrorString	STRING(100)	FALSE	描述错误的字符串

5.3.3 文件

SMC_READCAM

这个功能块是库“SM3_Basic.library”的一部分。它用来在运行时下载 CAM 并使其对 MC_CamTableSelect 和 MC_CamIn 有效。要下载的 CAM 必须之前在 CAM 编辑器中被创建并被存进*.CAM 文件中。可下载的 CAM 的大小由全局常量 gc_SMC_FILE_MAXCAMEL (元素个数)和 gc_SMC_FILE_MAXCAMTAP (CAM 开关动作数量)来限制。

	类型	初始值	描述
VAR_INPUT			
bExecute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
sFileName	STRING(80)	''	
VAR_OUTPUT			
bDone	BOOL	FALSE	如果 CAM 被读入则为 TRUE
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
bError	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR		错误指示，见 SMC_Error。
CAM	MC_CAM_REF		关于 CAM，见 MC_CAM_REF

见：SMC_WriteCAM

SMC_WRITECAM

这个功能块是库“SM3_Basic.library”的一部分。该功能块在运行时向*.CAM-file 文件写入一个在 CAM 编辑器中创建的 CAM。

	类型	初始值	描述
VAR_IN_OUT			
CAM	MC_CAM_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
bExecute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
sFileName	STRING(80)	''	
VAR_OUTPUT			
bDone	BOOL	FALSE	当 CAM 被写入后为 TRUE
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。

bError	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

见：SMC_ReadCAM

SMC_AXISDIAGNOSTICLOG

这个功能块是库“SM3_Basic.library”的一部分。它用来将属于一个轴的一系列参数值循环写入到文件中。这个输出文件是故障诊断的依据。

在向数据媒介中写入数据时通常需要一些时间，该模块把收集到的数据存储到一个 5k 字节大小的缓冲区中，直到模块动作 WriteToFile 被调用时数据才被写入。这个动作调用应该被放在低优先级的慢任务（大约 50ms）中，以防止实际的运动任务和运动本身被干扰。一旦超出缓冲区，模块将产生一个错误输出。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
bExecute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
bCloseFile	BOOL		一旦该输入为 TRUE，模块就关闭文件。
sFileName	STRING(80)	' '	文件名(前面加路径).
bSetPosition	BOOL		如果为 TRUE，与该轴向关联的值将被写入文件。
bActPosition	BOOL		如果为 TRUE，与该轴向关联的值将被写入文件。
bSetVelocity	BOOL		如果为 TRUE，与该轴向关联的值将被写入文件。
bActVelocity	BOOL		如果为 TRUE，与该轴向关联的值将被写入文件。
bSetAcceleration	BOOL	0	如果为 TRUE，与该轴向关联的值将被写入文件。
bActAcceleration	BOOL	0	如果为 TRUE，与该轴向关联的值将被写入文件。
bySeparatorChar	BYTE	9	字母的 ASCII 编码，它被加到两个相同日期之间。
sRecordSeparatorString	STRING(3)	'RN'	在数据末尾写入的字符串。
VAR_OUTPUT			
Done	BOOL	FALSE	如果指定距离已经被叠加到当前运动中为 TRUE
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。
bRecording	BOOL	FALSE	如果模块正在记录时为 TRUE。

5.3.4 PLCopen

SMC_BacklashCompensation

这个功能块是库“SM3_Basic.library”的一部分。它用来补偿出现的间隙（例如在带传动时）。为此主轴（通常是虚拟的）被镜像到真实的、应被补偿间隙的从轴上。

该模块象相位调整模块一样工作，相位依赖于正在移动的主轴的方向。如果主轴开始向一个确定的方向移动，从轴也将同样动作，此外从轴在这个方向上还有一个附加移动（距离：fBacklash）；如果主轴反向，从轴也反向并在新方向上多走两倍于 fBacklash 的距离。

注意，当模块开始工作时主、从轴应该在同一位置，因为如果不是的话，从轴也会被定位到主轴上。

	类型	初始值	描述
VAR_IN_OUT			
Master	AXIS_REF		关于主轴，见 AXIS_REF_SM3。
Slave	AXIS_REF		关于从轴，见 AXIS_REF_SM3。
VAR_INPUT			
bExecute	BOOL		输入值的上升沿将启动该功能块的执行。它将保持活动状态直到从轴由于要用于其他模块而被分离，或者一个错误发生。
fBacklash	LREAL		要补偿的距离/间隙
fCompensationVel	LREAL		补偿要用到的附加速度。
fCompensationAcc	LREAL		补偿要用到的附加加速度。
fCompensationDec	LREAL		补偿要用到的附加减加速度。
eBacklashMode	SMC_BACKLASH_MODE		SMC_BACKLASH_MODE 是一个 VAR_GLOBAL 结构，它描述了补偿模式： SMC_BL_AUTO：在主轴移动方向上；确定补偿方向。 SMC_BL_POSITIVE：在正方向上补偿，依赖于主轴的运动。 SMC_BL_OFF：不补偿。
eBacklashStartState	SMC_BACKLASH_STARTSTATE		SMC_BACKLASH_STARTSTATE 是一个 VAR_GLOBAL 结构，它描述了当该模块开始起作用时轴的状态： SMC_BL_START_NEGATIVE：从轴被向负方向上拉，就是说：在负方向上的运动不需要补偿；在正方向上的运动将被两倍于 fBacklash 距离补偿。 SMC_BL_START_POSITIVE：从轴被向负方向上拉，就是说：在正方向上的运动不需要补偿；在负方向上的运动将被两倍于 fBacklash 距离补偿。 SMC_BL_START_NONE：从轴上没有牵引力；在正方向或负方向上的运动将导致相应的距离为 fBacklash 的运动补偿。
VAR_OUTPUT			
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
bCommandAborted	BOOL	FALSE	如果该命令已被其他命令终止则为 TRUE。
bError	BOOL	FALSE	在功能块内部发生错误的信号。
iErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

SMC_READSETPOSITION

这个功能块是库“SM3_Basic.library”的一部分。它用来读取驱动设备的当前设置位置。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
Enable	BOOL	FALSE	必须被设为 TRUE 以使功能块能够执行。
VAR_OUTPUT			
Valid	BOOL	FALSE	如果参数有效则为 TRUE。
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。
Position	LREAL	0	传动装置的位置。

SMC_SETTORQUE

这个功能块是库“SM3_Basic.library”的一部分。如果驱动设备处于控制器模式“torque”时，它可用来创建一个力矩。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
bEnable	BOOL	FALSE	必须被设为 TRUE 以使功能块能够执行。
fTorque	LREAL		力矩[Nm](回转电机)或[N](直线电机)
VAR_OUTPUT			
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
bError	BOOL	FALSE	在功能块内部发生错误的信号。
nErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

SMC_CAMBOUNDS

这个功能块是库“SM3_Basic.library”的一部分。它计算从设备的最大位置、速度和加速度值，该设备以绝对模式与一个主设备相连，主设备按规定的最大速度和加速度/减加速度移动。

当 CAM 磁盘被创建并被在线修改、最大速度是否被遵守需要优先检查时，该模块十分有用。

	类型	初始值	描述
VAR_IN_OUT			
CAM	MC_CAM_REF		关于 CAM，见 AXIS_REF_SM3
VAR_INPUT			
bExecute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
dMasterVelMax	LREAL	1	主设备速度的绝对最大值。
dMasterAccMax	LREAL	0	主设备加速度/减加速度的绝对最大值。
dMasterScaling	LREAL	1	应用于 CAM 的主设备的缩放因数
dSlaveScaling	LREAL	1	应用于 CAM 的从设备的缩放因数
VAR_OUTPUT			
bDone	BOOL		如果计算已被终止则为 TRUE。
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。

bError	BOOL	FALSE	在功能块内部发生错误的信号。
nErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。
dMaxPos	LREAL		从设备位置最大值[u]。
dMinPos	LREAL		从设备位置最小值[u]。
dMaxVel	LREAL		从设备速度最大值[u]。
dMinVel	LREAL		从设备速度最小值[u]。
dMaxAccDec	LREAL		从设备最大加速度/减加速度值 [u]。
dMinAccDec	LREAL		从设备最小加速度/减加速度值 [u]。

见：SMC_CAMBounds_Pos

SMC_CAMBOUNDS_POS

这个功能块是库“SM3_Basic.library”的一部分。与 SMC_CAMBounds 相对的，该功能块仅计算从设备的位置最大、最小值，该设备以绝对模式与一个主设备相连，主设备按规定的最大速度和加速度/减加速度移动。

当 CAM 磁盘被创建并被在线修改、最大速度是否被遵守需要优先检查时，该模块十分有用。

	类型	初始值	描述
VAR_IN_OUT			
CAM	MC_CAM_REF		关于 CAM，见 AXIS_REF_SM3
VAR_INPUT			
bExecute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
dMasterVelMax	LREAL	1	主设备速度的绝对最大值。
dMasterAccMax	LREAL	0	主设备加速度/减加速度的绝对最大值。
dMasterScaling	LREAL	1	应用于 CAM 的主设备的缩放因数
dSlaveScaling	LREAL	1	应用于 CAM 的从设备的缩放因数
VAR_OUT			
bDone	BOOL		如果计算已被终止则为 TRUE。
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
bError	BOOL	FALSE	在功能块内部发生错误的信号。
nErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。
dMaxPos	LREAL		从设备位置最大值[u]。
dMinPos	LREAL		从设备位置最小值[u]。
dMaxVel	LREAL		从设备速度最大值[u]。
dMinVel	LREAL		从设备速度最小值[u]。
dMaxAccDec	LREAL		从设备最大加速度/减加速度值 [u]。
dMinAccDec	LREAL		从设备最小加速度/减加速度值 [u]。

见：SMC_CAMBounds

SMC_CAMREGISTER

这个功能块是库“SM3_Basic.library”的一部分。它代表一个挺杆控制单元，它和 MC_CamIn 一样，作用于 MC_CAM_REF 结构，撤销原轨迹信息而只读挺杆信息。

	类型	初始值	描述
VAR_IN_OUT			
Master	AXIS_REF		关于主轴，见 AXIS_REF_SM3。

CamTable	MC_CAM_REF		描述一个包含 tappet 描述的（也许是空的）CAM，见 MC_CAM_REF。
bTappet	ARRAY [1..MAX_NUM_TAPPETS] OF BOOL		Tappet 位。
VAR_INPUT			
Enable	BOOL	FALSE	必须设为 TRUE，以使得功能块能够执行。
MasterOffset	LREAL	0	要与主设备位置相加得偏移。
MasterScaling	LREAL	1	主设备分布的缩放因数。
TappetHysteresis	LREAL	0	tappet 迟滞的大小。
DeadTimeCompensation	LREAL	0	以秒为单位的死区时间补偿。预期的主轴位置将通过线性外插法计算。
VAR_OUTPUT			
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。
EndOfProfile	BOOL		指示 CAM 分布周期结束的脉冲输出。

SMC_CamEditor

用于在可视化中显示 cam 表 (见 “Online Cam Editor”).

	类型	初始值	描述
VAR_IN_OUT			
cam	MC_CAM_REF		Cam 表的名称
VAR_INPUT			
bEnable	BOOL	FALSE	若设为 FALSE，则 cam 表可能没有改动。
VAR_OUTPUT			
bCAMchanged	BOOL	FALSE	该标志表示在可视化中，cam 已经改变。
bError	BOOL	FALSE	功能块中出现错误的符号
nErrorID	SMC_ERROR	0	错误代码 ⇘“SMC_ERROR”

SMC_GETCAMSLAVESETPOSITION

这个功能块是库“SM3_Basic.library”的一部分。如果（从）轴通过 CAM 与其他（主）轴的运动相连接，该模块计算从轴的当前目标位置。因此两个轴都不会被移动或影响。

该模块可被用来得到从设备的设置位置，该设备通过 CAM 工作台与主轴相连。

由于相应的值在一个周期内被算出，所以输出 Done 不需要。

	类型	初始值	描述
VAR_IN_OUT			
Master	AXIS_REF		关于主轴，见 AXIS_REF_SM3。
Slave	AXIS_REF		关于从轴，见 AXIS_REF_SM3。
VAR_INPUT			
Enable	BOOL	FALSE	必须设为 TRUE 以使得功能块能够执行。
MasterOffset	LREAL	0	主 table 的偏移

SlaveOffset	LREAL	0	从 table 的偏移
MasterScaling	LREAL	1	主分布的缩放因数
SlaveScaling	LREAL	1	从分布的缩放因数
CamTableID	MC_CAM_ID	TRUE	CAM table 的标识，MC_CamTableSelect 的输出，见 MC_CamTableSelect
VAR_OUTPUT			
fStartPosition	LREAL	FALSE	依照 cam 和主设备实际位置的从设备位置。
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

见：SMC_GetCamSlaveSetPositionEx

SMC_GETTAPPETVALUE

这个功能块是库“SM3_Basic.library”的一部分。它评估 MC_CamIn 功能块的输出 Tappet 并包含当前 tappet 状态。

	类型	初始值	描述
VAR_IN_OUT			
Tappets	SMC_TappetData		将被MC_GetTappetValue 评估的Tappet 信号，见 SMC_TappetData
VAR_INPUT			
iID	INT		将被评估的 Tappet 组 ID
bInitValue	BOOL		在第一次调用时将被分配的 tappet 的初始值
bSetInitValueAtReset	BOOL		如果为 TRUE，在 CamIn 功能块重启时 tappet 的值将被设置为 bInitValue。 如果为 FALSE，在 CamIn 功能块重启时 tappet 的值将被保持
VAR_OUTPUT			
bTappet	BOOL	FALSE	Tappet 的值

MC_CAMIN

这个功能块是库“SM3_Basic.library”的一部分。它占用 CAM。

	类型	初始值	描述
VAR_IN_OUT			
Master	AXIS_REF		关于主轴，见 AXIS_REF_SM3。
Slave	AXIS_REF		关于从轴，见 AXIS_REF_SM3。
VAR_INPUT			
Execute	BOOL	FALSE	输入值的上升沿将启动该函数块的执行。
MasterOffset	LREAL	0	主轴工作台偏移。
SlaveOffset	LREAL	0	从轴工作台偏移。
MasterScaling	LREAL	1	主设备分布缩放因数。
SlaveScaling	LREAL	1	从设备分布缩放因数。
StartMode	MC_StartMode	absolute	绝对、相对或 ramp_in（于当前位置）。
CamTableID	MC_CAM_ID		所用的 CAM 工作台的标识符，MC_CamID 的输出。

VelocityDiff	LREAL	0	对于 ramp_in 速度差的最大值。
Acceleration	LREAL	0	ramp_in 的加速度。
Deceleration	LREAL	0	ramp_in 的减加速度。
TappetHysteresis	LREAL	0	挺杆迟滞大小。
VAR_OUTPUT			
InSync	BOOL	FALSE	如果从设备根据CAM 工作台同步于主设备则值为 TRUE。
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
CommandAborted	BOOL	FALSE	如果该命令已被其他命令终止则为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。
EndOfProfile	BOOL	FALSE	脉冲输出指示了 CAM 分布的周期结束。
Tappets	SMC_TappetData		要被 SMC_GetTappetValue 评估的挺杆信号。

注意：

主设备不一定是固定的。

如果在执行 MC_CamIn 时实际的主设备和从设备的位置并不符合于偏移值，则要么发生一个错误，要么系统自动处理出现的偏差。

StartMode：CAM 要么“绝对”地要么“相对”地被定位于主－从设备的位置。“ramp-in”是供货商定义的值。它可以与附加参数结合起来，比如主设备距离参数、加速度参数或者其他供货商确定的参数，而从设备 ramp-in 到凸轮分布中（“飞行结合”）。

从设备偏置的机械模拟是一个以附加固定层厚度焊接的凸轮。因此，从设备位置总有固定偏移，这个偏移假设为线性引导从设备挺杆，可被解释为主设备轴的轴线偏移。

MC_CAMOUT

这个功能块是库“SM3_Basic.library”的一部分。它将立即从主轴中释放从轴。

	类型	初始值	描述
VAR_IN_OUT			
Slave	AXIS_REF		关于从轴，见 AXIS_REF_SM3
VAR_INPUT			
Execute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
VAR_OUTPUT			
Done	BOOL	FALSE	如果凸轮被放开为 TRUE。
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

注意：假设该命令被后面还有别的对于 MC_Stop, MC_GearIn 实例的命令。如果没有其他命令跟随，则默认条件应该是：保持前一个速度。

MC_CAMTABLESELECT

这个功能块是库“SM3_Basic.library”的一部分。它通过设置与相关工作台的连接来选择 CAM 工作台。

	类型	初始值	描述
VAR_IN_OUT			
Master	AXIS_REF		关于主轴，见 AXIS_REF_SM3。
Slave	AXIS_REF		关于从轴，见 AXIS_REF_SM3。
CamTable	MC_CAM_REF		关于 CAM 描述，见 MC_CAM_REF
VAR_INPUT			
Execute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
Periodic	BOOL	TRUE	TRUE =周期的, FALSE = 非周期的
MasterAbsolute	BOOL	TRUE	TRUE = 绝对， FALSE = 相对 坐标
SlaveAbsolute	BOOL	TRUE	TRUE = 绝对， FALSE = 相对 坐标
VAR_OUTPUT			
Done	BOOL	FALSE	如果预选已经完成，则为 TRUE
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。
CamTableID	MC_CAM_ID		要被功能块MC_CamID 使用的CAM 工作台标识号。

注意：

一个虚拟轴可被用作主轴。

当输出 Done 为 TRUE 时，CamTableID 有效且可以在 MC_CamIn 中使用。

MC_GEARIN

这个功能块是库“SM3_Basic.library”的一部分。它通过设置与相关工作台的连接来选择 CAM 工作台。

	类型	初始值	描述
VAR_IN_OUT			
Master	AXIS_REF		关于主轴，见 AXIS_REF_SM3。
Slave	AXIS_REF		关于从轴，见 AXIS_REF_SM3。
VAR_INPUT			
Execute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
RatioNumerator	INT	1	传动比分子
RatioDenominator	LREAL	1	传动比分母
Acceleration	LREAL	0	传动时的加速度
Deceleration	LREAL	0	传动时的减加速度
VAR_OUTPUT			
InGear	BOOL	FALSE	所命令的传动已完成时为 TRUE。
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
CommandAborted	BOOL	FALSE	如果该命令已被其他命令终止则为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

注意：

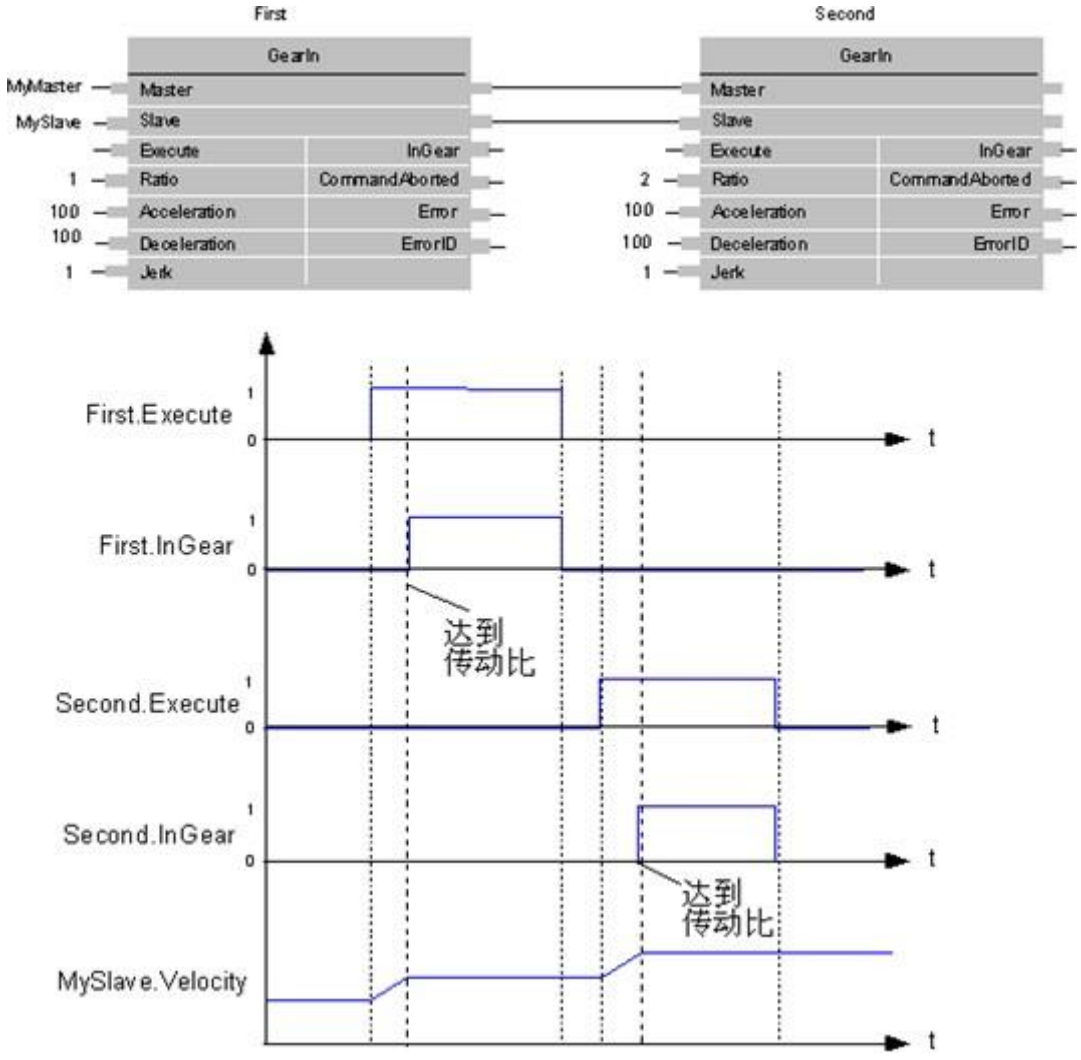
从设备增加到主速度的比率。在达到主设备速度后，从设备将被锁定。在同步过程中的所有距离损失将不被考虑。

在 MC_GearIn 运行时，传动比可被连续的 MC_GearIn 命令改变。这不需要先调用 MC_GearOut。

首次达到传动比时 InGear 将被设置。

在 InGear 设置以后，位置锁定或者只是速度锁定由系统指定。

传动时间图：



MC_GEARINPOS

这个功能块是库“SM3_Basic.library”的一部分。它通过设置与相关工作台的连接来选择 CAM 工作台。

	类型	初始值	描述
VAR_IN_OUT			
Master	AXIS_REF		关于主轴，见 AXIS_REF_SM3。
Slave	AXIS_REF		关于从轴，见 AXIS_REF_SM3。
VAR_INPUT			
Execute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
RatioNumerator	INT	1	传动比分子

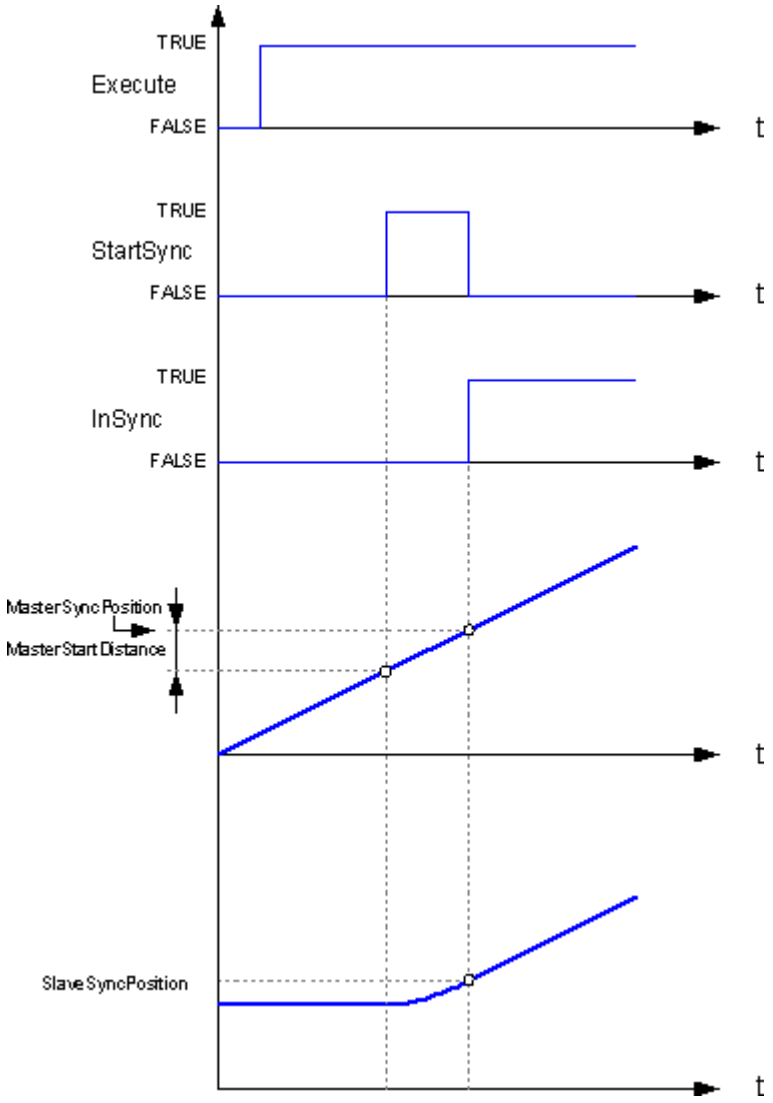
RatioDenominator	INT	1	传动比分母
MasterSyncPosition	REAL	0	同步运转时，主设备位置。
SlaveSyncPosition	REAL	0	同步运转时，从设备位置。
MasterStartDistance	REAL	0	传动过程的主设备距离（从设备轴将从这里开始进入同步）。
VAR_OUTPUT			
StartSync	BOOL		所命令的传动已开始时为 TRUE。
InSync	BOOL		所命令的传动已完成时为 TRUE。
Busy	BOOL		当功能块执行还没结束时为 TRUE。
CommandAborted	BOOL	FALSE	如果该命令已被其他命令终止则为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

注意：

如果 MasterStartDistance 已设置，所有之前的运动将继续，直到主设备在正确的方向上到达“MasterSyncPosition – MasterStartDistance”。在这点上输出 StartSync 被及时设置。当在同步发生之前，在“从”轴上执行了一个“Stop”命令，这将禁止同步化而且函数块将发布“CommandAborted”。

如果没有指定 MasterStartDistance，系统能自己基于其它相关输入为 StartSync 计算设定点。

MC_GearInPos 的时间图：



MC_GEAROUT

这个功能块是库“SM3_Basic.library”的一部分。它通过设置与相关工作台的连接来选择 CAM 工作台。它使从轴脱离主轴。

	类型	初始值	描述
VAR_IN_OUT			
Slave	AXIS_REF		关于从轴，见 AXIS_REF_SM3。
VAR_INPUT			
Execute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
VAR_OUTPUT			
Done	BOOL	FALSE	TRUE，如果 Cam 已被分离。
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

注意：现在假设的是该命令后还有其它命令，例如 instance MC_Stop, MC_GearIn。如果没有其他命令跟随，默认条件应该是：保持上一个速度。

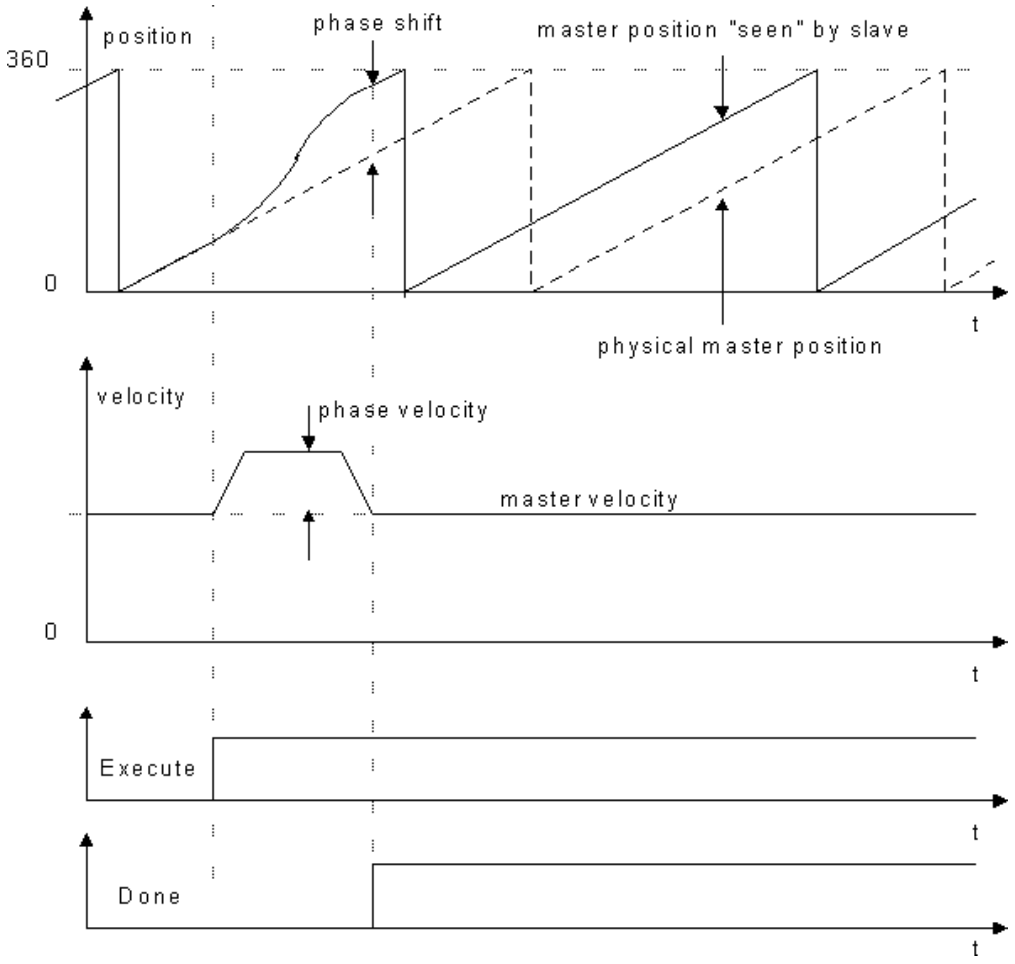
MC_PHASING

这个功能块是库“SM3_Basic.library”的一部分。它将占用 CAM。

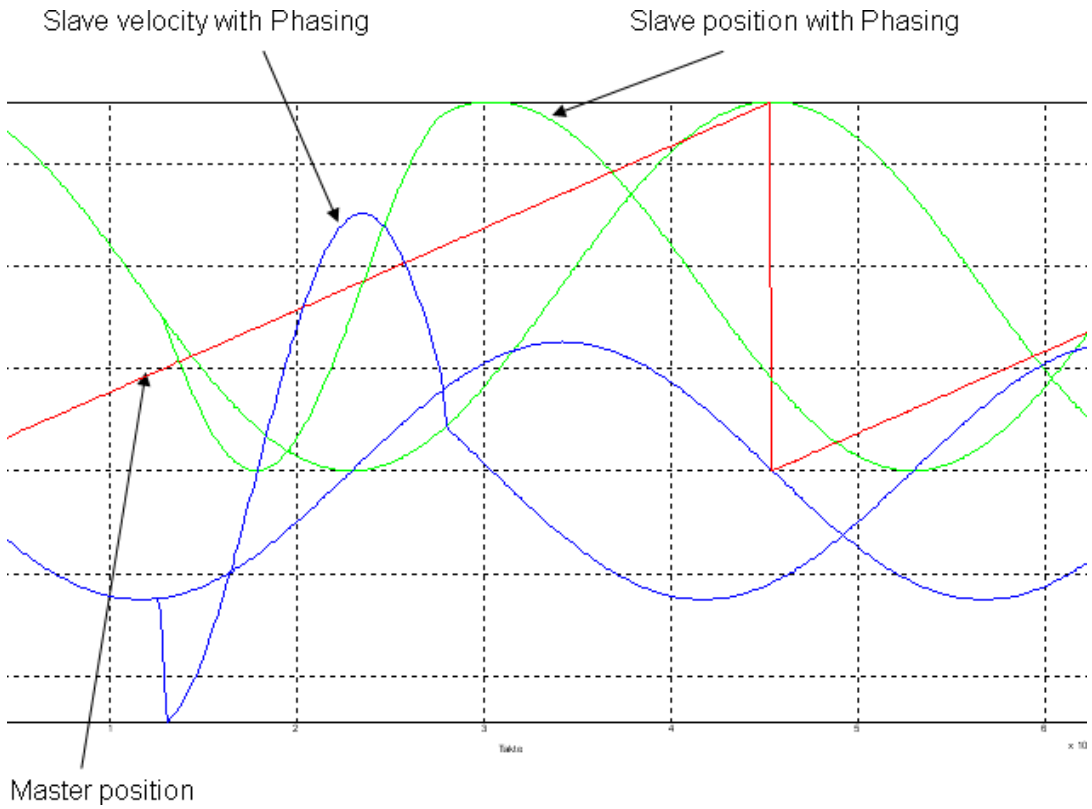
	类型	初始值	描述
VAR_IN_OUT			
Master	AXIS_REF		关于主轴，见 AXIS_REF_SM3。
Slave	AXIS_REF		关于从轴，见 AXIS_REF_SM3。
VAR_INPUT			
Execute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
PhaseShift	LREAL	0	主设备相差[u]
Velocity	LREAL	0	达到的相差的最大速度[u/s]
Acceleration	LREAL	0	达到的相差的最大加速度[u/s²]
Deceleration	LREAL	0	达到的相差的最大减加速度[u/s²]
VAR_OUTPUT			
Done	BOOL	FALSE	如果所命令的相位调整已经达到则为 TRUE。
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
CommandAborted	BOOL	FALSE	如果该命令已被其他命令终止则为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

注意：相移的相位、速度、加速度和减加速度是由该功能块控制的。

MC_Phasing 的定时例子：



例子：MC_Phasing 效果演示



MC_ACCELERATIONPROFILE

这个功能块是库“SM3_Basic.library”的一部分。它用来控制一个时间-加速度已锁定的运动分布。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
TimeAcceleration	MC_TA_REF		关于时间/加速度描述，见 MC_TA_REF。
VAR_INPUT			
Execute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
ArraySize	INT	0	数组维数。
AccelerationScale	LREAL	1	总体加速度缩放因数[技术单位]。
Offset	LREAL	0	总体分布[单位]偏移。
VAR_OUTPUT			
Done	BOOL	FALSE	如果分布已完成为 TRUE。
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
CommandAborted	BOOL	FALSE	如果该命令已被其他命令终止则为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

注意：MC_TA_REF 是特殊数据类型的提供者。一个该数据类型的例子如下：时间/加速度对的内容可以被表示为 Δ 时间/加速度，Δ 表示两个相邻点之间的时间差：

```
TYPE MC_TA
STRUCT
delta_time : TIME;
```

```
acceleration: REAL;
END_STRUCT
END_TYPE

TYPE MC_TV_TABLE
STRUCT
Number_of_pairs : INT;
IsAbsolute : BOOL;
MC_TA_Array : ARRAY [1..N] of MC_TA;
END_STRUCT
END_TYPE
```

在不同分布间的转换可被管理。

做为 MC_AccelerationProfile 替代选择，与虚拟主机连接的 CAM 函数块也可被使用。

例子：

某个加速度分布是建立在许多“A 到 B”的定位点之上。虽然这很容易想象，但是实际上需要很多程序段来达到一个平滑的分布。这种要求经常超出低端伺服电机的性能。

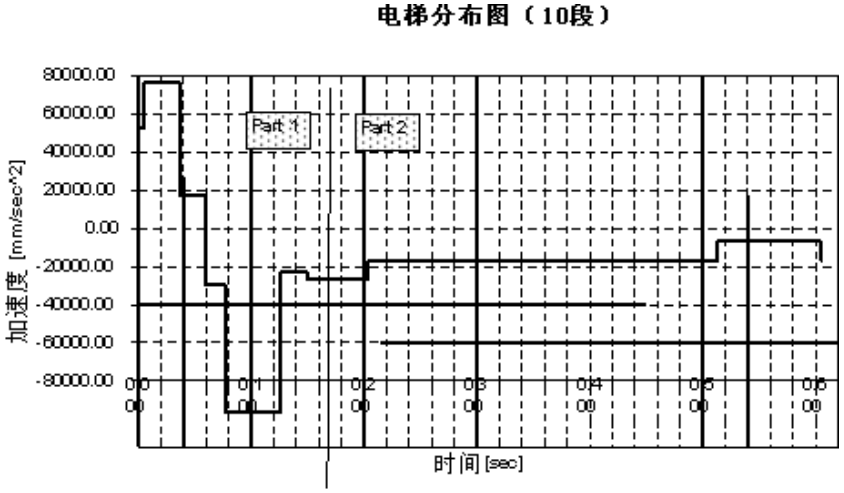
另一种办法，就是使用适当数量的恒定加速度段来定义一个近似符合的运动分布。用这种方法很多低端伺服电机的性能就得到了扩展。

因此您只能选择符合：

位置和时间分布；

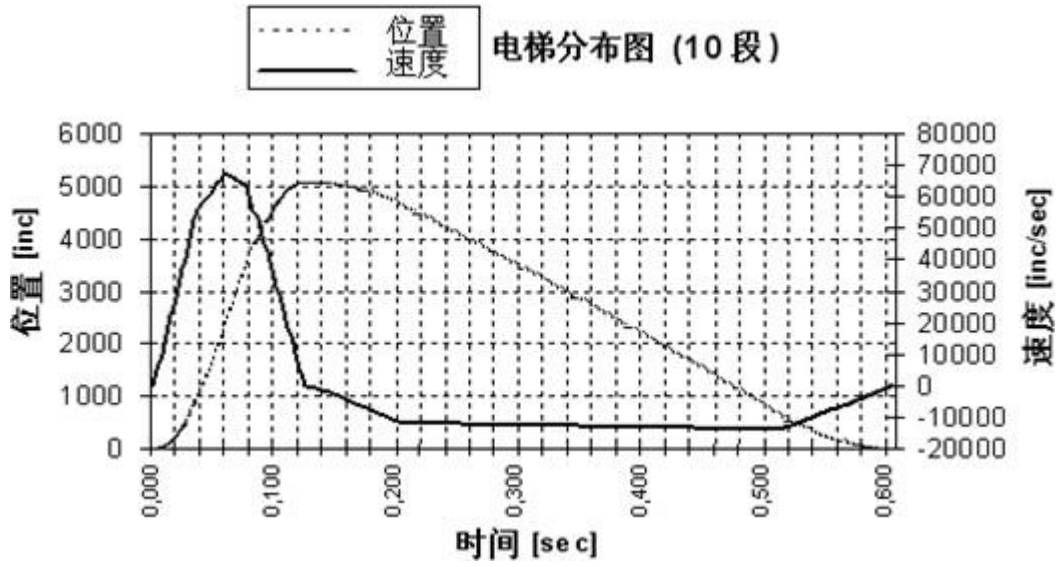
或者主轴和从轴。

这种方法的优点在于分布图的简洁描述性，以及它在较低处理器功率需求时的自然平滑性。缺点是较高的编程抽象性。



仅有10段的加速度分布：

得到的位置分布：



MC_HALT

这个功能块是库“SM3_Basic.library”的一部分。它用来命令一个受控运动停止。功能块将终止所有进行中的功能块的执行。轴将切换到“DiscreteMotion”状态，直到速度变为 0。当输出 DONE 设为 TRUE 时，状态将变为“StandStill”。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
Execute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
Deceleration	LREAL	0	加速度的模[单位/s^2]
VAR_OUTPUT			
Done	BOOL	FALSE	如果已达到0速度为 TRUE。
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
CommandAborted	BOOL	FALSE	如果该命令已被其他命令终止则为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

注意：

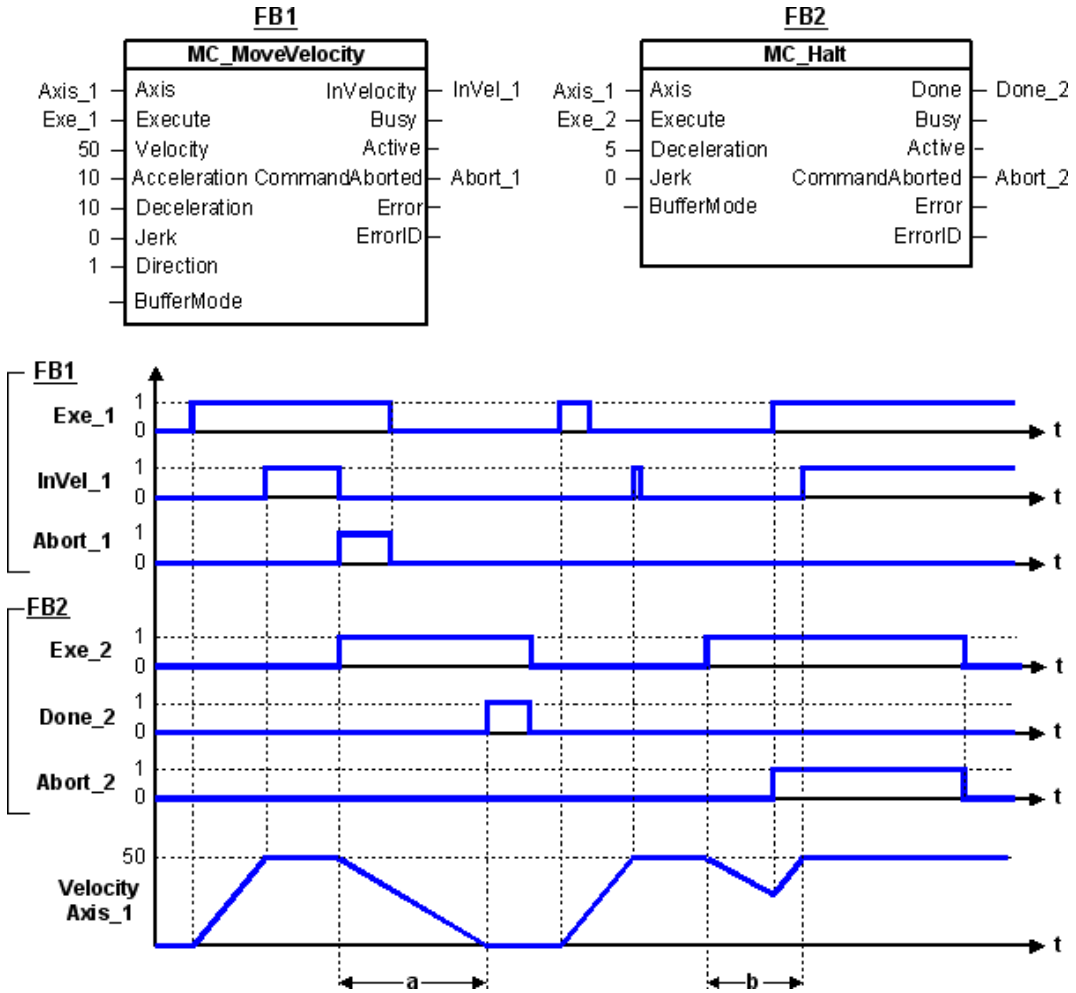
MC_Halt 用来在正常操作条件下停止轴。关于无缓冲模式：在轴减速时别的运动命令可被设置为立即终止 MC_Halt 。

当 MC_Halt 活动时，其他命令也可被发布。例如，无人驾驶车辆探测到障碍物必须停下来。所以 MC_Halt 将被发布。如果在达到停止状态之前障碍物被移除，则其它设定的运动命令将继续执行，实际上这使得车辆不会停下来。

例子：

该例展示了与 MC_MoveVelocity 连接的正在被执行的 MC_Halt 的行为。一个旋转的轴被 MC_Halt 减速。在它被执行时，另一个命令将覆盖 MC_Halt 命令。与 MC_Stop 相对，MC_Halt 允许被重写。所以轴可以在还没有停下来时再次加速。

MC_Halt 的例子：



MC_HOME

这个功能块是库“SM3_Basic.library”的一部分。它的执行引发轴执行“search home”序列。该序列的详情依赖于加工工艺并可以由轴的参数设置。“Position”输入用来在探测到参考信号时设置绝对位置。该功能块将以“StandStill”终止。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
Execute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
Position	LREAL	0	当参考信号被探测到时的绝对位置[u]。
VAR_OUTPUT			
Done	BOOL	FALSE	当达到 StandStill 状态时为 TRUE。
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
CommandAborted	BOOL	FALSE	如果该命令已被其他命令终止则为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

MC_MOVEABSOLUTE

这个功能块是库“SM3_Basic.library”的一部分。它用来命令受控运动到一个指定的绝对位置。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
Execute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
Position	LREAL	0	运动的目标位置（以技术单位[u]表示）(负或正)。
Velocity	LREAL	0	速度最大值（总是为正）(不是必须达到）[u/s]
Acceleration	LREAL	0	加速度值（总是为正）(电机能量增长） [u/s^2]
Deceleration	LREAL	0	减加速度值（总是为正）(电机能量减少） [u/s^2]
Direction	MC_DIRECTION	shortest	枚举类型，详见 MC_Direction；允许的值：对于线性/有限轴：正，负；对于旋转/求模轴：正，负当前值，最短，最快
VAR_OUTPUT			
Done	BOOL	FALSE	如果达到最终位置为 TRUE
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
CommandAborted	BOOL	FALSE	如果该命令已被其他命令终止则为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

注意：

如果没有其他动作挂起，该动作完成时速度为 0。

如果达到所命令的位置只有一种数学方法（就像在线性系统中那样），输入方向的值将被忽略。

对模数轴有效的绝对位置的范围是[0,360[（不包括 360），或者一个相当的范围。但是应用程序可能移动 MC_MoveAbsolute 要求位置到相应的模范围内。对相对位置来说，360 是适用的。

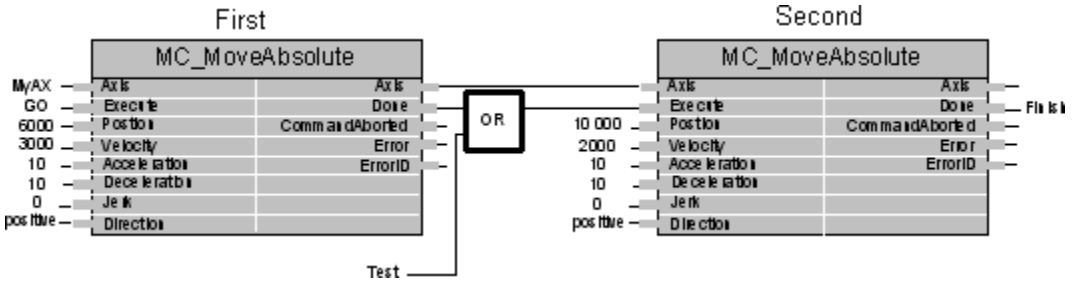
枚举类型“shortest”集中描述最短路径的轨迹。方向的选择基于当命令发布时实际具有的位置。

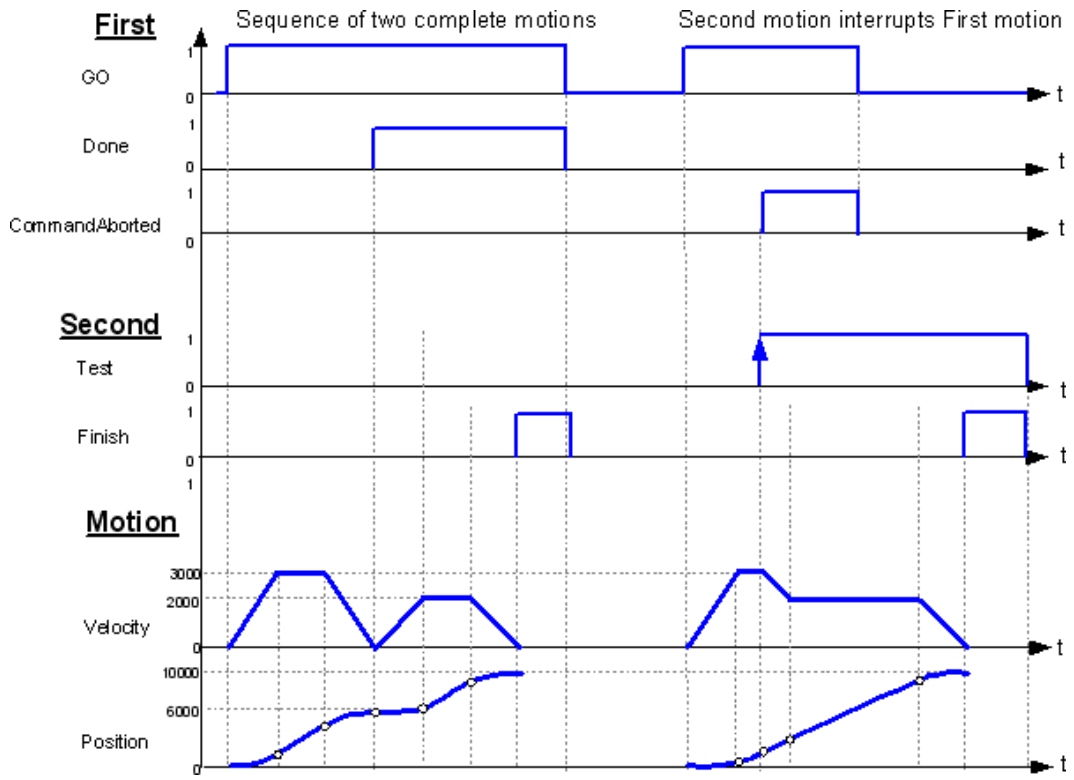
例子：

下图展示了 MC_MoveAbsolute 功能块的两个实例“First”和“Second”连接的例子：

时间图的左部画出了“Second”在“First”之后调用的情况。当“First”达到要求的位置 6000（速度为 0）时，输出 Done 将引发“Second”向位置 10000 移动。

时间图的右部画出了当“First”还在执行时“Second”被启动的情况。这时，“First”的运动被中断，并被在“First”速度恒定期间传递的 Test 信号终止。“Second”功能块将直接移向位置 10000，虽然并没有到达位置 6000。





MC_MOVEADDITIVE

这个功能块是库“SM3_Basic.library”的一部分。它用来在离散运动状态下命令一个受控运动，该运动把指定相对距离与新近要求的位置相加。新近要求的位置可能是已被终止的前一个 MC_MoveAdditive 运动的结果。如果功能块活动于连续模式下，则在执行时指定的相对距离将被加到实际距离上。

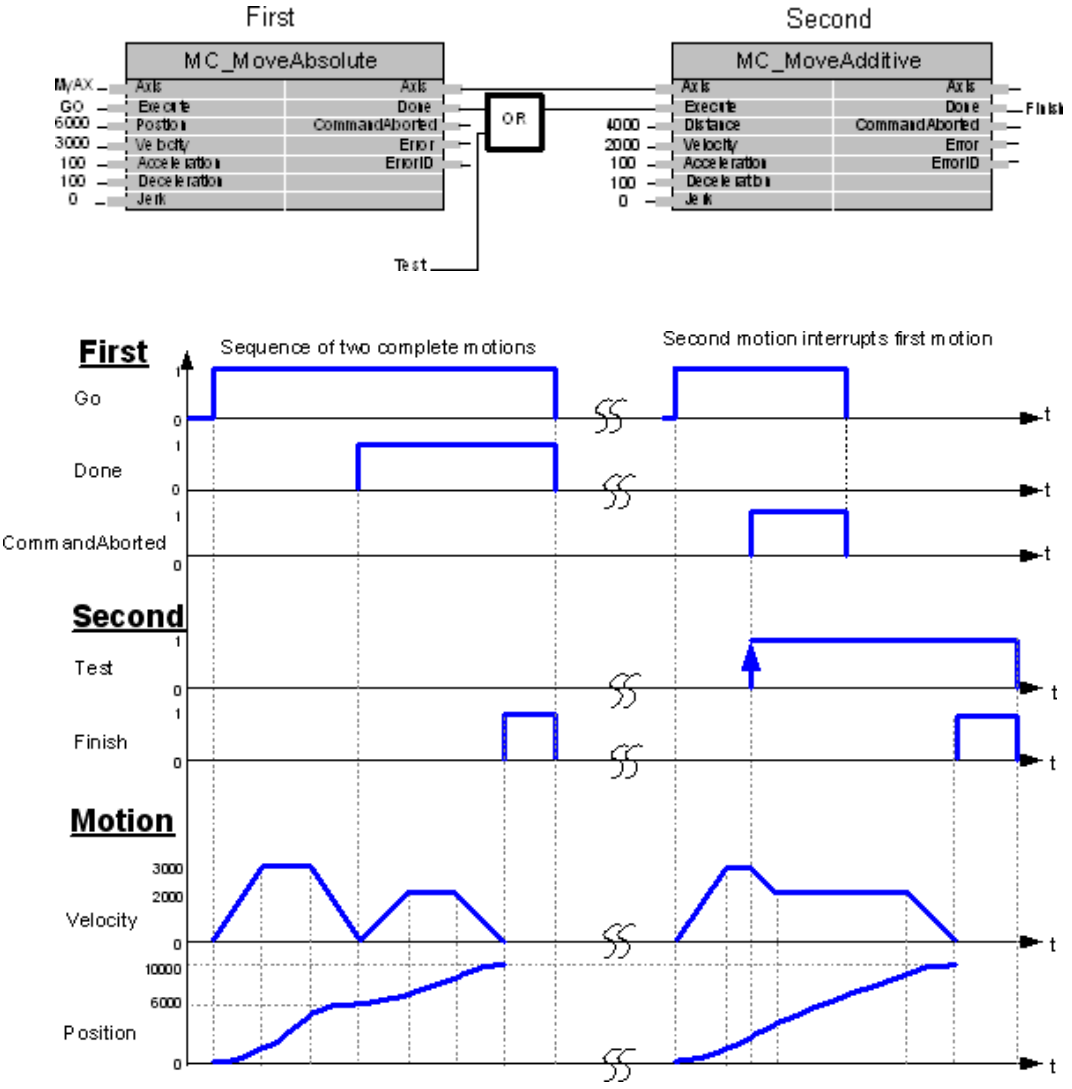
	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
Execute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
Distance	LREAL	0	运动的相对距离（以技术单位[u]表示）
Velocity	LREAL	0	最大速度（总为正）（不是必须达到）[u/s]。
Acceleration	LREAL	0	加速度值（总是为正）（电机能量增长） [u/s²]
Deceleration	LREAL	0	减加速度值（总是为正）（电机能量减少） [u/s²]
VAR_OUTPUT			
Done	BOOL	FALSE	如果达到最终位置为 TRUE
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
CommandAborted	BOOL	FALSE	如果该命令已被其他命令终止则为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

例子：

下图展示了当轴处于离散运动模式下，两个连接着的、MC_MoveAbsolute 功能块的实例“First”和“Second”的例子：

时间图的左部画出了“Second”在“First”之后调用的情况。当“First”达到要求的位置 6000（速度为 0）时，输出 Done 将引发“Second”向位置 10000 移动。

时间图的右部画出了当“First”还在执行时“Second”被启动的情况。这时，“First”的运动被中断，并被在“First”速度恒定期间传递的 Test 信号终止。在把距离 4000 加到前一个要求位置 6000 上以后，“Second”功能块将向得到的最终位置 10000 移动。



MC_MOVERELATIVE

这个功能块是库“SM3_Basic.library”的一部分。它用来命令一个受控运动，在执行时该运动指定了相对于轴的实际位置的距离。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
Execute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
Distance	LREAL	0	运动的相对距离（以技术单位[u]表示）。
Velocity	LREAL	0	速度最大值（总是为正）（不是必须达到）[u/s]
Acceleration	LREAL	0	加速度值（总是为正）（电机能量增长） [u/s^2]
Deceleration	LREAL	0	减加速度值（总是为正）（电机能量减少） [u/s^2]

VAR_OUTPUT			
Done	BOOL	FALSE	如果达到要求的距离为 TRUE
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
CommandAborted	BOOL	FALSE	如果该命令已被其他命令终止则为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

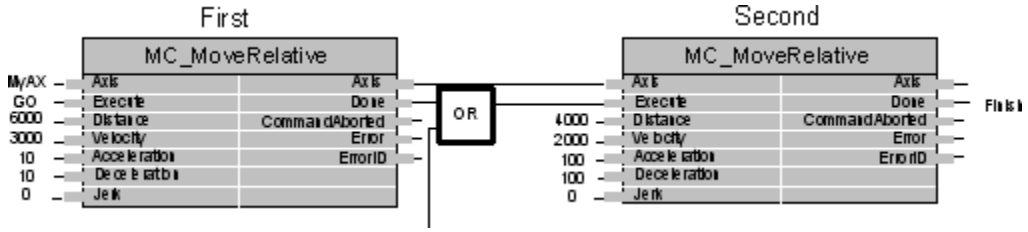
注意：如果没有其他动作挂起，该动作完成时速度为 0。

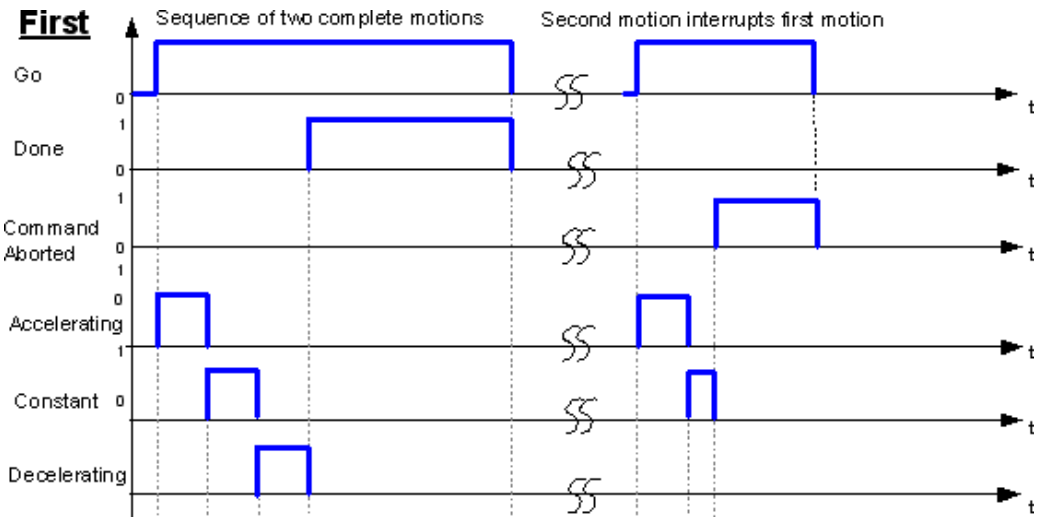
例子：

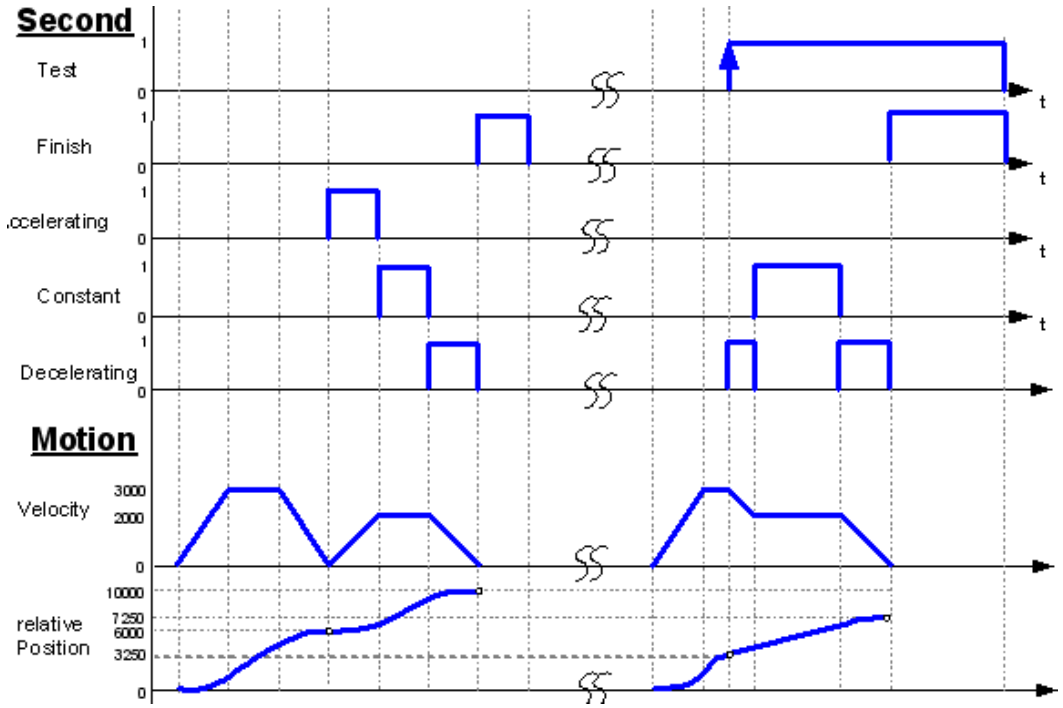
下图展示了“MC_MoveRelative”功能块的两个实例“First”和“Second”连接的例子：

时间图的左部画出了“Second”在“First”之后调用的情况。当“First”达到要求的位置 6000（速度为 0）时，输出 Done 将引发“Second”向位置 10000 移动。

时间图的右部画出了当“First”还在执行时“Second”被启动的情况。这时，“First”的运动被中断，并被在“First”速度恒定期间传递的 Test 信号终止。将距离 4000 加上实际位置 3250 后，“Second”功能块将把轴移向得到的新位置 7250。







MC_MOVESUPERIMPOSED

这个功能块是库“SM3_Basic.library”的一部分。它用来命令一个受控运动，该运动将指定的相对距离加到已有的运动中。已有的运动不会被中断，而是叠加附加运动。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
Execute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
Distance	LREAL	0	要叠加的距离（以技术单位[u]表示）
VelocityDiff	LREAL	0	与正在进行的运动的最大速度差（不是必须达到）[u/s]
Acceleration	LREAL	0	加速度值（总是为正）（电机能量增长） [u/s²]
Deceleration	LREAL	0	减加速度值（总是为正）（电机能量减少） [u/s²]
VAR_OUTPUT			
Done	BOOL	FALSE	如果指定距离已经被叠加到当前运动中为 TRUE
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
CommandAborted	BOOL	FALSE	如果该命令已被其他命令终止则为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

注意：

当活动时，除 MC_MoveSuperImposed 以外的所有其它处于终止模式的命令将终止这两个命令：MC_MoveSuperImposed 以及它下面的运动命令。如果处于其它模式，则下面的运动命令将不会被终止。

如果当一个 MC_MoveSuperImposed 活动时另一个 MC_MoveSuperImposed 也被启动，则当前正运行的 MC_MoveSuperImposed 命令将被终止并被新 MC_MoveSuperImposed 命令替代，同时下面的运动命令仍然有效。

MC_MoveSuperimposed 功能块将改变速度，如果合适的话，它还会改变处于所有相关状态的正在运行的运动的要求位置。

在 StandStill 状态 MC_MoveSuperimposed 功能块的作用象 MC_MoveRelative 一样。

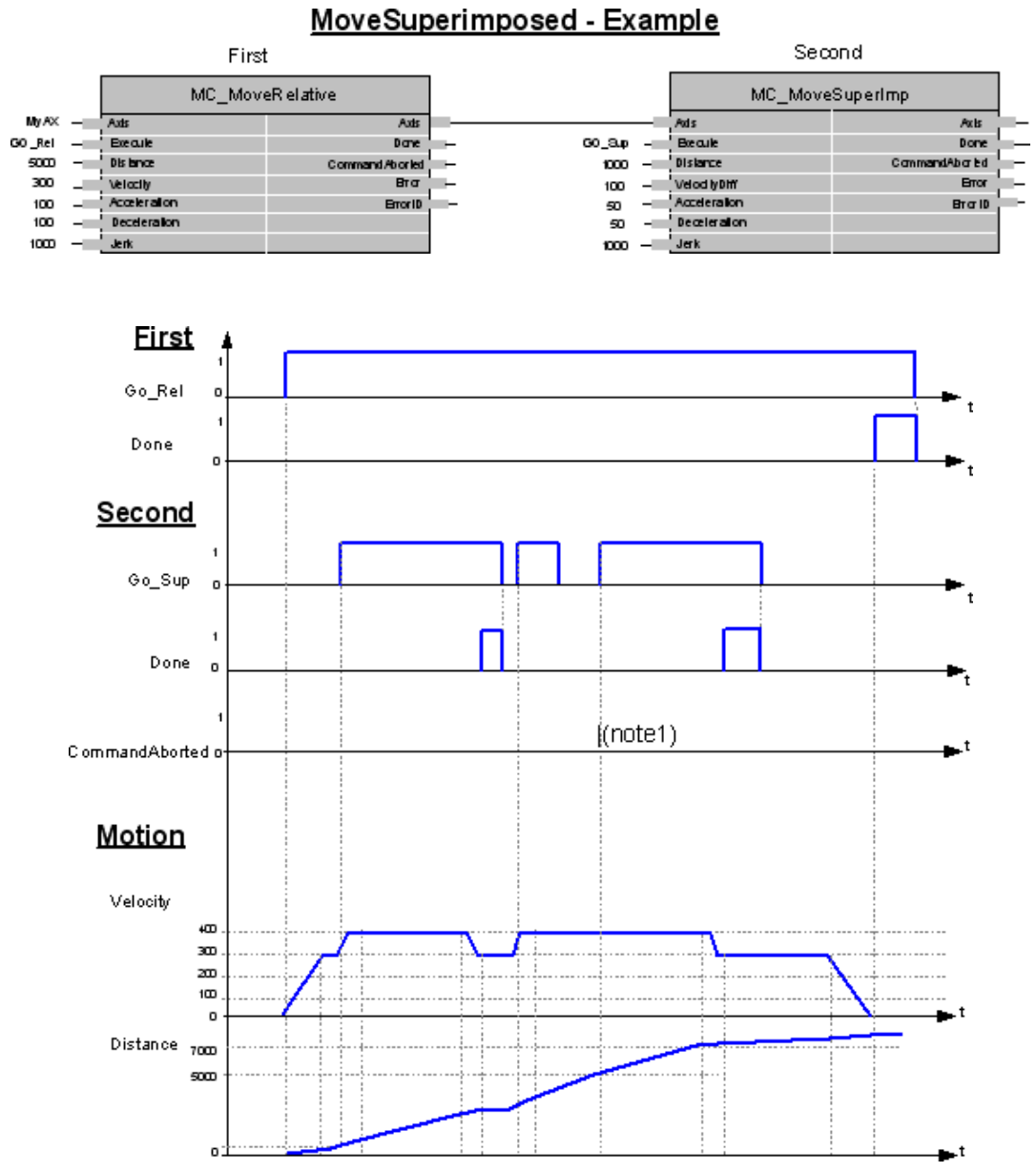
加速度值和减加速度值是当前运动的相加值，而不是绝对值。下面的功能块将在同一时间周期内完成自己的工作，而不管 MC_MoveSuperimposed 的发生。

当 MC_Phasing 作用于主设备一方时，MC_MoveSuperimposed 作用于从轴。

例子：

下图展示了 MC_MoveSuperImposed 功能块的两个实例“First”和“Second”连接的例子。这里 CommandAborted 是不可见的，因为新命令作用在同一个实例上。终止位置在 7000 和 8000 之间，它取决于对 MC_MoveSuperimposed 设置的第二个命令的终止时间。

MC_MoveSuperimposed 时间图：



MC_MOVEVELOCITY

这个功能块是库“SM3_Basic.library”的一部分。它用来以特定速度命令一个永不终止的受控运动。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
Execute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
Velocity	LREAL	0	与正在进行的运动的最大速度差（不是必须达到）[u/s]
Acceleration	LREAL	0	加速度值（总是为正）（电机能量增长） [u/s2]
Deceleration	LREAL	0	减加速度值（总是为正）（ 电机能量减少） [u/s2]
Direction	MC_Direction	current	可能值：正，负，当前值；不能应用的！：最短，最快
VAR_OUTPUT			
InVelocity	BOOL	FALSE	如果首次达到命令的速度则为 TRUE。
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
CommandAborted	BOOL	FALSE	如果该命令已被其他命令终止则为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

注意：

要停止运动，该功能块必须被其它发布新命令的功能块中断。

如果该功能块被其它功能块终止或者遇到“Execute”的下降沿，则信号“InVelocity”必须被重置。

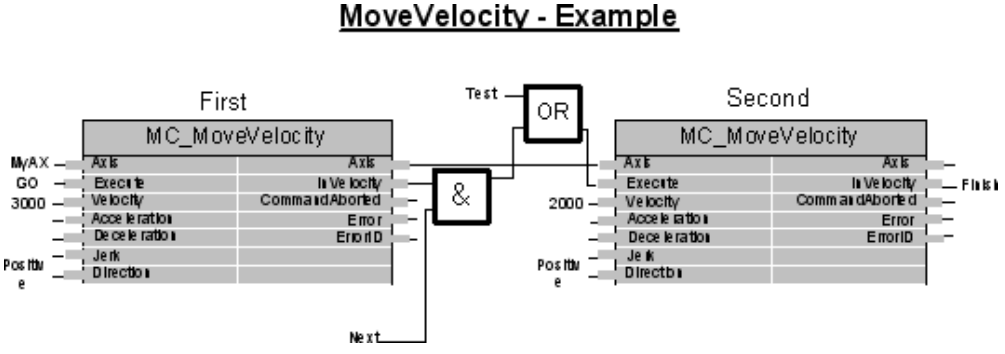
与 MC_MoveSuperimposed 相连接时，一旦轴的速度定位点已达到所要求的速度，输出“InVelocity”将保持 TRUE。

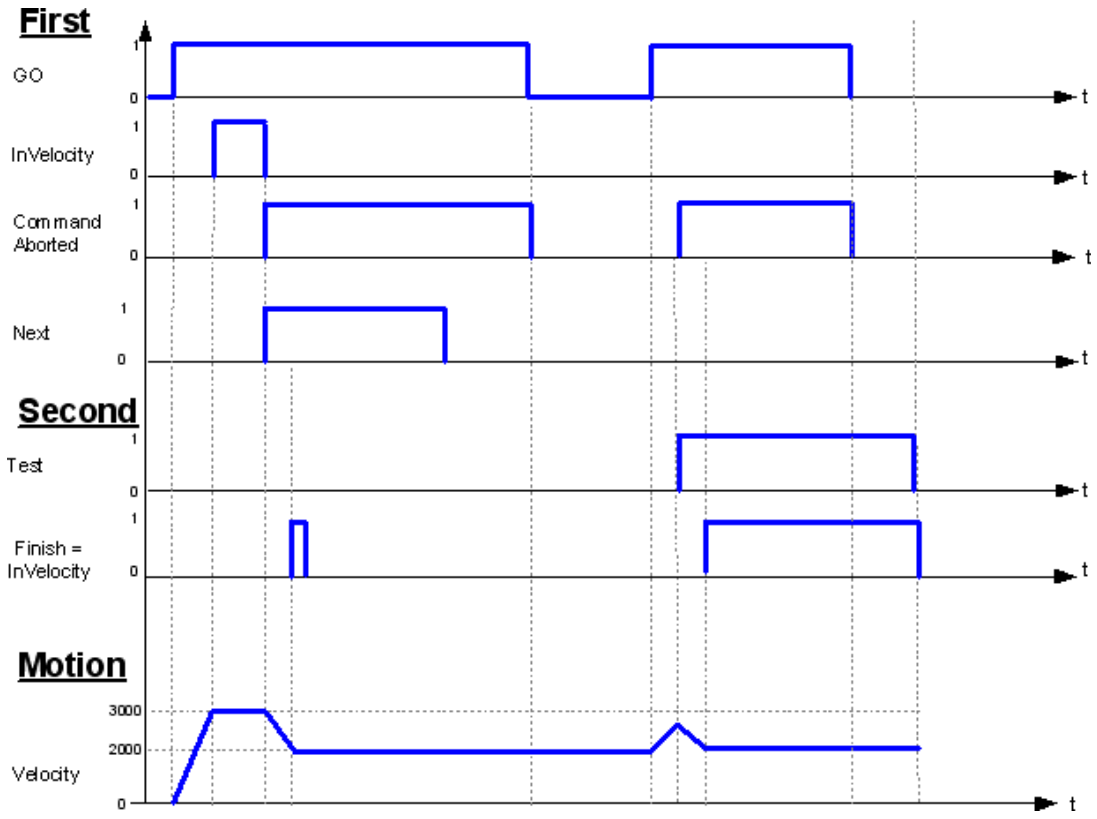
例子：

下图展示了 MC_MoveVelocity 功能块的两个实例“First”和“Second”连接的例子。

时间图的左部画出了“Second”在“First”之后调用的情况。当“First”达到要求的速度 3000 时，“输出 First.InVelocity AND 信号 Next”将引发“Second”向距离 2000 移动。

时间图的右部画出了当“First”的输出 InVelocity 还没被设置为 TRUE 时“Second”被启动的情况。“First”仍在执行。尽管“First”功能块仍在向 3000 加速，它还是会因为启动功能块“Second”的 Test 信号而被中断并被终止。现在，功能块“Second”将减低速度至 2000。





MC_POSITIONPROFILE

这个功能块是库“SM3_Basic.library”的一部分。它用来命令一个时间－位置锁定的运动分布。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
TimePosition	MC_TP_REF		关于时间/位置描述，见 MC_TP_REF 的描述。
VAR_INPUT			
Execute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
ArraySize	INT	0	数组维数
PositionScale	LREAL	1	总体位置缩放因数[t.u.]
Offset	LREAL	0	分布的总体偏移[u]
VAR_OUTPUT			
Done	BOOL	FALSE	如果分布已经完成则为 TRUE。
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
CommandAborted	BOOL	FALSE	如果该命令已被其他命令终止则为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

注意：

MC_TP_REF 是指定数据类型的提供者。该数据类型的例子如下：时间/位置的内容可表示为 DeltaTime/Pos，这里 Delta 表示相邻两点间在时间上的差值：

TYPE MC_TP

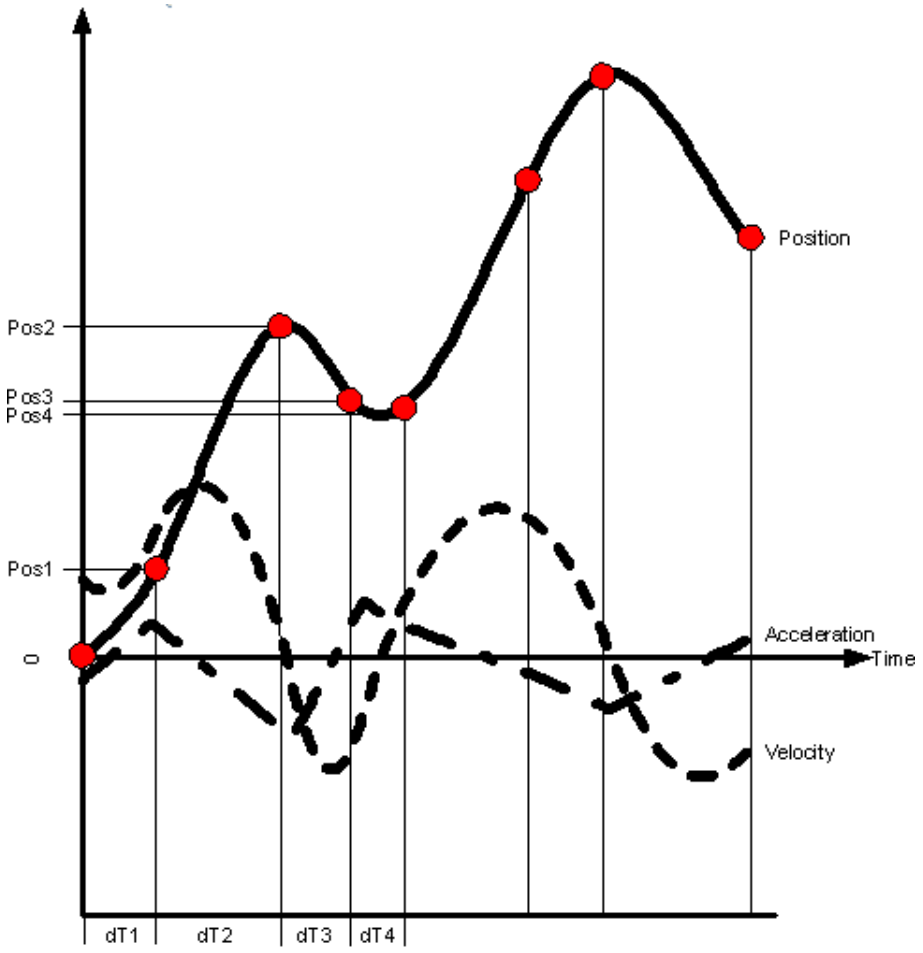
STRUCT

```
delta_time : TIME;
  position : REAL:
END_STRUCT
END_TYPE

TYPE MC_TP_TABLE
STRUCT
Number_of_pairs : INT;
IsAbsolute : BOOL;
MC_TP_Array : ARRAY [1..N] of MC_TP;
END_STRUCT
END_TYPE
```

在不同分布间的切换可被管理。

做为 MC_PositionProfile 的替代，可以使用与虚拟主设备相连的 CAM 功能块。

例子：时间/位置图示



deltaTime	absPos
dT1	Pos1
dT2	Pos2
dT3	Pos3
dT4	Pos4

MC_POWER

这个功能块是库“SM3_Basic.library”的一部分。它用来控制 power stage（“开”或“关”）。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
Enable	BOOL	FALSE	必须设为 TRUE 使功能块能够执行。
bRegulatorOn	BOOL	FALSE	必须设为 TRUE 以打开 power stage。
bDriveStart	BOOL	FALSE	必须设为 TRUE 以关闭 quickstop 装置。
VAR_OUTPUT			
Status	BOOL	FALSE	如果轴准备好移动则为 TRUE。
bRegulatorRealState	BOOL	FALSE	power stage 的有效状态。
bDriveStartRealState	BOOL	FALSE	如果驱动器没有被 quickstop 装置锁住则为 TRUE。
Busy	BOOL	FALSE	如果该命令已被其他命令终止则为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

注意：

调用 MC_Power 时用 Enable=FALSE 将导致轴变为状态 Disabled。

调用处于不可用状态的 MC_Power （用 Enable=TRUE）要么将导致停滞（如果轴没有错误发生的话），要么导致 ErrorStop （如果错误存在）。

当 Command 在一段时间内为 TRUE 并且 Status 保持为 FALSE，可以用功能块 Timer 和 AND 函数设置一个错误变量。这样可以用 power stage 指示硬件故障。

当电源故障时（也可能在操作中），转换到 ErrorStop 状态将被执行。

MC_READACTUALPOSITION

这个功能块是库“SM3_Basic.library”的一部分。它将返回参考轴的实际位置。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
Enable	BOOL	FALSE	当为 TRUE 时，参数值连续读出。
VAR_OUTPUT			
Valid	BOOL	FALSE	如果有效输出可用则为 TRUE。
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。
Position	LREAL	0	新的绝对位置（以轴的单位表达[u]）。

MC_READAXISERROR

这个功能块是库“SM3_Basic.library”的一部分。它用来描述与功能块无关的普通轴错误。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
Enable	BOOL	FALSE	当为 TRUE 时，参数值连续读出。
VAR_OUTPUT			
Valid	BOOL	FALSE	如果有效输出可用则为 TRUE。
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。
AxisError	BOOL	FALSE	指示轴错误的标志。
AxisErrorID	DWORD	0	轴错误的厂家设定值。
SWEndSwitchActive	BOOL	FALSE	如果超过软件限制则为 TRUE。

注意：该功能块相当于用 MC_ReadParameter 读取 AxisErrorID 函数。

MC_READBOOLPARAMETER

这个功能块是库“SM3_Basic.library”的一部分。它返回厂家指定的 BOOL 类型的参数。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
Enable	BOOL	FALSE	当为 TRUE 时，参数值连续读出。
ParameterNumber	DINT	0	参数编号。做为 VAR CONST 声明的符号参数名也可 使用。
VAR_OUTPUT			
Valid	BOOL	FALSE	如果参数可用则为 TRUE。
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。
Value	BOOL	FALSE	指定参数的值。

MC_READSTATUS

这个功能块是库“SM3_Basic.library”的一部分。关于当前进程中的运动，它返回轴的详细状态。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
Enable	BOOL	FALSE	当为 TRUE 时，参数值连续读出。
VAR_OUTPUT			
Valid	BOOL	FALSE	如果有效输出可用则为 TRUE。
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

Disabled	BOOL	FALSE	如果轴状态为Disabled 则为TRUE（见PLCopen 状态图）
Errorstop	BOOL	FALSE	如果轴状态为 Errorstop 则为 TRUE（见 PLCopen 状态图）
Stopping	BOOL	FALSE	如果轴状态为Stopping 则为TRUE（见 PLCopen 状态图）
StandStill	BOOL	FALSE	如果轴状态为Standstill 则为TRUE（见PLCopen 状态图）
DiscreteMotion	BOOL	FALSE	如果轴状态为 DiscreteMotion 则为 TRUE（见 PLCopen 状态图）
ContinuousMotion	BOOL	FALSE	如果轴状态为ContinuousMotion 则为TRUE（见 PLCopen 状态图）
SynchronizedMotion	BOOL	FALSE	如果轴状态为 SynchronizedMotion 则为 TRUE（见 PLCopen 状态图）
Homing	BOOL	FALSE	如果轴状态为 Homing 则为 TRUE（见 PLCopen 状态图）
ConstantVelocity	BOOL	FALSE	如果电机以恒定速度移动则为 TRUE（见 PLCopen 状态图）
Accelerating	BOOL	FALSE	如果电机能量正在增加则为 TRUE
Decelerating	BOOL	FALSE	如果电机能量正在减少则为 TRUE
FBErrorOccured	BOOL	FALSE	如果功能块错误被探测到并且还没有由 SMC_ClearFBError 清除，则为 TRUE

MC_READPARAMETER

这个功能块是库“SM3_Basic.library”的一部分。它返回厂商指定的参数。如果有必要，返回值将被转化为 LREAL。如果不可能，厂商必须提供自己的功能块来读取参数。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
Enable	BOOL	FALSE	当为 TRUE 时，参数值连续读出。
ParameterNumber	DINT	0	参数 ID，见 AXIS_REF_SM3。做为VAR CONST 声明的符号参数名也可 使用。
VAR_OUTPUT			
Valid	BOOL	FALSE	如果参数可用则为 TRUE。
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。
Value	LREAL	0	指定参数的值。

MC_RESET

这个功能块是库“SM3_Basic.library”的一部分。它通过重置所有内部关于轴的错误，把 ErrorStop 转换成 StandStill 状态－这会影响功能块实例的输出。

	类型	初始值	描述
VAR_IN_OUT			

Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
Execute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
VAR_OUTPUT			
Done	BOOL	FALSE	如果重置被执行则为 TRUE。
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

注意：如果 MC_RESET 应用于别的状态，那么状态 ErrorStop 应由厂家设定。

MC_STOP

这个功能块是库“SM3_Basic.library”的一部分。它的执行将导致受控运动停止并将轴设置为状态“Stopping”。因此，所有正在进行的功能块执行将被终止。对于一个处于“Stopping”状态的轴，任何功能块不允许对它执行任何运动。当该轴达到速度 0，输出 Done 将被立即设置为 TRUE。当 Execute 是 TRUE 或者还没达到零速度时，轴将保持状态“Stopping”。 一旦“Done”设为 TRUE（就是说“Execute”为 FALSE），轴将进入状态“StandStill”。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
Execute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
Deceleration	LREAL	0	加速度值[u/s^2]
VAR_OUTPUT			
Done	BOOL	FALSE	如果零速度已达到且“Execute”不再为 TRUE 时，为 TRUE。
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

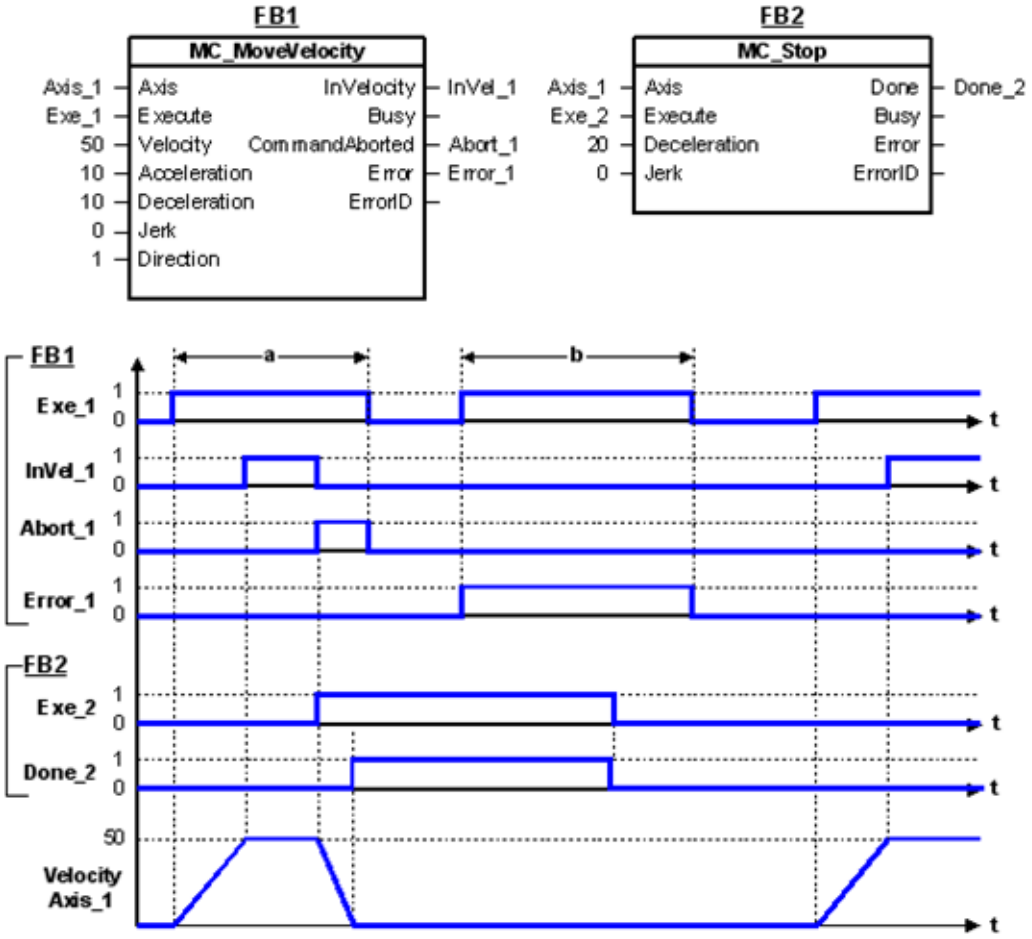
注意：当 Execute 为高电平，轴保持状态“Stopping”，而且不可执行任何其他命令。

例子：下面的例子展示了与 MC_MoveVelocity 连接的 MC_STOP 的动作。

一个旋转的轴被功能块 MC_Stop 减速。

当 MC_Stop 参数“Execute”＝TRUE 时，轴将拒绝任何运动命令。功能块 MC_MoveVelocity 将报告一个指示 MC_Stop 忙的错误。

与 MC_MoveVelocity 连接的 MC_STOP 的行为：



MC_VELOCITYPROFILE

这个功能块是库“SM3_Basic.library”的一部分。它用来命令一个时间－速度锁定的运动分布。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
TimeVelocity	MC_TV_REF		关于时间/速度描述，见 MC_TV_REF
VAR_INPUT			
Execute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
ArraySize	INT	0	数组大小
VelocityScale	LREAL	1	整体速度缩放因数[t.u.]
Offset	LREAL	0	整体分布偏移[u]
VAR_OUTPUT			
Done	BOOL	FALSE	如果分布已经完成则为 TRUE。
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
CommandAborted	BOOL	FALSE	如果该命令已被其他命令终止则为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

注意：

在不同分布间的切换可被管理。

做为 MC_VelocityProfile 的替代，与虚拟主设备相连的 CAM 功能块可被使用。

MC_WRITEBOOLPARAMETER

这个功能块是库“SM3_Basic.library”的一部分。它可用来修改厂家设定的布尔类型的参数。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
Execute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
ParameterNumber	DINT	0	参数 ID，见 AXIS_REF_SM3。
Value	BOOL	FALSE	指定参数要被写入的值。
VAR_OUTPUT			
Done	BOOL	FALSE	如果指定距离已经被叠加到当前运动中为 TRUE
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

MC_WRITEPARAMETER

这个功能块是库“SM3_Basic.library”的一部分。它可用来修改厂家设定的参数。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
Execute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
ParameterNumber	DINT	0	参数 ID，见 AXIS_REF_SM3。
Value	LREAL	0	指定参数要被写入的值。
VAR_OUTPUT			
Done	BOOL	FALSE	如果指定距离已经被叠加到当前运动中为 TRUE
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

MC_ABORTTRIGGER

这个功能块是库“SM3_Basic.library”的一部分。它是用来终止与触发事件（例如 MC_TouchProbe）相连接的功能块。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3
TriggerInput	TRIGGER_REF		关于触发信号源，见 TRIIGGER_REF
VAR_INPUT			
Execute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
VAR_OUTPUT			

Done	BOOL	FALSE	当闭锁被成功终止时为 TRUE。
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error

MC_DIGITALCAMSWITCH

这个功能块是库“SM3_Basic.library”的一部分。和电动机轴上的开关类似：功能块控制一组离散输出位元，来开关一组就像是由机械凸轮控制的、与轴相连的开关。向前和向后运动都是允许的。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于开关量连接的轴，见 AXIS_REF_SM3
Switches	MC_CAMSWITCH_REF		关于开关动作，见 HMC_CAMswitch
Outputs	MC_OUTPUT_REF		关于直接相关于参考轨迹的信号输出， 见 MC_OUTPUT_REF
TrackOptions	MC_TRACK_REF		关于包含与属性相关的跟踪的结构，例如每次输出/跟踪时的开关补偿。见 MC_TRACK_REF。
VAR_INPUT			
Enable	BOOL	FALSE	必须被设为 TRUE，使得功能块可被执行。
EnableMask	DWORD	16#FFFFFFFF	32位布尔值允许不同的跟踪。最不重要的数据对应最小的 TrackNumber 。
TappetMode	MC_TAPPETMODE	tp_mode_auto	位置输出模式，见 MC_TAPPETMODE
VAR_OUTPUT			
InOperation	BOOL	FALSE	如果所有已命令的跟踪都允许则为 TRUE。
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

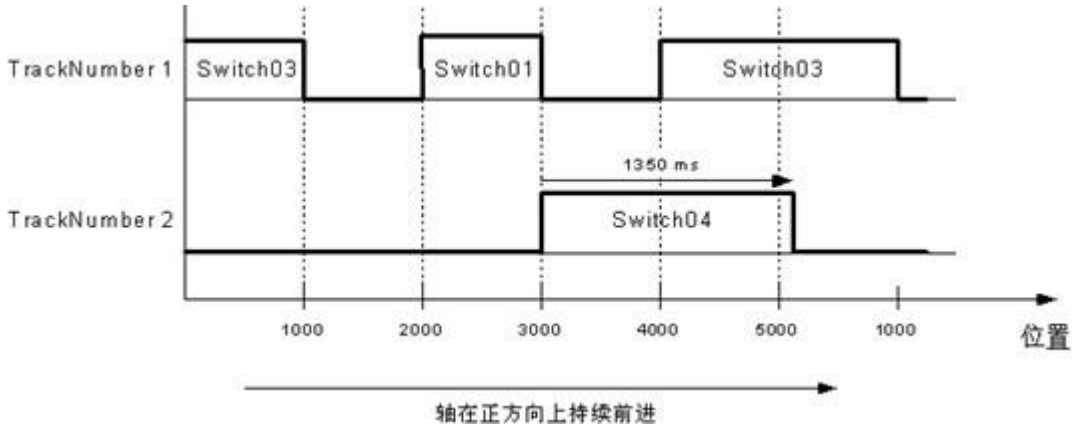
注意：该功能有时称为 PLS — 相位或位置或可编程限制开关。

例子：假设下列参数值设为：

参数	类型	开关01	开关02	开关03	开关04
TrackNumber	INT	1	1	1	2
FirstOnPosition [u]	REAL	2000	2500	4000	3000
LastOnPosition [u]	REAL	3000	3000	1000	-
AxisDirection	INT	1=正向	2=负向	0=双向	0=双向
CamSwitchMode	INT	0=位置	0=位置	0=位置	1=时间
Duration	TIME	-	-	-	1350

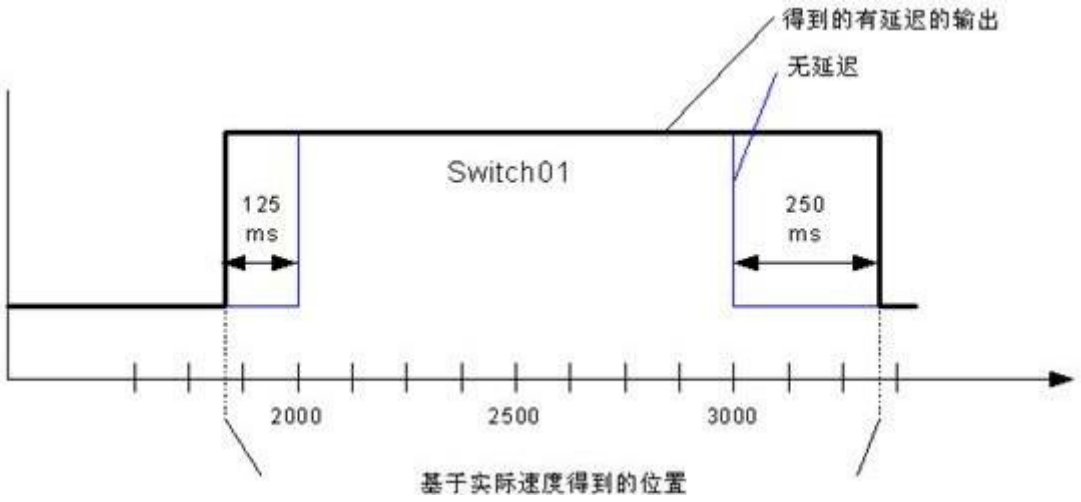
进一步假设，没有开/关补偿也没有迟滞现象发生。下图展示了当轴持续在正方向上前进时的输出行为。该轴是一个模长为 5000 单位的模数轴。

DigitalCamSwitch 的例子：



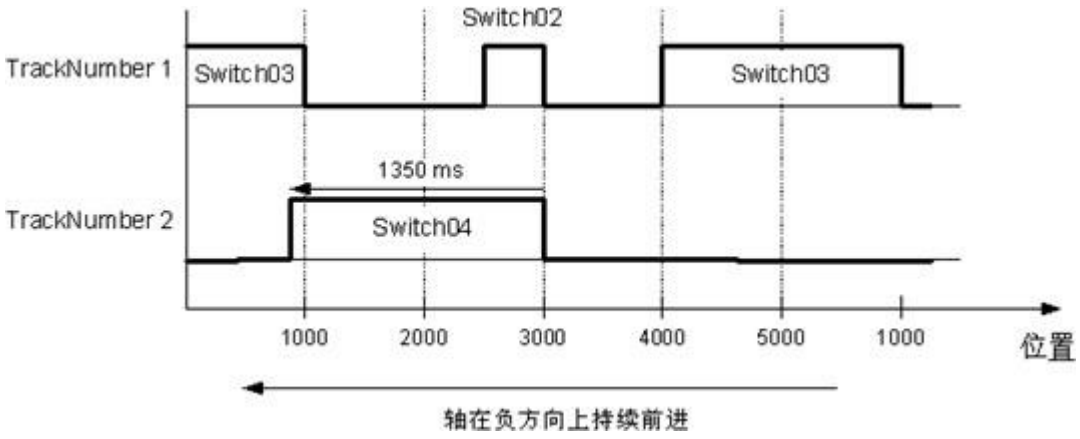
现在假设 OnCompensation（-125ms），OffCompensation (+250ms)存在。

Switch01 的详细描述：



当轴持续在负方向上前进时的输出行为，无开/关补偿，无迟滞现象。

在负方向上的例子：



MC_READACTUALTORQUE

这个功能块是库“SM3_Basic.library”的一部分。当 Enable 保持为 TRUE 时，功能块将返回实际转矩或力的值。一旦数据输出“Torque”变为有效，Done 将被设为 TRUE。如果 Enable 被重置，数据将失去它的有效性，Done 也将不管新数据的有效性而被重置。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
Enable	BOOL	FALSE	当为 TRUE 时，参数值连续读出。
VAR_OUTPUT			
Valid	BOOL	FALSE	如果有效输出可用则为 TRUE。
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。
Torque	LREAL	0	实际转矩或力的值（以技术单位表达）。

MC_READACTUALVELOCITY

这个功能块是库“SM3_Basic.library”的一部分。当 Enable 保持为 TRUE 时，功能块将返回实际速度的值。一旦数据输出“Velocity”变为有效，Done 将被设为 TRUE。如果 Enable 被重置，数据将失去它的有效性，Done 也将不管新数据的有效性而被重置。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
Enable	BOOL	FALSE	当为 TRUE 时，参数值连续读出。
VAR_OUTPUT			
Valid	BOOL	FALSE	如果有效输出可用则为 TRUE。
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。
Velocity	LREAL	0	实际速度的值（以[技术单位/秒]表达）。

MC_SETPOSITION

这个功能块是库“SM3_Basic.library”的一部分。它用来移动一个轴的坐标系。因此，该功能块将用相同的值操作指定点位置和轴的实际位置而不引起任何运动。这种重新校准可以为实例参考位置使用。在运动时调用该功能块不改变所命令的位置，该位置将被置于移动的坐标系中。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
Execute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
Position	LREAL		位置单位[u]（表示“距离”如果 Mode = RELATIVE）
Mode	BOOL	FALSE	TRUE = 相对的, FALSE = 绝对的
VAR_OUTPUT			
Done	BOOL	FALSE	如果位置已经被分配了新值则为 TRUE。
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

注意：“RELATIVE”表示，参数位置的值将和实际值相加。ABSOLUTE 意思是，实际值将被设置为参数位置的值。

MC_TOUCHPROBE

这个功能块是库“SM3_Basic.library”的一部分。它用来在触发事件发生时记录轴的位置。

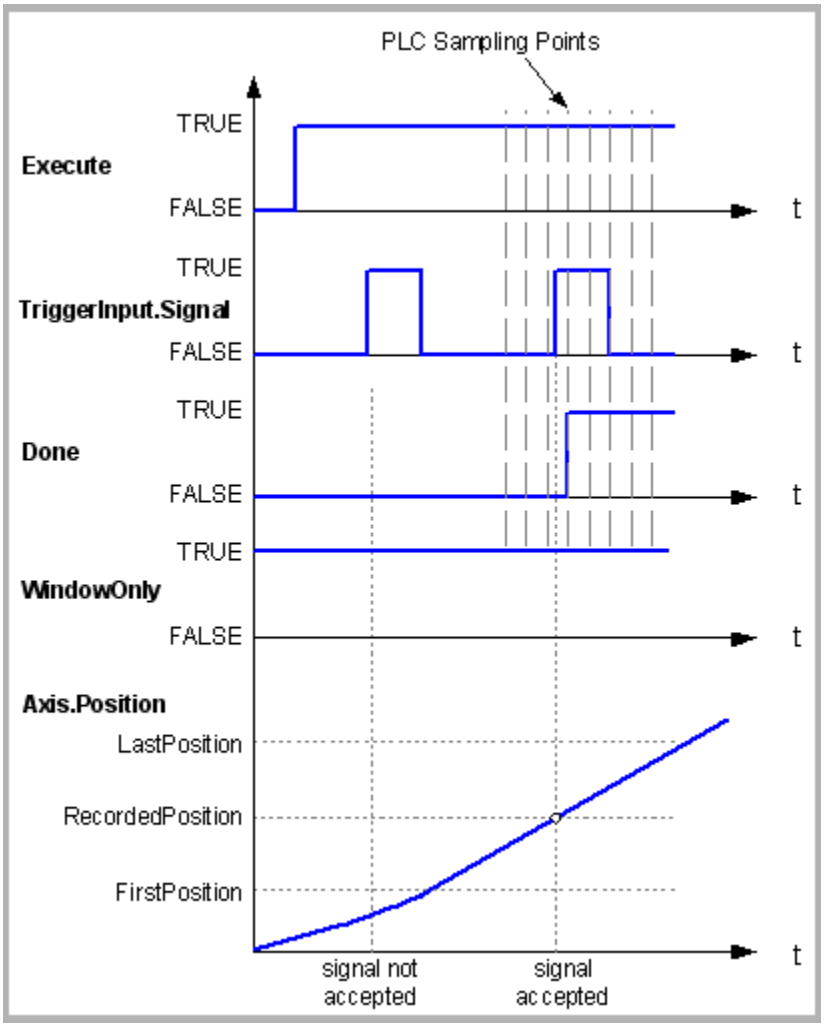
	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，在指定的事件上有触发输入时，应被记录的位置。见 AXIS_REF_SM3。
TriggerInput	TRIGGER_REF		关于触发信号源，见 TRIGGER_REF。
VAR_INPUT			
Execute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
WindowOnly	BOOL	FALSE	如果为 TRUE，触发事件将仅在指定的窗口被接受（定义见下）。
FirstPosition	LREAL	0	要被接受的触发事件的开始位置（在正方向上）（以“技术单位[u]”表示）。
LastPosition	LREAL	0	要被接受的触发事件的终止位置（在正方向上）（以“技术单位[u]”表示）。
VAR_OUTPUT			
Done	BOOL	FALSE	如果触发事件已经被记录则为 TRUE。
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。
RecordedPosition	LREAL	0	触发事件发生的位置（以“技术单位[u]”表示）。
CommandAborted	BOOL	FALSE	如果该命令已被其他命令终止则为 TRUE。

注意：

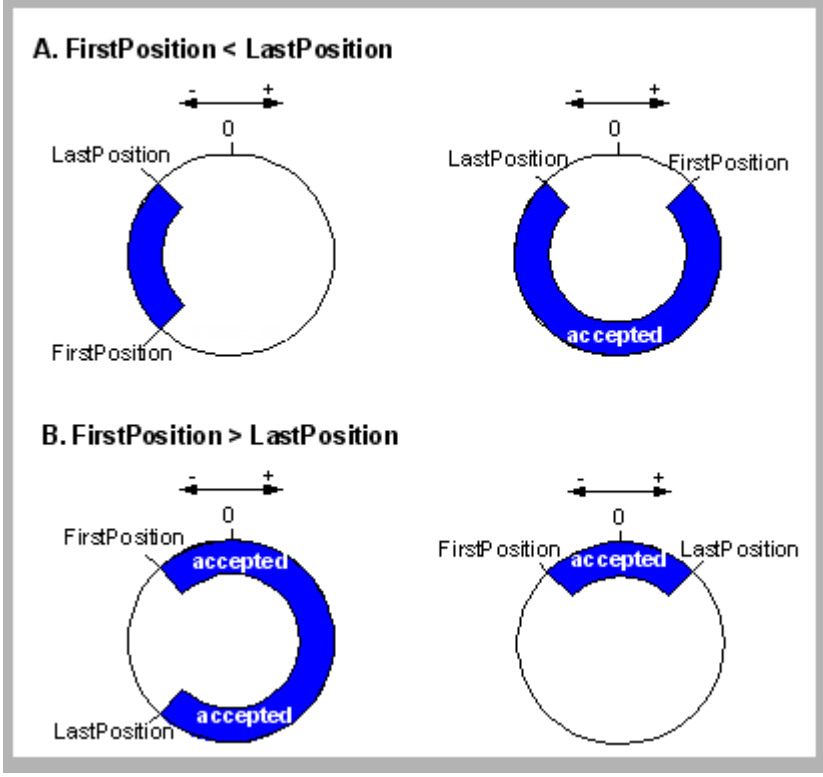
功能块是单步执行的，就是说，只有紧跟“Execute”上升沿的第一个事件是对记录有效的。其他事件将被忽略。

一个函数实例只能代表一个命令。

Touchprobe 的定时例子：



Examples of windows, where trigger events are accepted (for modulo axes)



SMC_MOVECONTINUOUSABSOLUTE

这个功能块是库“SM3_Basic.library”的一部分。它执行绝对运动，但是与 MC_MoveAbsolute 不同，它不是以速度＝0 到达目标位置的，而是以指定速度。它用来生成 via-movements。当到达目标位置时，功能块设置 InEndVelocity 并保持速度直到被其他驱动设备中断。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
Execute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
Position	LREAL	0	运动的目标位置 (以技术单位[u]表示)
Velocity	LREAL	0	速度最大值(不是必须达到)[u/s]
EndVelocity	LREAL	0	未赋值(!) 终止速度[u/s]
EndVelocityDirection	MC_Direction	current	决定终端速度的方向，详见 MC_Direction；允许的值：正，负当前值，不可应用的：最短，最快
Acceleration	LREAL	0	加速度值(电机增加能量) [u/s²]
Deceleration	LREAL	0	减加速度值(电机减少能量) [u/s²]
Direction	MC_Direction	shortest	枚举类型，详见 MC_Direction；允许的值：对于线性/有限轴：正，负；对于旋转/求模轴：正，负当前值，最短，最快
VAR_OUTPUT			
InEndVelocity	BOOL	FALSE	如果所命令的位置已达到则为 TRUE
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
CommandAborted	BOOL	FALSE	如果该命令已被其他命令终止则为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

SMC_MOVECONTINUOUSRELATIVE

这个功能块是库“SM3_Basic.library”的一部分。它执行相对运动，但是与 MC_MoveRelative 不同，它不是以速度＝0 到达目标位置的，而是以指定速度。它用来生成 via-movements。当到达目标位置时，功能块设置 InEndVelocity 并保持速度直到被其他驱动设备中断。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		关于轴，见 AXIS_REF_SM3。
VAR_INPUT			
Execute	BOOL	FALSE	输入值的上升沿将启动该功能块的执行。
Distance	LREAL	0	运动的相对距离(以技术单位[u]表示)
Velocity	LREAL	0	速度最大值(不是必须达到)[u/s]
EndVelocity	LREAL	0	未赋值(!) 终止速度[u/s]
EndVelocityDirection	MC_Direction	current	决定终端速度的方向，详见 MC_Direction；允许的值：正，负，当前值，不可应用的：最短，最快
Acceleration	LREAL	0	加速度值(电机增加能量) [u/s²]

Deceleration	LREAL	0	减加速度值(电机减少能量) [u/s²]
VAR_OUTPUT			
InEndVelocity	BOOL	FALSE	如果所命令的距离已达到并且轴正以所命令的终速度运转，则为 TRUE。
Busy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
CommandAborted	BOOL	FALSE	如果该命令已被其他命令终止则为 TRUE。
Error	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示，见 SMC_Error。

MC_Jog

MC_Jog 以试运行的方式在一个轴上进行不间断的人工移动。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		参考轴 ↘AXIS_REF_SM3
VAR_INPUT			
JogForward	BOOL		若 JogForward 为 TRUE，轴则按照给定的动态值(速度，加速度，减速度) 朝正方向移动。如果输入 JogBackward 同时为 TRUE，则不执行任何动作。
JogBackward	BOOL		若 JogBackward 为 TRUE，轴则按照给定的动态值(速度，加速度，减速度) 朝负方向移动。如果输入 JogForward 同时为 TRUE，则不执行任何动作。
Velocity	LREAL		定义移动的速度
Acceleration	LREAL		定义移动时的加速度
Deceleration	LREAL		定义移动时的减速度
VAR_OUTPUT			
Busy	BOOL	FALSE	在 JogForward 或 JogBackward Busy 运动时为 TRUE。当这些指令输入为 FALSE，轴将减速至0，直到停止，Busy 才被设为 FALSE。
CommandAborted	BOOL	FALSE	若此功能块被其它功能块中断则为 TRUE。 然后将 CommandAborted 设置与 JogForward 或 JogBackward 一样，则至少为一个周期。
Error	BOOL	FALSE	只要 JogForward 或 JogBackward 被设置，功能块内部发生错误时的符号，至少一个周期。
ErrorID	SMC_ERROR	0	错误代码，见 SMC_Error

SMC_Inch

SMC_Inch 用于在一个轴上进行逐步运动，一步一步。

	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		参考相关轴_AXIS_REF_SM3
VAR_INPUT			
InchForward	BOOL		若 InchForward 为 TRUE，轴则按照给定的动态值

			(速度，加速度，减速度) 朝正方向移动，直到距离被覆盖。输入将被再次设为 TRUE 或 FALSE 来启动后面的运动。 若 InchForward 在距离被覆盖之前设为 FALSE，轴将立即减速至0，Busy 将被设为 FALSE。 如果输入 InchBackward 同时为 TRUE，则不执行任何运动。
InchBackward	BOOL		若 InchBackward 为 TRUE，轴则按照给定的动态值 (速度，加速度，减速度) 朝负方向移动，直到距离被覆盖。输入将被再次设为 TRUE 或 FALSE 来启动后面的运动。 若 InchBackward 在距离被覆盖之前设为 FALSE，轴将立即减速至0，Busy 将被设为 FALSE。 如果输入 InchForward 同时为 TRUE，则不执行任何运动。
Distance	LREAL		定义被覆盖的距离
Velocity	LREAL		定义移动的速度
Acceleration	LREAL		定义移动时的加速度
Deceleration	LREAL		定义移动时的减速度
VAR_OUTPUT			
Busy	BOOL	FALSE	当 InchForward 或 InchBackward 动作激活时，Busy 为 TRUE。 如果命令输入为 FALSE，轴将立即减速至0，之后 Busy 被设为 FALSE。
CommandAborted	BOOL	FALSE	若此功能块被其它功能块中断则为 TRUE。 然后将 CommandAborted 设置与 InchForward 或 InchBackward 一样，则至少为一个周期。
Error	BOOL	FALSE	只要 InchForward 或 InchBackward 被设置，功能块内部发生错误时的符号，至少一个周期。
ErrorID	SMC_ERROR	0	错误代码，见 SMC_Error

5.3.5 Simple Test

SMC_StartupDrive

这个功能块是库“SM3_Basic.library”的一部分。它包含一组经常使用的功能块，它用来测试和试运转一个轴。请注意，这里面还有一个兼容的视图模板。

	类型	描述
VAR_IN_OUT		
Axis	AXIS_REF	关于轴，见 AXIS_REF_SM3。

6 SM3_CNC Library 库使用说明

SM3_CNC.Library 库里的模块用来（依照 DIN 66025）解码在 CNC 编辑器中创建的轨迹。解码是为了转化轨迹到一个结构中，以使其能够被 IEC 程序所用（轨迹预处理），或能够被修改、被插补以及被转换以使其对于驱动设备是可读的（例如，SMC_NCDecoder，ToolCorr，SMC_AvoidLoop，SMC_SmoothPath，SMC_RoundPath，SMC_Interpolator）。

模块的功能故障允许某些组件－比如解码器－被为特殊需要而设计的模块所代替。此外，其他模块－比如某个轨迹预处理或者优化－可以被毫无问题地加入到已有组件中，这样不需要考虑库中组件间的适配性。因此，程序逻辑完全由 PLC 程序处理，只有纯动作信息才由库模块处理。

库 SM3_CNC.Library 的数据结构（例如 SMC_POSINFO，SMC_VECTOR3D）用来描述位置、轨迹段和向量，并管理 GEOINFO 对象（QUTQUEUE 结构）。

表 5-1 SM3_CNC.Library 库指令一览表

数据类型记录
SMC_CNC_REF
SMC_GCODE_TEXT
SMC_GCODEVIEWER_DATA
SMC_GCODE_WORD
SMC_GEOINFO
SMC_M_PARAMETERS
SMC_OUTQUEUE
SMC_POSINFO
SMC_VECTOR3D
全局变量记录
SMC_AL_STATUS
SMC_DEC_STATUS
SMC_DIRECTION
SMC_INT_STATUS
SMC_INT_VELMODE
SMC_MOVTYP
SMC_TC_STATUS
SMC_TOOLCORRMODE
SMC_VECTORDIR
功能块记录
在 SMC_CNC_POUs\File 文件夹内：
SMC_READNCFILE
SMC_READNCQUEUE
SMC_VARLIST
SMC_CNC_POUs\SoftMotion CNC\Coordinate Transformation 文件夹内：
SMC_DETERMINECUBOIDBEARING
SMC_COORDINATETRANSFORMATIONS3D
SMC_INVCOORDINATETRANSFORMATION3D
SMC_TEACHCOORDINATESYSTEM

SMC_UNITVECTORTORPY
SMC_ROTATEQUEUE2D
SMC_SCALEQUEUE3D
SMC_TRANSLATEQUEUE3D
SMC_CNC_POUs\SoftMotion CNC\Direct Axis Control 文件夹内：
SMC_CONTROLAXIXBYPOS
SMC_CONTROLAXIXBYPOSVEL
SMC_CNC_POUs\SoftMotion CNC\GCode Viewer 文件夹内：
SMC_GCODEVIEWER
SMC_CNC_POUs\SoftMotion CNC\Posinfo Functions 文件夹内：
SMC_POSINFO
SMC_CNC_POUs\SoftMotion CNC\OutQueue Functions 文件夹内：
SMC_APPENDOBJ
SMC_DELETEOBJ
SMC_GETCOUNT
SMC_GETOBJ
SMC_GETOBJFROMEND
SMC_OUTQUEUEINIT
SMC_RESTOREQUEUE
SMC_SETQUEUECAPACITY
SMC_CNC_POUs\SoftMotion CNC\SoftMotion Function Blocks 文件夹内：
SMC_AVOIDLOOP
SMC_CHECKFORLIMITS
SMC_CHECKVELOCITIES
SMC_EXTENDEDVELOCITYCHECKS
SMC_INTERPOLATOR
SMC_INTERPOLATOR2DIR
SMC_INTERPOLATOR2DIR_SLOWTASK
SMC_LIMITADDAXISDYNAMICS
SMC_LIMITCIRCULARVELOCITY
SMC_NCDECODER
SMC_ROUNDPATH
SMC_SMOOTHPATH
SMC_TOOLCORR
SMC_XINTERPOLATOR
SMC_GETMPARAMETERS
SMC_PREACKNOWLEDGEMFUNCTIONS
SM_Trafo_POUs\Additional FBs 文件夹内：
SMC_CALCDIRECTIONFROMVECTOR
SM_Trafo_POUs\Gantry Systems 文件夹内：
SMC_TRAFO_GANTRY2
SMC_TRAFO_GANTRY2TOOL1
SMC_TRAFO_GANTRY2TOOL2
SMC_TRAFO_GANTRY3
SMC_TRAFO_GANTRYCUTTER2
SMC_TRAFO_GANTRYCUTTER3

SMC_TRAFO_GANTRYH2 SMC_TRAFOF_GANTRY2 SMC_TRAFOF_GANTRY2TOOL1 SMC_TRAFOF_GANTRY2TOOL2 SMC_TRAFOF_GANTRY3 SMC_TRAFOF_GANTRYCUTTER2 SMC_TRAFOF_GANTRYCUTTER3 SMC_TRAFOF_GANTRYH2 SMC_TRAFOV_GANTRY2 SMC_TRAFOV_GANTRY3 SMC_TRAFOV_GANTRYCUTTER2 SMC_TRAFOV_GANTRYCUTTER3 SMC_TRAFOV_GANTRYH2
SM_Trafo_POUs\Parallel Systems 文件夹内： SMC_TRAFO_BIPOD_ARM SMC_TRAFO_TRIPOD SMC_TRAFO_TRIPOD_ARM SMC_TRAFOF_BIPOD_ARM SMC_TRAFOF_TRIPOD SMC_TRAFOF_TRIPOD_ARM
SM_Trafo_POUs\Scara Systems 文件夹内： SMC_TRAFO_SCARA2 SMC_TRAFO_SCARA3 SMC_TRAFOF_SCARA2 SMC_TRAFOF_SCARA3

6.1 数据类型

SMC_CNC_REF

这个数据类型是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。这个数据结构管理被解析了的 G 代码文件。

组件	类型	初始值	描述
wCNCREFStructID	WORD	16#BA56	wCNCREFStructID 的固定值允许模块在内部检查，被分配的结构变量是否是 SMC_CNC_REF 类型的。
bRestart	BOOL	0	如果在 IEC 程序内部实现了跳转（G20），bRestart 将被 SMS_NCDecoder 设置以使数据结构产生器重初始化所有指针并重新生成该数据结构。
nElements	UDINT		Pgc 的 SMC_GCODE_WORD 结构的数量
diReadPos	UDINT	0	（内部使用）
udiBuffer	UDINT	16#FFFFFFFF	（内部使用）
pgc	POINTER TO	0	指向第一个 SMC_GCODE_WORD 类型结构

	SMC_GCODE_WORD		的指针
piStartPosition	SMC_POSINFO		CNC 程序的起始位置
strProgramName	STRING	‘‘	CNC 程序的名称

SMC_GCODE_TEXT

这个数据类型是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。数据源与 SMC_NCDecoder 模块的同名输出相连。它用来向功能块 SMC_GCodeViewer 传递 CNC 文件最后一行已解码的字符串。

组件	类型	初始值	描述
str	STRING	‘‘	代表当前行的字符串
iLineNumber	INT	0	行号（从 N 字）
bNewLine	BOOL	FALSE	当本次循环中一个新行已被解码时为 TRUE；其它情况为 FALSE
bClearList	BOOL	FALSE	当解码器重新被启动时为 TRUE，这意味着存有最后几行的缓冲区将被清空。

SMC_GCODEVIEWER_DATA

这个数据类型是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。

组件	类型	初始值	描述
wGCodeViewertructID	WORD	16#6911	内部 ID
iLine	INT	0	行号
str	STRING	‘‘	STRING 类型的 G 代码行。

SMC_GCODE_WORD

这个数据类型是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。它包括某个 G 代码字如“N20”或“G1”的数据。

组件	类型	初始值	描述
fValue	LREAL	0	LREAL 类型的值
diValue	DINT	0	DINT 类型的值
pAdr	POINTER TO BYTE	0	如果不等于0，则字的值由一个变量赋给。pAdr 指向这个变量。
byLetter	BYTE	0	字的 ASCII 代码，如69或“F”。
byVarType	BYTE	0	仅当 pAdr 不为0时。指定变量类型： 1：(*TYPE_INT*) 2：(*TYPE_BYTE*) 3：(*TYPE_WORD*) 4：(*TYPE_DINT*) 5：(*TYPE_DWORD*) 6：(*TYPE_REAL*) 14:(*TYPE_SINT*) 15:(*TYPE_USINT*)

			16:(*TYPE_UINT*) 17:(*TYPE_UDINT*) 22:(*TYPE_LREAL*)

SMC_GEOINFO

这个数据类型是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。该结构是被设计用来存储轨迹对象的，也就是已编程轨迹的一段，比如一条直线、一段圆弧或者样条曲线段。除了这些对象的几何信息外更多的信息也可被保存，比如在轨迹开始/结束时的速度或者标签。

组件	类型	初始值	描述
iObjNo	INT	0	任意对象编号（与实际轨迹描述无关）。
iSourceLine_No	INT	0	CNC 程序代码行编号（与实际轨迹描述无关）。
iMoveType	SMC_MOVTYP		枚举的有效值指的是以下对象类型： LIN 1 直接移动（G01） CLW 2 顺时针方向的圆 CCLW 3 逆时针方向的圆 SPLINE 5 样条曲线，拟和（G05，G06） ELLCLW 8 顺时针椭圆 ELLCCLW 9 逆时针椭圆 LINPOS 100 直接定位 INTPOS 110 盲定位（起始点未知，将由 SMC_Interpolator 添加） MCOMMAND 120 附加选项，M 选项
piStartPos	SMC_POSINFO		结构精确地描述对象初始位置（当 iMoveType=INITPOS 时将被忽略）。
piDestPos	SMC_POSINFO		结构精确地描述对象结束位置。
dP1	LREAL	0	这些参数包含下面哪些信息取决于 iMoveType： LIN, LINPOS -（完整信息存储于 Start_Pos 和 Dest_Pos 中） CLW, CCLW dp1：圆心的 X 坐标 dp2：圆心的 Y 坐标 dp3：圆的半径 SPLINE 样条曲线参数 ELLCLW, ELLCCLW dp1：圆心的 X 坐标 dp2：圆心的 Y 坐标 dp3：主轴1向量的 X 部分 dp4：主轴1向量的 Y 部分 dp5：主轴长度 dp6：次轴长度 dp7：主轴方向（度） dp8：dp6/dp5的约束 INITPOS – MCOMMAND 通过插补器能够被 SMC_GetMParameters 显示的附加参数。
dP2	LREAL	0	
dP3	LREAL	0	
dP4	LREAL	0	
dP5	LREAL	0	
dP6	LREAL	0	
dP7	LREAL	0	
dP8	LREAL	0	
dT1	LREAL	0	LIN，LINPOS -（完整信息存储于 Start_Pos 和

dT2	LREAL	1	Dest_Pos 中） CLW，ELLCLW dT1：具有数学方向性（0=东，90=北，180=西，270=南）的起始角度（角度制）。 dT2：圆的圆心角（比如，90=1/4圆，180=半圆） CCLW, ELLCCLW dT1：具有数学方向性（0=东，90=北，180=西，270=南）的起始角度（角度制）。 dT2：圆的负圆心角（比如，-90=1/4圆，-180=半圆） SPLINE t 参数的开始和结束值（见样条曲线描述），默认值为0和1。 ELLCLW, ELLCCLW <-删除 INITPOS -
dToolRadius	LREAL	0	对于用SMC_ToolCorr 或SMC_RoundPath 进行轨迹处理，信息是必要的，其他无关。
dVel	LREAL	0	目标速度已达到（轨迹单位/秒）
dVelEnd	LREAL	0	对象结束时规定的速度（轨迹单位/秒）
dVelEndStored	LREAL	0	调用 SMC_CheckVelocities 之前在段末尾的最大速度。
dAccel	LREAL	100	允许加速度的最大值（轨迹单位/秒2）。
dDecel	LREAL	100	允许减加速度的最大值（轨迹单位/秒2 ）。
dLength	LREAL	0	以轨迹单位表示的对象长度
byInternMark	LREAL	0	第0位设置 停止对象后面的工具半径校正。 第1位设置 开始对象左面的工具半径校正 第2位设置 开始对象右面的工具半径校正 第3位设置 停止对象后面的轨迹圆整/平滑化 第4位设置 停止对象处的轨迹圆整/平滑化 第5位设置 开始对象处的轨迹圆整 第6位设置 停止对象后面的防环圈 第7位设置 停止对象处的防环圈
dHelpPos	LREAL		辅助开关量的相对位置（0：对象启动，1：对象结束，和 G 代码：O 类似）
iHelpID	LREAL		辅助开关量的 ID（G 代码：H ）。如果当前对象时 MCOMMAND 类型的，iHelpID[0]将包含 M 选项的编号。
adVelAddAx	LREAL		如果不等于0，该错误的值将指定速度值： 0：对所有 G0命令使用的速度 1：Z 轴的速度最大值；由 SMC_ExtendedVelocityChecks 使用 2：P 轴的速度最大值；由 SMC_ExtendedVelocityChecks 使用 3：Q 轴的速度最大值；由 SMC_ExtendedVelocityChecks 使用 4：U 轴的速度最大值；由 SMC_ExtendedVelocityChecks 使用 5：V 轴的速度最大值；由 SMC_ExtendedVelocityChecks 使用

			6：W 轴的速度最大值；由 SMC_ExtendedVelocityChecks 使用
adAccAddAx	LREAL		如果不等于0，该错误的值将指定加速度值： 0：对所有 G0命令使用的加速度/减加速度 1：Z 轴的加速度/减加速度最大值；由 SMC_ExtendedVelocityChecks 使用 2：P 轴的加速度/减加速度最大值；由 SMC_ExtendedVelocityChecks 使用 3：Q 轴的加速度/减加速度最大值；由 SMC_ExtendedVelocityChecks 使用 4：U 轴的加速度/减加速度最大值；由 SMC_ExtendedVelocityChecks 使用 5：V 轴的加速度/减加速度最大值；由 SMC_ExtendedVelocityChecks 使用 6：W 轴的加速度/减加速度最大值；由 SMC_ExtendedVelocityChecks 使用
adDecAddAx	BYTE		

SMC_M_PARAMETERS

这个数据类型是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。它可以为当前活动的 M 函数定义附加参数，这些参数可被 SMC_GetMParameters 计值，以显示各个 M 函数的附加值。

组件	类型	初始值	描述
dP1	LREAL	0	给 M 函数的参数，可被 SMC_GetMParameters 使用。
dP2	BYTE	0	
dP3	BYTE	0	
dP4	LREAL	0	
dP5	LREAL	0	
dP6	LREAL	0	
dP7	LREAL	0	
dP8	LREAL	0	

SMC_OUTQUEUE

这个数据类型是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。该结构用来管理一个已定义大小的表中的 GEOINFO 对象。

SoftMotion 模块 SMC_NCDecoder，SMC_SmoothPath，SMC_RoundPath，SMC_AvoidLoop 和 SMC_ToolCorr – 提供指向一个被内部处理的 OUTQUEUE 结构的指针 – 将自动初始化该结构。

模块 SMC_SmoothPath，SMC_RoundPath，SMC_ToolCorr，SMC_AvoidLoop 和 SMC_Interpolator 必须被指针传递到正确的 OUTQUEUE 表上。如果这个表是可程序化的并且被“手动”填充，正确的初始化也应手动进行。为此，头两个变量（buffer, size）必须给定。

强烈建议使用上述函数与 SMC_OUTQUEUE 结构共同使用，并避免在初始化后其他参数的任何改变。

组件	类型	初始值	描述
wOUTQUEUEStructID	WORD	16#9878	此变量的这个固定值允许模块检查，所提

<table>
<tr><td></td><td></td><td></td><td>供的机构变量是否是 SMC_OutQueue 类型的。</td></tr>
<tr><td>pbyBuffer</td><td>POINTER TO ARRAY[0..0] OF SMC_GEOINFO</td><td></td><td>分配给 GEOINFO 对象的连续的内存缓冲区地址。变量 pbyBuffer 必须在分配缓冲区的 IEC 程序中被赋值。分配是在声明部分进行的，比如，使用数组（BUFFER: ARRAY [0..99] OF SMC_GEOINFO，有能存储100个 SMC_GEOINFO 对象的内存空间）。</td></tr>
<tr><td>nCapacity</td><td>UDINT</td><td></td><td>包括了多个 SMC_GEOINFO 对象的容量。函数 SMC_SetCapacity 初始化此域。</td></tr>
<tr><td>nReadPos</td><td>UDINT</td><td></td><td>包括了列表中第一个对象的多个 SMC_GEOINFO 对象的位置。</td></tr>
<tr><td>nWritePos</td><td>UDINT</td><td></td><td>包括了多个 SMC_GEOINFO 对象的下一个空闲位置。</td></tr>
<tr><td>bFull</td><td>BOOL</td><td></td><td>一旦表被填满到超出安全缓冲区，通过模块 APPENDOBJ 该变量将被设置为 TRUE。比如，只剩下3个 GEOINFO 对象的空间。当一些元素被从表中移除后，DELETEOBJ 将重置 bFull 的值为 FALSE。</td></tr>
<tr><td>bEndOfList</td><td>BOOL</td><td></td><td>为避免数据欠载运行（SMC_Interpolator），被一个 OUTQUEUE 结构传递的 SoftMotion 模块将不会随该队列的处理而启动，而是直到它被完全填满。但是，当对最后一个轨迹的 SMC_GEOINFO 对象起作用时，OUTQUEUE 不再完整。所以，为了保持处理继续，一旦最后一个 GEOINFO 对象被存储后 bEndOfList 必须被设置为 TRUE。如果在此之后表是空的，但应被再次填满，bEndOfList 必须被重置为 FALSE。</td></tr>
<tr><td>byGenerator</td><td>BYTE</td><td>250</td><td>队列的创始者。插补器将使用此信息来核查 SMC_CheckVelocities 已经按规定被处理了。有效值是：
<table>
<tr><td>SMC_NCDecoder</td><td>1</td></tr>
<tr><td>SMC_AvoidLoop</td><td>10</td></tr>
<tr><td>SMC_LimitCircularVelocity</td><td>11</td></tr>
<tr><td>SMC_RoundPath</td><td>12</td></tr>
<tr><td>SMC_SmoothPath</td><td>13</td></tr>
<tr><td>SMC_ToolCorr</td><td>14</td></tr>
<tr><td>SMC_RotateQueue2D</td><td>30</td></tr>
<tr><td>SMC_ScaleQueue3D</td><td>31</td></tr>
<tr><td>SMC_TranslateQueue3D</td><td>32</td></tr>
<tr><td>SMC_CheckVelocities</td><td>254</td></tr>
</table></td></tr>
</table>

			CNC-Editor	255	
bReCheckVelocities	BOOL		内部变量		
strProgramName	STRING		CNC 程序名		

SMC_POSINFO

这个数据类型是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。对一个特殊位置点该结构描述了它的坐标和附加轴的位置。

组件	类型	初始值	描述
iFrameNo	INT	0	与 SoftMotion 模块不相关的附加信息可被用户存在该变量中。
wAuxData	WORD	7	由 SMC_Interpolator 计算的位置轴的逐比特描述。初始值 2#111将使 x-，y-和 z-轴被插补。第一位组将影响 dX 位置的计算，比如第7位将影响 dA2的处理。
wSProfile	WORD	0	与 wAuxData 类似地，wSProfile 的值规定了附加轴，比如 X-或者 Y-轴，应该以恒定跃升被线性（FALSE）插补或者被 S 形插补（TRUE）。第2位对应 Z 轴，第6位对应 P，第7位对应 Q，第8位对应 U，第9位对应 V，第10位对应 W。
dX	LREAL	0	坐标系中的 x 位置。
dY	LREAL	0	坐标系中的 y 位置。
dZ	LREAL	0	坐标系中的 z 位置。
dA	LREAL	0	目前没用。
dB	LREAL	0	目前没用。
dC	LREAL	0	目前没用。
dA1	LREAL	0	附加轴位置 A1.
dA2	LREAL	0	附加轴位置 A2.
dA3	LREAL	0	附加轴位置 A3.
dA4	LREAL	0	附加轴位置 A4.
dA5	LREAL	0	附加轴位置 A5.
dA6	LREAL	0	附加轴位置 A6.

SMC_VECTOR3D

该结构体描述了三维的速度。

组件	类型	初始值	描述
dX	LREAL	0	X 坐标轴方向的速度
dY	LREAL	0	Y 坐标轴方向的速度
dZ	LREAL	0	Z 坐标轴方向的速度

6.2 全局变量

SMC_AL_STATUS

这个枚举变量是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。它的值指明了 FB 实例的实际状态。可能的状态：

枚举成员	关联量	描述

AL_OFF	0	防环圈功能关
AL_ON	1	防环圈功能开
AL_END	2	对象终止处理

SMC_DEC_STATUS

这个枚举变量是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。它的值指示了 FB 实例的实际状态。可能的状态：

枚举成员	关联量	描述
WAIT_PROG	0	程序还未找到。
READ_WORD	1	读字。
PROG_READ	2	已到程序尾。

SMC_DIRECTION

这个枚举变量是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。它的值指明了功能块实例的实际插补方向。可能的状态：

枚举成员	关联量	描述
IPO_positive	1	插补正在正方向上进行（模拟原始轨迹）
IPO_standstill	0	插补暂停
IPO_negative	-1	插补正在负方向上进行（与原始轨迹方向相反）

SMC_INT_STATUS

这个枚举变量是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。它的值指示了 FB 实例的实际状态。可能的状态：

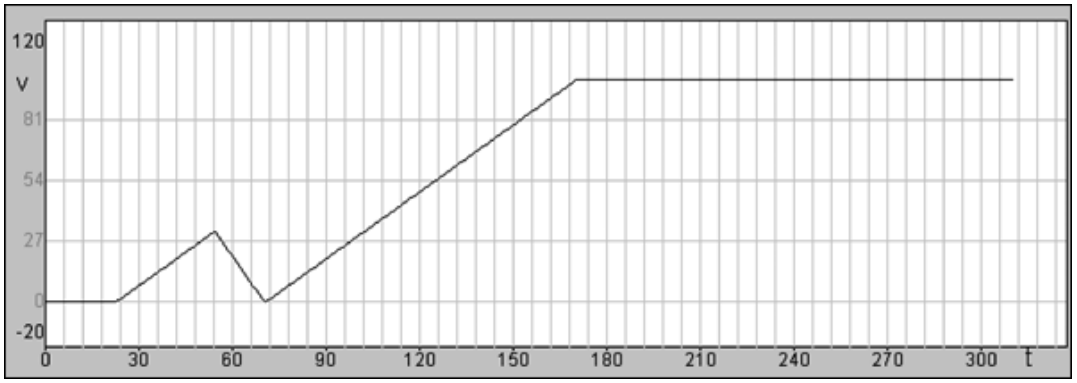
枚举成员	关联量	描述
IPO_UNKNOWN	0	内部状态。
IPO_INIT	1	FB 处于初始化阶段。
IPO_ACCEL	2	插入正在速度增加的状态下进行。
IPO_CONSTANT	3	插入正在速度恒定的状态下进行。
IPO_DECEL	4	插入正在速度减少的状态下进行。
IPO_ FINISHED	5	插入结束。
IPO_WAIT	6	当前插入暂停，例如由停止输入或一个 M 命令引发。

SMC_INT_VELMODE

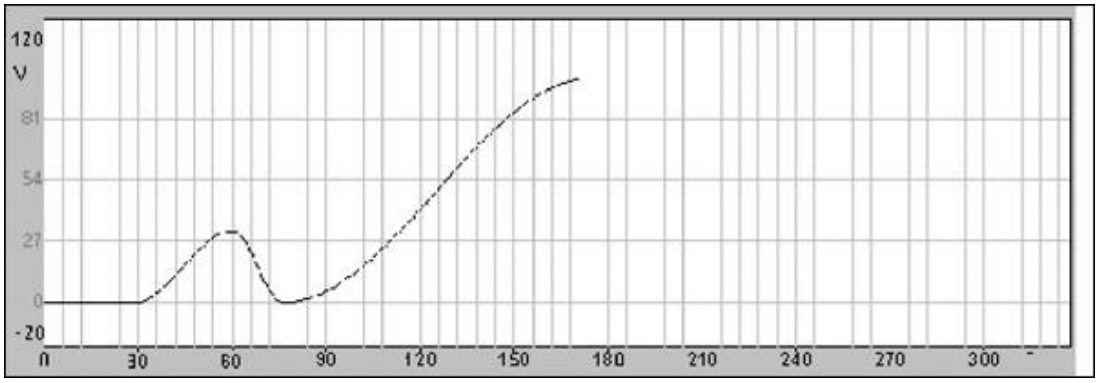
这个枚举变量是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。它的值确定了速度分布图。可能的状态：

枚举成员	关联量	描述
TRAPEZOID	0	速度分布为梯形。
SIGMOID	1	在恒定跃度下，速度分布是 S 形。

由 SMC_Interpolator 计算的梯形速度速度分布（byVel_Mode = 0）的例子：



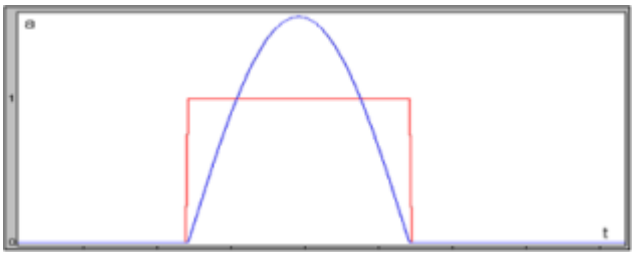
由 SMC_Interpolator 计算的 S 形速度分布（byVel_Mode = 0）的例子：



在上述两个例子中加速度的最大值小于减加速度的最大值（Decel）。这导致了速度曲线在加速和减速时斜率的不同。

这种速度分布的优点在于，与其相关的加速度－与梯形对比－是连续的，因此减轻了机械的负担。但这个优点也被稍稍增加的计算时间抵销掉一部分。

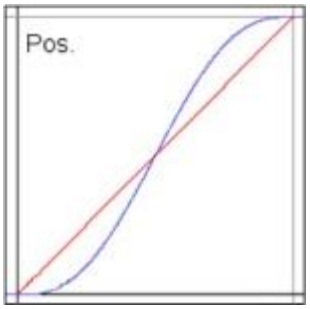
S 形的速度分布（蓝色）是这样设计的：改变梯形（红色）但是不要改变通过这个轨迹所需的时间，在开始和结束时所受到的加速度的限制， 必然要靠中途更高的加速度来补偿。这样， 在 SMC_GEOINFO 对象中存储的加速度和减加速度的最大值可能会被超过最多 π/2 倍。



就像对 dOverride 所做的描述一样，不能进行在线改值，直到当前正在运行的加速或减速被终止。

为了能够 S 形地（如下图中蓝色所画）而不是线性地（红色）插补附加轴，相应的位必须在当前对象的 piStartPos 的 wSProfile 变量中被设置。这使得附加轴不会在 X-，Y-，Z-空间里关于轨迹长度被线性的插补，而是在一个多边形的空间中关于轨迹长度被线性插补。

因此，在一个轨迹段的开始和结束处相对于轴位置会得到一个 0 速度和 0 加速度的 S 形的分布。



SMC_MOVTYP

这个枚举变量是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。它是 SMC_GEOINFO 的一个成员，它指明了对象的几何形状。可能的状态：

枚举值	关联量	描述
LIN	1	直接移动（G01）。
CLW		顺时针方向的圆（G02）。
CCLW		逆时针方向的圆（G03）。
SPLINE	5	样条拟和曲线（G05, G06）。
SPLINECORR	7	<-删除此线，已过时。
ELLCLW	8	顺时针椭圆(G08).
ELLCCLW	9	逆时针椭圆(G09).
LINPOS	100	直接定位(G00).
LINNODELTA	101	直接移动，但不经过 X/Y，而是经过 Z。
LINADDAXES	102	直接移动，但不经过 X/Y/Z。
INITPOS	120	盲定位（开始点未知，将由 SMC_Interpolator 添加）。
MCOMMAND	120	附加选项，M 选项。

SMC_TC_STATUS

它的值代表 FB 实例的实际状态，可能的状态值如下：

枚举值	关联量	描述
TC_ORIG	0	对象没有半径修正工具
TC_RIGHT	1	限制右移
TC_LEFT	2	限制左移
TC_RAMPIN_RIGHT	3	下一个对象向右倾斜
TC_RAMPIN_LEFT	4	下一个对象向左倾斜

SMC_TOOLCORRMODE

它的值代表 SMC_Toolcorr 运行时的模式。

枚举值	关联量	描述
TC_BLIND_POS	0	修正命令 (G41/G42)启动后，插入一段长度运动到插入点。
TC_RAMP_IN	1	修改命令 (G41/G42)启动后，下一个对象将爬坡到达设定的补偿量曲线。

SMC_VECTORDIR

它作为 SMC_CalcDirectionFromVector 的输入变量，用来定义速度的倾角、反向或垂直。

枚举值	关联量	描述
SMC_tangential	0	无补偿，切线方向。
SMC_orthogonal_r	-90	补偿-90; 顺时针转90º 方向。
SMC_orthogonal_l	90	补偿90; 逆时针转90º 方向。
SMC_opp_tangential	180	补偿180；相反的方向。

6.3 SMC_CNC_POUs

6.3.1 File

SMC_READNCFILE

这个功能块是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。它从控制器的文件系统中读取一个 NC-ASCII 文档，使得该文档对 SMC_NCDecoder 可用。因此在运行时一个 NC 程序可被读入并被执行。

组件	类型	初始值	描述
VAR_IN			
bExecute	BOOL	FALSE	当为 TRUE 时，功能块被激活。
sFileName	STRING(80)	''	文件路径。
pvl	POINTER TO SMC_VARLIST		指向一个 SMC_VARLIST 对象。如果 CNC 程序中没有变量，该输入不设置。
pBuffer	POINTER TO BYTE		指向一片足够大的，在 IEC 应用程序中分配的，自由数据区的指针。强烈建议为缓冲区 ARRAY[0..x] OF SMC_GCODE_WORD 分配空间，以保证有正确的数据对齐。在一些平台上，ARRAY[0..x]的定义有可能导致非对齐数据的访问。
dwBufferSize	DWORD	0	以字节表示的缓冲区的大小。
fDefaultVel	LREAL	0	默认速度；当速度在 CNC 文件中未指定时使用。
fDefaultAcc	LREAL	0	默认加速度；当加速度在 CNC 文件中未指定时使用。
fDefaultDecel	LREAL	0	默认减加速度；当减加速度在 CNC 文件中未指定时使用。
VAR_OUT			
bDone	BOOL	FALSE	如果参数有效则为 TRUE。
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
bError	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示。
bExecuteDecoder	BOOL	FALSE	用来触发SMC_NCDecoder 模块输入执行的信号。
ncprog	SMC_CNC_REF	STRUCT	CNC 程序，后面的 SMC_NCDecoder 模块

		(nElements:=0)	的输入。

SMC_READNCQUEUE

这个功能块是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。它从 PLC 文件系统中读取一个由 CNC 编辑器创建的 OutQueue 文件，并提供一个 OutQueue 结构，典型地该结构是由解码器处理的。

组件	类型	初始值	描述
VAR_IN			
bExecute	BOOL	FALSE	当为 TRUE 时，功能块被激活。
sFileName	STRING(80)	''	文件路径。
pBuffer	POINTER TO BYTE		指向一片足够大的，在 IEC 应用程序中分配的，自由数据区的指针。
dwBufferSize	DWORD	0	以字节表示的缓冲区的大小。
VAR_OUT			
bDone	BOOL	FALSE	如果参数有效则为 TRUE。
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
bError	BOOL	FALSE	在功能块内部发生错误的信号。
ErrorID	SMC_ERROR	0	错误指示。
poqDataOut	POINTER TO SMC_OUTQUEUE		指向已读入的队列的指针。
iObjectsRead	UDINT	0	到目前为止已被读入并被写入队列的 SMC_GEOINFO 对象编号。

SMC_VARLIST

IEC 61131-3 标准不可能通过字符串的符号名来决定一个变量的值。但这是必要的，假如变量的功能由 SMC_CNC_REF 提供给用户使用，也应该可以从一个文件中读取 CNC 程序。这个可以被结构 SMC_VARLIST 所管理。

SMC_VARLIST

组件	类型	初始值	描述
wVarListID	WORD		内部使用。
wNumberVars	WORD	0	包括使用的所有变量的数字。
psvVarList	Pointer to SMC_SingleVar	0	指向SMC_SingleVar 类型数组的第一个元素。

SMC_SingleVar

组件	类型	初始值	描述
stVarName	STRING	''	包含使用在 CNC 程序中的变量名的大写字母
pAdr	Pointer to BYTE	0	指向名为 stVarName 的变量。
eVarType	SMC_VARTYPE	SMC_TYPE_UNKNOW	声明变量类型
diValue	DINT	0	代表整型值
fValue	LREAL	0	代表浮点型值

例子：

在文件中通过使用 SMC_ReadNCFile 读取 CNC 程序，这里有两个变量 g_rTestX (REAL) 和 g_byCommand (BYTE):

N0

G$g_byCommand$ Xg_rTestX因

此你必须定义以下变量：

g_byCommand: BYTE;

g_rTest: REAL;

asv: ARRAY[0..1]OF SMC_SingleVar := [

(strVarName := 'G_BYCOMMAND', eVarType := SMC_TYPE_BYTE, p_Adr := ADR(g_byCommand)),

(strVarName := 'G_RTESTX', eVarType := SMC_TYPE_REAL, pAdr := ADR(g_rTest)))];

vl: SMC_VarList := (wNumberVars := 2, psvVarList := ADR(asv[0]));

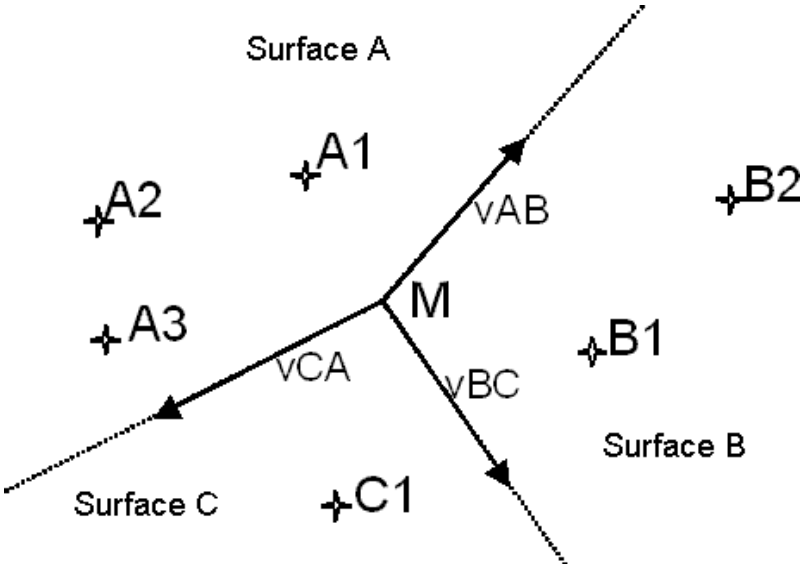
如果一个变量不能被替代，将会出现一个错误，并且模块将会中止。

6.3.2 SoftMotion CNC

Coordinate Transformation 文件夹内

SMC_DETERMINECUBOIDBEARING

这个功能块是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。它用给出的 6（3/2/1）个点确定了一个立方体（角标志，边对齐）在空间中的位置。



注意：关于坐标变换的详细信息见 survey on Coordinate Transformations

组件	类型	初始值	描述
VAR_IN			
A1	SMC_VECTOR3D		定位立方体面 A 的三个点中的第一个，并且该点不在面相交线上。

A2	SMC_VECTOR3D		定位立方体面 A 的三个点中的第二个，并且该点不在面相交线上。

A3	SMC_VECTOR3D		定位立方体面 A 的三个点中的第三个，并且该点不在面相交线上。
B1	SMC_VECTOR3D		定位立方体另一个与 A 相邻的面 B 的两个点中的第一个。该点在面 A 上的投影不能与 B2的投影重合。
B2	SMC_VECTOR3D		定位立方体另一个与 A 相邻的面 B 的两个点中的第二个。该点在面 A 上的投影不能与 B1的投影重合。
C1	SMC_VECTOR3D		与面 A 和 B 相邻的另一个面上的点。
VAR_OUT			
M	SMC_VECTOR3D		面 A，B 和 C 的共同交点。
vAB	SMC_VECTOR3D		由 M 点开始的沿 A 和 B 的交线的单位向量。
vBC	SMC_VECTOR3D		由 M 点开始的沿 B 和 C 的交线的单位向量。
vCA	SMC_VECTOR3D		由 M 点开始的沿 A 和 C 的交线的单位向量。
bError	BOOL		在函数块内部发生错误的信号
nError	WORD	0	错误描述： 0：无错误（bError=FALSE） 1：A1，A2，A3共线 2：B1和 B2在 A 上的投影重合

SMC_COORDINATETRANSFORMATIONS3D

这个功能块是 “CoDeSys SoftMotion” 的库 “SM3_CNC.library” 中的一部分。它计算在旧坐标系中给出、并转换到新坐标系下的点的坐标。坐标变换是由原点的变换和新单位向量（按旧坐标系给出）来定义的。

注意：变换模块的详细信息见 survey on Coordinate Transformations。

组件	类型	初始值	描述
VAR_IN			
biIn	BOOL	SMC_PosInfo	用旧坐标表示的点
vX	STRING(80)	SMC_Vector3D	新坐标系的单位向量
vY	POINTER TO BYTE	SMC_Vector3D	新坐标系的单位向量
vZ		SMC_Vector3D	新坐标系的单位向量
vTranslation	DWORD	SMC_Vector3D	原点的变换向量＝在旧坐标系中的新坐标系的原点
VAR_OUT			
piOut	BOOL	SMC_PosInfo	用新坐标表示的点

SMC_INVCOORDINATETRANSFORMATION3D

这个功能块是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。它计算一个在新坐标系下给出的点的坐标并将其换算到旧坐标系中。为此，逆坐标变换由原点平移和新单位向量（在旧坐标系中给出）定义。

注意：变换模块的详细信息见 survey on Coordinate Transformations。

组件	类型	初始值	描述

VAR_IN			
biIn	BOOL	SMC_PosInfo	新坐标系下的点。
vX	STRING(80)	SMC_Vector3D	新坐标系的单位向量
vY	POINTER TO BYTE	SMC_Vector3D	新坐标系的单位向量
vZ		SMC_Vector3D	新坐标系的单位向量
vTranslation	DWORD	SMC_Vector3D	原点平移向量＝旧坐标系中的新坐标系的原点。
VAR_OUT			
piOut	BOOL	SMC_PosInfo	旧坐标系下的点。

SMC_TEACHCOORDINATESYSTEM

它将帮助用户学习计算新坐标系中的向量。

输入新坐标系的原点和 x-,y-轴上的任意点作为输入数据。

该模块的输出是坐标转换和新坐标系的单位向量。另外，估算输入数据的性质（x,y 轴之间的夹角是否为 90 度）。

备注：有关变换模块的详细信息，请参考坐标转换。

组件	类型	初始值	描述
VAR_IN			
vOrigin	SMC_Vector3D		新坐标系的原点
vPointOnX	SMC_Vector3D		X-轴上点的位置
vPointOnY	SMC_Vector3D		Y-轴上点的位置
VAR_OUT			
vX	SMC_Vector3D		新坐标系下单位向量的 x-坐标
vY	SMC_Vector3D		新坐标系下单位向量的 y-坐标
vZ	SMC_Vector3D		新坐标系下单位向量的 z-坐标
vTranslation	SMC_Vector3D		转换向量原点=新坐标系在老坐标中的原点
fQuality	LREAL		输入数据的指令在0（(vX, vY 无效）与（(vX,vY 正交）之间。

SMC_UNITVECTORTORPY

从新坐标系的单位向量计算 RPY-角度（参考旧坐标系）。

备注：有关变换模块的详细信息，请参考坐标转换。

组件	类型	初始值	描述
VAR_IN			
vX	STRING(80)	SMC_Vector3D	新坐标系下单位向量的 x-坐标（参考旧坐标系）
vY	Pointer TO BYTE	SMC_Vector3D	新坐标系下单位向量的 y-坐标（参考旧坐标系）
vZ		SMC_Vector3D	新坐标系下单位向量的 z-坐标（参考旧坐标系）
VAR_OUT			
dA		LREAL	弧度的卷角
dB		LREAL	弧度的仰角

dC		LREAL	弧度的偏航角
bError		BOOL	功能未执行完时为 TRUE
nError	BOOL	WORD	错误类型描述： 1：无错误(bError = FALSE) 2：向量长度不够 1 3：向量没有正交 4：无右手系统

SMC_ROTATEQUEUE2D

这个功能块是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。在该模块执行时，存储在 poqDataIn 中的轨迹将被绕 Z 轴旋转给出的角度 dPhi[°]。正角度是指数学上的正向旋转（逆时针）。

输入变量 poqDataIn 是一个指向 SMC_OUTQUEUE 结构的指针，该结构描述了要被旋转的轨迹。输入变量 bEnable 的初始值 FALSE 将禁止轨迹的旋转。一旦 bEnable 被设为 TRUE，所有在 poqDataIn 中找到的 GEOINFO 对象将被处理。当 bEnable 被重置为 FALSE 时，模块将不执行任何其他修改。

用 FALSE 初始化的输入变量 bReset 将使在 poqDataIn 中实际找到的 GEOINFO 对象不被旋转，但是要被添加的对象仍然旋转。

在处理期间，旋转的特征参数 dPhi 的突然改变可能会引起轨迹中断，相应输入的修改将不被考虑，直到 SMC_OUTQUEUE 的 poqDataIn 为空，或者直到进行重置（bReset）。

为了在（X，Y）平面内绕不是（0，0）的点进行旋转，比如绕点（xP，yP），必须先用向量（-xP,-yP,0）执行一次平移，然后再以想要的角度 dPhi 进行旋转，接着再用向量（xP,yP,0）做一次逆平移。

组件	类型	初始值	描述
VAR_IN			
bExecute	BOOL	FALSE	当为 TRUE 时，功能块被激活。
bAppend	BOOL	FALSE	如果在 bExecute 上升沿是 FALSE，则输出队列 poqDataOut 已清空。
poqDataIn	POINTER TO SMC_OUTQUEUE	0	指向一片足够大的，在 IEC 应用程序中分配的，自由数据区的指针。
dPhi	LREAL	0	旋转角度。
nSizeOutQueue	UDINT	0	该变量包含 GEOINFO 结构对象表将被写入的数据缓冲区的大小。该缓冲区必须至少5倍大于 GEOINFO 本身。否则，SMC_NCDecoder 将不执行任何动作。它的大小可预定义，但是只能通过重置修改。建议按下述方法创建一个缓冲： ExampleBuf: Array of SMC_GeoInfo. 划出的缓冲区大小可由操作 sizeof(ExampleBuf)得到。
dbyBufferOutQueue	POINTER TO BYTE	0	该输入必须指向分配给 OUTQUEUE 结构的存储区的第一个字节。该区域必须至少和在 nSizeOutQueue 中定义的一样大。典型地，

			存储区分配是通过在 IEC 程序地声明部分定义一个数组 SMC_GEOINFO（例如，BUF: ARRAY[1..50] OF SMC_GEOINFO；一个缓冲可存储50个路径元素）来进行的。该值可被预定义，但是只能通过重置来改变。
VAR_OUT			
bDone	BOOL	FALSE	一旦 poqDataIn 的输入数据被完全处理该变量将被设置为 TRUE。之后该模块将不再执行任何动作直到重置发生。如果输入 bExecute 为 FALSE，bDone 将被设为 FALSE。
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
bError	BOOL	FALSE	在功能块内部发生错误的信号。
wErrorID	SMC_ERROR	SMC_NO_ERROR	错误指示。
poqDataOut	POINTER TO SMC_OUTQUEUE	0	指向得到的 SMC_OutQueue 的指针。

SMC_SCALEQUEUE3D

这个功能块是“CoDeSys SoftMotion” 的库“SM3_CNC.library” 中的一部分。在它执行时，在 poqDataIn 所包含的轨迹将被因数 fScaleFactor 拉伸。

输入变量 poqDataIn 是一个指向 SMC_OUTQUEUE 结构的指针，它描述了要被缩放的轨迹。一开始，输入变量 bEnable 的初始值 FALSE 将禁止轨迹缩放。一旦 bEnable 被设为 TRUE，所有在 poqDataIn 找到的 GEOINFO 对象将被处理。当 bEnable 被重置为 FALSE 时，模块不执行其他任何修改。

输入变量 bReset 的初始值 FALSE 的效果是，在 poqDataIn 中实际找到的 GEOINFO 对象将不被旋转，而只有那些将被加入的才会被旋转。

组件	类型	初始值	描述
VAR_IN			
bExecute	BOOL	FALSE	当为 TRUE 时，功能块被激活。
bAppend	BOOL	FALSE	如果在 bExecute 的上升沿为 FALSE，则输出队列已被清除。
poqDataIn	POINTER TO SMC_OUTQUEUE	0	指向一片足够大的，在 IEC 应用程序中分配的，自由数据区（缓冲）的指针。
dScaleFactor	LREAL	1	引用于轨迹上的缩放比例因数。
nSizeOutQueue	UDINT	0	该变量包含 GEOINFO 结构对象表将被写入的数据缓冲区的大小。该缓冲区必须至少5倍大于 GEOINFO 本身。否则，SMC_NCDecoder 将不执行任何动作。它的大小可预定义，但是只能通过重置修改。建议按下述方法创建一个缓冲： ExampleBuf: Array of SMC_GeoInfo. 正确的缓冲区大小可由操作

			sizeof(ExampleBuf)得到。
dbyBufferOutQueue	POINTER TO BYTE	0	该输入必须指向分配给 OUTQUEUE 结构的存储区的第一个字节。该区域必须至少和在 nSizeOutQueue 中定义的一样大。典型地，存储区分配是通过在 IEC 程序地声明部分定义一个数组 SMC_GEOINFO（例如，BUF: ARRAY[1..50] OF SMC_GEOINFO；一个缓冲可存储50个路径元素）来进行的。该值可被预定义，但是只能通过重置来改变。
VAR_OUT			
bDone	BOOL	FALSE	一旦 poqDataIn 的输入数据被完全处理该变量将被设置为 TRUE。之后该模块将不再执行任何动作直到重置发生。如果输入 bExecute 为 FALSE，bDone 将被重设为 FALSE。
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
bError	BOOL	FALSE	在功能块内部发生错误的信号。
wErrorID	SMC_ERROR	SMC_NO_ERROR	错误指示。
poqDataOut	POINTER TO SMC_OUTQUEUE	0	指向得到的 SMC_OutQueue 指针。

SMC_TRANSLATEQUEUE3D

通过它的执行存储在 poqDataIn 中的路径将会按照 vec 转换，vec 是 SMC_VECTOR3D 的数据类型。

输入变量 poqDataIn 是结构 SMC OUTQUEUE 的指针，描述旋转的路径。

输入变量 bEnable 的初始化值 FALSE 将阻止路径转换。只要 bEnable 被设置为 TRUE，所有在 poqDataIn 中搜索到的 GEOINFO-objects 将会被处理。当 bEnable 被重置为 FALSE，模块不会执行任何进一步的修改。

输入变量 bReset 为 FALSE 时，造成在 poqDataIn 中找到的 GEOINFO 对象不能旋转，而只是被添加。

在处理过程中突然修改旋转/平移属性参数可能会导致轨迹停顿，任何相关输入的改动只会在 SMC OUTQUEUE 为空或复位（bReset）才会起作用。

组件	类型	初始值	描述
VAR_IN			
bExecute	BOOL	FALSE	为 TRUE 时，功能块被激活。
bAppend	BOOL	FALSE	如果在 bExecute 的上升沿为 FALSE，输出列将被清空。
poqDataIn	POINTER TO SMC_OUTQUEUE	0	指向 SMC OUTQUEUE 的指针，它描述被旋转或平移的轨迹。
vec	SMC_VECTOR3D	1	

nSizeOutQueue	UDINT	0	此变量包含的数据缓冲区的大小，GEOINFO 的结构对象清单也将被写入。这个缓冲区必须至少是 GEOINFO 结构本身的5倍。否则 SMC NCDecoder 不会执行任何指令。它的大小可能是预先确定的，但可能只是在复位后修改。它建议建立由下面的例子描述的缓冲区。 Example buf： SMC GEOINFO 阵列。 适当的缓冲区大小将会被操作员 SIZEOF（ExampleBuf）的使用检索。
dbyBufferOutQueue	POINTER TO BYTE	0	缓冲区大小，以 byte 为单位。
VAR_OUT			
bDone	BOOL	FALSE	当 poqDataIn 的输入数据被处理完成时，这个变量就会被设置为 TRUE。此后，该模块将不会执行任何进一步的行动，知道它被重置。如果输入 bExecute 为 FALSE，bDone 将被重置为 FALSE。
bBusy	BOOL	FALSE	当功能块的执行没有完成时为 TRUE。
bError	BOOL	FALSE	当功能块没有执行完成时为 TRUE。
wErrorID	SMC_ERROR	SMC_NO_ERROR	错误编号。
poqDataOut	POINTER TO SMC_OUTQUEUE	0	结构 SMC_OutQueue 的指针。

Direct Axis Control 文件夹内

SMC_CONTROLAXISBYPOS

这个功能块是库“SM3_CNC.library”中的一部分。它能够被用来写驱动结构的目标位置以及测试跳转的结构。它通常用于 CNC 和 SMC_Interpolator。对于轴的设置值的直接控制，随 SMC_Follow* 启动的功能块中的一个应被使用。

组件	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		轴参考，见 AXIS_REF_SM3
VAR_INPUT			
iStatus	SMC_INT_STATUS		SMC_Interpolator 的状态输出
bEnable	BOOL	FALSE	允许功能块执行时必须置为 TRUE
bAvoidGaps	BOOL		如果为 TRUE，模块将监视位置和速度。如果速度超过限制，该限制存储在轴中（存储在 AXIS_REF_SM3. fSWMaxVelocity 中并且在驱动对话框的“Maximum values“中配置），模块将设置输出 bStopIpo 并按照参数 fGapVelocity, fGapAcceleration 和 fGapDeceleration 移动轴到

			这个位置。之后它将删除输出 bStopIpo。
fSetPosition	LREAL		设置轴位置。典型地这是个变换模块的输出。
fGapVelocity	LREAL	1	bTimePut 将被设置成 TRUE，如果自功能块激活后过去的时间超过了fTimeOut 并且bInPosition 还没有被设置。这种机制在 dTimeOut 被设为0时失效。
fGapAcceleration	LREAL	1	桥接突变的运动参数
fGapDeceleration	LREAL	1	桥接突变的运动参数
VAR_OUT			
bBusy	BOOL	FALSE	当功能块的执行没结束时为 TRUE。
bCommandAborted	BOOL		如果命令已被另一个命令终止时为 TRUE。
bError	BOOL		功能块内部发生错误的信号。
iErrorID	SMC_ERROR		错误标识，见 SMC_Error。
bStopIpo	BOOL	FALSE	模块已探测到一次速度或位置突变并且刚刚调整到一个新位置时为 TRUE。由于这个原因，这个输出应该与 SMC_Interpolator 的 EmergencyStop 输入连起来，使得插补器等待直到轴被正确定位。

SMC_CONTROLAXISBYPOSVEL

这个功能块是库“SM3_CNC.library”中的一部分。除了在添加时速度可以被定义以外，它的使用等同于 SMC_ControlAxisByPos。它通常用于 CNC 和 SMC_Interpolator。对于轴的设置值的直接控制，随 SMC_Follow*启动的函数块中的一个应被使用。

组件	类型	初始值	描述
VAR_IN_OUT			
Axis	AXIS_REF		轴参考，见 AXIS_REF_SM3
VAR_INPUT			
iStatus	SMC_INT_STATUS		SMC_Interpolator 的状态输出
bEnable	BOOL	FALSE	允许功能块执行时必须置为 TRUE
bAvoidGaps	BOOL		如果为 TRUE，模块将监视位置和速度。如果速度超过限制，该限制存储在轴中（存储在 AXIS_REF_SM3. fSWMaxVelocity 中并且在驱动对话框的“Maximum values“中配置），模块将设置输出 bStopIpo 并按照参数 fGapVelocity, fGapAcceleration 和 fGapDeceleration 移动轴到这个位置。之后它将删除输出 bStopIpo。
fSetPosition	LREAL		设置轴位置。典型地这是个变换模块的输出。
fSetVelocity	LREAL		将速度设置为技术单位/秒
fGapVelocity	LREAL		bTimePut 将被设置成 TRUE，如果自函数块激活后过去的时间超过了 fTimeOut 并且 bInPosition 还没有被设置。这种机制在 dTimeOut 被设为0时失效。
fGapAcceleration	LREAL		桥接突变的运动参数
fGapDeceleration	LREAL		桥接突变的运动参数

VAR_OUT			
bBusy	BOOL	FALSE	当功能块的执行没结束时为 TRUE。
bCommandAborted	BOOL		如果命令已被另一个命令终止时为 TRUE。
bError	BOOL		功能块内部发生错误的信号。
iErrorID	SMC_ERROR		错误标识，见 SMC_Error。
bStopIpo	BOOL	FALSE	模块已探测到一次速度或位置突变并且刚刚调整到一个新位置时为 TRUE。由于这个原因，这个输出应该与SMC_Interpolator 的 EmergencyStop 输入连起来，使得插补器等待直到轴被正确定位。

GCode Viewer 文件夹内

SMC_GCODEVIEWER

这个功能块是 “CoDeSys SoftMotion” 的库 “SM3_CNC.library” 中的一部分。它可以与 SMC_NCDecoder 结合使用。它以 ARRAY OF STRING 类型收集和存储某个 GeoInfo 对象和转存的文本表达（G 代码），这些信息可经由－比如说表－在视图中显示。经由一个记录，一般是与插补器输出 iActObjectSourceNo 相连接的，模块得到那一行已经被处理过的信息，并将这行删除。所以一个对于实际 G 代码的可视化显示元素可被创建。

注意，功能块的输出并不完全符合存储在文本文件或编辑器中的 G 代码，例如假设没有空行和注释，有可能使用的变量已经被替代。

该功能块在 SMC_NCDecoder 模块的任务运行环境中被调用：

必须要有足够大的缓冲区。 SMC_GCODEVIEWER_DATA 对象的个数必须至少和位于 SMC_NCDecoder 模块以及轨迹预处理模块缓冲区中的 SMC_GEOINFO 对象的总数一样大。

组件	类型	初始值	描述
VAR_IN_OUT			
GCodeText	SMC_GCODE_TEXT		数据源与 SMC_NCDecoder 模块的同名输出相连。
VAR_IN			
bEnable	BOOL		在入口处的上升沿将启动函数块的执行。
iActObjectSourceNo	INT		当前被插补器处理的行的编号。
uiNumberOfBufferEntries	UINT		由 pBuffer 传递的数组大小。
pBuffer	POINTER TO SMC_GCODEVIEWER_DATA		为功能块分配的，做为缓冲区的数组地址。
VAR_OUT			
bError	BOOL		功能块内部发生错误的信号。
bBusy	BOOL		如果功能块的执行还没结束为 TRUE。
wErrorID	SMC_ERROR		错误标识。
asGCode	ARRAY[0..c_uiLines] OF STRING		当前活动的 G 代码行。

Posinfo Functions 文件夹内

SMC_POSINFO

这个数据类型是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。对一个特殊位置点该结构描述了它的坐标和附加轴的位置。

组件	类型	初始值	描述
iFrameNo	INT	0	与 SoftMotion 模块不相关的附加信息可被用户存在该变量中。
wAuxData	WORD	7	由 SMC_Interpolator 计算的位置轴的逐比特描述。初始值 2#111将使 x-，y-和 z-轴被插补。第一位组将影响 dX 位置的计算，比如第7位将影响 dA2的处理。
wSProfile	WORD	0	与 wAuxData 类似地，wSProfile 的值规定了附加轴，比如 X-或者 Y-轴，应该以恒定跃升被线性（FALSE）插补或者被 S 形插补（TRUE）。第2位对应 Z 轴，第6位对应 P，第7位对应 Q，第8位对应 U，第9位对应 V，第10位对应 W。
dX	LREAL	0	坐标系中的 x 位置。
dY	LREAL	0	坐标系中的 y 位置。
dZ	LREAL	0	坐标系中的 z 位置。
dA	LREAL	0	目前没用。
dB	LREAL	0	目前没用。
dC	LREAL	0	目前没用。
dA1	LREAL	0	附加轴位置 A1.
dA2	LREAL	0	附加轴位置 A2.
dA3	LREAL	0	附加轴位置 A3.
dA4	LREAL	0	附加轴位置 A4.
dA5	LREAL	0	附加轴位置 A5.
dA6	LREAL	0	附加轴位置 A6.

OutQueue Functions 文件夹内

SMC_APPENDOBJ

这个功能块是“CoDeSys SoftMotion” 的库“SM3_CNC.library” 中的一部分。该函数块用来处理 SMC_OUTQUEUE 结构体对象。在使用它时，由输入 pgi 传递的 GEOINFO 对象将被添加到列表尾部（poq），前提是列表被重新正确的初始化并且还未被填满。如果成功，该功能块将返回一个指向列表中最新元素的指针，否则返回 0。

组件	类型	初始值	描述
VAR_IN			
poq	指向SMC_OUTQUEUE 的指针		列表对象应被增加
pgi	指向 SMC_GEOINFO 的指针		应被添加的对象
VAR_OUT			
SMC_AppendObj	指向 SMC_GEOINFO 的指针		指向列表中新被添加的元素的指针

SMC_DELETEOBJ

这个功能块是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。该布尔型函数用来处理

一个 SMC_OUTQUEUE 结构对象。在使用时，第 N 个对象被从表（poq）删除，计数从 0 开始。如果 N-1 大于存储在表中的 GEOINFO 对象个数，则什么都不做并返回 FALSE 而不是 TRUE。

组件	类型	初始值	描述
VAR_IN			
poq	POINTER TO SMC_OUTQUEUE		对象将从中被删除的列表。
N	UINT		被删除的对象在表中的序号（注意，N=0表示表中的第一个元素）。
VAR_OUT			
SMC_DeleteObj	BOOL SMC_DELETEOBJ(POQ: POINTER TO SMC_OUTQUEUE, N: UINT)		指向表中被删除对象的指针。

SMC_GETCOUNT

这个功能块是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。类型为 UNIT 的函数将返回存储在 SMC_OUTQUEUE 表（POQ）中的对象个数。

组件	类型	初始值	描述
VAR_IN			
Poq	POINTER TO SMC_OUTQUEUE		输出表中的元素应被计数
VAR_OUT			
SMC_GetCount	UINT		表中元素个数

SMC_GETOBJ

这个功能块是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。假如输入 poq 被正确初始化并含有至少 N+1 个元素，函数将返回一个指向表中第 N 个 GEOINFO 对象的指针，计数从表的开头算起；所以对于 N=0 第一个元素被返回。

组件	类型	初始值	描述
VAR_IN			
poq	POINTER TO SMC_OUTQUEUE		对象将被从中选择的表。
N	UNIT		当计数从表中第一个元素即 N=0开始时，元素表中对象的位置。
VAR_OUT			
SMC_GetObj	POINTER TO SMC_GEOINFO		指向与编号 N 相关联的表对象指针。

SMC_GETOBJFROMEND

这个功能块是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。假如输入 poq 被正确初始化并含有至少 N+1 个元素，函数将返回一个指向表中第 N 个 GEOINFO 对象的指针，计数从表的结尾算起；所以对于 N=0 最后一个元素被返回。

组件	类型	初始值	描述

VAR_IN			
poq	POINTER TO SMC_OUTQUEUE		对象将被从中选择的表。
N	UNIT		当计数从表中最后一个元素即 N=0开始时，元素表中对象的位置。
VAR_OUT			
SMC_GetObjFromEnd	POINTER TO SMC_GEOINFO		指向与编号 N 相关联的表对象指针。

SMC_OUTQUEUEINIT

这个功能块是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。它设置 SMC_OUTQUEUE 数据结构的变量为它的默认值。

组件	类型	初始值	描述
VAR_IN			
poq	POINTER TO SMC_OUTQUEUE		要被重置的 Outqueue。
VAR_OUT			
SMC_OutQueueInit	BOOL		虚拟输出

SMC_RESTOREQUEUE

这个功能块是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。该函数块将恢复一个已被插补的或已被处理了的结构。仅当整个轨迹能存储到由 poq 引用的表中时，这才是可能的。

组件	类型	初始值	描述
VAR_IN			
Enable	BOOL	FALSE	必须设置为 TRUE 以使处理能够进行。
poq	POINTER TO SMC_OUTQUEUE		要恢复的 Outqueue。
VAR_OUT			
SMC_RestoreQueue	BOOL		如果操作已成功结束则为 TRUE。

SMC_SETQUEUECAPACITY

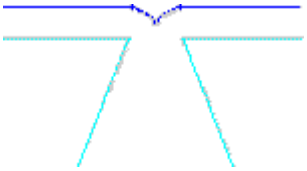
这个功能块是 “CoDeSys SoftMotion” 的库 “SM3_CNC.library” 中的一部分。用于设置 SMC_OUTQUEUE 数据结构的变量空间。

组件	类型	初始值	描述
VAR_IN			
poq	POINTER TO SMC_OUTQUEUE		要被重置的 Outqueue。
nSizeBytes	UDINT		队列缓冲区的大小（以 byte 为单位）

SoftMotion Function Blocks 文件夹内

SMC_AVOIDLOOP

这个功能块是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。该功能块被用来做轨迹预处理。它拷贝一个轨迹，但并不剪断其内部含有的环圈。如果在原始轨迹中一个交叉点被检测到，这个轨迹将在交叉点被剪断，环圈将被移除，轨迹由其他曲线连接。因此将得到一条连续的、没有环圈的轨迹。



典型应用请看 SMC_ToolCorr 模块。

功能块 SMC_AvoidLoop 的设计如下所述：

该模块传递所有在输入 SMC_OUTQUEUE 结构体中找到的 SMC_GEOINFO 对象。如果其中一个对象的 Intern_Mark 变量的第 7 位被设置了，则防环圈功能被激活。这个函数能够检查：当前对象和后面的以及前一个 SMC_GEOINFO 对象是否有交点，这些对象的 Intern_Mark 变量的第 6 位是被设置的。第 7 位被置高将终止防环圈功能。如果没有找到交点，则对象将被不改变地复制到输出 SMC_OUTQUEUE。否则第一个交叉的对象将在交点被剪断，位于两个交叉对象之间的 SMC_GEOINFO 对象将被移除，新轨迹将从两个交叉对象中的第二个继续。SMC_NCDecoder 将设置 Intern_Mark 的第 6 和第 7 位，作为对使用说明 G61/G60 的反应。

注意：SMC_AvoidLoop 的效率依赖于适当的输入 SMC_OUTQUEUE 的大小。一个包含了多个对象、且其数量已超出能够在 SMC_OUTQUEUE 中保存的数量的环圈无法被探测到！

组件	类型	初始值	描述
VAR_IN			
bExecute	BOOL	FALSE	当值为 TRUE 时功能块激活
bAbort	BOOL	FALSE	如果为 TRUE，中止对此功能块的当前处理。
bAppend	BOOL	FALSE	只要此输入保持 FALSE，DataOutQueue 在每次重置时将被清除。只要其保持 TRUE，新进入的数据将被写到 DataOutQueue 的尾部。
poqDataIn	POINTER TO SMC_ OUTQUEUE	0	这个变量指向 SMC_OUTQUEUE 结构对象，它包含轨迹的 SMC_GEOINFO 对象；典型地它指向前一个模块地输出..DataOut（例如 SMC_NCDecoder）。它将被置为合适的大小（见上）。
nSizeOutQueue	UDINT	0	此变量包含了数据缓冲区的大小，GEOINFO 结构对象的列表将被写入该缓冲中。该缓冲必须至少是 GEOINFO 本身的 5 倍大小。否则，SMC_NCDecoder 将不执行任何动作。它的大小可预定义，但以后只能通过重置被改变。建议按照如下所述直接创建该缓冲： ExampleBuf: Array[1…50] of SMC_GeoInfo. 划出的缓冲大小可通过操作数 sizeof(ExampleBuf)重新得到。
pbyBufferOutQueue	POINTER TO	0	该输入必须指向分配给OUTQUEUE 结构的存储区的第一个字节。这片区域至少要和在

	ARRAY[0..0] OF SMC_GEOIN FO		nSizeOutQueue 中定义的大小一样。典型地存储缓冲区的分配是在 IEC 程序的声明部分进行的，由定义一个 SMC_GEOINFO 类型的数组来实现（例如：BUF: ARRAY[1..50] OF SMC_GEOINFO; 这里一个缓冲可存储50个轨迹元素）。该值可以被预定义，但以后只能通过重置被改变。
VAR_OUT			
bDone	BOOL	FALSE	一旦…DataIn 的输入数据被完全处理完，该变量就被设为 TRUE。之后模块不再执行任何动作直到被重置。如果输入 bExecute 是 FALSE，bDone 将被重置为 FALSE。
bBusy	BOOL	FALSE	当函数块还未完成动作时为 TRUE
bError	BOOL	FALSE	功能块执行时其内部发生错误的信号。
wErrorID	SMC_ERROR	SMC_NOERROR	错误识别。
poqDataOut	POINTER TO SMC_OUTQUEUE	0	该变量指向一个 SMC_OUTQUEUE 结构，该结构管理无环圈轨迹的 SMC_GEOINFO 对象
iStatus	SMC_AL_STATUS	0	该枚举变量表示模块的当前状态。可能状态： AL_OFF 0 防环圈功能关 AL_ON 1 防环圈功能开 AL_END 2 对象处理终止

SMC_CHECKFORLIMITS

用于检查轨迹是否离开其特定的矩形范围。

组件	类型	初始值	描述
VAR_IN			
bExecute	BOOL	FALSE	为 TRUE 时，功能块被激活。
bAbort	BOOL	FALSE	为 TRUE 时，此功能块的当前处理将会被中止。
poqDataIn	POINTER TO SMC_OUTQUEUE	0	这个变量指向 SMC_OUTQUEUE 结构对象，它包含轨迹的 SMC_GEOINFO 对象；典型地它指向前一个模块地输出 poqDataOut（例如 SMC_NCDecoder/SMC_SmoothPath ）。SMC_OUTQUEUE
wAddAxis	WORD	0	标识需要被测试的轴： X：Bit0 Y：Bit1 Z：Bit2 P：Bit6 Q：Bit7 U：Bit8 V：Bit9

			W：Bit10
piMin	SMC_POSINFO		范围的下限
piMax	SMC_POSINFO		范围的上限
VAR_OUT			
bBusy	BOOL	FALSE	TRUE，当功能块的执行未完成时，路径对象依然继续运行。
bError	BOOL	FALSE	功能块执行时其内部发生错误的信号。
wErrorID	SMC_ERROR	SMC_NOERROR	错误代码
wAxisOutOfLimits	WORD		表示哪个轴已经离开范围
iSourceLine_No	INT		打印超过限制的对象的行号
poqDataIn	POINTER TO SMC_OUTQUEUE	0	该输出指向 SMC_GEOINFO 结构对象，该对象包含带有允许速度值的轨迹，并且应提供给插补器。
wLimitSwPositive	WORD		表示轴超过上限
wLimitSmNegative	WORD		表示轴超过下限

SMC_CHECKVELOCITIES

这个功能块是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。该模块检查特殊轨迹段的轨道速度。如果 OutQueue 不是被编辑器创建， 而是由 IEC 程序创建（ 例如， 由 SMC_NCDecoder），该模块在每一次被插补器调用之前先被直接调用。

该函数的主要任务是检查有急剧弯曲的轨迹，并在这种弯曲的地方将速度减为 0。

组件	类型	初始值	描述
VAR_IN			
bExecute	BOOL	FALSE	当时 TRUE 时，功能块被激活。
bAbort	BOOL	FALSE	如果为 TRUE，中止此功能块的当前处理。
poqDataIn	POINTER TO SMC_OUTQUEUE	0	该输入指向 SMC_OUTQUEUE 结构对象，它描述了轨迹的 SMC_GEOINFO 对象；典型地它指向前一个模块（例如 SMC_NCDecoder / SMC_SmoothPath）的输出 poqDataOut。
dAngleTol	LREAL	0	该输出描述允许角度（漂角），这个角度是在轨迹急剧弯曲时不导致停止的最大角度。
VAR_OUT			
bBusy	BOOL	FALSE	当功能块的执行没结束和路径对象的处理仍在运行时为 TRUE
bError	BOOL	FALSE	功能块内部发生错误的信号。
wErrorID	SMC_ERROR	SMC_NOERROR	错误标识
poqDataOut	POINTER TO SMC_OUTQUEUE	0	该输出指向 SMC_OUTQUEUE 结构对象，该对象包含带有允许速度值的轨迹，并且应提供给插补器。

SMC_EXTENDEDVELOCITYCHECKS

这个功能块是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。它降低轨迹的速度、加速度和减加速度，以使得对于附加轴（Z，P，Q，U，V，W）这几个量不会超过在 SMC_GEOINFO 变量 adVelAddAx[1..5]，adAccAddAx[1..5] 和 adDecAddAx[1..5]中设置的值。典型地，这些极限值通过使用 G 代码字 FZ，FP，FQ，FU，FV，FW 和 EZ，EP，EQ，EU，EV，EW 在其中定义。

组件	类型	初始值	描述
VAR_IN			
bExecute	BOOL	FALSE	为 TRUE 时功能块被激活。
bAppend	BOOL	FALSE	如果在 bExecute 的上升沿时为 FALSE，则输出队列 poqDataOut 为空。
poqDataIn	POINTER TO SMC_ OUTQUEUE	0	该输入指向 SMC_OUTQUEUE 结构对象，该对象描述了轨迹的SMC_GEOINFO 对象；典型地，它指向前一个模块的输出 poqDataOut（例如，SMC_NCDecoder/ SMC_SmoothPath）。
nSizeOutQueue	UDINT	0	该变量包含数据缓冲的大小，GEOINFO 的结构对象表将写入其中。此缓冲区必须至少5倍于 GEOINFO 结构本身。否则，SMC_NCDecoder 将不执行任何动作。它的大小可被预定义，但是只能通过重置来改变。 建议按照如下方法创建一个缓冲： ExampleBuf: Array [1…50]of SMC_GeoInfo 划出的缓冲区大小可用如下操作得到： sizeof(ExampleBuf)。
pbyBufferOutQueue	POINTER TO ARRAY[0..0] OF SMC_ GEOINFO	0	该输入必须指向分配给 OUTQUEUE 结构的存储区的第一个字节。该区域必须至少和在 nSizeOutQueue 中定义的一样大。典型地，存储区分配是通过在 IEC 程序的声明部分定义一个数组 SMC_GEOINFO （例如， BUF: ARRAY[1..50] OF SMC_GEOINFO；一个缓冲可存储50个路径元素）来进行的。该值可被预定义，但是只能通过重置来改变。
VAR_OUT			
bDone	BOOL	FALSE	
bError	BOOL	FALSE	功能块内部发生错误的信号。
wErrorID	SMC_ERRO R	SMC_NO ERROR	错误指示。
poqDataOut	POINTER TO SMC_ OUTQUEUE	0	该变量指向一个 SMC_OUTQUEUE 结构，该结构管理无环圈轨迹的 SMC_GEOINFO 对象。

SMC_INTERPOLATOR

这个功能块是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。这个功能块用来把由 SMC_GEOINFO 对象描述的一个连续轨迹转化为离散轨迹位置点，这些点考虑到了速度分布图和时间图。

注意：在轨迹运行终止时 SMC_OUTQUEUE 是空的。如果您想再次处理相同的轮廓，您既可以通过解码器和轨迹预处理器将 CNC 程序转化为 SMC_OUTQUEUE 结构，也可以使用功能块 SMC_RESTOREQUEUE（也是 SM_CNC.lib 的一部分）。后者可能要求 OUTQUEUE 缓冲区足够大以装下整个轨迹。

组件	类型	初始值	描述
VAR_IN			
bExecute	BOOL	FALSE	当为 TRUE 是，函数块被激活。
poqDataIn	POINTER TO SMC_OUTQUEUE	0	该变量指向 SMC_OUTQUEUE 结构对象, 这个对象包含未平滑化轨迹的 SMC_GEOINFO 对象；典型地它指向前一个 SMC_CheckVelocities 模块的输出*..DataOut* 。
bSlow_Stop	BOOL	FALSE	如果该变量被设置为 FALSE，则不停顿地通过该轨迹。否则，SMC_Interpolator 将使速度按已定义的分布图(由 VelMode 确定的)减至0，当前 GEOINFO 对象（dDecel，见下）的延迟达到最大值，并且等待直到 bSlow_Stop 被重设为 FALSE。
bEmergency_Stop	BOOL	FALSE	一旦该输入得到 TRUE 值，SMC_Interpolator 将立即引发一个停止，就是说位置保持不变。因此，速度将被立即设置为0。
bWaitAtNextStop	BOOL	FALSE	在该变量为 FALSE 时（默认值），则不停顿地通过该轨迹。否则，SMC_Interpolator 将在下一次正常停止时（即速度为0的位置点，或者典型地，速度为 0 的轨迹角度） 保持位置不变， 并暂停直到 bWaitAtNextStop 被重置为 FALSE。
dOverride	LREAL	1	该变量可用来处理重写。dOverride 的值不能低于 0.01。特定对象的预定速度将被 dOverride 缩放；故预定速度在在线模式下可以被升高和降低。例如，dOverride=1（默认值）的效果是：程序预定的速度将被执行，dOverride=2则将该速度翻倍后执行。 请注意：重写值可在任何时候被修改，但是只有在当前过程中没有任何加速度和减加速度时修改才被应用！
iVelMode	SMC_INT_VELMODE	TRAPEZOID	该输入定义了速度分布图。值“TRAPEZOID”表示梯形，“SIGMOID”表示 S 形。见 SMC_INT_VELMODE 的进一步说明。
dwIpoTime	DWORD	0	该变量在每次调用时都必须被设置。它以微妙来表示周期时间。
dLastWayPos	LREAL	0	该输入允许用户测量被插补器扭曲了的轨迹的拉伸度。输出 dWayPos 是 dLastWayPos 和当前周期内已走过的距离之和。如果 dLastWayPos=0, dWayPos 就表示的是当前轨迹段的长度。如果 dLastWayPos 被设置成和输出 dWayPos 相等，则 dWayPos 总是以当前轨迹段为单位增长，以得到已走过轨迹的总长度。因此, dLastWayPos 可在任意时间被设置（或重置）为0或者一个不同的值。
bAbort	BOOL	FALSE	该输入设为 TRUE 将终止轮廓的处理。

bSingleStep	BOOL	FALSE	该输入的作用是使插补器在两个轨迹对象转化时（或具有相同正切值的转化时）停止一个周期。如果在运动时 bSingleStep 被设为 TRUE，插入器将在对象结束时停止而不超过预定的减加速度值。 如果插补器应该在下一个可能的停止位置停止（也就是速度为0的点），则 bWaitAtNextStop 必须被使用。
bAcknM	BOOL	FALSE	该输入用来确认一个队列中的附加选项（M 选项）。如果输入为 TRUE，该选项将被删除并且轨迹处理将继续。
bQuick_Stop	BOOL	FALSE	如果该输入为 TRUE，插补器将减少速度到0，直到 bQuick_Stop 被重置为 FALSE。这种减少是按照已定义的速度分布（byVelMode，见下）进行的，减加速度是由 dQuickDeceleration 的最大值和当前轨迹中的程序延迟给出的。
dQuickDeceleration	LREAL	0	用于 bQuick_Stop 的减加速度值。
VAR_OUT			
bDone	BOOL	FALSE	当输入数据(poqDataIn)被完全处理完毕后，该变量将被立即设置为 TRUE。此后,该功能块不再执行其他任何动作直到被重置。如果输入 bExecute 为 FALSE，bDone 将被重置为 FALSE。
bBusy	BOOL	FALSE	当功能块的执行还没结束时为 TRUE。
bError	BOOL	FALSE	当功能块内部发生错误时的信号。
wErrorID	SMC_ERROR	SMC_NOERROR	错误指示。
piSetPosition	SMC_POSINFO		该枚举变量描述了在进行插补时最终发生的错误。当错误发生后，处理将停止直到进行重置。 每次调用 SMC_Interpolator 都考虑到了预定义的参数、速度历史和上一个位置，计算出下一个点。一旦第一个 GEOINFO 对象的处理结束，它将被从 poqDataIn-SMC_OUTQUEUE 结构中删除。

<table>
<tr><td>iStatus</td><td>SMC_INT_STATUS</td><td>IPO_INIT</td><td>该枚举变量反映了功能块的当前状态。可能的状态：
<table>
<tr><td>IPO_UNKNOWN</td><td>0</td><td>在 SMC_Interpolator 完全传递后不可发生的内部状态。</td></tr>
<tr><td>IPO_INT</td><td>1</td><td>模块处于初始状态；DataIn 既没有被完全填满也从没有被完全填满。</td></tr>
<tr><td>IPO_ACCEL</td><td>2</td><td>模块正在加速。</td></tr>
<tr><td>IPO_CONSTANT</td><td>3</td><td>模块正在以恒定速度移动。</td></tr>
<tr><td>IPO_DECEL</td><td>4</td><td>模块正在减速。</td></tr>
<tr><td>IPO_FINISHED</td><td>5</td><td>GEOINFO 表的处理已被终止。任何之后到达DataIn 的任何其他 GEOINFO 对象都不会被处理。</td></tr>
<tr><td>IPO_WAIT</td><td>6</td><td>模块正在等待，因为发生了下列之一情况：
Emergency_Stop = TRUE
Slow_Stop = TRUE 且 Vel = 0
Wait_At_Next_Stop = TRUE 且 Vel = 0</td></tr>
</table></td></tr>
<tr><td>bWorking</td><td>BOOL</td><td>FALSE</td><td>该输出与 SMC_ControlAxisByPos 功能块的输入 bEnable 相关联。当表的处理已经开始且还没有结束时 （ IPO_ACCEL 或 IPO_CONSTANT 或 IPO_DECEL 或 IPO_WAIT），该变量值为 TRUE。否则, bWorking 将被置为 FALSE.</td></tr>
<tr><td>iActObjectSourceNo</td><td>INT</td><td>-1</td><td>DataIn 队列的当前被传递的 SMC_GEOINFO 对象的组件 iSourceLine_No。如果 SMC_Interpolator 不（再）工作（Working = FALSE）， 则值被置为“1”。</td></tr>
<tr><td>dVel</td><td>LREAL</td><td>0</td><td>该变量包含在给定时间 Ipo_Time 内从前一个位置移动到 Set_Position 的物体的当前速度。</td></tr>
<tr><td>vecActTangent</td><td>SMC_VECTOR3D</td><td></td><td>该结构包含了对位置Set_Position 有效的轨迹的方向。如果 Vel = 0 , vecAct_Tangent 被0填充。</td></tr>
<tr><td>iLastSwitch</td><td>INT</td><td>0</td><td>该输出包含了上次传递的开关量编号。注意：如果在一个周期内传递了数个开关量，只有最后一个将被处理。</td></tr>
<tr><td>dwSwitches</td><td>DWORD</td><td>0</td><td>这个 DWORD 描述了当前的所有开关量1到32的状态。DWORD 的第0位表示开关1，第31位表示开关量32。因此，与 iLastSwitch 相对的，所有的开关量都被考虑到。</td></tr>
<tr><td>dWayPos</td><td>LREAL</td><td></td><td>描述见上（输入 dLastWAyPos）
在每次调用时 SMC_Interpolator 将计算和提交--考虑到已给的参数、速度历史和上一个轨迹位置—后面的轨迹点。如果第一个 GEOINFO 对象已被处理，它将被从 poqDataIn-SMC_OUTQUEUE 结构中删除。</td></tr>
<tr><td>wM</td><td>WORD</td><td>0</td><td>如果插补器传递一个 M 对象，比如描述附加选项的一行，该输出将被设置成与 M 函数相关联的值，插补器将停止直到它得到输入 bAcknM 的确认。</td></tr>
</table>

SMC_INTERPOLATOR2DIR

这个功能块是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。从函数以及输入输出的配置看，该模块相当于功能块 SMC_Interpolator，但是它还能够反向地插补一个轨迹。

为此，输入 dOverride 可被指定为一个负值以使 SMC_Interpolator2Dir 在负方向上插补。例如，该输入可被指定为手轮的模拟速度输入，允许用户以想要地速度将手轮向前和向后移动。

该模块的使用依赖于下列先决条件：

轨迹必须被完整存储在 poqDataIn 中。由于模块可能会向前和向后交替运动，所以完整的轨迹必须可在内存中获得。

SMC_INTERPOLATOR2DIR 的使用还需要调用 SMC_Interpolator2Dir_SlowTask 模块。

组件	类型	初始值	描述
VAR_IN			
bExecute	BOOL	FALSE	当为 TRUE 是，功能块被激活。
poqDataIn	POINTER TO SMC_OUTQUEUE	0	该变量指向 SMC_OUTQUEUE 结构对象, 这个对象包含未平滑化轨迹的 SMC_GEOINFO 对象；典型地它指向前一个 SMC_CheckVelocities 模块的输出.*.DataOut* 。
bSlow_Stop	BOOL	FALSE	如果该变量被设置为 FALSE，则不停顿地通过该轨迹。否则，SMC_Interpolator 将使速度按已定义的分布图(由 VelMode 确定的)减至0，当前 GEOINFO 对象（dDecel，见下）的延迟达到最大值， 并且等待直到 bSlow_Stop 被重设为 FALSE。
bEmergency_Stop	BOOL	FALSE	一旦该输入得到 TRUE 值，SMC_Interpolator 将立即引发一个停止，就是说位置保持不变。因此，速度将被立即设置为0。
bWaitAtNextStop	BOOL	FALSE	在该变量为 FALSE 时（默认值），则不停顿地通过该轨迹。否则，SMC_Interpolator 将在下一次正常停止时（即速度为0的位置点，或者典型地，速度为0的轨迹角度）保持位置不变，并暂停直到 bWaitAtNextStop 被重置为 FALSE。
dOverride	LREAL	1	该变量可用来处理重写。dOverride 的值不允许低于0.01。特定对象的预定速度将被 dOverride 定标；所以预定速度在在线模式下可以被升高和降低。例如，dOverride=1 （默认值）的效果是：程序预定的速度将被执行，dOverride=2则将该速度翻倍后执行。 请注意：重写值可在任何时候被修改，但是只有在当前过程中没有任何加速度和减加速度时修改才被应用！
iVelMode	SMC_INT_VELMODE	TRAPEZOID	该输入定义了速度分布图。值“TRAPEZOID”表示梯形，“SIGMOID”表示 S 形。见 SMC_INT_VELMODE 的进一步说明。
dwIpoTime	DWORD	0	该变量在每次调用时都必须被设置。它以微妙来

			表示周期时间。
dLastWayPos	LREAL	0	该输入允许用户测量被插补器扭曲了的轨迹的拉伸度。输出 dWayPos 是 dLastWayPos 和当前周期内已走过的距离之和。如果 dLastWayPos=0, dWayPos 就表示的是当前轨迹段的长度。如果 dLastWayPos 被设置成和输出 dWayPos 相等，则 dWayPos 总是以当前轨迹段为单位增长，以得到已走过轨迹的总长度。因此，dLastWayPos 可在任意时间被重置为0或者一个不同的值。
bAbort	BOOL	FALSE	该输入设为 TRUE 将终止轮廓的处理。
bSingleStep	BOOL	FALSE	该输入的作用是使插补器在两个轨迹对象转化时（或具有相同正切值的转化时）停止一个周期。如果在运动时 bSingleStep 被设为 TRUE，插入器将在对象结束时停止而不超过预定的加减速值。如果插补器应该在下一个可能的停止位置停止（也就是速度为0的点），则 bWaitAtNextStop 必须被使用。
bAcknM	BOOL	FALSE	该输入用来确认一个队列中的附加选项（M 选项）。如果输入为 TRUE，该选项将被删除并且轨迹处理将继续。
bQuick_Stop	BOOL	FALSE	如果该输入为 TRUE，插补器将减少速度到0，直到 bQuick_Stop 被重置为 FALSE。这种减少是按照已定义的速度分布（byVelMode，见下）进行的，减加速度是由 dQuickDeceleration 的最大值和当前轨迹中的程序延迟给出的。
dQuickDeceleration	LREAL	0	用于 bQuick_Stop 的减加速度值。
bStartAtEnd	BOOL		TRUE:插补应在输入队列的结尾开始。
VAR_OUT			
bEndOfPath	BOOL	FALSE	在指定方向上已到达轨迹终点。
bBusy	BOOL	FALSE	当功能块的执行还没结束时为 TRUE。
bError	BOOL	FALSE	当功能块内部发生错误时的信号。
wErrorID	SMC_ERROR	SMC_NOERROR	错误指示。
piSetPosition	SMC_POSINFO		该枚举变量描述了在进行插入时最终发生的错误。当错误发生后，处理将停止直到进行重置。每次调用 SMC_Interpolator 都考虑到了预定义的参数、速度历史和上一个位置，计算出下一个点。一旦第一个 GEOINFO 对象的处理结束，它将被从 poqDataIn-SMC_OUTQUEUE 结构中删除。

<table>
<tr><td>iStatus</td><td>SMC_INT_STATUS</td><td>IPO_INIT</td><td>该枚举变量反映了函数块的当前状态。可能状态：
<table>
<tr><td>IPO_UNKNOWN</td><td>0</td><td>在SMC_Interpolator 完全传递后不可发生的内部状态。</td></tr>
<tr><td>IPO_INT</td><td>1</td><td>模块处于初始状态；DataIn 既没有被完全填满也从没有被完全填满。</td></tr>
<tr><td>IPO_ACCEL</td><td>2</td><td>模块正在加速。</td></tr>
<tr><td>IPO_CONSTANT</td><td>3</td><td>模块正在以恒定速度移动。</td></tr>
<tr><td>IPO_DECEL</td><td>4</td><td>模块正在减速。</td></tr>
<tr><td>IPO_FINISHED</td><td>5</td><td>GEOINFO 表的处理已被终止。任何之后到达 DataIn 的任何其他 GEOINFO 对象都不会被处理。</td></tr>
<tr><td>IPO_WAIT</td><td>6</td><td>模块正在等待，因为发生了下列之一的情况：Emergency_Stop = TRUE
Slow_Stop = TRUE 且 Vel = 0
Wait_At_Next_Stop = TRUE 且 Vel = 0</td></tr>
</table></td></tr>
<tr><td>bWorking</td><td>BOOL</td><td>FALSE</td><td>当表的处理已经开始且还没有结束时（IPO_ACCEL 或 IPO_CONSTANT 或 IPO_DECEL 或 IPO_WAIT），该变量值为 TRUE。否则, bWorking 将被置为 FALSE.</td></tr>
<tr><td>iActObjectSourceNo</td><td>INT</td><td>-1</td><td>DataIn 队列的当前被传递的 SMC_GEOINFO 对象的组件 iSourceLine_No。如果 SMC_Interpolator 不（再）工作（Working = FALSE）， 则值被置为“-1”。</td></tr>
<tr><td>dVel</td><td>LREAL</td><td>0</td><td>该变量包含在给定时间Ipo_Time 内从前一个位置移动到 Set_Position 的物体的当前速度。</td></tr>
<tr><td>vecActTangent</td><td>SMC_VECTOR3D</td><td></td><td>该结构包含了对位置 Set_Position 有效的轨迹的方向。如果 Vel = 0 , vecAct_Tangent 被0填充。</td></tr>
<tr><td>iLastSwitch</td><td>INT</td><td>0</td><td>该输出包含了上次传递的开关量的编号。注意：如果在一个周期内传递了数个开关量，只有最后一个将被处理。</td></tr>
<tr><td>dwSwitches</td><td>DWORD</td><td>0</td><td>这个 DWORD 描述了当前的所有开关量1到32的状态。DWORD 的第0位表示开关1，第31位表示开关32。因此，与 iLastSwitch 相对的，所有的开关量都被考虑到。</td></tr>
<tr><td>dWayPos</td><td>LREAL</td><td></td><td>描述见上（输入 dLastWAyPos）
在每次调用时 SMC_Interpolator 将计算和提交--考虑到已给的参数、速度历史和上一个轨迹位置—后面的轨迹点。如果第一个 GEOINFO 对象已被处理，它将被从 poqDataIn-SMC_OUTQUEUE 结构中删除。</td></tr>
<tr><td>nDirection</td><td>SMC_DIRECTION</td><td></td><td>当前方向，可能的值：IPO_positive, PO_negative 和 IPO_standstill。</td></tr>
<tr><td>wM</td><td>WORD</td><td>0</td><td>如果插补器传递一个 M 对象，比如描述附加选项</td></tr>
</table>

			的一行，该输出将被设置成与 M 函数相关联的值，插补器将停止直到它得到输入 bAcknM 的确认。

SMC_INTERPOLATOR2DIR_SLOWTASK

这个功能块是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。该模块负责产生反向轨迹。它被从 SMC_Interpolator2Dir 中分割出来，这样它就能在完全拉伸的低性能系统中被封装给一个低优先级任务。

组件	类型	初始值	描述
VAR_IN_OUT			
ipo2d	SMC_Interpolator2Dir		SMC_Interpolator2Dir 实例，反向轨迹为其生成。
VAR_IN			
dAngleTol	LREAL	0	反向轨迹的角容差。典型地，与原始轨迹的角容差相同。
nSizeOutQueue	UDINT		该变量包含存储反向轨迹的数据缓冲区大小。该缓冲区必须足够大以存放整条轨迹。
pbyBufferOutQueue	POINTER TO ARRAY[0..0] OF SMC_GEOINFO		指向为存储反向轨迹而分配的存储区的首字节的指针。

SMC_LIMITADDAXISDYNAMICS

这个功能块是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。它降低轨迹的速度、加速度和减加速度，以使得到的对于附加轴（Z, P, Q, U, V, W）的速度、加速度和减加速度的值不超过功能块输入值 dMaxVel, dMaxAcc and dMaxDec。

组件	类型	初始值	描述
VAR_IN			
bExecute	BOOL	FALSE	当为 TRUE 时，功能块被激活。
bAppend	BOOL	FALSE	如果在 bExecute 的上升沿为 FALSE，输出队列 poqDataOut 被清空。
poqDataIn	POINTER TO SMC_OUTQUEUE	0	该变量指向 SMC_OUTQUEUE 结构对象，该对象包含未平滑化的轨迹的 SMC_GEOINFO 对象；典型地它指向前一个模块的输出 DataOut。
wAddAxis	WORD		各比特位制定了操作应用于哪些轴： Z Bit2 P Bit6 Q Bit7 U Bit8 V Bit9 W Bit10 例子：值16#104导致 Z 和 U 轴上的限制。
dMaxVel	LREAL	0	在定义的轴上的速度最大值。
dMaxAcc	LREAL	0	在定义的轴上的加速度最大值。
dMaxDec	LREAL	0	在定义的轴上的减加速度的最大值。

nSizeOutQueue	UDINT	0	该变量包含 GEOINFO 结构对象表将要写入的缓冲区大小。该缓冲区必须至少5倍大于 GEOINFO 本身。否则，SMC_NCDecoder 将不执行任何动作。它的大小可预定义，但是只能通过重置修改。建议按下述方法创建一个缓冲：ExampleBuf: Array of SMC_GeoInfo. 划出的缓冲区大小可由操作 sizeof(ExampleBuf) 得到。
pbyBufferOutQueue	POINTER TO BYTE	0	该输入必须指向分配给 OUTQUEUE 结构的存储区的第一个字节。该区域必须至少和在 nSizeOutQueue 中定义的一样大。典型地，存储区分配是通过在 IEC 程序地声明部分定义一个数组 SMC_GEOINFO（例如，BUF: ARRAY[1..50] OF SMC_GEOINFO；一个缓冲可存储50个路径元素）来进行的。该值可被预定义，但是只能通过重置来改变。
VAR_OUT			
bDone	BOOL	FALSE	一旦输入数据（poqDataIn）被完全处理该变量将被设置为 TRUE。之后该功能块将不再执行任何动作直到重置发生。如果输入bExecute 为 FALSE，bDone 将被设为 FALSE。
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
bError	BOOL	FALSE	在功能块内部发生错误的信号。
wErrorID	SMC_ERROR	SMC_NOERROR	错误指示。
poqDataOut	POINTER TO SMC_OUTQUEUE	0	该变量指向一个 SMC_OUTQUEUE 结构，该结构管理无环圈轨迹 SMC_GEOINFO 对象。

SMC_LIMITCIRCULARVELOCITY

这个功能块是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。

该模块检查 OutQueue 中的特定元素并限制圆形元素沿其半径反方向的轨迹速度。当以恒定速度从直线运动到一个以 r 为半径的圆上时，轨迹加速度将从 0 跃升至 $\sqrt{\frac{v^2}{r}}$ 。为限制这种跃升不超过规定值，在直线和圆弧的过渡点的速度 $\sqrt{\frac{v^2}{r}}$ 不能被超过。

该模块控制两个模块间的过渡（直线到圆弧、圆弧到直线或者圆弧到圆弧），并使第一个元素的终点速度与之适应，以使加速度跃升不会超过 dMaxAccJump 。

此外，模块通过降低圆的轨迹速度到一个合适的程度来限制圆上的轨迹加速度至 dMaxAcc 。

组件	类型	初始值	描述
VAR_IN			
bExecute	BOOL	FALSE	当为 TRUE 是，功能块被激活。

bAbort	BOOL	FALSE	如果为 TRUE，中止此功能块的当前处理。
bAppend	BOOL	FALSE	当输入保持为 FALSE 时，DataOutQueue 将在每次重置时被清空。当它保持为 TRUE 时，新进入的数据将被写到 DataOutQueue 的结尾。
poqDataIn	POINTER TO SMC_ OUTQUEUE	0	该变量指向 SMC_OUTQUEUE 结构对象，该对象包含轨迹的 SMC_GEOINFO 对象；典型地它指向前一个模块地输出 *DataOut* (例如 SMC_NCDecoder).
dMaxAcc	LREAL	0	该输入变量给出对于圆弧允许的加速度最大值。等于0的值将不会引发检查。
dMaxAccJump	LREAL	0	该输入变量给出两个对象过渡时允许的加速度跃升最大值。等于0的值将不会引发检查。
nSizeOutQueue	UDINT	0	该变量包含 GEOINFO 结构对象表将被写入的数据缓冲区的大小。该缓冲区必须至少5倍大于 GEOINFO 本身。否则，SMC_NCDecoder 将不执行任何动作。它的大小可预定义，但是只能通过重置修改。建议按下述方法创建一个缓冲：ExampleBuf: Array[1…50] of SMC_GeoInfo. 划出的缓冲区大小可由操作 sizeof (ExampleBuf) 得到。
pbyBufferOutQueue	POINTER TO ARRAY[0..0] OF SMC_ GEOINFO	0	该输入必须指向分配给 OUTQUEUE 结构的存储区的第一个字节。该区域必须至少和在 nSizeOutQueue 中定义的一样大。典型地，存储区分配是通过在 IEC 程序地声明部分定义一个数组 SMC_GEOINFO（例如，BUF: ARRAY[1..50] OF SMC_GEOINFO；一个缓冲可存储50个路径元素）来进行的。该值可被预定义，但是只能通过重置来改变。
VAR_OUT			
bDone	BOOL	FALSE	一旦从..DataIn 来的输入数据被完全处理该变量将被设置为 TRUE。之后该模块将不再执行任何动作直到重置发生。如果输入 bExecute 为 FALSE，bDone 将被设为 FALSE。
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
bError	BOOL	FALSE	在功能块内部发生错误的信号。
wErrorID	SMC_ERRO R	SMC_NOER ROR	错误指示。
poqDataOut	POINTER TO SMC_ OUTQUEUE	0	该变量指向一个 SMC_OUTQUEUE 结构，该结构管理新的 SMC_GEOINFO 对象。

SMC_NCDECODER

这个功能块是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。该模块用来将一个在

CNC 编辑器中被创建的 CNC 程序（DIN 6625，G 代码）转化为 SoftMotion-GEOINFO 结构对象的表。在每次循环中程序里的一行将被解码。

组件	类型	初始值	描述
VAR_IN_OUT			
ncprog	SMC_CNC_REF		在这个变量中 CNC 程序（SMC_CNC_REF 结构）将被传递。该程序可能已经由 IEC 程序创建或者在已在 CNC 编辑器内被创建。
VAR_IN			
bExecute	BOOL	FALSE	当为 TRUE 时，功能块被激活。
bAbort	BOOL	FALSE	如果为 TRUE，中止此功能块的当前处理。
bAppend	BOOL	FALSE	当输入保持为 FALSE 时，DataOutQueue 将在每次重置时被清空。当它为 TRUE 时，新进入的数据将被写到 DataOutQueue 的结尾。
bStepSuppress	BOOL	FALSE	当该输入为 TRUE 时，CNC 程序中以'/'开始的行将被忽略。否则这些行将被处理。
piStartPosition	SMC_POSINFO	0	插入的移动应该从这里开始的点的位置。
nSizeOutQueue	UDINT	0	该变量包含 GEOINFO 结构对象表将被写入的数据缓冲区的大小。该缓冲区必须至少5倍大于 GEOINFO 本身。否则，SMC_NCDecoder 将不执行任何动作。它的大小可预定义，但是只能通过重置修改。建议按下述方法创建一个缓冲：ExampleBuf: Array[1…50] of SMC_GeoInfo. 划出的缓冲区大小可由操作 sizeof(ExampleBuf) 得到。
pbyBufferOutQueue	POINTER TO ARRAY[0..0] OF SMC_GEOINFO	0	该输入必须指向分配给 OUTQUEUE 结构的存储区的第一个字节。该区域必须至少和在 nSizeOutQueue 中定义的一样大。典型地，存储区分配是通过在 IEC 程序地声明部分定义一个数组 SMC_GEOINFO（例如，BUF: ARRAY[1..50] OF SMC_GEOINFO；一个缓冲可存储50个路径元素）来进行的。该值可被预定义，但是只能通过重置来改变。
VAR_OUT			
bDone	BOOL	FALSE	一旦从..DataIn 来的输入数据被完全处理该变量将被设置为 TRUE。之后该模块将不再执行任何动作直到重置发生。如果输入 bExecute 为 FALSE，bDone 将被设为 FALSE。
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
bError	BOOL	FALSE	在功能块内部发生错误的信号。
wErrorID	SMC_ERROR	SMC_NOERROR	错误指示。
poqDataOut	POINTER TO SMC_OUTQUEUE	0	该变量指向一个 SMC_OUTQUEUE 结构，该结构管理被解码的 SMC_GEOINFO 对象。

<table>
<tr><td>iStatus</td><td>SMC_DEC_STATUS</td><td>WAIT_PROG</td><td>该枚举变量反映了模块的当前状态。可能状态：
<table>
<tr><td>WAIT_PROG</td><td>0</td><td>程序还未找到。</td></tr>
<tr><td>READ_WORD</td><td>1</td><td>读字。</td></tr>
<tr><td>PROG_READ</td><td>2</td><td>到达程序尾部。</td></tr>
</table></td></tr>
<tr><td>iLineNumberDecoded</td><td>INT</td><td>0</td><td>该变量包含最后处理的行的行号（不是句子的编号）。</td></tr>
<tr><td>GCodeText</td><td>SMC_GCODE_TEXT</td><td></td><td></td></tr>
</table>

SMC_ROUNDPATH

这个功能块是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。SMC_ROUNDPATH 功能块与 SMC_SmoothPath 模块十分相象。它由两线相交圆整截得的圆弧的角度。

dRadius 的值是主要的参数。就是说仅当 dRadius=0 时对象的值才被考虑。如果定义的值大于两个 SMC_GEOINFO 对象其中一个的长度的一半，则这个半长被使用。

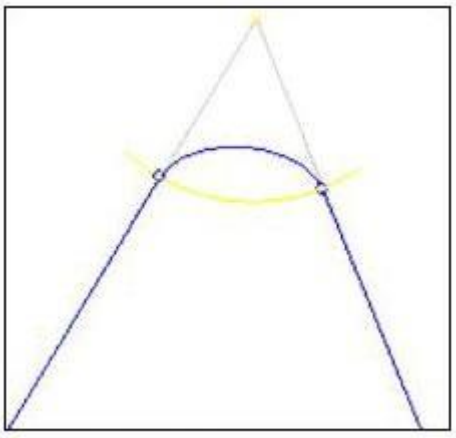


Fig. 139: Rounded corner

函数块 SMC_RoundPath 的处理过程：

所有在 Input-OUTQUEUE 结构中被找到的 SMC_GEOINFO 对象将被依次检查。如果对于某个对象，变量 Intern_Mark 的第 5 比特被设置了，那么从那时起只要后面对象的 Intern_Mark 的第 3 比特也被设置， 角度就一直被圆整。SMC_NCDecoder 模块将设置这些比特位做为对指令（G52/G50）的反应。换句话说：轨迹圆整功能在指令 G50 and G51 之间对于所有对象都将被执行。

组件	类型	初始值	描述
VAR_IN			
bExecute	BOOL	FALSE	当为 TRUE 时，功能块被激活。
bAbort	BOOL	FALSE	如果为 TRUE，中止此功能块的当前处理。
bAppend	BOOL	FALSE	当该输入被设置为 FALSE 时，DataOutQueue 将在每次重置时被清除。当它被设置为 TRUE 时，新进入的数据将被写到 DataOutQueue 的结尾。
poqDataIn	POINTER TO SMC_OUTQUEUE	0	该变量指向 SMC_OUTQUEUE 结构对象，这个对象包含未平滑化轨迹的 SMC_GEOINFO 对象；典型地它指向前一个模块的输出.*DataOut*（比如 SMC_NCDecoder）。

dRadius	LREAL	0	该输入变量定义了包括值-添加到相应的 SMC_GEOINFO-对象（G-代码参数 D）的 ToolRadius-值–到一个角度的（最小）距离的值，到达这个角度时特定对象将被切开并被样条曲线代替（见上）。该值可被在线修改。所以可以（由 SMC_GEOINFO）离线预定义并在线修改。默认：0。
dAngleTol	LREAL	0	该输入规定了临界角的值，如果一个轨迹弯曲到这个值以上将不会被平滑化。
nSizeOutQueue	UDINT	0	该变量包含GEOINFO 结构对象表将被写入的数据缓冲区的大小。该缓冲区必须至少5倍大于 GEOINFO 本身。否则，SMC_NCDecoder 将不执行任何动作。它的大小可预定义，但是只能通过重置修改。建议按下述方法创建一个缓冲：ExampleBuf: Array[1…50] of SMC_GeoInfo. 正确的缓冲区大小可由操作 sizeof(ExampleBuf) 得到。
pbyBufferOutQueue	POINTER TO ARRAY[0..0] OF SMC_ GEOINFO	0	该输入必须指向分配给OUTQUEUE 结构的存储区的第一个字节。该区域必须至少和在 nSizeOutQueue 中定义的一样大。典型地，存储区分配是通过在IEC 程序地声明部分定义一个数组 SMC_GEOINFO（例如，BUF: ARRAY[1..50] OF SMC_GEOINFO；一个缓冲可存储50个路径元素）来进行的。该值可被预定义，但是只能通过重置来改变。
VAR_OUT			
bDone	BOOL	FALSE	一旦从..*DataIn* 输入的数据被完全处理该变量将被设置为 TRUE。之后该模块将不再执行任何动作直到重置发生。如果输入 bExecute 为 FALSE，bDone 将被设为 FALSE。
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
bError	BOOL	FALSE	在功能块内部发生错误的信号。
wErrorID	SMC_ERRO R	SMC_NO ERROR	错误指示。
poqDataOut	POINTER TO SMC_ OUTQUEUE	0	该输出变量指向 SMC_OUTQUEUE 结构，该结构管理圆整过的 SMC_GEOINFO 对象。

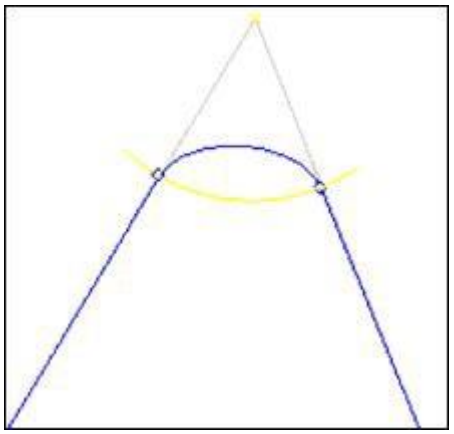
SMC_SMOOTHPATH

SMC_SmoothPath 功能块可用于轨迹预处理。它可使轨迹尖角(path angles)变得圆滑，从而生成一个圆滑的轨迹（slur path）。

它用于对轨迹的精度要求不高，而对速度要求较高，且不允许速度降到 0 的情况。

为达到此目的，轨迹会在离尖角一定的距离处被剪切，并插入一段养条曲线。距离的大小由第一个需要进行圆滑的 SMC GEOINFO 结构和其中一个输入决定。另一方面， 该距离由输入

dEdgeDistance 决定。两个值的和被作为圆弧的半径，圆弧的中心在轨迹角中，圆弧将与包围它的对象相交。



SMC_SmoothPath 程序

模块检查 input-SMC OUTQUEUE 结构中所有的 SMC GEOINFO 对象。如果发现其中一个对象的 Intern Mark 变量的 bit4 被置位了，那么轨迹圆滑(slurring)功能将被激活，直到出现 Intern Mark 变量的 bit4 和 bit3 位来作为对 G51/G50 指令的反应。也就是说，对所有处于 G51 和 G50 指令之间的对象都将进行 smooth-path 处理。

FUNCTION_BLOCK SMC_SmoothPath

组件	类型	初始值	描述
VAR_IN			
bExecute	BOOL	FALSE	为 TRUE 时，该功能块被激活。
bAbort	BOOL	FALSE	为 TRUE 时，此功能块的当前操作将会中止。
bAppend	BOOL	FALSE	若此输入为 FALSE，poqDataOut queue 会在每次重置时被清空。若为 TRUE 时，新输入的数据将会被添加到 poqDataOut-queue 的结尾。
poqDataIn	POINTER TO SMC_ OUTQUEUE	NULL	此变量指向 SMC_OUTQUEUE-结构对象，其中包含了平滑路径的 SMC GEOINFO 对象。通常是指向前面模块 (e.g.SMC_NCDecoder) 的输出 Dataout。
dEdgeDistance	LREAL	0	这种输入变量包含值，其增加到相应的 SMC GEOINFO-object 的 toolradius 值，支配距离（最小的）到一个角度，特定对象将被剪切并被齿条代替（见上文）。这个值可以在线修改。因此，它可能在在线状态下纠正一个离线重定义（通过 SMC GEOINFO-structure）。
dAngleTol	LREAL	0.001	这个输入指定角度公差的值，直到一个不平滑的路径转弯。
nSizeOutQueue	UDINT	0	这个变量包含数据缓冲区的大小，GEOINFO 结构对象的列表将被写入。这个缓冲区必须至少是 GEOINFO 结构本身的5倍，大约有2KB 左右。否则 SMC NCDecoder 不会执行任何指令。它的大小可能是预先确定，但可能只在复位后修改。建议创建一个诸如下例的缓冲区： ExampleBuf: ARRAY[1..50] OF SMC_GeoInfo;

			通过使用操作符 sizeof(ExampleBuf)，正确的缓冲区大小将会被返回。
pbyBufferOutQueue	POINTER TO ARRAY[0..0] OF SMC_GEOINFO	NULL	这个输入必须指向存储区的第一个字节，该存储区由 OUTQUEUE structure 分配。该区域必须至少像 nSizeOutqueue 定义的那样大。典型的，存储区的分配通过定义一个 SMC GEOINFO 的阵列（若 BUF：SMC GEOINFO 的阵列[1...50]，对于一个可以存储50个路径元素的缓冲区）由 IEC-program 的声明部分完成。该值可能是预先确定。
eMode	SMC_SMOOTHPATHMODE	SP_SPLINE3	此输入定义了应用在平滑轴的元素类型：SP_SPLINE3 or SP_SPLINE5。
bSymmetricalDistances	BOOL	TRUE	如果 bSymmetricalDistances 被设定，它将用于检查已经完成角度的较短距离对象是否比 D 短。如果为 TRUE，则将会使用较短距离；如果为 FALSE，将会使用角度距离 D。
eAddAxMode	SMC_SMOOTHPATHADDAXMODE	SPAA_LATE	该输入定义了附加轴的圆滑元素类型。
VAR_OUT			
bDone	BOOL	FALSE	一旦从poq*DataIn* 输入的数据被完全处理该变量将被设置为 TRUE。之后该模块将不再执行任何动作直到重置发生。如果输入 bExecute 为 FALSE，bDone 将被设为 FALSE。
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
bError	BOOL	FALSE	在功能块内部发生错误的信号。
wErrorID	SMC_ERROR	SMC_NOERROR	错误代码
poqDataOut	POINTER TO SMC_OUTQUEUE	NULL	该输出变量指向 SMC_OUTQUEUE 结构，该结构管理圆整过的 SMC_GEOINFO 对象

平滑曲线

G-code

N000 G01 X10 Y0 F100

N010 G51 D2 (平滑角度和角度距离 2 打开)

N020 G01 X10 Y20

N030 G01 X20 Y20

N040 G01 X20 Y0

N050 G50 (平滑角度关闭)

N060 G01 X30

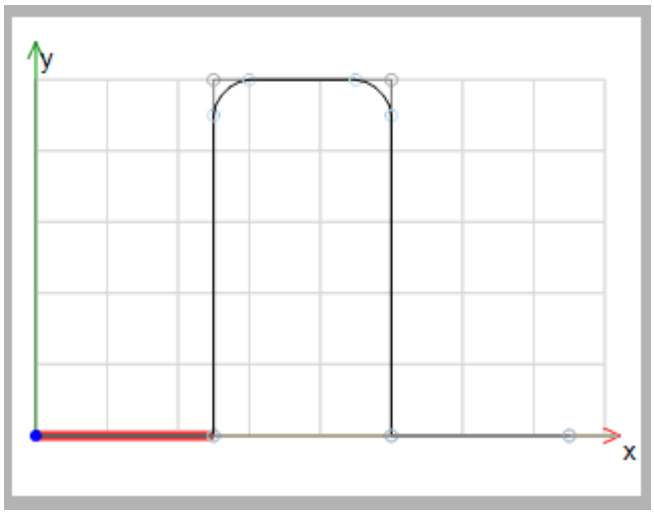


图 1　平滑角度的路径

以上图片表示平滑角度的路径。位置为 10/0 时平滑打开，位置为 20/0 时平滑关闭。这些位置之间的所有角度均通过曲线平滑。

输入 bSymmetricalDistances

G-code

N000 G01 X10 Y0 F100

N010 G51 D10 (平滑角度和角度距离 2 打开)

N020 G01 X10 Y20

N030 G01 X20 Y20

N040 G01 X20 Y0

N050 G50 (平滑角度关闭)

N060 G01 X30

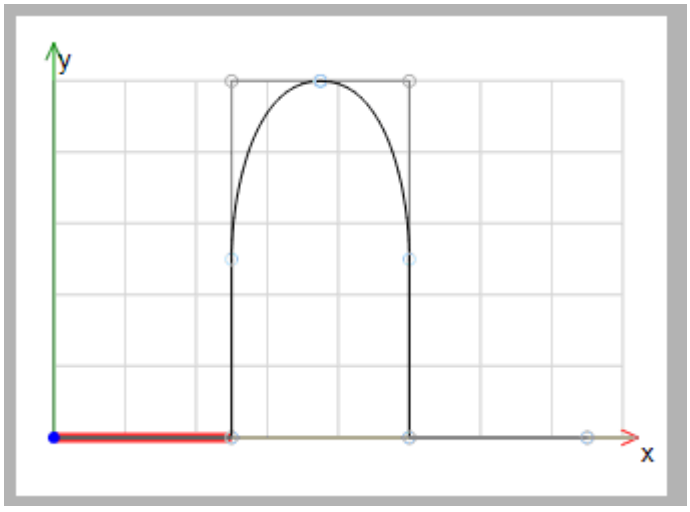


图 2　bSymmetricalDistances=FALSE 时的路径

当角度距离大于对象的一半时，以上例子显示了功能块的曲线。第一个图片显示了 bSymmetricalDistances=FALSE 的曲线。这里对象太短，所以它平滑到角度的中心（见水平线）。

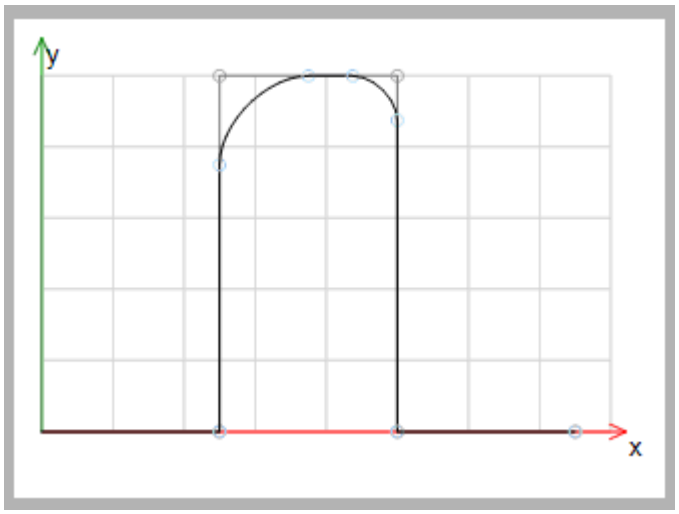


图 3　bSymmetricalDistances=TRUE 时的路径

此处设置了 bSymmetricalDistances。直到第一个角度被检测到，不论较短的对象的一半长度比角度距离由 D(10)定义）大。因此，两个对象均将被应用于角度距离 5。到达第 2 个角度，线元素缩短至长度 5，因此剩下的一半对象得到长度的 2.5，2.5 的角度距离被应用于两个对象。

输入 eMode

G-code

N000 G51 D10 (平滑角度和角度距离 2 打开)

N010 G01 X0 Y10

N020 G01 X10 Y10

N030 G01 X10 Y0

N040 G50 (平滑角度关闭)

通过设置功能块 SMC_SmoothPath 的输入 eMode，用户可以定义应用于平滑的元素类型。默认类型为 "Spline3" (SP_SPLINE3)，也就是每个维度的一个三阶多项式。然而，这个可以变为 "线 5"。五阶多项式有优势，通过二次导数可以完成前一个/下一个对象之间连续连接。这影响了一个有限的 jerk。缺点是计算时间比使用三阶线的多。

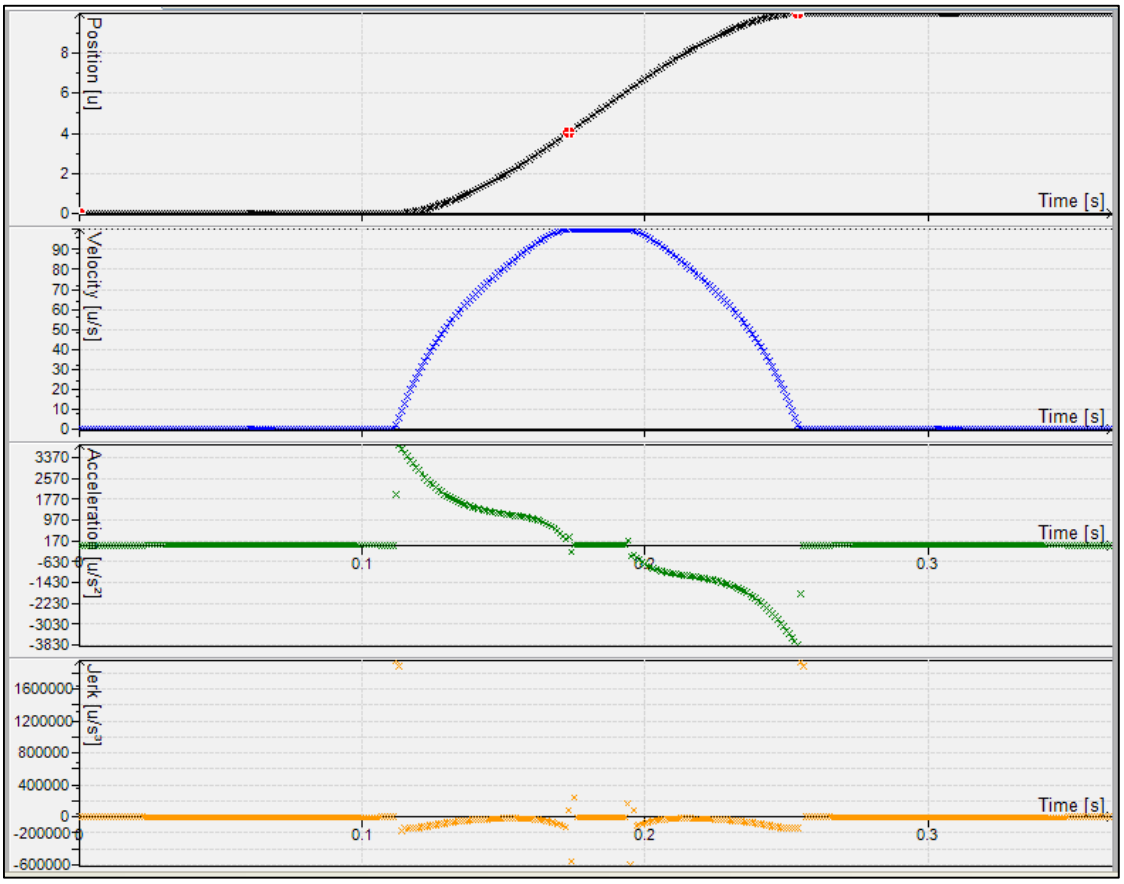


图 4　eMode=Spline3 时沿 X 轴的动态曲线

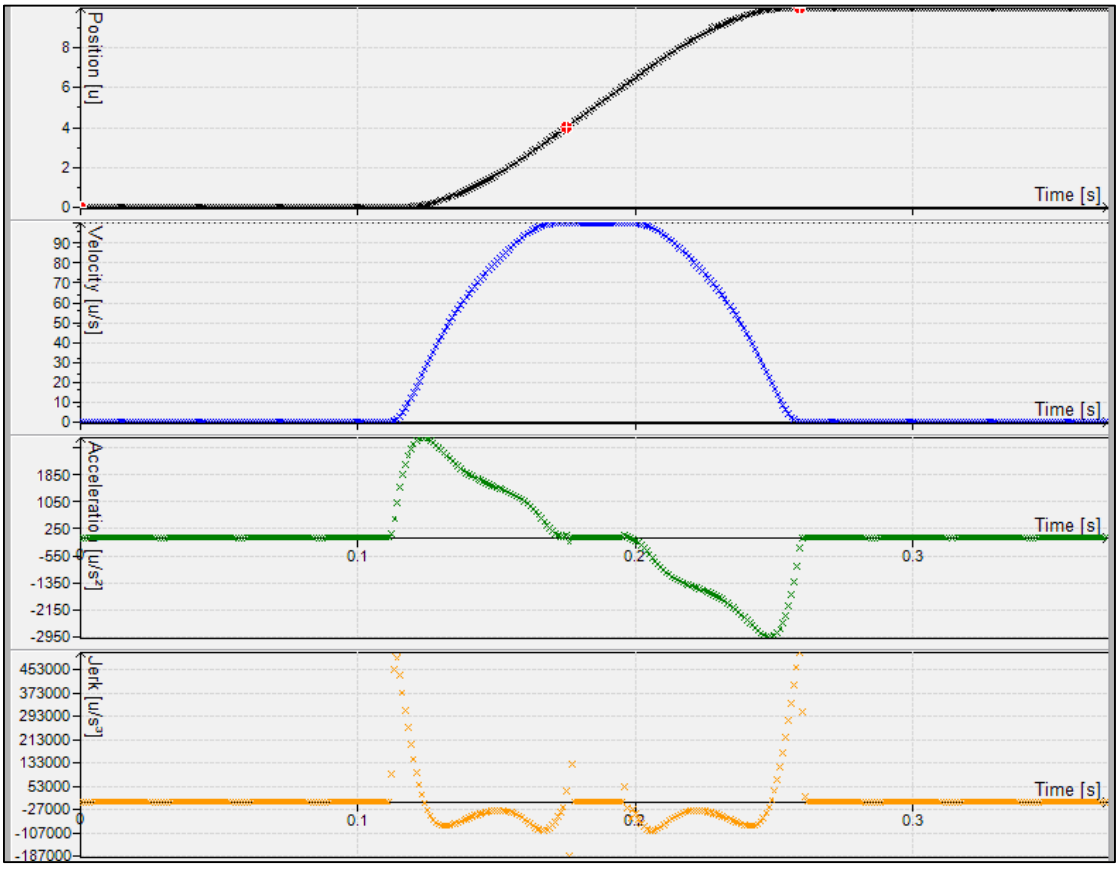
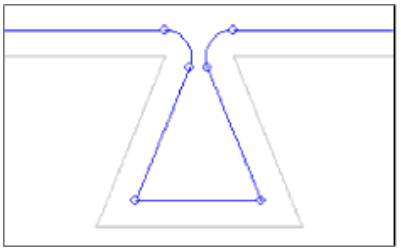


图 5　eMode=Spline5 时沿 X 轴的动态曲线

SMC_TOOLCORR

The SMC_ToolCorr-module 可用于轨迹预处理。它以原文件为基础创建一个新路径。偏移轨迹中的每个点到原轨迹点和紧靠着的邻点（刀具半径补偿）的距离都是一定的。因此，偏移轨迹保证了它上面的每个点与原轨迹都有一个固定距离。典型的应用都是用一个磨床来磨削一个轮廓，并且已经定义了磨盘直径。为了补偿磨盘的半径，磨盘必须沿合适的偏移轨迹来运动，此偏移轨迹可以用 SMC ToolCorr 模块来生成。



注意以下限制：如果外轮廓和磨削半径是按如下方式来选择：轨迹中的交汇点会在偏移轨迹中得到解决。那么它在通过偏移轨迹时，会破坏想要得到的轮廓（见如下例子）。为了避免这种交叉，可使用 SMC AVOIDLOOP 模块。

SMC_ToolCorr module 程序：

要一个一个地检查 INPUT-OUTPUTQUEUE 结构中的所有 SMC GEOINFO 对象。如果某个对象中 INTERN MARK 变量的 BIT1（BIT2）被置位了，那么从此处开始并按运动方向，轨迹会根据当前设置的刀具半径向左（右）偏移。

为了得到一个连续的轨迹，两种模式(EMODE)之一将被选择：

TC RAMP IN：目前将用于坡道进入 OFFSETTED 路径，这意味着开始位置保持不变，但是最终位置被补偿。所有进一步的对象也被补偿直到刀具半径修正通过 INTERN MASK 的 BIT0 的设定关闭。然后，该对象的开始位置仍然是补偿，但最终位置是原来的。

TC BLIND POS：定位对象（SMC GEOINFO.IMOVETYPE=100）插入， RESP. 如果这样的定位对象已经在对象之前，它将会直接转换到转换路径的起始点。每个进一步对象也将补偿，直到 INTERN MARK 的 BIT0 置位，其会阻止刀具半径修正。此外，在此时路径的连续诉讼由使用配置对象保证。

SMC NCDECODER 将会在说明 G41/G42/G40 的回应内设置这些 bits。换句话说：该道具半径更正工作将完成 FPR 所有定位在 G41 和 G40 RESP. G42 和 G40 之间的对象。

组件	类型	初始值	描述
VAR_IN			
bExecute	BOOL	FALSE	为 TRUE 时，该功能块被激活。
bAppend	BOOL	FALSE	若此输入为 FALSE，poqDataOut queue 会在每次重置时被清空。若为 TRUE 时，新输入的数据将会被添加到 poqDataOut-queue 的结尾。
poqDataIn	POINTER TO SMC_OUTQUEUE		此变量指向 SMC_OUTQUEUE-结构对象，其中包含了平滑路径的 SMC GEOINFO 对象。通常是指向前面模块 (e.g.SMC_NCDecoder) 的输出 Dataout。应该使用适当的尺寸（见上文）。

dToolRadius	LREAL	0	此变量包含的值，到当前的 SMC_GEOINFO-object ToolRadius，决定了刀具将被移动的轨迹半径（见上文）。此值可以在线修改。因此可以在线预定义值（通过 SMC_GEOINFO-structure）和在线修改。 考虑到块偏移时被初始化刀具半径补偿，可能导致路径补偿中止，因此需要避免这种情况。但是可以在重启或一个阶段来执行半径补偿，保证模块当前未偏移一个块(Status = TC_ORIG)。
nSizeOutQueue	UDINT		这个变量包含的数据缓冲区的大小，GEOINFO 结构对象的列表将被写入。这个缓冲区必须至少是 GEOINFO 结构本身的5倍，大约有2KB 左右。否则，SMC NCDecoder 不会执行任何指令。它的大小可能是预先确定的，但可能只在复位后修改。 建议参考如下例子建立缓冲区： ExampleBuf: Array[1..50] of SMC_GeoInfo. 通过使用操作符 sizeof(ExampleBuf)，可以取回正确的缓冲区大小。
pbyBufferOutQueue	POINTER TO ARRAY[0..0] OF SMC_GEOINFO		这个输入必须指向存储区的第一个字节，该存储区由 OUTQUEUE structure 分配。该区域必须至少像 nSizeOutqueue 定义的那样大。典型的，存储区的分配通过定义一个SMC GEOINFO 的阵列（若 BUF：SMC GEOINFO 的阵列[1...50]，对于一个可以存储 50 个路径元素的缓冲区） 由 IEC-program 的声明部分完成。 该值可能是预先确定。
eMode	SMC_TOOLCORRMODE	TC_RAMP_IN	定义偏移路径和无偏移路径之间转换的模式。
VAR_OUT			
bDone	BOOL	FALSE	一旦从 poq*DataIn* 输入的数据被完全处理该变量将被设置为 TRUE。之后该模块将不再执行任何动作直到重置发生。
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
bError	BOOL	FALSE	在功能块内部发生错误的信号。
wErrorID	SMC_ERROR	0	错误代码
poqDataOut	POINTER TO SMC_OUTQUEUE		该输出变量指向 SMC_OUTQUEUE 结构，该结构管理圆整过的 SMC_GEOINFO 对象。
iStatus	SMC_TC_STATUS	TC_ORIG	此枚举变量表示模块的当前状态，可能的状态为： TC_ORIG: 0 没有使用刀具对该对象进行半径校正。 TC_RIGHT: 1 没有使用刀具对该对象进行半径校正。 TC_LEFT: 2 没有使用刀具对该对象进行半径校正。 TC_END: 4

			没有使用刀具对该对象进行半径校正。

SMC_XINTERPOLATOR

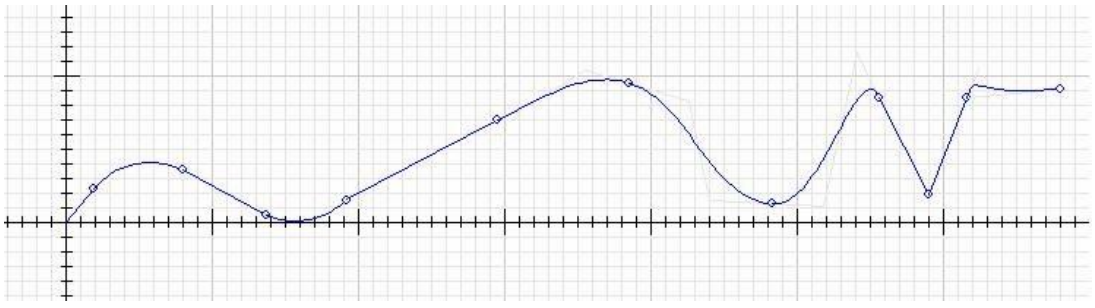
该 SMC_XInterpolator 功能块实现了 CAM 和 CNC 的混合。假设你想把一个工件切一个特定的形状（由 G 代码描述），根据工件由另一个进程的移动（例如，沿 X 轴移动）和其他的轴（y，z 等）应当按照工件（X）当前位置控制，并且目标由路径要点给出。

工件的运动通常紧跟 X 方向（其它情况可以按此机械的计划）

组件	类型	初始值	描述
VAR_IN_OUT			
X_AXIS	AXIS_REF_SM3		X 轴
VAR_IN			
bExecute	BOOL	FALSE	为 TRUE 时，该功能块被激活。
poqDataIn	POINTER TO SMC_OUTQUEUE	0	此变量指向 SMC_OUTQUEUE-结构对象，其中包含了将被插补的 SMC GEOINFO 路径对象。
dLastWayPos	LREAL	0	该输入使用户测量被插补器覆盖的路径长度。 输出dWayPos 在当前循环是 dLastWayPos 和被覆盖距离的总和。 如果 dLastWayPos=0，则 dWayPos 得到当前路径段的长度。 如果你将 dLastWayPos 设置与输出 dWayPos 相同， dWayPos 将永远通过当前路径段增加，你将得到路径长度的总和。因此 dLastWayPos 一直可被设为 0 或其它值。
bAbort	BOOL	FALSE	此输入设为 TRUE 时，一个轮廓的运行将会被中止。
eDirection	MC_DIRECTION	positive	该输入决定，工件是否沿 X 轴向正或负方向移动。不允许其它的值。
dXOffset	LREAL	0	相对于 X 轴位置的偏移。
VAR_OUT			
bDone	BOOL	FALSE	一旦从 poq*DataIn* 输入的数据被完全处理该变量将被设置为 TRUE。之后该模块将不再执行任何动作直到重置发生。如果输入 bExecute 为 FALSE，bDone 将被设为 FALSE。
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
bError	BOOL	FALSE	在功能块内部发生错误的信号。
wErrorID	SMC_ERROR	SMC_NOERROR	错误代码
piSetPosition	Pointer TO SMC_POSINFO		按照要求计算 piSetPosition 包含的目标位置 piSetPosition 是一个 SMC_POSINFO-structure，其不仅包括将要到达的轨迹的直角坐标，还包括附加轴的位置。
iStatus	SMC_INT_STA	IPO_INIT	此枚举变量描述了模块的当前状态，可能的状态

	TUS		如下： IPO_UNKNOWN (0)：内部状态，当 SMC_Interpolator 运行完之后将不会出现。 IPO_ACCEL (2)：模块当前加速状态 IPO_CONSTANT (3)：模块当前按一致速度运行。 IPO_DECEL (4)：模块当前制动。 IPO_FINISHED (5)：GEOINFO-list 操作被终止。 GEOINFO-objects 随后到达DataIn，将会被执行。
bWorking	BOOL	FALSE	当列表开始执行且并未终止时，此输出被设为 TRUE (IPO_ACCEL 或 IPO_CONSTANT 或 IPO_DECEL 或 IPO_WAIT). 否则 bWorking 为 FALSE.
iActObjectSourceNo	INT	-1	包含当前通过的 SMC_GEOINFO 条目 iSourceLine_No-DataIn-queue 的对象。如果 SMC_Interpolator 再也不奏效(bWorking = FALSE)，则设置为“-1”。
dVel	LREAL	0	此变量包含当前速度结果， 如果一个对象在 Ipo_Time 时间内从之前的坐标移动到 piSetPosition。.
vecActTangent	SMC_VECTOR3D		此结果包含恰当的路径方向 piSetPosition，若 dVel = 0， vecAct_Tangent 将只包含0。
iLastSwitch	INT	0	此变量包含最后通过的辅助标记的数量。 如果在一个循环中多个辅助标记也可以通过，仅最后一个需要总被丢弃。
dwSwitches	DWORD	0	This DWORD 包含所有辅助标记1~32之间的当前开关状态。 DWORD 的 Bit0 表示辅助标记1, Bit31 表示辅助标记 32。因此，除了 iLastHelpMark，辅助标记的损失可以被消除。
dWayPos	LREAL		参考输入 dLastWayPos 的描述。

如果XInterpolator 是活动的，将在指定的匹配当前 X 坐标的路径搜索该点，且它将在 pisetposition 中堆放相应的路径点。为了对每个 X 坐标实现与阶跃无关的路径，特有的路径点通常必须存在。例如，下面的路径是有效的：



SMC_GETMPARAMETERS

这个功能块是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。插补器正处理一个 M 函数，该模块用来轮询已为该 M 函数设置的参数（K，L，O，见 HERE）。

组件	类型	初始值	描述

VAR_IN_OUT			
Interpolator	SMC_Interpolator		插补器实例。
VAR_IN			
bEnable	BOOL	FALSE	当为 TRUE 时，功能块被激活。
VAR_OUT			
bMActive	BOOL	FALSE	值 TRUE 指示了当前被处理的 M 函数。
dK	LREAL	0	由字 K 指定的 M 参数。
dL	LREAL	0	由字 L 指定的 M 参数。
MParameters	SMC_M_PARAMETERS		被全局数据结构 gSMC_MParameters 或被由 O 传递的变量指定的 M 参数。

SMC_PREACKNOWLEDGEMFUNCTIONS

这个功能块是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。它用来在插补器到达该模块之前确认一个 M 函数。这样在 M 函数上的停顿就能够避免了。

组件	类型	初始值	描述
VAR_IN			
bEnable	BOOL	FALSE	当为 TRUE 时，功能块被激活。
iM	INT	FALSE	M 函数的编号。
poqDataIn	POINTER TO SMC_OUTQUEUE	0	指向 SMC_OUTQUEUE 结构对象, 这个对象包含要处理的数据。
VAR_OUT			
bDone	BOOL	FALSE	一旦输入数据（poqDataIn）被完全处理该变量将被设置为 TRUE。之后该模块将不再执行任何动作直到重置发生。如果输入 bExecute 为 FALSE，bDone 将被设为 FALSE。
bBusy	BOOL	FALSE	当功能块执行还没结束时为 TRUE。
bError	BOOL	FALSE	在功能块内部发生错误的信号。
wErrorID	SMC_ERROR	SMC_NOERROR	错误指示。

6.3.3 SM_Trafo_POUs

Additional FBs 文件夹内

SMC_CALCDIRECTIONFROMVECTOR

这个功能块是“CoDeSys SoftMotion”的库“SM3_CNC.library”中的一部分。使用它时工具的方向（dAlpha）可被计算。

注意：关于变换模块的详细信息请看 SoftMotion_Transformations 的概览。

组件	类型	初始值	描述
VAR_IN			
v	SMC_VECTOR3D		切向量 v。一般的，v 和插补器中的输出 vecActTangent 一样。
eDir	SMC_VECTOR	SMC_tangen	输入 eDir 确定了，方向是否必须被计算成与轨迹

	3D	tial	切线平行（SMC_tangential），或者反向（SMC_opp_tangential）或者与轨迹正交（SMC_orthogonal_r (在轨迹切线右侧) 以及 SMC_orthogonal_l (在轨迹切线左侧)）。
VAR_OUT			
dDir	LREAL		角 dDir 以角度制测量。在插补器停顿时，也就是当 v 是0向量时，它将保持不变。eDir 基本上用做一个方向轴的 SMC_ControlAxisByPos 的设定量，或者是用做变换的输入 dAlpha。

Gantry Systems 文件夹内

SMC_TRAFO_GANTRY2

对于龙门系统不能进行变换，因此模块只能给 x,y 轴加上偏移量。

备注：有关转换模块的详细资料，请参考 SoftMotion Transformations.

组件	类型	初始值	描述
VAR_IN			
pi	SMC_POSINFO		目标位置向量 (x,y)，插补器的输出。
dOffsetX	LREAL		X 轴的附加偏移量
dOffsetY	LREAL		
VAR_OUT			
dx	LREAL		X 轴的目标位置
dy	LREAL		Y 轴的目标位置

SMC_TRAFO_GANTRY2TOOL1

功能块用来反求龙门系统的刀具轴向偏置，也就是相对 Z 轴的轴向偏置，它们近似在一条直线上。

备注：有关转换模块的详细资料，请参考 SoftMotion Transformations。

组件	类型	初始值	描述
VAR_IN			
pi	SMC_POSINFO		目标位置向量 (x,y)，插补器的输出。
dOffsetX	LREAL		X 轴的附加偏移量
dOffsetY	LREAL		Y 轴的附加偏移量
VAR_OUT			
dAlpha	LREAL		刀具偏置角度，可以由 SMC_CalcDirectionFromVector 计算得出。
dToolA	LREAL		刀具大小 = 距离（设定位置）-机械位置
dx	LREAL		X 轴的目标位置
dy	LREAL		Y 轴的目标位置

SMC_TRAFO_GANTRY2TOOL2

功能块用来反求龙门系统的刀具轴向偏置，也就是相对 Z 轴的轴向偏置，它们近似成直角。

备注：有关转换模块的详细资料，请参考 SoftMotion Transformations。

组件	类型	初始值	描述
VAR_IN			
pi	SMC_POSINFO		目标位置向量 (x,y)，插补器的输出。
dOffsetX	LREAL		X 轴的附加偏移量
dOffsetY	LREAL		Y 轴的附加偏移量
VAR_OUT			
dAlpha	LREAL		刀具偏置角度，可以由 SMC_CalcDirectionFromVector 计算得出。
dToolA	LREAL		刀具 A 的大小 = 距离（设定位置）-机械位置
dToolB	LREAL		刀具 B 的大小 = 距离（设定位置）-机械位置
dx	LREAL		X 轴的目标位置
dy	LREAL		Y 轴的目标位置

SMC_TRAFO_GANTRY3

对于龙门系统不能进行变换，因此模块只能给 x,y,z 轴加上偏移量。

备注：有关转换模块的详细资料，请参考 SoftMotion Transformations。

组件	类型	初始值	描述
VAR_IN			
pi	SMC_POSINFO		目标位置向量(x,y)，插补器的输出
dOffsetX	LREAL		X 轴附加偏移量
dOffsetY	LREAL		Y 轴附加偏移量
dOffsetZ	LREAL		Z 轴附加偏移量
VAR_OUT			
dx	LREAL		X 轴目标位置
dy	LREAL		Y 轴目标位置
dz	LREAL		Z 轴目标位置

SMC_TRAFO_GANTRYCUTTER2

为一个正在被控制并指出沿着路径切线旋转轴的三维龙门系统计算一个逆向变换。

备注：有关转换模块的详细资料，请参考 SoftMotion Transformations。

组件	类型	初始值	描述
VAR_IN			
pi	SMC_POSINFO		目标位置向量(x,y)，插补器的输出
v	SMC_VECTOR3D		向量的实际轨迹切线，插补器的输出。
dOffsetX	LREAL		X 轴附加偏移量
dOffsetY	LREAL		Y 轴附加偏移量
dOffsetR	LREAL		转动轴附加偏移量。
iDirectionR	INT	1	旋转方向
VAR_OUT			
dx	LREAL	0	X 轴目标位置
dy	LREAL	0	Y 轴目标位置
dr	LREAL	0	目标转换位置（以度为单位）

SMC_TRAFO_GANTRYCUTTER3

为一个正在被控制并指出沿着路径切线旋转轴的三维龙门系统计算一个逆向变换。

备注：有关转换模块的详细资料，请参考 SoftMotion Transformations。

组件	类型	初始值	描述
VAR_IN			
pi	SMC_POSINFO		目标位置向量 (x,y)，插补器的输出
v	SMC_VECTOR3D		向量的实际轨迹切线，插补器的输出。
dOffsetX	LREAL		X 轴附加偏移量
dOffsetY	LREAL		Y 轴附加偏移量
dOffsetZ	LREAL		Z 轴附加偏移量
dOffsetR	LREAL		转动轴附加偏移量。
iDirectionR	INT	1	旋转方向
VAR_OUT			
dx	LREAL	0	X 轴目标位置
dy	LREAL	0	Y 轴目标位置
dz	LREAL	0	Z 轴目标位置
dr	LREAL	0	目标转换位置（以度为单位）

SMC_TRAFO_GANTRYH2

这个转变模块用来提供解决有着固定的驱动器的 H 龙门系统的反向转变。

备注：有关转换模块的详细资料，请参考 SoftMotion Transformations。

组件	类型	初始值	描述
VAR_IN			
pi	SMC_POSINFO		目标位置向量 (x,y)，插补器的输出
dOffsetX	LREAL		X 轴附加偏移量
dOffsetY	LREAL		Y 轴附加偏移量
VAR_OUT			
da	LREAL		a 轴目标位置
db	LREAL		b 轴目标位置

SMC_TRAFOF_GANTRY2

功能模块用来操作一个没有动作的三维龙门操作系统，因此，该模块仅在 X,Y 轴各自增加了一个偏移。每一个 SMC_TRAFOF_GANTRY2 模块的实例都能与一个命名为 SMC_VISU_GANTRY2 的可视化模板相连接。

备注：有关转换模块的详细资料，请参考 SoftMotion Transformations。

组件	类型	初始值	描述
VAR_IN_OUT			
DriveX	AXIS_REF_SM3		参考 X 轴。
DriveY	AXIS_REF_SM3		参考 Y 轴。
VAR_IN			
dOffsetX	LREAL		X 轴的偏移量。相当于 SMC_TRAFO_Gantry2.

dOffsetY	LREAL		Y 轴的偏移量。相当于 SMC_TRAFO_Gantry2.
minX	LREAL		X 轴方向的移动下限（用于可视化）
maxX	LREAL		X 轴方向的移动上限（用于可视化）
minY	LREAL		Y 轴方向的移动下限（用于可视化）
maxY	LREAL		Y 轴方向的移动上限（用于可视化）
VAR_OUT			
dx	LREAL		GEO 坐标系中 X 的位置。
dy	LREAL		GEO 坐标系中 Y 的位置。
dnx	LREAL		标准位置的 X 分量 (大小介于 [0,1]).
dny	BOOL		标准位置的 Y 分量 (大小介于 [0,1]).
ratio	LREAL		X 间隔与 Y 间隔的比值
dnOffsetX	LREAL		X 轴可视化偏移量
dnOffsetY	LREAL		Y 轴可视化偏移量

SMC_TRAFOF_GANTRY2TOOL1

功能块用来反求龙门系统的刀具轴向偏置，也就是相对 Z 轴的轴向偏置，它们近似在一条直线上。详情请参阅运动学相关知识。

备注：有关转换模块的详细资料，请参考 SoftMotion Transformations。

组件	类型	初始值	描述
VAR_IN_OUT			
DriveX	AXIS_REF_SM3		参考 X 轴
DriveY	AXIS_REF_SM3		参考 Y 轴
VAR_IN			
dOffsetX	LREAL		X 轴的偏移量，相当于 SMC_TRAFO_Gantry2.
dOffsetY	LREAL		Y 轴的偏移量，相当于 SMC_TRAFO_Gantry2.
dAlpha	LREAL		刀具偏置角度，可以由 SMC_CalcDirectionFromVector 计算得出。
dToolA	LREAL		刀具尺寸=距离（设定位置）-（机械位置）
minX	LREAL		X 轴方向的移动下限（用于可视化）
maxX	LREAL		X 轴方向的移动上限（用于可视化）
minY	LREAL		Y 轴方向的移动下限（用于可视化）
maxY	LREAL		Y 轴方向的移动上限（用于可视化）
VAR_OUT			
dx	LREAL		GEO 坐标系中 X 的位置。
dy	LREAL		GEO 坐标系中 Y 的位置。
dnx	LREAL		标准位置的 X 分量 (大小介于 [0,1]).
dny	BOOL		标准位置的 Y 分量 (大小介于 [0,1]).
dnl	LREAL		刀具的标准长度
ratio	LREAL		X 间隔与 Y 间隔的比值
dnOffsetX	LREAL		X 轴可视化偏移量
dnOffsetY	LREAL		Y 轴可视化偏移量

SMC_TRAFOF_GANTRY2TOOL2

功能块用来反求龙门系统的刀具轴向偏置，也就是相对 Z 轴的轴向偏置，它们近似成直角。

备注：有关转换模块的详细资料，请参考 SoftMotion Transformations。

组件	类型	初始值	描述
VAR_IN_OUT			
DriveX	AXIS_REF_SM3		参考 X 轴
DriveY	AXIS_REF_SM3		参考 Y 轴
VAR_IN			
dOffsetX	LREAL		X 轴的偏移量，相当于 SMC_TRAFO_Gantry2.
dOffsetY	LREAL		Y 轴的偏移量，相当于 SMC_TRAFO_Gantry2.
dAlpha	LREAL		刀具偏置角度，可以由 SMC_CalcDirectionFromVector 计算得出。
dToolA	LREAL		A 轴的工具尺寸（见图示）
dToolB	LREAL		B 轴的工具尺寸（见图示）
minX	LREAL		X 轴方向的移动下限（用于可视化）
maxX	LREAL		X 轴方向的移动上限（用于可视化）
minY	LREAL		Y 轴方向的移动下限（用于可视化）
maxY	LREAL		Y 轴方向的移动上限（用于可视化）
VAR_OUT			
dx	LREAL		GEO 坐标系中 X 的位置。
dy	LREAL		GEO 坐标系中 Y 的位置。
dnx	LREAL		标准位置的 X 分量 (大小介于 [0,1]).
dny	BOOL		标准位置的 Y 分量 (大小介于 [0,1]).
dntx	LREAL		刀具的三角形刀口标准位置的 X 分量
dnty	LREAL		刀具的三角形刀口标准位置的 Y 分量
dnl1	LREAL		A 刀具的标准长度
dnl2	LREAL		B 刀具的标准长度
ratio	LREAL		X 间隔与 Y 间隔的比值
dnOffsetX	LREAL		X 轴可视化偏移量
dnOffsetY	LREAL		Y 轴可视化偏移量

SMC_TRAFOF_GANTRY3

功能模块用来操作一个没有动作的三维龙门操作系统，因此，该模块仅在 X,Y,Z 轴各自增加了一个偏移。每一个 SMC_TRAFOF_GANTRY3 模块的实例都能与一个命名为 SMC_VISU_GANTRY3 的可视化模板相连接。

备注：有关转换模块的详细资料，请参考 SoftMotion Transformations。

组件	类型	初始值	描述
VAR_IN_OUT			
DriveX	AXIS_REF_SM3		X 轴
DriveY	AXIS_REF_SM3		Y 轴
DriveZ	AXIS_REF_SM3		Z 轴
VAR_IN			
dOffsetX	LREAL		X 轴的偏移量，相当于 SMC_TRAFO_Gantry3
dOffsetY	LREAL		Y 轴的偏移量，相当于 SMC_TRAFO_Gantry3

dOffsetZ	LREAL		Z 轴的偏移量，相当于 SMC_TRAFO_Gantry3
minX	LREAL		X 轴方向的移动下限（用于可视化）
maxX	LREAL		X 轴方向的移动上限（用于可视化）
minY	LREAL		Y 轴方向的移动下限（用于可视化）
maxY	LREAL		Y 轴方向的移动上限（用于可视化）
VAR_OUT			
dx	LREAL		GEO 坐标系中 X 的位置。
dy	LREAL		GEO 坐标系中 Y 的位置。
dz	LREAL		GEO 坐标系中 Z 的位置。
dnx	LREAL		标准位置的 X 分量 (大小介于 [0,1])，用于可视化
dny	BOOL		标准位置的 Y 分量 (大小介于 [0,1])，用于可视化
ratio	LREAL		X 间隔与 Y 间隔的比值
dnOffsetX	LREAL		标准位置的 X 偏移量 (大小介于 [0,1])，用于可视化
dnOffsetY	LREAL		标准位置的 Y 偏移量 (大小介于 [0,1])，用于可视化

SMC_TRAFOF_GANTRYCUTTER2

该模块将解决有着指向沿着切线方向的旋转轴的三维龙门系统的前向转变。

备注：有关转换模块的详细资料，请参考 SoftMotion Transformations。

组件	类型	初始值	描述
VAR_IN_OUT			
DriveX	AXIS_REF_SM3		参考轴 X
DriveY	AXIS_REF_SM3		参考轴 Y
DriveR	AXIS_REF_SM3		旋转轴
VAR_IN			
dOffsetX	LREAL		X 轴的偏移量，类似于 SMC_TRAFO_Gantry2
dOffsetY	LREAL		Y 轴的偏移量，类似于 SMC_TRAFO_Gantry2
dOffsetR	LREAL		旋转轴的偏移量
iDirectionR	INT	1	旋转方向
minX	LREAL		X 轴方向的移动下限（用于可视化）
maxX	LREAL		X 轴方向的移动上限（用于可视化）
minY	LREAL		Y 轴方向的移动下限（用于可视化）
maxY	LREAL		Y 轴方向的移动上限（用于可视化）
VAR_OUT			
dx	LREAL		X 轴目标位置
dy	LREAL		Y 轴目标位置
dr	LREAL		目标旋转位置（以度为单位）
dnx	LREAL		标准位置的 X 坐标
dny	BOOL		标准位置的 Y 坐标
ratio	LREAL		X 间隔与 Y 间隔的比例
dnOffsetX	LREAL		标准偏移量的 X 坐标

dnOffsetY	LREAL		标准偏移量的 Y 坐标

SMC_TRAFOF_GANTRYCUTTER3

它将解决指向当前轨迹的切线方向的旋转轴的三维龙门系统的前向转变。

备注：有关转换模块的详细资料，请参考 SoftMotion Transformations。

组件	类型	初始值	描述
VAR_IN_OUT			
DriveX	AXIS_REF_SM3		X 轴
DriveY	AXIS_REF_SM3		Y 轴
DriveZ	AXIS_REF_SM3		Z 轴
DriveR	AXIS_REF_SM3		旋转轴
VAR_IN			
dOffsetX	LREAL		X 轴的偏移量
dOffsetY	LREAL		Y 轴的偏移量
dOffsetZ	LREAL		Z 轴的偏移量
dOffsetR	LREAL		旋转轴的偏移量
iDirectionR	INT	1	旋转位置
minX	LREAL		X 轴方向的移动下限
maxX	LREAL		X 轴方向的移动上限
minY	LREAL		Y 轴方向的移动下限
maxY	LREAL		Y 轴方向的移动上限
VAR_OUT			
dx	LREAL		GEO 坐标系中 X 的位置。
dy	LREAL		GEO 坐标系中 Y 的位置。
dz	LREAL		GEO 坐标系中 Z 的位置。
dr	LREAL		旋转的目标位置（单位为度）
dnx	LREAL		标准位置的 X 分量 (大小介于 [0,1])，用于可视化
dny	BOOL		标准位置的 Y 分量 (大小介于 [0,1])，用于可视化
ratio	LREAL		X 间隔与 Y 间隔的比例
dnOffsetX	LREAL		标准位置的 X 偏移量(大小介于 [0,1])，用于可视化
dnOffsetY	LREAL		标准位置的 Y 偏移量(大小介于 [0,1])，用于可视化

SMC_TRAFOF_GANTRYH2

这个转变模块用来提供解决有着固定的驱动器的 H 龙门系统的前向转变。

备注：有关转换模块的详细资料，请参考 SoftMotion Transformations。

组件	类型	初始值	描述
VAR_IN_OUT			
DriveA	AXIS_REF_SM3		参考轴 A 的位置
DriveB	AXIS_REF_SM3		参考轴 B 的位置
VAR_IN			
dOffsetX	LREAL		X 轴的偏移量

dOffsetY	LREAL		Y 轴的偏移量
minX	LREAL		X 轴方向的移动下限（用于可视化）
maxX	LREAL		X 轴方向的移动上限（用于可视化）
minY	LREAL		Y 轴方向的移动下限（用于可视化）
maxY	LREAL		Y 轴方向的移动上限（用于可视化）
VAR_OUT			
dx	LREAL		GEO 坐标系中 X 的位置。
dy	LREAL		GEO 坐标系中 Y 的位置。
dnx	LREAL		标准位置的 X 分量 (大小介于 [0,1])，用于可视化
dny	BOOL		标准位置的 Y 分量 (大小介于 [0,1])，用于可视化
ratio	LREAL		X 间隔与 Y 间隔的比例（用于可视化）
dnOffsetX	LREAL		标准位置的 X 偏移量 (大小介于 [0,1])，用于可视化
dnOffsetY	LREAL		标准位置的 Y 偏移量 (大小介于 [0,1])，用于可视化

SMC_TRAFOV_GANTRY2

反向转换模块被用来提供处理考虑轨迹速度轨迹方向作为轴的控制变量的二维龙门系统。

备注：有关转换模块的详细资料，请参考 SoftMotion Transformations。

组件	类型	初始值	描述
VAR_IN			
pi	SMC_POSINFO		目标位置向量 (x,y)，插补器的输出
dVel	LREAL		轨迹速度，插补器的输出
v	SMC_VECTOR3D		轨迹切线方向向量，插补器的输出
dOffsetX	LREAL		X 轴的附加偏移量
dOffsetY	LREAL		Y 轴的附加偏移量
VAR_OUT			
dx	LREAL		X 坐标的目标位置
dvx	LREAL		X 坐标的目标速度
dy	LREAL		Y 坐标的目标位置
dvy	LREAL		Y 坐标的目标速度

SMC_TRAFOV_GANTRY3

反向转换模块被用来提供处理考虑轨迹速度轨迹方向作为轴的控制变量的三维龙门系统。

备注：有关转换模块的详细资料，请参考 SoftMotion Transformations。

组件	类型	初始值	描述
VAR_IN			
pi	SMC_POSINFO		目标位置向量 (x,y)，插补器的输出
dVel	LREAL		轨迹速度，插补器的输出
v	SMC_VECTOR3D		轨迹切线方向向量，插补器的输出
dOffsetX	LREAL		X 轴的附加偏移量
dOffsetY	LREAL		Y 轴的附加偏移量
dOffsetZ	LREAL		Z 轴的附加偏移量

VAR_OUT			
dx	LREAL		X 轴目标位置
dvx	LREAL		X 轴目标速度
dy	LREAL		Y 轴目标位置
dvy	LREAL		Y 轴目标速度
dz	LREAL		Z 轴目标位置
dvz	LREAL		Z 轴目标速度

SMC_TRAFOV_GANTRYCUTTER2

反向转换模块被用来处理考虑轨迹速度轨迹方向作为轴的控制变量的二维龙门系统。

备注：有关转换模块的详细资料，请参考 SoftMotion Transformations。

组件	类型	初始值	描述
VAR_IN			
pi	SMC_POSINFO		目标位置向量 (x,y)，插补器的输出
dVel	LREAL		轨迹速度，插补器的输出
v	SMC_VECTOR3D		轨迹切线方向向量，插补器的输出
dOffsetX	LREAL		X 轴的附加偏移量
dOffsetY	LREAL		Y 轴的附加偏移量
dOffsetR	LREAL		旋转轴偏移量
iDirectionR		1	旋转方向
VAR_OUT			
dx	LREAL	0	X 轴的目标位置
dvx	LREAL	0	X 轴的目标速度
dy	LREAL	0	Y 轴的目标位置
dvy	LREAL	0	Y 轴的目标速度
dr	LREAL	0	目标旋转位置（以度为单位）

SMC_TRAFOV_GANTRYCUTTER3

反向转换模块被用来提供处理考虑轨迹速度轨迹方向作为轴的控制变量的三维龙门系统。

备注：有关转换模块的详细资料，请参考 SoftMotion Transformations。

组件	类型	初始值	描述
VAR_IN			
pi	SMC_POSINFO		目标位置向量 (x,y)，插补器的输出
dVel	LREAL		轨迹速度，插补器的输出
v	SMC_VECTOR3D		轨迹切线方向向量，插补器的输出
dOffsetX	LREAL		X 轴的附加偏移量
dOffsetY	LREAL		Y 轴的附加偏移量
dOffsetZ	LREAL		Z 轴的附加偏移量
dOffsetR	LREAL		旋转轴偏移量
iDirectionR		1	旋转方向
VAR_OUT			
dx	LREAL	0	X 轴的目标位置
dvx	LREAL	0	X 轴的目标速度

dy	LREAL	0	Y 轴的目标位置
dvy	LREAL	0	Y 轴的目标速度
dz	LREAL	0	Z 轴的目标位置
dvz	LREAL	0	Z 轴的目标速度
dr	LREAL	0	目标旋转位置（以度为单位）

SMC_TRAFOV_GANTRYH2

反向转换模块被用来提供处理考虑轨迹速度轨迹方向作为轴的控制变量的二维 H 龙门系统。

备注：有关转换模块的详细资料，请参考 SoftMotion Transformations。

组件	类型	初始值	描述
VAR_IN			
pi	SMC_POSINFO		目标位置向量 (x,y)，插补器的输出
dVel	LREAL		轨迹速度，插补器的输出
v	SMC_VECTOR3D		轨迹切线方向向量，插补器的输出
dOffsetX	LREAL		X 轴的附加偏移量
dOffsetY	LREAL		Y 轴的附加偏移量
VAR_OUT			
da	LREAL		a 轴的目标位置
dva	LREAL		a 轴的目标速度
db	LREAL		b 轴的目标位置
dvb	LREAL		b 轴的目标速度

Parallel Systems 文件夹内

SMC_TRAFO_BIPOD_ARM

转换模块用来提供处理双脚架臂的后向转换。

备注：有关转换模块的详细资料，请参考 SoftMotion Transformations。

组件	类型	初始值	描述
VAR_IN			
pi	SMC_POSINFO		目标位置向量 (x,y)，插补器的输出
dArmLength1	LREAL		第1臂连接到电机的长度
dArmLength2	LREAL		第2臂到第1臂相关连接的长度。
dDistance	LREAL		两个电机的水平距离
dOffsetA	LREAL	0	A 轴附加偏移量
dOffsetB	LREAL	0	B 轴附加偏移量
VAR_OUT			
bError	BOOL		若位置无效，则为 TRUE
dA	LREAL		轴 A 的目标位置（以度为单位）
dB	LREAL		轴 B 的目标位置（以度为单位）

SMC_TRAFO_TRIPOD

转换模块用来提供处理三脚架的后向转换。

备注：有关转换模块的详细资料，请参考 SoftMotion Transformations。

组件	类型	初始值	描述
VAR_IN			
pi	SMC_POSINFO		环的目标位置向量 (x,y) ，插补器的输出。
dInnerRadius	LREAL		内环半径
dOuterRadius	LREAL		外环半径
dLength	LREAL		连接的长度
dDistance	LREAL		两个联动装置连接的距离。
dRotationOffest	LREAL	0	在 A 轴和原点(0,0)中间的精确的导向角度。
dOffsetA	LREAL	0	A 轴附加偏移量
dOffsetB	LREAL	0	B 轴附加偏移量
dOffsetC	LREAL	0	C 轴附加偏移量
VAR_OUT			
bError	BOOL		若位置无效，则为 TRUE
dA	LREAL		A 轴位置（以度为单位）
dB	LREAL		B 轴位置（以度为单位）
dC	LREAL		C 轴位置（以度为单位）

SMC_TRAFO_TRIPOD_ARM

转换模块用来提供处理三脚架臂的后向转换。

备注：有关转换模块的详细资料，请参考 SoftMotion Transformations。

组件	类型	初始值	描述
VAR_IN			
pi	SMC_POSINFO		目标位置向量 (x,y)，插补器的输出
dArmLength1	LREAL		第1臂连接到电机的长度
dArmLength2	LREAL		第2臂到铰链的长度
dArm1Radius	LREAL		第1臂的半径
dStewartRadius	LREAL	0	铰链的半径
dDistance	LREAL		两个联动装置之间的距离
dRotationOffest	LREAL	0	在 A 轴和原点（0,0）中间的精确导向角度。
dOffsetA	LREAL	0	A 轴附加偏移量
dOffsetB	LREAL	0	B 轴附加偏移量
dOffsetC	LREAL	0	C 轴附加偏移量
dMaxAngleBallJoint	LREAL		球关节的最大正/负角度。
VAR_OUT			
bError	BOOL		若位置无效时，则为 TRUE。
dA	LREAL		A 轴的位置（以度为单位）
dB	LREAL		B 轴的位置（以度为单位）
dC	LREAL		C 轴的位置（以度为单位）

SMC_TRAFOF_BIPOD_ARM

转换模块用来提供处理双脚架臂的前向转换。

备注：有关转换模块的详细资料，请参考 SoftMotion Transformations。

组件	类型	初始值	描述

VAR_IN_OUT			
DriveA	AXIS_REF		参考轴 A.
DriveB	AXIS_REF		参考轴 B.
VAR_IN			
dArmLength1	LREAL		第1臂连接到电机的长度
dArmLength2	LREAL		第2臂到第1臂相关连接的长度
dDistance	LREAL		两个电机之间的距离。
dOffsetA	LREAL	0	A 轴附加偏移量
dOffsetB	LREAL	0	B 轴附加偏移量
VAR_OUT			
bError	BOOL		若位置无效，则为 TRUE。
dx	LREAL		工作台的 X 位置
dy	LREAL		工作台的 Y 位置

SMC_TRAFOF_TRIPOD

转换模块用来提供处理三脚架的前向转换。

备注：有关转换模块的详细资料，请参考 SoftMotion Transformations。

组件	类型	初始值	描述
VAR_IN_OUT			
DriveA	AXIS_REF		参考轴 A
DriveB	AXIS_REF		参考轴 B
DriveC	AXIS_REF		参考轴 C
VAR_IN			
dInnerRadius	LREAL		内环半径
dOuterRadius	LREAL		外环半径
dLength	LREAL		联结/支撑的长度
dDistance	LREAL		内外环半径上的两个合作机构直接连接的距离。
dRotationOffest	LREAL	0	在 A 轴和原点（0,0）中间的精确的导向角度
dOffsetA	LREAL	0	A 轴的附加偏移量
dOffsetB	LREAL	0	B 轴的附加偏移量
dOffsetC	LREAL	0	C 轴的附加偏移量
VAR_OUT			
bError	BOOL		若位置无效，则为 TRUE。
dx	LREAL		GEO 坐标系中内环中心的 X 位置
dy	LREAL		GEO 坐标系中内环中心的 Y 位置
dz	LREAL		GEO 坐标系中内环中心的 Z 位置
dnx	LREAL		工作台标准位置的 X 坐标（用于可视化）
dny	LREAL		工作台标准位置的 Y 坐标（用于可视化）
dRatioInnerOuter	LREAL		内圈半径到外圈半径的定量角度（用于可视化）
adnxi	ARRAY [0..5] OF LREAL		杆的标准起始位置的 X 坐标（用于可视化）
adnyi	ARRAY [0..5] OF LREAL		杆的标准起始位置的 Y 坐标（用于可视化）
adnxo	ARRAY [0..5]		杆的标准结束位置的 X 坐标（用于可视化）

	OF LREAL		
adnyo	ARRAY [0..5] OF LREAL		杆的标准结束位置的 Y 坐标（用于可视化）

SMC_TRAFOF_TRIPOD_ARM

转换模块用来提供处理三脚架臂的前向转换。

备注：有关转换模块的详细资料，请参考 SoftMotion Transformations。

组件	类型	初始值	描述
VAR_IN_OUT			
DriveA	AXIS_REF		参考轴 A.
DriveB	AXIS_REF		参考轴 B.
DriveC	AXIS_REF		参考轴 C.
VAR_IN			
pi	SMC_POSINFO		目标位置向量 (x,y)，插补器的输出
dArmLength1	LREAL		第1臂连接到电机的长度
dArmLength2	LREAL		第2臂到铰链的长度。
dArm1Radius	LREAL		圆周半径通过所有的三个电机轴
dStewartRadius	LREAL	0	铰链的半径
dDistance	LREAL		内外环半径上两个合作机构之间的连接的距离。
dRotationOffest	LREAL	0	A 轴与原点（0,0）中间的精确的导向角度
dOffsetA	LREAL	0	A 轴的附加偏移量
dOffsetB	LREAL	0	B 轴的附加偏移量
dOffsetC	LREAL	0	C 轴的附加偏移量
dMaxAngleBallJoint	LREAL		球关节的最大正/负角度
VAR_OUT			
dx	LREAL		GEO 坐标系中内环/工作台中心的 X 位置
dy	LREAL		GEO 坐标系中内环/工作台中心的 Y 位置
dz	LREAL		GEO 坐标系中内环/工作台中心的 Z 位置

Scara Systems 文件夹内

SMC_TRAFO_SCARA2

转换模块用来提供处理第二关节 Scara 系统的后向转换。

备注：有关转换模块的详细资料，请参考 SoftMotion Transformations。

组件	类型	初始值	描述
VAR_IN			
pi	SMC_POSINFO		目标位置向量 (x,y)，插补器的输出
dOffsetA	LREAL		A 轴附加偏移量（以度为单位）
dOffsetB	LREAL		B 轴附加偏移量（以度为单位）
dArmLength1	LREAL		第1臂的长度
dArmLength2	LREAL		第2臂的长度
bElbowLow	BOOL		肘关节向下（TRUE）或肘关节向上（FALSE）

VAR_OUT			
bError	BOOL		若位置无效，则为 TRUE。
dA	LREAL		A 轴的位置（以度为单位）
dB	LREAL		B 轴的位置（以度为单位）

SMC_TRAFO_SCARA3

转换模块用来提供处理第三关节 Scara 系统的后向转换。

备注：有关转换模块的详细资料，请参考 SoftMotion Transformations。

组件	类型	初始值	描述
VAR_IN			
pi	SMC_POSINFO		目标位置向量 (x,y). 插补的输出
dDirection	LREAL		最后一个关节的方向角（以度为单位），(0度为W，90度)
dOffsetA	LREAL		A 轴附加偏移量（以度为单位）
dOffsetB	LREAL		B 轴附加偏移量（以度为单位）
dOffsetC			C 轴附加偏移量（以度为单位）
dArmLength1	LREAL		第1臂的长度
dArmLength2	LREAL		第2臂的长度
dArmLength3	LREAL		第3臂的长度
bElbowLow	BOOL		肘部（第1和第2关节）向下（TRUE）或向下（FALSE）
VAR_OUT			
bError	BOOL		若位置无效，则为 TRUE
dA	LREAL		A 轴的位置（以度为单位）
dB	LREAL		B 轴的位置（以度为单位）
dC	LREAL		C 轴的位置（以度为单位）

SMC_TRAFOF_SCARA2

转换模块用来提供处理第二关节 Scara 系统的前向转换。

备注：有关转换模块的详细资料，请参考 SoftMotion Transformations。

组件	类型	初始值	描述
VAR_IN_OUT			
DriveA	AXIS_REF		参考轴 A.
DriveB	AXIS_REF		参考轴 B.
VAR_IN			
dOffsetA	LREAL	0	A 轴的附加偏移量（等同于 SMC_TRAFO_SCARA2）
dOffsetB	LREAL	0	B 轴的附加偏移量（等同于 SMC_TRAFO_SCARA2）
dArmLength1	LREAL		第1臂的长度
dArmLength2	LREAL		第2臂的长度
VAR_OUT			
bError	BOOL		若位置无效，则为 TRUE。
dx	LREAL		GEO 坐标系中 X 的位置的工作台
dy	LREAL		GEO 坐标系中 Y 的位置的工作台

dAlpha	LREAL		关节角度（无偏移量时的轴位置，以度为单位），用于可视化。
dBeta	LREAL		关节角度（无偏移量时的轴位置，以度为单位），用于可视化。
dpx	LREAL		第1关节的标准位置向量的 X 坐标[-1,1]，用于可视化
dpy	LREAL		第1关节的标准位置向量的 Y 坐标[-1,1]，用于可视化
dnx	LREAL		工作台标准位置向量的 X 坐标[-1,1]，用于可视化
dny	LREAL		工作台标准位置向量的 Y 坐标[-1,1]，用于可视化
dR1	LREAL		相对臂长（用于可视化）
dR2	LREAL		相对臂长（用于可视化）

SMC_TRAFOF_SCARA3

转换模块用来提供处理第三关节 Scara 系统的前向转换。

备注：有关转换模块的详细资料，请参考 SoftMotion Transformations。

组件	类型	初始值	描述
VAR_IN_OUT			
DriveA	AXIS_REF		参考轴 A
DriveB	AXIS_REF		参考轴 B
DriveC	AXIS_REF		参考轴 C
VAR_IN			
dOffsetA	LREAL	0	A 轴的附加偏移量（以度为单位）（等同于 SMC_ TRAFO_SCARA3)
dOffsetB	LREAL	0	B 轴的附加偏移量（以度为单位）（等同于 SMC_ TRAFO_SCARA3)
dOffsetC	LREAL		C 轴的附加偏移量（以度为单位）（等同于 SMC_ TRAFO_SCARA3)
dArmLength1	LREAL		第1臂的长度
dArmLength2	LREAL		第2臂的长度
dArmLength3	LREAL		第3臂的长度
VAR_OUT			
bError	BOOL		若位置无效，则为 TRUE。
dx	LREAL		GEO 坐标系中位置的 X 分量
dy	LREAL		GEO 坐标系中位置的 Y 分量
dAlpha	LREAL		关节角度（无偏移量时的轴位置，以度为单位），用于可视化。
dBeta	LREAL		关节角度（无偏移量时的轴位置，以度为单位），用于可视化。
dGamma	LREAL		关节角度（无偏移量时的轴位置，以度为单位），用于可视化。
dpx	LREAL		第1关节的标准位置向量的 X 坐标（-1,1）（用于可视化）
dpy	LREAL		第1关节的标准位置向量的 Y 坐标（-1,1）（用于可视化）
dppx	LREAL		第2关节的标准位置向量的 X 坐标（-1,1）（用于可视

			化）
dppy	LREAL		第2关节的标准位置向量的 Y 坐标（-1,1）（用于可视化）
dnx	LREAL		工作台的标准位置向量的 X 坐标（-1,1）（用于可视化）
dny	LREAL		工作台的标准位置向量的 Y 坐标（-1,1）（用于可视化）
dR1	LREAL		相对臂长（用于可视化）
dR2	LREAL		相对臂长（用于可视化）
dR3	LREAL		相对臂长（用于可视化）

台安科技（无锡）有限公司
江苏省无锡市新吴区高浪东路29号
0510-85227555
www.taian-technology.com