

BECKHOFF 自动化新技术



TwinCAT PLC 编程手册

德国倍福电气有限公司

2005 年 5 月

目录

1	TwinCAT PLC Control 简介.....	1
1.1	引言.....	1
1.2	项目组件.....	3
1.3	语言.....	10
1.3.1	编程语言.....	10
1.3.2	指令表 (IL).....	10
1.3.3	结构化文本 (ST).....	12
1.3.4	顺序功能图 (SFC).....	17
1.3.5	功能块图 (FBD).....	21
1.3.6	连续功能图编辑器 (CFC).....	21
1.3.6	梯形图 (LD).....	22
1.4	调试, 联机功能.....	24
1.5	IEC 61131-3.....	26
2	样例程序.....	27
2.1	样例程序.....	27
3	子项组件.....	36
3.1	主窗口.....	36
3.2	选项.....	38
3.3	项目管理.....	53
3.4	对象.....	74
3.5	编辑功能.....	87
3.6	联机功能.....	92
3.7	窗口.....	101
3.8	帮助系统.....	102
3.9	日志.....	103
4	编辑器.....	105
4.1	概述.....	105
4.2	声明编辑器.....	107
4.3	文本编辑器.....	118
4.4	指令表编辑器.....	122
4.5	结构化文本编辑器.....	124
4.6	图形编辑器.....	125
4.7	功能块图编辑器.....	127

4.8	梯形图编辑器.....	132
4.9	连续功能图编辑器.....	137
4.10	顺序功能图编辑器.....	150
5	资源.....	159
5.1	概述.....	159
5.2	全局变量.....	160
5.3	PLC 配置.....	163
5.4	任务配置.....	164
5.5	抽样跟踪.....	167
5.6	监视和接收管理器.....	172
6	库管理.....	175
6.1	库管理器.....	175
7	工程接口 (ENI).....	177
7.1	概述.....	177
8	附录.....	180
8.1	数据类型.....	180
8.1.1	概述.....	180
8.1.2	标准数据类型.....	181
8.1.2.1	BOOL.....	181
8.1.2.2	BYTE.....	181
8.1.2.3	WORD.....	182
8.1.2.4	DWORD.....	182
8.1.2.5	SINT.....	182
8.1.2.6	USINT.....	182
8.1.2.7	INT.....	182
8.1.2.8	UINT.....	182
8.1.2.9	DINT.....	183
8.1.2.10	UDINT.....	183
8.1.2.11	REAL.....	183
8.1.2.12	LREAL.....	183
8.1.2.13	STRING.....	183
8.1.2.14	TIME.....	184
8.1.2.15	TIME_OF_DAY.....	184
8.1.2.16	DATE.....	184
8.1.2.17	DATE_AND_TIME.....	184
8.1.3	用户数据类型.....	185
8.1.3.1	数组.....	185
8.1.3.2	指针.....	186
8.1.3.3	枚举 (ENUM).....	186

8.1.3.4	结构 (STRUCT)	187
8.1.3.5	参考类型 (别名)	188
8.1.3.6	子范围类型	189
8.2	操作符	190
8.2.1	概述	190
8.2.2	IEC 操作符概述	190
8.2.3	数值操作符	193
8.2.3.1	ABS	193
8.2.3.2	ACOS	193
8.2.3.3	ASIN	194
8.2.3.4	ATAN	194
8.2.3.5	COS	194
8.2.3.6	EXP	194
8.2.3.7	EXPT	194
8.2.3.8	LN	195
8.2.3.9	LOG	195
8.2.3.10	SIN	195
8.2.3.11	SQRT	195
8.2.3.12	TAN	195
8.2.4	算术操作符	197
8.2.4.1	ADD	197
8.2.4.2	MUL	197
8.2.4.3	SUB	197
8.2.4.4	DIV	198
8.2.4.5	MOD	198
8.2.5	位串操作符	199
8.2.5.1	AND	199
8.2.5.2	OR	199
8.2.5.3	XOR	199
8.2.5.4	NOT	200
8.2.6	位移操作符	200
8.2.6.1	SHL	200
8.2.6.2	SHR	201
8.2.6.3	ROL	202
8.2.6.4	ROR	202
8.2.7	选择操作符	204
8.2.7.1	SEL	204
8.2.7.2	MAX	204
8.2.7.3	MIN	205
8.2.7.4	LIMIT	205
8.2.7.5	MUX	205
8.2.8	比较操作符	207
8.2.8.1	GT	207
8.2.8.2	LT	207

8.2.8.3	LE	207
8.2.8.4	GE	208
8.2.8.5	EQ	208
8.2.8.6	NE.....	209
8.2.9	选择不同的操作符	210
8.2.9.1	INDEXOF.....	210
8.2.9.2	SIZEOF.....	210
8.2.9.3	ADR（地址操作符）	210
8.2.9.4	^（内容操作符）	210
8.2.9.5	CAL（调用操作符）	210
8.2.9.6	BITADR.....	211
8.2.10	类型转换操作符.....	212
8.2.10.1	BOOL_TO 转换	212
8.2.10.2	TO_BOOL 转换	212
8.2.10.3	STRING_TO 转换	212
8.2.10.4	TO_STRING 转换	212
8.2.10.5	TIME_TO 转换	213
8.2.10.6	DATE_TO 转换.....	213
8.2.10.7	TOD_TO 转换	213
8.2.10.8	“DT_TO 转换.....	213
8.2.10.9	REAL_TO-/LREAL_TO 转换	214
8.2.10.10	整型数类型之间的转换.....	214
8.2.10.11	TRUNC	214
8.3	操作数	215
8.3.1	常数.....	215
8.3.1.1	BOOL 常数	215
8.3.1.2	TIME 常数	215
8.3.1.3	DATE 常数	215
8.3.1.4	TIME_OF_DAY 常数	215
8.3.1.5	DATE_AND_TIME 常数	216
8.3.1.6	数值常数.....	216
8.3.1.7	REAL/LREAL 常数	216
8.3.1.8	STRING 常数	216
8.3.1.9	类型符.....	217
8.3.2	变量.....	218
8.3.2.1	变量.....	218
8.3.2.2	地址.....	218
8.3.2.3	存取数组、结构和 POU 变量.....	219
8.3.2.4	变量的位寻址	219
8.3.2.5	功能.....	219
8.3.2.6	系统标志	219
8.3.2.6.1	概述	219
8.3.2.6.2	SYSTEMINFO.....	220
8.3.2.6.3	SYSTEMTASKINFOARR.....	220

8.4	系统功能.....	221
8.4.1	CheckBounds 功能.....	221
8.4.2	CheckDivByte 功能.....	222
8.4.3	CheckDivReal 功能.....	222
8.4.4	CheckDivWord 功能.....	223
8.4.5	CheckDivDWord 功能.....	223
8.4.6	CheckRangeSigned 功能.....	224
8.4.7	CheckRangeUnsigned 功能.....	225
8.5	使用键盘.....	227
8.6	创建错误表.....	231
8.7	命令行命令.....	246

1 TwinCAT PLC Control 简介

1.1 引言

什么是 TwinCAT PLC Control?

TwinCAT PLC Control 是为 PLC 设计的一种完整的开发环境。TwinCAT PLC Control 为 PLC 编程提供了一种简便的方法，可以自由地处理功能强大的 IEC 语言。编辑器和调试功能的使用则基于先进编程语言和已验证的程序开发环境。

TwinCAT PLC Control 概述

一个项目是如何构成的?

一个项目置于一个在项目后期命名的文件内。首先打开一个默认的“Task Configuration（任务配置）”。任务的名字是“Standard（标准）”。在一个新项目中建立的第一个 POU（程序组织单元）将自动地命名为“MAIN（主程序）”。你可以在任务配置中重新命名这个 POU。TwinCAT PLC Control 能够区分一个项目中不同类型的对象：POU，数据类型和资源。对象管理器（Object Organizer）包括一个表，该表列出了项目中所有的对象。

怎样建立项目?

首先，你应该选择目标系统。然后配置任务。你可以建立解决你的问题所需要的 POU。现在你可以应用所期望的编程语言对你所需要的 POU 进行编程。一旦编程完成，你就可以编译该项目，并修改可能存在的任何错误。

如何测试项目?

一旦已经修改完成所有的错误，联机 PLC 并将你的项目“下载”到 PLC 内。现在，TwinCAT PLC Control 处于“联机”模式。测试你的项目是否为正确顺序。为此，手动设置输入变量并观察输出是否为期望值。你还可以观察 POU 中本地变量的数值顺序。在“Watch and Receipt Manager（监视和接收管理器）”中，你可以配置你想要检查的数据记录。

当程序出现错误时，你可以设置断点。如果程序在该断点处停止，你可以及时在该断点处检查项目所有的变量值。并按照顺序操作（单步），检查程序的逻辑正确性。

TwinCAT PLC Control 还有另外一种调试功能：你可以设置程序变量以及输入和输出为某些值。你可以使用流控制来检查哪些程序行已经运行。使用“Sampling Trace（抽样跟踪）”，可以在扩展的时间范围内跟踪和显示变量的实际变化过程。

“Log（日志）”则按时间顺序记录在联机对话期间的操作、用户采取的动作和各种内部过程。

整个项目可随时进行文档化或输出到一个文本文件。

其它功能

整个项目可随时进行文档化或输出到一个文本文件。也可将它翻译成其它语言。

ENI: 通过 ENI 接口（“工程接口”），可以将编程系统连接到一个外部数据库。在那里可以存储创建一个自动化项目过程中所需要的各种数据。外部数据库的使用保证了数据的一致性，这些数据以后可以被多个用户、项目和程序所共享。

小结

TwinCAT PLC Control 是一个完整的开发工具，用来对你的 PLC 进行编程，它能大量节省建立应用程序所需要的时间。

1.2 项目组件

一个项目包含一个 PLC 程序中的所有对象。项目保存在项目后期命名的一个文件内。一个项目包括以下对象：

POU（程序组织单元），数据类型，资源和软件库。

POU（程序组织单元）

功能、功能块和程序都是 POU，它们可以使用动作（Action）加以补充。

每个 POU 都由一个声明部分和一个程序本体组成。程序由 IEC 编程语言中的一种语言编写（这些语言包括 IL、ST、SFC、FBD、LD 或 CFC）。TwinCAT PLC Control 支持所有 IEC 的标准 POU。如果你需要在项目中使用这些标准 POU，则必须在你的项目中包括标准库（standard.lib）。

一个 POU 可调用其它 POU。然而不允许进行递归调用。

功能

一个功能即是一个 POU，当对它进行处理时，它可准确地生成数据元素（诸如由几个字段或结构等元素组成），并在文本语言调用时作为表达式内的一个操作符出现。

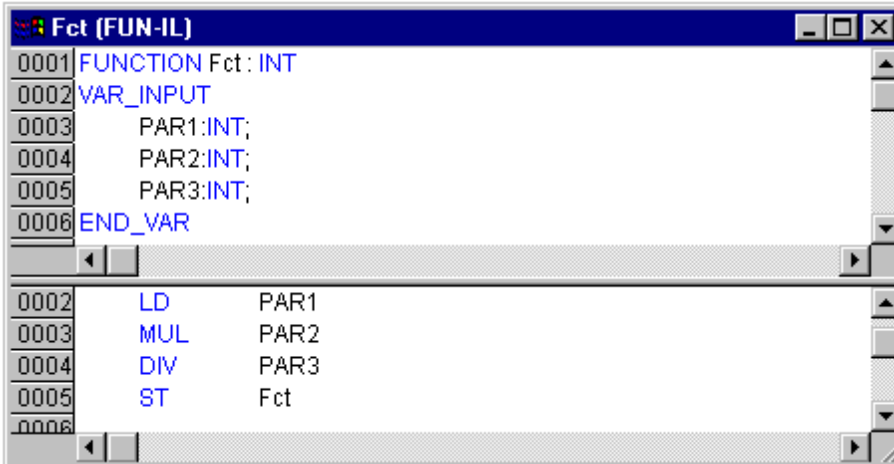
当声明一个功能时，不要忘记功能必须返回一个类型。这意味着，在功能名后，你必须输入一个冒号，后跟一个类型。

一个正确的功能声明有如下示例那样的形式：

```
FUNCTION Fct:INT
```

此外，必须对功能分配一个结果。这意味着，功能名作为一种输出变量使用。

以下为使用 IL（指令表）编写的一个功能例子，它有三个输入变量，并返回由前二个变量乘积除以第三个变量的结果：



```
Fct (FUN-IL)
0001 FUNCTION Fct:INT
0002 VAR_INPUT
0003     PAR1:INT;
0004     PAR2:INT;
0005     PAR3:INT;
0006 END_VAR
0002 LD     PAR1
0003 MUL    PAR2
0004 DIV    PAR3
0005 ST     Fct
0006
```

由 ST（结构化文本）编写的一个功能调用可作为表达式中的一个操作数。功能没有任何内部条件。这意味着，由相同变元（输入参数）调用的一个功能总是产生相同的值（输出）。

注意：如果你在一个功能内声明一个保持型的本地变量，这是没有意义的。变量不会保存到保留区！

调用上述功能的示例：

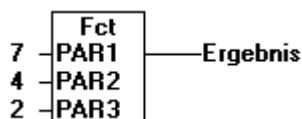
使用 IL：

```
LD 7 Fct 2,4 ST Result
```

使用 ST 语言:

```
Result := Fct(7, 2, 4);
```

使用 FBD:

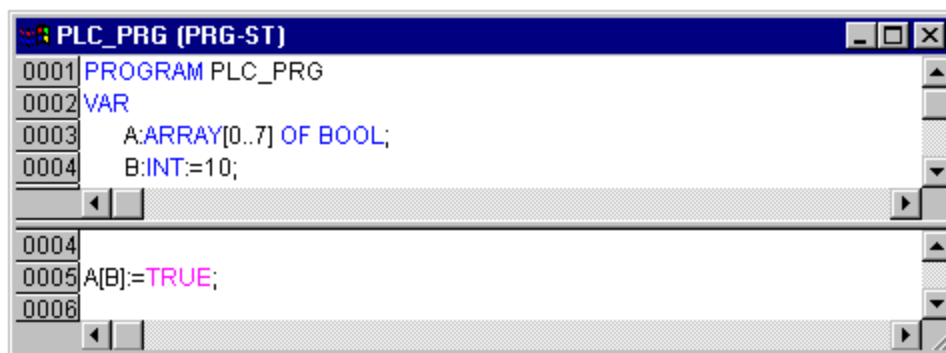


使用 SFC 时，一个功能调用只出现在一个步或一个转换内。

注意:

如果在项目中应用名称“CheckBounds”来定义一个功能，你可以使用这个名字来检查项目中是否有范围溢出！该功能名是系统预定义的，而且系统只识别这个标识符。

下面的典型程序是使用 CheckBounds 功能测试超出一个定义数组的边界。CheckBounds 功能可以确认 TRUE 值没有分配给 A[10]，而是分配给上边界 A[7]，在这一位置，A[7] 仍然有效。因此，Check Bounds 功能可以用来检查是否超出数值边界。



注意:

如果在你的项目中，使用了 CheckDivByte、CheckDivWord、CheckDivDWord 和 CheckDivReal 定义的功能，并且使用了操作符 DIV，则它们可以用于检查除数的有效性，例如用来避免出现被 0 除。该功能名是系统预定义的，而且系统只识别这个标识符。

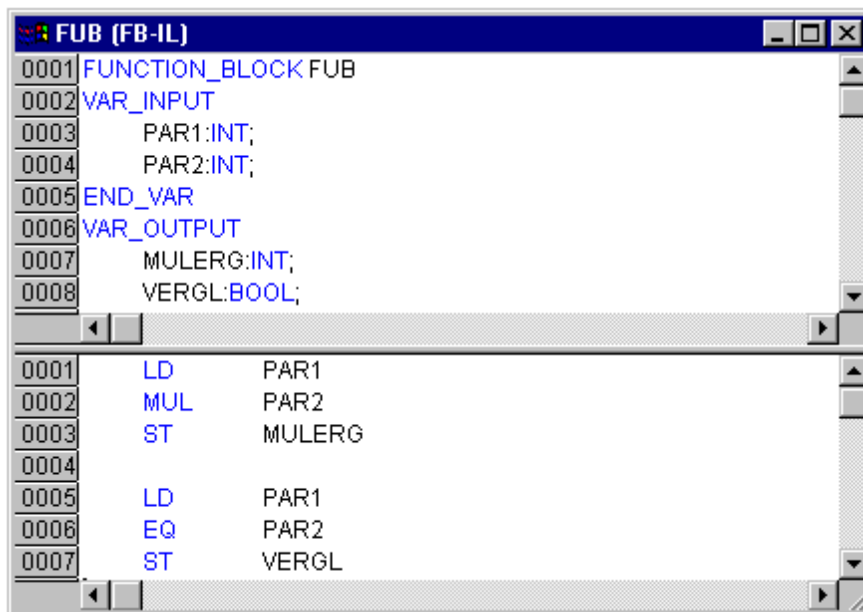
如果你使用 CheckRangeSigned 和 CheckRangeUnsigned 来定义功能，则超出由子范围类型声明的变量范围可以被截断。

所有这些检查的功能名都被系统作为应用程序的保留名。

功能块

一个功能块即是一个 POU，它提供程序运行过程中的一个或多个值。与功能相比，功能块不提供返回值。

以下是有二个输入变量和二一个输出变量的一个功能块的例子，使用 IL 语言编写。其中一个输出是二个输入的乘积，另一个输出则比较它们是否相等：



功能块实例

可以创建一个功能块的拷贝或实例（复制）。每个实例都具有其自身的标识符（实例名），以及包含有输入、输出和内部变量的数据结构。实例可以作为本地或全局变量加以声明，而功能块名是作为标识符类型予以指示的。

示例，名称为“INSTANCE”的 FUB 功能块的实例：

```
INSTANCE:FUB;
```

功能块总是通过上述实例调用的。

只能从一个功能块实例的外部存取输入和输出参数，而不能从其内部变量存取这些参数。

存取一个输入变量的示例：

功能块 FB 有一个类型 INT 的输入变量 in1。

```
PROGRAM prog
```

```
VAR
```

```
    inst1:fb;
```

```
END_VAR
```

```
    LD 17
```

```
    ST inst1.in1
```

```
    CAL inst1
```

```
END_PROGRAM
```

功能块和程序的声明部分可包含实例声明。在功能内不允许有实例声明。

对一个功能块实例的访问除非它们是全局声明的，否则只限于声明它们的 POU。一个功能块实例的实例名可用作一个功能或一个功能块的输入。

注意：

处理一个功能块后，所有的值直到下一次处理之前都加以保留。因此，通过相同变元的功能块调用并不总是返回相同的输出值！

提示：

如果至少有一个功能块变量是一个保留变量，则全部实例都存储在保留区。

调用一个功能块

通过建立一个功能块实例，并用以下语法规定所期望的变量，则可以从其它 POU 存取一个功能块的输入和输出变量。

<实例名>.<变量名>

当你打开功能块时，如果要设置输入参数（输入变量的值），你可以使用文本语言 IL 和 ST，通过对括号（位于功能块实例名后）内的参数赋值来设置输入参数（使用“:=”进行赋值，如同在声明位置的变量初始化）。

请注意，POU 的输入/输出变量 (VAR_IN_OUT) 将转变成为一个指针。因此，在调用时不能对它们赋值常数，而且不能进行外部的读或写操作。

应用 ST 语言编写的调用 POU fubo 的 VAR_IN_OUT 变量 inout1 示例：

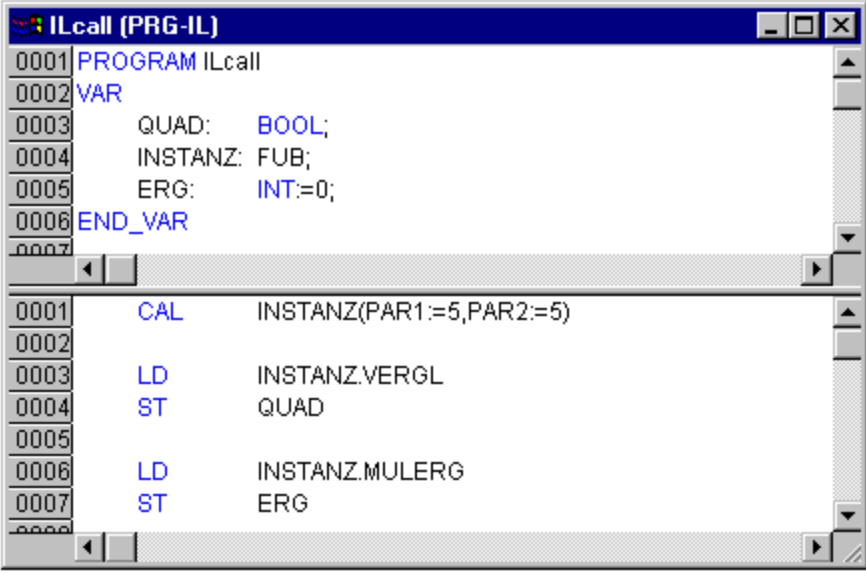
```
VAR
  inst:fubo;
  var1:int;
END_VAR

var1:=2;
inst(inout1:=var1);
```

不允许: inst(inout1:=2); 或 inst.inout1:=2;

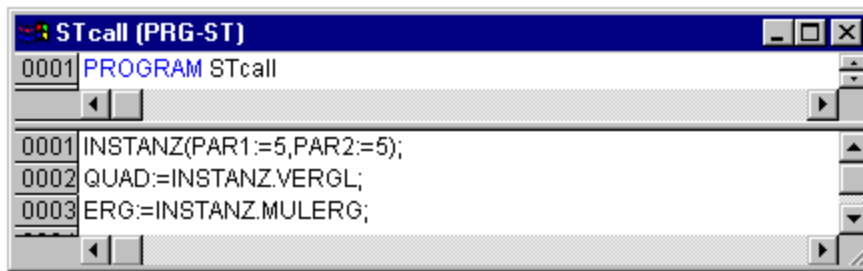
调用上述功能块 FUB 的例子。乘法结果保存在变量 ERG 内，比较结果保存在 QUAD 内。并声明功能块 FUB 的实例名为 INSTANCE:

使用 IL 语言，按以下程序调用功能块：



```
ILcall (PRG-IL)
0001 PROGRAM ILcall
0002 VAR
0003     QUAD:  BOOL;
0004     INSTANZ: FUB;
0005     ERG:   INT:=0;
0006 END_VAR
0007
0001     CAL     INSTANZ(PAR1:=5,PAR2:=5)
0002
0003     LD     INSTANZ.VERGL
0004     ST     QUAD
0005
0006     LD     INSTANZ.MULERG
0007     ST     ERG
```

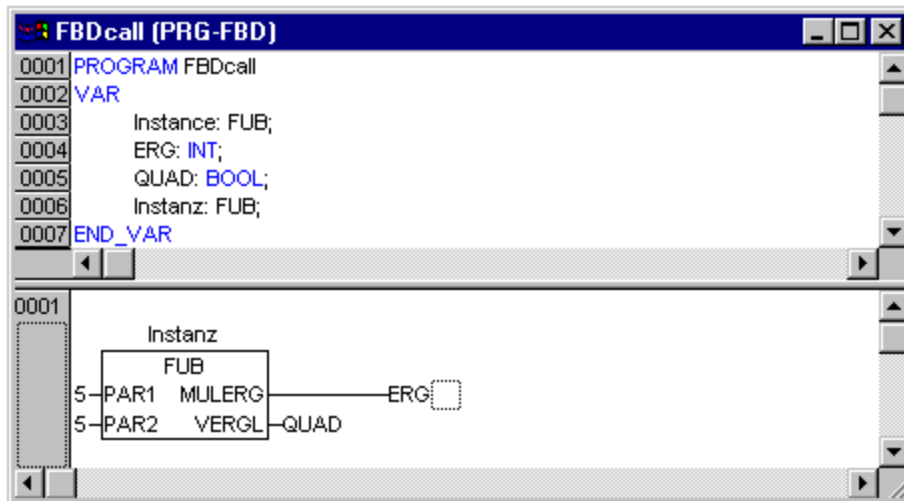
在以下例子中，使用 ST 语言表示的调用。声明部分同 IL 语言：



```

STcall (PRG-ST)
0001 PROGRAM STcall
0001 INSTANZ(PAR1:=5,PAR2:=5);
0002 QUAD:=INSTANZ.VERGL;
0003 ERG:=INSTANZ.MULERG;
  
```

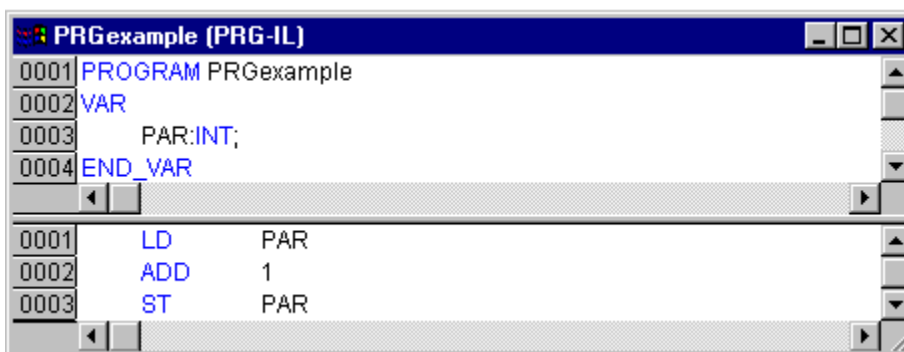
使用 FBD 语言，屏幕显示如下（声明部分同 IL 语言）：



使用 SFC 语言，只能按步调用功能块。

程序

一个程序即是一个 POU，它在运行过程中返回若干个值。并全局识别整个项目的程序。从上一次运行程序直到下一次运行程序，一直保留所有的值。



```

PRGexample (PRG-IL)
0001 PROGRAM PRGexample
0002 VAR
0003     PAR:INT;
0004 END_VAR
  
```

```

0001 LD    PAR
0002 ADD   1
0003 ST    PAR
  
```

程序可以被调用。不允许在一个功能内调用一个程序。也不存在程序的实例。如果一个 POU 调用一个程序，而且如果程序的值随之改变，那么，这些改变将被保留到下一次调用程序（即使从其它 POU 内已调用了该程序）。这不同于调用一个功能块。那里只改变功能块给定实例中的值。因此仅当调用相同实例时，这些改变才起作用。一个程序声明以关键字 PROGRAM 开始，以 END_PROGRAM 结束。

上述程序调用的示例：

使用 IL 语言：

CAL PRG Example

LD PRGexample.PAR

ST ERG

使用 ST 语言:

PRGExample;

Erg := PRGexample.PAR;

使用 FBD 语言:

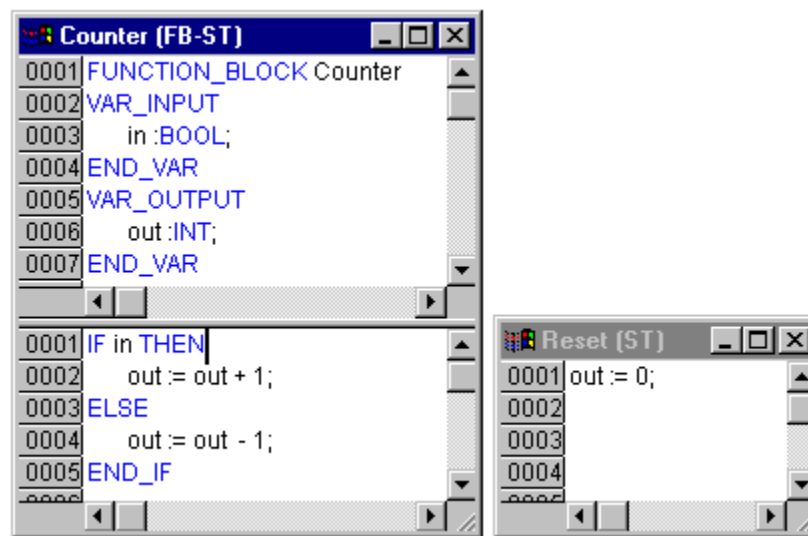


如果从程序 PRGexample 来的变量 PAR 是通过一个带有 0 的主程序初始化,然后以上述命名的程序调用逐一调用其它程序,那么,程序中的 ERG 结果将有值 1、2 和 3。如果交换调用顺序,那么,给定结果参数的值也应相应的改变。

动作

作为功能块和程序的附加特性,你可以定义各种动作(Action)。一个动作是使用步实现的。它可以使用与“normal(常规)”实现体不同的语言编写。每个动作有一个名字。

一个动作是与关联的功能块或程序的数据一起工作的。它与“常规”实现体一样,使用相同的输入/输出变量和局部变量。



参见上图中的示例:

当调用功能块 Counter 时,输出变量随输入变量 'in' 增加或减小。当调用附属于功能块的动作 Reset 时,输出变量设置为 0。在这两种情况下,编写相同的输出变量 Out。

相应于 <实例名>.<动作名>,通过 <程序名>.<动作名> 调用一个动作。如果必须在主模块内调用动作,则在文本编辑器中,只使用动作名,在图形编辑器中,功能块的调用不需要实例名。

示例

对所有示例的声明:

PROGRAM PLC_PRG

VAR

Inst : Counter;

END_VAR

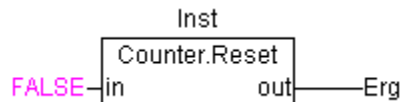
使用 IL 语言:

```
CAL Inst.Reset(In := FALSE)
LD Inst.out
ST ERG
```

使用 ST 语言:

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

使用 FBD 语言:



注意:

有关 SFC 动作的详细信息, 请参见“SFC”章节。

在 IEC 61131-3 标准中, 只描述用于 SFC 的动作。

资源

为了配置和组织项目以及跟踪变量值, 你需要使用资源:

- “全局变量 (Global Variable)”, 它可以在整个项目中使用。
- 用来配置硬件的“PLC 配置 (PLC Configuration)”。
- 引导程序执行各种任务的“任务配置 (Task Configuration)”。
- 用于变量图形化显示的“抽样跟踪 (Sampling Trace)”。
- 用来显示变量和设置默认变量的“监视和接收管理器 (Watch and Receipt Manager)”。

参见“资源”章节。

软件库

你可以在项目中包括一系列的软件库。你可以像用户定义变量那样使用这些库的 POU、数据类型以及全局变量。“standard.lib”库是程序的一个标准部分, 供你随时使用。

参见章节“库管理器”。

数据类型

除标准数据类型之外, 用户还可以定义自己的数据类型。并可以建立结构、枚举类型和子类型。

参见附录中的“标准数据类型”和“用户定义的数据类型”。

1.3 语言

1.3.1 编程语言

TwinCAT PLC Control 支持所有的 IEC61131-3 编程语言。有两种文本编程语言和三种图形编程语言。

文本编程语言

- 指令表 (IL)
- 结构化文本 (ST)

图形编程语言

- 功能块图 (FBD)
- 梯形图 (LD)
- 连续功能图 (CFC)
- 顺序功能图 (SFC)

1.3.2 指令表 (IL)

指令表 (IL) 由一系列指令组成。每个指令在一个新行的开始，包含有一个操作符以及取决于操作符类型的一个或几个以逗号分隔的操作数。在指令前面可以有标识符号 (标记)，后跟一个冒号 (:)。

注释必须是一行的最后元素。指令之间可以插入空行。

例：

```
LD 17
ST lint (* 注释 *)
GE 5
JMPC next
LD idword
EQ istruct.sdword
STN test
next:
```

使用 IL 语言的修饰符和操作符

在 IL 语言中，可以使用以下操作符和修饰符。

修饰符：

- JMP、CAL、RET 带有 C：仅当前面的表达式结果是 TRUE (真) 时，才执行指令。
- JMPC、CALC、RETC 带有 N：仅当前面的表达式结果是 FALSE (假) 时，才执行指令。
- 其它情况的 N：操作数的求反运算 (不是累加器)

以下列出所有用于 IL 语言的操作符以及可能的修饰符和操作符相应的含义：

操作符	修改符	含义
LD	N	使当前的结果等于操作数

ST	N	在操作数位置保存当前结果
S		如当前结果是 TRUE, 将布尔操作数设置为 TRUE
R		如当前结果是 TRUE, 将布尔操作数设置为 FALSE
AND	N,(按位逻辑“与”
OR	N,(按位逻辑“或”
XOR	(按位“异或”
ADD	(加法
SUB	(减法
MUL	(乘法
DIV	(除法
GT	(>
EQ	(=
NE	(<>
LE	(<=
LT	(<
JMP	CN	跳转到标号
CAL	CN	调用功能块
RET	CN	从功能块调用返回
)		后期评估操作

你可以在附录表中找到所有的 IEC 操作符。

使用修饰符的一个 IL 程序示例:

```
LD TRUE (*在累加器中装载 TRUE*)
ANDN BOOL1 (*BOOL1 变量取反进行“与”运算*)
JMPC label (*如果结果是“TRUE”，则跳转到标号“Label”*)
LDN BOOL2 (*装载 BOOL2 取反值*)
ST ERG (*在 ERG 中存储 BOOL2*)
label:

LD BOOL2 (*存储 BOOL2 的值*)
ST ERG (*在 ERG 中存储 BOOL2*)
```

使用 IL 语言还可以在操作符后放置括号。从而，括号内的值被认为是一个操作数。

例:

```
LD 2
MUL 2
ADD 3
ST Erg
```

这里, Erg 值是 7。然而, 如果加上括号:

```
LD 2
MUL( 2
ADD 3
)
ST Erg
```

这里，Erg（结果）的最终值为 10，从而，只有到达“)””，才能求值操作 MUL；这是因为操作 MUL5 是以后计算出来的。

1.3.3 结构化文本（ST）

结构化文本由一系列指令组成。可以执行如同高级语言所确定的 (“IF..THEN..ELSE”) 或循环 (WHILE..DO) 语句。

例：

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

表达式

表达式是一种结构，它返回计算后的值。表达式由操作符和操作数组成。操作数可以是常数、变量、功能调用，或其它表达式。

表达式的求值

按照优先级规则，通过对操作符的处理可执行表达式的求值。首先处理最高优先级的操作符，然后处理次优先级的操作符，依次类推，直至处理完所有的操作符为止。有相同优先级的操作符是从左到右处理的。

以下按操作符的优先级大小列出 ST 操作符表：

操作	符号	优先级
置于括号内	(表达式)	最高优先级
功能调用	功能名 (参数表)	
幂运算	**	
负值	-	
取反	NOT	
乘	*	
除	/	
取模	MOD	
加	+	
减	-	
比较	<,>,<=,>=	
等于	=	
不等于	<>	

布尔“与”	AND	
布尔“异或”	XOR	
布尔“或”	OR	最低优先级

以下是使用 ST 语言的指令，表中同时给出示例：

指令	示例
赋值	A:=B; CV := CV + 1; C:=SIN(X);
调用功能块并使用 FB 形式	CMD_TMR(IN := %IX5, PT := 300);A:=CMD_TMR.Q;
RETURN	RETURN;
IF	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	FOR I:=1 TO 100 BY 2 DO IF ARR[I] = 70 THEN J:=I; EXIT; END_IF; END_FOR;
WHILE	WHILE J<= 100 AND ARR[J] <> 70 DO J:=J+2; END_WHILE;
REPEAT	REPEAT J:=J+2; UNTIL J= 101 OR ARR[J] = 70 END_REPEAT;
EXIT	EXIT;
空指令	;

结构化文本中的指令

前面提到，结构化文本（ST）用于结构化的编程，亦即 ST 为诸如编程循环等这些常用的结构提供预先规定的结构。这样做的好处是可以降低出现差错的可能性，并提高程序的可读性。

例如，让我们比较使用 IL 和使用 ST 编写的二个相同功能的程序：

使用 IL 语言编写的计算二次方幂的一个循环：

```
Loop :
LD      Counter
EQ      0
JMPC   end
LD      Var1
MUL     2
ST      Var1
LD      Counter
SUB     1
ST      Counter
JMP     Loop
End:
LD      Var1
ST      ERG
```

使用 ST 语言编写同样的循环程序为:

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

```
Erg:=Var1;
```

可以看出, 使用 ST 编写的循环程序不仅编程速度快, 而且更容易阅读, 尤其是在大型结构中的嵌套循环时更是如此。

使用 ST 语言编写的不同结构有以下意义:

赋值操作符

在一个赋值语句的左侧有一个操作数(变量、地址), 对它赋值右侧表达式的值, 赋值操作符为 :=

示例: `Var1 := Var2 * 10;`

执行该行后, Var1 值为 10 倍的 Var2 值。

使用 ST 语言编写的功能块调用

使用 ST 语言编写的功能块调用, 采用的是调用功能块的实例名, 以及将括号内的参数赋值。在以下例子中, 通过对参数 IN 和 PT 赋值, 调用一个定时程序。然后将结果变量 Q 赋值给变量 A。

如同使用 IL 那样, 结果变量使用功能块名, 后跟一个“点(.)”以及变量名:

```
CMD_TMR(IN := %IX5, PT := T#300MS);
```

```
A:=CMD_TMR.Q
```

RETURN (返回) 指令

RETURN 指令可按条件用于结束一个功能。

IF 指令

使用 IF 指令可检查一个条件, 以及按照这个条件执行指令。

语法:

```

IF <Boolean_printout1> THEN
<IF_instructions>
{ELSIF <Boolean_printout2> THEN
<ELSIF_instructions1>
.
.
ELSIF <Boolean_printout n> THEN
<ELSIF_instructions n-1>
ELSE
<ELSE_instructions>}
END_IF;

```

括号 {} 中的部分是任选项。

如果上述指令集中 <Boolean_printout1> 返回的是 TRUE，那么只执行 <IF_Instructions>，其它指令都不执行。否则逐个计算以 <Boolean_printout2> 开始的布尔表达式，直至表达式中的一个返回 TRUE 时为止。然后，只计算这个布尔表达式后和下一个 ELSE 或 ELSIF 前的那些指令。如果没有一个布尔表达式返回 TRUE，则只计算 <ELSE_instructions>。

示例：

```

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

```

这里，当温度下降到低于 17 度时，加热装置接通。否则加热装置一直保持断开。

CASE 指令

使用 CASE 指令可在一个结构中，以相同的条件变量组合几个条件指令。

语法：

```

CASE <Var1> OF
<Value 1>: <instruction 1>
<Value 2>: <instruction 2>
...
<Value n>: <instruction n>
ELSE <ELSE-instruction>
END_CASE;

```

按以下方式处理一个 CASE 指令：

- 若 <Var1> 的变量值为 <Value i>，则执行指令 <Instruction i>。
- 若 <Var1> 没有指示值，则执行 <ELSE Instruction>。
- 若对几个变量值执行相同的指令，则可用逗号分隔并逐一写出这些值，从而规定共同执行的条件。

示例：

```
CASE INT1 OF
1, 5:   BOOL1 := TRUE;
        BOOL3 := FALSE;
2:     BOOL2 := FALSE;
        BOOL3 := TRUE;
ELSE
        BOOL1 := NOT BOOL1;
        BOOL2 := BOOL1 OR BOOL2;
END_CASE;
```

FOR 循环

使用 FOR 循环可编程重复的过程。

语法:

```
INT_Var :INT;

FOR <INT_Var> := <INIT_VALUE> TO <END_VALUE> {BY <stepsize>} DO
    <instructions>
END_FOR;
```

括号 {} 中的部分是任选项。

只要计数器 <INT_Var> 不大于 <END_VALUE> 就一直执行 <Instructions>。这在执行 <Instructions> 之前进行检查, 以便在 <INIT_VALUE> 大于 <END_VALUE> 时不执行 <Instructions>。当执行 <Instructions> 时, <INT_Var> 总是增加 <Step size (步长)>。步长可以是任何整数。若没有给定这个值, 则它设置为 1。由于 <INT_Var> 只会逐步变大, 因而循环也必须结束。

示例:

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

假设 Var1 的默认值设定为 1, 则经过 FOR 循环后, 它的值为 32。

WHILE 循环

可以像 FOR 循环那样使用 WHILE 循环, 其区别是, 后者的终止条件可以是任何布尔表达式。这意味着, 你可以指定条件, 当条件满足时就执行循环。

语法:

```
WHILE <Boolean expression> DO
<instruction>
END_WHILE;
```

只要 < Boolean expression > 返回 TRUE, 就重复执行 <Instruction>。如第一次求值时 < Boolean expression > 已经是 FALSE, 则不会执行 <Instruction>。若 < Boolean expression > 从不出现值 FALSE, 则 <Instruction> 将无休止地重复, 并导致一个相应的死循环。

注意: 程序员必须确认, 不会形成无休止的循环。可通过改变循环语句指令部分中的条件来检查是否会出现这种情况 (例如通过一个计数器的加计数或减计数)。

示例:

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

WHILE 和 REPEAT 循环在某种意义上要比 FOR 循环的功能更强,这是因为在执行循环之前,我们不需要知道循环次数。因而在某些场合,我们运用这二种循环类型。然而,如果循环次数是明确的,则使用 FOR 循环,这是因为采用这种循环不会出现无休止的循环。

REPEAT 循环

REPEAT 循环与 WHILE 循环不同,这是因为前者只在循环已完成后才检查终止条件。这意味着,与终止条件的运行无关,这种循环至少运行一次。

语法:

```
REPEAT
<instructions>
UNTIL <Boolean expression>
END_REPEAT;
```

直到<Boolean expression>返回 TRUE 为止,一直执行 <Instructions>。如果在第一个 TRUE 求值时已经生成 <Boolean expression>,则只执行一次 <Instructions>。若 <Boolean_expression> 从不出现值 TRUE,则 <Instructions> 将无休止地重复,并导致一个相应的死循环。

注意: 程序员必须确认,不会形成无休止地循环。可通过改变循环语句指令部分中的条件来检查是否会出现这种情况(例如通过一个计数器的加计数或减计数)。

示例:

```
REPEAT
    Var1 := Var1*2;
    Counter := Counter-1;
UNTIL
    Counter=0
END_REPEAT
```

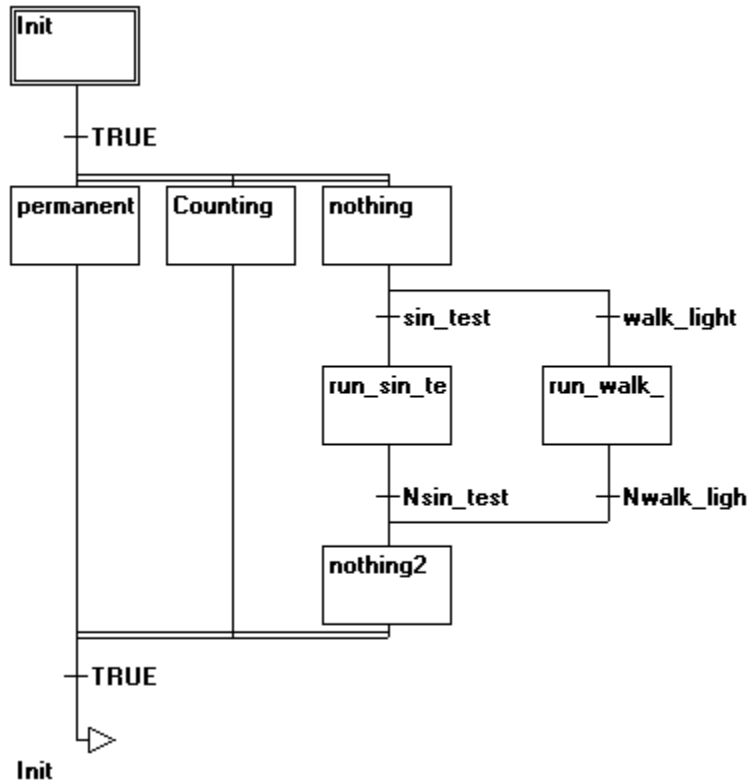
EXIT 指令

若在 FOR、WHILE 或 REPEAT 循环中出现 EXIT 指令,则与终止条件无关,结束最内层的循环。

1.3.4 顺序功能图 (SFC)

顺序功能图是一种面向图形表示的语言,使用顺序功能图,可以按时间顺序描述程序内的不同动作 (Action)。

使用顺序功能图语言的一个网络示例:



步

使用顺序功能图语言编写出的 POU 是由一系列的步组成的，通过定向连接（转换）将这些步彼此连接。步有二种类型。

- 简化类型由一个动作和一个表示该步是否有效的标记组成。若要实施一个步的动作，则在该步的右上角出现一个小三角形。
- 一个 IEC 步由一个标记和一个或几个分配的动作组成。相应的动作出现在步的右边。

动作

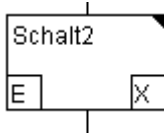
一个动作可以包含一系列由 IL 或由 ST 编写的指令，由 FBD 或由 LD 编制的若干网络，或再次以顺序功能图（SFC）表示的网络。对于简化步，一个动作总是连接到一个步。为了编辑一个动作，在动作所属的步上双击鼠标。或选择步并选择菜单命令“Extras”（附加）“Zoom Action/Transition”（缩放动作/转换）。此外，步可以有输入或输出动作。

IEC 步的动作位于“Object Organizer（对象管理器）”中，并直接处于其 SFC-POU 下。通过双击或通过按压其编辑程序中的 <Enter> 键进行装载。使用“Project（项目）”“Add Action（添加动作）”可建立新的动作。

进入或退出动作

除了步的动作外，你还可对一个步附加一个进入动作和一个退出动作。进入动作只执行一次，正好在该步有效之后。退出动作只在该步失效之前执行一次。带有进入动作的步由一个左下角“E”指示，退出动作由一个右下角“X”指示。可用任何一种语言来实现进入和退出动作。为了编辑进入或退出动作，在步的相应角双击鼠标。只能对简化步定义一个进入和退出动作，而不能对一个 IEC 步定义进入和退出动作。

带有进入和退出动作的步示例：



转换/转换条件

在各步之间有所谓的转换。一个转换条件必需具有值 **TRUE** 或 **FALSE**。从而，它可以由一个布尔变量、一个布尔地址或者是一个布尔常数组成。它也可包含一系列有布尔结果的指令，这些指令或是用 **ST** 语言，（例如， $(i \leq 100) \text{ AND } b$ ），或是使用任何所希望的语言（参见“附加”“缩放动作/转换”）编写，但是，一个转换也许并不包含程序、功能块或赋值！

注意：

除了转换外，包含的模式也可用于跳转到下一步；参见 **SFCtip** 和 **SFCtipmode**。

激活步

调用 **SFC POU** 后，首先执行属于初始化步的动作（用一个双边框表示）。正在执行步动作的步称为有效步。若步是有效的，则每个循环执行一次相应的动作。在 **Online**（联机）模式，有效步用蓝色表示。每个步都附属有一个标志，该标志存储步的条件。应用布尔结构元素 **<步名字>.x** 的逻辑值来表示步标志（步的有效或失效条件）。若相应的步是有效的，这个布尔变量有值 **TRUE**，若是失效的，则为值 **FALSE**。这个变量是隐式地声明的，并可用于 **SFC POU** 的任何动作和转换。在一个控制周期内，执行属于有效步的所有动作。此后，若后继步的转换条件为 **TRUE**，则有效步相应动作的后继步成为有效步。当前的有效步将在下一个周期内执行。

注意：

若有效步包含一个输出动作，则这个有效步仅在下个周期内执行（假定后继的转换是 **TRUE**）。

IEC 步

与简化步一起，也提供使用 **SFC** 语言编写的标准 **IEC** 步。为了能使用 **IEC** 步，你必需将库 **TcSystem.Lib** 连接到你的项目内。

可将任何数量的动作分配给一个 **IEC** 步。**IEC** 动作并不像简化步那样将输入步或输出动作固定在某些步上，而是独立于步进行存储，并可在一个 **POU** 内多次地重复使用。为此目的，**IEC** 步必须使用命令 **'Extras"Associate action'**（‘附加’‘关联动作’）与单个步相关联。

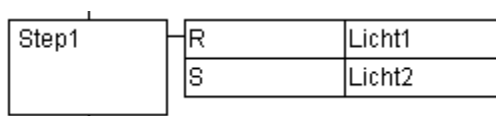
与动作一起，可将布尔变量分配给步。

动作和布尔变量的有效和失效可应用所谓修饰符加以控制。延时是可能的。由于一个动作可以依旧是有效的（若已处理下一步），例如经过修饰符 **S (Set)**，因而可实现并行处理。

每一次用 **SFC** 块来置位或复位一个关联的布尔变量。这意味着，随着每次调用，更改 **TRUE** 或 **FALSE** 的值，或重新回到原值。

与 **IEC** 步相关联的动作示于步右侧的一个二分框内。左边区域包含修饰符，（可能有时间常数），右边区域包含动作名。

带有二个动作的 **IEC** 步示例：



为了方便地跟踪过程，所有联机模式下的有效动作如同有效步那样以蓝色表示。在每个循环后进行一次检查，察看哪一个动作是有效的。

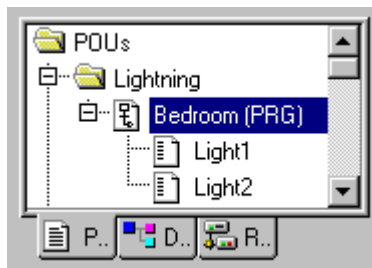
还要注意在动作中使用时间修饰符的一些约束条件，在相同的循环内，这些约束条件是可以被重复使用的。

注意:

若已撤消了一个动作，它还可能再执行一次。这意味着，每个动作至少会执行二次（以及有修饰符 P 的动作）。若出现一个调用首先执行失效的动作，然后每次以字母的次序执行有效的动作。

一个新插入的步是否是一个 IEC 步，取决于是否已选用菜单命令 "Extras" (附加) "Use IEC-Steps" (使用 IEC 步)。

在“Object Organizer (对象管理器)”内，动作直接附加在相应的 SFC POU 下面。“Project (项目)” “Add Action (添加动作)”可建立新的动作。为了使用 IEC 步，必需在项目内包括专用的 SFC 库 lecsfc.lib。



为了使动作与 IEC 步相关联，提供以下的修饰符：

N	不存储	只要步有效动作就有效
R	复位优先	动作失效
S	置位 (存储的)	动作有效并在复位之前一直保持有效。
L	时间“有限的”	在一定的时间内动作是有效的。
D	时间“延迟的”	只要步始终保持有效，经过一定的时间后，动作有效。
P	脉冲	若步是有效的，动作只执行一次。
SD	存储的和时间‘延迟的’	经过一定的时间后该动作有效，并且保持直到出现一个‘复位’。
DS	“延迟的”和“存储的”	只要步仍然保持有效，经过一定的时间后该动作有效，并且保持有效，一直到出现一个‘复位’。
SL	“存储的”和时间“有限的”	在一定时间内该动作是有效的。

注意:

如果在二个直接紧邻的步中采用有修饰符的相同动作（它影响时间流），时间修饰符在第二次使用时不能成为可操作的。为了避免出现这种情况，必须插入一个中间步。这样，当通过附加循环时可以重新使动作状态初始化。

SFC 语言内的隐含变量

SFC 语言具有隐含声明的可用变量。每个步都有一个保存该步状态的标志。步标记（步的有效或失效状态）对 IEC 步而言称为 <步名>.x，或对简化步而言称为 <步名>。当关联的步有效时，这个布尔变量具有值 TRUE，当它失效时，其值为 FALSE。它可用于 SFC 块的每个动作和转换。我们可使用变量 <动作名>.x 进行一次询问，以了解一个 IEC 动作是有效的还是失效的。

选择分支

SFC 语言中的两个或多个分支可定义为选择分支（**alternative branch**）。每个选择分支必需由一个转换来开始和结束。选择分支可包含并行分支和其它选择分支。一个选择分支由一条水平线开始（选择分支开始），并由一条水平线或一次跳转结束（选择分支结束）。

如果位于分支开始线之前的步是有效的，则从左向右求值每个选择分支的第一个转换。如果从左边开始的第一个转换其转换条件为值 **TRUE**，则开放相应的支路，随后的步均为有效。

对于 IEC 步而言，隐含变量 `<步名>.t` 可用来询问各步的有效时间。其它程序也可存取隐含变量。示例：`boolvar1:=sfc.step1.x`；这里 `step1.x` 是隐含布尔变量，它表示 POU Sfc1 中的 `step1`（步 1）的 IEC 步状态。

并行分支

SFC 语言中的两个或多个分支可定义为并行分支（**parallel branch**）。每个并行分支必需由一个步来开始和结束。并行分支可包含选择分支或其它并行分支。一个并行分支由一条双线开始（并行分支开始），并由一条双线或一次跳转结束（并行分支结束）。

如果并行开始线以前的步是有效的，而且这个步以后的转换条件是值 **TRUE**，则所有并行分支的第一步都成为有效的（见“有效步”）。至此，这些分支彼此并行地进行处理。如果所有以前的步都有效，而且该步前面的转换条件生成值 **TRUE**，则并行结束线后的步变为有效。

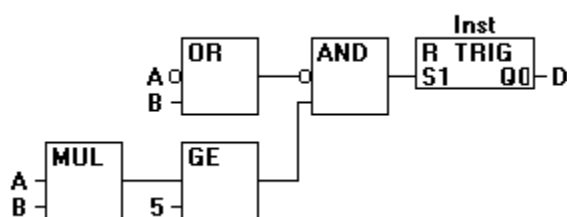
跳转

跳转（**jump**）即是到一个步的连接，该步名出现在跳转符号下。由于不允许建立向上的引导或彼此交叉的连接，因此需要跳转。

1.3.5 功能块图（FBD）

功能块图是一种图形化的编程语言。它以列举的若干网络运行，每个网络包含一种结构，该结构可以是逻辑表达式、算术表达式、功能块调用、跳转或是返回指令。

在 TwinCAT PLC Control 中，功能块图中可能出现的典型网络示例如下：



1.3.6 连续功能图编辑器（CFC）

连续功能图编辑器的操作不同于带有网络的功能块图（FBD），而是带有可自由布置的元素。这就允许（例如）反馈。在连续功能图编辑器中可能出现的典型网络示例：



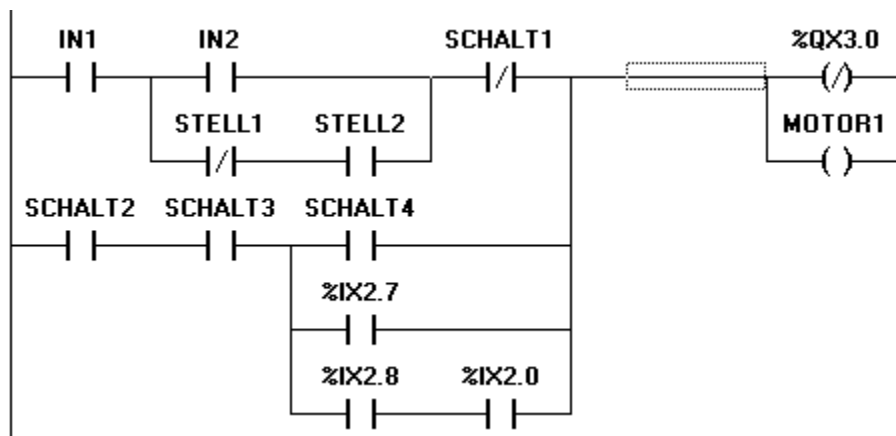
1.3.6 梯形图 (LD)

梯形图也是一种面向图形的编程语言，它近似于电路结构。

一方面，梯形图适用于建立逻辑开关，另一方面，我们也可以如同使用 FBD 那样建立各种网络。因此，LD 在控制其它类型 POU 调用方面是非常有用的。（更为详细的情况在后面叙述）。梯形图由一系列网络组成。网络局限于由左、右各一条垂直的电流线所限定的左、右侧内。其中间是一个电路图，由接点、线圈和连接线组成。

每条电路左侧由一系列接点组成，从左至右传递条件“ON”或“OFF”，这相当于布尔值 TRUE 和 FALSE。每个接点都有一个布尔变量。若这个变量是 TRUE，则沿着连接线从左至右传递条件。否则，右侧连接接收值 OFF。

TwinCAT PLC Control 可能出现的典型梯形图网络示例：



接点

LD 中的每条网络由网络左侧开始的接点（接点由二条并行线 \parallel 表示）组成，这些接点从左至右表示条件“On”或“Off”。其条件相当于布尔值 TRUE 和 FALSE。每个接点有一个布尔变量。若这个变量是 TRUE，则条件从左至右借助于连接线通过，否则，右边连接接收值“Out”。接点可并联连接，因此并行分支必须有一个分支传输值“On”，才能使并联支路传输值“On”。接点亦可以串联，因此所有接点必须传输条件“On”，以便使最后一个接点传输“On”条件。以上相当于一条并联或串联电路。也可以对一个接点求反，通过在接点符号中的斜线“/”加以识别。因此，若变量是 FALSE，则传输“On”值。

线圈

LD 中的网络右侧，可以有任何数量的线圈，它用括号“()”表示。它们只能并联。一个线圈将连接的值从左传输到右，并将它复制到一个相应的布尔变量。在进入线处，可以出现值 ON（相当于布尔变量 TRUE），或值 OFF（相当于布尔变量 FALSE）。也可对接点和线圈求反（示例中，接点 SWITCH1 和线圈 %QX3.0 是求反的）。若一个线圈是求反的（可通过线圈符号中的斜线“/”识别），则它将求反值复制到相应的布尔变量内。若一个接点是求反的，则它仅当相应的布尔值为 FALSE 时才接通。

梯形图中的功能块

与接点和线圈一起，你也可以插入功能块和程序。在网络中，它们必须有带布尔值的一个输入和一个输出，并可在相同位置上像接点那样使用，也就是说在 LD 网络的左侧。

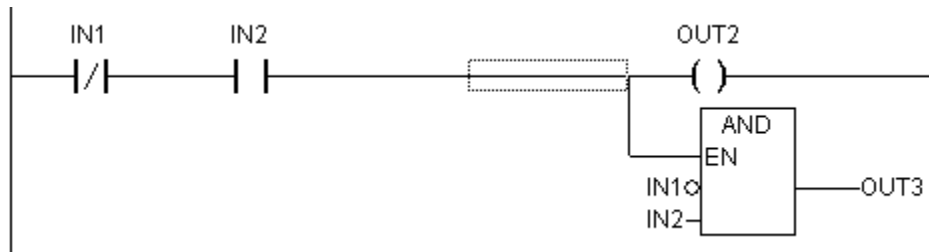
置位/复位线圈

也可将线圈定义为置位线圈或复位线圈。置位线圈可通过线圈符号“(S)”中的“S”来识别。它从不改写在相应布尔变量中的值 TRUE。这就是说，若变量一旦设置为 TRUE，则它将保持下去。复位线圈可通过线圈符号“(R)”中的“R”来识别。它从不改写在相应布尔变量中的值 FALSE。这就是说，若变量一旦设置为 FALSE，则它将始终保持下去。

如同 FBD 的 LD

当使用 LD 时，有可能要利用接点开关的结果控制其它 POU。一方面，你可使用线圈将结果作为一种全局变量，随后将它用于其它地方。然而，你也可以将可能的调用直接插入到你的 LD 网络内。为此，可引入一个带有 EN 输入的 POU。这种 POU 是完全常规的操作数、功能、程序、或有一个附加输入（以“EN”标记）的功能块。EN 输入总是为布尔类型，其含义是：当 EN 有值 TRUE 时，计算有 EN 输入的 POU。一个 EN POU 与线圈并联连接，因而 EN 输入将连接到接点和线圈之间的连接线。若 ON 信息经过这条线传输，则这个 POU 将完全正常地进行计算。从这样一个 EN POU 出发，你可建立类似于 FBD 的网络。

带有一个 EN POU 的一个 LD 网络部分



1.4 调试，联机功能

抽样跟踪

通过抽样跟踪，可根据触发事件跟踪变量值的顺序。这是预定义的布尔变量（触发变量）的上升沿或下降沿。TwinCAT PLC Control 允许最多跟踪 20 个变量。有一个 64 kB 的环形缓冲区。

调试

使用 TwinCAT PLC Control 的调试功能，可方便地查找错误所在。

断点

一个断点是程序内处理停止的地方。从而有可能在程序内的一些特定位置观察变量的值。

可以在所有的编辑器内设置断点。在文本编辑器中，断点设置在行上；在 FBD 和 LD 中，设置在网络号上；而在 SFC 中，设置在步上。

在功能块实例中不能设置断点。

单步

单步意味着：

- 使用 IL 语言：直到下一个 CAL、LD 或 TMP 命令之前，一直执行程序。
- 使用 ST 语言：执行下一个指令。
- 使用 FBD, LD：执行下一个网络。
- 使用 SFC：直到下一个步之前，动作一直继续。
- 使用 CFC：执行 CFC 程序中的下一个 POU（Box）。

通过逐步操作，你可以检查程序的逻辑正确性。

单循环

若已选用单循环，则在每次循环后执行被停止。

在联机情况下改变值

在操作过程中，可将变量一次设定在某个值（写入值），或在每个循环后再次设定为某个值（强制值）。在联机工作方式，通过在该值上双击可更改变量值。通过对每个其它类型的变量进行从 TRUE 到 FALSE（或相反）的布尔变量改变，我们可得到对话框“**Write Variable xy**”（写变量 xy），在该对话框中可编辑变量的实际值。

监视

在联机工作方式，所有可显示的变量都是从控制器中读出的，并以实时方式显示。你可以在声明和程序编辑器中看到这种显示；也可从监视和接收管理器 and 可视化设备中读出变量的当前值。若要监视功能块的实例变量，应首先打开相应的实例。

在监视 VAR_IN_OUT 变量时，输出为间接引用的值。在十字准线上单击或在行上双击，显示会被扩大或被截短。

在监视指针时，将在声明部分输出指针和间接引用的值。在程序部分则只有指针输出：

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

另外，还相应地显示间接引用值的指针。

在十字准线上单击或在行上双击，则显示会被扩大或被截短。

监视 ARRAY 部件：除了以常数索引数组部件外，还可显示由变量索引的部件：

```
anarray[1] = 5
```

```
anarray[i] = 1
```

若索引由表达式（例如 [i+j] 或 [i+1]）组成，则不能显示部件。

仿真

在仿真过程中，不是在 PLC 中处理已建立的 PLC 程序，而是在 TwinCAT PLC Control 的计算空间内处理运行。提供所有的联机功能。因此不需要 PLC 硬件，你就可以测试程序的逻辑正确性。

仿真工作方式只能用于 Buscontroller (BCxx00)。若你在 PC (Code generation i386) 上使用 TwinCAT，不需要物理 I/O，你可以直接在运行系统中仿真程序。

“Log”（日志）

联机方式运行时，日志按时间顺序记录用户动作、内部过程、状态改变和异常事件。日志用于监视和用于跟踪错误。

1.5 IEC 61131-3

IEC 61131-3 标准是用于可编程序控制器编程语言的国际标准。在 TwinCAT PLC Control 中提供的编程语言符合该国际标准的要求。按照该标准，一个程序由以下元素组成：

- 结构
- POU
- 全局变量

与 IEC 61131-3 有关的文献：

- Flavio Bonifatti et al.: IEC 1131-3 Programming Methology, Seyssins: CJ International, 1997. ISBN 2-9511585-0-5
- Karl-Heinz John, Michael Tiegelkamp: IEC 61131-3 Programming Industrial Automation Systems, Berlin: Springer-Verlag, 2001. ISBN 3-540677526
- R. W. Lewis: Programming industrial control systems using IEC 1131-3, London: IEC Publishing, 1998. ISBN 0 85296 950 3

2 样例程序

2.1 样例程序

现在让我们开始编写一个小样例程序。它用于一个小型交通讯号装置，假定用来控制一个路口的两个交通讯号。两个交通讯号的红灯/绿灯阶段交替，为了避免发生意外，我们在这两个阶段之间插入黄灯或黄灯/红灯过渡阶段。后者的持续时间小于前者。在这个例子中，你会理解，如何使用 IEC61131-3 标准的语言资源来表示与时间有关的程序，如何借助于 TwinCAT PLC Control 来编辑该标准的不同语言。

建立 POU

首先选中 TwinCAT PLC Control，并选择命令“File（文件）”→“New”（新建）。在出现的对话框中，第一个 POU 应用名 PLC_PRG。POU 的类型应确定是一个程序。在这种情况下，我们选用该 POU 的语言为顺序功能图 (SFC)。现在应用菜单条或快捷菜单（在“Object Organizer（对象管理器）”中按鼠标右键），通过命令“Project（项目）”→“Object Add（添加对象）”建立两个对象。对于以功能块图（FBD）语言编写的名为 TRAFFICSIGNAL 的一个功能块，以及也是功能块类型的一个 POU WAIT，我们将要以指令表 (IL) 加以编程。

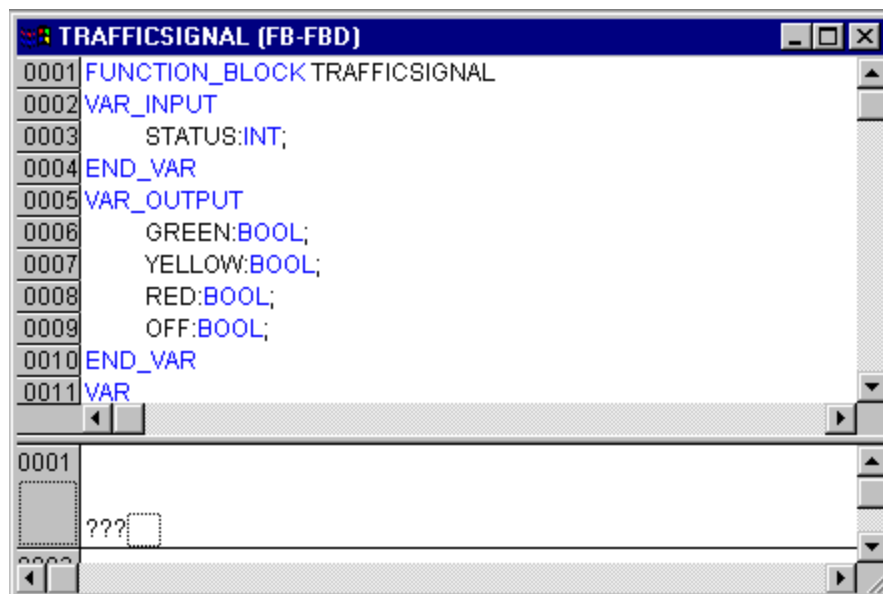
在 POU TRAFFICSIGNAL，我们将各交通讯号阶段分配给信号灯，亦即，我们要确认在红灯阶段和在黄灯/红灯阶段，红灯要发红光；在黄灯和黄灯/红灯阶段，黄灯要发黄光，依次类推。

在 WAIT 中，我们将对一个简单的定时器编程，它作为输入将接收以毫秒为单位的时间长度，并作为输出会在定时时间到达瞬间立即生成 TRUE。

PLC_PRG 将在结束处组合每件事，以便在所需要的时段使合适的灯在合适的时间发光。

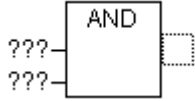
现在让我们返回到 POU TRAFFICSIGNAL。在编辑器的声明中，一个名为 STATUS、类型为 INT 的变量将作为输入变量声明（在关键字 VAR_INPUT 和 END_VAR 之间）。STATUS 有五种可能的状态，分别用于 TRAFFICSIGNAL 绿灯、黄灯、黄灯/红灯、红灯和熄灭阶段。

因此，我们的 TRAFFICSIGNAL 有四个输出，即 RED（红灯）、YELLOW（黄灯）、GREEN（绿灯），以及熄灭。你应声明这四个变量。从而功能块 TRAFFICSIGNAL 的声明部分如下：

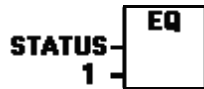


功能块 TRAFFICSIGNAL, 声明部分

现在要从 POU 的输入 STATUS 确定输出变量值。为此, 进入到 POU 的本体。点击第一个网络的左边字段(有数字 1 的灰色字段)。现在你已选择出第 1 个网络。随即选择菜单项“Insert”(插入)→“Operator”(操作符)。在第一个网络中, 插入操作符 AND 和两个输入的一个框图:



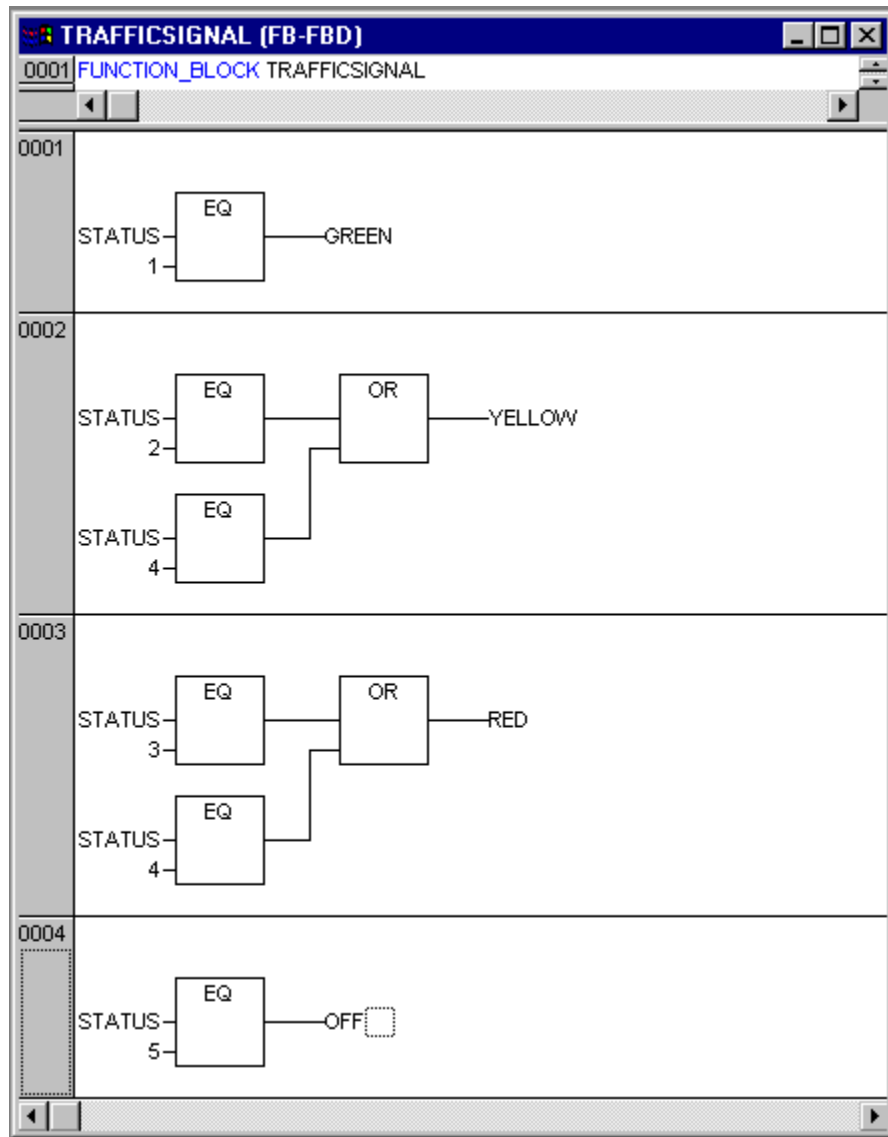
用鼠标指针单击文本 AND 并将该文本改为 EQ。从两个输入的的第一个输入选择三个问号并输入变量 STATUS。然后选择第二个输入的三个问号并输入常数“1”。你得到如下的网络:



现在, 点击 EQ 框后的一个位置。现在选择 EQ 操作的输出。选择命令“Insert”(插入)→“Assignment”(赋值)。将三个问号“???”改为 GREEN (绿灯)。至此, 你已建立了以下结构的一条网络:



STATUS 与 1 相比较, 其结果赋值给 GREEN。若预置状态值为 1, 这条网络会切换到 GREEN。对于其它的 TRAFFICSIGNAL 颜色或对于 OFF (熄灭), 我们还需要三条网络。你可用命令“Insert”(插入)→“Network (after)” (网络 (后向)) 建立这三条网络。这些网络应如样例所示那样建立。最后, 完成的 POU 如下:



为了将一个操作符插入到其它操作符前面，你必须选择一个位置，在这一位置，你要连接到被插入操作符的输入，并馈送到框内。然后利用命令“Insert”（插入）→“Operator”（操作符）。或者用其它方法，如同第一条网络那样建立这些网络。现在，第一个 POU 已完成。按照 STATUS 的输入值，TRAFFICSIGNAL 控制所要求的交通灯颜色。

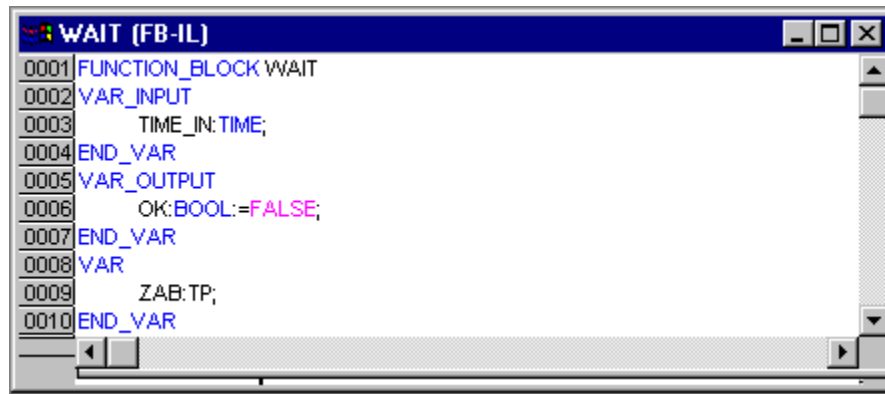
对于 POU WAIT 中的定时器，我们需要一个来自标准库的 POU。因此，用命令“Window”（窗口）→“Library Manager”（库管理器）打开库管理器。选择命令“Insert”（插入）→“Additional library”（附加库）。出现用于打开文件的对话框。从库表中选择 standard.lib（标准库）。

现在，让我们回到 POU WAIT。假定这个 POU 成为一个定时器，我们可用它来确定每个 TRAFFICSIGNAL 阶段的时段长度。POU 作为输入变量，它接收类型 TIME（时间）的变量 TIME_IN；而作为输出，它生成 OK 的一个布尔值，并在所需要的时段结束时，它应是 TRUE。通过在声明结束处（在分号之前）插入“:= FALSE”来设置 FALSE。

对于我们的目的而言，我们需要 POU TP（一个时钟脉冲发生器）。它有两个输入（IN, PT）和两个输出（Q, ET）。TP 有以下作用：

只要 IN 是 FALSE，则 ET 为 0，Q 为 FALSE。一旦 IN 提供值 TRUE，则在输出 ET 处立即以毫秒单位计算时间。当 ET 达到值 PT 时，ET 不再计数。与此同时，只要 ET 小于 PT，Q 就生成 TRUE。一旦 PT 达到预设置值，Q 就立即变为 FALSE。此外，你从附录中的标准库中还可以找到所有 POU 的简要描述。

为了将 POU 用于 POU WAIT，必需从 TP 建立一个局部实例。为此，需要说明类型 TP（在关键字 VAR、END_VAR 之间）的局部变量 ZAB（用于定时）。这样，WAIT 的声明部分是这样：



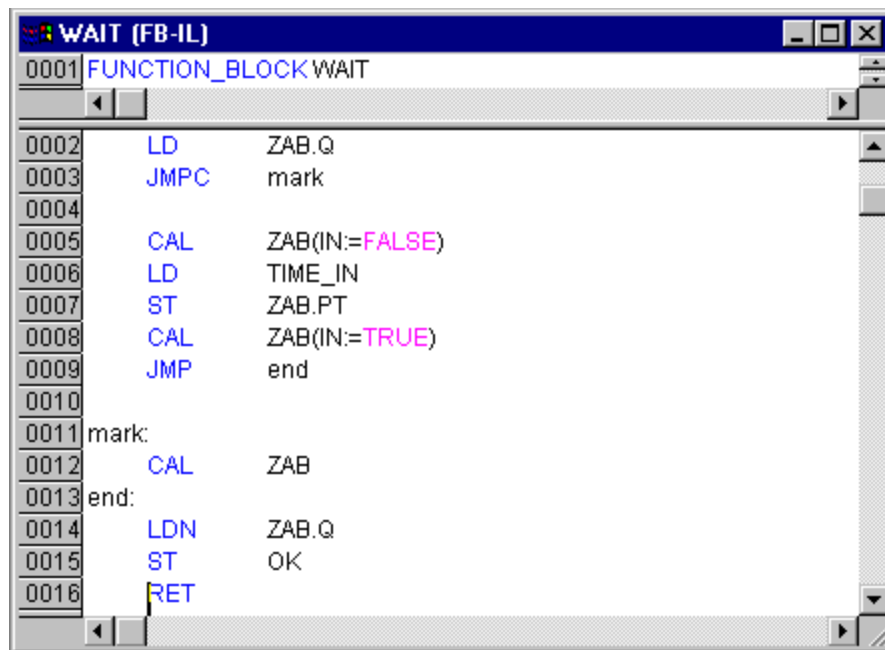
```

0001 FUNCTION_BLOCK WAIT
0002 VAR_INPUT
0003     TIME_IN:TIME;
0004 END_VAR
0005 VAR_OUTPUT
0006     OK:BOOL:=FALSE;
0007 END_VAR
0008 VAR
0009     ZAB:TP;
0010 END_VAR

```

功能块 WAIT，声明部分

为了建立所需要的定时器，必须按以下方式对 POU 本体进行编程：



```

0001 FUNCTION_BLOCK WAIT
0002 LD     ZAB.Q
0003 JMPC  mark
0004
0005 CAL   ZAB(IN:=FALSE)
0006 LD     TIME_IN
0007 ST     ZAB.PT
0008 CAL   ZAB(IN:=TRUE)
0009 JMP   end
0010
0011 mark:
0012 CAL   ZAB
0013 end:
0014 LDN   ZAB.Q
0015 ST     OK
0016 RET

```

功能块 WAIT，指令部分

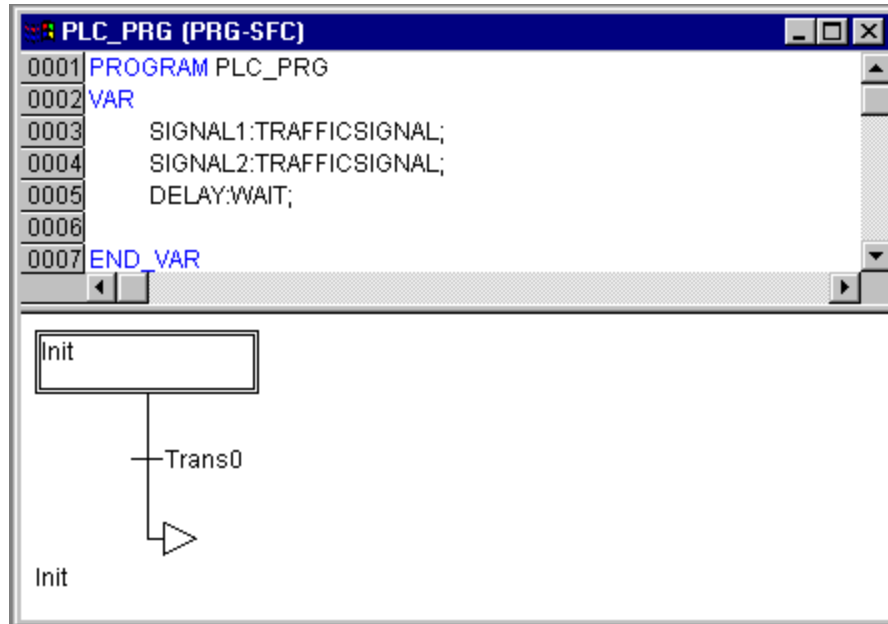
首先提出的问题是，Q 是否已经设置为 TRUE（就像已经执行了计数），在这种情况下，调用 ZAB 不改变任何事情，但是我们宁可不用输入来调用功能块 ZAB（为了检查时段是否已经超时）。

其它情况下，我们在 ZAB 的变量 IN 设置为 FALSE，因此，同时在 ET 处设置为 0，在 Q 处为 FALSE。用这种方式，所有的变量都设置在所要求的初始状态。接下来我们将变量 TIME_IN 的时间赋值到变量 PT，并用 IN:=TRUE 调用 ZAB。现在，在功能块 ZAB 内，直到 ET 达到值 TIME_IN 之前一直对它进行计算，然后，Q 设置为 FALSE。

每次执行 WAIT 后，将 Q 的求反值保存在 OK 内。一旦 Q 是 FALSE，OK 就生成 TRUE。

就在这个时刻完成定时器计数。至此需将两个功能块 WAIT 和 TRAFFIGSLGNAL 组合在主程序 PLC_PRG 中。

首先，我们声明所需要的变量。这就是功能块 TRAFFICSIGNAL 的两个实例（TRAFFICSIGNAL1，TRAFFICSIGNAL2）和类型 WAIT（作为延时的 DELAY）。PLC_PRG 如下：



程序 PLC_PRG，版本 1，声明部分

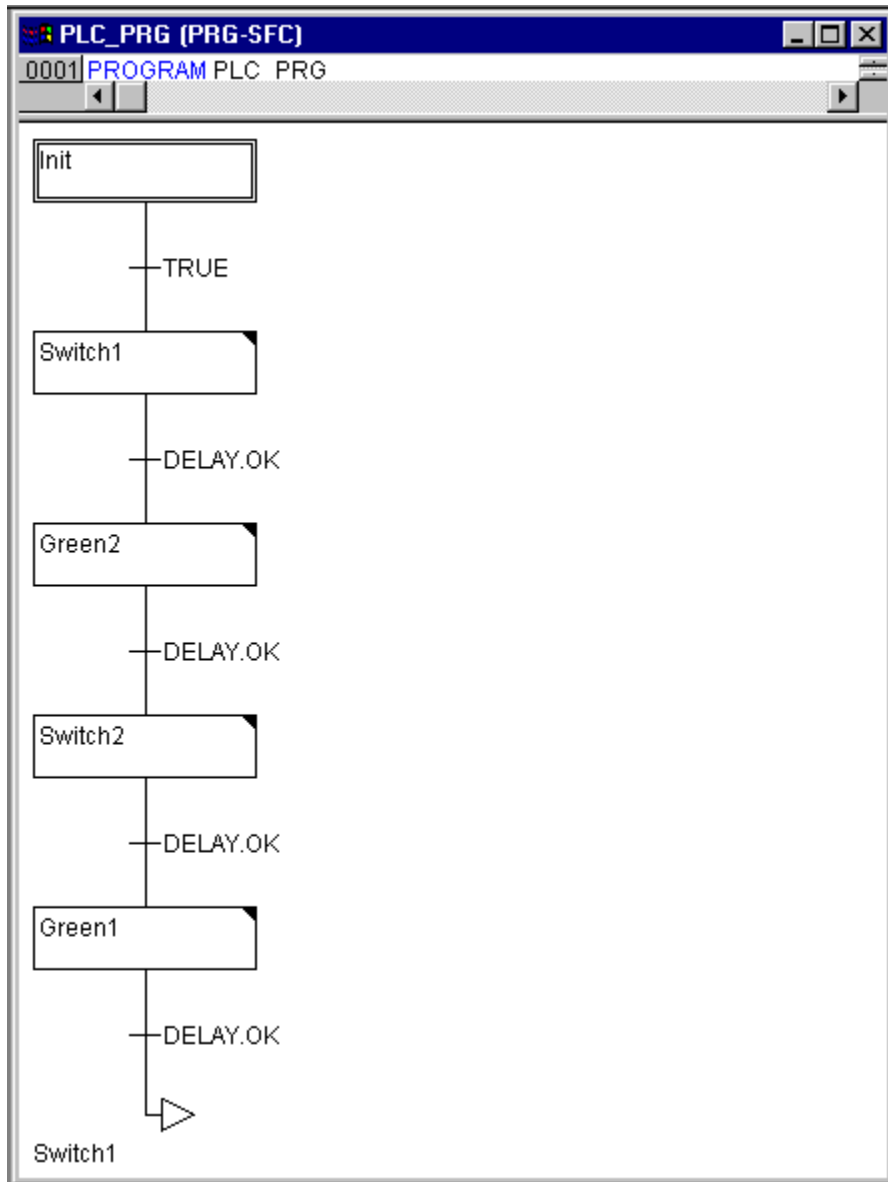
SFC 语言的 POU 初始图总是由后随转换“Trans0”的一个动作“Init”和返回到 Init 的一个跳转组成。需要将它们适当地进行扩展。

首先，在对各个动作（action）和转换（transition）进行编程之前，先确定图的结构。需要为每个 TRAFFICSIGNAL 阶段编程一个步。通过标记 Trans0 和选择命令“Insert”（插入）→“Step transition (behind)”（步转换（后向）），将步插入。并重复三次这个步骤。

如果你直接在一个转换和一个步名上点击，则可作出标记，并进行更改。对 Init 后的第一个转换命名为“TRUE”，所有其它转换命名为“DELAY.OK”。

第一个转换总是切换，而所有其它转换为，当 DELAY 时为 OK，并生成 TRUE，即完成设置时段。

各步（从上到下）命名为 Switch1、Green2、Switch2、Green1，当然，Init 保持其 Name（名）。“Switch”（切换）意味着，每当一个黄灯阶段，在 Green1 TRAFFICSIGNAL1 和在 Green2 TRAFFICSIGNAL2 将是绿灯。最后，更改 Switch1 后的 Init 返回地址。如果你已经正确地完成了上述的每个操作步骤，则显示下图：

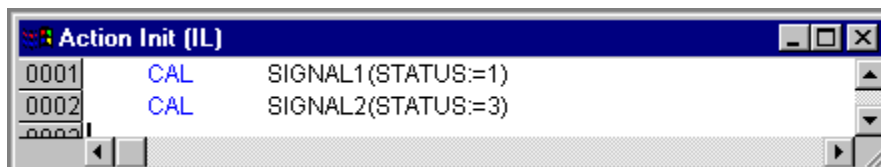


程序 PLC_PRG, 版本 1, 指令部分

现在, 我们将完成对各步的编程。若在一个步的字段上双击, 则打开一个对话框, 从而打开一个新的动作。在我们的示例中, 我们将使用 IL (指令表) 语言。

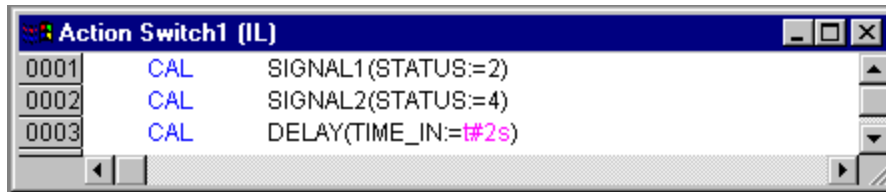
动作和转换条件

在步动作 Init, 对变量进行初始化, 即 TRAFFICSIGNAL1 的 STATUS 应是 1 (绿灯)。TRAFFICSIGNAL2 的状态应是 3 (红灯)。从而动作 Init 如下所示:



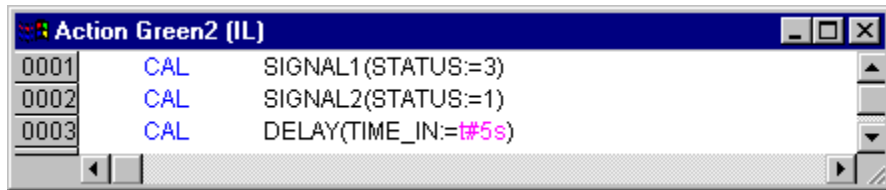
动作 Init

Switch1 将 TRAFFICSIGNAL1 改变为 2 (黄灯), 将 TRAFFICSIGNAL2 改变为 4 (黄灯-红灯)。此外, 现在设定一个 2000 毫秒的延时。动作如下:



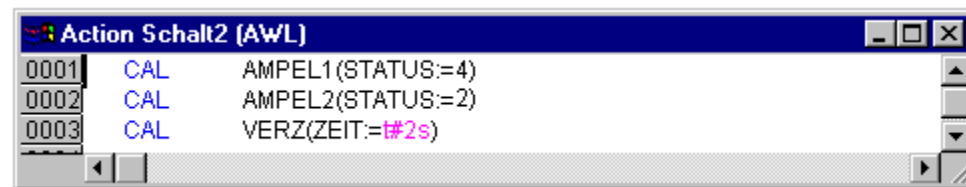
动作 Switch1

对于 Green2, TRAFFICSIGNAL1 是红灯 (STATUS:=3), TRAFFICSIGNAL2 是绿灯 (STATUS:=1), 延时设定在 5000 毫秒。



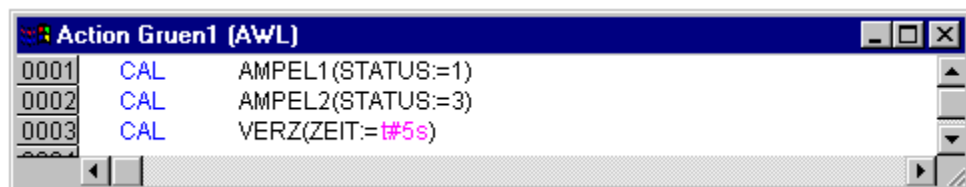
动作 Green2

在 Switch2, TRAFFICSIGNAL1 的 STATUS 改变为 4 (黄灯-红灯), TRAFFICSIGNAL2 改变为 2 (黄灯)。现在设定一个 2000 毫秒的延时。



动作 Switch2

Green1, TRAFFICSIGNAL1 是绿灯 (STATUS:=1), TRAFFICSIGNAL2 是红灯 (STATUS:=3), 延时设定在 5000 毫秒。



动作 Green1

至此已完成程序的第 1 个版本。现在可以编辑这个程序和测试仿真。

为了保证图表至少有一个选择分支, 以便使我们在夜晚能断开交通信号灯, 需要在图中包括一个计数器, 它在 TRAFFICSIGNAL 计数到一定次数的循环后将设备断开。

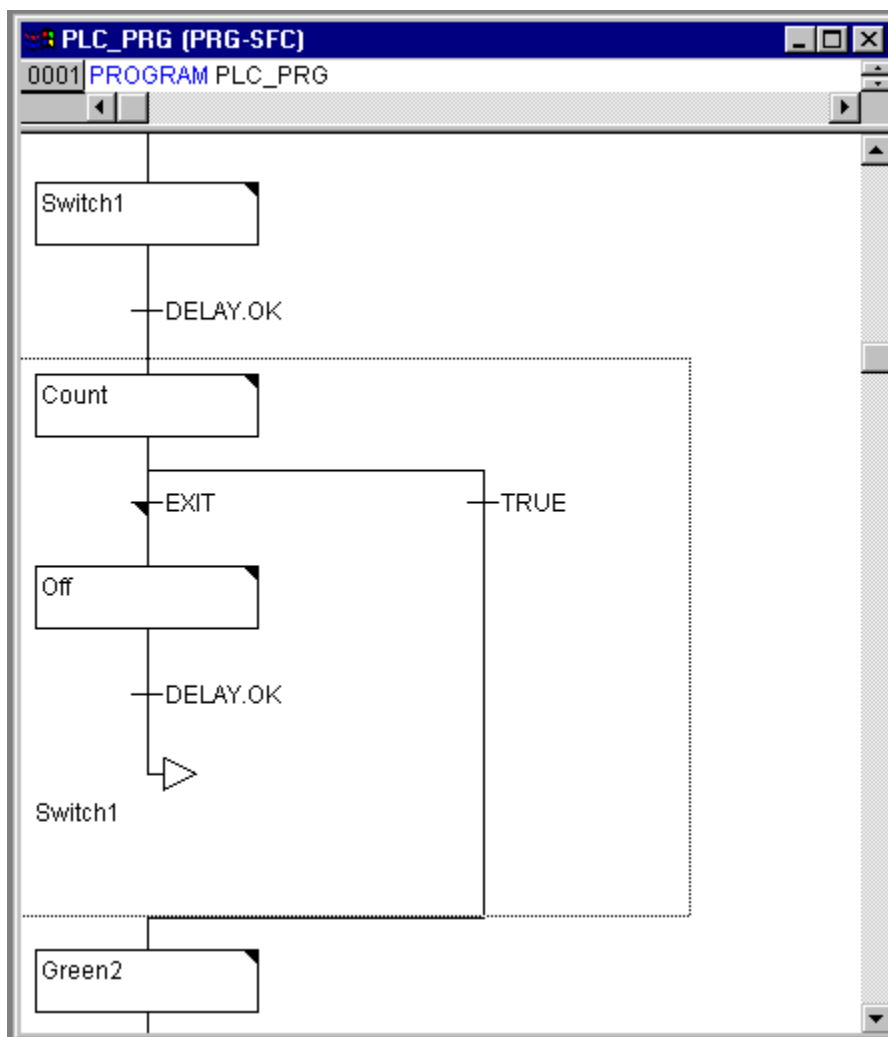
首先, 我们需要一个类型为 INT 的新变量 COUNTER。应用 PLC_PRG 声明部分中常用的方式对它进行声明, 并用 0 将它在 Init 处初始化。

Action Init (IL)		
0001	CAL	SIGNAL1(STATUS:=1)
0002	CAL	SIGNAL2(STATUS:=3)
0003		
0004	LD	0
0005	ST	COUNTER

动作 Init, 版本 2

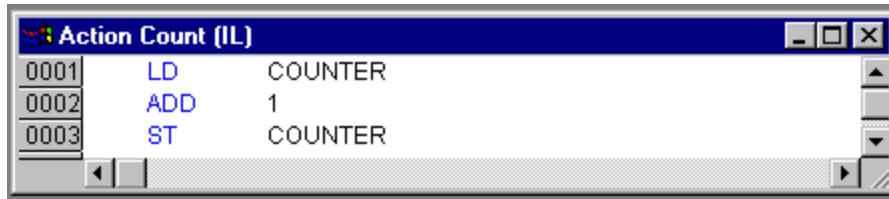
现在, 在 Switch1 后选择转换, 并插入一个步, 然后插入一个转换。选择最终转换, 并将一个选择分支插入到它的左边。在左边转换后面插入一个步和一个转换。在最终的新转换后面, Switch1 后插入一个跳转。

新部分的命名如下: 两个新步的上面应为“Count”(计算), 而下面则为“Off”(断开) 这些转换分别为(从上到下, 从左到右) EXIT、TRUE 和 DELAY.OK。新的部分为黑色框部分。



交通讯号灯装置

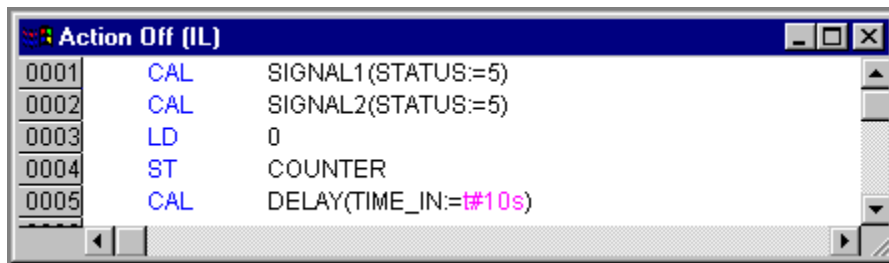
现在要实现两个新的动作和一个新的转换条件。在步 Count 处，唯一发生的事情是 COUNTER（计数器）增量 1：



EXIT 转换检查计数器是否大于某个数（如假定为 7）：



在 OFF 处，两个信号灯的状态都设定在（OFF），COUNTER 复位到 0，设定 10 秒的延时。

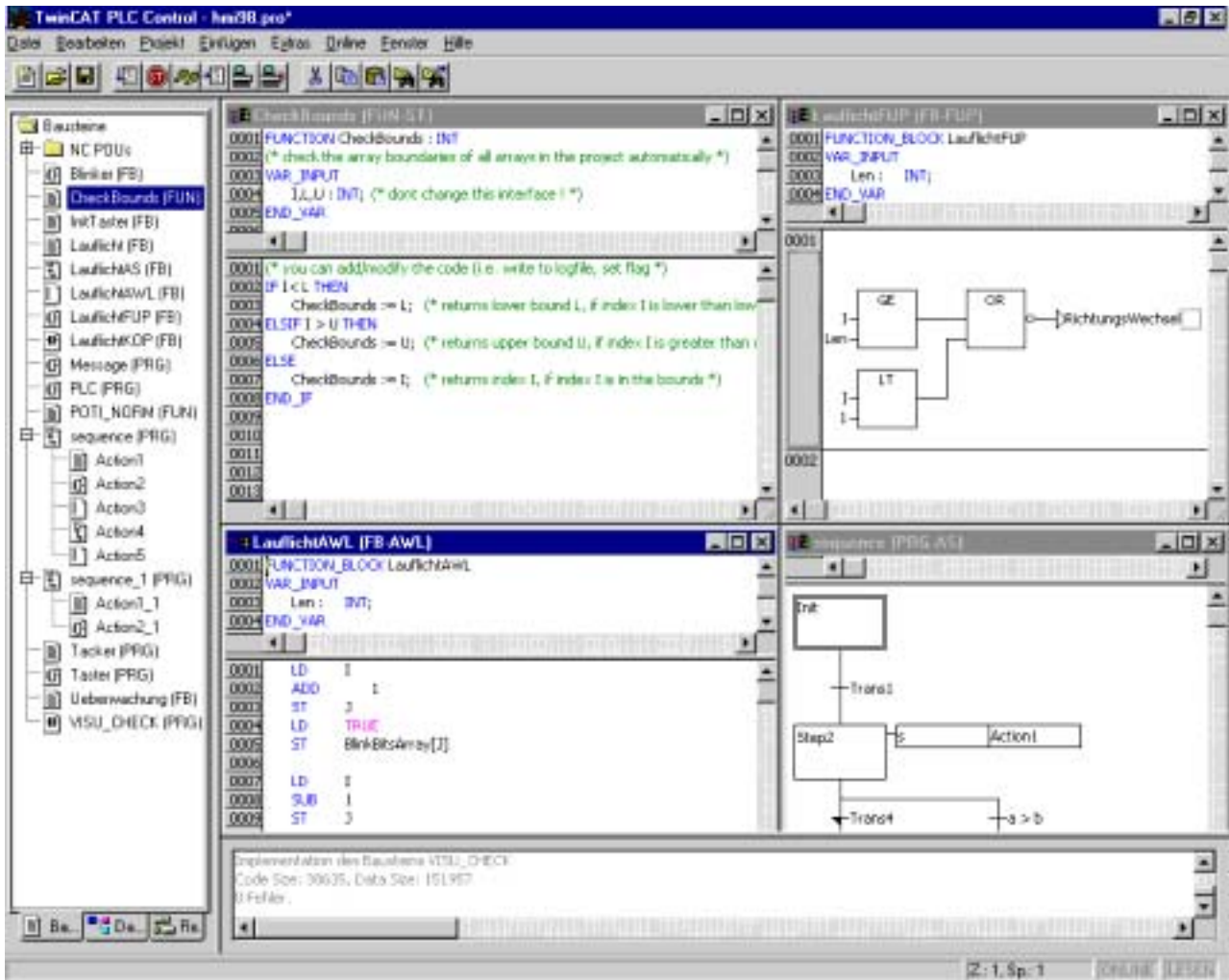


在我们所假定的情况，当 TRAFFICSIGNAL 完成 7 次循环后，夜幕降临，经 10 秒后，TRAFFICSIGNAL 将自身断开。然后，我们再次迎来白天，交通信号灯装置将自身接通，整个过程从头开始。

现在，测试你的程序。为此，你必需对它进行编译（命令“Project”（项目）→“Rebuild all”（全部重建）），登录（命令“Online”（联机）→“Login”（登录）），然后装入（命令“Online”（联机）→“Download”（装载））。如果你选择命令“Online”（联机）→“Run”（运行），则可按时间顺序跟踪主程序的各个步。现在，POU PLC_PRG 的窗口已变为监视窗。在声明编辑器中的“+”符号上双击，变量显示下拉，你可以看到各个变量的值。

3 子项组件

3.1 主窗口

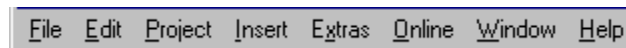


在 TwinCAT PLC 的主窗口中可以找到以下元素（从上到下）：

- 菜单条
- 工具条（选项）：带有快速选择菜单命令的按钮。
- 对象管理器，带有 POU、数据类型（Data type）以及资源（Resource）的属性页
- 在对象管理器和 TwinCAT PLC Control 的“Work space”（工作区）之间的一个垂直屏幕分隔器。
- 编辑器窗口所在位置的“Work space”（工作区）
- 消息窗（选项）
- 状态条（选项）：带有项目当前状态的信息

菜单条

菜单条位于主窗口的上边。它包含所有命令。



工具条

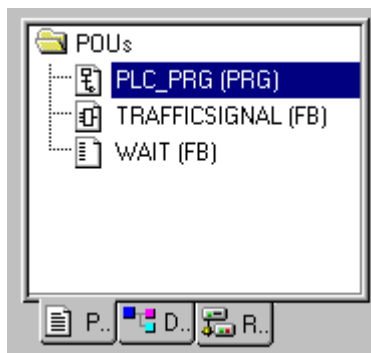
通过在一个符号上点击鼠标，你可以更快地选择一个命令。可以选择的符号将自动地与激活的窗口相适应。仅当鼠标在符号上点击然后释放时，才能执行命令。如果你将鼠标指针短时停留在工具条上的一个符号时，则在“Tooltip”（工具提示）中显示该符号的名称。为了查看工具条上的每个符号的说明，选择编辑器中有关你所需要的信息“Help”（帮助），然后点击你感兴趣的工具条符号即可。工具条的显示是可选择的（参见“Project”（项目）→“Options”（选项）桌面）。



对象管理器

“Object Organizer（对象管理器）”总是位于 TwinCAT PLC Control 的左边。在其底部有三个带有符号的属性页，用于三种类型对象：POU、数据类型以及资源。如果需要在相应的对象类型之间进行选择，可以用鼠标点击相应的属性页，或使用左、右方向键。

你将在“对象”章节中，了解如何使用“Object Organizer（对象管理器）”中的对象。



Object Organizer（对象管理器）

屏幕分隔器

屏幕分隔器是两个不重叠窗口之间的边框。在 TwinCAT PLC Control 中，“Object Organizer”（对象管理器）和主窗口“Work space”（工作区）之间，接口（声明部分）和 POU 的实现体（指令部分）之间以及“Work space”（工作区）和消息窗之间都有屏幕分隔器。

你可以使用鼠标指针来移动屏幕分隔器。按下鼠标左键并在屏幕上移动鼠标指针，就可移动屏幕分隔器。

必须明确的是，屏幕分隔器总是保持在其绝对位置上，即使窗口尺寸已发生变化时也是如此。如果感觉屏幕分隔器没有显示，只要简单地放大你的窗口即可。

“Work space”（工作区）

“Work space”（工作区）位于 TwinCAT PLC Control 中主窗口的右侧。所有编辑器的对象和库管理器都在这个区域中打开。有关编辑器的说明，请参见“TwinCAT PLC Control 编辑器”中“资源”和“库管理器”章节。在菜单项“Window”（窗口）下，你可以找到用于窗口管理的所有命令。

消息窗

消息窗通过主窗口中工作区下面的一个屏幕分隔器分隔。它包含以前的编译、检查或比较的所有消息。如果你用鼠标双击消息窗内的一条消息或按 <Enter>, 编辑器就会打开所选择行的对象。通过命令“Edit”（编辑）→“Next error”（下一个错误）以及“Edit”（编辑）→“Previous error”（上一个错误），你可以在各个错误消息之间快速跳转。消息窗的显示是可选择的（参见“Window”（窗口）→“Messages”（消息））。

状态条

在 TwinCAT PLC Control 中，位于主窗口框底部的状态条可为你提供有关当前项目和菜单命令的信息。如果一个项相关联，则状态条右侧将以黑体的形式出现，否则为灰体。当以联机模式工作时，则“Online”（联机）将以黑体表示。如果以脱机模式工作，则显示灰体。在联机模式，你可以从状态条中看到是否处于仿真（SIM），程序是否正在运行（RUNS），是否设置了一个断点（BP），或者变量是否强制（FORCE）。借助于文本编辑器，可以指示当前光标位置所在的行号和列号（例如行：5，列：11）。如果你已选择了一个命令，但尚未对它进行确认，则在状态条中会出现一个简短的描述。状态条的显示是可选择的（参见“Project”（项目）→“Options”（选项）桌面）。

上下文菜单快捷键：<Shift>+<F10>

取代使用菜单条执行命令，你可以使用鼠标右键。这时出现的菜单包含选择对象或激活编辑器所最常用的命令。可选择使用的命令，将自动地与激活的窗口相适应。

3.2 选项

对于 TwinCAT PLC Control，可能只有一个视图。然而，在 TwinCAT PLC Control 中，你可以配置主窗口视图（从而有不止一个视图）。此外，你还可以作出其它设置。为此，可以自由使用命令“Project”（项目）→“Options”（选项）。从而，你作出的设置（除非另有确定）保存在文件“TwinCAT PLC Control.ini”中，并在下次 TwinCAT PLC Control 起动时有效。

“Project”（项目）→“Options”（选项）

使用该命令可以打开用于选项设置的对话框。选项分为不同类别。通过鼠标点击对话框左侧或通过方向键选择所需要的类别，并在对话框右侧更改选项。

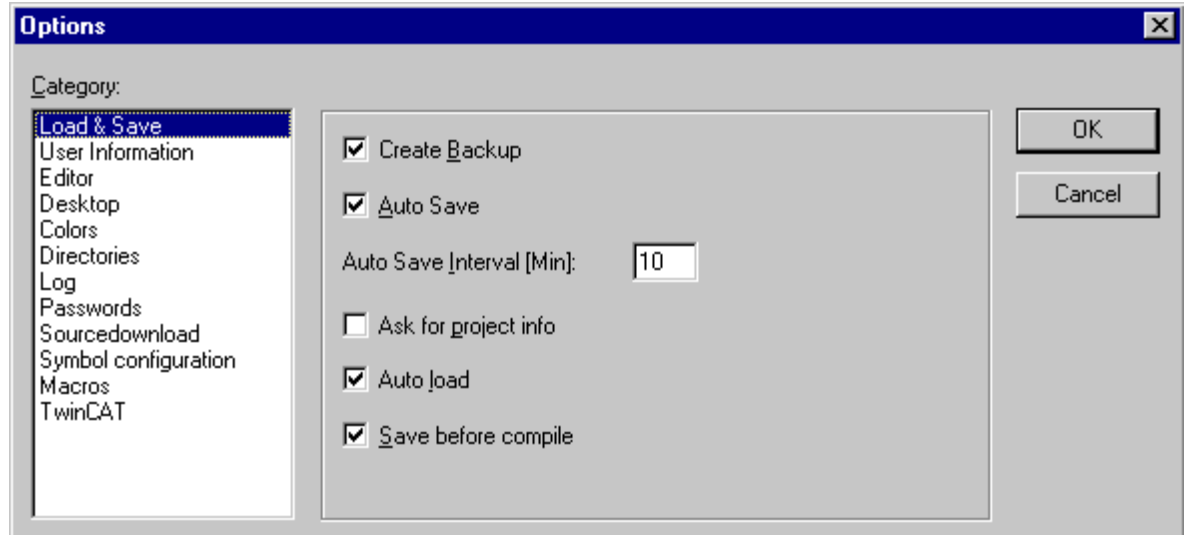
你可以任意选择以下类别：

- 装载和保存（Load & Save）
- 用户信息（User information）
- 编辑器（Editor）
- 桌面（Desktop）
- 颜色（Color）
- 目录（Directories）
- 日志（Log）
- 建立（Build）
- 密码（Passwords）
- 源代码下装（Sourcedownload）
- 符号配置（Symbol configuration）
- 项目资源控制（Project Source control）

- 宏指令 (Macros)
- TwinCAT

装载和保存

如果你选择该类别，则得到以下的对话框：



类别 Load & Save（装载和保存）的选项对话框

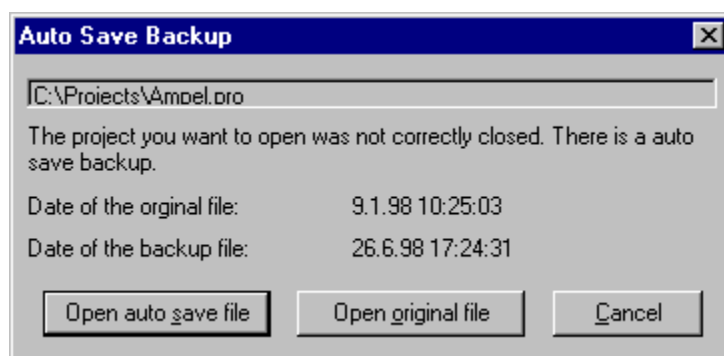
建立备份

当选中一个选项时，在选项前出现一个对勾（√）。如果你选择“Option”（选项）→“Create Backup”（建立备份），那么 TwinCAT PLC Control 就会在每次保存时使用扩展名“.bak”建立一个备份。使用这种方式，你总是能恢复上次保存之前的各种版本。

自动保存

若你选择选项 Auto Save（自动保存），则在工作时，你的项目会按照一个设置的时间间隔（Auto Save Interval，自动保存间隔）使用扩展名“.asd”不断地保存一个临时文件。这个文件将在一次正常退出时，从程序中删去。如果出于某种原因，TwinCAT PLC Control 非正常地停止运行（例如由于停电），则不会删去这个文件。

当你重新打开这个文件时，会出现以下屏幕：



现在，你可决定是否要打开原始文件或是打开自动保存的文件。

请求项目信息

如果你请求项目信息选项 **ASK**（请求），则当保存一个新项目，或在一个新名下保存一个项目时，自动调用项目信息。你可用命令“**Project**”（项目）→“**Project info**”（项目信息）使项目信息可视化，也可以处理这些信息。

自动装载

你若选择选项“**Auto Load**”（自动装载），则在下次启动 **TwinCAT PLC Control** 时，自动装载上次打开的项目。在启动 **TwinCAT PLC Control** 时装载项目，还可通过将项目输入到命令行来实现。

编译前保存

在每次编译之前保存项目。

退出时提醒引导项目：

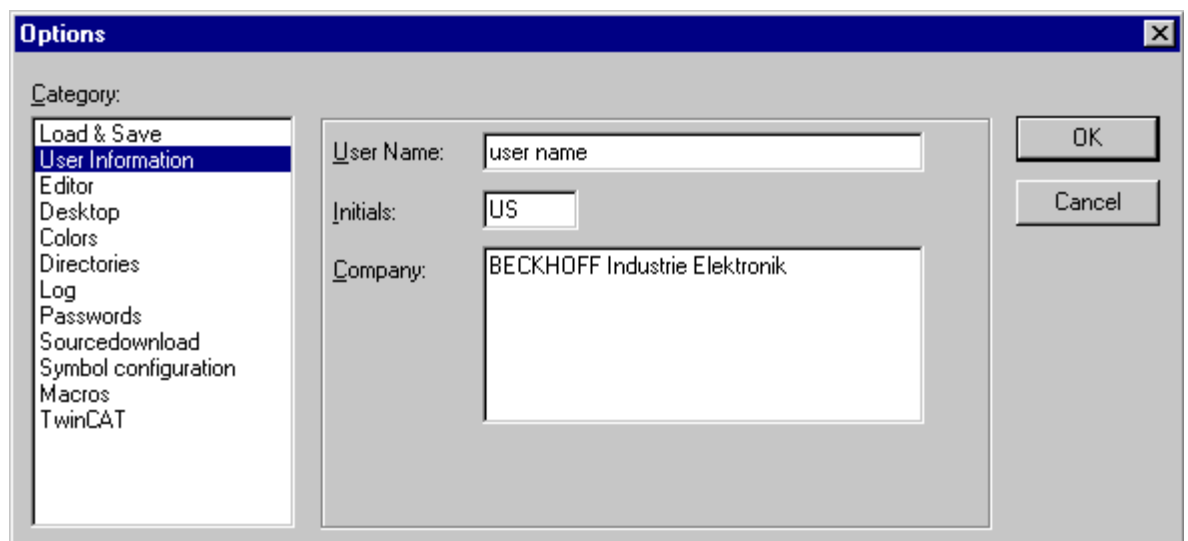
自上次装载一个引导项目以来，如果一个项目已被修改并装载，而没有建立一个新的引导项目，则在离开该项目之前，对话框会向你提出建议：“**No boot project created since last download.Exit anyway ?**”（自从上次装载以来未曾建立引导项目。是否退出）。

保存 ENI 用户认证信息：

用户名和密码，因为它们可以插入在用于 **ENI** 数据库的“**Login**”（登录）对话框内，将与项目一起保存。

用户信息

如果你选择该类别，则得到以下对话框：

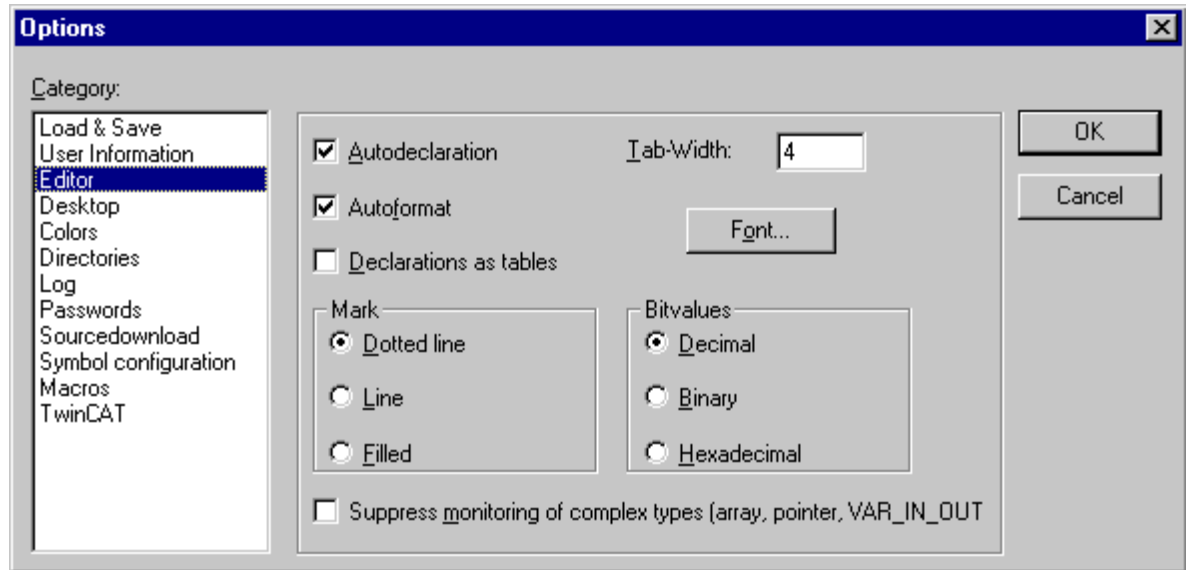


The screenshot shows a Windows-style dialog box titled "Options". On the left, there is a list of categories under the heading "Category:". The categories are: Load & Save, User Information (which is selected and highlighted in blue), Editor, Desktop, Colors, Directories, Log, Passwords, Sourcedownload, Symbol configuration, Macros, and TwinCAT. To the right of the list, there are three input fields: "User Name:" with the text "user name", "Initials:" with the text "US", and "Company:" with the text "BECKHOFF Industrie Elektronik". On the far right, there are two buttons: "OK" and "Cancel".

用户信息（**User information**）包括用户名、用户的原始数据（**Initials**）以及供职的公司（**Company**）。每个登入项都可修改，并与项目一起保存。

编辑器

如果你选择该类别，则得到以下对话框：



当选中一个选项时，在选项前会出现一个对勾（✓）。你可以为“Editor”（编辑器）进行以下设定：

- Autodeclaration（自动声明）
- Autoformat（自动格式）
- 表格式声明
- 制表符宽度
- 字型
- 文本选择显示
- 位值显示

Autodeclaration（自动声明）

如果你已选择 Autodeclaration（自动声明），那么（后跟一个尚未声明变量的输入），则在所有的编辑器中会出现一个对话框，应用这个对话框可以声明该变量。

Autoformat（自动格式）

如果已选择“Options”（选项）对话框“Editor”（编辑器）项中的选项“Autoformat”（自动格式），则 TwinCAT PLC Control 执行 IL 编辑器和声明编辑器的自动格式。

当你完成一行时，则进行以下的格式化：

- 以小写字母写出的操作符用大写表示；
- 制表符（Tab）插入到统一划分的列内。

成员列表：

若选中这个选项，则可利用“Intellisense”（智能感应）功能。这意味着，如果你在应插入一个标识符的地方插入一点，则会打开一个选择表，提供所有在项目中出现的全局变量。在编辑器、监视和接收管理器以及在抽样跟踪中，都可利用智能感应功能。

表格式声明

如果选择“Options”（选项）对话框中“Editor”（编辑器）项中的选项“Declarations as tables”（表格式声明），就可代替通常的声明编辑器来编辑一个表中的变量。这个表的布置如同一个索引属性表，其中有用于输入、输出、局部，以及输入/输出变量的属性页。对每一个变量而言，提供有“Name”（名称）、“Address”（地址）、“Type”（类型）、“Initial”（初始值）和“Comment”（注释）字段。

在“Options”（选项）对话框的“Editor”（编辑器）项中的“Tab-Width”（制表符宽度）字段，你可以确定在编辑器中制表符的宽度。默认设定值为四个字符，而字符宽度取决于所选择的字型大小。

禁止复杂类型监视（数组，指针，VAR_IN_OUT）：

如果选中这个选项，则数组、指针、VAR_IN_OUT 等复杂的数据类型将不再显示于联机模式时的监视窗内。

显示 POU 符号：

如果选中这个选项，则在 POU 框内显示各种符号（假定这些符号可在库文件夹内作为位映像使用）。位映像文件的名称必须由 POU 名和扩展名“.bmp”组成。例：用于 POU TON 的符号文件必需命名为 TON.bmp:

标记（Mark）：

在你的图形编辑器中，你可以选择虚线矩形（Dotted）、实线矩形（Line）或者有底纹的矩形（Filled）。在选择位置前出现一个点的地方选中选项。

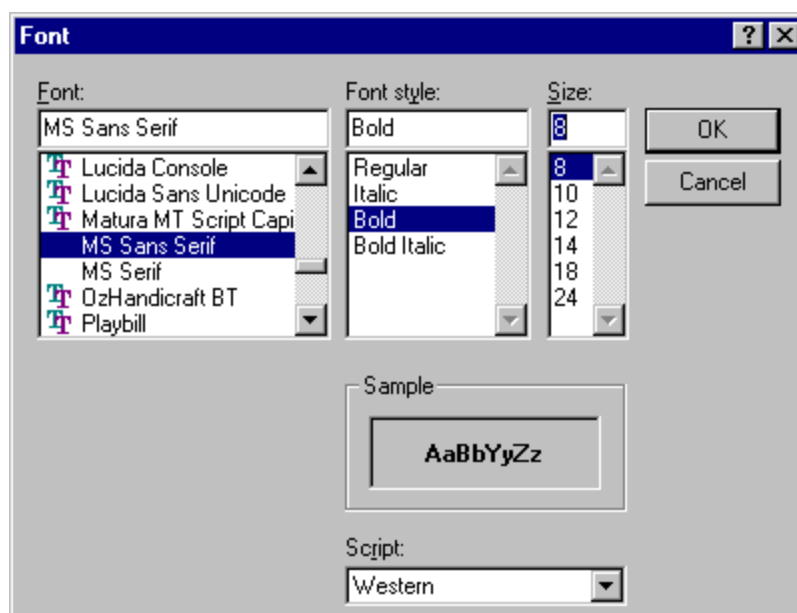
位值（Bitvalues）：

在监视过程中，你可以选择数据（类型 BYTE, WORD, DWORD）显示为十进制还是十六进制或二进制。在选择位置前出现一个点的地方选中选项。

字型（Font）

通过在“Options”（选项）对话框的“Editor”（编辑器）项中，点击“Font”（字型），可选择 TwinCAT PLC Control 编辑器中的所有字型。对所有制图操作而言，字型大小是基本单位。如果选择一个较大的字型，可以放大打印输出，即使是对于 TwinCAT PLC Control 的每个编辑器也是如此。

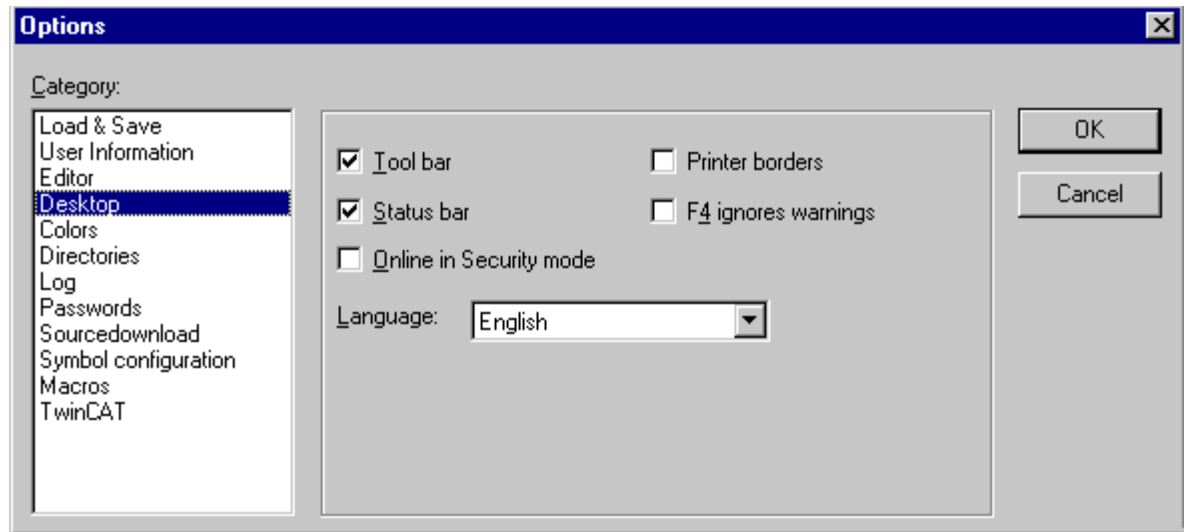
输入命令之后，打开字型对话框，以便选择字型、字体和字型大小。



用于设定字型的对话框

桌面 (Desktop)

如果你选择该类别，则得到以下的对话框：



“Desktop”（桌面）类别对话框选项

工具条

如果已选择工具条选项，带有快速选择命令按钮的工具条在菜单条下面变为可视。

状态条

如果已选择状态条选项，在 TwinCAT PLC Control 主窗口下边缘的状态条变为可视。

安全模式下的联机

在联机方式时，应用命令“Run”（运行）、“Stop”（停止）、“Toggle Breakpoint”（切换断点）、“Single cycle”（单循环）、“Write values”（写入值）、“Force values”（强制值）以及“Release force”（解除强制），则出现带有确认请求的一个对话框，确认是否要实际执行命令。

显示打印范围：

在每个编辑器窗口，用红色虚线标出当前设定的打印范围。其尺寸取决于打印机属性（纸张尺寸，朝向）以及设定打印布局的“Content”（内容）字段的尺寸（菜单命令：“File”（文件）→“Documentation Settings”（文档设置））。

F4 忽略报警：

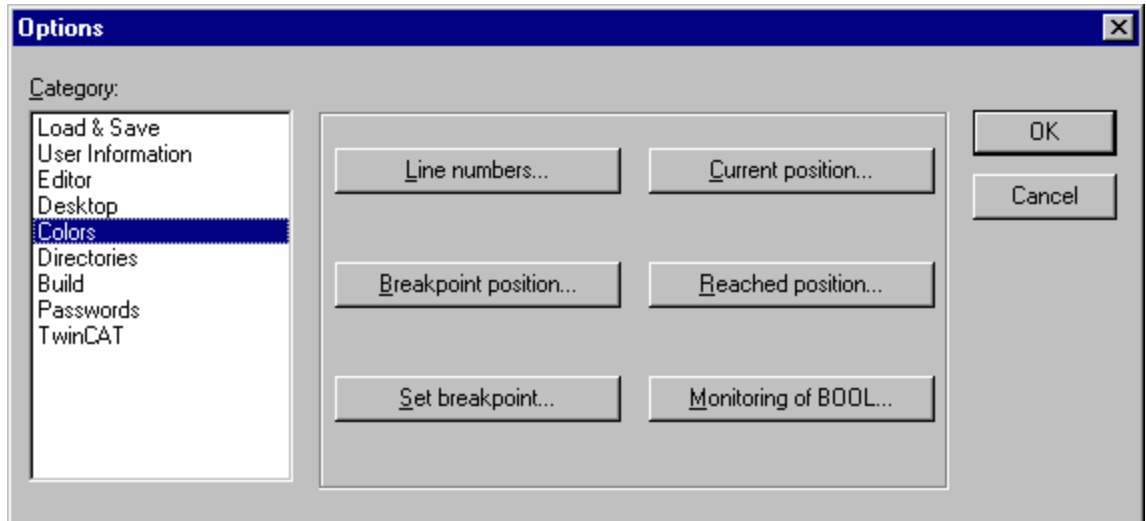
编译后，在一个消息窗内按 F4 时，则光标只跳转到有出错消息的行，忽略报警消息。

语言

在这里可以规定菜单和对话框文本应以何种语言显示。

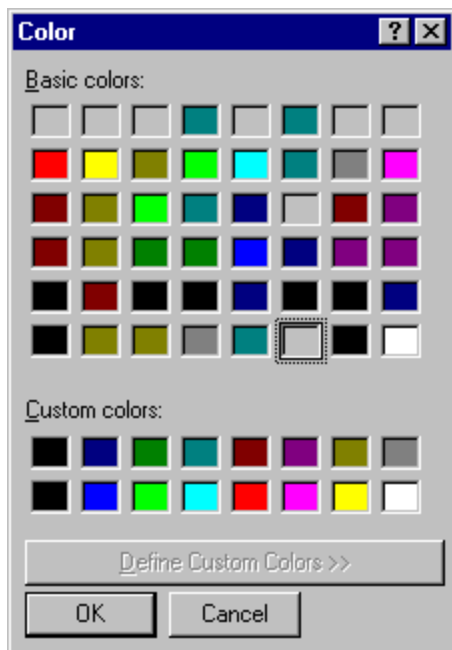
颜色

如果你选择该类别，则得到以下的对话框：



“Color”（颜色）类别的选项对话框

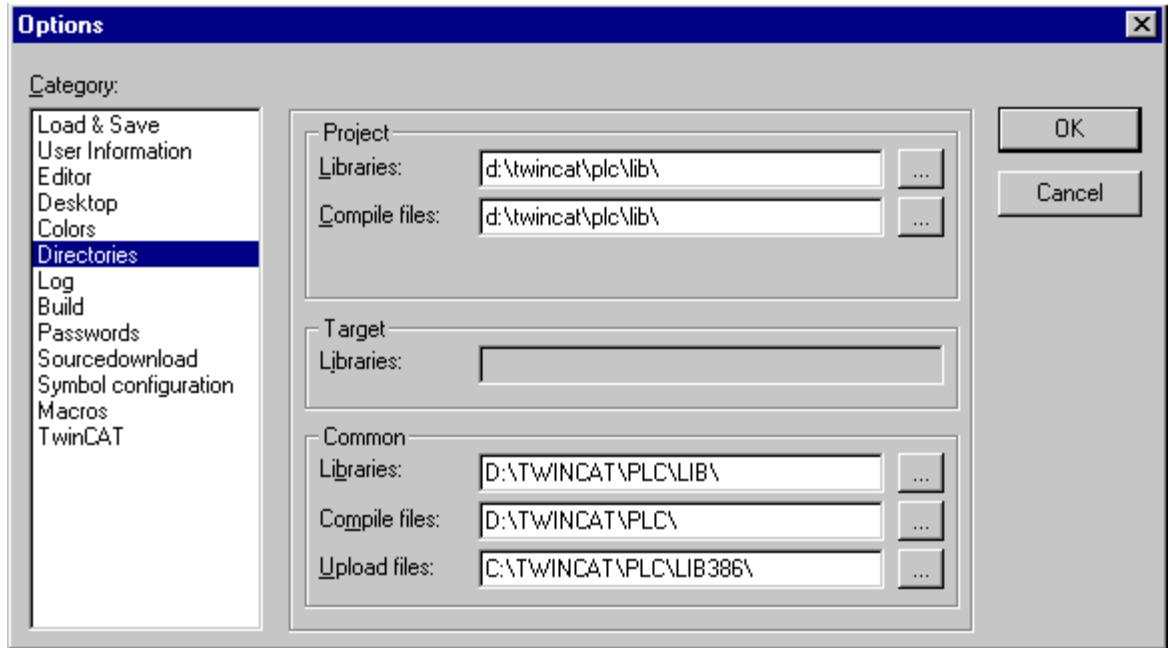
你可编辑 TwinCAT PLC Control 的默认颜色设定。可选择是否要为行号（默认设定：浅灰色）、断点位置（深灰色）、设定断点（浅蓝色）、当前位置（红色）、到达位置（绿色）或布尔值监视（蓝色）改变颜色设定。如果你已选择指示按钮中的一个按钮，则打开用于输入颜色的对话框。



用于选择颜色的对话框

目录 (Directories)

如果你选择该类别，则得到以下的对话框：



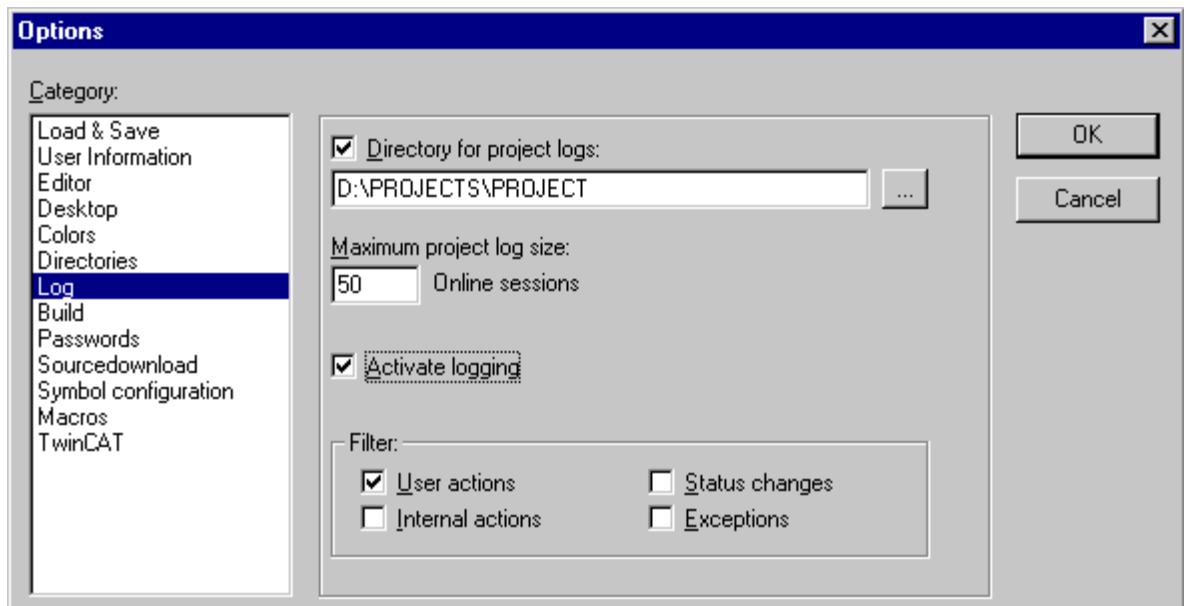
在“Project”（项目）和“Common”（公用）区内可进入“Directories”（目录），以便 TwinCAT PLC Control 可应用目录来搜索库和控制器配置文件，以及保存编辑和资源上传文件。如果你选中（...）字段后的按钮，则打开目录选择对话框。对于库和配置文件而言，可为每种文件提供多个路径，各个路径以分号“;”隔开。

“Project”（项目）区中的信息是以项目保存的；“Common”（公用）区的信息则写入到编程系统的初始文件，从而用于所有项目。

TwinCAT PLC Control 通常首先在“Project”（项目）中的目录中搜索，然后搜索进入到“Target System”（目标系统）中的目录（在“Target”（目标）文件中规定），最后搜索在“Common”（公用）下列出的目录。如果找到两个名称相同的文件，则使用在目录中首先搜索到的那个文件。

日志 (Log)

如果你选择该类别，则得到以下的对话框：



在这个对话框中，你可配置一个文件，其作用如同一个项目日志，记录联机方式运行期间所有的用户动作和内部过程。

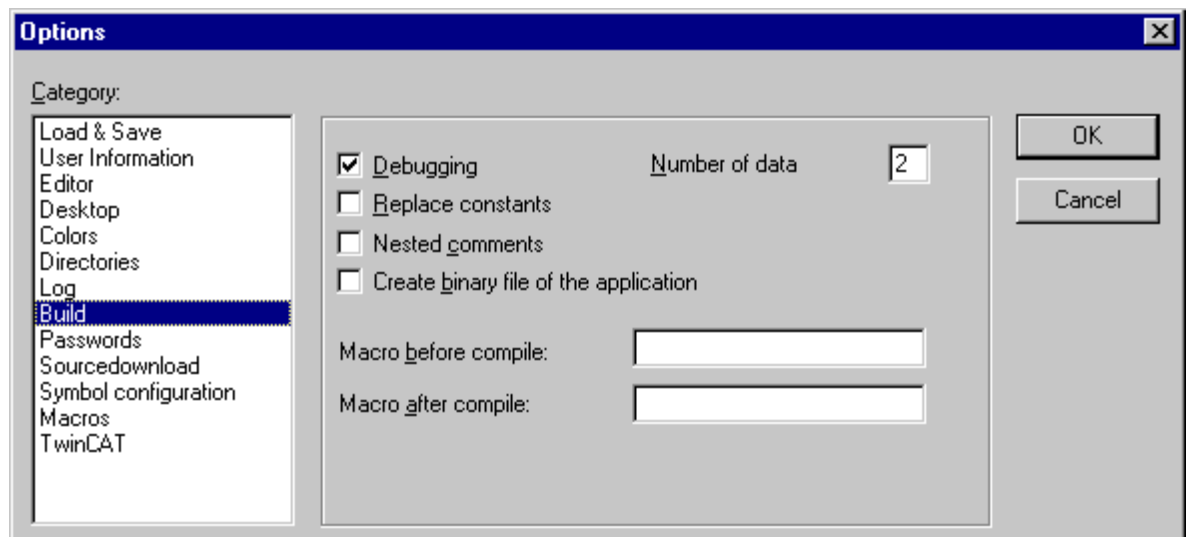
如果被打开的一个现有项目还未生成日志，则打开一个对话框，它提醒注意这一事实，现在正在建立日志，在下次记录过程后，它将接收其第一个输入。

当保存项目时，日志以二进制的文件自动地保存在项目目录内。如果你愿意使用不同的目标目录，你可为项目日志选中“**Directory**”（目录）选项，并将合适的搜索路径输入到编辑字段。为此，可用按钮访问“**Select Directory**”（选择目录）对话框。

使用扩展名（.log）自动将项目名分配给日志文件。要记录的“**Online sessions**”（联机事务）最大数量是由最大项目日志的大小所规定的。如果在记录过程中超过这个数，则删除最早的记录项，以便释放出空间用来记录最新的登入项。可在“**Option**”（选项）字段“**Activate logging**”（激活记录）内接通或断开日志功能。可在“**Filter**”（筛选）区选择要记录的是哪些动作：用户动作，内部动作，状态变化，异常情况等等。只有属于这些类别并经检验的动作，才能出现在“**Log**”（日志）窗口内，并写入到日志文件。

建立（Build）

如果你选择该类别，则得到以下的对话框：



“Build”（建立）对话框选项

注：

选项“**Debugging**”（调试）和“**Online changes**”（联机更改）的选择和取消选择并不影响当前应用 PC 机的 TwinCAT。它们可在 TwinCAT 选项下进行设置！对于用于总线端子控制器（Bus Terminal Controller）的 TwinCAT，这种设定可能会导致没有断点。当替换时，可提供更多的程序保存区。

当选择“**Replace constant**”（替代常数）选项时，每个常数值都是直接装载的，并且在联机方式，常数以绿色显示。此时，将不能对常数进行强制、写入和监视。如果选项被取消，则通过变量存取，可将值装载到一个保存位置（事实上，这样做允许写入变量值，但是系统需要更长的处理时间）。

如果“**Nested comments option**”（嵌套注释）选项是有效的，可将注释放置在其它的注释内。示例：

```
(*
a:=inst.out; (* 需要核对 *)
b:=b+1;
*)
```

这里，由第一个括号开始的注释并不是由“需要核对”后的括号关闭，而是只由最后的括号关闭。

建立应用程序的二进制文件

如果选中应用程序的选项“Create binary file of the application”（建立应用程序的二进制文件），则在编译过程中，在项目目录中建立生成代码（引导项目）的一个二进制映像。文件名：<project_name（项目名）>.bin。经过比较，使用命令“Online”（联机）→“Create Boot project”（建立引导项目），可在控制器上建立引导项目。当选中一个选项时，在选项旁会出现一个对勾（√）。

数据数量

通过输入“Number of data”（数据数量），你可确定，将在控制器内为项目数据保留多少空间。这个空间是需要的，以便当增加新的变量时，仍能完成“Online Change”（联机更改）。如果在编译过程中，出现消息“The global variables require too much storage”（全局变量需要更多的空间），则应增加你已在这里输入的数量。这种设定只对总线控制器 BCxxxx 这类产品有效。对 TwinCAT PC，数据段的数量是常数。保存量的大小可在 TwinCAT 选项下进行更改。

宏指令

为了对整个编译过程进行控制，可建立二种宏指令：一种是在编译过程前执行的“Macro”（宏指令），位于编译字段前的宏指令；另一种是在编译过程后执行的“Macro”（宏指令），位于编译字段后的宏指令。然而，以下的宏指令命令不能在这里使用：新建文件，打开文件，关闭文件，文件另存为，文件退出，联机，项目编译，项目检查，项目建立，调试，监视表。

“Build Option”（建立选项）对话框中的所有登录项都与项目一起保存。

密码（Passwords）：

如果你选择该类别，则得到以下的对话框：



The screenshot shows a dialog box titled "Options" with a list of categories on the left. The "Passwords" category is selected. The main area contains four password input fields with labels: "Password:", "Confirm Password:", "Write Protection Password:", and "Confirm Write Protection Password:". There are "OK" and "Cancel" buttons on the right side of the dialog.

类别“Password”（密码）的选项对话框

为了保护你的文件未经授权进入，TwinCAT PLC Control 提供一个密码选项，用于防止你的文件被打开或被改动。

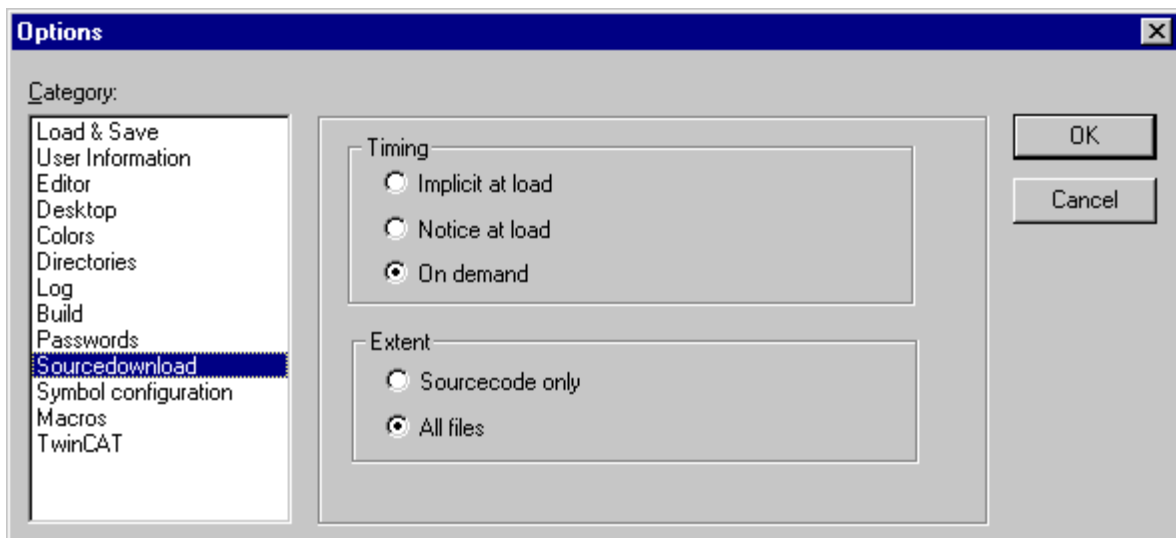
将所需要的密码输入到字段“**Password**”（密码）中。对于每个输入的字符，都将在该字段中出现一个相应的星号（*）。你还需在字段“**Confirm Password**”（确认密码）中重复相同的密码字。通过 **OK** 关闭对话框。如果你得到消息：“**The password does not agree with the confirmation**”（密码与确认密码不符），则在两次输入中，有一次你输入的字符可能有错误。在这种情况下，重复输入两次，直到对话框关闭而没有出错消息时为止。如果你现在保存文件然后将它打开，你会得到一个对话框，在框内要求你输入密码。因此，仅当你输入正确的密码时，项目才会打开。否则，TwinCAT PLC Control 报告：“**The password is not correct**”（密码不正确）。

与打开文件一样，你也可使用一个密码来保护文件不被改动。为此，你必须将一个密码输入到字段“**Write Protection Password**”（写入保护密码）中，并在下面字段内确认该输入。一个有写保护的项目可以不用密码打开。为此，如果在打开某一文件时，TwinCAT PLC Control 告诉你输入写保护密码，则只要简单地按一下“**Cancel**”（取消）即可打开。至此，你可编译项目，将它装载到 PLC 和进行仿真等等，但是你不能对它进行改动。

当然，很重要的一点是，你必须记住两个密码。然而，如果万一你忘了一个密码，则可与你的 PLC 制造商联系。密码与项目一起保存。为了建立各种不同的访问权限，你可使用菜单命令“**Project**”（项目）→“**Object access rights**”（对象访问权限）→“**Passwords for user groups**”（用于用户组别的密码），规定用户组别。

源代码下载（Sourcedownload）

如果你选择该类别，则得到以下的对话框：



你可选择在哪个“**Timing**”（时刻）和以什么“**Extent**”（范围）将项目装载到控制系统内。选项“**Sourcecode only**”（仅源代码）只包含在项目文件（extension .pro）内。选项“**All files**”（所有文件）还包括诸如相关的库文件、可视化位映像、配置文件等。

使用选项“**Implicit at load**”（隐含装载）能使所选择的文件范围，通过命令“**Online**”（联机）→“**Download**”（装载）自动地装载到控制系统内。

使用选项“**Notice at load**”（装载时提示）提供一个对话框，当使用命令“**Online**”（联机）→“**Download**”（装载）命令时，会提出问题“**Do you want to write the source code into the controller system?**”（你是否需要将源代码装载到控制系统内？）。按“**Yes**”（是），则自动地将所选择的文件范围装载到控制系统内；按“**No**”（否）则结束提问。

当使用选项“**On demand**”（根据要求）时，所选择的文件范围必须通过命令“**Online**”（联机）→“**Sourcecode download**”（源代码装载）确定装载到控制系统内。

保存在控制系统内的项目可通过命令“File”（文件）→“Open”（打开），从 PLC 打开项目以进行检查。在这一过程中文件将被解压缩。详见“File”（文件）→“Open”（打开）章节！

符号配置 (Symbol configuration)

注：这种形式的符号管理仅出于兼容性考虑，因而只用于 TwinCAT OPC 服务器。对于所有其它情况，有一个（在每次编译时）名为 <项目名>.tpy 的新建立的 XML 文件。

这里给出的对话框用来配置符号文件。它将作为一种文本文件 <项目名>.sym 建立，或项目目录内的二进制文件 <项目名>.sdb。

若选择选项“**Create symbol entries**”（建立符号登录项），则用于项目变量的符号登录项将在项目的每次编译时自动地在一个符号文件中建立。

另外，如果选中选项“**Dump XML symbol table**”（导出 XML 符号表），则在项目目录中还会建立一个包含符号信息的 XML 文件。它将命名为 <项目名>.SYM_XML。

以下的选项仅当选中选项“**Export variables of object**”（输出对象变量）时才起作用：

“**Export data entries**”（输出数据登录项）：用于存取全局变量的登录项是为对象的结构和数组而建立的。

“**Export structure components**”（输出结构成员）：一种专门的登录项，是为对象结构的每个可变成员而建立的。

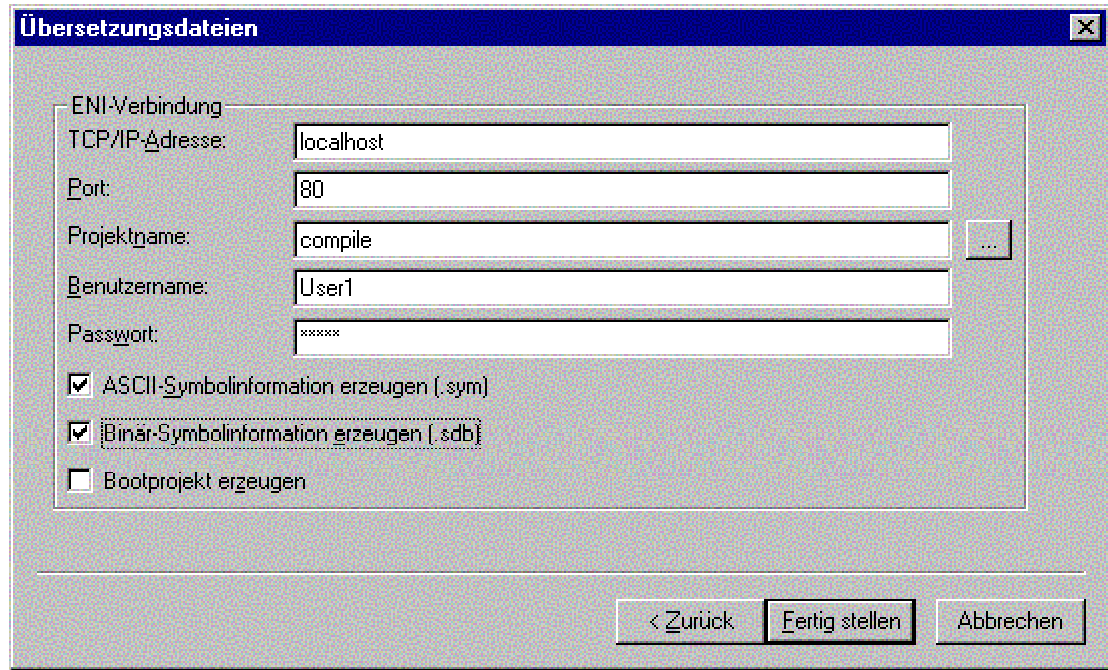
“**Export array entries**”（输出数组登录项）：一种专门的登录项，是为对象数组的每个可变成员而建立的。

“**Write Access**”（写存取）：对象的变量，可由 OPC 服务器更改。

一旦完成当前选择的用于 POU 选项的设定，还可选择其它 POU 并给出选项配置。这样可逐步完成任何所需数量的 POU 选择。当选择 OK 关闭对话框时，对话框打开以来所进行的所有配置均被确认。

项目资源控制

这个对话框是项目数据库设定选项的一部分（命令“Project”（项目）→“Options”（选项）→“Project source control”（项目资源控制））。由此，你可规定类别“Compile files”（编译文件）的对象在数据库中如何处理。（此外，还可利用两个其它的对话框为类别“Project objects”（项目对象）和“Shared objects”（共享对象）的对象规定这种处理。）



有关输入字段“**TCP/IP-Address**”（TCP/IP 地址）、“**Port**”（端口）、“**Project name**”（项目名），请参见对话框“**Project objects/Shared objects**”（项目对象/共享对象）的描述。

建立 ASCII 符号信息 (.sym)

建立 ASCII 符号信息 (.sym) 无论什么时候，若选中该选项，则建立一个符号文件 *.sym（文本格式）
建立二进制符号信息 (.sdb) 或 *.sdb（二进制格式），该文件将写入到数据库。符号文件中的登录项如同在“**Project**”（项目）选项类别“**Symbol configuration**”（符号配置）中所规定那样建立。

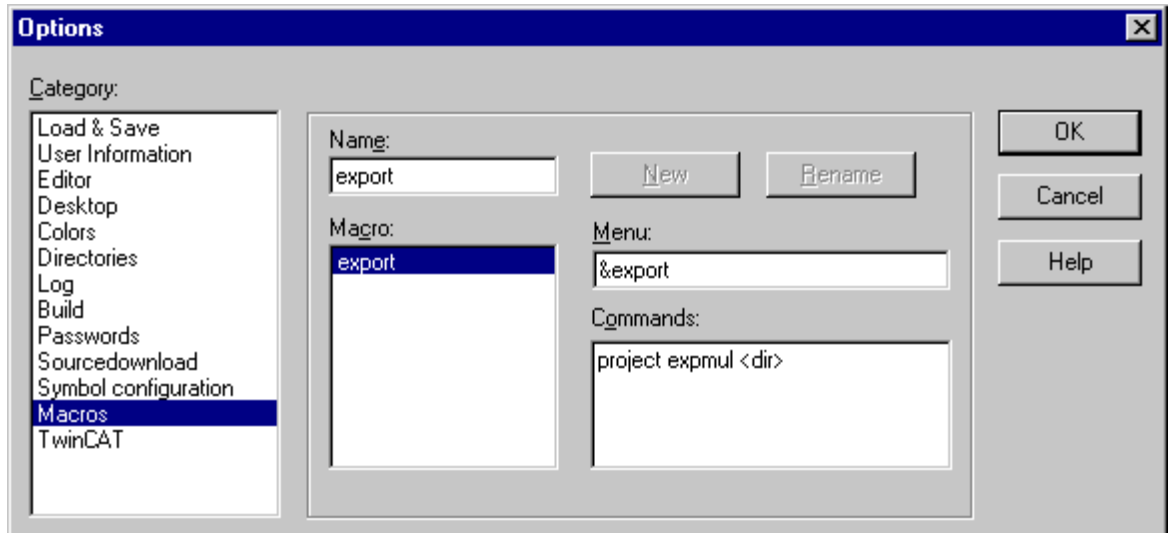
建立引导项目 无论什么时候，若选中该选项，则建立引导项目，这个文件将自动写入到数据库。

如果你正在进行初始配置，对话框就会逐个打开。在这种情况下，一个向导程序（**Wizard**）（按钮“**Next**”）会引导你，并且在第一个对话框中输入的设定值将会自动地复制到其他对话框；因此，如果你需要不同的值，你只要改变这些设定值即可。

“**Cancel**”（取消）将关闭对话框，而不保存在当前打开的对话框中所作的修改（然而保存以前对话框所作出的设定值）。返回到主对话框“**Options**”（选项）→“**Project source control**”（项目资源控制）。

宏指令 (Macros)

如果你选择该类别，则得到以下的对话框：



在该对话框中，可使用“batch mechanism”（批处理）的命令来定义宏指令，然后可在“Edit”（编辑）→“Macros”（宏指令）菜单内调用这些宏指令。

完成以下步骤可以定义一个新的宏指令：

1. 在输入字段“Name”（名称），输入一个用于建立宏指令的名称。按“New”（新建）按钮后，该名称会传输到“Macrolist”（宏指令表）字段内，并显示为已选中。宏指令表是以树形结构表示的。局部定义的宏指令逐一向下排列。如果宏指令库（见下面）是集成的，则库名称将列表，并且通过鼠标点击这些登录项前面的“+”或“-”符号，你可以打开或关闭一个库元素表。
2. “Menu”（菜单）字段用来定义菜单登录项，用它可使宏指令出现在“Edit”（编辑）→“Macros”（宏指令）菜单内。为了能将一个单独的字母作为一种快捷键，在这个字母前必须加一个符号“&”。
示例：名称“Ma&cro 1”生成菜单登录项“Macro 1”。
3. 在编辑器字段“Commands”（命令），可定义和/或编辑构成新建立的或新选择的宏指令的命令。所有批处理的命令以及对于批处理均有效的所有关键字都是允许的。通过按“Help”（帮助）按钮可获得一个列表。通过按 <Ctrl><Enter> 可选中一个新的命令行。按鼠标右键，可获得带有纯文本编辑器功能的快捷菜单。应用引号可将彼此有关的命令按组别组合在一起。
4. 如果你要建立其它宏指令，则在按 OK 按钮关闭对话框之前，再次完成以上步骤 1-3。

如果你要删除一个宏指令，则需将它从宏指令表中选出，然后按 。

如果你要重新命名一个宏指令，则将它从宏指令表中选出，在编辑字段“Name”中插入一个新名，然后按“Rename”（重命名）按钮。

如果要“edit”（编辑）一个现有的宏指令，则将它从宏指令表中选出，然后编辑字段“Menu”（菜单）和/或“Commands”（命令）。当按 OK 按钮时，将保存此修改。

只要按 OK 按钮关闭对话框，所有宏指令的实际描述将立即保存在项目内。

在“Edit”（编辑）→“Macros”（宏指令）菜单中的宏指令菜单登录项按其定义的次序显示。在作出菜单选择之前，不会测试宏指令。

宏指令库

宏指令可保存在外部宏指令库内。这些库可包含在其它项目内。

- 建立一个包含当前打开项目宏指令的宏指令库：

按“**Create**”（建立）按钮。得到对话框“**Merge project**”（合并项目），在那里列出所有可供使用的宏指令。选择所需要的宏指令项，然后以 OK 确认。所选择的对话框关闭，对话框“**Save Macrolibrary**”（保存宏指令库）打开。在这里为新库插入一个名称和路径，并按“**Save**”（保存）按钮。将建立名为 <库名>.mac 的宏指令库，然后关闭对话框。

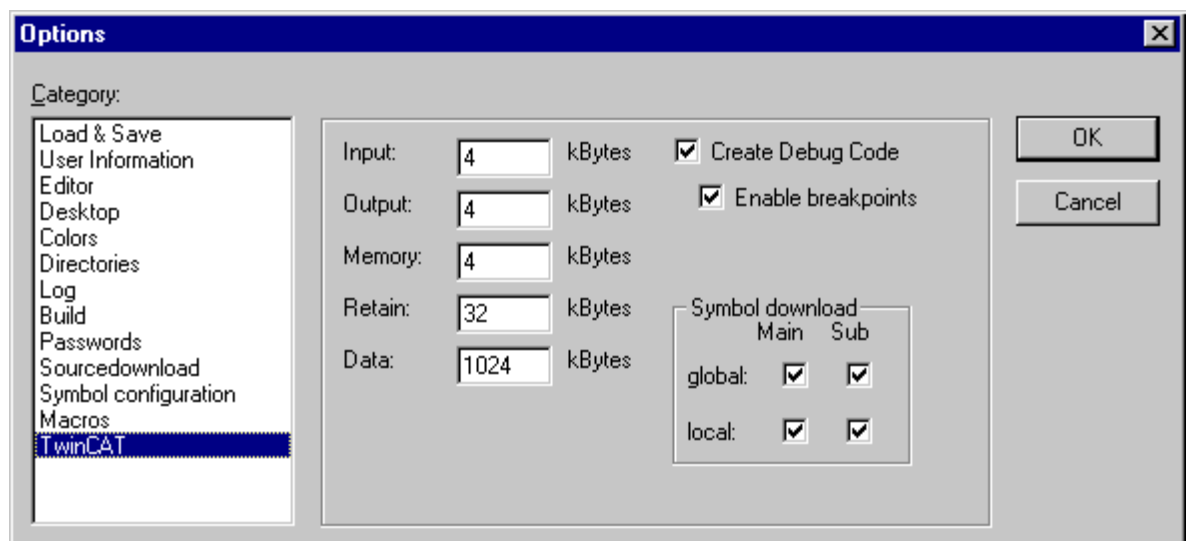
- 在当前打开的项目中包含一个宏指令库 <库名>.mac:

按“**Include**”（包含）按钮。将打开对话框“**Open Macrolibrary**”（打开宏指令库），显示带有扩展名 *.mac 的文件。选择所需要的库，然后按“**Open**”（打开）按钮。则对话框关闭，该库将会加到“**Macrolist**”（宏指令表）的树状结构内。

提示：还可导出一个项目的宏指令（命令“**Project**”（项目）→“**Export**”（导出））

TwinCAT

如果你选择该类别，则得到以下的对话框：



类别 TwinCAT 的选项对话框

使用该菜单，可设置用于所有输入、输出和标志变量的可编址空间大小。任何未分配的变量都位于这个空间区域的外面。右边的复选框可用来决定是否允许有断点。如果来自 PLC 的变量要由一个可视化系统通过名称读出的话，那么 PLC 的运行系统中的所有需要的变量名都必须是已知的。

通常建议采用动态符号。应用这种类型的符号管理，可对某种符号的请求在运行系统中动态装配符号。与静态符号管理相比，其优点可以减小所需要的空间，缩短编译和装载时间。处理次数限制在 8192 次。不需要的处理要重新卸载。

如果出于兼容性原因，要采用静态符号管理，则可在这个菜单内规定保存在 PLC 运行系统中的变量数。局部和全局变量名的管理可在矩阵的行中规定。“Main”只生成结构或字段名。“Sub”还为所有的结构或字段成员生成变量名。

示例：

```
abc: ARRAY[1..2] OF BOOL;
```

如果你只选择“Main”，则只能得到用于“abc”的符号。如果你选择“Sub”，你还可以得到用于成员“abc[1]”和“abc[2]”的符号。

3.3 项目管理

涉及整个项目的命令可在菜单项“File”（文件）和“Project”（项目）下找到。“Project”（项目）下的一些命令涉及对象，因而在“对象”一章中描述。

菜单“File”（文件）下的条目：

“File”（文件）→“New”（新建）

使用该命令，你可使用名称“Untitled”（无标题）建立一个空项目。在保存时必须改变这个名称。

“File”（文件）→“Open”（打开）

使用该命令，可打开一个已存在的项目。如果一个项目已打开和更改，则 **TwinCAT PLC Control** 就会询问这个项目是否需要保存。

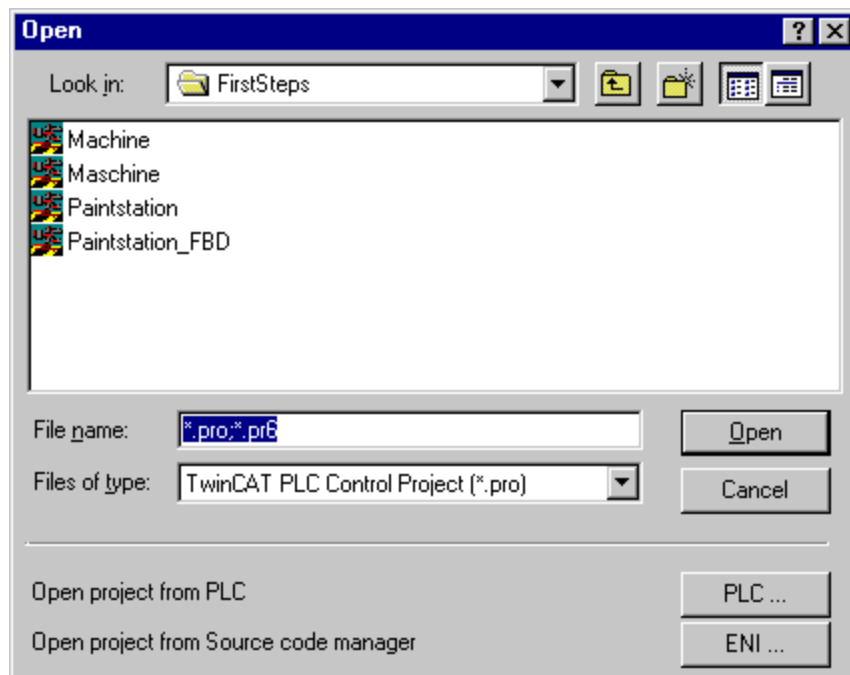
将出现用于打开文件的对话框，然后必须选择带有扩展名“*.pro”的一个项目文件，或带有扩展名“*.lib”的一个库文件。但该文件必须存在。使用命令“Open”（打开）不可能建立一个项目。

要将一个项目文件从 PLC 上装，则只需按“Open project from PLC”（打开来自 PLC 的项目）处的“PLC”。接着会得到对话框“Communication parameters”（通讯参数）（参见菜单命令“Online”（联机）→“Communication parameters”（通讯参数）），当尚未与 PLC 连接时，可以用来设定传输参数。一旦建立了联机连接，系统会检查计算机硬盘是否已存在相同名称的项目文件。当你收到对话框“Load the project from the controller”（装载来自控制器的项目）时，此时你可决定是否要将控制器正在使用的文件来代替本地文件。（该顺序与“Online”（联机）→“Load source code”（装载源代码）顺序相反，后者是将项目源文件保存在控制器内。不要与“Create Boot project”（建立引导项目）相混淆！）

注：

请注意，在任何情况下，当你将项目从 PLC 装载到你的本地目录时，都应给这个项目命名一个新名，否则它是未命名的。

如果尚未将项目装载到 PLC，你会得到一个出错消息。



在 TwinCAT PLC Control 中用于打开文件的标准对话框

使用选项“Open project from Source code manager”（从源代码管理器打开项目），可打开保存在 ENI 项目数据库中的一个项目。当你要访问一个作为数据库的 ENI 服务器时，这是一个先决条件。按“ENI”按钮…，得到一个对话框，在那里你可连接到与数据库类别“Project objects”（项目对象）有关的服务器。插入相应的访问数据（“TCP/IP-Address”（TCP/IP 地址），“Port”（端口），“User name”（用户名），“Password”（密码），“Read only”（只读））和数据库文件夹“Project name”（项目名），从中得到对象并用“Next”（下一步）确认。打开的项目对话框将被关闭，而另一个对话框将被打开，在那里要为数据库类别“Shared objects”（共享对象）插入存取数据。如果你按“Finish”（完成）按钮，对话框将关闭，并自动检索和显示被定义文件夹的对象。如果需要在数据库控制下继续保持项目对象，则打开“Project”（项目）选项对话框，设定所需要的参数。

最近打开的文件列在命令“File”（文件）→“Exit”（退出）下面。如果你选择其中之一，则打开这个项目。

如果项目定义了“Passwords”（密码）或“User groups”（用户组），则输入密码时会出现一个对话框。

“File”（文件）→“Close”（关闭）

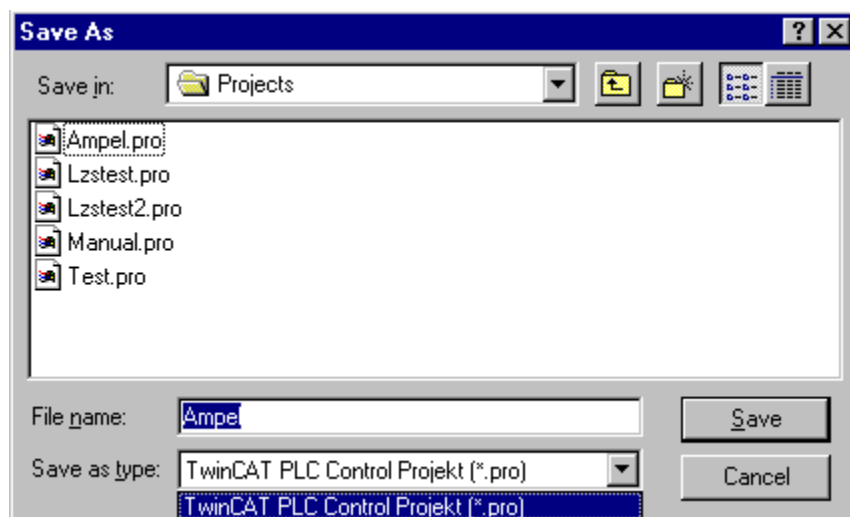
使用该命令，可以关闭当前打开的项目。若项目已经改动，则 TwinCAT PLC Control 会询问这些改动是否需要保存。若被保存的项目带有名“Untitled”（无标题），则必须为它起一个名（见“File”（文件）→“Save as”（另存为））。

“File”（文件）→“Save”（保存）（快捷键）：<Ctrl>+<S>

使用该命令，你可将任何改动的项目进行保存。如果要保存的项目为“Untitled”（无标题），你必须为它起一个名（见“File”（文件）→“Save as”（另存为））。

“File”（文件）→“Save as”（另存为）

使用该命令，可将当前项目作为其它文件进行保存，或作为库进行保存。这样做并不改变原始项目文件。在选择该命令后，出现“Save”（保存）对话框。此时，可选择一个现有的文件名，或输入一个新的文件名，并选择所需文件类型。



“Save as”（另存为）对话框

若项目需要一个新的名进行保存，则选择文件类型 TwinCAT PLC Control Project (*.pro)。若你选择文件类型 Project Version 2.7 (*.pro)，则当前项目如同它是以版本 2.7 建立那样进行保存。因而会丢失 2.8 版本的专有数据！然而，可用版本 2.7 执行项目。

你也可将当前项目作为库进行保存，以便将它用于其它项目。如果你已在 TwinCAT PLC Control 内编程你的 POU，则选择文件类型“**Internal library (*.lib)**”（内部库 (*.lib)）。

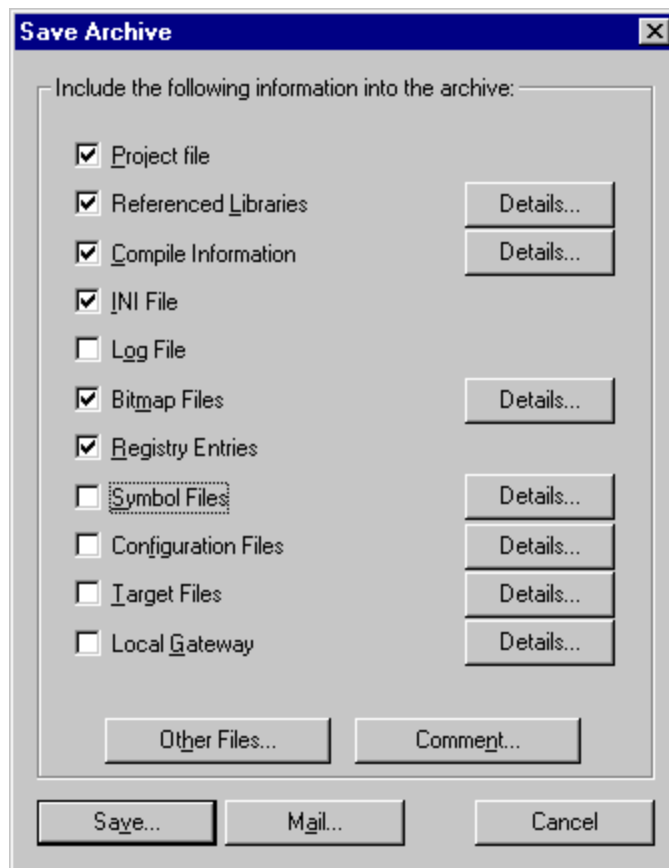
如需要集成由其它语言（例如 C 语言）实现的 POU，则选择文件类型“**External library (*.lib)**”（外部库）。这意味着，保存接收外部库文件名的其它文件，但带有扩展符“*.h”。这种文件的构成如 C 文件，带有对所有 POU、数据类型和全局变量的声明。然后点击 OK。

当前的项目保存在指定的文件内。若新文件名已存在，则会向你提示是否需要覆盖这个文件。

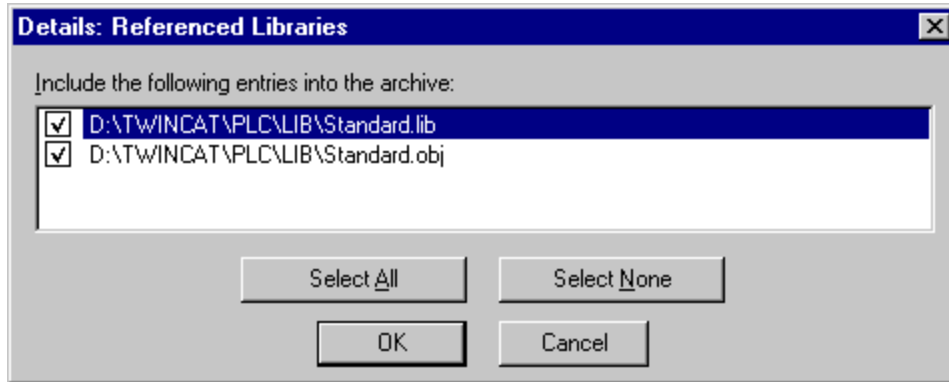
当“saving as a library”（另存为一个库）时，整个项目是编译的。如果此时出现错误，系统则会告诉你，必须具备一个正确的项目，以便建立一个库。从而该项目作为一个库进行保存。

“File”（文件）→ “Save/Mail Archive”（保存/将存档文件作为邮件发送）

使用该命令，可设置和建立一个项目存档文件。所有被引用以及与一个 TwinCAT PLC 项目一起使用的文件均可压缩在一个 zip 文件内。zip 文件可保存或直接通过电子邮件发送。如果需要转发一组所有项目相关的文件，该特性是很有用的。当执行该命令时，对话框“Save Archive”（保存存档文件）将打开：



在这里你可以规定哪一种文件类别应添加到存档 zip 文件内：通过选中/撤消相应的复选框来选择或取消选择一种类别。为此，可通过以鼠标单击复选框或双击类别名。如果一个类别被标记 ，则这个类别的所有文件都会加到 zip 文件；如果标记 ，则没有文件被添加。要选择某一类别的单个文件，可按相应的按钮“Details”（详细）。

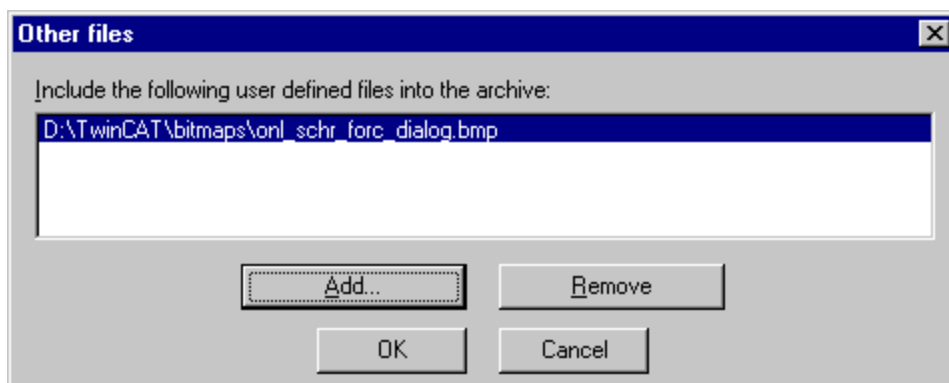


打开对话框“**Details**”（详细）则给出一个可供使用的文件表。在这个对话框中，可选择/取消选择所需文件：使用按钮“**Select All**”（选择全部）或“**Select None**”（不选择）来设置整个文件表。通过鼠标单击复选框，或双击表登录项，或当标记出表登录项时按空格键，可以选择/取消选择一个单独的文件。使用“**Save**”（保存）来关闭“**Detail**”（详细）对话框，以便保存新的设定。

在主对话框“**Save Archive**”（保存存档文件）中，没有选中所有文件的类别复选框将以灰色背景出现。提供有以下的文件类别，表的右侧表示哪一个文件可添加到 zip 文件。

类别	有关的文件
“Project File”（项目文件）	<项目名>.pro（TwinCAT PLC Control 项目文件）
“Referenced Libraries”（引用库）	*.lib, *.obj, *.hex（库，以及相应的对象和十六进制文件）
“Compile Information”（编译信息）	*.ci（编译信息），*.ri（装载/引用信息）<temp>.*（临时编译和装载文件），也用于仿真
“Log”（日志）	*.log（项目日志文件）
“INI File”（INI 文件）	TwinCAT PLC Ctrl.ini
“Registry Entries”（注册表登录项）	
“Symbol Files”（符号文件）	*.sdb, *.sym（从项目生成的符号信息）
“Log”（日志）	*.log（项目日志文件）
“Bitmap Files”（位映像文件）	*.bmp（用于项目 POU 和可视化的位映像）

如需要将任何其它文件添加到 zip，则按“**Other File**”（其它文件）按钮。将打开对话框“**Other files**”（其它文件），在这个对话框你可建立一张所需文件表。



按“**Add**”（添加）按钮，可打开标准对话框，以打开一个文件，在那里你可浏览文件。选择一个文件并以“**Open**”（打开）确认。该文件就会添加到“**Other files**”（其它文件）对话框中的表内。为每个你要添加的文件重复上述步骤。若要从表上删除登录项，按“**Remove**”（删除）按钮即可。当选择的文件表确定时，以 **OK** 关闭对话框。若要将一个 **Readme** 文件（自述文件）加到存归文件 **zip** 中，则按“**Comment**”（注释）按钮。此时将打开一个文本编辑器，可以输入文本。若以 **OK** 关闭这个对话框，在 **zip** 文件建立过程中，将会添加一个“**readme.txt**”文件。

除了输入的注释外，还包括有关建立日期的信息。

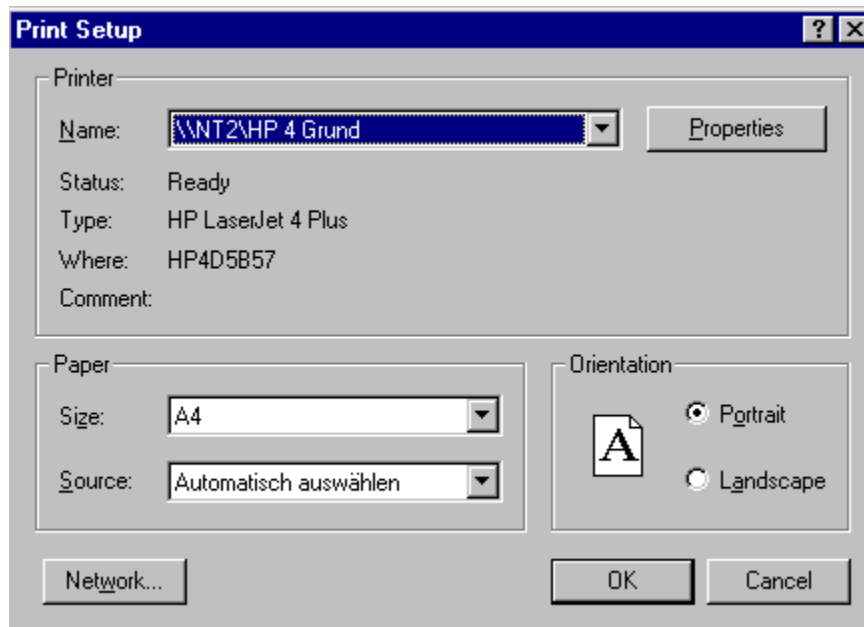
生成 zip 存档文件:

若已作出了所有需要的选择，在主对话框中按:

- “**Save...**”（保存），以便建立和保存 **zip** 文件：将打开用于文件保存的标准对话框，并可输入 **zip** 文件应保存的路径。**zip** 文件的文件名默认为 <项目名>.zip。使用“**Save**”（保存）确认，并开始建立该文件。在建立过程中，可显示当前的进程状态，并在消息窗中列出随后的步骤。
- “**Mail...**”（邮件），用来建立临时的归档 **zip** 文件并自动生成一个包括 **zip** 作为附件的空邮件。仅当 **MAPI**（Messaging Application Programming Interface，消息应用程序编程接口）已正确地安装在系统上时才有这个属性，否则会生成一个出错消息。在邮件准备过程中，显示进程状态并在消息窗中列出动作步骤。完成该动作后，自动撤消临时 **zip** 文件。
- “**Cancel**”（取消），用来取消动作；不生成 **zip** 文件。

“File”（文件）→ “Print”（打印）快捷键：<Ctrl>+<P>

使用该命令，可打印当前窗口的内容。选择该命令后，会出现“**Print**”（打印）对话框。选择所需选项或打印机，然后点击 **OK**。打印当前窗口。所有的编辑器都提供彩色输出。



打印对话框

你可确定“**number of the copies**”（复制数量），并将版本打印给一个文件。通过按“**Properties**”（属性）按钮，可打开对话框，设置打印机。你可用命令“**Document settings**”（文档设定）→“**File**”（文件）确定打印输出的布局。在打印期间，对话框向你显示已打印好的页数。当你关闭这个对话框时，在下一页后停止打印。为了归档整个项目，可使用命令“**Project**”（项目）→“**Document**”（归档）。若你要为项目建立一个文档框架，则可打开一个全局变量表，并使用命令“**Extras**”（附加）→“**Make Docuframe file**”（制作文档框架文件）。

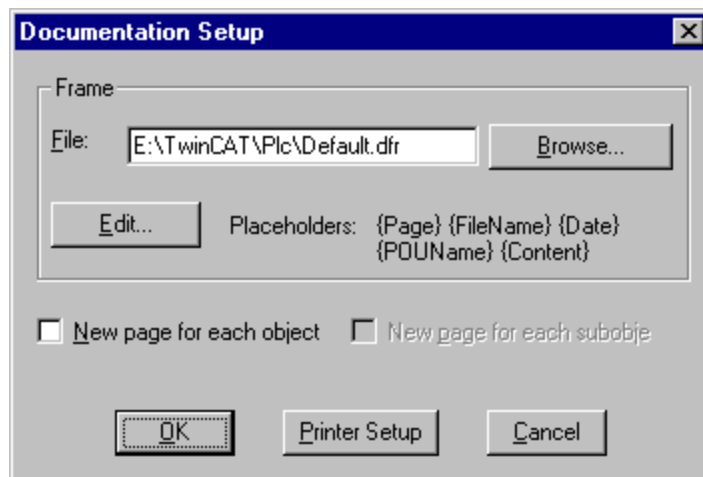
如果光标是在消息窗，则打印出所有的内容，逐行表示，并如窗口中所显示的那样。

可能打印的内容：

- 建立输出
- 交叉引用表
- 搜索结果
- 比较结果
- 批处理

“File”（文件）→“Printer setup”（打印机设置）

使用该命令，你可确定打印页面的布局。现在，打开以下对话框：



页面布局对话框

在字段“**File**”（文件）内，你可输入带有扩展名“.dfr”的文件名，在扩展名中应保存页面布局。设定的默认目标是文件 DEFAULT.DFR。

如需要改变一个现有的布局，则可经过目录树浏览，通过按“**Browse**”（浏览）按钮就能找出所需要的文件。

也可选择是否为每个对象和每个“**subobject**”（子对象）开始一个“**new page**”（新的页面）。

使用“**Printer Setup**”（打印机设置）按钮，打开打印机配置。若在“**Edit**”（编辑）按钮上点击，则出现用于建立页面布局框架。该框架可以确定页数、日期、文件名和 POU 名，也可将图形放置在页面上和应打印文档资料的文本区中。



在页面布局上粘贴用于占位符的窗口。

使用菜单命令“**Insert**”（插入）→“**Placeholder**”（占位符）和随后的五个占位符（“**Page**”（页），“**POU name**”（POU 名），“**File name**”（文件名），“**Date**”（日期），“**Content**”（内容））之间选择，通过在布局上拖动一个矩形同时按鼠标左键，可将一个占位符插入到布局内。在打印输出时，它们将如下替代：

命令	占位符	说明
页面	{Page}	这里出现打印输出时的当前页码。
POU 名	{POUName}	这里出现当前的 POU 名
文件名	{FileName}	这里出现项目名
日期	{Date}	这里出现当前的日期。
内容	{Content}	这里出现 POU 的内容。

此外，使用菜单命令“**Insert**”（插入）→“**Bitmap**”（位映像），可以将一个位映像图形（例如一个公司的标识）插入到页面内。选择图形后，可以使用鼠标将一个矩形拖放到布局的这个位置。若模板改变，则当窗口关闭时，TwinCAT PLC Control 会询问这些改变是否应保存。

“File”（文件）→“Exit”（退出）快捷键：<Alt>+<F4>

使用该命令，可退出 TwinCAT PLC Control。若一个项目被打开，则如同在命令“File”（文件）→“Save”（保存）中所描述那样关闭项目。

“Projects”（项目）命令可以提供的动作：

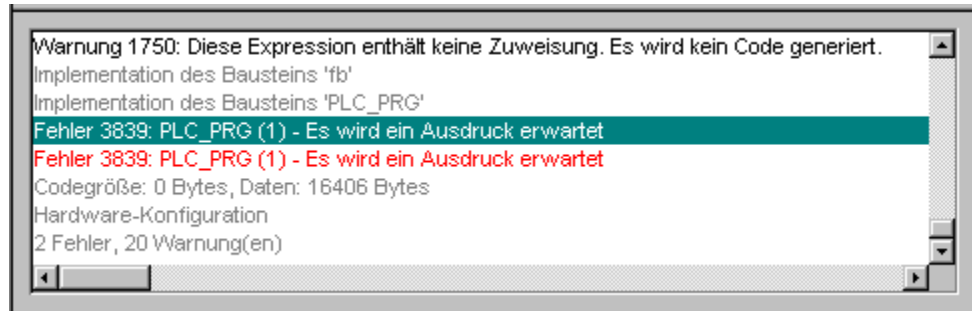
“Project”（项目）→“Build”（建立），<Ctrl>+<F8>

项目是通过命令“**Project**”（项目）→“**Build**”（建立）编译的。编译过程基本上是递增式的，也就是说只重新编译更改的 POU。如果初次执行命令“**Project**”（项目）→“**Clear all**”（全部清除），还可获得一种非递增式的编译。

对于支持联机更改的目标系统而言，在下次装载时要将所有 POU 装载到控制器中，在编译后使用蓝色箭头标记。

如果使用命令“**Online**”（联机）→“**Log-in**”（登录）连接控制器，则自动进行完成命令“**Project**”（项目）→“**Build**”（建立）的编译过程。

在编译过程中，将打开一个消息窗，显示编辑过程以及在编辑过程中也许会出现的任何错误和报警。错误和报警用编号标出。使用 F1，你可得到有关当前所选错误的更多信息。在附录中可以找到出错消息表。



若在“Load & Save”（装入和保存）类别的选项对话框中，选择选项“**Save before compilation**”（编译前保存），则在编译前保存项目。

注:

在编译过程中可建立交叉引用，并与编译信息一起保存。为了能够使用命令“**Show Call Tree**”（显示调用树）、“**Show Cross Reference**”（显示交叉引用）以及“**Project**”（项目）→“**Check**”（检查）菜单中的命令“**Unused Variables**”（未用变量）、“**Concurrent Access**”（权限并存）和“**Multiple Write Access on output**”（多重输出写存取）等，在任何更改后必须重新建立项目。

“Project”（项目）→“Rebuild all”（全部重建）

使用命令“**Project**”（项目）→“**Rebuild all**”（全部重建），不同于递增式的编译（“**Project**”（项目）→“**Build**”（建立）），项目是完全重新编译的。装载信息并不丢弃，而是如同用命令“**Clear All**”（全部清除）那样处理。

“Project”（项目）→“Clear all”（全部清除）

使用该命令，可全部清除从上次装载（装载信息）以来和从上次编译以来的信息。选择该命令后，将出现一个对话框，报告“**Online Change**”（联机更改）已不可能。这时，可以取消或者确认该命令。

注:

“**Clear all**”（全部清除）后，仅当带有上次装载的项目信息的*.ri 文件第一次显式地保存在项目目录之外时（见“装入装载信息”），才有可能出现 PLC 项目的上一次登录，并在登录之前，可以重新装入。

“Project”（项目）→“Load Download-Information”（装入装载信息）

使用该命令，可以重新装入属于项目的装载信息（如果这种信息保存在不同于项目所在的目录中的话）。选择了该命令后，可打开标准对话框“**File Open**”（打开文件）。

每次装载时，装载信息会自动地保存到一个文件内，该文件名为 <项目名> <目标标识符>.ri，并放入到项目目录内。项目打开时，文件装入，并在登录时用它来核对 PLC 项目是否适合于当前打开的 TwinCAT PLC Control 项目（标识符检查）。此外，它还可用来检查哪些 POU 的代码已进行更改。对于支持联机更改功能的系统，在联机更改过程中，只有这些有更改的 POU 才装入到 PLC。但是：如果项目目录中的*.ri 文件通过命令“**Project**”（项目）→“**Clean all**”（全部清除）所删除，你只能重新装入“**Download-Information**”（装载信息）（如果你已将*.ri 文件保存在其它目录中的话）。

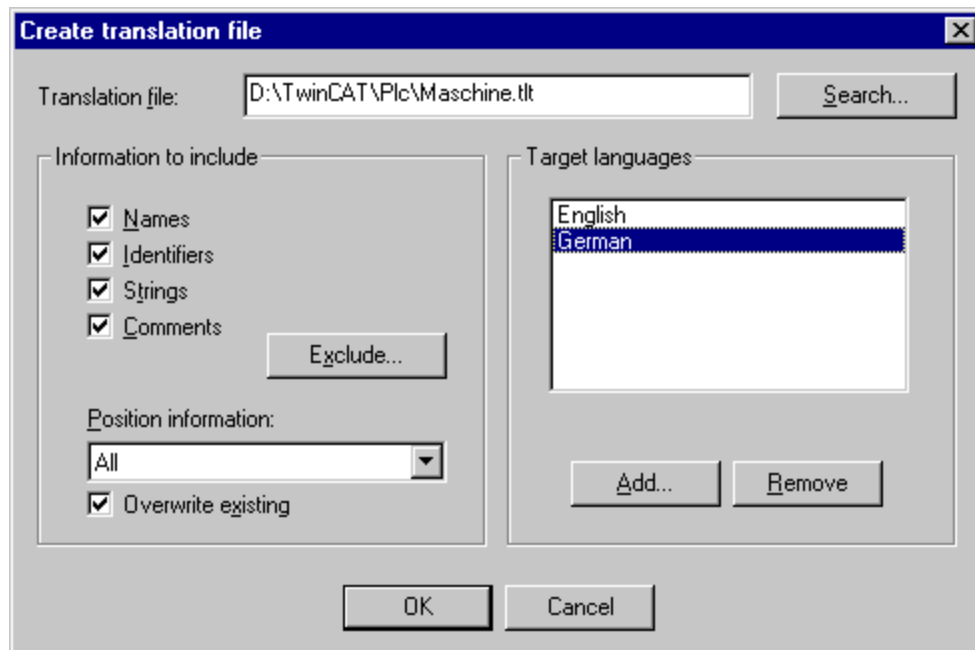
“Project”（项目）→ “Translate into another language”（翻译成其它语言）

使用该菜单命令，可将当前的项目文件翻译成其它语言。这是通过读入一个翻译文件执行的。该翻译文件由项目生成，并借助于文本编辑器从外部强化，翻译为所需要国家的语言。共有两个菜单子项：

- “Create translation file”（创建翻译文件）
- “Translate project”（翻译项目）

“Create translation file”（创建翻译文件）

菜单命令“Project”（项目）→ “Translate into another language”（翻译成其它语言）中的该命令可引导“Create translation file”（创建翻译文件）对话框：



在“Translation file”（翻译文件）字段，输入一条显示文件保存所在的路径。默认的文件扩展名是 *.tit；这是一个文本文件。也可以使用扩展名 *.txt。如果文件要与 EXCEL（电子表格软件）或 WORD（文字处理软件）一起使用的话，建议采用后一种扩展名即 *.txt。

如果已经存在你要处理的翻译文件，则可以为这个文件提供路径或使用“Search”（搜索）按钮搜索标准的 Windows 文件选择对话框。

来自项目的以下信息可以任选地传送给正在修改的或正在建立的翻译文件，以便将它们用于文件翻译：“Names”（名，例如在 Object Organizer（对象管理器）中的标题“POUs”），“Identifiers”（标识符），“Strings”（字符串），“Comments”（注释）。此外，也可传输用于这些项目元素的“Position information”（位置信息）。

如果相应的选项已检查，则来自当前项目的信息将作为语言符号输出到一个最近建立的翻译文件，或附加在一个已经存在的文件中。如果没有选择有关的选项，则属于其相应目录的信息（不论它来自哪一个项目）将从翻译文件中删除。

“Position information”（位置信息）：它规范描述文件路径、POU 以及排列语言符号位置，供翻译时使用。有三种选项可供选择：

“None”（无）：不产生项目信息

“First occurrence”（首先出现）：首先出现元素的位置附加到翻译文件内。

“All”（所有）：确定所有出现相应元素的位置。

如果要编辑的是早先建立的一个翻译文件，则该文件已包含比当前选择翻译文件更多的位置信息，不论该文件是从哪一个项目中生成，均应对它进行截短或删除。

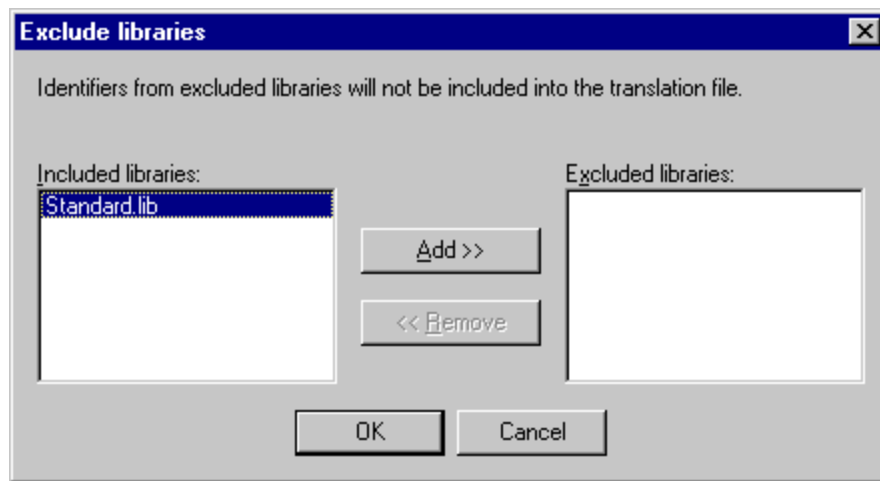
注：

每个元素（语言符号）最多可生成 64 个位置规范，即使用户在“Create Translation File”（创建翻译文件）对话框中，在“Position Information”（位置信息）下选择“All”（所有）亦是如此。

“Overwrite existing”（覆盖现有位置）：在翻译文件中，当前正在处理的现有位置信息，不论是由哪个项目生成的，都将覆盖。

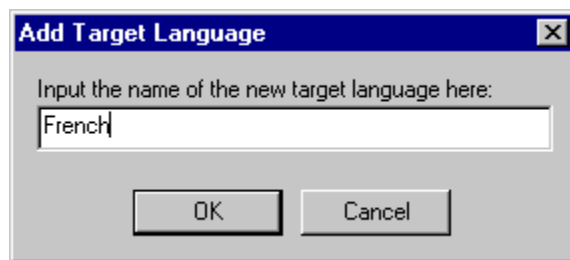
“Target languages”（目标语言）：这个表包含用于所有语言的标识符（这些标识符包含在翻译文件内），以及当完成“Create translation file”（创建翻译文件）对话框时要加入的标识符。

使用“Exclude”（排除）按钮，可打开“Exclude libraries”（排除库）对话框。



在这里，可选择包含至项目内的库，它们的标识符信息并未传送给翻译文件。为了完成这一工作，用鼠标选择左边“Included libraries”（包含的库）表中相应的登录项，并用“Add”（添加）按钮将它放置在右边“Excluded libraries”（排除的库）表。类似地，可用“Remove”（删除）按钮删除早已放置在那里的登录项。按 OK 按钮确认设定并关闭对话框。

使用“Add”（添加）按钮，可打开对话框“Add Target Language”（添加目标语言）：



必须在编辑器字段输入一个语言标识符，在该字段的开始或结束处不能有空格。

按 OK 关闭“Add Target Language”（添加目标语言）对话框，然后新的目标语言出现在目标语言表内。

使用“Remove”（删除）按钮，可从表中删除一个选择的登录项。

你也可以通过 OK 按钮确认“Create translation file”（创建翻译文件）对话框，以便生成一个翻译文件。

如果已经存在一个同名的翻译文件，你会得到以下有待回答“Yes”（是）或“No”（否）的确认消息：

"The specified translation file already exists. It will now be altered and a backup copy of the existing file will be created. Do you want to continue?" (“指定的翻译文件已存在。现在要对它进行改动并建立现有文件的备用拷贝。你是否要继续做？”）

如选择“**No**”，则返回，对“**Create translation file**”（创建翻译文件）的对话框不起作用。

如选择“**Yes**”，将在相同目录中建立带有文件名“**Backup_of_<translation file>.xlt**”（备份翻译文件）的现有翻译文件的拷贝，而相应的翻译文件将按照已输入的选项加以修改。

当生成翻译文件时会出现以下情况：

- 对每种新的目标语言，为每个要显示的语言符号生成一个占位符（“**##TODO**”）。
- 如果处理一个现有的翻译文件，出现在翻译文件内的语言文件登录项（但不在目标语言表内），不论是从哪个项目中生成的，都将被删除。

“Editing of the translation file”（编辑翻译文件）

必须打开翻译文件并作为文本文件保存。符号“**##**”用来标记关键字。文件中的“**##TODO**”占位符可被有效的翻译代替。对于每个语言符号，都会生成一个段落，该段落以“**##NAME_ITEM**”开头，以“**##END_NAME_ITEM**”结束。（对于注释，相应地使用“**##COMMENT_ITEM**”等等）。

参阅以下翻译文件段落的例子，该例子用于处理项目“**ST_Visu**”POU 的一个名。目标语言应是英语（美式英语）和法语。在这个例子中，已加上应翻译的项目元素的位置信息：

翻译前：

```
##NAME_ITEM
[D:\projects\Bspdt_22.pro::ST_Visualisierung:0]
ST_Visualisierung
##English :: ##TODO
##French :: ##TODO
##END_NAME_ITEM
```

翻译后

```
##NAME_ITEM
[D:\projects\Bspdt_22.pro::ST_Visualisierung:0]
ST_Visualisierung
##English :: ST_Visualization
##French :: ST_Visu
##END_NAME_ITEM
```

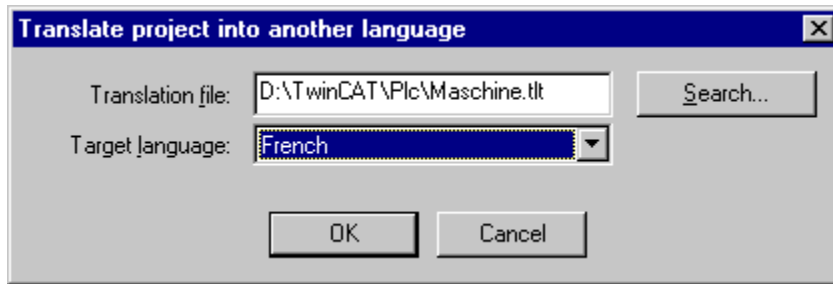
请检查经翻译后的“**Identifier**”（标识符）和“**Names**”（名）所涉及的标准是否依旧有效，并且字符串和注释是否在正确的括号内。

注：

以下的翻译文件部分，若没有详尽的了解不应改动：“**Language block**”（语言块），“**Flag block**”（标记块），“**Position information**”（位置信息），“**Original texts**”（原始文本）。

“Translate Project (into another Language)”（翻译项目，译成其它语言）

通过菜单命令“**Project**”（项目）→“**Translate into Another Language**”（翻译成其它语言），可打开“**Translate Project into Another Language**”（将项目翻译成其它语言）对话框。



若使用合适的翻译文件，可将当前的项目翻译成其它语言。

注:

如果你需要将项目版本以原始建立的语言保存，则在翻译之前应以不同名保存一份项目的拷贝。翻译过程不能被取消。

在字段“**Translation file**”（翻译文件）内，提供要使用的翻译文件的路径。通过按“**Search**”（搜索）按钮，你可进入标准的 Windows 文件选择对话框。

字段“**Target language**”（目标语言）包含输入到翻译文件内的一个语言标识符表，从中可选择所需要的目标语言。

使用 **OK** 按钮选中当前项目的翻译，借助于规定的翻译文件将当前项目译成所选择的目标语言。在翻译过程中，显示一个进程对话框，并显示出错信息（如有的话）。项目翻译后，将关闭对话框和关闭所有打开的项目编辑器的窗口。

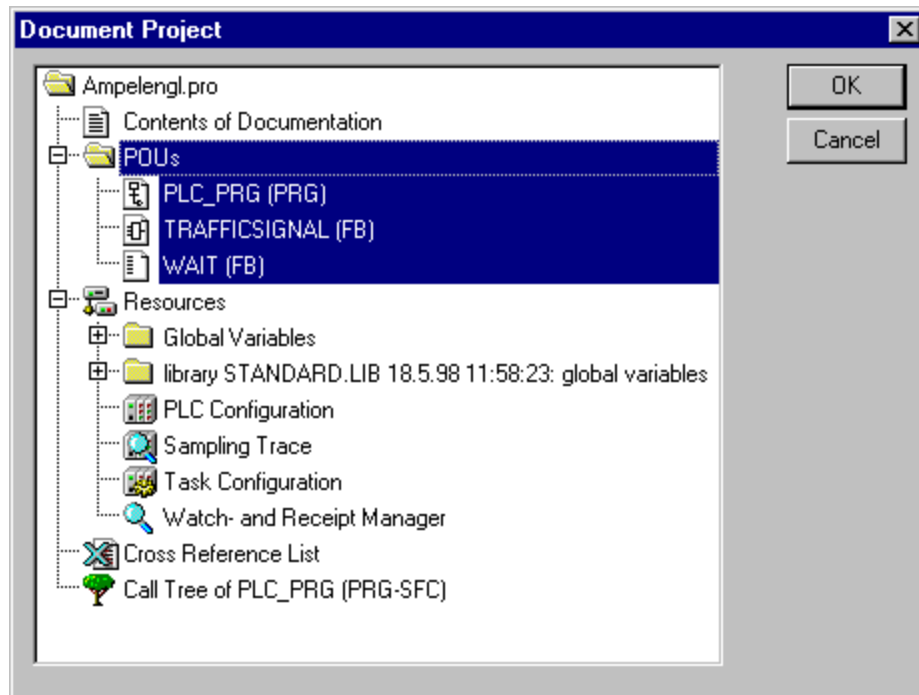
使用“**Cancel**”（取消）按钮，可关闭对话框而不修改当前项目。如果翻译文件包含错误的登录项，则在按 **OK** 后会显示出错信息，给出文件路径和相应的行，例如：`[C:\Programs\projects\visu.tlt (78)]; Translation text expected.`（期望的翻译文本）

“Project”（项目）→ “Document”（文档）

使用该命令，可以打印整个项目的文档文件。一份完整的文档文件诸要素包括：

- POU
- 文档文件内容
- 数据类型
- 资源（全局变量，抽样跟踪，PLC 配置，任务配置，监视和接收管理器）
- POU 的调用树和数据类型，以及
- 交叉引用表

对最后二项而言，项目建立时必须无错误。



项目文档文件对话框

只打印在对话框中以蓝色高亮度显示的区域。

如果你要选择整个项目，则选择第一行的项目名。另一方面，你只要选择一个单独的对象，则在相应的对象上点击，或者用方向键将虚线框矩形移到所需要的对象上。在对象符号之前带有一个加号的对象为包含有其它对象的复合对象。在加号（“+”）上点击，可以扩展复合对象，在负号（“-”）上点击，它会重新关闭。当你选择一个复合对象时，也就选择了所有相关的对象。通过按 <Shift> 按钮，你可选择一组对象，而通过按 <Ctrl> 按钮，你可选择几个不同的对象。

一旦你已作出选择，即可点击 OK。则出现“Print”（打印）对话框。然后使用菜单命令“File”（文件）→“Printer setup”（打印机设置）确定打印页面布局。

“Project”（项目）→“Export”（导出）

使用 TwinCAT PLC Control，可导入或导出项目。由此可实现在不同 IEC 编程系统之间交换程序。

有一种标准化的交换格式用于 IL、ST 和 SFC 语言编写的 POU（IEC 61131-3 的通用格式）。

对于以 LD 和 FBD 语言编写的 POU 和其它对象，由于在 IEC61131-3 中尚未列出其文本格式，因而 TwinCAT PLC Control 有其自身的填写格式。

将选择对象写入到一个 ASCII 文件内。

可以导出 POU、数据类型和资源。此外，在库管理器中，将信息链接到库的登录项也可以导出（不是库本身！）

重要事项:

如果在图形编辑器的注释中包含一个单引号（'），重新输入一个输出的 FBD 或 LD POU 会引起一个错误，这是因为它将解释为一个字符串的开始！

一旦你已在对话框窗口中作出了选择（如同使用菜单命令“**Project**”（项目）→“**Document**”（文档）那样），你可以决定，是否需要将选择的部分导出到一个文件或以单独的文件导出（每个对象一个文件）。选中或取消选择“**One file for each object**”（每个对象一个文件），然后点击 **OK**。则出现用于保存文件的对话框。相应于对象导出文件的一个目录，输入带有扩展名“**.exp**”的一个文件名，它将在目录中以文件名 <对象名.exp> 保存。

“Project”（项目）→“Import”（导入）

在结果对话框中，为打开文件而选择所需要的导出文件。数据输入到当前项目内。如果有同名的对象已经存在于相同的项目内，则会出现一个带有问题“**Do you want to replace it?**”（你是否要代替它？）的对话框：如果你回答“**Yes**”（是），则项目中的对象就被来自导入文件的对象所代替。如果你回答“**NO**”（不），则新对象的名将加上一个下划线和一个数字（例如：“**_0**”，“**_1**”，…）。回答 **Yes**、**all** 或 **No all**，为所有对象完成所有这类工作。

如果输入的信息用来链接一个库，则库将装载并附加到库管理器中的表格的末尾。如果这个库已经装载在项目内，则不再重新装入。然而，如果正在导入的导出文件显示库的保存时间不同，则与装载项目相类似，在库管理器中，以一个记号“*****”标记库名（例如：**standard.lib*30.3.99 11:30:14**）。若找不到库，则会出现一个信息对话框：“**Cannot find library {<path>\}<name> <date> <time>**”（找不到程序库{<路径>} <名> <日期> <时间>），如同装载一个项目那样。该导入登录在信息窗内。

“Project”（项目）→“Merge”（合并）

使用该命令，你可从其它项目将对象（POU，数据类型和资源）以及库的链接合并到你的项目内。当给出该命令时，首先出现用于打开文件的标准对话框。当你已在那里选出一个文件时，则会出现一个对话框，从中你可选择所需要的对象。选择过程如同在“**Project**”（项目）→“**Document**”（文档）所描述。如果有同名的对象已经存在于项目内，则新对象名会再加上一个下划线和一个数字（例如：“**_1**”，“**_2**”…）。

“Project”（项目）→“Compare”（比较）

该命令用来比较两个项目或将一个项目的实际版本与上次保存的版本相比较。

“Overview”（概述）：

在以下叙述中，名“**actual project**”（实际项目）用于当前正在使用的项目，名“**reference project**”（参考项目）用于已调用的项目，以便进行比较。在执行命令后，实际项目和参考项目将在一个分为二分格的窗口内以“**compare mode**”（比较模式）显示。发现有差异的 POU 名，以颜色标示出。对于编辑器 POU 以及 POU 的内容都以对等（**vis-a-vis**）方式显示。比较模式的结果和表示方式取决于：1. 进行比较，已选中何种筛选程序，在比较过程中，这对空白和注释的考虑会产生影响；2. 各行或各网络或各单元内的修改是否作为一种全新的插入 POU 来评价或不是这样评价。参考项目的版本可接受单个的差别或“**all equally marked**”（标记全部等同）的差别。接受意味着，以参考项目的版本取代实际项目。

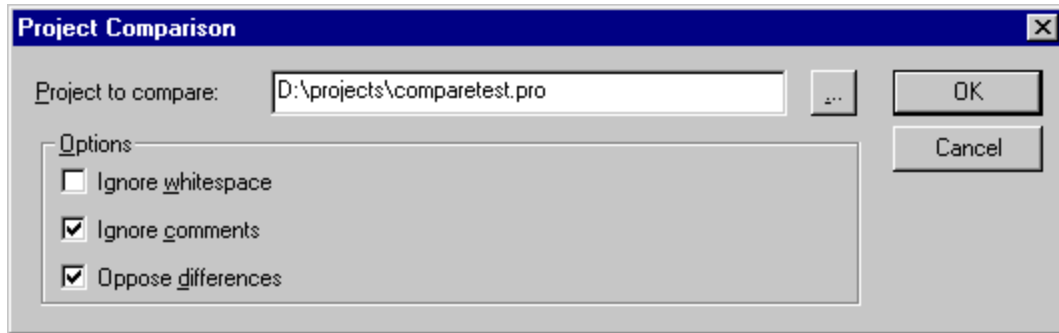
在一个消息上双击，可选择这个对象中的第一次改动。

注：

请注意：在比较模式（见状态条：**COMPARE**）项目不能编辑！

执行比较：

在执行命令“**Project**”（项目）→“**Compare**”（比较）后，则打开“**Project Comparison**”（项目比较）对话框：



在“**Project to compare**”（比较项目）处插入参考项目的路径。如需要以标准对话框打开一个项目，则按下按钮。若你插入实际项目名，则该项目的当前版本将与上次保存的版本进行比较。

若项目是在 ENI 数据库的资源控制下，则装载的版本可与数据库中的实际版本进行比较。为此，选中选项“**Compare with ENI-Project**”（与 ENI 项目比较）。

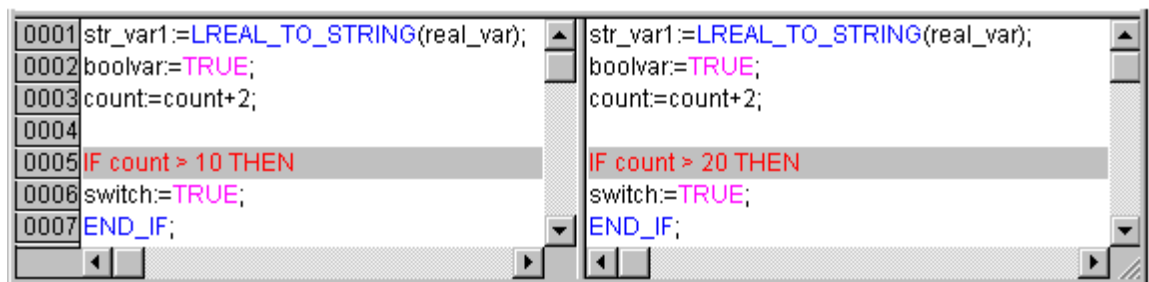
可以选中/取消与比较有关的以下一些选项：

- “**Ignore whitespaces**”（忽略空白）：对包含不同数量的空白，将检测为没有差异。
- “**Ignore comments**”（忽略注释）：对注释，将检测为没有差异。
- “**Oppose differences**”（对照差异）：若行、网络或 POU 中的元素已发生改变，在比较模式，它将与其它项目的版本相对照（用红色标出，见下面）直接显示在对半分开的窗口内。若取消这个选项，则在参照项目中，相应的行作为“**deleted**”（已删除）行显示；而在实际的项目中作为“**inserted**”（已插入）（蓝/绿色，见下面）行显示。这意味着，对其它项目中的相同行将不直接对照显示。

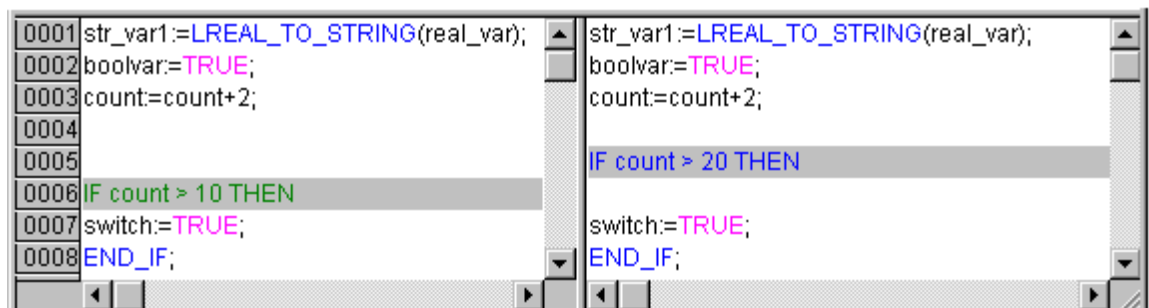
示例：

在实际项目中，行 0005（左侧）已修改。

Option 'Oppose differences' activated:



Option 'Oppose differences' not activated:

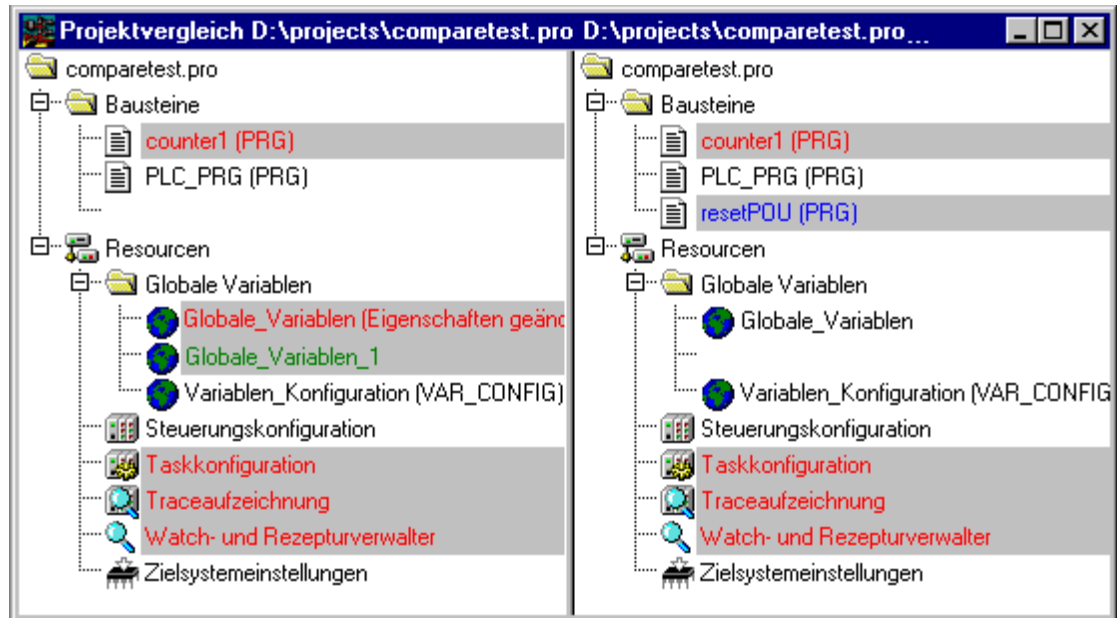


当按 **OK** 按钮关闭对话框“**Project Comparison**”（项目比较）时，比较将按照各个设定选项执行。

“Representation of the comparison result”（比较结果的表示）：

1. 比较模式中的项目概况：

执行项目比较后，打开一个对半分开的窗口，显示比较模式表示的项目。在标题栏中你可找到项目路径：“项目比较<实际项目路径> <参考项目路径>”。实际项目在窗口的左半部表示，参考项目则在右边表示。除了相应于对象管理器结构外，每个结构树在最上面位置表示项目名。



不同的 POU 以特定颜色的阴影标记和最终以一个附加的文本标出：

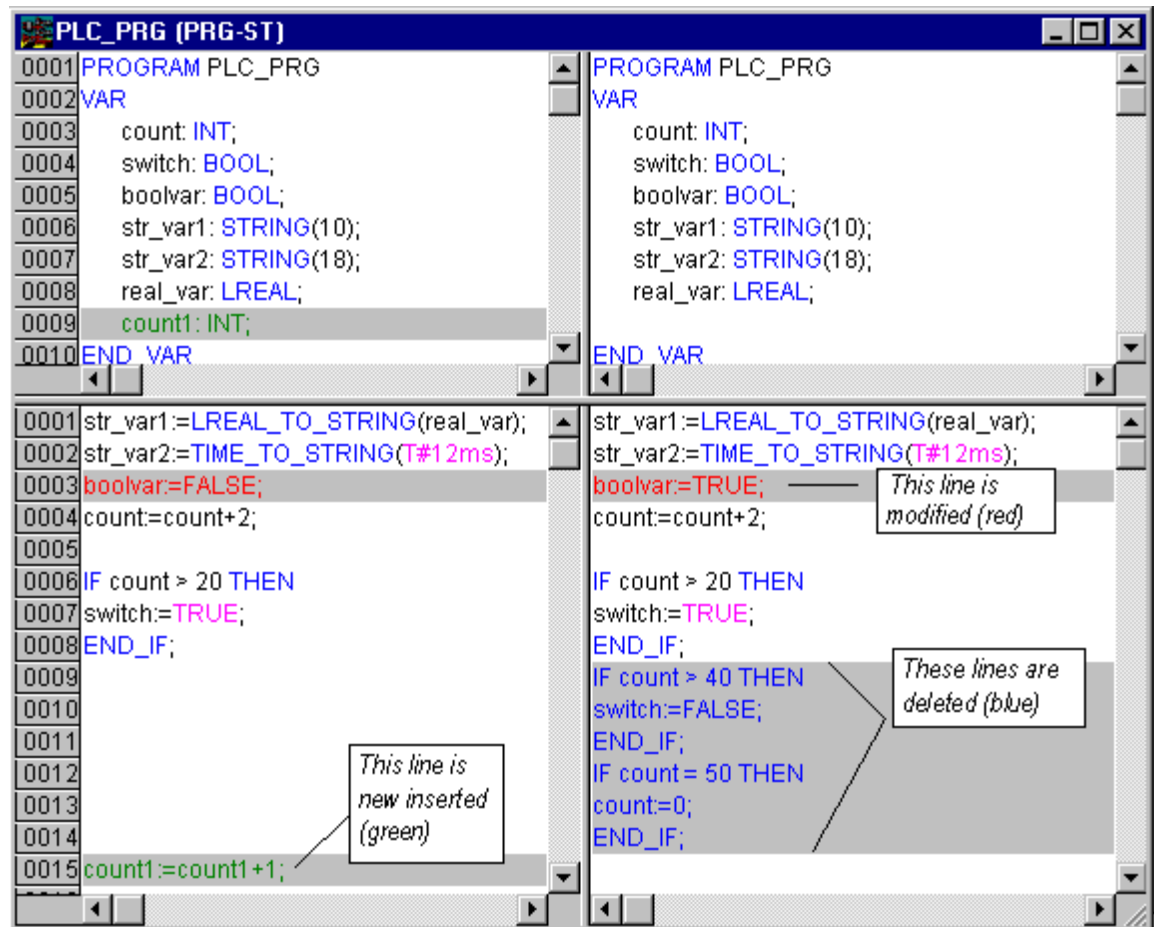
- 红色：已修改的单元；在窗口的两个对半分部分以红色字母显示。
- 蓝色：只能用于比较项目中的单元；在实际项目结构概况中的相应位置插入一个空格。
- 绿色：只能用于实际项目中的单元；在实际项目结构概况中的相应位置插入一个空格。
- 黑色：没有检测出差异的单元
- “**Properties changed**”（属性更改）：这个文本附加于项目结构树的 POU 名上，如果已检测出 POU 的属性有差异的话。
- “**(Access rights changed)**”（存取权限更改）：这个文本附加于项目结构树的 POU 名上，如果已检测出 POU 的存取权限有差异的话。

2. 比较模式中的 POU 内容：

通过双击结构概况中的行（由于有修改，因而该行以红色标出），则打开 POU。

如果这是一个文本或图形编辑器 POU，则以一个对半分开的窗口打开。参考项目（右侧）与实际项目（左侧）相对放置。在比较过程中被认为最小的单元是行（声明编辑器，ST，IL）、网络（FBD，LD）或元素（CFC，SFC）。如上所述，以相同的颜色用于项目概况。

示例：



如果不是一个编辑器 POU，而是任务配置、目标设置等等，则可在项目结构中双击相应的行，以不同的窗口打开实际项目和参考项目的 POU 版本。将显示没有更多详细差异的那些项目 POU。

比较模式工作（菜单“Extras”（附加），上下文菜单）：

如果将光标置于两部分窗口中菜单“Extras”指示差异的一个行上，上下文菜单提供以下命令选择，它取决于是否工作于项目概况或是工作于 POU。

“**Next difference**” <F7>（下一个差异）：光标跳转到其下一个单元，在那里指示有一个差异。（项目概况中的行，POU 中的行/网络/元素）

“**Previous difference**”（上一个差异）<Shift><F7>：光标跳转到上一个单元，在那里指示有一个差异（项目概况中的行，POU 中的行/网络/元素）

“**Accept change**”（接受更改）（<空格键>）：对于所有相关的单元（例如后续的各行），有相同类别的差异标记，则实际项目接受参考项目的版本。相应的单元将显示（带有相应的颜色）在窗口的左侧。如果它标记为红色（刚作更改）的一个单元，则通过实际项目中标记为黄色的文本使接受便于识别。

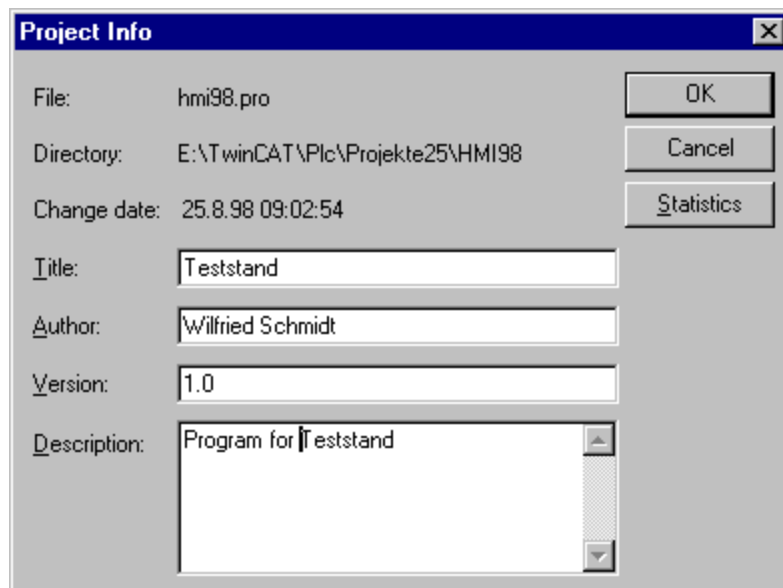
“**Accept changed item**”（接受更改的项）（<Strg><Space bar>）：只有当前放置光标的单个单元（行，网络，元素）能被实际项目接受。相应的单元将显示（带有相应的颜色）在窗口的左侧。如果它标记为红色（刚作更改）的一个单元，则通过实际项目中标记为黄色的文本使接受便于识别。

“**Accept properties**”（接受属性）（仅在项目概况中）：光标当前位置的 POU 对象属性，将如同它们在参考项目中所设置的那样为实际项目所接受。

“**Accept access rights**”（接受访问权限）（仅在项目概况中）：光标当前位置的 POU 对象访问权限，为实际项目所接受，如同它们在参考项目中所设置的那样。

“Project”（项目）→ “Project Info”（项目信息）

在该菜单项下，可保存有关项目的信息。给出该命令时，将打开以下对话框：



用于输入项目信息的对话框

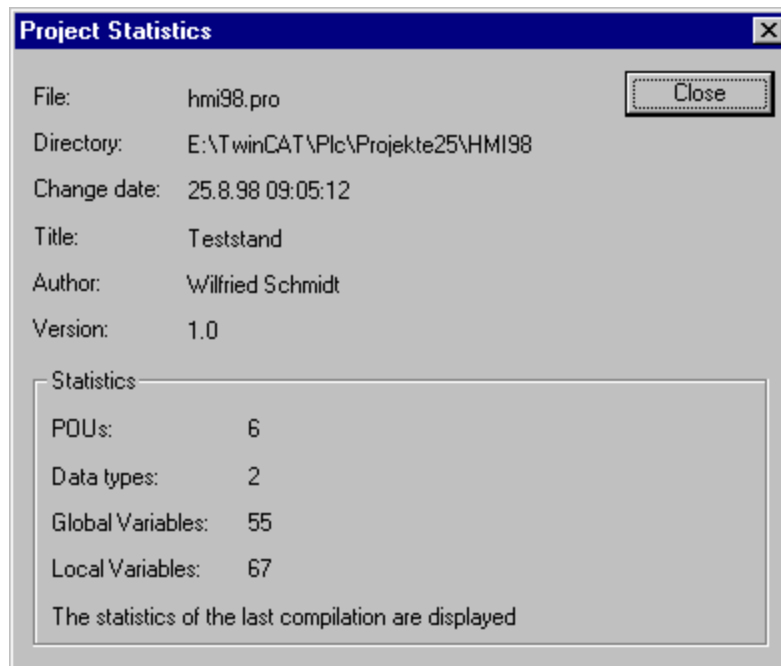
显示以下的项目信息：

- 文件名
- 目录路径
- 最近更改的时间（更改日期）

以上信息是不能改动的。此外，你还可附加以下信息：

- 项目标题
- 作者姓名，
- 版本号，以及
- 项目说明

以上信息为选项。当你按“Statistics”（统计）按钮时，你会收到有关项目的统计资料。它包括诸如 POU 数量、数据类型、以及在最后编译时所跟踪的局部变量和全局变量。



项目统计示例

如果你在选项对话框中的类别“**Load & Save**”（装载和保存）选择选项“**Ask for project info**”（请求项目信息），则当保存一个新项目，或在一个新名下保存一个项目时，会自动调用项目信息。

“Project”（项目）→ “Global Search”（全局搜索）

使用该命令，你可以在全局变量对象、PLC 配置、任务配置以及在库的声明部分搜索 POU 中文本位置、数据类型。

当输入该命令时，可打开一个对话框，从中你可选择所需要的对象。选择过程如同命令“Project”（项目）→ “Document”（文档）中所描述的那样。

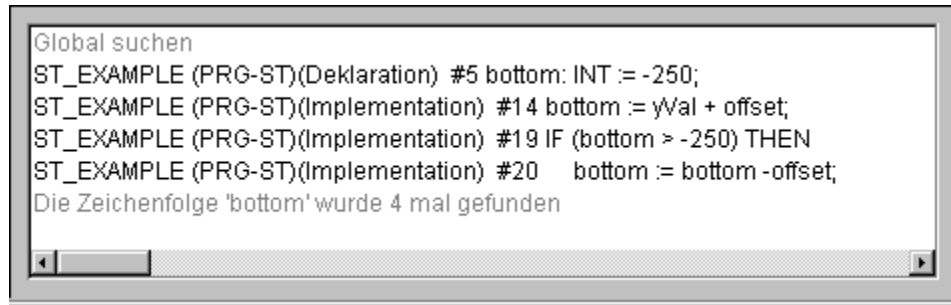
当你用 **OK** 确认选择时，则出现搜索对话框。

当通过菜单条中的符号调用“**Global Search**”（全局搜索）命令时，立即出现这个对话框；然后在项目的所有可搜索部分自动地完成搜索。通过“**Search for**”（用于搜索）字段的组合框可选择最近输入的搜索字符串。如果在一个对象中找到一个文本串，则将这个对象装载到相应的编辑器内，或装载到库管理器内，并显示找到的文本串的位置。找到的文本显示，以及搜索和找出下一个功能属性类似于命令“**Edit**”（编辑）→ “**Search**”（搜索）的行为属性。

如果你选择“**In message window**”（在消息窗中）按钮，则会在消息窗内，以表格形式在所选择对象中逐行列出要显示的全部位置，这些位置是由一系列符号来搜索的。之后，显示找出的位置数量。

如果报告窗没有被打开，则显示报告窗。对于所找到的每个位置，显示以下信息：

- 对象名
- 在声明（Decl）中找到的或在 POU 实现体（Impl）部分找到的位置，以及（如有的话）网络号。
- 文本编辑器中的整个行
- 图形编辑器中的完整文本单元



如果你用鼠标双击消息窗内的一行或按 <Enter> 按钮, 则打开装载对象的编辑器。并标出与对象有关的行。使用功能键 <F4> 和 <Shift>+<F4>, 可以在显示行之间快速跳转。

“Project” (项目) → “Global Replace” (全局替换)

使用该命令, 可以搜索 POU 中的文本位置、数据类型或全局变量的对象, 并用其它文本代替该文本。这些工作的执行如同使用菜单命令 “Project” (项目) → “Global Search” (全局搜索) 或 “Edit” (编辑) → “Replace” (替换) 那样。然而, 没有提供用于选择的库, 并且在消息窗内可能也没有显示。

“Project” (项目) → “Check” (检查)

使用该命令, 可以打开以下命令的子菜单:

- Unused Variables (未使用的变量)
- Overlapping memory areas (重叠的存储区)
- Access conflict (存取冲突)
- Multiple writes to output (输出多重写操作)

结果将显示在消息窗内。

这些功能的每一种都可用于测试最近的编译状态。因此, 在可以进行测试之前, 项目必须至少无错误地编译一次; 否则菜单项为灰色显示。

结果将显示在消息窗内。

未使用的变量:

该功能可以用于搜索已声明但尚未用于程序中的变量。这类变量以 POU 名和行输出, 例如: PLC_PRG (4) – var1。库中的变量将不作检查。

结果将显示在消息窗内。

重叠的存储区:

使用该功能, 可以测试在特定的存储区, 通过 “AT” 声明的变量位置是否出现重叠。例如, 当分配变量 “var1 AT %QB21 : INT” 和 “var2 AT %QD21 : DWORD” 时就出现重叠, 这是因为二者都使用字节 21。

这样, 输出如下:

```
%QB21 is referenced by the following variables:
PLC_PRG (3): var1 AT %QB21
PLC_PRG (7): var2 AT %QD21
```

结果显示在消息窗内。

存取冲突:

使用 “Project” (项目) → “Check” (检查) 菜单中的这个功能, 可以搜索由不止一个任务引用的存储区。这里, 对读存取和写存取不作区分。输出 (示例) 如下:

```
%MB28 is referenced in the following tasks :
Task1 - PLC_PRG (6): %MB28 [read-only access]
Task2 - POU1.ACTION (1) %MB28 [write access]
```

结果显示在消息窗内。

输出多重写操作:

使用“Project”（项目）→“Check”（检查）菜单中的这个功能，可搜索存储区，在这类存储区，一个单独项目在不止一个地方获得写入存取。从而有如下的输出：

```
%QB24 is written to at the following locations:
PLC_PRG (3): %QB24
PLC_PRG.POU1 (8): %QB24
```

结果显示在消息窗内。

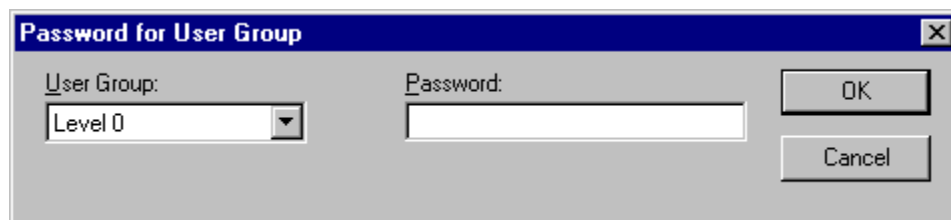
用户组

在 TwinCAT PLC Control 中，最多可建立八个有不同存取权限、对 POU 数据类型和资源进行存取的用户组。可建立对单个对象或对所有对象的存取权限。只有某个用户组的成员才可以打开某个项目。这样，一个用户组的成员必须使用一个密码来识别其本人。

用户组从 0 至 7 编号，组别 0 有管理者权限，亦即只有组别 0 的成员可以确定密码并具备对所有组别和/或对象进行存取的权限。

当推出一个新项目时，最初所有的密码都是空的。直到已为组别 0 设定了一个密码时为止，作为组别 0 的一个成员能自动地进入这个项目。

当装载项目时，假如已存在为用户组 0 设定的一个密码，则当该项目打开时，所有组都需要一个密码。为此，出现以下的对话：



用于输入密码的对话框

在对话框左侧的组合框“User group”（用户组别），输入你所属的组别，在对话框的右侧输入相应的密码。按 OK。如果密码与存储密码不相符合，则出现以下消息：

“The password is not correct”（密码不正确）。

只有输入了正确的密码才能打开项目。

重要事项:

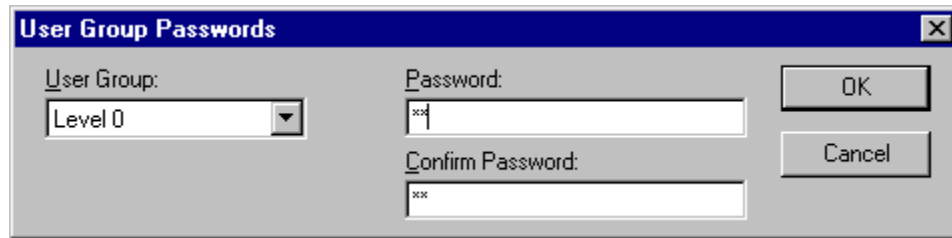
如果密码并未分配给所有的用户组，则通过未分配密码的一个组可打开一个项目！

重要事项:

使用命令“Passwords for user group”（用户组的密码），你可分配密码，并且使用命令“Object”（对象）、“Access rights”（存取权限），你可对一个对象或对所有对象规定存取权限。

“Project”（项目）→“Passwords for user groups”（用于用户组的密码）

使用该命令可打开用户组密码分配对话框。这个命令只能由组别 0 的成员执行。当给出这个命令时，出现以下的对话框：



密码分配对话框

在左边组合框“**User group**”（用户组别）你可选择组别。在字段“**Password**”（密码）输入该组别所需的密码。对于每个输入的字符，在该字段中相应出现一个星号（*）。你必须在字段“**Confirm password**”（确认密码）内重复相同的密码。输入每个密码后，以 **OK** 关闭对话框。如果你得到消息：

“**The password does not agree with the confirmation**”（通行字与确认不符），表示你在二次输入中有一次为输入错误。

在这种情况下，重复地输入二次，直到对话框关闭而没有出错消息时为止。然后，如有必要通过再次调用命令，为下一个组别分配一个密码。应用命令“**Object**”（对象）→“**Access rights**”（存取权限）为单个对象或所有对象分配存取权限。

3.4 对象

现在，我们要说明如何应用对象来工作，以及可利用那些帮助来保持跟踪一个项目（文件夹，调用树，交叉引用表，…）

对象

POU、数据类型以及资源（全局变量，袖样跟踪，PLC 配置，任务配置，以及监视和接收管理器）都被定义为“对象”。为使项目结构化而插入的文件夹只是部分涉及。项目的所有对象都包含在对象管理器中。

如果你将鼠标指针停留在对象管理器中的一个 POU 上一段时间，则在“**Tooltip**”（工具提示）中显示该 POU 的类型（程序，功能或功能块）。对于全局变量而言，工具提示显示关键字（**VAR_GLOBAL**，**VAR_CONFIG**）。

使用拖放操作，你可在一个对象类型内移动对象（也包括文件夹，参见“文件夹”）。为此，选择对象并通过按压鼠标左键，将对象移到需要的地方。如果该移动引起名字冲突，则最新引入的元素将被附加系列号而加以唯一识别（例如：“**Object_1**”）

文件夹

为了保持跟踪较大的项目，必须系统地在文件夹中将 POU、数据类型和全局变量进行编组。

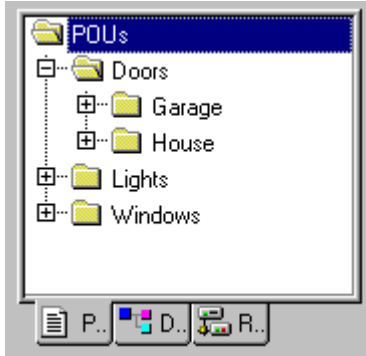
可建立你所要求的多层次文件夹。如果在一个关闭的文件夹符号前有一个加号，则该文件夹包含有对象和/或其它的文件夹。在加号上点击，可打开文件夹，并显示子对象。在负号（它代替加号）上点击，可以重新关闭文件夹。

在上下文菜单中，你可以找到相同功能的命令“**Expand nodes**”（扩展节点）和“**Collapse nodes**”（收缩节点）。

使用拖放操作，可在其对象类型内移动文件夹的对象和文件夹。为此，选择对象并通过按住鼠标左键将该对象拖放到所需要的位置。

注:

文件夹对程序没有影响，只是用来使你的项目结构更为清晰。



对象管理器中的文件夹示例

“New Folder”（新建文件夹）

使用该命令，一个新文件夹可作为一种结构对象插入。如果已选择一个文件夹，则新文件夹是在已选出的文件夹下建立的。否则，它在相同层次上建立。如果选择的是一个动作，则新文件夹将插入在该动作所属的 POU 的层次上。当选择了一个对象或对象类型，而且已按鼠标右键或 <Shift>+<F10> 时，则出现包含这个命令的对象管理器的快捷菜单。

新插入的文件夹初始名称为“新文件夹”。请参阅以下文件夹命名惯例：

- 在层次结构中处于相同层次的文件夹必须有彼此相区别的名字。处于不同层次的文件夹可以有相同的名字。
- 一个文件夹不能有与位于相同层次上的对象相同的名字。

如果在相同层次上早已有名字“新文件夹”，则有这个名字的每个附加的文件夹都自动地接收一个附加的系列号（例如，“New Folder 1”）。重新命名一个早已使用的名字是不可能的。

“Expand nodes”（扩展节点）、“Collapse nodes”（收缩节点）

使用扩展命令，位于选择对象内的对象被显式地展开。使用收缩命令，则不再显示其子对象。可双击鼠标键或按 <Enter> 键来打开或关闭文件夹。当已选出一个对象或对象类别，而且你已按鼠标右键或 <Shift>+<F10> 键时，将出现包含有这个命令的对象管理器的快捷菜单。

“Project”（项目）→ “Object Delete”（删除对象）快捷键：

使用该命令，当前选择的对象（POU，数据类型或全局变量），或带有子对象的文件夹可从对象管理器中删除，也可以从项目中删除。为了安全起见，会向你再次询问对此操作的确认。若对象编辑器窗是打开的，则自动关闭。若你用命令“Edit”（编辑）→“Cut”（剪切）进行删除，则对象暂时保留在剪贴板上。

“Project”（项目）→ “Object Insert”（插入对象）快捷键：<Ins>

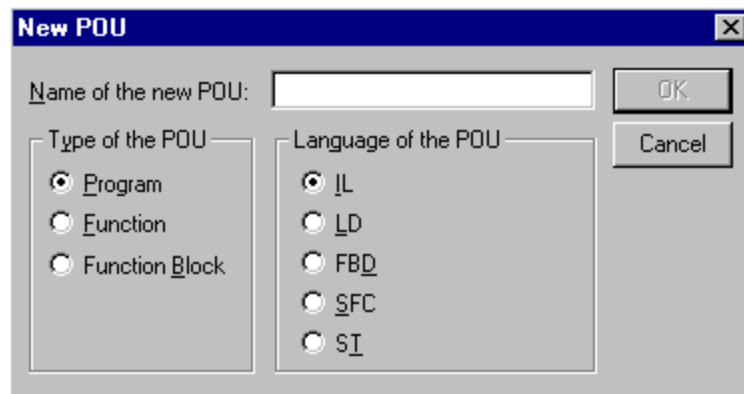
使用该命令，你可以建立一个新对象。对象的类型（POU，数据类型或全局变量）取决于对象管理器中所选择的属性页。在对话框中输入新建对象的名字。记住，对象的名字也许尚未使用过。

请注意以下约束条件：

- 一个 POU 的名字不能包括任何空格
- 一个 POU 不能有与其它 POU 有相同的名字，或一个数据类型相同的名字。

- 一个数据类型不能接受与其它数据类型相同的名字或一个 POU 相同的名字
- 一个全局变量表不能有与其它全局变量表相同的名字。
- 在同一个 POU 中一个动作不能有与其它动作相同的名字。
- 一个可视界面不能有与其它可视界面相同的名字。

在所有其它情况下，允许有同样的命名。这样（例如），属于不同 POU 的动作可以有相同的名字，而一个可视界面也许有与一个 POU 相同的名字。对 POU 而言，POU 类型（程序、功能或功能块）以及所使用的编程语言必须加以选择。“Program”（程序）是“Type of the POU”（POU）类型的默认值，而“Language of the POU”（POU 语言）的默认值是最近建立 POU 的语言。如果建立的 POU 是一种功能类型，则应将所需要的数据类型输入到“Return Type”（返回类型）文本输入字段。这里允许所有基本的和定义的数据类型（数组，结构，枚举，别名）。例如可使用输入助手（例如，通过〈F2〉）。



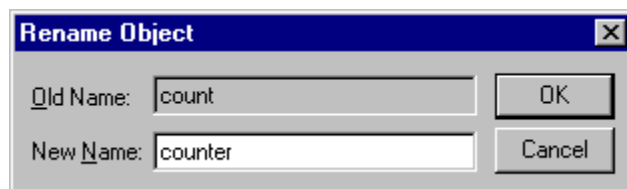
按 OK 按钮后，只有不与上述命名规则相冲突时，才有可能在对象管理器中建立新对象，并出现相应的输入窗口。

如果使用命令“Edit”（编辑）→“Insert”（插入），则插入当前剪贴板上的对象，而且不出现对话框。如果插入的对象名字与命名规则（见上面）相冲突，则可通过附加一个前面带下划线的字符系列号使其成为唯一的名字（例如“Rightturnsig_1”）。

如果项目是在 ENI 数据库中的资源控制下，也许会（取决于“Project source control”（项目资源控制）对话框中的“Project”（项目）选项的设定）自动地向你提问，你要在哪一种数据库类别中来处理新的对象。在这种场合，对话框“Properties”（属性）打开，你可以将对象分配给数据库对象类别中的一个类别。

“Project”（项目）→“Object Rename”（对象重新命名）快捷键：<Spacebar>

使用该命令，可将一个新的名字赋予当前选择的对象或文件夹。记住，对象的名字也许尚未使用过。如果对象编辑窗是打开的，则当更改名字时，自动更改其标题。

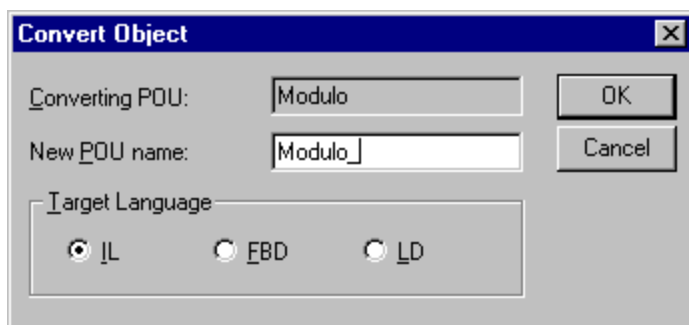


重新命名 POU 的对话框

“Project”（项目）→“Object Convert”（对象转换）

这个命令只能与 POU 一起使用。你可将 POU 从 SFC、ST、FBD、LD 和 IL 语言转换成 IL、FBD 和 LD 三种语言中的一种。为此，必须编译项目。选择你要转换成的语言，并给出一个新的 POU 名。记住，POU 的名字也许尚未使用过。然后压按 OK 按钮，这样，新的 POU 就添加到你的 POU 表内。

转换过程中出现的处理类型相应于编译时所使用的类型。

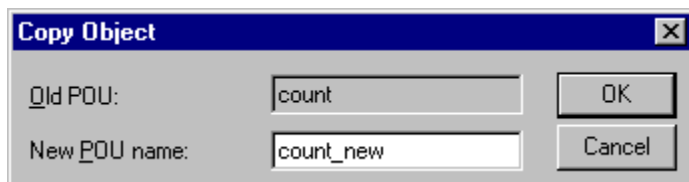


用于转换 POU 的对话框

“Project” (项目) → “Object Copy” (对象复制)

使用该命令，可复制一个选出的对象并将它在一个新名下保存。将新的对象名输入到结果对话框内。对象名也许尚未使用过。

另一方面，如果你用命令“Edit” (编辑) → “Copy” (复制)，则对象暂时保留在剪贴板上，并且不出现对话框。



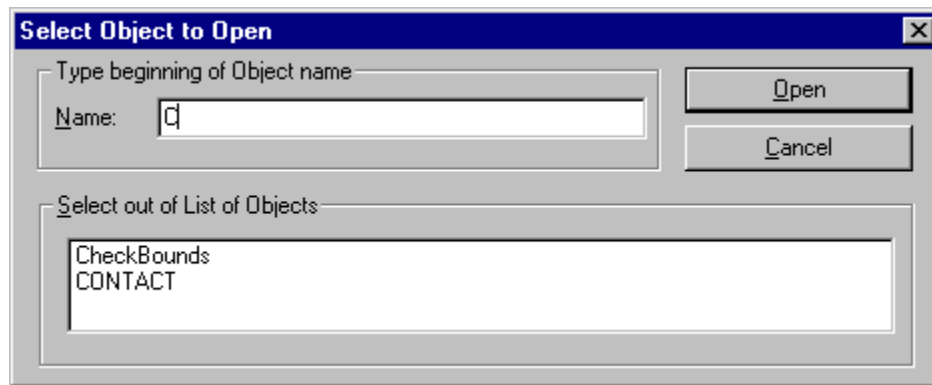
用于复制 POU 的对话框

“Project” (项目) → “Object Open” (对象打开) 快捷键: <Enter>

使用该命令，你可将“Object Organizer” (对象管理器) 中一个选出的对象装载到相应的编辑器内。如果该对象的窗口早已打开，则焦点指向该对象，并移到前台，现在可对它进行编辑。打开一个对象还有两种其它方法：

- 用鼠标在所需要的对象上双击，或
- 将对象名的第一个字母输入到“Object Organizer” (对象管理器) 内。然后打开一个对话框，其中显示带有这个初始字母的所有可供使用的对象类型的对象。选择所需要的对象，并点击“Open” (打开) 按钮，以便将这个对象装入编辑窗内。从而该对象在“Object Organizer” (对象管理器) 中被标记，而在所有层次的对象路径中，其层次高于包含文件夹的对象也均会被展开。这个由对象类型“Resources” (资源) 支持的选项仅用于全局变量。

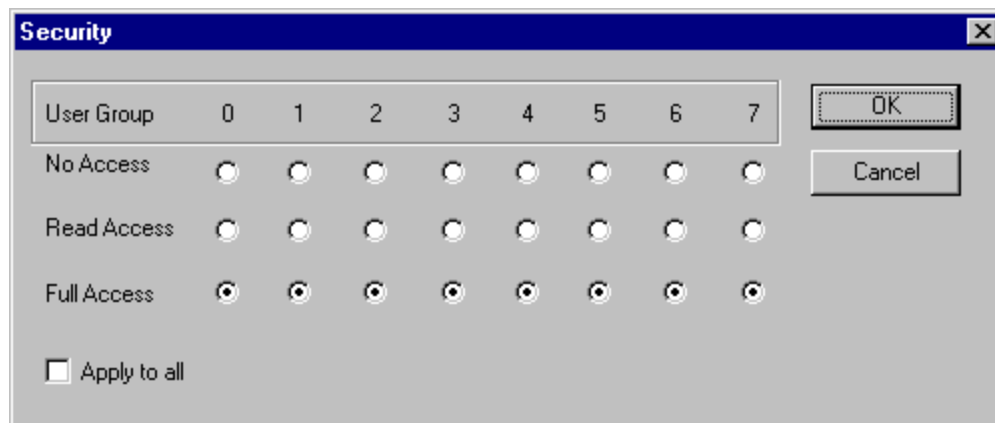
最后一种可能性在带有多个对象的项目中十分有用。



选择要打开对象的对话框

“Project”（项目）→ “Object Access rights”（对象存取权限）

使用该命令，你可打开用于将存取权限分配给不同用户组别的对话框。将出现以下对话框：



分配存取权限的对话框

现在用户组别 0 的成员可将各个存取权限分配给每个用户组。有三种可能的设置：

- “No Access”（无存取）：用户组的成员不能打开该对象。
- “Read Access”（阅读存取）：用户组的成员可以打开这个对象，但不能更改。
- “Full Access”（完全存取）：用户组的成员可以打开并更改这个对象。

设置或是指“Object Organizer（对象管理器）”中当前选择的对象，如果选择“Apply to all”（应用于所有）选项，则指所有 POU 数据类型和项目的资源。如果已向用户组别 0 分配了密码，则当使用密码请求打开项目时，就进行对一个用户组的存取权限分配。

“Project”（项目）→ “Object properties”（对象属性）

使用该命令，可以打开当前标记在“Object Organizer”（对象管理器）中的对象“Properties”（属性）对话框。在选项卡“Access rights”（存取权限）上，你可找到在执行命令“Project”（项目）→“Object Access Rights”（对象存取权限）时所得到的相同的对话框。取决于对象和项目设置，你可以定义对象属性是否存在可供使用的附加选项卡：

- 如果一个全局变量表是当前在“Object Organizer”（对象管理器）中选择的，则可利用选项卡“Global variable list”（全局变量表），该表中定义了涉及表有效化和网络变量数据交换的参数。在这个表中还可以修改条目。如果你通过在“Object Organizer”（对象管理

器)中的“Global Variables”(全局变量)中选择一个条目并执行命令“Add Object”(添加对象)来建立一个新的全局变量,也可打开这个对话框。

- 如果这个项目连接到一个“data base”(数据库),则提供一个“Database-connection”(数据库连接)选项卡。在这里,你可显示并修改当前的对象分配。该对象分配是提供给与类别“Local”(局部)有关的一个数据库类别。

“Project”(项目)→“Data base link”(数据库链接)

仅当你已选中了类别“Project source control”(项目资源控制)的项目选项对话框中的选项“Use source control (ENI)”(使用资源控制(EIN))后,才能使用这个菜单项。它附有一个子菜单,在那里你可找到以下处理对象的命令,与当前连接的 ENI 数据库中的项目有关:

- 登录(用户登录到 ENI 服务器)

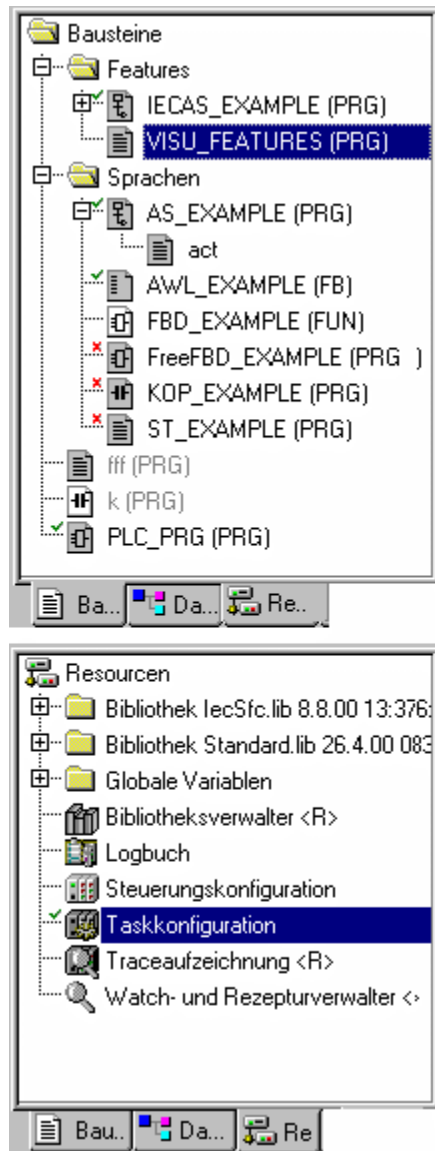
如果一个对象在“Object Organizer”(对象管理器)中被标记,并执行命令“Data Base Link”(数据库链接)(从快捷菜单,鼠标右键),则可使用以下命令来执行相应的数据库动作:

- “Define”(定义)
- “Get Latest Version”(获得最新版本)
- “Check Out”(检验)
- “Check In”(登记)
- “Undo Check Out”(取消检验)
- “Show differences”(显示差异)
- “Show Version History”(显示版本历史)

如果选中“Project”(项目)菜单中的命令“Data Base Link”(数据库链接),则可使用附加菜单项,它涉及项目的所有对象:

- “Multiple Define”(多重定义)
- “Get All Latest Versions”(获得所有最新版本)
- “Multiple Check Out”(多重检验)
- “Multiple Check In”(多重登记)
- “Multiple Undo Check Out”(多重取消检验)
- “Project Version History”(项目版本历史)
- “Label Version”(标记版本)
- “Add Shared Objects”(添加共享对象)
- “Refresh Status”(刷新状态)

参阅以下的命令描述。

**加灰色阴影的图标:**

对象保持在数据库中

对象名前的绿色检验符:

对象在局部项目内检验。

对象名前的红色十字号:

对象当前被其它用户检验。

对象名后面的 <R>:

对象是只读的，不能被编辑。

注:

有些对象（任务配置，抽样跟踪，PLC 配置，目标设定，监视和接收管理器），只要它们尚未检验，则按照默认值分配一个 <R>。这意味着，一旦你开始编辑对象，就不会自动向你询问这个对象是否应检验；但这并不意味着，你不能编辑这个对象。如果实际上没有写存取，则不能使用命令“Check out”（检验）。

“Project”（项目）→ “Data base command”（数据库命令）→ “Define”（定义）

利用菜单“Project”（项目）→“Data Base Link”（数据库链接）中的该命令可规定“Object Organizer”（对象管理器）中当前标记的对象是否应保存在数据库中，或只是局部地保存在项目中。打开一个对话框，在那里，你可从“Project”（项目）或“Shared objects”（共享对象）两种数据库类别中选用一种，或选取类别“Local”（局部）。

所有在数据库中管理的对象图标在“Object Organizer”（对象管理器）中以灰色阴影显示。

“Project”（项目）→“Data base command”（数据库命令）→“Get latest version”（获得最新版本）

使用菜单“Project”（项目）→“Data Base Link”（数据库链接）中的该命令来复制对象的当前版本，它在“Object Organizer”（对象管理器）中标记，并从数据库复制到你的项目。这将改写本机版本。与“Check Out”（检验）动作相对应，该对象并不锁定数据库中的其它用户。

“Project”（项目）→“Data base command”（数据库命令）→“Check Out”（检验）

使用菜单“Project”（项目）→“Data Base Link”（数据库链接）中的该命令来检验对象，它在“Object Organizer”（对象管理器）中标记，可从数据库并通过数据库来锁定其它用户对它的使用。

当执行该命令时，用户会得到一个对话框“Check out object”（检验对象）。可附加一个注释，它将保存在数据库中对象的版本历史记录中。

以 OK 按钮关闭对话框之后，检验的对象将在本地项目的对象管理器中以绿色对勾（√）标记。对于其它用户而言，则以红色十字号标记，对他们来说是不能编辑的。

“Project”（项目）→“Data base command”（数据库命令）→“Check In”（登记）

使用菜单命令“Project”（项目）→“Data Base Link”（数据库链接）中的该命令来登记对象，它在对象管理器中标记，并登记到数据库。从而，在数据库中建立对象的一个新版本。原有版本将随之保存。

当执行该命令时，用户会得到一个对话框“Check in object”（登记对象）。可附加一个注释，它将保存在数据库中对象版本的历史记录内。

使用 OK 按钮关闭对话框后，对象管理器中对象名前的绿色检验符号被删除。

“Project”（项目）→“Data base command”（数据库命令）→“Undo Check Out”（取消检验）

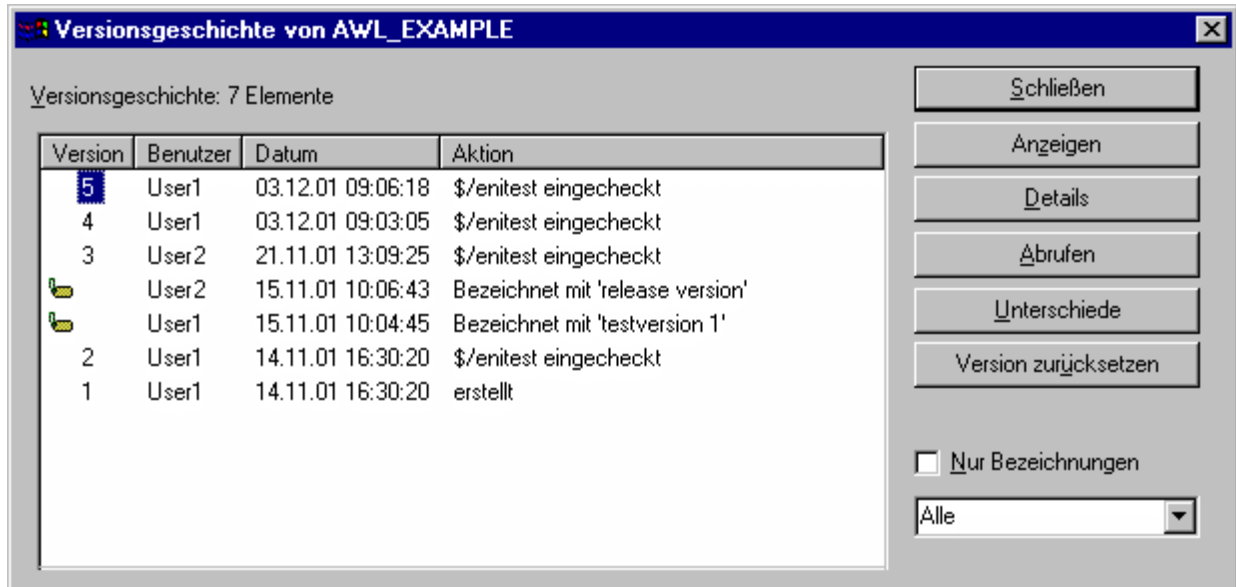
使用菜单“Project”（项目）→“Data Base Link”（数据库链接）中的该命令可清除当前在对象管理器中标记的对象的“Checking out”（检验）。从而，已作出的对象局部修改也被清除。不出现对话框。对象未改动的最后版本将保存在数据库内，可被其它用户重新存取。对象管理器中，对象名前的红色十字号将消失。

“Project”（项目）→“Data base command”（数据库命令）→“Show Differences”（显示差异）

使用菜单“Project”（项目）→“Data Base Link”（数据库链接）中的该命令来显示当前被用户打开的对象（在划分为二部分的窗口内）。如同项目比较中所述那样标记出版本的差异。

“Project”（项目）→“Data base command”（数据库命令）→“Show Version History”（显示版本历史）

使用菜单“Project”（项目）→“Data Base Link”（数据库链接）中的该命令打开对象的 <对象名> 版本历史对话框。该对象当前在对象管理器中标记。在这个对话框中列出已登记在数据库内或已在那里作出标记的对象的所有版本：



“Version”（版本）：

已逐一登记的对象版本的数据库专用编号。有标记的版本没有版本号，但用一个标记图标标出。

“User”（用户）：

用户名，已执行了登记或标记工作

“Date”（日期）：

工作日期和时间戳

“Action”（动作）：

已执行的动作类型。可能的类型有：“建立”（首次已在数据库中登记的对象），“登记”（除了第一个以外的所有对象的登记），以及“用<标记>标记”的对象（已将一个标记分配给对象的这个版本）。

按钮：

“Close”（关闭）：将对话框关闭。

“Display”（显示）：在窗口中打开当前在表中标记的版本。标题栏显示：“ENI: <数据库中的项目名>/<对象名>”

“Details”（详细）：

将打开对话框“Details of Version History”（详细版本历史）：

“File”（文件）（项目名和数据库中的对象），“Version”（版本）（见上面），“Date”（日期）（见上面），“User”（用户）（见上面），“Comment”（注释）（当登记对象或标记时插入注释）。用按钮“Next”（下一个）或“Previous”（上一个），可跳转到“Version history of...”（...的版本历史）对话框中，表内下一个或上一个条目的详细窗口。

“Get latest version”（得到最近的版本）：在表中作出标记的版本将装入 TwinCAT PLCControl 中，在那里将改写本地版本。

“Differences”（差异）：如在表中只标出一个对象的一个版本，则该命令将导致这个版本与最近（实际）的数据库版本进行比较。若两个版本都打上标记，则对它们进行比较，差异显示在一个对半分的窗口内。这与项目比较时的情况相类似。

“Reset version”（恢复版本）：在表中标记的版本将设定为最近版本。以后登记的所有版本都将被删除！这样做有助于恢复一个对象的早期状态。

“Labels only”（只用标记）：若选中这个选项，则只显示表中以标记标出的对象的版本。

选择框位于“Labels only”（只用标记）选项下面：在这里你可找到所有用户名，这些用户已为当前项目的对象执行了任何数据库动作。如果你需要得到关于所有用户或只是某个用户的版本历史，可选择“All”（所有）或某一个名。

“Project”（项目）→ “Data base command”（数据库命令）→ “Multiple Define”（多重定义）

使用菜单“Project”（项目）→“Data Base Link”（数据库链接）中的该命令，可将几个对象一次分配给某个数据库类型。对话框“Properties”（属性）如同命令“Define”（定义）所述那样打开。选择所需要的项，然后以 OK 按钮关闭对话框。此后会打开对话框“ENI-Selection”（ENI 选择），列出所有用于选出类别的项目 POU（例：如果你选择类别“shared objects”（共享对象），则选择窗口只提供“Resources”（资源）标记的 POU）。POU 以树形结构表示，符合对象管理器的结构。选择所需要的 POU，并以 OK 按钮确认。

“Project”（项目）→ “Data base command”（数据库命令）→ “Get All Latest Versions”（得到所有最近版本）

使用菜单“Project”（项目）→“Data Base Link”（数据库链接）的该命令，可调用当前打开项目的每个对象的最近版本，该版本在源代码控制下从数据库保存。考虑到以下情况：

如果此时已在数据库项目文件夹内保存了其它对象，则现需要将它们附加到 PLC 控制系统中的本地项目内。

如果此时已将对象从数据库中删除，则这些对象不会在本地项目中删去，但是，它们将自动地分配到类别“Local”（本地）。如这些对象可供本地项目使用，则只调用类别“Shared Objects”（共享对象）的对象最新版本。详细情况请参阅命令“Get latest version”（得到最新版本）。

“Project”（项目）→ “Data base command”（数据库命令）→ “Multiple Check Out”（多重检验）

使用菜单“Project”（项目）→“Data Base Link”（数据库链接）中的该命令，可一次检验几个对象。为此，打开对话框“ENI-Selection”（ENI 选择），列出项目的所有 POU。选择需要检验的对象并以 OK 按钮确认。详细情况请参阅命令“Check Out”（检验）。

“Project”（项目）→ “Data base command”（数据库命令）→ “Multiple Check In”（多重登记）

使用菜单“Project”（项目）→“Data Base Link”（数据库链接）中的该命令，可一次登记几个对象。为此，打开对话框“ENI-Selection”（ENI 选择），列出项目的所有 POU。选择需要登记的对象并以 OK 按钮确认。详细情况请参阅命令“Check In”（登记）。

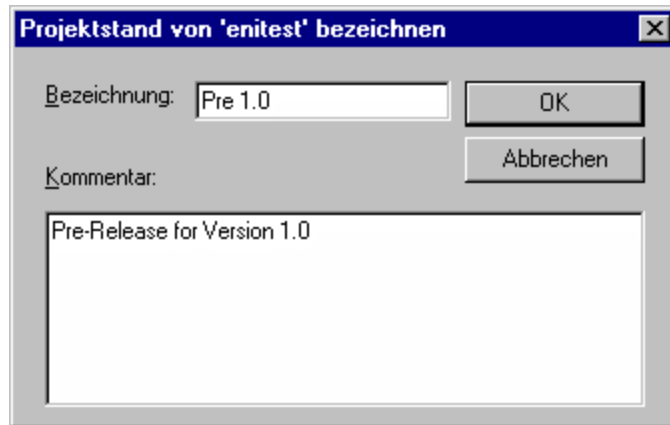
“Project”（项目）→ “Data base command”（数据库命令）→ “Project Version History”（项目版本历史）

使用菜单“Project”（项目）→“Data Base Link”（数据库链接）中的该命令，为当前打开的项目查阅其版本历史。但是，仅当所选择的数据库系统支持这个功能时才能进行此项操作。将打开对话框“History of <data base project name>”（数据库项目名）的历史。它按时间顺序显示项目特定对象的动作（建立，登记，标记）。在“Version history”（版本历史）后面显示对象总数。这个对话框可如同命令“Show Version History”（显示版本历史）所描述那样进行，但是要注意以下几点：

- 命令“Reset Version”（恢复版本）只供单个对象使用。
- 命令“Get latest version”（得到最新版本）将调用本地项目内当前标记登录的版本中所有对象！这意味着，在 CoDeSys 中的对象将以较早的版本改写。但是：本地对象（它们还不是较早版本中项目的一部分）不会从本地项目中清除掉！

“Project”（项目）→ “Data base command”（数据库命令）→ “Project Label Version”（项目标记版本）

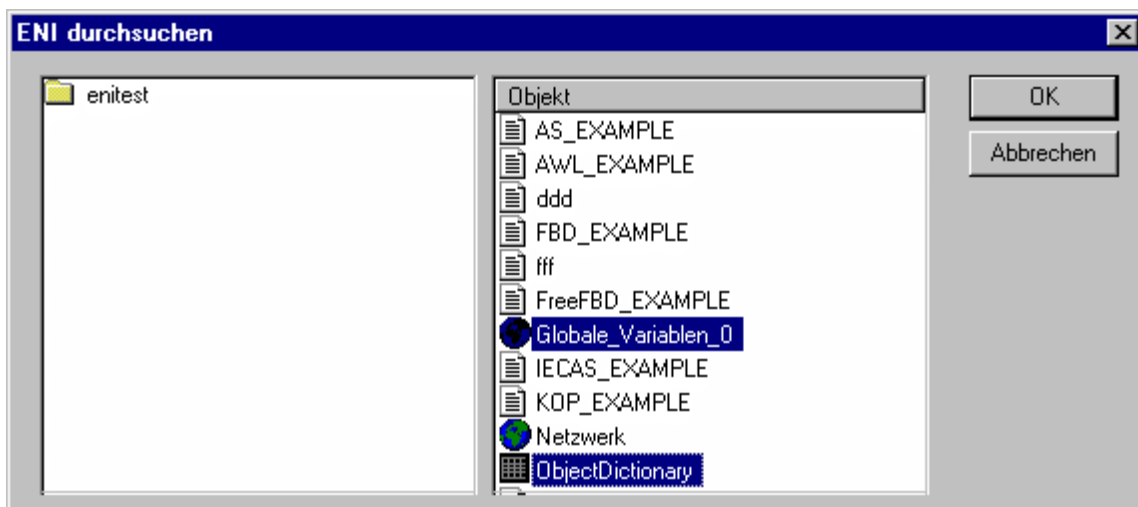
使用菜单“Project”（项目）→“Data Base Link”（数据库链接）中的该命令，将一个“label”（标记）放置在项目每个对象的实际版本上，这样在以后可准确地再调用这个项目版本。打开对话框“Label <data base project name>”（标记<数据库项目名>）。插入一个“Label”（标记）名并任选一个“Comment”（注释）。当你以 OK 按钮确认时，关闭对话框，标记和动作“labeled with <label name>”（以<标记名>标记）将出现在版本历史表内，也出现在项目历史中单个对象的历史内。标记的版本不会得到一个版本号，只是以一个标记图标在“Version”（版本）栏内标示。如果在“Version History”（版本历史）对话框中，选中选项“Labels only”（只用标记），则只列出带标记的版本。



“Project”（项目）→“Data base command”（数据库命令）→“Add Shared Objects”（添加共享对象）

使用菜单“Project”（项目）→“Data Base Link”（数据库链接）中的该命令，可显式地要求将数据库类别“Shared Objects”（共享对象）的新对象添加到本地打开的项目上。对于类别“Project Objects”（项目对象）的对象，这是不必要的，因为命令“Get (all) latest version(s)”（得到（所有）最新版本）会自动地调用在数据库项目文件夹中找到的所有对象，甚至包括尚未用于本地项目中的一些对象。但是，对于类别“Shared Objects”（共享对象）的对象而言，在这种情况下，只调用早已在本地项目中提供的那些对象。

这样，执行命令“Add Shared Objects”（添加共享对象），打开对话框“Browse ENI”（浏览 ENI）。在窗口右侧的表显示用于数据库文件夹（它是当前从窗口左侧表中选出的）的所有对象。选取所需要的对象并以 OK 按钮确认，或双击条目，将对象插入到当前打开的项目上。



“Project”（项目）→“Data base command”（数据库命令）→“Refresh Status”（刷新状态）

使用菜单“Project”（项目）→“Data Base Link”（数据库链接）中的该命令可更新对象管理器中的显示，使你能看到与项目资源控制有关的对象实际状态。

“Project”（项目）→“Data base command”（数据库命令）→“Login”（登录）

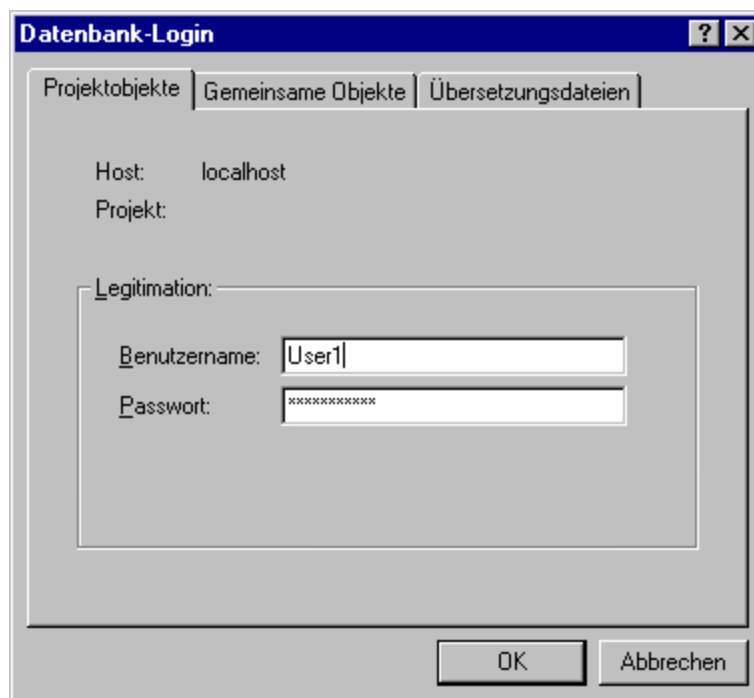
使用菜单“Project”（项目）→“Data Base Link”（数据库链接）中的该命令，可打开对话框“Login”（登录），在那里你可经过 ENI 服务器输入用于 ENI 数据库的存取数据。存取数据还应在 ENI 服务器“ENI Admin, User Management”（“ENI 管理员”，“用户管理”）中定义，以及—取决于当前所使用的数据库，还应在数据库的用户管理中定义。执行该命令后，首先打开用于类别“Project objects”（项目对象）的“Login”（登录）对话框：

“Data base”（数据库）：项目对象

“Host”（主机）：计算机的地址，ENI 服务器在该地址运行（必须与“项目资源控制”的项目选项对话框中的“TCP/IP 地址”字段的输入项相对应。）。

“Project”（项目）：数据库项目名（必须与“Project source control”（项目资源控制）/“Project Objects”（项目对象）的项目选项对话框中“Project name”（项目名）字段的输入项相对应）。

“Credentials”（用户认证信息）：插入“User name”（用户名）和“Password”（密码）。



按 OK 按钮确认设定。对话框被关闭，自动打开用于“Shared objects”（共享对象）的对话框“Login”（登录）。

使用上述“Project objects”（项目对象）对话框相同方式输入存取数据，并以 OK 按钮确认。

“Project”（项目）→“Add action”（添加动作）

该命令用来生成一个动作，分配给对象管理器中的一个选出的程序块。在出现的对话框中选择动作名，以及相应执行该动作的语言。

新的动作安放在对象管理器中用户的程序块下面。在程序块前面出现一个加号（+）。以鼠标单击加号使动作对象出现，并在程序块前面出现一个负号（-）。单击负号，使这个动作消失并重新出现加号。通过快捷命令“Expand Node”（扩展节点）和“Collapse Node”（收缩节点）也可实现这个动作。

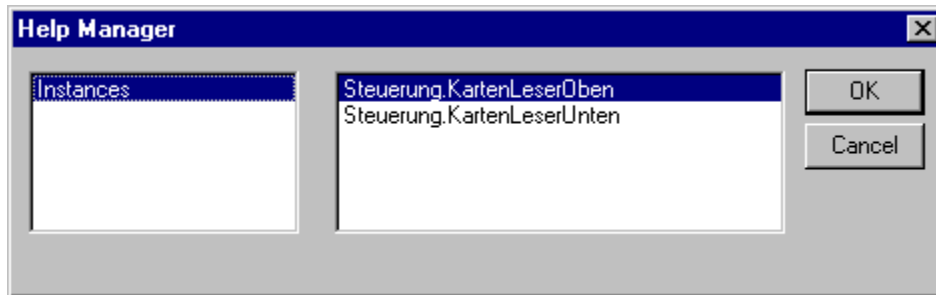
在这个动作上双击或按 <Return>，则将一个动作装入到编辑器内。

“Project”（项目）→ “Open Instance”（打开实例）

使用该命令可打开显示对象管理器中选出的功能块实例。以相同的方式，双击对象管理器中的功能块，进入选择对话框，其中列出功能块及其实现的实例。在这里选择所需要的实例或实现并以 OK 按钮确认。该窗口内显示所需要的项。

注:

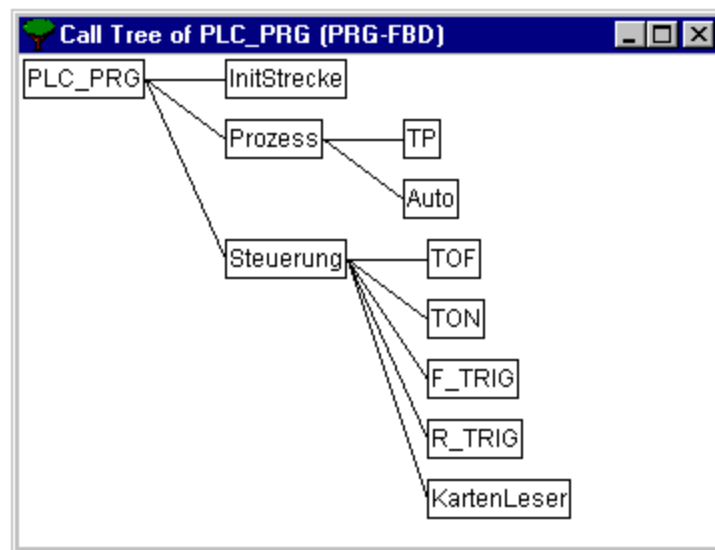
如你需要观察实例，首先，你必须登录！（项目已编译而且没有错误，然后使用菜单命令“Online”（联机）→“Login”（登录）装载到 PLC）。



打开实例的对话框

“Project”（项目）→ “Show call tree”（显示调用树）

使用该命令，可打开一个窗口，它显示对象管理器中所选出对象的调用树。为此，项目必须是编译的（见命令“Rebuild all”（全部重建））。调用树包括 POU 的调用和数据类型的引用。



调用树示例

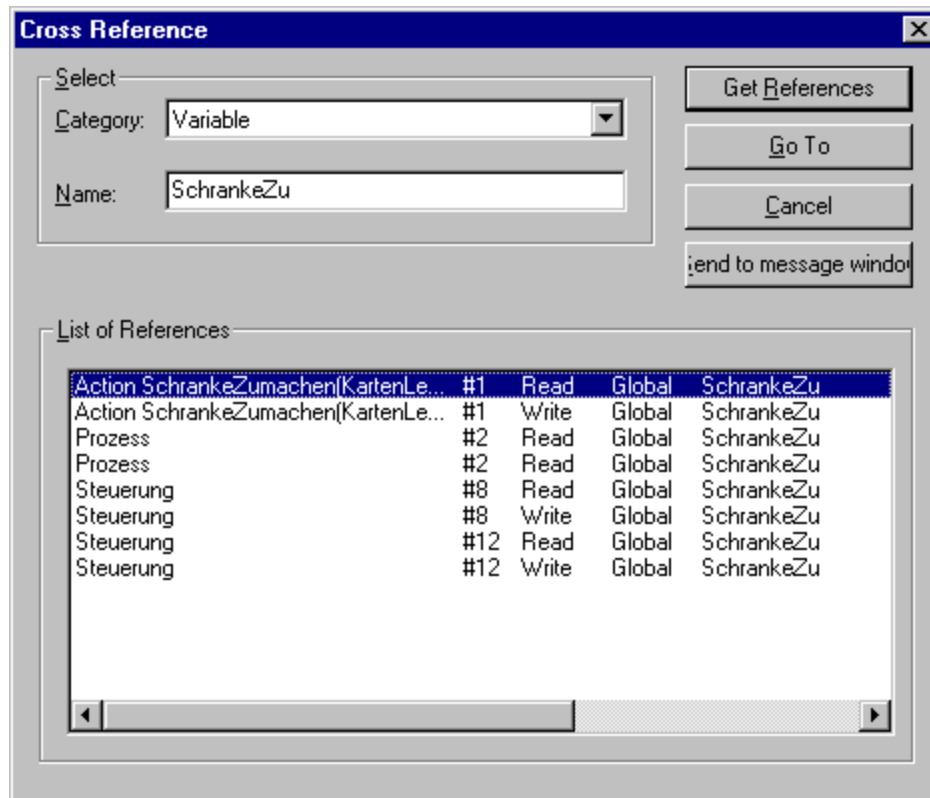
“Project”（项目）→ “Show cross reference list”（显示交叉引用表）

使用该命令可打开一个对话框，它能够为你输出变量、地址、或 POU 的所有应用点。为此，项目必须是编译的（见命令“Rebuild all”（全部重建））。首先选择类别“Variable”（变量）、“Address”（地址），或 POU，然后输入所需要的元素名。为了获得输入类别的所有元素，在“Name”（名）中输入一个星号“*”。

通过点击“**Get References**”（得到引用），你可得到所有应用点的表。与 POU 和行号或网络号一起，还可规定变量名和地址（如果有的话）。“Domain space”（域空间）用于显示这是一个局部变量还是一个全局变量；地址栏表示变量在当前位置上是以“reading”（读取）还是用“writing”（写入）存取。

当你选出交叉引用表中的一行并按“**Go To**”（转换）按钮或在行上双击时，则显示 POU 在其编辑器中的相应点。以这种方式，你可跳转到所有的应用点而不需要费时的搜索。

为了使处理过程更为简单，可使用“**Send to message window**”（发送到消息窗）按钮，将当前的交叉引用表带到消息窗，并在那里转换成相应的 POU。



对话框和交叉引用表示例

3.5 编辑功能

你可以在所有的编辑器中使用以下命令，其中有些命令位于对象管理器中。所有的命令都位于菜单项“**Edit**”（编辑）下面。

“Edit”（编辑）→“Undo”（取消）快捷键：〈Ctrl〉+〈Z〉

该命令取消在当前打开的编辑器窗口中或在对象管理器中执行的最新动作。通过重复选择该命令，可以取消所有动作，恢复到窗口打开时的状态。该特性可用于编辑器中的 POU、数据类型和全局变量以及对象管理器中的所有动作。

使用命令“**Edit**”（编辑）→“**Redo**”（恢复），可恢复已取消的一个动作。

“Edit”（编辑）→“Redo”（恢复）快捷键：〈Ctrl〉+〈Y〉

在当前打开的编辑器窗口或对象管理器中使用该命令，你可以恢复已经取消（“Edit”（编辑）→“Undo”（取消））的一个动作。如同前面常使用的“Undo”（取消）命令一样，你也可以执行“Redo”（恢复）命令。“Undo”（取消）命令和“Redo”（恢复）命令仅对当前窗口有效。

注：

“Undo”（取消）命令和“Redo”（恢复）命令仅应用于当前窗口。每个窗口都有其自身的动作列表。如果你需要在几个窗口中取消一些动作，你必须选中相应的窗口。当在对象管理器中取消或恢复时，光标必须放在这里，即需激活相应的窗口。

“Edit”（编辑）→“Cut”（剪切）快捷键：<Ctrl>+<X> 或 <Shift>+

该命令将编辑器中当前选择的内容传送到剪切板。选择的内容从编辑器中删除。在对象管理器中，同样适用于所选择的对象，但并不是所有的对象（例如 PLC 配置）都能被删除。值得注意的是，并非所有的编辑器都支持剪切命令，其应用范围仅限于某些编辑器。选择形式取决于相应的编辑器：在文本编辑器（IL，ST 和声明）中，该选择是字符表。在 FBD 和 LD 编辑器中，该选择是若干个网络，它们由网络号字段中的虚线矩形或带有前置行、框和操作符的框来表示。在 SFC 编辑器中，该选择是一系列环绕步的虚线矩形部分。

为了粘贴剪切板的内容，可以使用命令“Edit”（编辑）→“Paste”（粘贴）。在 SFC 编辑器中，也可以使用命令“Extras”（附加）→“Insert parallel branch (right)”（插入并行分支（右向）），或“Extras”（附加）→“Paste after”（后向粘贴）。

为了将选择内容复制到剪切板而不删除它，使用命令“Extras”（附加）→“Copy”（复制）。

为了删除选择区域而不改变剪切板内容，使用命令“Edit”（编辑）→“删除”（Delete）。

“Edit”（编辑）→“Copy”（复制）快捷键：<Ctrl> + <C>

该命令将编辑器中当前选择的内容复制到剪切板。它不改变编辑器窗口的内容。对于对象管理器，同样适用于所选择的对象，但并不是所有的对象都能被复制（例如 PLC 配置）。值得注意的是，并非所有的编辑器都支持复制命令，其应用范围仅限于某些编辑器。对于所选择的类型而言，其应用规则和“Edit”（编辑）→“Cut”（剪切）命令相同。

“Edit”（编辑）→“Paste”（粘贴）快捷键：<Ctrl> + <V>

将剪切板内容粘贴到编辑器窗口的当前位置上。在图形编辑器中，仅当插入一个正确的结构结果时，才可以执行该命令。在对象管理器中，对象从剪切板中进行粘贴。值得注意的是，并非所有的编辑器都支持粘贴命令，其应用范围仅限于某些编辑器。取决于编辑器的类型，当前位置的定义可以不同：对于文本编辑器（IL，ST，声明）而言，当前位置就是你通过点击鼠标确定的光标闪烁位置（垂直线）。在 FBD 和 LD 编辑器中，当前位置是网络号字段中带有虚线矩形的第一个网络。剪切板的内容被插入到该网络之前。如复制的是结构的一部分，则将它插入在所选择的元素之前。在 SFC 编辑器中，当前位置由虚线矩形环绕的选择区域确定。取决于选择和剪切板的内容，这些内容或是插入在选择区域之前，或是插入在所选择区域左边的一个新分支中（并行或可选）。

在 SFC 中，可以使用命令“Extras”（附加）→“Insert parallel branch (right)”（插入并行分支（右向））或“Extras”（附加）→“Paste after”（后向粘贴），以便插入剪切板的内容。

“Edit”（编辑）→“Delete”（删除）快捷键：

从编辑窗中删除选择的区域，这并不改变剪切板的内容。在对象管理器中，同样适用于所选择的对象，但并不是所有的对象都能被删除（例如 PLC 配置）。对于所选择的类型而言，其应用规则和“Edit”（编辑）→“Cut”（剪切）命令相同。在库管理器中，该选择表示当前选择的库名称。

“Edit”（编辑）→“Find”（查找）

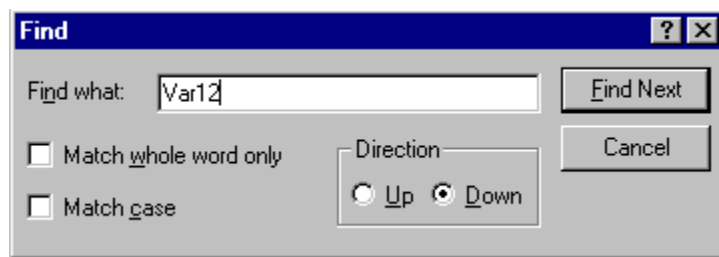
该命令在当前编辑窗内搜索某一段文本。“Find”（查找）对话框将打开。在按“Cancel”（取消）按钮之前，它一直保持打开。

在“Find what”（查找什么）字段，你可以输入你要查找的字符串。

此外，你可以确定正在查找的文本是否用于“Match whole word”（匹配整个字），或者是否还要考虑“Match case”（匹配大小写），以及是否要从当前光标位置开始“Up or Down starting”（向上或向下搜索）。

使用“Find next”（查找下一个）按钮开始查找，它从选择位置开始，并以所选择的方向继续进行查找。如果找到了文本段落，则该处高亮显示。如果找不到该段落，则有一个消息报告这个情况。查找可以重复不断地进行，直至到达编辑器窗内容的开始或结束处为止。

需要记住的是，找到的文本可能被“Find”（查找）对话框覆盖。



查找对话框

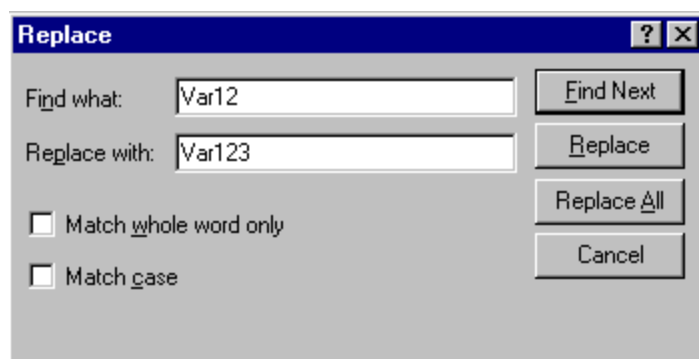
“Edit”（编辑）→ “Find Next”（查找下一个）快捷键：〈F3〉

使用该命令，你可以使用“Edit”（编辑）→ “Find”（查找）中最近所使用的相同参数来进行查找。

“Edit”（编辑）→ “Replace”（替换）

使用该命令，如同命令“Edit”（编辑）→ “Find”（查找）那样查找某段文本，并用另一段文本替换。当你选择了该命令后，将出现查找和替换对话框。在按“Cancel”（取消）按钮或“Close”（关闭）之前，这个对话框将一直保持打开。

按“Replace”（替换）按钮将当前选择的文本替换为字段“Replace with”（替换为）中的文本内容。按“Replace all”（全部替换）按钮，将当前位置之后字段“Find next”（查找下一个）中出现的每个文本替换为字段“Replace with”（替换为）中的文本内容。该过程结束时，将有一个消息报告替换的次数。



查找和替换对话框

“Edit”（编辑）→ “Input Assistant”（输入助手）快捷键：〈F2〉

使用该命令，可打开一个对话框，用于在编辑窗内的当前光标位置处选择可能的输入。你可以在左侧栏中选择所需要的输入类别；在右侧栏中选择所需要的条目，然后用 OK 按钮确认你的选择。这样，可以在该位置插入你的选择。

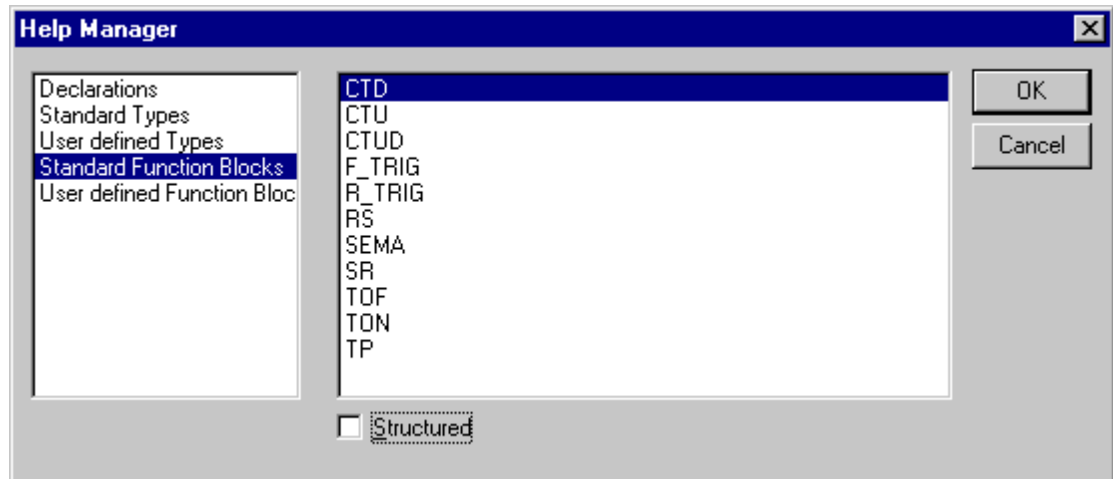
提供的类别取决于编辑窗内的当前光标位置，即取决于在该点可以输入什么（例如，变量、操作符、POU、类型转换等）。

如果选项 **“With arguments”**（使用自变量）是有效的，则插入所选择的元素时，用它来指定要传输的自变量，例如：选择功能块 `fu1`，它定义了输入变量 `var_in`：`fu1(var_in:=)`;

如果插入功能 `func1`，并使用 `var1` 和 `var2` 作为其传输参数：`func1(var1,var2)`

一般可以在使用的成员之间进行结构化显示和非结构化显示切换。它可以通过选中/取消 **“Structured Display”**（结构化显示）选项来实现。

- **“Unstructured Display”**（非结构化显示）：

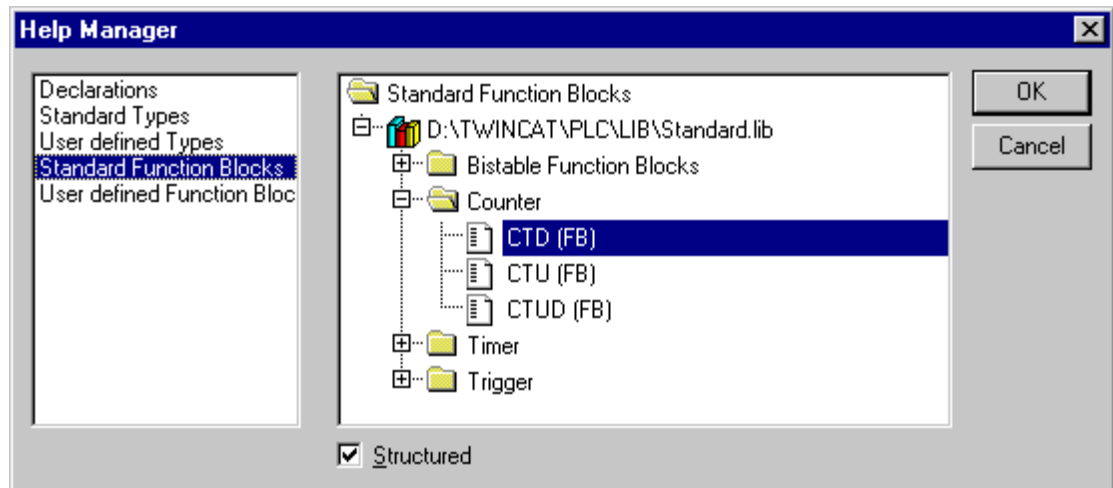


每个类别中的 POU、变量或数据类型可简单地以字母顺序线性排列分类。

在不同位置（例如在监视表内），需要有多级的变量名。在这种场合，“输入助手”（Input Assistant）对话框显示一个包含所有 POU 的表和用于全局变量的一个单独的点。在每个 POU 名称后是一个点。如果一个 POU 通过双击或通过按 **<Enter>** 按钮进行选择，则打开一个属于该 POU 的变量表。如果有实例和数据类型，则还可以打开分层显示中的其它层次。OK 按钮传输最近选出的变量。

在某些位置（例如在监视表内），多层变量名是必要的。首先，“Input Assistant”（输入助手）对话框中包含所有 POU 的表，以及用于全局变量的一个单独的点。在每个 POU 名后是一个点。通过鼠标双击或按 **<Enter>** 按钮，则打开一个用于 POU 的变量表。如果需要，可再次打开实例和数据类型。按 OK 按钮可以确认选出的变量。

- **“Structured Display”**（结构化显示）：



若选择“**Structured display**”（结构化显示），则按层次排列 POU、变量或数据类型。这对标准程序、标准功能、标准功能块、定义的程序、定义的功能、定义的功能块、全局变量、局部变量、定义的类型、监视变量都是可能的。直观显示和分层显示对应于对象管理器；如果元素在一个库中被引用，则这些元素以字母顺序在顶层插入。相应的层次结构如同在“**Library Manager**”（库管理器）中那样显示。

功能块的输入和输出变量（它们被描述成局部或全局变量）在实例名（例如 `Inst_TP ET`, `Inst_TP IN` 等）下，列在类别“**Local Variables**”（局部变量）或“**Global Variables**”（全局变量）内。为此，选择实例名（例如 `Inst_TP`）并以 **OK** 按钮确认。

如果在这里选择一个功能块实例，则可选择选项“**With arguments**”（使用自变量）。当文本语言 **ST** 和 **IL** 在进行任务配置时，可插入功能块的实例名和输入参数。

例如，如果选择 `Inst (DeklarationInst: TON;)`，则插入以下代码：

```
Inst (IN:= ,PT:=)
```

若未选择选项，则只插入实例名。在图形语言中，或在“**Watch window**”（监视窗）内，一般只插入实例名。

结构部件以模拟功能块实例的形式显示。

对枚举而言，每个枚举值在枚举类型下列出。其顺序是：来自库的枚举，来自数据类型的枚举，来自 POU 的局部枚举。

一般规则是，包含有子对象的各行是不能选择的（除了实例，见上述），但可以进行扩展层次显示或收缩层次显示（对多层次变量名而言）。

如果“**Input Assistant**”（输入助手）在“**Watch and Receipt Manage**”（监视和接收管理器）、或在跟踪配置对话框中的跟踪变量选择中被调用，则有可能作出多重选择。当按 **<Shift>** 按钮时，你可选择一系列变量；当按 **<Ctrl>** 按钮时，你可分别选择单个变量。选出的变量是这样标记的：如果在范围选择过程中，所选择的行不包括有效变量（例如 POU 名），则这些行不包括在选择范围内。当进行单个选择时，这类行将不能被标记。

在监视窗和跟踪配置中，有可能从“**Input Assistant**”（输入助手）对话框传送结构、数组或实例。当用鼠标按钮双击并伴随有元素扩展或收缩层次结构的显示时，这种情况下的选择只能通过 **OK** 加以确认。因此，选出的变量逐行插入到监视窗内，即每个选出的变量写入在单独的行上。在跟踪变量的场合，每个变量是插入在跟踪变量表的一个单独的行内。

如果跟踪变量的最大数，即 20，在选出变量的插入过程中被超出的话，则会出现一个出错消息“**A maximum of 20 variables is allowed**（最大允许 20 个变量）。”因此，其它多选出的变量不能插入到表内。

注：

有些条目（例如“**Global Variables**”（全局变量））只能在编译后，在“**Input Assistant**”（输入助手）对话框中更新。

“Edit”（编辑）→“Declare Variable”（声明变量）快捷键：<Shift>+<F2>

该命令打开一个变量声明对话框。当选项“**Project**”（项目）→“**Options**”（选项）→“**Editor**”（编辑器）→“**自动声明**”（Autodeclaration）选中时，以及当声明编辑器使用一个新的未定义变量时，这个对话框才自动打开。

“Edit”（编辑）→“Next Error”（下一个错误）快捷键：<F4>

在一个项目编译出现错误后，该命令可以显示下一个错误。相应的编辑窗有效，并选择编译错误的位置。同时，在消息窗内显示相应的出错消息。

“Edit”（编辑）→“Previous Error”（下一个错误）快捷键： <Shift>+<F4>

在一个项目编译出现错误后，该命令可以显示上一个错误。相应的编辑窗有效，并选择编译错误的位置。同时，在消息窗内显示相应的出错消息。

“Edit”（编辑）→“Macros”（宏指令）

该菜单项指向项目定义的所有宏指令的列表。（有关生成宏指令的信息参见“Project”（项目）→“Options”（选项）→“Macros”（宏指令））。当选择一个可执行的宏指令时，可打开对话框“Process Macro”（处理宏指令）。显示宏指令名和当前有效的命令行。可用按钮“Cancel”（取消）停止宏指令的处理。在这种情况下，随时可结束当前处理的命令。随后，在消息窗内显示一条相应的消息，并在“Online”（联机）操作期间的日志中出现：“<Macro>:Execution interrupted by user”（<宏指令>：被用户中断执行）。可以脱机和联机执行宏指令，但是在每种情况下，只有相应模式中提供的那些命令才能执行。

3.6 联机功能

可供使用的联机命令在菜单项“Online”（联机）下汇集。有些命令的执行取决于当前的编辑器。联机命令只有在登录后才可以使用的。

借助于“Online Change”（联机更改）功能，你可以在运行的控制器上，对程序进行更改。有关这方面的情况，请参见“Online”（联机）→“Log-in”（登录）

“Online”（联机）→“Login”（登录）快捷键： <F11>

该命令将编程系统与 PLC 结合在一起（或启动仿真程序）并转换为联机工作方式。

如果当前项目从打开以后、或从上次修改以后尚未编译，则应对其进行编译（使用命令“Project”（项目）→“Build”（建立））。如在编译过程中出现错误，则 TwinCAT PLC Control 不会转换为“Online”（联机）方式。

如果当前项目从上次装载以后，在控制器中进行了更改，但未关闭，而且上次装载的信息没有使用命令“Project”（项目）→“Clear all”（全部清除）进行清除，则在命令“Login”（登录）后，将打开对话框并提示：“The program has been changed.Load changes?(Online Change)”（程序已更改。装入更改吗？（联机更改））。回答 **Yes** 对它进行确认，则在登录时，项目的更改部分被装载到控制器中。回答 **No** 则表示登录时，不更改上次装载到控制器中的程序。使用“Cancel”（取消）取消该命令。<Load all>（全部装入）可将整个项目重新装载到控制器。

登录成功后，则可以使用所有联机功能（如果相应的设定已输入到“Project”（项目）→“Options”（选项）类别“Build”（建立）中）

使用“Online”（联机）→“Logout”（退出登录）命令从联机方式返回到脱机方式。

出错情况

错误：“不能建立与 PLC 的连接”。

TwinCAT PC: 检查你的 TwinCAT 系统是否运行（任务条中的 TwinCAT 图标应为绿色）。如果没有启动，则通过鼠标右键点击该图标，然后选择系统并启动 TwinCAT 系统。TwinCAT 图符的颜色从红色变为绿色。

TwinCAT BC: 检查“Online”（联机）→“Communication Parameter”（通讯参数）中选择的参数是否与你的 PLC 参数匹配。尤其需要注意检查接口号是否正确。（如果你将其设置为 COM1，则电缆也应和 COM1 进行物理插接）。此外，还应检查 PLC 和“Program System”（程序系统）中的波特率是否彼此相符。

错误: “程序已被修改！是否装入新程序？”

在编辑器中打开的项目与当前在 PLC 中找到的程序不匹配。因此，不可能进行监视和调试。你可以选择“No”，退出登录，然后打开正确的项目，或者使用“Yes”，将当前的项目装入到 PLC。

信息: “The program has been changed.Load changes?(Online Change)”（程序已更改。装入更改吗？（联机更改））。

项目正在控制器上运行。目标系统支持“Online Change”（联机更改），控制器上的项目已经被替换为最近装载的或最近“Online Change”（联机更改）的项目。现在你可以决定，是否要在控制器运行程序时装入这些改动，或者取消该命令。另外，你也可以通过选择“Load all”（全部装入）按钮来装入整个编译代码。

“Online”（联机）→“Logout”（退出登录）快捷键： <F12>

断开与 PLC 的连接。使用命令“Online”（联机）→“Login”（登入）转换到联机工作方式。

“Online”（联机）→“Download”（装载）

该命令将编译项目装入到 PLC 中（装载，不要与“Online”（联机）→“download source code”（装载源代码）相混淆！）。

装载信息保存在一个称为<项目名>000000ar.ri 的文件内，它在“Online Change”（联机更改）过程中使用，将当前的程序与最近装入到控制器的程序作比较，以便只将改变的程序部分进行重新装入。使用命令“Project”（项目）→“Clear all”（全部清除）可以删除该文件。

“Online”（联机）→“Run”（运行）快捷键： <F5>

该命令可启动 PLC 中的程序或“Simulation Mode”（仿真方式）中的程序。该命令可在“Online”（联机）→“Download”（装载）命令后立即执行，或通过“Online”（联机）→“Stop”（停止）命令结束 PLC 中的用户程序后执行，或当用户程序是在一个断点以及当“Online”（联机）→“Single Cycle”（单次循环）命令已执行时执行。

“Online”（联机）→“Stop”（停止）快捷键： <Shift>+>F8>

可在程序的两次循环之间停止执行 PLC 中的程序或“Simulation Mode”（仿真方式）中的执行。使用“Online”（联机）→“Run”（运行）命令重新启动程序。

“Online”（联机）→“Reset”（复位）

该命令复位（除了保持型变量 VAR RETAIN 之外）所有变量，使其返回到相应的特定值，即初始化值（包括那些声明为 VAR PERSISTENT 的变量！）。如果你已经使用了特定值将变量初始化，该命令就会将变量复位到该初始化值。所有其它变量都复位为标准的初始化值（例如，整数的初始化值为 0）。作为一种保护措施，TwinCAT PLC Control 在改写所有变量之前要求你对自己的决定进行确认。当电源故障或程序正在运行时将控制器停止，然后再启动（热启动），将会出现上述情况。使用“Online”（联机）→“Run”（运行）命令可以重新启动程序。

“Online”（联机）→“Reset (original)”（复位（原始状态））

该命令将包括持久保持型变量（PERSISTENT）在内的所有变量复位到其初始化值，并清除控制器上的用户程序。控制器返回到其原始状态。

“Online”（联机）→ “Toggle Breakpoint”（切换断点）快捷键： <F9>

使用该命令，可在当前窗口的目前位置上设置一个断点。如果在目前位置上已经设置了一个断点，则将清除该断点。可以设置断点的位置取决于激活窗口中编写 POU 所使用的语言。在文本编辑器（IL, ST）中，断点设置在光标位置的行上，如果该行是一个断点位置的话（以深灰色的行号字段识别）。你也可以在行号字段上点击来设置或清除文本编辑器中的一个断点。使用 FBD 和 LD 图形编辑语言时，断点设置在当前选择的网络处。为了在 FBD 或 LD 编辑器中设置或清除一个断点，也可在网络号字段点击。使用 SFC 语言时，断点设置在当前选定的步处。使用 SFC 语言时，你也可以双击鼠标和使用 <Shift> 按钮来设置或清除一个断点。如果已经设置了一个断点，则行号字段或网络号字段或步将以浅蓝色的背景颜色显示。如果程序运行时到达一个断点，则程序就会停止运行，相应的字段就会以红色背景颜色显示。为了继续运行程序，使用“Online”（联机）→“Run”（运行），“Online”（联机）→“Step in”（步进入）或“Online”（联机）→“Step Over”（单步）命令。

你也可以使用“Breakpoint”（断点）对话框来设置或清除断点。

“Online”（联机）→ “Breakpoint Dialogbox”（断点对话框）

使用该命令可打开一个对话框，对整个项目的断点进行编辑。该对话框也可以显示目前设置的所有断点。

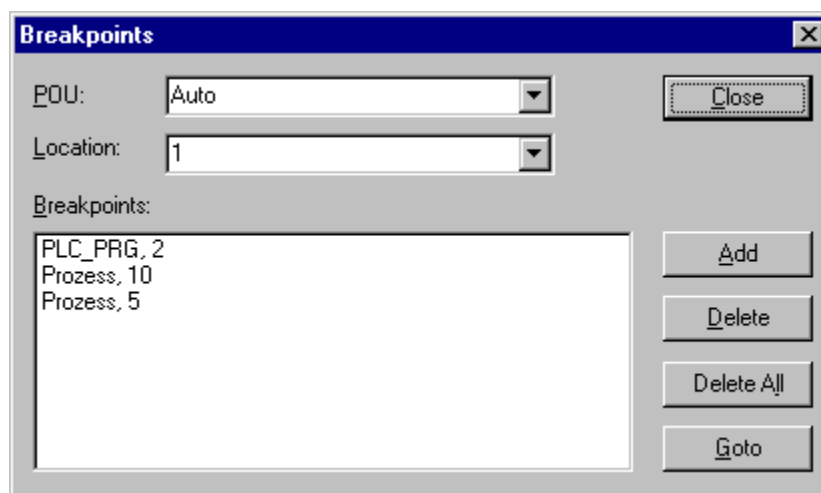
为了设置一个断点，在 POU 组合框中选择一个 POU，在你要设置断点的“Location”（位置）组合框中选择行或网络，然后按“Add”（添加）按钮。则断点将添加到断点设置表内。

为了删除一个断点，在高亮度显示断点的设置表中选择将要删除的断点，并按“Delete”（删除）按钮。

“Delete All”（全部删除）按钮可以删除全部的断点。

为了定位到编辑器中设置有某个断点的位置，在高亮度显示的断点设置表中选择相应的断点，并按“Go to”（转向）按钮。

要设置或删除断点，也可以使用“Online”（联机）→“Toggle Breakpoint”（切换断点）命令。



断点编辑对话框

“Online” →（联机）“Step Over”（单步）快捷键： <F10>

该命令导致执行单步。如果一个 POU 被调用，则执行该命令后程序被停止。对 SFC 而言，则执行一个完整的动作。如果当前的指令是调用一个功能或一个功能块，则将完整地执行该功能或功能块。使用“Online”（联机）→“Step In”（步进入）命令，以便定位到调用功能或功能块的第一个指令位置。如果已经到达调用功能或功能块的最后一条指令，则程序将转到 POU 的下一个指令。

“Online”（联机）→ “Step In”（步进入）快捷键： <F8>

单步执行一次，程序在调用 POU 的第一条指令前停止。如有必要，可以切换到打开的 POU 中。如果当前位置是调用的功能或功能块，该命令将会继续执行到调用的 POU 中的第一条指令位置。对所有其它情况而言，该命令和“Online”（联机）→“Step Over”（单步）的作用相同。

“Online”（联机）→“Single Cycle”（单循环）快捷键：<Ctrl>+<F5>

该命令执行一次 PLC 的单循环，并在执行该循环后停止。该命令可以不断重复地进行，以便继续进行单次循环。当执行“Online”（联机）→“Run”（运行）命令时，结束“Single Cycle”（单循环）。

“Online”（联机）→“Write Values”（写入值）快捷键：<Ctrl>+<F7>

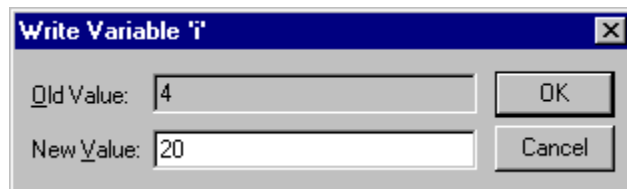
使用该命令，可以在一个循环开始处将一个或多个变量——只需一次！—设定为用户定义的数值。

可以更改所有单个成员变量的值，只要它们在“Monitoring”（监视）中是可视的。

在执行“Write values”（写入值）命令之前，必须先准备好要写入的变量值。

1. 值的定义

- 对于非布尔变量，当执行时，可以在声明变量的行上，或标记有变量的行上双击鼠标，或按 <Enter> 键。从而打开对话框“Write variable <x>”（写变量 <x>），并在该对话框中输入要写入的变量值。



写入新变量值的对话框

- 对于布尔变量，当在声明的变量行上双击鼠标时，其值将被切换（在 TRUE 和 FALSE 之间切换，不允许使用其它值）；并且不出现对话框。

“Writing”（写入）设定值在括号内显示，并位于原变量值之后以青绿色表示。例如 $a=0 \text{ <:=34>}$ 。

注：

例外：在 FBD 和 LD 编辑器中，该值以青绿色显示，变量名后面没有括号。

可为任意多个变量设定值。

为变量写入的输入值，也可以使用相同的方式进行更改或删除。这与命令“Online”（联机）→“Write/Force dialog”（写入/强制对话框）中可能出现的情况相类似（见下面）

前面提到的写入值保存在一个“writelist”（写入表）（“Watchlist”（监视表））内，在它们被命令“Force values”（强制值）确定地写入、删除或传送到一个强制表之前，一直保留在写入表中。

2. 值的写入

命令“Write Values”（写入值）可在两个地方找到：

- 菜单“Online”（联机）中的命令“Write Values”（写入值）。
- “Editing the writelist and the forcelist”（编辑写入表和强制表）对话框中的按钮“Write Values”（写入值）。

当执行“Write values”（写入值）命令时，所有在写入表中的值，在循环开始时一次性地写入到控制器中相应的变量，然后从写入表中删除（如果执行命令“Force values”（强制值），要强制的变量也从写入表中删除，并被传送到强制表！）

注:

使用顺序功能图语言 (SFC) 时, 由各个值汇集而成的转换表达式不能用 “Write values” (写入值) 命令更改。这是因为, 监视表达式的 “Total value” (总值), 并不是各个变量的显示值 (例如, “a AND b” 如果两个变量实际上都有值 TRUE, 才显示为 TRUE (真))。

另一方面, 使用 FBD 语言时, 在一个表达式中, 只有第一个变量 (例如, 作为功能块的输入) 被监视。从而 “Write values” (写入值) 命令只能用于这个变量。

“Online” (联机) → “Force values” (强制值) 快捷键: <F7>

使用该命令, 一个或多个变量可永久性地设置为用户定义的值。该设置可以在运行系统的循环开始和循环结束时进行。

一个循环中的时间顺序为: 1. 读取输入, 2. 强制值, 3. 处理代码, 4. 强制值, 5. 写输出。

上述功能一直保持激活, 直到被用户明确地暂停运行, 或编程系统退出登录时为止 (命令 “Online” (联机) → “Release force” (解除强制))。

为了设置新的值, 首选应创建一个写入表, 其过程和 “Online” (联机) → “Write values” (写入值) 所描述的一样。写入表中包含的变量相应地在 “Monitoring” (监视) 中标记。只要执行命令 “Online” (联机) → “Force values” (强制值), 写入表就立即被传输到一个强制表中。有可能出现已经存在一个激活的强制表情况, 此时, 可以按照需要进行更新。写入表被清空, 新值使用红色显示, 并作为其强制值。下一次使用 “Force values” (强制值) 命令时, 可将修改的强制表传输到程序。

注:

强制表是写入表中的变量首次强制时创建的, 而写入表则是在表写入第一个变量之前就已经存在的。

强制一个变量的命令意味着, 在以下菜单或对话框中找到的变量才能输入到强制表:

- “Online” (联机) 菜单中的命令 “Force Values” (强制值)
- “Editing the writelist and the forcelist” (编辑写入表和强制表) 对话框中的 “Force Values” (强制值) 按钮

注:

在使用顺序功能图语言时, 由各个值汇集而成的转换表达式的值, 不能用 “Force values” (强制值) 命令更改。这是因为, 显示监视值是其表达式的 “Total value” (总值), 而不是各个变量的值 (例如, “a AND b” 仅显示为 TRUE (真), 如果两个变量实际上都有值 TRUE 的话)。

另一方面, 使用 FBD 语言时, 在表达式中, 只有第一个变量 (例如, 作为功能块的输入) 被监视。因此, “Force values” (强制值) 命令只能用于这个变量。

“Online” (联机) → “Release Force” (解除强制) 快捷键: <Shift>+<F7>

该命令结束控制器中的变量强制。变量值又可以使用正常方式进行更改。

强制变量可以在 “Monitoring” (监视) 框中使用红色显示的值进行识别。你不仅可以删除整个强制表, 而且也可以标记要解除强制的单个变量。

为了删除整个强制表, 则必须对所有变量解除强制。可采用以下方法中的一种:

- 菜单 “Online” (联机) 中的命令 “Release Force” (解除强制)。
- “Editing the writelist and the forcelist (编辑写入表) / (强制表)” 对话框中的 “Release Force” (解除强制) 按钮。
- 使用对话框 “Remove Write-/Forcelist” (清除写入表/强制表) 中的 “Release Force” (解除强制) 命令删除全部强制表。如果你选择命令 “Release Force” (解除强制), 同时还

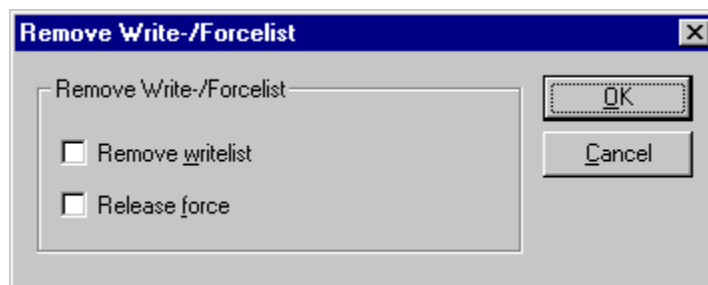
存在一个写入表时，则打开该对话框。

为了解除**单个变量**的强制，你首先应对这些变量进行标记。可以按照下面描述的方法进行。之后，所选择的变量使用青绿色扩展符 **<Release Force>**（解除强制）进行标记：

- 在声明为非布尔的变量行上双击鼠标时，打开对话框“**Write variable <x>**”（写入变量 <x>）。按压“**<Release Force for this variable>**”（解除该变量强制）按钮。
- 在声明为布尔变量的行上重复双击鼠标时，则将在该行的结束位置切换显示“**<Release Force>**” <解除强制>。
- 在“**Online**”（联机）菜单中打开“**Write/Force-Dialog**”（写入/强制对话框），删除位于“**Forced value**”（强制值）一栏中的编辑字段值。

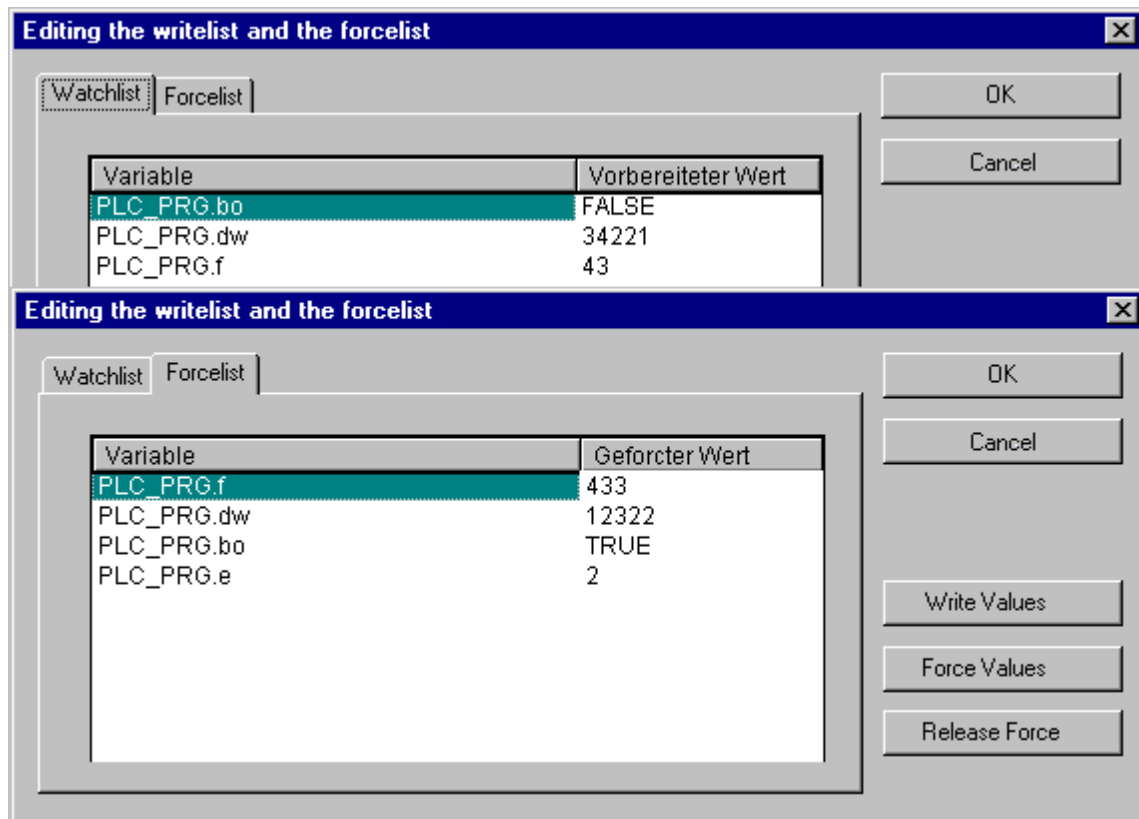
当所有需要“<解除强制>”的变量设置在声明窗中显示时，选择命令“**<Force>**”（强制），从而将强制表中的修改值传输到程序中。

如果你执行命令“**<Release Force>**”（解除强制）时，当前的写入表（见“**Online**”（联机）→“**Write Values**”（写入值））不为空的话，对话框“**Remove Write-/Forcelist**”（清除写入表/强制表）将被打开。用户可以在这里决定是否“**Release Force**”（解除强制），或者另外想要“**Remove the writelist**”（清除写入表），或者清除两个表。



“**Online**”（联机）→“**Write/Forcen Dialog**”（写入/强制对话框）

该命令可打开一个对话框，它在两个属性页中分别显示当前的写入表（**Watchlist**）和强制表（**Forcelist**）。每个变量的名称和写入/强制值均显示在一个表内。



变量通过命令“**Online**”（联机）→“**Write Values**”（写入值）到达监视表，并通过命令“**Online**”（联机）→“**Force Values**”（强制值）传送到强制表。通过鼠标点击一个登入项，打开一个编辑器字段，可在“**Prepared Value**”（预设值）或“**Forced Value**”（强制值）栏内编辑变量的值。如果登入项的类型不一致，则显示一个出错消息。如果一个值被删除，则表示从写入表中删除了这个登入项，或者，除“**Cancel**”（删除）命令外的任何其它命令，均可关闭对话框来通告变量已暂停强制。

以下命令对应于“**Online**”（联机）菜单中的命令，可由按钮提供使用：

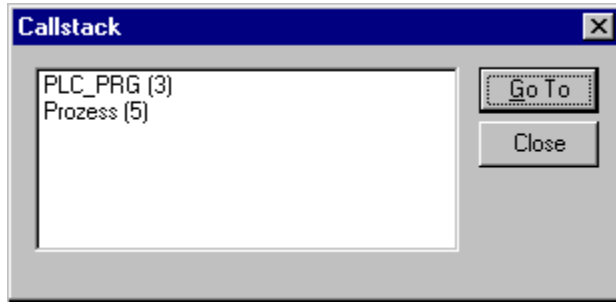
“**Force Values**”（强制值）：当前写入表中的所有条目都被传送到强制表中，即控制器中的变量值被强制。所有标记为“**Release Force**”（解除强制）的变量将不再被强制。对话框随后被关闭。

“**Write Values**”（写入值）：当前写入表中的所有条目被一次性地写入到控制器相应的变量中。对话框随后被关闭。

（**Release Force**）（解除强制）：强制表中的所有条目将被删除，或者，如果存在一个写入表，则出现对话框“**delete write/forcelist**”（删除写入/强制表），用户可在此决定是否解除强制，或删除写入表，或将两者都删除。对话框随后将被关闭，或根据选择后的不同情况，关闭对话框。

“**Online**”（联机）→“**Show Call Stack**”（显示调用栈）

当 PLC 在一个断点处停止运行时，可以执行该命令。你将获得一个当前 POU 的“**Call Stack**”（调用栈）对话框。



调用栈示例

第一个 POU 总是指定为调试任务的调用程序，因为这是开始执行的地方。

最后一个 POU 总是正在执行的 POU。当你已经选择了一个 POU 之后，并已按“Go to”按钮，选出的 POU 被装入到它的编辑器内，并显示正在处理的行或网络。

“Online”（联机）→ “Flow Control”（流控制）

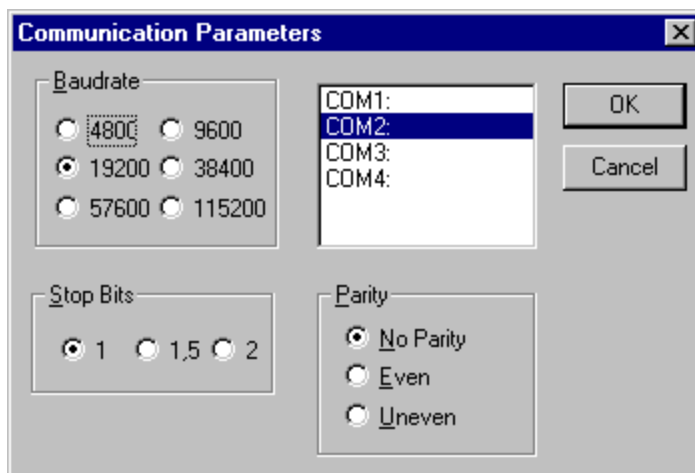
如果你已选择“flow control”（流控制），则在菜单项前面会出现一个对勾（√）。随后，每行或每个网络所作的标记，表示它们已在最后一次 PLC 循环中执行。运行的行号或网络号字段以绿色显示。在 IL 编辑器中还有一个附加字段，显示累加器的当前内容。在功能块图和梯形图编辑器中，则插入了一个不传输任何布尔值的连接线附加字段。当验证这些输出和输入时，则在这个字段中显示连接线的传输值。连接线仅传输布尔值，当传输值为 TRUE 时，则以阴影蓝色表示。这样可以始终监视信息流。

“Online”（联机）→ “Simulation”（仿真）

“Simulation Mode”（仿真模式）只是用于总线控制器（BC），而不是用于 TwinCAT PC。如果选择了“Simulation Mode”（仿真模式），则在菜单项前面会出现一个对勾（√）。在仿真模式下，用户程序运行在 Windows 下的同一台 PC 中。该模式用来测试项目。PC 和“Simulation Mode”（仿真模式）之间的通讯使用 Windows 消息机制。如果程序不处于仿真模式，则程序运行于 PLC 中。该标志的状态保存在项目中。

“Online”（联机）→ “Communication Parameters”（通讯参数）

通过串行接口的传输参数可以在一个对话框中输入。重要的是，这些参数应与 PLC 中输入的参数相匹配。



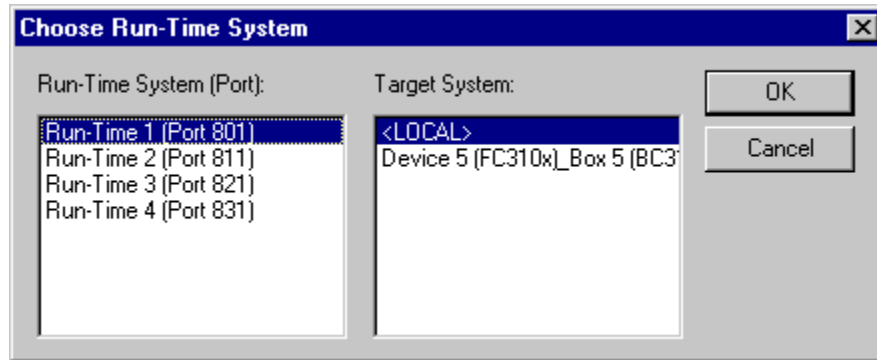
通讯参数输入对话框

可调整的参数包括：波特率；传输是否带有“Even”（偶校验）、“Odd”（奇校验），或“No Parity”（无校验）；停止位的个数；以及进行传输的接口（COM1，COM2，等等）。选择的参数保存在项目中。

“Online”（联机）→ “Choose Runtime System”（选择运行系统）

仅对 TwinCAT PC:

你可以选择一个运行系统。可以看到本地（最大为 4 个）和远程操作的运行系统。



运行系统的信息保存在 PLC 项目的文件中。因此，当重新扫描 TwinCAT 系统管理程序配置后，可选择的端口号是可见的。

“Online”（联机）→ “Sourcecode download”（源代码装载）

该命令将项目的源代码装入到控制器系统内。这不能与项目编译时所生成的代码相混淆！你可在“Project”（项目）→ “Options”（选项）→ “Sourcedownload”（源代码装载）对话框中输入用于“Download”（装载）（时间，大小）的选项。

“Online”（联机）→ “Create bootproject”（创建引导项目）

使用该命令（已联机），已编译的项目将在控制器上以这种方式创建引导，即控制器再启动时，可以自动装入项目运行。取决于目标系统，引导项目的存储方式不同。例如，对于 PC 平台，则在 TwinCAT 引导目录中创建一个文件：TCPLC_P_x.wbp（X 是运行系统序号，取值为 1 到 4）。

“Online”（联机）→ “Write file to controller”（将文件写入到控制器）

该命令用来将任何所需要的文件装入到控制器。它打开“Write file to controller”（将文件写入到控制器）对话框，你可以在此选择所需要的文件。在使用“Open”（打开）按钮将对话框关闭后，文件装入到控制器中，并在相同的名称进行保存。装入过程伴随有一个进程对话框。

使用命令“Online”（联机）→ “Load file from controller”（从控制器装入文件），你可以检索以前装入在控制器中的一个文件。

“Online”（联机）→ “Load file from controller”（从控制器装入文件）

使用该命令，你可检索以前通过命令“Online”（联机）→ “Write file to controller”（将文件写入到控制器）装入到控制器中的一个文件。你可以看到“Load file from controller”（从控制器装入文件）对话框。在“Filename”（文件名）下，提供所需要的文件名，并在选择窗内输入你的计算机用于装入的目录。可以使用“Save”（保存）按钮关闭该对话框。

如果目标系统是 BCxxxx，则启动以下菜单项

“Online”（联机）→ “coupler”（藕合器）

以下是用于 TwinCAT BC 使用的专用命令：

“**K-Bus Reset**”（K 总线复位）：完成端子总线复位。

“**Coupler Reset**”（耦合器复位）：耦合器将重新启动

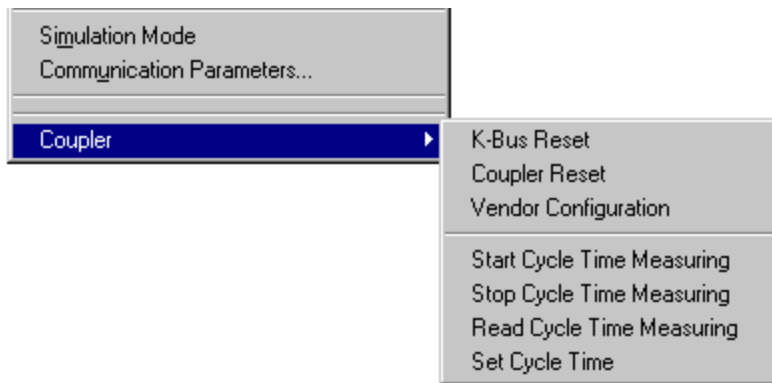
“**Vendor Configuration**”（制造商配置）：装入出厂默认设置值（清除引导项目），耦合器必须重新启动，使耦合器初始复位。

“**Start Cycle Time Measuring**”（启动循环时间测量）：启动 BCxxxx 的循环时间测量。

“**Stop Cycle Time Measuring**”（停止循环时间测量）：停止 BCxxxx 的循环时间测量。

“**Read Cycle Time Measuring**”（读循环时间测量）：读出最小循环时间（从测量开始以来）、最大循环时间、平均循环时间（取自最后的 200 个循环）、实际循环时间和 PLC 循环次数。

“**Set Cycle Time**”（设定循环时间）：可以在此设定目标的循环时间和后台任务时间。如果实际循环时间超过目标循环时间，则下一次循环时间将相应延迟（循环时间不再是常数）。通常，目标循环时间的计算如下：1.25 倍平均循环时间（从循环测量读出），后台任务时间：0.25 倍平均循环时间。指定的时间设定值可精确到 1ms。



3.7 窗口

在“Window”（窗口）菜单项下，可以找到用于管理窗口的所有命令。例如，自动建立窗口、打开库管理器、以及用于在打开的窗口之间进行切换。在菜单的结束处，你会发现一个按顺序排列的所有打开窗口表。通过点击相应的条目，你可以切换到所需要的窗口。在激活的窗口前会出现一个对勾（✓）。

“Window”（窗口）→ “Tile Horizontal”（水平排列显示）

使用该命令，你可在工作区内水平布置所有窗口。它们不会彼此重叠，并将填满整个工作区。

“Window”（窗口）→ “Tile Vertical”（垂直排列显示）

使用该命令，你在工作区内可垂直布置所有窗口，它们不会彼此重叠，并将填满整个工作区。

“Window”（窗口）→ “Cascade”（叠放显示）

使用该命令，你能在工作区内以叠放方式，一个接一个地布置所有窗口。

“Window”（窗口）→ “Arrange Symbols”（排列图标）

使用该命令，你能在工作区的下端按行排列所有最小化的窗口。

“Window” (窗口) → “Close all” (关闭所有)

使用该命令，你可关闭在工作区中所有打开的窗口。

“Window” (窗口) → “Messages” (消息) 快捷键: <Shift>+<Esc>

使用该命令，你可打开或关闭最近一次编译、验证或比较过程的消息窗。如果消息窗是打开的，则在该命令前面将出现一个对勾 (✓)。

“Window” (窗口) → “Library Manager” (库管理器)

使用该命令，你可打开或关闭库管理器。

“Window” (窗口) → “Log” (日志)

使用该命令，你可打开或关闭“Log” (日志) 窗口，此窗口可显示联机会话的相关事务。

3.8 帮助系统

如果你在工作中遇到有关 TwinCAT PLC Control 的任何问题，可利用联机帮助系统帮助你解决问题。在那里你可以找到包括在该手册中的所有资料。帮助直接引用 TwinCAT 信息系统。因此，必须安装 TwinCAT 信息系统。

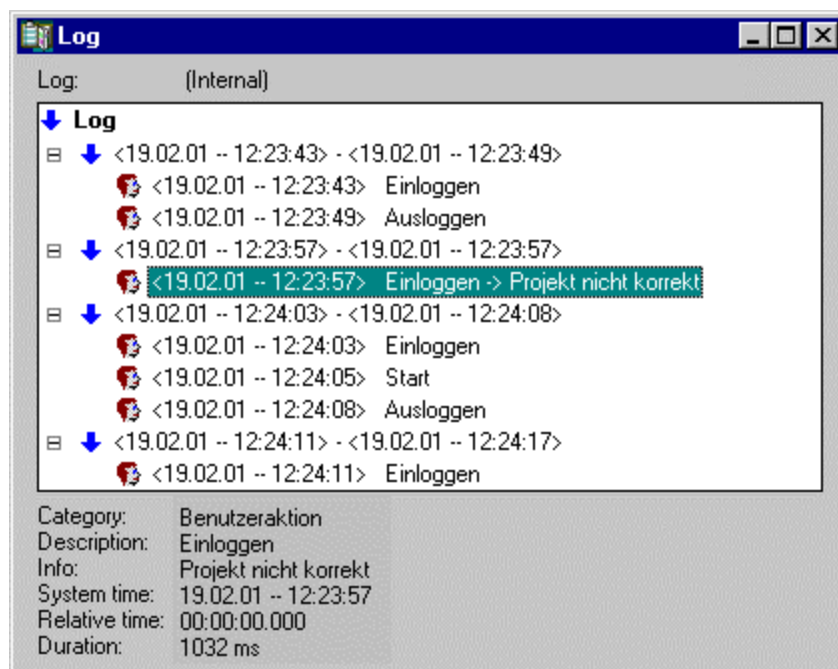
3.9 日志

日志用于保存“Online”（联机）会话过程中按时间顺序出现的各种动作。为此，应建立一个二进制日志文件（*.log）。在这之后，用户可将来自相应项目日志的摘录保存到一个外部日志中。

可以使用“Offline”（脱机）或“Online”（联机）方式打开日志窗，从而作为一种直接的联机监视。

“Window”（窗口）→ “Log”（日志）

选择菜单命令“Window”（窗口）→ “Log”（日志），即可打开。



在日志窗中，当前显示日志的文件名在“Log”（日志）后出现：如果这是当前项目的日志，则显示文字“（Internal）”（内部）。

登录项在日志窗内显示。最新的登录项总是出现在底部。

只有在菜单“Project”（项目）→ “Options”（选项）→ “Log”（日志）的“Filter”（筛选）字段中选中的动作才被显示。有关当前选择的登录项的可用信息在日志窗下部显示：

类别：属于特定日志登录项的类别。可能有以下四种类别：

- **“User action”（用户动作）：**用户已执行的“Online”（联机）动作（典型情况来自联机菜单）。
- **“Internal action”（内部动作）：**“Online”（联机）时已经执行的内部动作（例如删除缓冲区或调试初始化）。
- **“Status change”（状态改变）：**运行系统的状态已经变化（例如，到达一个断点，则从“Running”（运行）转换为“Break”（中断运行））。
- **“Exception”（异常）：**发生异常，例如通讯错误。

“Description”（说明）：动作的类型。用户动作名与其相应的菜单命令名相同；所有其它的动作都用英文表示，而且与其相应的联机 XXX() 功能有相同的名称。

“**Info**”（信息）：该字段包含一个错误描述，即一个动作期间也许发生的错误。如果没有发生错误，该字段为空。

“**System time**”（系统时间）：动作开始的系统时间，以最接近的秒为单位计算时间。

“**Relative time**”（相对时间）：从“**Online**”（联机）会话开始进行测量的时间，以最接近的毫秒为单位计算时间。

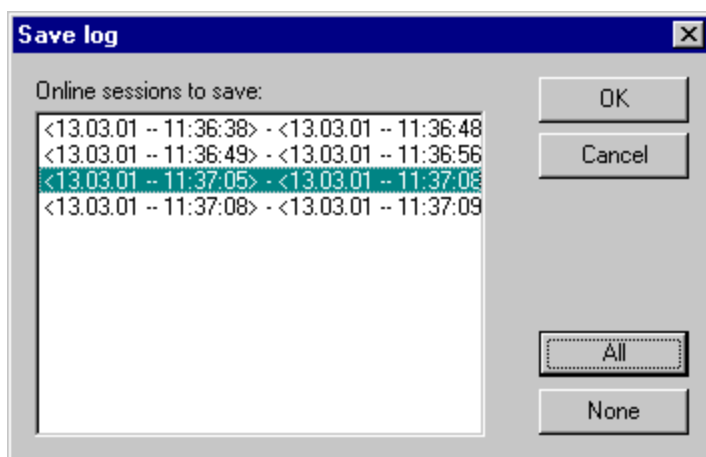
“**Duration**”（持续时间）：以毫秒为单位计算的动作持续时间。

“Load...”（装入...）

使用标准的文件打开对话框可装入和显示一个外部的日志文件 *.Log。出现在项目中的日志不会被该命令所覆盖。如果日志窗被关闭，然后又重新打开，或开始一个新的“**Online**”（联机）会话，则装入的日志版本将再次被项目日志所替代。

“Save...”（保存...）

这个菜单项仅在当前显示项目日志时才可选择。它允许将一个项目日志的摘录保存到一个外部文件中。为此，显示以下的对话框，在该对话框中，可选择要保存的“**Online**”（联机）会话：



成功作出选择后，保存文件的标准对话框（“**Save Log**”（保存日志））被打开。

“Display Project Log”（显示项目日志）

该命令仅在当前显示一个外部日志时才可选择。它将显示返回到项目日志。

“Storing the project log”（保存项目日志）

不论日志是否保存到一个外部文件（见上面），项目日志自动保存在一个名为 <项目名>.log 的二进制文件中。如果在“**Project**”（项目）→“**Options**”（选项）→“**Log**”（日志）对话框中没有显式地给出一个不同的路径，则该文件保存在与保存项目相同的目录内。

保存“**Online**”（联机）会话的最大条目数可在“**Project**”（项目）→“**Options**”（选项）→“**Log**”（日志）对话框中输入。如果在记录过程中超过该值，则最早的会话将被删除，以便为最新的会话留出空间。

4 编辑器

4.1 概述

POU（程序组织单元）的所有编辑器由一个声明部分和一个本体组成。这两个部分被屏幕分隔器隔开，如有必要可用鼠标点击，将其拖动或上、下移动。本体可由其它文本或图形编辑器组成；声明部分总是一个文本编辑器。

打印边界

如在对话框“**Workspace**”（工作区）中的项目选项中选择“**Show print range**”（显示打印范围）选项，则当打印编辑器内容时，以红色虚线表示其垂直和水平边界。所使用的打印机属性，以及打印布局的尺寸，都是在“**File**”（文件）→“**Printer Setup**”（打印机设定）菜单中选择的。如果没有设定打印机，或没有输入打印布局，则可采用默认的配置（Default.DFR 和默认打印机）。如在“**Documentation settings**”（文档设定）中选择选项“**New page for each object**”（每个对象的新页）或“**New page for each sub-object**”（每个子对象的新页），则只显示水平边界。不显示底部边界。

注：

仅当选择缩放系数为 100% 时，才有可能准确地显示打印边界。

注释

用户注释必须包含在专用符号“**(*)**”和“**(**)**”之内。示例：（* 这是一个注释。*）

注释可以放在所有文本编辑器内，在任何需要的地方，也就是说，在所有的变量声明、IL 和 ST 语言，以及在自定义的数据类型内。如果“**Project**”（项目）是用模板（**template**）打印输出的，则在声明变量时输入的注释，在每个变量后以基于文本的程序成分出现。

在 FBD 和 LD 图形编辑器中，可为每个网络输入注释。为此，搜索你要注释的网络，并启动“**Insert**”（插入）→“**Comment**”（注释）命令。对 CFC 语言，有专用的 POU 注释，这个 POU 可任意放置。

在 SFC 语言，你可在编辑步属性的对话框中，输入有关步的注释。

如果启动“**Project**”（项目）→“**Options**”（选项）→“**Build Options**”（建立选项）对话框中相应的选项，还允许采用嵌套式的注释。在“**Online mode**”（联机模式）时，如果你将鼠标光标停留在一个变量上一段时间，则会在工具提示条中显示该变量的类型以及可应用的地址和注释。

“Zoom to POU”（对 POU 的缩放）

使用该命令，将选择的 POU 装载到它的编辑器内。如果光标放置在文本编辑器中的一个 POU 名称上，或者，如果 POU 框是在图形编辑器中选出，也可以使用上下文菜单（<F2>）或“**Extras**”（附加）菜单完成。如果你正在处理库中的 POU，则库管理器被打开，并显示相应的 POU。

打开实例

该命令相应于“**Project**”（项目）→“**Open instance**”（打开实例）。如果光标置于文本编辑器中的一个功能块名称上，或如果功能块框在图形编辑器中选出，也可以使用上下文菜单或“**Extras**”（附加）菜单选择该命令。

智能感应功能

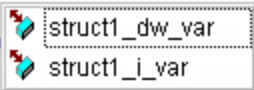
如果在类别“**Editor**”（编辑器）的项目选项对话框中，启动选项“**List components**”（成员列表），则可将“**Intellisense**”（智能感应）功能作用于所有的编辑器、“**Watch- and Receiptmanager**”（监视和接收管理器）、“**Visualization**”（可视化）系统以及“**Sampling Trace**”（抽样跟踪）：

- 如果你使用一个点“.”替换一个标识符插入，则会出现一个列出项目的所有局部变量和全局变量的选择框。你可以选择这些成员中的一个，并按“**Return**”（返回）键将它插入到点的后面。你也可以通过双击表的列表项来插入这个成员。
- 如果你在一个功能块实例或一个结构变量后输入一个点操作符，则会显示一个选择框，在框内列出相应功能块的所有输入和输出变量或列出结构的成员变量，在这个框中，可以选择你需要的成员然后通过按“**Return**”（返回）键或通过双击表项完成输入。

示例:

插入“struvar.” → 将提供结构 struct1 的成员变量

0001	PROGRAM ST_EXAMPLE
0002	VAR
0003	struvar:struct1;
0001	struvar.
0002	
0003	b1:=((D
0004	



4.2 声明编辑器

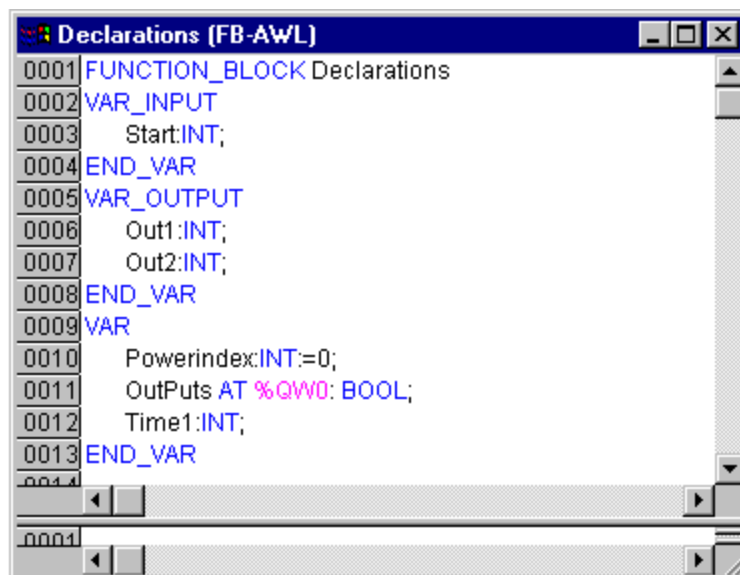
声明编辑器用来声明 POU 的变量和全局变量，用于数据类型声明，以及用于“Watch and Receipt Manager”（监视和接收管理器）。它提供对常用 Windows 功能的存取，如果安装了相应的驱动程序，甚至可以使用“IntelliMouse”（智能鼠标）的功能。

在“Overwrite”（改写）模式，状态条上以黑色显示“OV”；使用 <Ins> 键，可在“Overwrite”（改写）和“Insert”（插入）方式之间进行切换。声明的变量由语法颜色支持。

最重要的命令可以在上下文菜单（鼠标右键）中找到。

“Declaration Part”（声明部分）

只用于该 POU 的所有变量都在 POU 的声明部分进行声明。这些变量包括：输入变量，输出变量，输入/输出变量，局部变量，保持型变量和常数。变量声明的语法基于 IEC61131-3 标准。以下是 TwinCAT PLC Control 编辑器中正确声明变量的一个示例：



声明编辑器

“Input Variable”（输入变量）

在关键字 VAR_INPUT 和 END_VAR 之间，所有变量均被作为一个 POU 的输入变量进行声明。这意味着在调用位置，变量的值由调用给出。

示例：

```
VAR_INPUT
    in1:INT; (* 1. Inputvariable*)
END_VAR
```

“Output Variable”（输出变量）

在关键字 VAR_OUTPUT 和 END_VAR 之间，所有变量均被作为一个 POU 输出变量进行声明。这意味着，这些值都返回到作出调用的 POU。在那里，它们可以被应答和作其它使用。

示例：

```
VAR_OUTPUT
    out1:INT; (* 1. Outputvariable*)
END_VAR
```

“Input and Output Variable”（输入和输出变量）

在关键字 VAR_IN_OUT 和 END_VAR 之间，所有变量均被作为一个 POU 输入和输出变量进行声明。

注:

通过这个变量，可更改被传送的变量值（作为一个指针传送，参考调用）。这意味着，这种变量的输入值不能是一个常数。鉴于此，一个功能块的 VAR_IN_OUT 变量甚至不能通过 <functionblockinstance> <in/outputvariable>（功能块实例，输入/输出变量）直接从外部进行读取或写入。

示例:

```
VAR_IN_OUT
    inout1:INT; (* 1. Input/Outputvariable *)
END_VAR
```

“Local Variables”（本地变量）

在关键字 VAR 和 END_VAR 之间，所有变量均被作为一个 POU 的本地变量进行声明。这些变量没有外部连接；换句话说，它们不能从外部进行操作。

示例:

```
VAR
    loc1:INT; (* 1. Local Variable*)
END_VAR
```

“Remanent Variables”（保持型变量）

保持型变量可以在通常的程序运行周期内保留其值。包括一般保持型变量和持久保持型变量。

- 保持型变量以关键字“**RETAIN**”（保持）标识。这些变量即使是在控制器未经控制的停机之后，以及在控制器的正常断开和接通之后（相当于在命令“**Online**”（联机）→“**Reset**”（复位））仍保持其值。当程序重新运行时，将进一步处理保存的值。一个具体的例子是在一条生产线上的零件计数器，断电恢复后，从保持值开始重新计数。与持久保持型变量不同，一般保持型变量在新装载程序时，重新初始化。

所有其它变量都是重新初始化的，要么是其初始化值、要么是标准初始化值。

示例:

```
VAR RETAIN
    rem1:INT; (* Retain Variable*)
END_VAR
```

注:

- 如果一个本地变量声明为“**VAR RETAIN**”（一般保持型变量），则这个变量将正确地保存在“保持区”（类似于一个全局保持型变量）
- 如果一个功能块的一个局部变量声明为 **VAR RETAIN**，则功能块的完整实例将保存在“保持区”（POU 的所有数据），但是，只有声明为保持型的变量将作为保持处理。
- 如果一个功能的一个局部变量声明为 **VAR RETAIN** 或 **VAR_PERSISTENT**，则没有任何效果。

“Persistent Variables”（持久保持型变量）

另一类保持型变量是“**PERSISTENT Variables**”（持久保持型变量）。这种变量和完整的符号保存在一起。因此，必须选择符号生成。使用“**RETAIN**”保存的变量是在 PLC 程序的“**Rebuild all**”（重建所有）之后进行初始化。而持久保持型变量保持其原有的值。使用“**Reset**”（复位），所有一般保持型变量均被初始化。为了初始化持久保持型变量，选择“**Reset all**”（全部复位）。当 TwinCAT 停机时，持久保持型变量被写入到一个专用文件中。这个文件包含持久保持型变量原有的值，并在 TwinCAT 启动时被读出。

示例：

```
VAR PERSISTENT
    Rem2:INT; (* Persistent Variable*)
END_VAR
```

注意：为了检查持久保持型变量和一般保持型变量的正确性，系统变量在 **SYSTEMINFO**（系统信息）结构中包含一个字节 **bootDataFlags**（引导数据标记）。该字节指示变量装入后的数据引导状态。字节的高四位指示持久保持型数据的状态，而字节的低四位指示一般保持型数据的状态。为了使用这个信息必须链接“**PlcSystem.lib**”库。

位号	说明
0	一般保持型变量：装载（没有错误）
1	一般保持型变量：无效（已装载后备拷贝，然而不存在有效数据）
2	一般保持型变量：请求（应装入一般保持型变量，TwinCAT 系统控制的一个设定值）
3	保留
4	持久保持型变量：装载（没有错误）
5	持久保持型变量：无效（已装载后备拷贝，然而不存在有效数据）
6	保留
7	保留

当 TwinCAT 停机时，持久保持型数据和一般保持型数据被分别写入到硬盘上的两个文件中。通过 TwinCAT 系统属性（PLC 属性页）可在 TwinCAT 系统控制中指定路径。标准的设置为“<Drive>\TwinCAT\Boot.”所有文件都有一个固定的文件名和扩展名：

文件名	说明
TCPLC_P_x.wbp	引导项目（X=运行系统的编号）
TCPLC_R_x.wbp	一般保持型变量（X=运行系统的编号）
TCPLC_T_x.wbp	持久保持型变量（X=运行系统的编号）
TCPLC_R_x.wb~	一般保持型变量的后备拷贝（X=运行系统的编号）
TCPLC_T_x.wb~	持久保持型变量的后备拷贝（X=运行系统的编号）

当 TwinCAT 停机时，如果持久保持型变量和/或一般保持型变量不能写入到文件中，标准的反应是装入备用文件。在这种情况下，bootDataFlags 的位 1（对于一般保持型变量）和/或位 5（对于持久保持型变量）被置位。

如果在任何条件下都不能使用后备文件，则应在 NT 注册表中进行设置。即在注册表编辑器中，找到以下键值：

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Beckhoff\TwinCAT\Plc]
"ClearInvalidRetainData"=dword:00000000
"ClearInvalidPersistentData"=dword:00000000
```

“ClearInvalidRetainData”的值，或“ClearInvalidPersistentData”的值，必须设置为 1。默认的设置 0。

常数，类型符

常数以关键字 **CONSTANT** 标识。可对它们进行本地或全局声明。

语法：

```
VAR CONSTANT
    <Identifier>:<Type> := <initialization>;
END_VAR
```

示例：

```
VAR CONSTANT
    con1:INT:=12; (* 1. Constant*)
END_VAR
```

你可以在附录中找到一些可用的常数表。参见该附录，请注意使用可能的类型常数（类型符）。

“External Variables”（外部变量）

输入到 POU 的全局变量以关键字 **EXTERNAL**（外部）命名。它们也出现在“Online mode”（联机模式）的声明部分“Watch”（监视）窗口内。如果 VAR_EXTERNAL 声明在任何方面都与全局声明不匹配，则出现以下的出错信息：“<VAR> 的声明与全局声明不匹配！”如果全局变量不存在，则出现以下出错信息：“未知的全局变量：<var>！”

示例：

```
VAR EXTERNAL
    var_ext1:INT:=12; (* 1st external variable *)
END_VAR
```

关键字

在所有的编辑器中，关键字都必须大写。关键字不能作为变量使用。

变量声明

变量声明使用以下语法：

```
<Identifier> {AT <Address>}:<Type> {:=<initialization>};
```

括号{}内的部分为选项。

就标识符而言，它是一个变量名，应注意的是，它不能包含空格或有元音变化的字符，不能重复声明，而且不能与任何关键字相同。不考虑变量的大/小写，换句话说，VAR1, Var1 和 var1 都是没有差别的。标识符中的下划线是有意义的，例如 A_BCD 和 AB_CD 被识别为不同的标识符。不允许在一个标识符的开始处或在标识符内连续使用多个下划线。

标识符有意义部分的长度是不受限制的。

变量和数据类型成员的所有声明均可包括初始化。它们由“:=”操作符引导。对于基本类型变量而言，这些初始化值是常数。对所有声明的变量而言，默认的初始化值为 0。

示例：

```
var1:INT:=12; (* Integer variable with initial value of 12*)
```

如果你需要将一个变量直接链接到一个确定的地址，那么，你必须用关键字 **AT** 来声明这个变量。为了更快地输入声明，可使用快捷键方式。在功能块中，你也可以使用不完整的地址语句来指定变量。为了使这种变量能够用于一个本地实例，在变量配置时，必须有一个输入项。

请注意自动声明的可能性！

AT 声明

如果你需要将一个变量直接链接到一个确定的地址，你必须以关键字 **AT** 来声明这个变量。这个过程的优势是，你可将一个有意义的名称分配给一个地址，而一个输入或输出信号的任何必要的变化只需在一个地方进行（例如在声明内）。请注意，需要有一个输入的变量不能进行写操作。另一个约束条件是，**AT** 声明只能用于本地变量和全局变量，不能用于来自 **POU**（程序组织单元）的输入变量和输出变量。

示例：

```
counter_heat7 AT %QX0.0: BOOL;  
lightcabinetimpulse AT %IX7.2: BOOL;  
download AT %MX2.2:BOOL;
```

如果将布尔变量分配给一个字节、字或双字地址，则它们占用带有 **TRUE** 或 **FALSE** 的一个字节，而不一定是偏移量后的第一个位！

“Insert”（插入）→ “Declarations keywords”（声明关键字）

你可以使用该命令打开一个有全部关键字的表。这个表可用于一个 **POU** 的声明部分。当一个关键字被选中并确认后，该关键字将被插入到当前光标位置。当你打开“Input Assistant”（输入助手）并选择“Declarations”（声明）类别时，你也会接收到这个表。

“Insert”（插入）→ “Types”（类型）

使用该命令，你将收到用于变量声明的可能类型的选择。当你进入“Input Assistant”（输入助手）(<F2>) 时也会收到这张表。

类型被划分为以下几种类别：

- 标准类型：BOOL、BYTE 等
- 自定义类型：结构，枚举类型等。
- 标准功能块的实例声明。
- 自定义功能块的实例声明。

TwinCAT PLC Control 支持 IEC61131-3 的所有标准类型。在附录中可以找到使用变量类型的例子。

语法着色

在文本编辑器和声明编辑器中，你可以看到以直观方式支持的变量实施和声明。因为文本是着色显示的，这样可以避免错误或尽快地发现错误。

如果遗留了一个没有闭合的注释，这样会注释指令，它能立即引起注意；例如关键字将不会由于不小心而拼错，等等。

采用以下的颜色突出显示：

蓝色	关键字
绿色	注释
粉红色	布尔值 (TRUE/FALSE)
红色	输入出错 (例如, 无效的时间常数, 关键字, 以小写字母写入, ...)
黑色	变量, 常数, 赋值操作符, ...

快捷键模式

TwinCAT PLC Control 的声明编辑器允许你使用快捷键模式。当你用 <Ctrl> <Enter> 结束一行时, 启动这个模式。

支持以下的快捷键:

- 所有的标识符, 直到一行的最后标识符将变成声明变量标识符。
- 由该行的最后标识符来确定声明的类型。

在这个意义上, 以下快捷键将表示为:

B 或 BOOL	结果为 BOOL
I 或 INT	结果为 INT
R 或 REAL	结果为 REAL
S 或 STRING	结果为 STRING

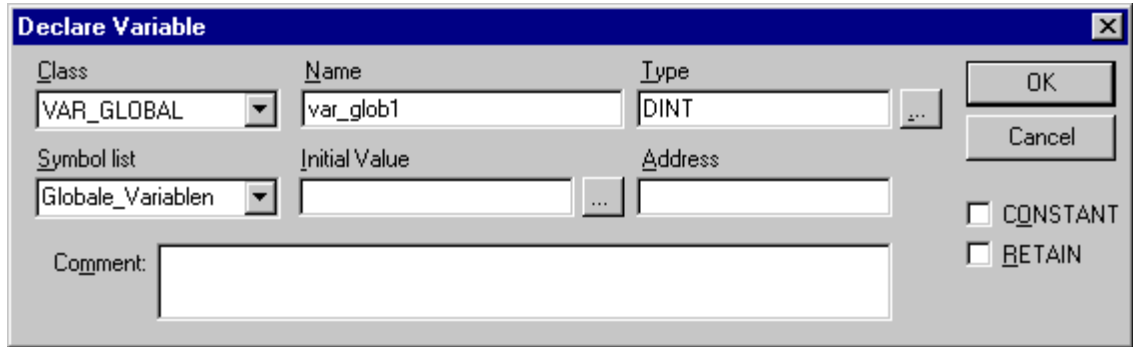
如果经过这些规则后仍未建立类型, 则类型是 **BOOL**, 而且最后的标识符将不再用作类型 (例 1)。取决于声明类型, 每个常数将转变成初始化或字符串 (例 2 和 3)。地址 (如在 %MD12) 则围绕 AT... 属性扩展 (例 4)。分号 (;) 后的文本将变为注释 (例 4)。行中的所有其它字符将被忽略 (例如, 例 5 中的感叹号)。

示例:

简短说明	声明
A	A:BOOL;
A B I 2	A, B:INT := 2;
ST1 S 2; A String	ST1: STRING(2); (* A String *)
X %MD12 R 5; Real Number	X AT %MD12: REAL := 5.0; (* Real Number *)
B !	B:BOOL;

“Autodeclaration” (自动声明)

如果采用选项对话框中的 “Autodeclaration” (自动声明), 则当输入一个尚未声明的变量后, 在所有的编辑器中将出现一个对话框。借助于这个对话框, 可以立即声明这个变量。

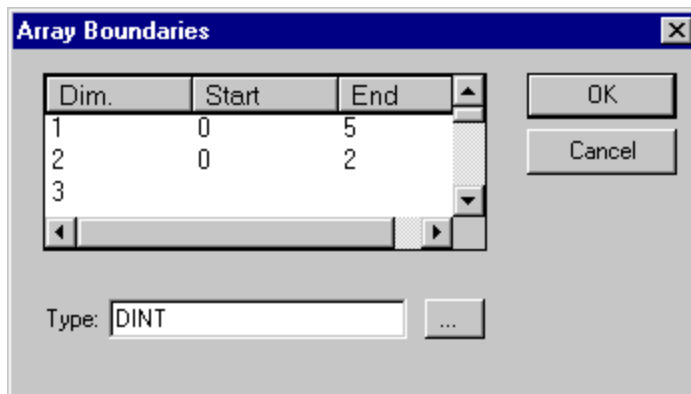


声明变量对话框

借助于类 (**Class**) 组合框, 你或是选择处理一个本地变量 (VAR), 输入变量 (VAR_INPUT), 输出变量 (VAR_OUTPUT), 输入/输出变量 (VAR_IN_OUT), 或是选择处理一个全局变量 (VAR_GLOBAL)。使用 “CONSTANT” 和 “RETAIN” 选项, 你可以选择定义一个常数, 或是一个保持型变量。

你输入到编辑器中的变量名已被输入到 “Name” (名称) 字段中, BOOL 已放置在 “Type” (类型) 字段中。

按钮... 将打开 “Input Assistant” (输入助手) 对话框, 它使你能从所有可能的数据类型中进行选择。如果选择 “ARRAY” (数组) 作为变量类型, 将出现输入数组边界的对话框。



数组声明编辑器

对于可能的三个维数 (**Dim**) 的每一个而言, 可以通过鼠标点击相应的字段, 打开一个编辑空间, 在 “Start” (开始) 和 “End” (结束) 下输入数组边界。数组的数据类型输入到 “Type” (类型) 字段中。为此, 按钮... 用于打开一个输入助手对话框。

当通过按压 OK 按钮离开数组边界对话框时, 在对话框中 “Type” (类型) 字段的输入项是建立在以 IEC 格式为基础的变量声明。例如: ARRAY [1..5, 1..3] OF INT

在 “Initial value” (初始值) 字段, 你可以输入正在声明的变量的初始值。如果这是一个数组或是一个有效结构, 你可以通过这个按钮或 <F2> 打开专用的初始化对话框, 或打开用于其它变量类型的输入助手对话框。

在用于数组的初始化对话框中, 为你提供一个数组成员表; 在鼠标点击 “:=” 空间后, 打开一个输入元素的初始值编辑字段。

在初始化对话框中, 结构和各个成员均在一个树形结构中显示。类型和默认初始值出现在变量名称后的括号内; 每个变量名称后都有 “:=”。在 “:=” 后的字段上以鼠标点击则打开一个编辑字段, 可以输入你所需要的初始值。如果成员是一个数组, 则可以通过鼠标点击数组名称前的加号, 从展开显示的数组中编辑这些字段的初始值。

当使用 OK 按钮结束初始化对话框后，数组或结构的初始化以 IEC 格式出现在声明对话框的初始值字段内。例如： `x:=5,field:=2,3,struct2:=(a:=2,b:=3)`

在地址字段，你可以将正在声明的变量绑定到一个 IEC 地址（AT 声明）。

如果可以应用的话，输入一个注释。通过组合按钮 `<Ctrl>+<Enter>`，使一行分断并使其成为格式化的注释。通过按 OK 按钮，关闭声明对话框，变量被输入到按 IEC 语法的相应声明编辑器内。

注：

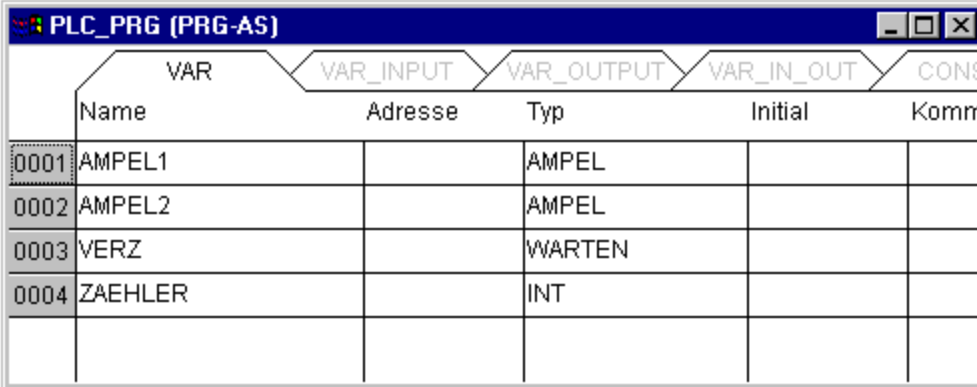
你还可以通过命令“Edit”（编辑）→“Declare Variable”（声明变量）（参见“编辑功能”）打开用于变量声明的对话框。“Online”（联机）模式时，如果光标停留在一个变量上，则可以通过 `<Shift> <F2>` 按钮打开“Autodeclare”（自动声明）窗口显示与当前变量有关的设置值。

声明编辑器中的行数

在脱机模式，单击一个特定的行号将标记整个文本行。在联机模式，单击一个特定的行号，将打开或关闭该行中的变量（如果包含有结构变量的话）。

名称	输入变量标识符
类型	输入变量类型。当实例化一个功能块时，输入功能块
初始化	输入可能的变量初始化值（相应于“:=”赋值操作符）
注释	输入一个注释

可以改变声明编辑器中的二种显示类型，而不会引起任何问题。在联机模式，这二种显示没有什么差别。



	VAR	VAR_INPUT	VAR_OUTPUT	VAR_IN_OUT	CONST
	Name	Adresse	Typ	Initial	Komm
0001	AMPEL1		AMPEL		
0002	AMPEL2		AMPEL		
0003	VERZ		WARTEN		
0004	ZAEHLER		INT		

作为表格显示的声明编辑器

“Insert”（插入）→“New Declaration”（新的声明）

使用该命令，你将一个新的变量带入到声明编辑器的声明表内。如果目前的光标位置位于该表的一个字段内，则新变量将粘贴在前面的一行内；否则新变量粘贴在表的最后。此外，你可以通过使用右箭头按钮或该表最后字段中的制表按钮，将新的声明粘贴在表的最后。你将收到位于“Name”（名称）字段中有名称的一个变量和在“类型”字段中的“BOOL”（默认设置）。你应更改这些值，使其成为所需要的值。名称和类型对变量的完整声明都是必须的。

伪指令

伪指令用来影响有关编译过程的变量属性。它可以与声明编辑器的一个程序行或其自身行的辅助文本一起使用。

伪指令包括在 `{ }` 括号内，忽略大写和小写的区别：

```
{<Instruction text>}
```

如果编译程序不能有意义地翻译指令文本，则整个伪指令将作为注释处理并读出。同时发出一个警告：“忽略编译程序的指示，<指令文本>！”

取决于伪指令的类型和内容，伪指令或是在它所在的行上起作用，或是在随后的所有行上起作用，直到被一个合适的伪指令结束时为止，或是以不同的参数执行相同的伪指令，或到达文件的结束处。对于文件，在这里我们是指：声明部分，实现部分，全局变量表，类型声明。

开括号可以直接后跟一个变量名。开括号和闭括号必须处于相同的行。

当前可以使用以下的伪指令：

```
{flag}: {flag [<flags>] [off|on]}
```

这个伪指令允许影响一个变量属性的声明。

<flags>可能是以下标记的组合：

noinit	变量没有初始化
nowatch	变量不能监视
noread	变量输出到符号文件，但没有读许可。
nowrite	变量输出到符号文件，但没有写许可
noread, nowrite	变量没有输出到符号文件

使用“on”修饰符，伪指令在所有随后的变量声明上起作用，直到以伪指令{flag off}结束时为止，或直到被另一个{flag <flags> on}伪指令结束时为止。没有“on”或“off”修饰符，伪指令只在当前的变量声明（这是由下一个分号关闭的声明）上有效。

示例：

变量 a 未初始化并且不能被监视。变量 b 未初始化：

```
VAR
  a : INT {flag noinit, nowatch};
  b : INT {flag noinit};
END_VAR
```

```
VAR
  {flag noinit, nowatch on}
  a : INT {flag noinit on};
  b : INT;
  {flag off}
END_VAR
```

变量都未初始化

```
{flag noinit on}
VAR
  a : INT;
  b : INT;
END_VAR
{flag off}
```

```

VAR
    {flag noinit on}
    a : INT;
    b : INT;
    {flag off}
END_VAR

```

标记 “**noread**” 和 “**nowrite**” 用于一个 POU，该 POU 读和/或写许可提供有约束的存取权限的选择变量。变量的默认值与用于 POU（变量在其中声明）的设置值相同。如果一个变量没有读许可也没有写许可，则不能将它输出到符号文件。

示例：

如果 POU 有读和写许可，则使用以下的伪指令：变量 **a** 只能以写许可输出，而变量 **b** 完全不能输出。

```

VAR
    a : INT {flag noread};
    b : INT {flag noread, nowrite};
END_VAR

```

变量 **a** 和 **b** 都不能输出到符号文件：

```

{flag noread, nowrite on}
VAR
    a : INT;
    b : INT;
END_VAR
{flag off}

```

伪指令附加在所有随后的变量声明上。

示例：

使用中的所有 POU 都将用读和写许可输出。

```

a : afb;
...
FUNCTION_BLOCK afB
VAR
    b : bfb {flag nowrite};
    c : INT;
END_VAR
...
FUNCTION_BLOCK bfB
VAR
    d : INT {flag noread};
    e : INT {flag nowrite};
END_VAR

```

“a.b.d”：无输出

“a.b.e”：仅有读许可输出

“a.c”：有读和写许可输出。

联机模式时的声明编辑器

在联机模式，声明编辑器变为监视窗口。在每行中，每个变量后都跟有等号(=)和变量值。如果在该点的变量没有进行定义，则会出现三个问号(???)。对于功能块而言，只显示打开实例的变量值(命令：“Project”(项目)→“Open instance”(打开实例))。

在每个多成员变量前有一个加号。通过按 <Enter>按钮或双击这个变量之后可以打开变量。本示例是打开交通信号结构。

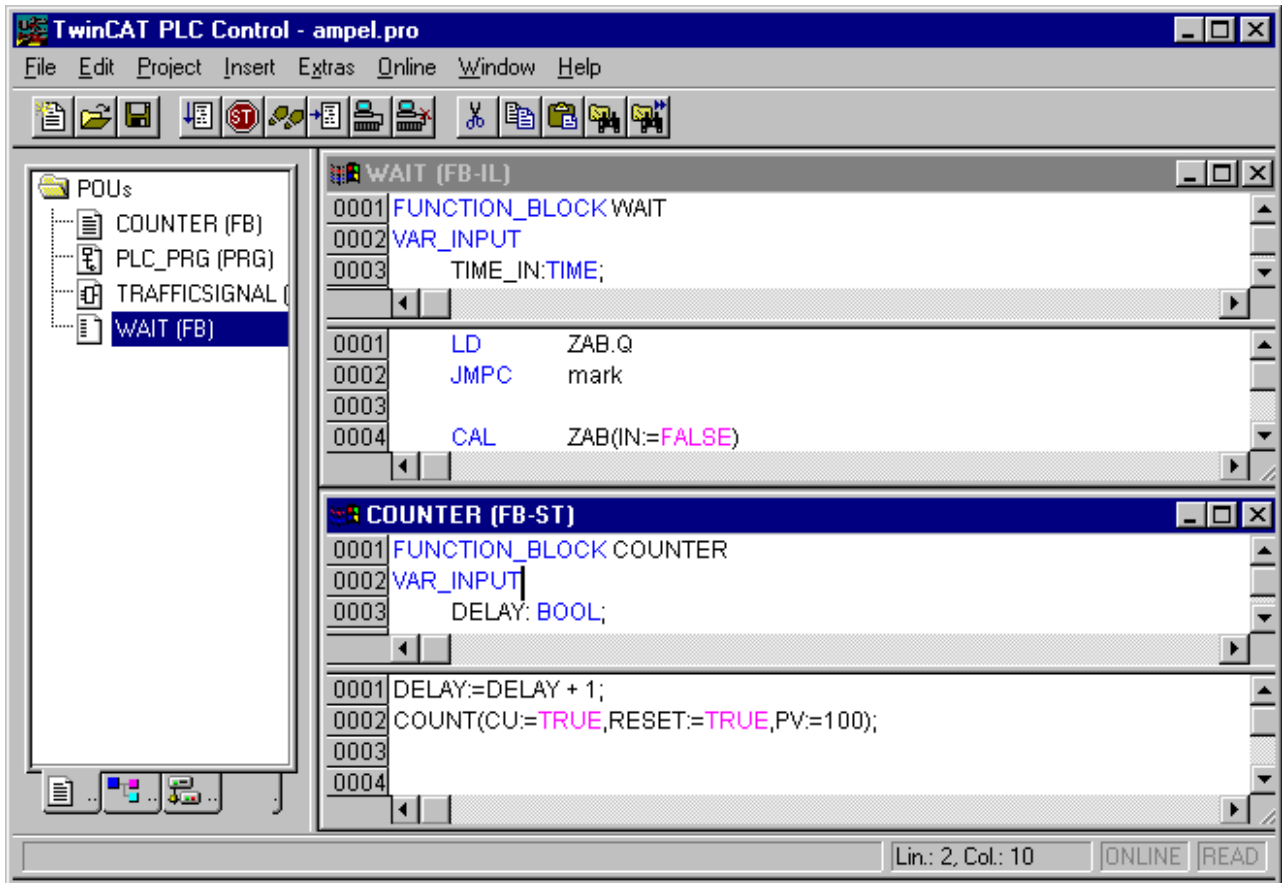
```
[-] AMPEL1
  .....STATUS = 3
  .....GRUEN = FALSE
  .....GELB = FALSE
  .....ROT = TRUE
  .....AUS = FALSE
```

当打开一个变量时，在其后将列出它的所有成员，并在变量前面出现一个减号。如果你再次双击或按<Enter>按钮，则关闭该变量，并重新出现加号。

当在一个单成员变量上按 <Enter> 键或双击它时，打开对话框，以便写入变量值。在这里，可以更改变量的当前值。如果是布尔变量，则不出现对话框；而是切换这些变量。这些新值将变为红色，并保持不变。如果使用“Online”(联机)→“Write values”(写入值)命令，则所有变量都置于选择表中，而且再一次转变成黑色显示。如果使用“Online”(联机)→“Force values”(强制值)命令，则所有变量都将设置为选择值，直到使用“Release force”(解除强制)命令时为止。在这种情况下，强制值的颜色变为红色。

4.3 文本编辑器

TwinCAT PLC Control 的文本编辑器（指令表编辑器和结构化文本编辑器）提供 Windows 文本编辑器的常规能力。文本编辑器的实施由语法着色支持。



指令表和结构化文本编辑器。

最重要的命令可以在上下文菜单中找到（鼠标右键或 <Ctrl>+<F10> 按钮）。文本编辑器使用以下特殊方式的菜单命令：

“Insert”（插入）→ “Operator”（操作符）

使用该命令，将在对话框中显示当前语言能提供的操作符。如果选择一个操作符并以 OK 关闭指令表，则加亮显示的操作符将插入到当前的光标位置。

“Insert”（插入）→ “Operand”（操作数）

使用该命令，将在对话框中显示所有变量。你可以选择需要显示一个全局变量表，还是显示本地变量表，或系统变量表。如果从中选择出一个操作数，并用 OK 按钮关闭对话框，则加亮显示的操作数会插入到当前的光标位置。

“Insert”（插入）→ “Function”（功能）

使用该命令，将在对话框中显示所有功能。你可选择，需要有一个显示用户定义的功能表，还是有一个标准功能表。如果从中选择出一种功能，并用 OK 按钮关闭对话框，则加亮显示的功能就会插入到当前的光标位置。如果在对话框中选择“With arguments”（使用自变量）选项，则插入必要的输入和输出变量。

“Insert”（插入）→ “Functionblock”（功能块）

使用该命令，在对话框中显示所有的功能块。你可以选择需要显示用户定义的功能块表，还是标准功能块表。如果从中选择一个功能块，并用 **OK** 按钮关闭对话框，则加亮显示的功能块就会插入到当前的光标位置。如果在对话框中选择“With arguments”（使用自变量）选项，则插入功能块必要的输入和输出变量。

调用带有输出参数的 POU

一个被调用 POU 的输出参数可直接在调用时由文本语言 IL 和 ST 赋值。例：afbinst 的输出参数 out1 被赋值为变量 a

```
IL: CAL afbinst(in1:=1, out1=>a)
```

```
ST: afbinst(in1:=1, out1=>a);
```

“Online”（联机）模式时的文本编辑器

编辑器的联机功能可以设置断点和单步处理（各个步）。与监视功能一起，用户从而有当今 Windows 标准语言调试程序的调试能力。

在“Online”（联机）模式，文本编辑器窗口被垂直地分为二半。在窗口的左侧，你可以找到正常的程序文本；在窗口的右侧，你可以看到显示的变量，这些值均在相应的行中变化。

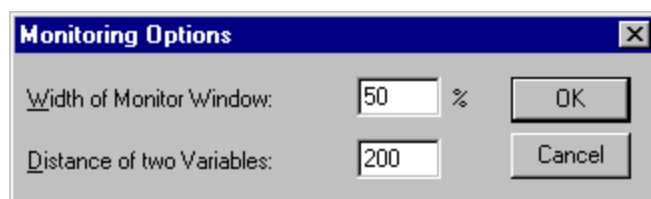
这类显示如同在声明部分中所显示的一样。即，当 PLC 正在运行时，将分别显示相关变量的当前值。

当监视表达式或位编址的变量时，应注意以下几点：在表达式场合，总是显示整个表达式的值。例：a AND b 以蓝色显示，若 a 和 b 都是 TRUE，则以“:=TRUE”显示。对于位编址的变量而言，总是对编址的位值进行监视（例如，a.3 以蓝色显示，或“:=TRUE”，若 a 的值为 4）

如果你将鼠标指针停留在一个变量上一段时间，则在“Tooltip”（工具条提示）内会显示有关该变量的名字和注释。

“Extras”（附加）→ “Monitoring Options”（监视选项）

使用该命令，你可以配置监视窗口。在文本编辑器中，当监视时窗口被分为二半。程序放在左半部分。右半部分则是监视的相应程序行中的所有变量。你可以指定“Monitor Window Width”（监视窗口宽度）以及行中的两个变量之间的“Distance”（距离）。在这种情况下，如果一个距离声明为 1，则相当于所选择字体的一个轮廓的高度。



监视选项对话框

断点位置

在 TwinCAT PLC Control 中，IL 中断点位置的几行代码在内部被组合成一个单行的 C 代码，断点不能在每个行都进行设置。断点位置可以包括在一个程序内的所有变量值可以改变的位置，或者在程序流程分支闭合处。（例外：功能调用。如有必要，必须在此设置功能的断点）。如果在中间位置安放一个断点，则该断点没有意义，这是因为从前面断点位置以来，数据不会有任何改变。可以在 IL 中以下位置设置断点：

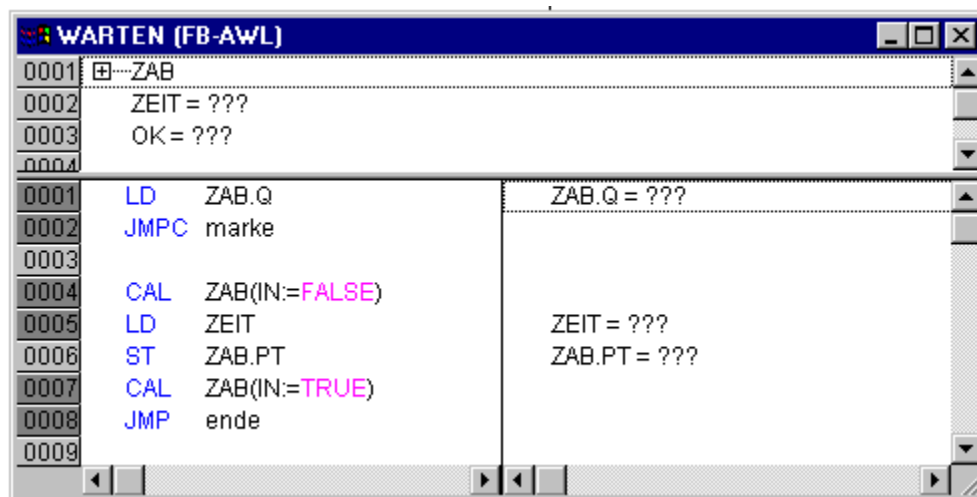
- 在 POU 的开始处

- 在每个 LD、LDN（或者，在 LD 位于一个标号处时，就在标号处）
- 在每个 JMP、JMPC、JMPCN
- 在每个标号处
- 在每个 CAL、CALC、CALCN
- 在每个 RET、RETC、RETCN
- 在 POU 结束处。

结构化文本允许以下位置设置断点：

- 在每个赋值处
- 在每个 RETURN 和 EXIT 指令处
- 正在评估的条件处（WHILE, IF, REPEAT）
- 在 POU 结束处。

断点位置以深灰色的行号字段显示标志。



IL 编辑器中可能的断点位置（较暗的行号字段）

如何设置一个断点？

为了设置一个断点，在你需要设置断点的行中点击行号字段。如果所选择的字段恰好是一个断点位置，则该行号字段的颜色就会从深灰色变成浅蓝色，并且将在 PLC 中有效。

删除断点

相应地，要删除一个断点，可以在需要删除断点的行号字段上点击。设置和删除断点还可以通过菜单（“Online”（联机）→“Toggle Breakpoint”（切换断点））完成，也可以使用功能键 <F9>，或使用工具条中的符号完成。

在断点处发生了什么？

如果 PLC 程序到达一个断点，则在屏幕上显示相应行的断点。并且 PLC 确定的行号字段将显示为红色。PLC 中的用户程序被停止。如果该程序处于断点位置，可以通过使用“Online”（联机）→“Run”（运行）释放该过程。

此外，使用命令“**Online**”（联机）→“**Step over**”（单步跳出）或“**Step in**”（单步进入），你可以使程序运行到下一个断点位置。如果你定位的指令是一个 **CAL** 命令，或者，下一个断点位置前是一个功能调用，则可以使用“**Step over**”（单步跳出）来旁路这个功能调用。使用“**Step in**”（单步进入）命令，你可以进入到打开的 POU 分支程序。

文本编辑器的行号

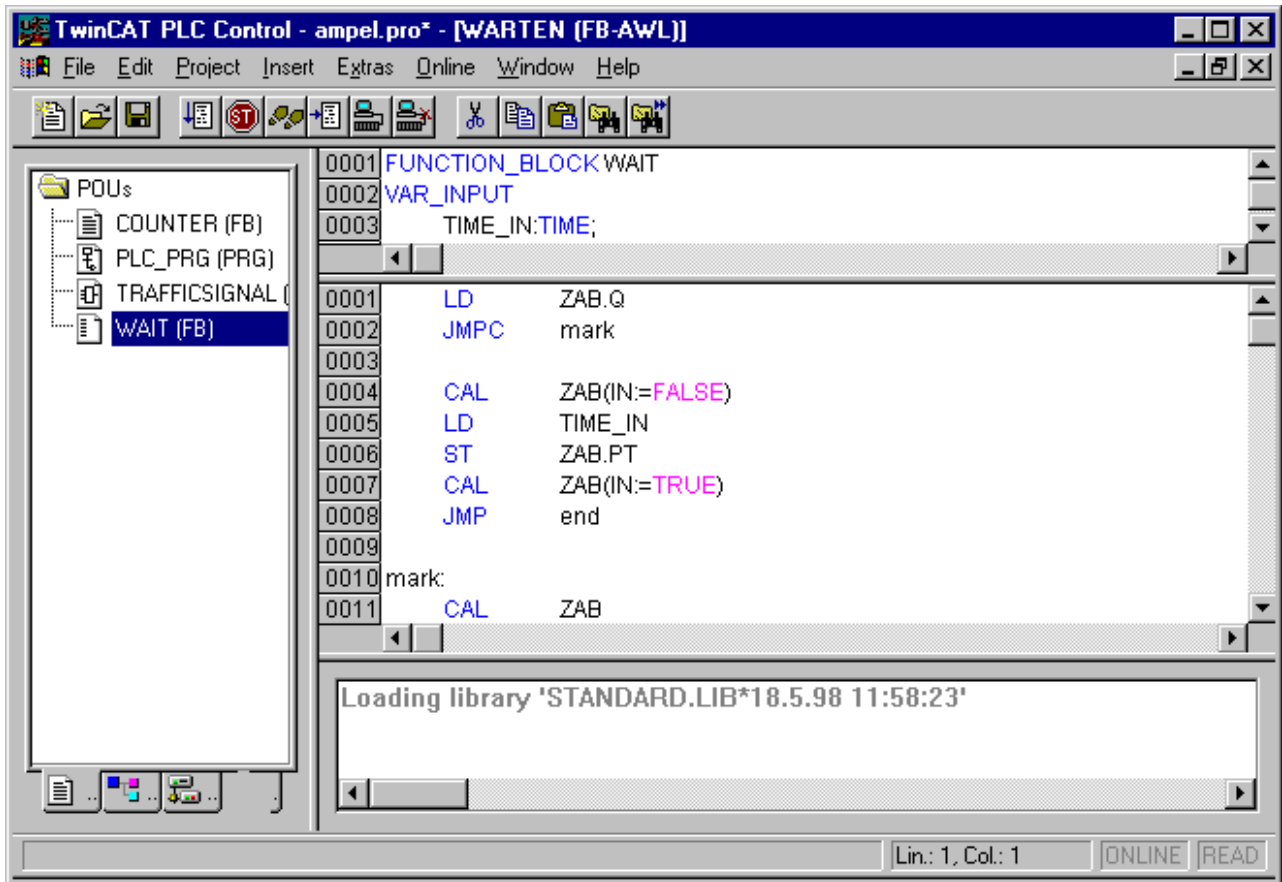
文本编辑器的行号给出 POU 程序实体的每个文本行的编号。在“**Off-line**”（脱机）模式，在一个特定行号上单击，将标记整个文本行。在“**Online**”（联机）模式，行号的背景颜色指示每行的断点状态：

- 深灰色：该行是断点的一个可能位置。
- 浅蓝色：已经在该行中设置了一个断点
- 红色：程序已经到达该断点

在“**Online**”（联机）模式，简单地单击鼠标将改变该断点行的状态。

4.4 指令表编辑器

这是以 IL 语言编写的一个 POU 如何在相应的 TwinCAT PLC Control 编辑器出现的示例：



IL 编辑器

用于 POU 的所有编辑器都由一个声明部分和一个本体组成。这两部分使用一个屏幕分隔器隔开。

指令表编辑器是一个文本编辑器，具备 Windows 文本编辑器的通用功能。最重要的命令可以在上下文菜单中找到（鼠标右键或<Ctrl>+<F10>）。

POU 的多行调用也是可能的：

示例：

```
CAL CTU_inst(
CU:=%IX10,
PV:=(
LD A
ADD 5
)
)
```

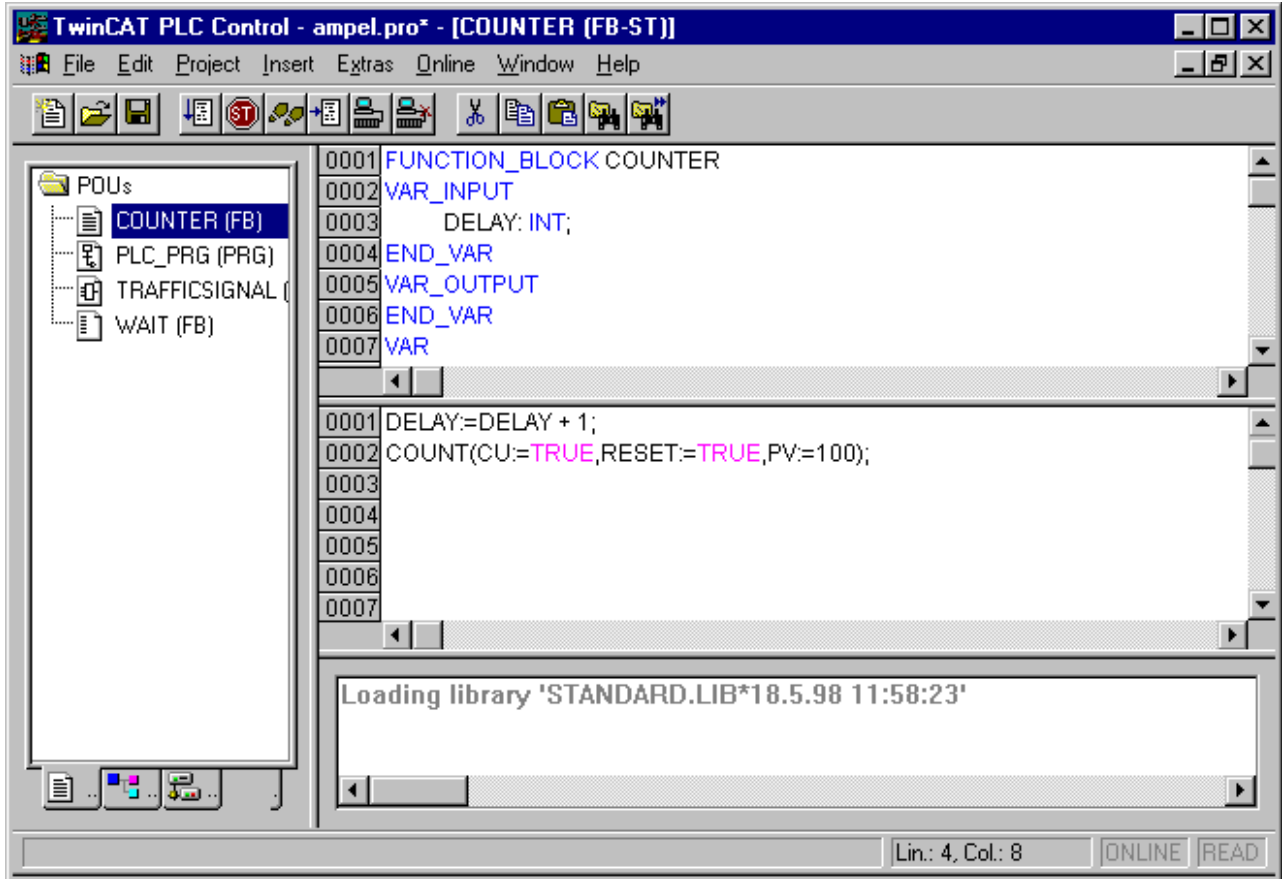
流控制

使用“**Online**”（联机）→“**Flow control**”（流控制）命令，一个显示累加器内容的附加字段被插入到 IL 编辑器左边的每行中。

有关“**Online**”（联机）模式的 IL 编辑器的更多情况可参阅“**Text Editors in Online Mode**”（联机方式的文本编辑器）。

4.5 结构化文本编辑器

这是以 ST 语言编写的一个 POU 如何在相应的 TwinCAT PLC Control 编辑器下出现的示例：



结构化文本编辑器

用于 POU 的所有编辑器都由一个声明部分和一个本体组成。这两部分以一个屏幕分隔器隔开。

结构化文本编辑器是一种文本编辑器，具备 Windows 文本编辑器的常用功能。最重要的命令可以在上下文菜单中找到（鼠标右键）。

有关 ST 语言方面的更多情况，请参阅“结构化文本（ST）”章节内容

4.6 图形编辑器

面向图形语言的编辑器，顺序功能图 SFC，LD 和 FBD，以及自由图形功能块图 CFC 的编辑器有许多共同之处。这些共同之处在下面的章节中汇总。图形编辑器也支持语法着色功能。

缩放

使用 SFC、LD、FBD 和 CFC 语言的对象，如 POU、动作、转换等，都可通过缩放功能进行放大或缩小。实现部分窗口内容的所有图形元素都受影响；声明部分则保持不变。

在标准形状时，每个对象都是以缩放率 100%显示的。设置的缩放率作为对象属性保存在项目内。

项目文档的打印总是以 100%显示率出现！

缩放率通过工具条中的一个选择表设置。可选择 25%至 400%之间的值；也可以手动输入 10%至 500%之间的值。

仅当光标停留在以图形语言建立的一个对象上，或停留在一个可视化对象上时，才可以选择一个缩放率。

即使是经过缩放的对象，在编辑器中仍然可以通过箭头键（方向键）进一步选择和到达光标位置。文本的大小则受缩放率和设置的字体大小限制。

所有编辑器菜单的执行特性（例如插入一个框）以及光标位置功能，在所有的缩放率下都可以使用（考虑相同的缩放率）。

在“Online”（联机）模式，每个对象按照已经设置的缩放率进行显示；“Online”（联机）功能的使用不受限制。

当使用“IntelliMouse”（智能鼠标）时，通过按<CTRL>键可放大/缩小对象，同时可以使用鼠标滚动轮向前或向后移动对象。

网络

在 LD 和 FBD 编辑器中，程序布置在一个网络表内。每个网络的左边以一个网络序号命名，并有一个由逻辑或算术表达式、程序、功能或功能块调用和跳转或返回指令组成的结构。

标号

每个网络都有一个作为选项的标号，其缺省值为空。通过点击网络的第一行（紧跟在网络号之后）可编辑这个标号。现在，你可以输入一个标号，后跟一个冒号。

网络注释

每个网络都可以拥有多行注释。在“Extras”（附加）→“Options”（选项）中，你可以输入用于网络注释的最大行数。这个输入项确定了最大注释数组的大小。（默认值为 4）你还可输入通常要为注释保留的行数（最小注释大小）。如果（例如）输入值为 2，则在每个网络开始处，在标号行后面将出现两个空行。默认值为 0，这样做的优点是允许有更多的网络放置于屏幕区内。

如果最小注释行数值大于 0，为了输入一个注释，你只要简单地点击注释行，然后输入注释即可。否则，你必须接着选择想要输入注释的网络，并使用“Insert”（插入）→“Comment”（注释）命令，插入注释行。和程序文本相比，注释使用灰色显示。

在“**Ladder editor**”（梯形图编辑器）中，你也可以将单独的注释分配给每个接点或线圈。为此，激活选项“**Comments per Contact**”（每个接点注释），并插入用于变量注释的编辑字段行，保存和显示注释的行数。如果完成了这些设置，则将在每个接点和线圈上面显示一个你可以插入文本的注释字段。如果选中选项“**Comments per Contact**”（每个接点注释），则在梯形图编辑器中还可以定义用于线圈相应接点变量名的行数“**Lines for variable text**”（变量文本行：）。它可以将长变量名通过拆分成几个行而进行完整的显示。在“**Ladder**”（梯形图）编辑器中，一旦网络长度超过给出的窗口尺寸，有些图形元素或许会看不见，可以强制拆开网络中的行。为此，可以激活选项“**Networks with Linebreaks**”（拆分网络）。

“Insert”（插入）→“Network (after)”（网络（后向））或“Insert”（插入）→“Network (before)”（网络（前向））快捷键：〈Shift〉+〈T〉（后向插入网络）

为了将一个新网络插入到 FBD 或 LD 编辑器内，选择“Insert”（插入）→“Network (after)”（网络（后向））或“Insert”（插入）→“Network (before)”（网络（前向））命令，这取决于你需要将新网络插入在当前网络之前或之后。当前网络可通过单击网络号加以改变。你可以在网络号的虚线矩形内识别它。通过使用〈Shift〉键和单击鼠标，你可以选择整个网络区域，即从当前网络到单击选择的网络。

联机模式的网络编辑器

在 FBD 和 LD 编辑器中，你只能在网络位置设置断点。已设置断点的网络其网络号字段将显示蓝色。过程将在该网络前的断点处停止。此时，网络号字段将显示红色。使用单步处理（步），你可以从网络跳转到网络。

在进入和退出 POU（程序组织单元）的网络时，将监视所有的值。

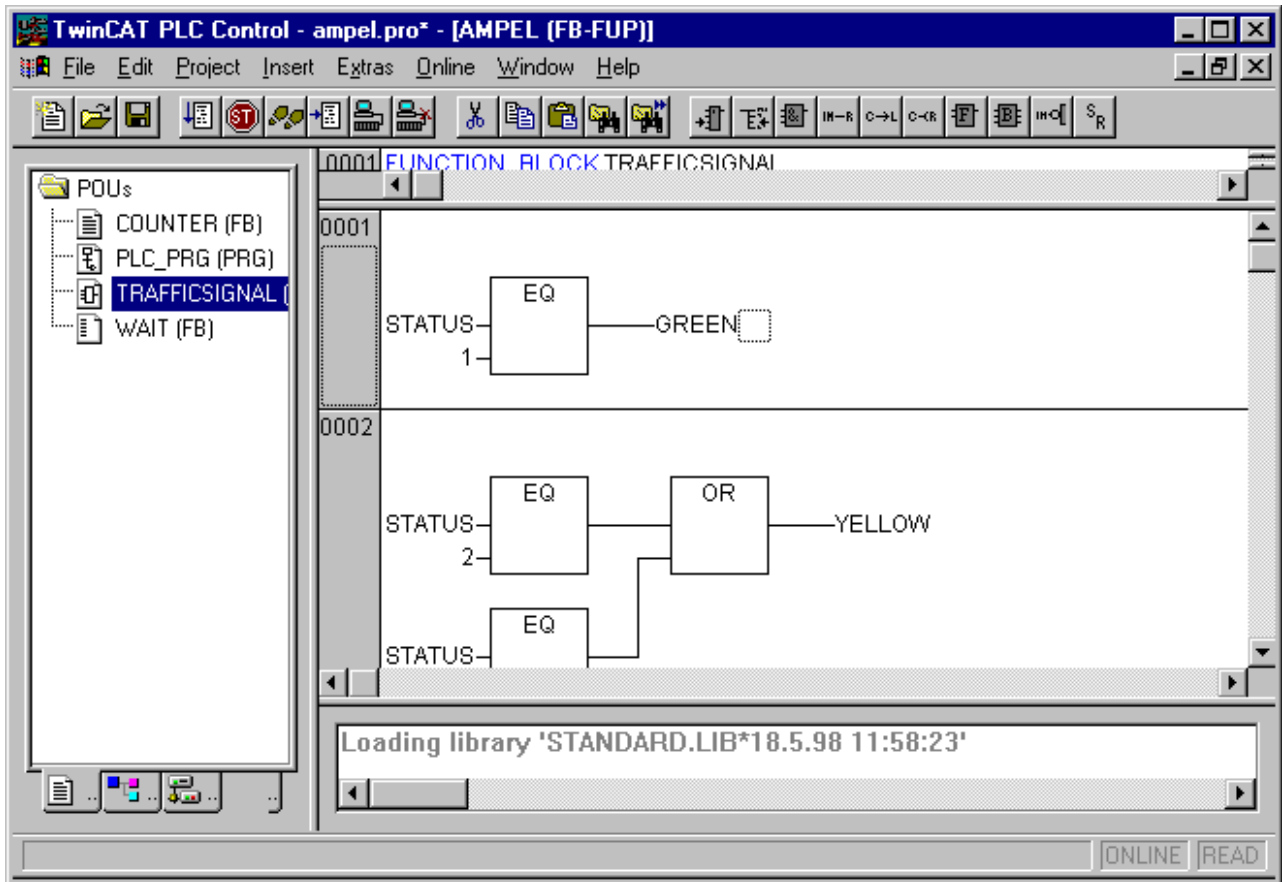
当监视表达式或位编址的变量时，应注意以下几点：在表达式中，例如 **a AND b**，作为转换条件或功能块输入使用，总是显示整个表达式的值（如果 **a** 和 **b** 都是 **TRUE**，则表达式:=**TRUE**，**a AND b** 显示成蓝色）。对于位编址的变量而言，总是对编址的位值进行监视（例如，如果 **a** 的值为 **4**，或表达式:= **TRUE**，则 **a.3** 以蓝色显示）

流控制是通过使用“**Online**”（联机）→“**Flow control**”（流控制）命令运行的。通过流控制，你可以观察网络中经过连接线正在传送的当前值。如果连接线并不携带布尔值，则在专用的插入字段中显示该值。监视字段未使用的变量(例如在功能 **SEL** 中)以灰色阴影显示。如果连接线携带布尔值，则当它携带 **TRUE** 时，将以阴影的蓝色显示。因此，当 **PLC** 正在运行时，你可以获取相关的流信息。

如果你将鼠标指针停留在一个变量上一段时间，则在“工具提示条”内显示有关该变量的名称和注释。

4.7 功能块图编辑器

这是以 FBD 语言编写的一个 POU，在相应的 TwinCAT PLC Control 编辑器下是如何表现的示例：



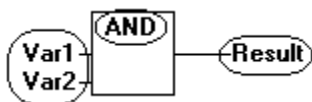
功能块图编辑器

功能块图编辑器是一种图形编辑器。它与网络表一起工作，在网络表中，每个网络结构包含一个相应显示的逻辑或算术表达式，调用功能块，功能，程序，跳转，或返回指令。最重要的一些命令可以在上下文菜单（鼠标右键）中找到。

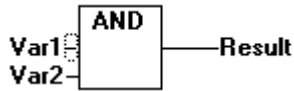
FBD 中的光标位置

每个文本都是可能有光标的位置。已选择的文本处于蓝色背景下，并且可以进行更改。你也可以通过一个虚线矩形来识别出当前的光标位置。以下示例说明了所有可能的光标位置：

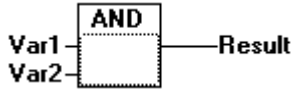
- 1) 每个文本字段（可能的光标位置在黑色框内）：



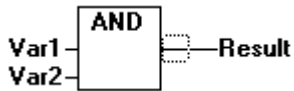
- 2) 每个输入：



3) 每个操作符、功能或功能块:



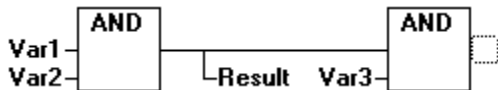
4) 输出, 如果后面有赋值或跳转:



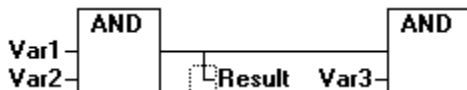
5) 在赋值, 跳转, 或返回指令上的交汇处:



6) 在每个网络右侧最外面对象的后面 (最后光标位置, 与选择网络相同的光标位置):



7) 直接在赋值前面的交叉线处:



如何设置光标

通过点击鼠标, 或借助于键盘, 可将光标设置到某个位置。使用方向键, 可在任何时候, 以所选择的方向跳转到最近的光标位置。包括文本字段在内的所有光标位置都可以使用这种方式到达。如果选择的是最后的光标位置, 则可以使用〈up〉或〈down〉箭头键选择以前或随后网络的最后光标位置。一个空网络只包含三个问号“???”。通过点击这三个问号的后部来选择最后光标位置。

“Insert” (插入) → “Assignment” (赋值) 快捷键: 〈Ctrl〉+ 〈A〉

该命令可插入一个赋值。取决于所选择的位置，可以有几种插入方式：直接插入在所选择的输入前面（光标位置 2），直接插入在所选择的输出后面（光标位置 4），直接插入在所选择的交叉线前面（光标位置 5），或在网络的结束处（光标位置 6）。对插入的赋值而言，选择时会伴随输入文本“???”，而赋值可以被需要赋值的变量替换。为此，你也可以使用“Input Assistant”（输入助手）。为了将一个附加的赋值插入到一个现有的赋值中，可以使用命令“Insert”（插入）→“Output”（输出）。

“Insert”（插入）→“Jump”（跳转）快捷键：〈Ctrl〉+〈L〉

该命令可插入一个跳转。取决于所选择的位置，可以有几种插入方式：直接插入在所选择的输入前面（光标位置 2），直接插入在所选择的输出后面（光标位置 4），直接插入在所选择的交叉线前面（光标位置 5），或在网络的结束处（光标位置 6）。对于一个插入的跳转而言，选择时会伴随输入文本“???”，而跳转可以替换需要赋值的标号。

“Insert”（插入）→“Return”（返回）快捷键：〈Ctrl〉+〈R〉

该命令插入一个 RETURN（返回）指令。取决于所选择的位置，可以有几种插入方式：直接插入在所选择的输入前面（光标位置 2），直接插入在所选择的输出后面（光标位置 4），直接插入在所选择的交叉线前面（光标位置 5），或在网络的结束处（光标位置 6）。

“Insert”（插入）→“Box”（框）快捷键：〈Ctrl〉+〈B〉

使用该命令，可插入操作符、功能、功能块和程序。首先，总是先插入一个“AND”操作符。通过对类型文本（AND）的选择和改写，可将这个操作符转换成其它操作符、功能、功能块和程序。你也可以使用“Input Assistant”（输入助手）（〈F2〉）选择所需要的 POU。如果新选择的功能块有另外的最小输入数，则将添加这些输入。如果新块的输入数量少于标准块的输入数量，则多余的输入将被删除。

在功能和功能块中，将显示输入和输出的参数名称。

在功能块框的上部有一个可以编辑的实例字段。如果通过改变类型文本调用了未知的其它功能块，则显示两个输入和给出类型的操作框。如果选择实例字段，可以进入变量选择类别的〈F2〉“Input Assistant”（输入助手）。

最新选择的 POU 插入在所选择的位置：

- 如选择一个输入（光标位置 2），则 POU 插入在这个输入之前。POU 的第一个输入链接到选出的输入左边的分支上。新的 POU 输出链接到选出的输入。
- 如选择一个输出（光标位置 4），则 POU 插入在这个输出之后。POU 的第一个输入链接到选出的输出。新的 POU 输出链接到与选择输出所链接的分支。
- 如选择一个 POU，一个功能或一个功能块（光标位置 3），则以前的图形元素将被新的 POU 所替代。
- 分支尽可能以其替代前相同的方式进行链接。如果原有的图形元素的输入要比新的图形元素多，则删除不可连接的分支。对输出可作同样的处理。
- 如选择一个跳转或返回，则 POU 将插入在这个跳转或返回的前面。POU 的第一个输入与选出的图形元素左边的分支相连接。POU 的输出链接到选出图形元素右边的分支。
- 如果选择网络的最后光标位置（光标位置 6），则 POU 将插入在最后的图形元素之后。POU 的第一个输入链接到选出位置左边的分支。

所有不能链接的 POU 输入将接收文本“???”。必须点击这个文本并转变成为所需要的常数或变量。

如在一个插入的 POU 右侧有一个分支，则这个分支将分配给第一个 POU 的输出。否则这个输出将没有赋值。

“Insert”（插入）→ “Input”（输入）快捷键：〈Ctrl〉+〈U〉

该命令插入一个输入操作符。对许多不同的操作符而言，输入的个数也许是有变化的。（例如，ADD 可以有 2 个或更多个输入。）为了通过输入来扩展这样一个操作符，你需要选择这样一种输入，在其前面是你需要插入的一个附加输入（光标位置 1）；如果要插入一个最少输入，或者你必须选择该操作符本身（光标位置 3）。

插入的输入以文本“???”填充。必须点击该文本并改变成所需要的常数或变量。为此，你还可以使用“输入助手”。

“Insert”（插入）→ “Output”（输出）

该命令将在一个现有赋值内插入一个附加赋值。这个功能用于所谓组合赋值的布局；亦即，当前位置的赋值被赋值给几个变量。

如果你选择一个赋值上的交叉线（光标位置 5），或直接在该交叉前的输出（光标位置 4）则该处原有的赋值之后将插入其它赋值。

如果选择直接在一个赋值之前的交叉线（光标位置 4），则在这个赋值之前将插入其它赋值。

插入的输出以文本“???”填充。必须点击这个文本并转变成所需要的变量。为此，你也可使用“输入助手”

“Extras”（附加）→ “Negation”（取反）快捷键：〈Ctrl〉+〈N〉

使用该命令，你可取反输入、输出、跳转或 RETURN(返回)指令。取反符号使用一个连接线加小圆表示。

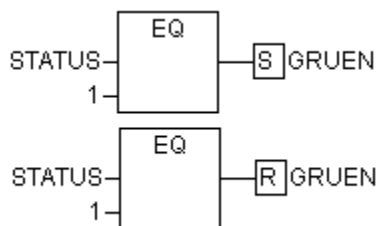
如选择一个输入（光标位置 2），则这个输入将是取反的。

如选择一个输出（光标位置 4），则这个输出将是取反的。

如果标记是一个跳转或一个返回，则这个跳转或返回将是取反的。通过再次取反，可以取消该取反。

“Extras”（附加）→ “Set/Reset”（置位/复位）

使用该命令，你可定义“Set Output”（置位输出）或“Reset Output”（复位输出）。“Set Output”（置位输出）的网络以[S]显示，而“Reset Output”（复位输出）的网络以[R]显示。



FBD 中的置位/复位输出

如果是“Output Set”（输出置位），其网络返回 TRUE，则“Output Set”（输出置位）设置为 TRUE。即使网络跳回到 FALSE，输出仍然保持在这个值。如果是“Output Reset”（输出复位），其网络返回 FALSE，则“Output Reset”（输出复位）设置为 FALSE。输出保持其原有值，即使网络跳回到 FALSE 亦是如此。通过多次执行该命令，输出将在置位，复位和正常输出之间交替变化。

“Extras”（附加）→ “Zoom”（缩放）快捷键：〈Alt〉+〈Enter〉

使用该命令，选中的 POU 将被装入到它的编辑器内（光标位置 3）。如果你正在处理来自库的 POU，则库管理器将被打开，并显示相应的 POU。

“Extras”（附加）→“Open instance”（打开实例）

该命令相应于“Project”（项目）→“Open instance”（打开实例）命令。

FBD 中的剪切，复制，粘贴和删除

在“Edit”（编辑）菜单项下，可以找到“剪切”（Cut），“复制”（Copy），“粘贴”（Paste）和“删除”（Delete）命令。

如选择一个交叉线（光标位置 5），则位于该交叉线下的赋值，跳转或 RETURN（返回）将被剪切，删除或复制。

如选择一个操作符、功能或功能块（光标位置 3），则所选择的对象本身，与输入有关的所有分支（除了第一个分支）一起将被剪切、删除或复制。否则，在光标位置之前的整个分支将被剪切、删除或复制。

经过复制或剪切后，删除或复制部分被置于粘贴板上，现在，如有需要，就可以粘贴。为了这样做，你首先需要选择粘贴点。

有效的粘贴点包括输入和输出。如果一个操作符、功能或功能块已被装载到粘贴板上（作为一种提示：在这种情况下除了第一个分支以外的所有连接分支都汇聚在粘贴板上），第一个输入与粘贴点前面的分支连接。否则，位于粘贴点之前的整个分支都将被粘贴板的内容所替换。在以上各种情况下，被粘贴的最后元素连接到位于粘贴点前面的分支上。

注：

通过剪切和粘贴可以解决以下问题：一个新的操作符粘贴在一个网络的中间位置。在操作符右侧的分支现在与第一个输入相连接，但是必须与第二个输入相连接。现在，你可以选择第一个输入并执行命令“Edit”（编辑）→“Cut”（剪切）。在此之后，你可以选择第二个输入并执行命令“Edit”（编辑）→“Paste”（粘贴）。以这种方法，分支取决于第二个输入。

联机模式的功能块图

在功能块图中，断点只能在网络上设置。如果已经在网络上设置了一个断点，则网络号字段将以蓝色显示。过程将在有断点的网络前停止运行。此时，网络号字段将变为红色。通过步进调试（单步），你可以从网络跳转到网络。

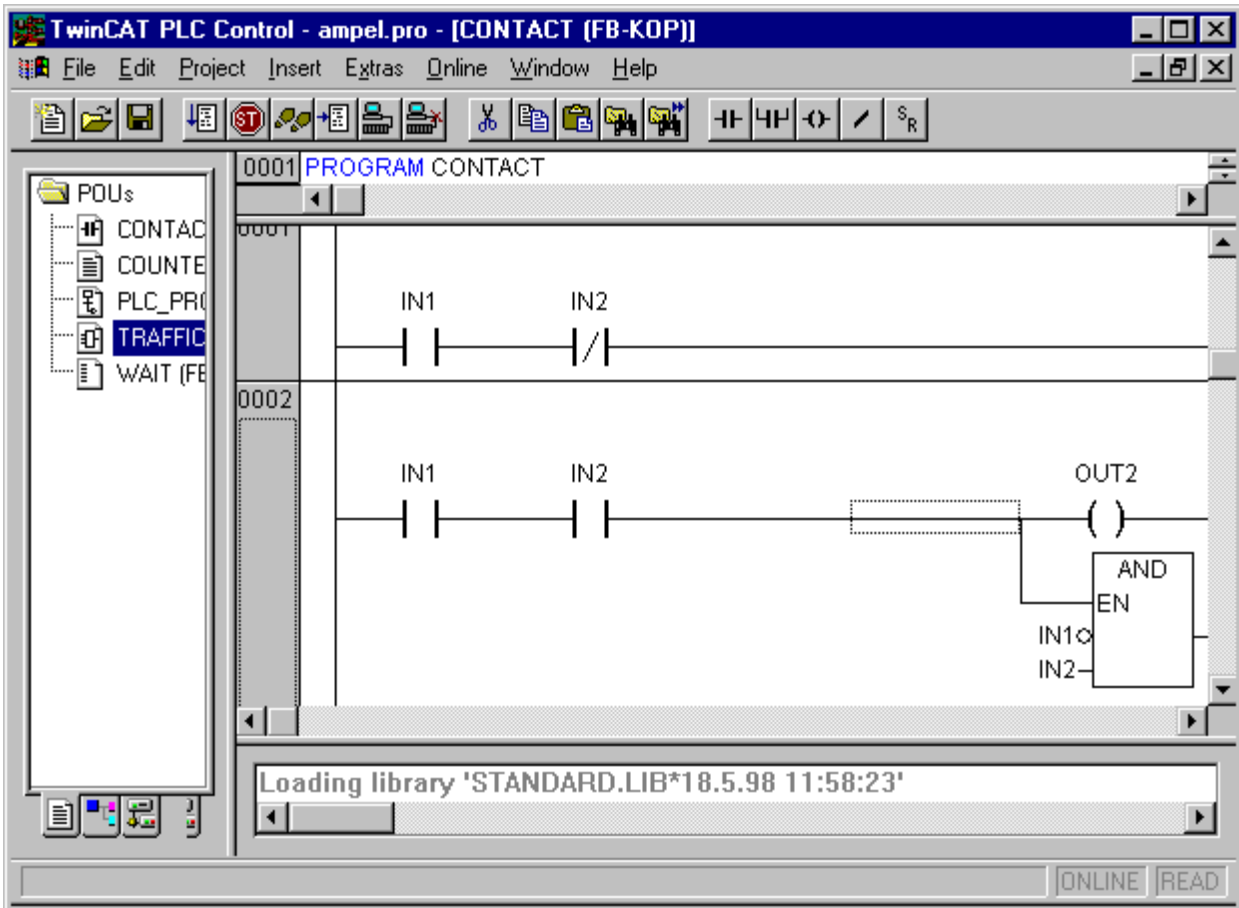
“Exception”（异常）：为每个变量显示当前值

双击一个变量，将打开写入变量的对话框。在这里，可以更改变量的当前值。对于布尔变量，则不出现对话框；因为这些变量值是切换型的。新的值将转变成红色，并保持不变。如果使用“Online”（联机）→“Write values”（写值）命令，则选择表上的所有变量将再次转变成黑色显示。使用“Online”（联机）→“Flow control”（流控制）命令，则可以使流控制运行。通过流控制，你可以观察网络中经过连接线正在传送的当前值。如连接线并不携带布尔值，则使用专用的插入字段显示该值。如连接线携带布尔值，则当它们携带 TRUE 时，这些布尔值以加阴影的蓝色显示。使用这种方法，你可在 PLC 运行时得到流信息。

如果你将鼠标指针停留在一个变量上一段时间，则在“工具提示条”内显示有关该变量的名称和注释。

4.8 梯形图编辑器

说明一个以 LD 语言编写的 POU 在 TwinCAT PLC Control 编辑器内是如何表现的示例：



梯形图中的 POU

用于 POU 的所有编辑器都由一个声明部分和一个本体组成。这两个部分以一个屏幕分隔器隔开。

LD 编辑器是一种图形编辑器。最重要的命令可以在上下文菜单中找到（鼠标右键或<Ctrl>+<F10>）。

有关图元的相关信息，参见“梯形图（LD）”

LD 编辑器中的光标位置

以下位置可以是光标位置，在这些位置，功能块和程序存取可作为接点处理。有 EN 输入的 POU 和连接其它 POU 的情况可以和“Function Block Diagram”（功能块图）中那样进行处理。有关编辑这种网络部分的信息可在“FBD 编辑器”一章中找到。

1. 每个文本字段（可能的光标位置以黑色框表示）



2. 每个接点或功能块。



3. 每一个线圈



4. 接点和线圈之间的连接线



梯形图以特定方式使用以下命令：

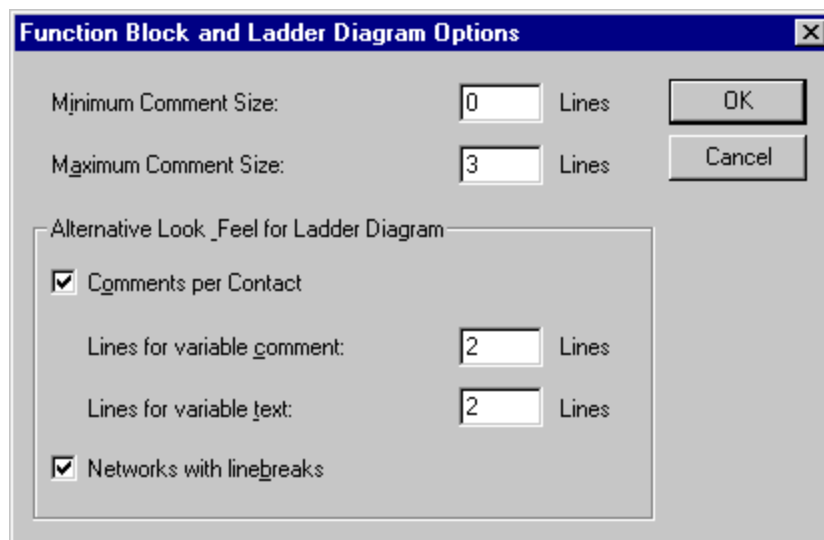
“Insert”（插入）→ “Contact”（接点）快捷键：〈Ctrl〉+ 〈O〉

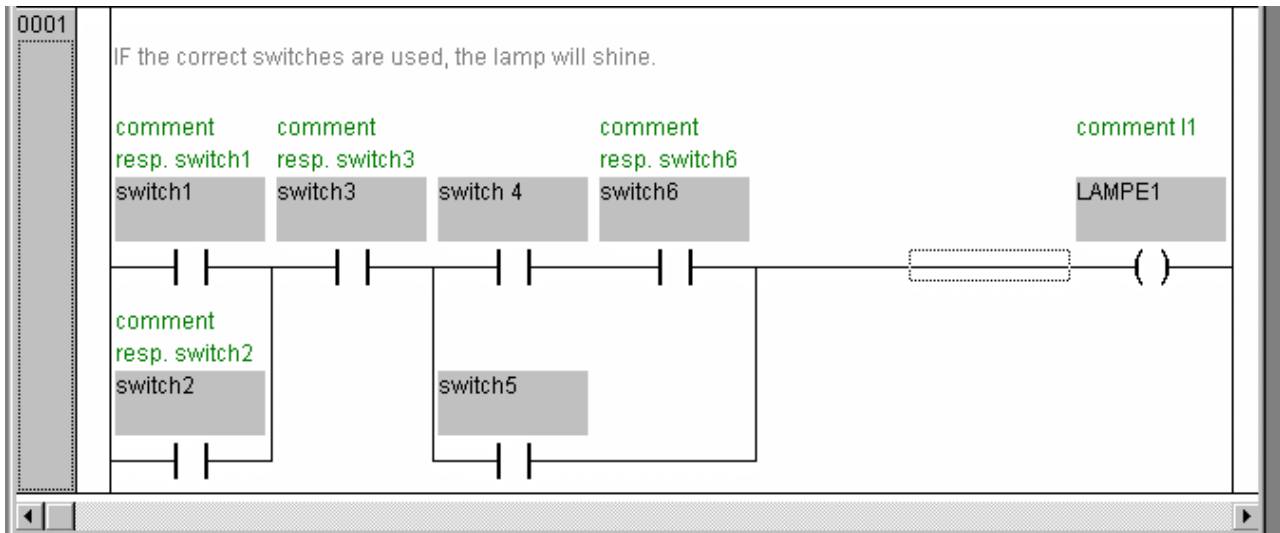
LD 编辑器中使用该命令以便将一个接点插入到网络中标记位置的前面。

如果标记的位置是一个线圈（光标位置 3）或是接点和线圈之间的连接线（光标位置 4），则新的接点将以串联的方式连接到以前的接点连接处。该接点以文本“???”给出。你可以点击该文本，并将它转变为所需要的变量或所需要的常数。为此，你还可以使用“Input Assistant”（输入助手）。

可以在对话框“Function Block and Ladder Diagram Options”（功能块和梯形图选项）→（“Extras”（附加）→ “Options”（选项））中，为变量注释选中选项“options Comments per Contact”（每个接点和行注释），从而为变量名保留一定数量的行。如果使用了长变量名，要使网络保持简短，这样做也许是有用的。

还要注意有拆分“Networks”（网络）的选项，你可以激活对话框“Function Block and Ladder Diagram Options”（功能块和梯形图选项）中的“Extras”（附加）→ “Options”（选项）命令。





“Insert”（插入）→ “Parallel Contact”（并联接点）快捷键：〈Ctrl〉+ 〈R〉

使用 LD 编辑器中的该命令，在网络内的标记位置插入一个并联接点。如果标记位置是一个线圈（光标位置 3）或是接点和线圈之间的连接（光标位置 4），则新的接点将并联地连接到整个以前的接点连接。该接点以文本“???”表示。你可以点击该文本，并将它转变为所需要的变量或所需要的常数。为此，你还可以使用“输入助手”。

“Insert”（插入）→ “Function Block”（功能块）快捷键：〈Ctrl〉+ 〈B〉

使用该命令以便将一个操作符，功能块，功能或程序作为一个 POU 插入。为此，必须标记接点和线圈之间的连接（光标位置 4），或一个线圈（光标位置 3）。新的 POU 起初被指名为 AND。如果你愿意，可以将这个名称指名为另外一个操作符。为此，你还可以使用“输入助手”。标准的和自定义的 POU 均可以使用。

POU 的第一个输入置于输入连接上，第一个输出置于输出连接上；因此这些变量必须定义为类型 BOOL（布尔）。POU 的所有其它输入和输出均用文本“???”填充。这些预置的项可以赋值其它常数，变量或地址。为此，你还可以使用“输入助手”。

“Insert”（插入）→ “Coil”（线圈）快捷键：〈Ctrl〉+ 〈L〉

你可以使用 LD 编辑器中的该命令将一个线圈并联地插入到原有的线圈上。如果标记的位置是接点和线圈之间的连接线（光标位置 4），则新的线圈将作为最后一个元素被插入。如果标记的位置是一个线圈（光标位置 3），则新线圈将直接插入在它的上面。线圈以默认设置的文本“???”填充。你可以点击该文本并将它转变为所需要的变量。为此，你还可以使用“Input Assistant”（输入助手）。

有 EN 输入的 POU

如果你需要将 LD 网络如同 PLC 一样打开其它的 POU，则必须将 EN 输入合并到一个 POU。这样的 POU 与线圈并联连接。除了这样的 POU 之外，还可以开发其它的网络（如在“Function Block Diagram”（功能块图）中所作的那样）。你可以在菜单项“Insert”（插入）→ “Insert at Blocks”（在块处插入）下找到插入 EN POU 的命令。

一个操作符、功能块或带有 EN 输入的功能，和功能块图中相应的 POU 相同（除了它的执行是在 EN 输入控制以外）这个输入附加在线圈和接点之间的连接线上。如果该连接线携带信息“On”（通），则将评估该 POU。

如果一旦已经创建了有 EN 输入的 POU，则该 POU 可以用于创建一个网络。这意味着，来自常用操作符的功能和功能块数据可以流入一个 EN POU，而一个 EN POU 可将数据带给这类用法的 POU。

因此，如果要和 FBD 语言那样，在 LD 编辑器中对一个网络进行程序设计，你只需首先将一个 EN 操作符插入到一个新网络内。然后，从这个 POU 可以继续构建和 FBD 编辑器中那样的网络。这种方式构成的网络，其性能和 FBD 中相应的网络一样。

“Insert”（插入）→ “Box with EN”（有 EN 的框）

使用该命令，将一个功能块、操作符、功能或有 EN 输入的程序插入到一个 LD 网络。标记位置必须是接点和线圈之间的连接线（光标位置 4）或是一个线圈（光标位置 3）。

新的功能块与线圈并联，并在其下插入；它起初被指定为名称 AND。如果你愿意，可以将这个名称指定为另外一个操作符。从出现的“输入助手”对话框中，你可以选择是插入一个用户定义的功能块，还是一个标准的（默认的）功能块。

“Insert”（插入）→ “Insert at blocks”（在块处插入）

使用该命令，你可以将附加的图形元素插入到一个已插入的 POU (也是有 EN 输入的 POU)内。在这个菜单项下面的命令，可以在相同的光标位置执行，如同“Function Block Diagram”（功能块图）（见 5.7 章）中相应的命令那样。

使用“Input”（输入），你可将一个新的输入添加到 POU。

使用“Output”（输出），你可将一个新的输出添加到 POU。

使用 POU，你可以插入一个新的 POU。其过程如同在“Insert”（插入）→ “POU”中所描述的那样。

使用“Assign”（赋值），你可将一个赋值插入到变量。起初，它显示三个问号“???”，你可以使用所需要的变量进行编辑和替代。为此，可以使用“Input assistance”（输入助手）

“Insert”（插入）→ “Jump”（跳转）

使用该命令，你可以在选择的 LD 编辑器原有线圈的结束处插入一个并联的跳转。如果进入线的传递值为“On”（通），跳转将执行到指示的标号。

标记位置必须是接点和线圈之间的连接线(光标位置 4)或者是一个线圈(光标位置 3)。跳转以文本“???”表示。你可以点击该文本，并将它改变为所需要的标号。

“Insert”（插入）→ “Return”（返回）

在 LD 编辑器中，你可以使用该命令将一个“返回”指令并联地插入在原有线圈的结束处。如果进入线传递值“On”（通），则该网络中的 POU 处理被中断。标记位置必须是接点和线圈之间的连接线（光标位置 4）或者是一个线圈（光标位置 3）。

“Extras”（附加）→ “Paste after”（后部粘贴）

LD 编辑器中使用该命令，则在标记位置后面串联地插入粘贴板内容的接点。该命令仅当粘贴板内容和标记位置是由接点组成的网络时才是可能的。

“Extras”（附加）→ “Paste below”（下部粘贴）快捷键，〈Ctrl〉+ 〈U〉

LD 编辑器中使用该命令，则在标记位置下面并联地插入粘贴板内容的接点。该命令仅当粘贴板内容和标记位置是由接点组成的网络时才是可能的。

“Extras”（附加）→ “Paste above”（上部粘贴）

LD 编辑器中使用该命令，则在标记位置上面并联地插入粘贴板内容的接点。该命令仅当粘贴板内容和标记位置是由接点组成的网络时才是可能的。

“Extras”（附加）→ “Negate”（取反）快捷键：〈Ctrl〉+ 〈N〉

使用该命令可以取反当前光标位置(光标位置 2 和 3)的一个接点,线圈,跳转或返回指令,或 EN POU 的输入或输出。

在线圈的括号内,或在表示接点的直线内,会出现一条斜线 (/) 或 | /)。如存在跳转、返回或 EN POU 的输入或输出,则在连接处会出现一个小圆圈(如同在 FBD 编辑器中那样)。

这样,在相应的布尔变量中线圈的写入值是输入连接的反值。此时,如果相关的布尔变量携带值 FALSE,则取反的接点将输入的状态切换到输出状态。

如果标记的是一个跳转或返回,则这个跳转或返回的输入将被取反。通过再次取反,可以取消该取反。

“Extras” (附加) → “Set/Reset” (置位/复位)

如果你在一个线圈上执行该命令,你将会得到一个“Set Coil”(置位线圈)。这样一种线圈从不会改写相应的布尔变量值 TRUE。这意味着,一旦你已经将这个变量的值设置为 TRUE,它将始终保持为 TRUE。

“Set Coil”(置位线圈)在线圈符号中使用“S”指定。

如果你再次执行该命令,你就会得到一个“Reset Coil”(复位线圈)。这样一种线圈从不会改写相应的布尔变量值 FALSE。这意味着,一旦你已经将这个变量的值设置为 FALSE,它将始终保持为 FALSE。“Reset Coil”(复位线圈)在线圈符号中使用“R”指定。

如果你重复执行该命令,该线圈就会在置位、复位和正常线圈之间交替变化。

“Extras” (附加) → “Open instance” (打开实例)

该命令相应于“Project”(项目) → “Open instance”(打开实例)命令。

联机模式的梯形图

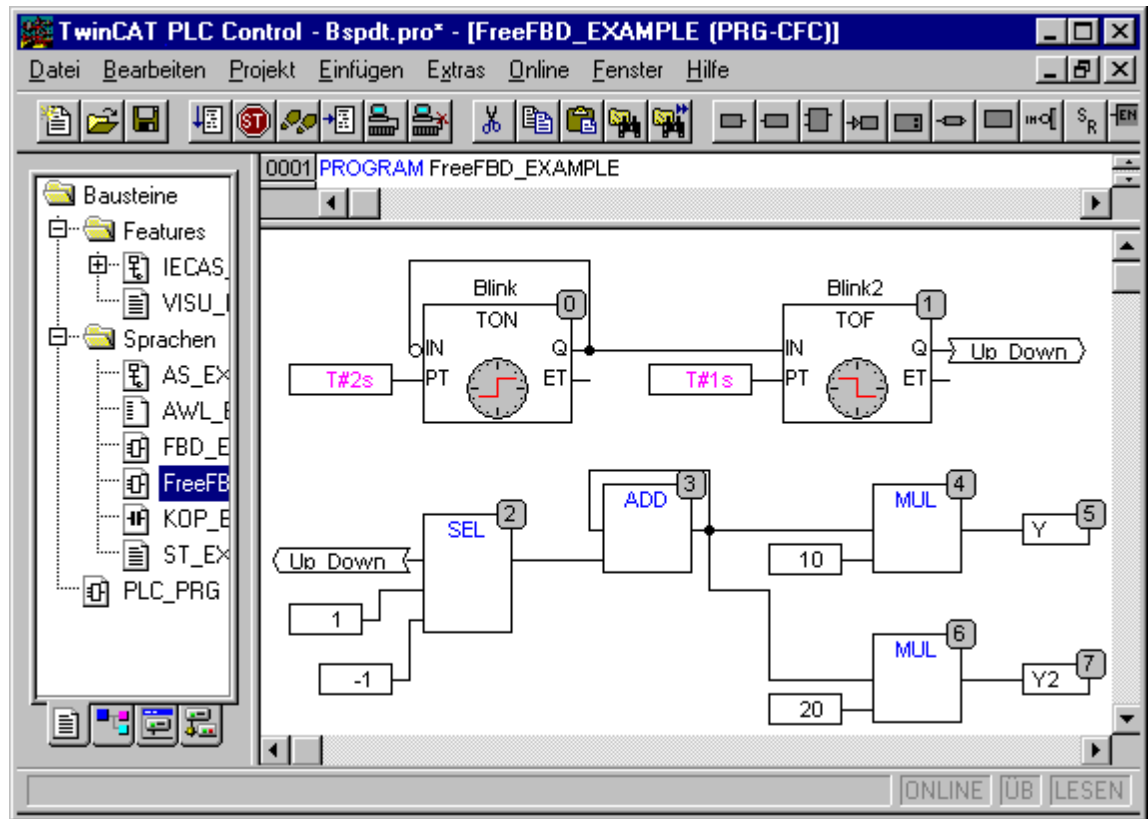
在联机模式时,“Ladder Diagram”(梯形图)中处于“On”状态的接点和线圈显示为蓝色。类似地,所有其上携带有“On”的线也显示为蓝色。在功能块的输入和输出上,则指明相应的变量值。

断点只能在网络上设置,通过使用步进,你可以在网络之间进行跳转。

如果你将鼠标指针停留在一个变量上一段时间,则在“工具提示条”内显示有关该变量的名称和注释。

4.9 连续功能图编辑器

它看起来像由连续功能图编辑器（CFC）生成的一个块（block）：



连续功能图编辑器

连续功能图编辑器不采用格式化网格（snap grid），从而可在任何地方布置图形元素。顺序处理表的图形元素包括块、输入、输出、跳转、标号、返回和注释。这些图形元素的输入和输出可以通过使用鼠标拖动进行连接。连接线是自动描绘的，并对现有的连接线按最短路径进行优化连接。当图形元素被移动时，连接线将自动调整。如果是由于缺乏空间而不能画出连接线，则会在输入和相关输出之间使用一条红色线代替连接线。一旦有足够的空间可以使用，该红色线立即转变为连接线。

与通常的功能块图 FBD 编辑器相比，连续功能图的一个优点是可以直接插入反馈路径。

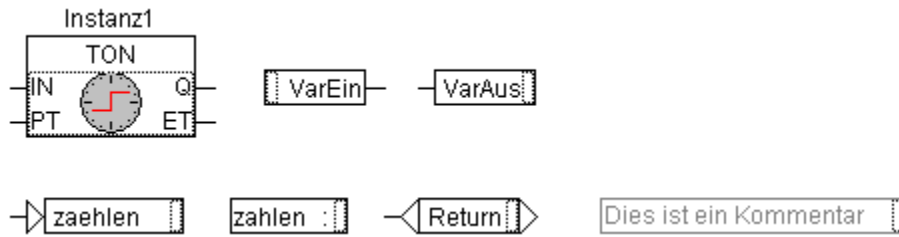
最重要的命令可以在上下文菜单中找到。

连续功能图编辑器 CFC 的光标位置

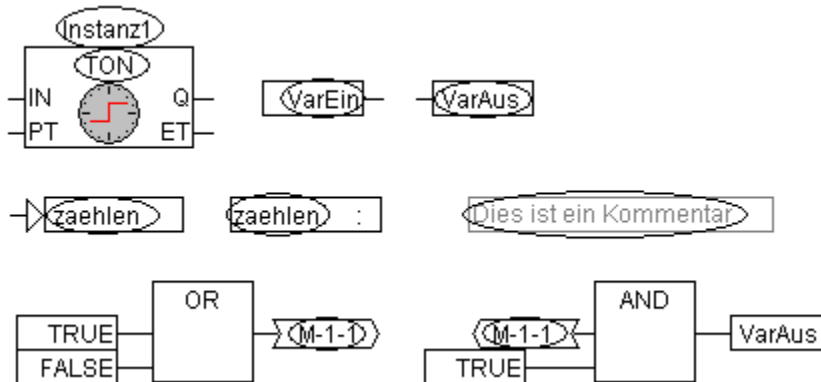
每个文本都是可能的光标位置。选择的文本以蓝色阴影表示，并可以修改。

在所有其它情况下，光标的当前位置由虚点线矩形组成。以下是附有示例的所有可能的光标位置列表：

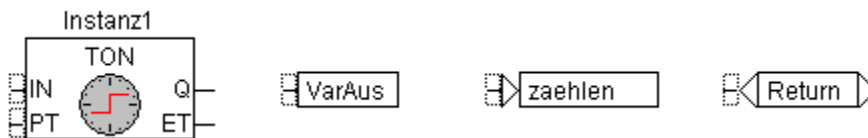
1. 图形元素块、输入、输出、跳转、标号、返回和注释的框架主体。



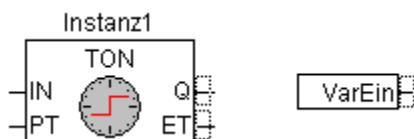
2. 图形元素块、输入、输出、跳转、标号、返回和注释的文本字段，以及用于连接标记符的文本字段：



3. 图形元素块、输入、输出、跳转和返回的输入：



4. 图形元素块和输入的输出：



“Insert”（插入）→ “Block”（块）快捷键：〈Ctrl〉+ 〈B〉

该命令可用于插入操作符、功能、功能块和程序。首先，它总是插入一个“AND”操作符。可以通过选择文本并进行改写而转变为其它操作符，功能块和各种程序。输入助手可以用来从所支持的块列表中选择所需要的块。如果新块有另外的最小输入数，系统会附加这些最小输入数。如果新块有一个较小的最大输入数，则删除最后的输入。

“Insert”（插入）→ “Input”（输入）快捷键：〈Ctrl〉+ 〈E〉

该命令用来插入一个输入。给出的文本“???”可以被选择的变量或常数替代。在这里也可以使用输入助手。

“输入”（Input）→ “输出”（Output）快捷键：〈Ctrl〉+ 〈A〉

该命令用来插入一个输出。给出的文本“???”可以被选择的变量替代。在这里也可以使用输入助手。则与输出相关联的输入值可分配给该变量。

“Insert”（插入）→“Jump”（跳转）快捷键：〈Ctrl〉+〈J〉

该命令用来插入一个跳转。给出的文本“???”可以被选择的程序应跳向的跳转标号替代。

跳转标号使用命令“Insert”（插入）→“Label”（标号）插入。

“Insert”（插入）→“Label”（标号）快捷键：〈Ctrl〉+〈L〉

该命令用来插入一个标号。给出的文本“???”可以被选择的跳转标号替代。在“Online”（联机）模式，用于生成 POU 结束的 RETURN（返回）标号是自动插入的。

跳转使用命令“Insert”（插入）→“Jump”（跳转）插入。

“Insert”（插入）→“Return”（返回）快捷键：〈Ctrl〉+〈R〉

该命令用来插入一个 RETURN（返回）命令。

注意，在“Online”（联机）模式，带有 RETURN 名称的跳转标号自动插入在编辑器的第一栏并且在最后一个图形元素之后；在步进方式时，系统自动跳转到执行离开 POU 之前的位置。

“Insert”（插入）→“Comment”（注释）快捷键：〈Ctrl〉+〈K〉

该命令用来插入一个注释。

使用〈Ctrl〉+〈Enter〉你可以得到一行新的注释行。

“Insert”（插入）→“Block Input”（块输入）快捷键：〈Ctrl〉+〈U〉

该命令用来插入一个块输入。对许多操作符而言，输入的数量是可变的（例如，ADD 可以有两个或更多个输入）。

要为这种操作符增加一个输入数量，必须选择操作符自身（光标位置 1）。

“Insert”（插入）→“In-Pin”（输入插针）→“Insert”（插入）→“Out-Pin”（输出插针）

一旦打开用于编辑的宏指令就可以使用这些命令。这两个命令用来插入作为宏指令输入和输出的输入插针和输出插针。它们与普通的 POU 输入和输出不同之处在于显示方式和没有位置索引。

“Extras”（附加）→“Negate”（取反）快捷键：〈Ctrl〉+〈N〉

该命令用来取反输入，输出，跳转或 RETURN（返回）命令。取反符号是在连接处有一个小圆圈。

当选用该命令时（光标位置 3），图形元素块的输入，输出，跳转或返回是取反的。

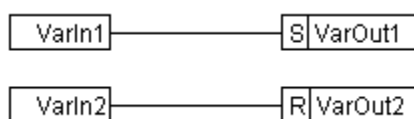
当选用该命令时（光标位置 4），图形元素块的输出或输入是取反的。

通过再次取反可以取消一个取反。

“Extras”（附加）→“Set/Reset”（置位/复位）

该命令只用于所选择的图形元素输出的输入（光标位置 3）。

用于置位的符号是 S，用于复位的符号是 R。



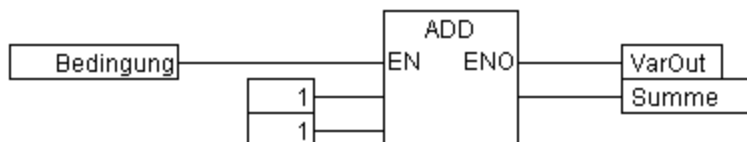
如果 VarIn1 给出 TRUE，则 VarOut1 设置为 TRUE。即使当 VarIn1 返回到 FALSE 时，VarOut1 仍然保持为 TRUE。

如果 VarIn2 给出 TRUE，则 VarOut2 复位为 FALSE。即使当 VarIn2 返回到 FALSE 时，VarOut2 仍然保持为 FALSE。

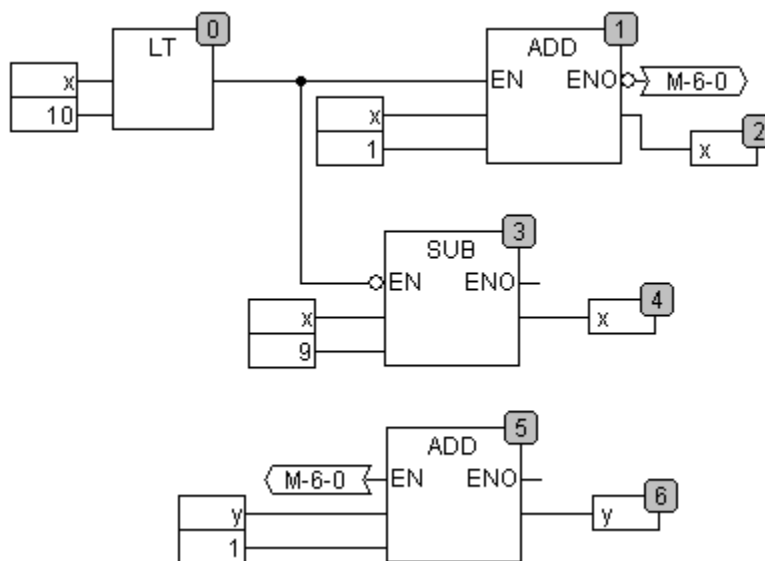
重复激活该命令，将导致输出在置位、复位和正常状态之间变化。

“Extras” (附加) → “EN/ENO” 快捷键: <Ctrl> + <O>

该命令用来给一个选择块 (光标位置 3) 附加一个布尔使能输入 EN (使能输入) 和布尔使能输出 ENO (使能输出)



该示例中，仅当布尔变量“Bedingung” (条件) 是 TRUE 时，才执行 ADD。在 ADD 已经执行后，VarOut 也被设置为 TRUE。当变量“Bedingung” (条件) 是 FALSE 时，则不执行 ADD，并且 VarOut 值将保持为 FALSE。下例说明了如何将 ENO 值用于其它块。

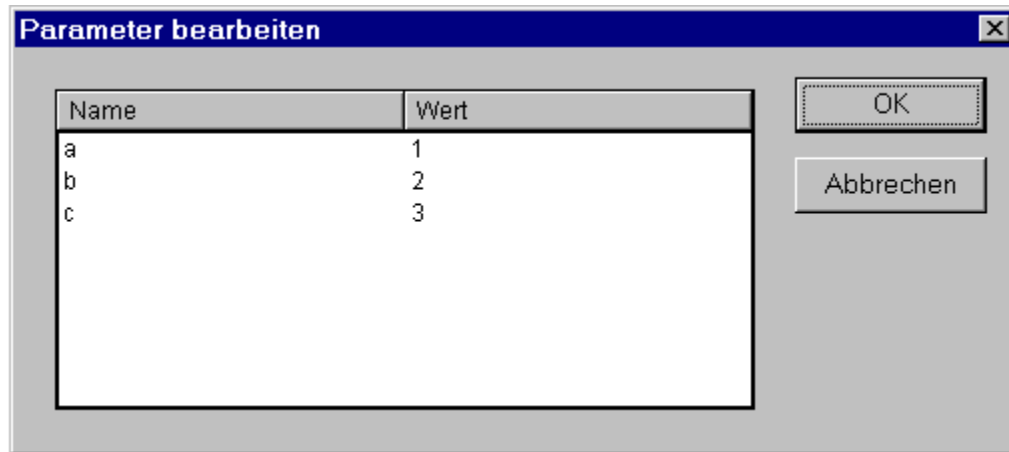


X 应初始化为 1，Y 应初始化为 0。块右上角中的数字表示命令执行的顺序。

X 每次的增量为 1，直到 X 到达值 10 时为止。并导致块 LT (0) 输出值 FALSE。SUB (3) 和 ADD (5) 将被执行。此时，X 被设置返回到值 1，而 Y 值递增一次。只要 X 小于 10，LT (0) 将被再次执行。因此，Y 完成记录 X 通过值范围 1 至 10 的次数。

“Extras” (附加)、“Properties...” (属性)

来自功能和功能块的常数输入参数 (VAR_INPUT CONSTANT) 不直接显示在连续功能图编辑器内。当选择上述 (光标位置 1) 块的框架主体，然后选择命令“Extras” (附加) → “Properties” (属性) 或简单地双击框架主体时，可以显示这些参数并更改其数值。编辑参数对话框被打开：



常数输入参数 (VAR_INPUT CONSTANT) 的值可以更改。这里，有必要在“value” (值) 栏内标出该参数值。再次单击鼠标或按空格键，可以编辑这些参数。按〈Enter〉键可以确认值的改变或按〈Escape〉键来拒绝此项更改。OK 按钮将保存所有的已经作出更改的值。

选择图形元素

在图形元素框架主体上 (光标位置 1) 点击，选择图形元素。

要对更多的图形元素作出标记，可按〈Shift〉键，并在所需要的图形元素上逐一点击，或用鼠标左键拖动鼠标到要加标记的图形元素。

命令“Extras” (附加) → “Select all” (全部选择) 可以一次性地对所有图形元素进行标记。

移动图形元素

使用方向键的同时按〈Shift〉键，可以移动一个或多个选中的图形元素。另一个可能性是通过按下鼠标左键来拖动图形元素。通过释放鼠标左键，只要图形元素没有掩盖其它图形元素或超过编辑器的预留尺寸就可放置这些图形元素。否则，标记的图形元素将返回到它的初始位置并发出一个报警音。

复制图形元素

使用命令“Edit” (编辑) → “Copy” (复制)，可以复制一个或多个选出的图形元素，并可通过命令“Edit” (编辑) → “Paste” (粘贴) 将其插入。

创建连接

一个图形元素的输入可以准确地连接到其它图形元素的输出。一个图形元素的输出可以连接到多个其它图形元素的输入。

图形元素 E2 的输入与图形元素 E1 的输出连接存在多种连接可能性。



将鼠标放在图形元素 E1 的输出上 (光标位置 4)，点击鼠标左键，使鼠标左键保持压下，并拖动鼠标光标到图形元素 E2 的输入上 (光标位置 3)，然后释放鼠标左键。在使用鼠标作这样的拖动过程中，完成了图形元素 E1 的输出连接。

将鼠标放在图形元素 E2 的输入上，点击鼠标左键，使鼠标左键保持压下，并将鼠标光标拖动到图形元素 E1 的输出上，然后释放鼠标左键。

移动图形元素 E1 或 E2 中的一个（光标位置 1）并通过以上方式放置图形元素：拖动鼠标左键，使图形元素 E2 的输出和图形元素 E1 的输入连接。

当图形元素 E2 是带有一个自由输入的块时，通过拖动鼠标，也可作出从 E1 的输出到 E2 的框架主体连接。当释放鼠标时，会创建与在 E2 最高位置处的自由输入连接。当 E2 块没有自由输入而是有附加输入的操作符时，则会自动生成一个新的输入。

通过使用这种方法，可以使一个块的输入和输出彼此连接在一起（反馈路径）。要建立两个插针之间的连接，可以在一个插针上点击鼠标左键，按鼠标按钮并拖动连接线到所需要的插针，然后在那里释放鼠标按钮。如果在连接线拖动过程中，超出了编辑器的工作区域，则自动出现滚屏。对于简单数据类型而言，在连接过程中完成类型测试。如果两个插针的类型不兼容，则光标变成“Forbidden”（禁止）。对于复杂的数据类型，则不执行测试。

删除连接

要删除图形元素 E1 输出和图形元素 E2 输入之间的连接有多种可能性：

选择图形元素 E1 的输出（光标位置 4），并按〈删除〉按钮，或执行命令“Edit”（编辑）→“Delete”（删除）。如果 E1 的输出与多个输入相连接，则同时删除多个连接。

选择图形元素 E2 的输入（光标位置 4），并按〈删除〉按钮，或执行命令“Edit”（编辑）→“Delete”（删除）。

使用鼠标选择 E2 的输入，使鼠标左键保持压下，并移去输入到 E2 的连接线。当鼠标左键在屏幕的自由区释放时，连接被删除。

更改连接

图形元素 E1 的输出和 E2 的输入之间的连接，可以简单地更改为 E1 的输出和图形元素 E3 的输入之间的连接。在 E2 的输入上点击鼠标（光标位置 3），使鼠标左键保持压下，并将鼠标光标拖动到 E3 的输入，然后释放鼠标按钮。

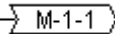
“Extras”（附加）→“Connection marker”（连接标记符）

使用一个连接符号（连接标记符）也可以用于替换一条连接线而表示连接。此时，该输出和相关的输入各有一个附加的、由唯一名称命名的连接标记相连。

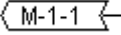
当在两个图形元素之间已经存在一个连接时，现在要以连接标记来表示，则连接线的输出被标记（光标位置 3），并选择菜单点“Extras”（附加）→“Connection marker”（连接标记符）。下图表示的是在这个菜单点选择前、后的一个连接情况



由程序给出的标准化的唯一名称，使用 M 开始，可以进行更改。连接符号名称作为输出参数保存，可以在输入和输出位置进行编辑。

1. 在输出位置编辑连接符号名：

如果连接符号的文本被替代，则输入位置的所有相关的连接符号都采用这个新的连接符号名。然而，我们不能选择已经属于其它连接标记符的名称，因为它违反了连接符号名称的唯一性。

2. 在输入位置编辑连接符号名: 

如果连接符号的文本被替代, 则将删除原有连接标记符的连接, 并创建一个新的连接, 换句话说, 只能给出其它已经存在的连接标记符名称。

可以将连接符号表示的连接转换为标准的连接方式, 只需在标记连接的输出 (光标位置 4) 再次选择菜单命令 “Extras” (附加) → “Connection marker” (连接标记符) 即可。

自由插入输入/输出

如果选择的图形元素正好是一个输入或输出插针, 则可以直接插入相应的输入或输出图形元素, 其编辑字段填充的是通过键盘上输入的字符串。

执行顺序

图形元素块、输出、跳转、返回和标号, 都拥有一个指示其执行顺序的编号。根据这种顺序编号, 各个图形元素都在运行时被分别进行评估。

当粘贴一个图形元素时, 系统将按照拓扑结构顺序 (从左到右, 从上到下) 自动给出一个顺序号。若顺序已经发生改变, 而且所有较高的顺序都已经递增了一个顺序号, 则新的图形元素将接受拓扑结构的继承图形元素编号。

当移动一个图形元素时, 其编号保持不变。

图形元素的顺序会影响结果, 在某些情况下必须加以改变。

如果图形元素的顺序已经被显示, 则相应的顺序执行号显示在图形元素的右上角。

“Extras” (附加) → “Order” (顺序) → “Display” (显示)

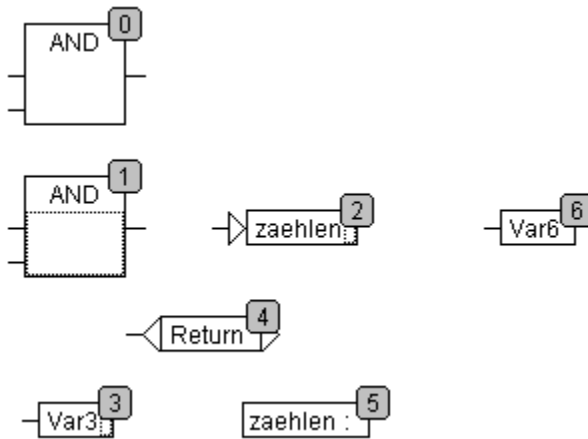
该命令打开和关闭执行顺序的显示。默认的设置是打开显示 (通过菜单点前面的对勾 (✓) 识别)。

执行编号的相对顺序出现在图形元素块、输出、跳转、返回和标号的右上角。

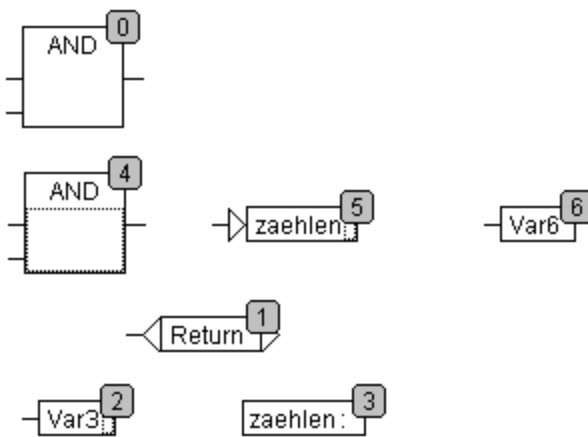
“Extras” (附加) → “Order” (顺序) → “Order topologically” (拓扑顺序)

当执行顺序是从左到右, 从上到下时, 则图形元素是以拓扑顺序安排的, 也就是说, 对于以拓扑形式排列的图形元素而言, 图形元素编号是从左到右, 从上到下增加的。与连接方式无关, 只有图形元素的位置才是重要的。

当执行命令 “Extras” (附加) → “Order” (顺序) → “Order topologically” (拓扑顺序) 时, 所有选择的图形元素都按拓扑结构安排。该过程使所有选择的图形元素都从顺序处理表中取出。然后, 这些图形元素被添加到保留的顺序处理表中, 并分别从右下角到左上角进行处理。每个标记的图形元素都被安放在按拓扑继承的顺序处理表之前, 亦即当编辑器中的所有图形元素都按照拓扑顺序系统进行排序时, 标记的图形元素插入在按拓扑顺序要比它后执行的图形元素前面。这可以通过以下示例清楚地进行说明。



选择编号 1、2 和 3 的图形元素。如果选择命令“**Order topologically**”（拓扑顺序），则它们首先从顺序处理表中取出。然后再将 **Var3**，跳转和 **AND** 操作符分别逐个插入。**Var3** 被安放在标号之前，并接受编号 2。接下来给跳转排序，并接受起初的编号 4，但是在 **AND** 插入后其编号变为 5。新的执行顺序是：



当引入一个新生成的图形元素块时，系统将默认地把它放在顺序处理表中按拓扑继承的编号之前。

“Extras”（附加）→ “Order”（顺序）→ “One forwards”（向前一步）

使用该命令，除了顺序处理表开始位置的图形元素外，所有选择的图形元素都在顺序处理表中向前移动一个编号。

“Extras”（附加）→ “Order”（顺序）→ “One backwards”（向后一步）

使用该命令，除了顺序处理表结束位置的图形元素外，所有选择的图形元素都在顺序处理表中向后移动一个编号。

“Extras”（附加）→ “Order”（顺序）→ “To the beginning”（到开始位置）

使用该命令，所有选择的图形元素都将移动到顺序处理表的前面，但是，仍然保持被选择图形元素组内的顺序。未选择的图形元素组内的顺序也同样保持。

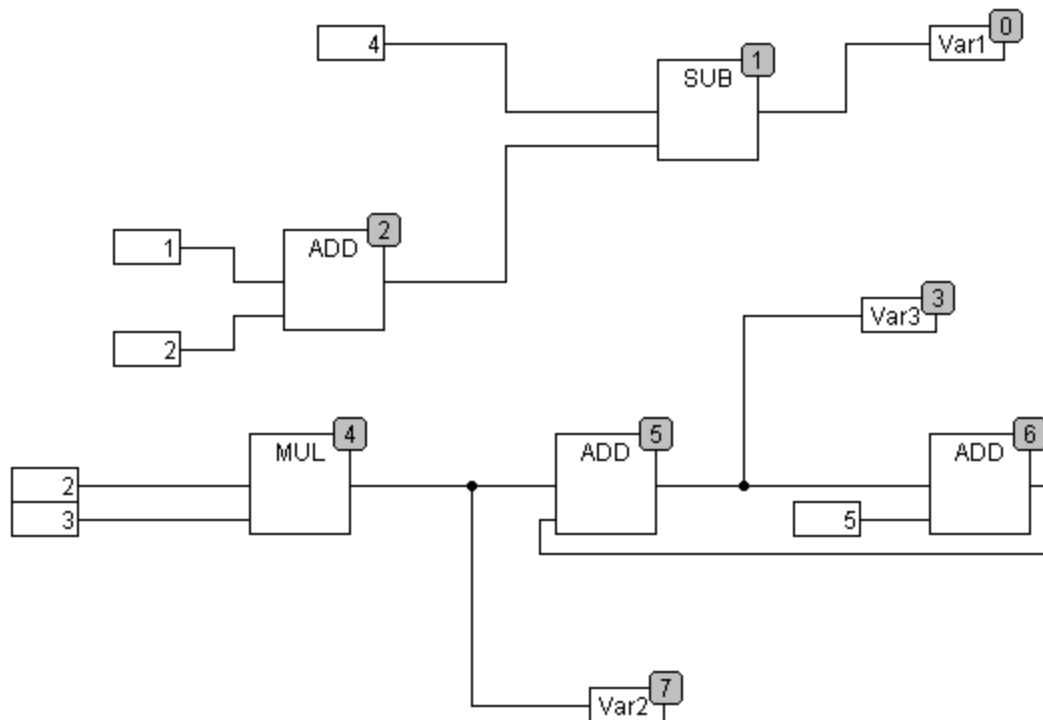
“Extras”（附加）→ “Order”（顺序）→ “To the end”（到结束位置）

使用该命令，所有选择的图形元素都将移动到顺序处理表的结束位置，但是，仍然保持被选择图形元素组内的顺序。未选择的图形元素组内的顺序也同样保持。

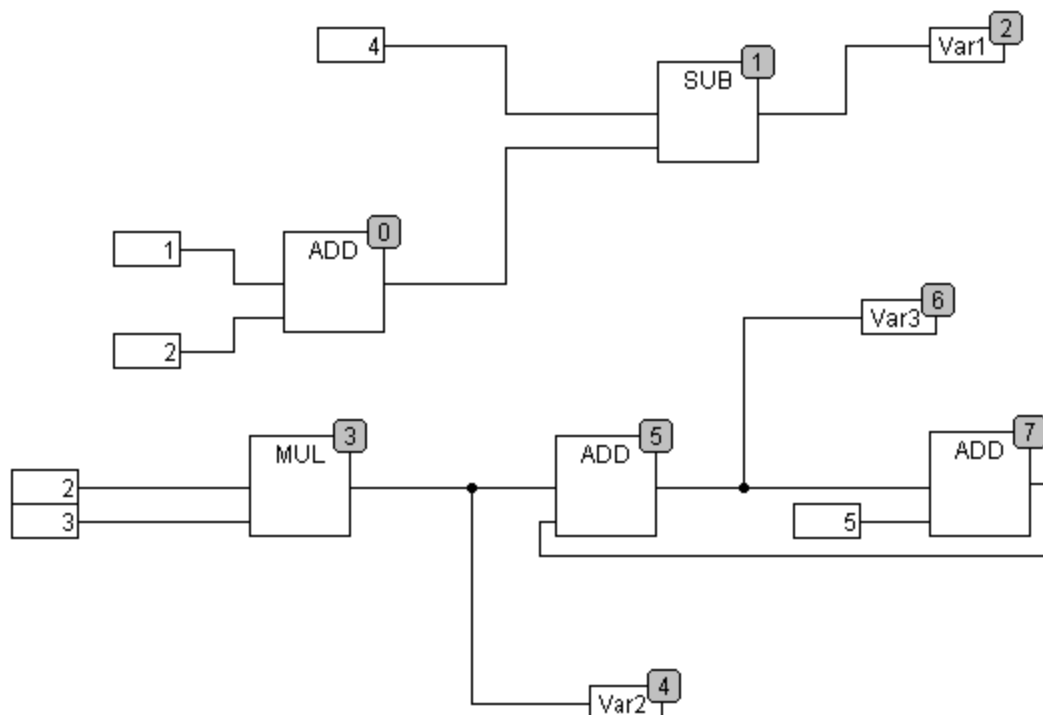
“Extras”（附加）→ “Order”（顺序）→ “Order everything according to data flow”（按数据流排序）

该命令影响所有图形元素。执行顺序由图形元素的数据流确定，而不是由其位置确定。

下图表示的是已经按照拓扑结构顺序排列的各个图形元素。



选择该命令后，则显示以下的图形元素顺序：



当选择该命令时，发生的第一件事情是，图形元素是按拓扑结构顺序排列的。从而创建一个新的顺序处理表。在已知输入值的基础上，计算机计算下一个可以被处理的是哪一个尚未被编号的图形元素。在上面的“network”（网络）中，例如 AND 块，由于其输入（1 和 2）值已知并可以立即处理。SUB 块由于必须先知道来自 ADD 的结果，因而只能随后处理。依此类推。

反馈路径在最后插入

数据流排序的优点是，在数据流排序系统中，与一个块的输出连接的一个输出框在前者的后面立即接上，而在拓扑结构排序时就不一定如此。在某些场合拓扑结构排序会提供与数据流排序不同的结果。从上面例子中可看到这一点。

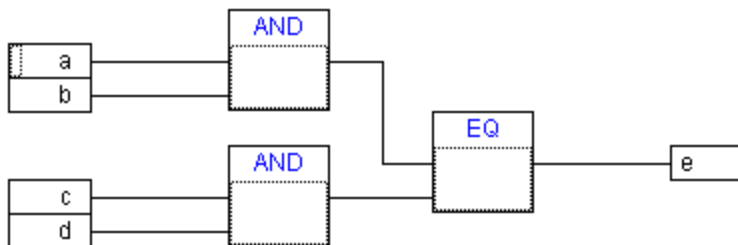
“Extras”（附加）→ “Create macro”（创建宏）

使用该命令，可以将同时选择的多个 POU 汇集在一个块内，并将它命名为一个宏。宏只能通过“Copy/Paste”（复制/粘粘）进行复制，而每个拷贝将成为独立的宏，可以分别地选择其名称。因此，宏不是参考程序。被创建的宏而剪切的所有连接，在宏内生成为输入或输出插针。输入连接生成为输入插针。紧邻插针的默认名为 In (n)。对于输出连接，则显示 Out (n)。在宏创建之前已经标记连接而受影响的连接，则在宏的 PIN 上保留其连接标记符。

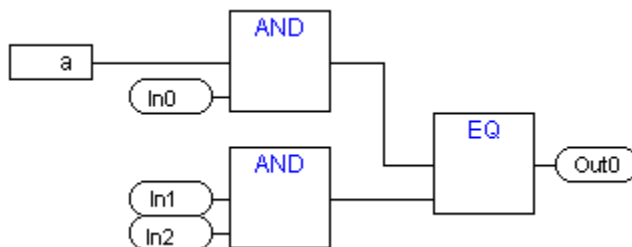
起初，一个宏被默认命名为“MACRO”。这可以在使用的宏“Name”（名称）字段内更改。若编辑该宏，则宏的名称将在编辑器窗口的标题栏内显示，并附加在 POU 名称旁。

示例：

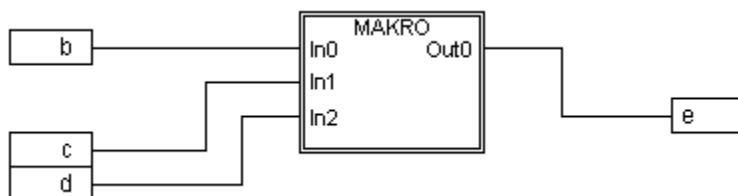
选择：



宏：



在编辑器中：



“Extras”（附加）→ “Edit Macro”（编辑宏）

使用该命令，或双击宏的本身，则会在相关的 POU 编辑器窗口中打开该宏。并在标题栏中附加在 POU 名称之后显示该宏的名称。

在创建宏时所生成的输入和输出插针框可以像普通的 POU 输入和输出那样进行处理。它们也可以进行移动，删除，添加等等。其差别只在于它们是如何显示的以及没有位置索引。插针框有圆角。插针框内的文本与宏显示中的插针名称相匹配。

宏插针框的顺序遵循宏图形元素的执行顺序。较低的索引顺序在较高的索引顺序之前，而较高的插针编号则在较低的插针编号之前。宏内的处理顺序是封闭的，换句话说，宏是作为一个块在主 POU 中的宏位置进行处理的。因此，控制执行顺序的命令只能在宏内进行。

“Extras”（附加）→ “Expand macro”（扩展宏）

使用该命令，选择的宏被重新展开，并且其内部包含的图形元素被插入到 POU 的宏位置。宏插针的连接被再次显示为图形元素的输入或输出连接。如果在宏框位置由于空间不足而使宏不能进行扩展，则宏将在此位置右侧和下方进行显示，直到有足够的空间可供使用时为止。

注：

如果项目在 TwinCAT 2.7 版本中进行保存，宏将完全类似地进行扩展。所有宏也将在转换为其它语言之前被扩展。

“Extras”（附加）→ “Back one macro level”（退回一个宏显示层）→ “Extras”（附加）→ “Back all macro level”（退回所有宏显示层）

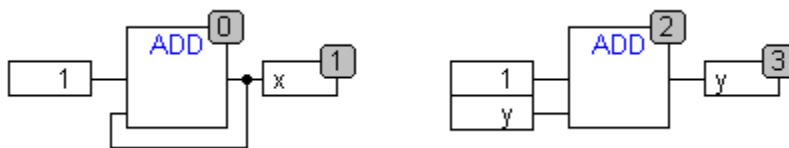
只要一个宏被打开进行编辑，就可以在工具条中使用这些命令。如果一个宏嵌套在另一个宏内，则可以转到下一个较高的显示层或最高显示层。

反馈路径

反馈路径只能直接显示在连续功能图编辑器内，而不能显示在通常的功能块图编辑器内。这里应注意的是，一个块的输出总是带有一个内部的立即变量。该立即变量结果的数据类型，对操作数而言，来自输入的最大数据类型。

常数的数据类型是从最小可能的数据类型获得的，即，常数“1”采用数据类型 SINT。现在，如果执行一次反馈和常数“1”的加法，则第一个输入给出数据类型 SINT，而第二个输入由于是反馈而未进行定义。这样，立即变量也是数据类型 SINT。只在此时才将立即变量值分配给输出变量。

下图显示了一个有反馈的加法和有一个变量的加法。这里，变量 X 和 Y 应是类型 INT。



二种加法之间是有差别的：

变量 Y 可以使用不等于零的值初始化，但对于左边加法的立即变量而言，却不是这样。

用于左边加法的立即变量使用数据类型 SINT，而右边的加法使用数据类型 INT。在第 129 次调用后，变量 X 和 Y 有不同的值。虽然变量 X 是类型 INT，由于立即变量已经溢出，因此，其值为?127。另一方面，变量 Y 的值为 129。

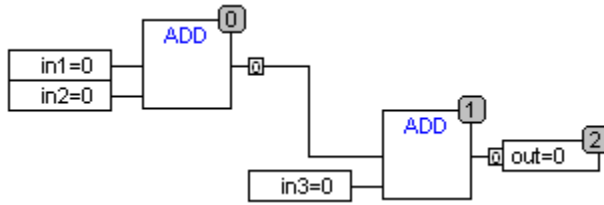
“Extras”（附加）→ “Open instance”（打开实例）

该命令相当于 “Project”（项目）→ “Open instance”（打开实例）命令。

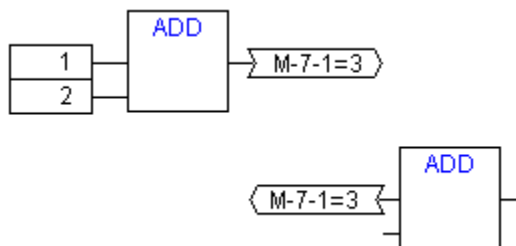
联机模式的 CFC

监视：

输入和输出值显示在输入或输出框内。不对常数进行监视。对于非布尔变量，输出框按照显示值的宽度进行扩展。对于布尔连接，如果值为 **TRUE**，则以蓝色显示变量名和连接。否则它们依旧是黑色的。在 **TRUE** 状态，内部布尔连接在联机模式时也以蓝色显示，否则以黑色显示。内部非布尔连接值则在该连接的带圆角的外部插针小框内进行显示。



宏中的插针监视和输入或输出框那样一样。



带有连接标记符的非布尔连接在连接标记符内显示其值。对于布尔连接，若连接线带有值 **TRUE**，则连接线以及标记符名都以蓝色显示，否则以黑色显示。

流控制

当流控制被打开时，已经通过切断而作为标记的连接使用项目选项中选出的颜色表示。

断点：

断点可设置在所有已有处理顺序索引的图形元素上。程序处理将在相应的图形元素的执行之前暂停，也就是说，对于 POU 和输出而言，在输入赋值之前；对跳转标号而言，在带有下一个索引的图形元素执行之前。图形元素的索引处理顺序在“Breakpoint”（断点）对话框中作为断点位置使用。

在一个选出的图形元素上的设置断点是以 F9 键或通过菜单项“Online”（联机）→“Breakpoint on/off”（断点开/关）或“Extras”（附加）菜单或在编辑器的上下文菜单内作出的。如果断点设置在一个图形元素上，则在下次执行命令“Breakpoint on/off”（断点开/关）时，它将被清除和翻转。此外，一个图形元素上的断点可通过双击该图形元素将断点翻转。

断点使用输入到项目选项中的颜色显示。

RETURN（返回）标号：

在联机模式，带有“RETURN”名称的跳转标号是在编辑器中的第一栏和最后一个图形元素之后自动生成的。该标号标记 POU 的结束，并当步进时跳转到正好在执行过程中离开 POU 之前。RETURN 标记不能插入到宏内。

步进：

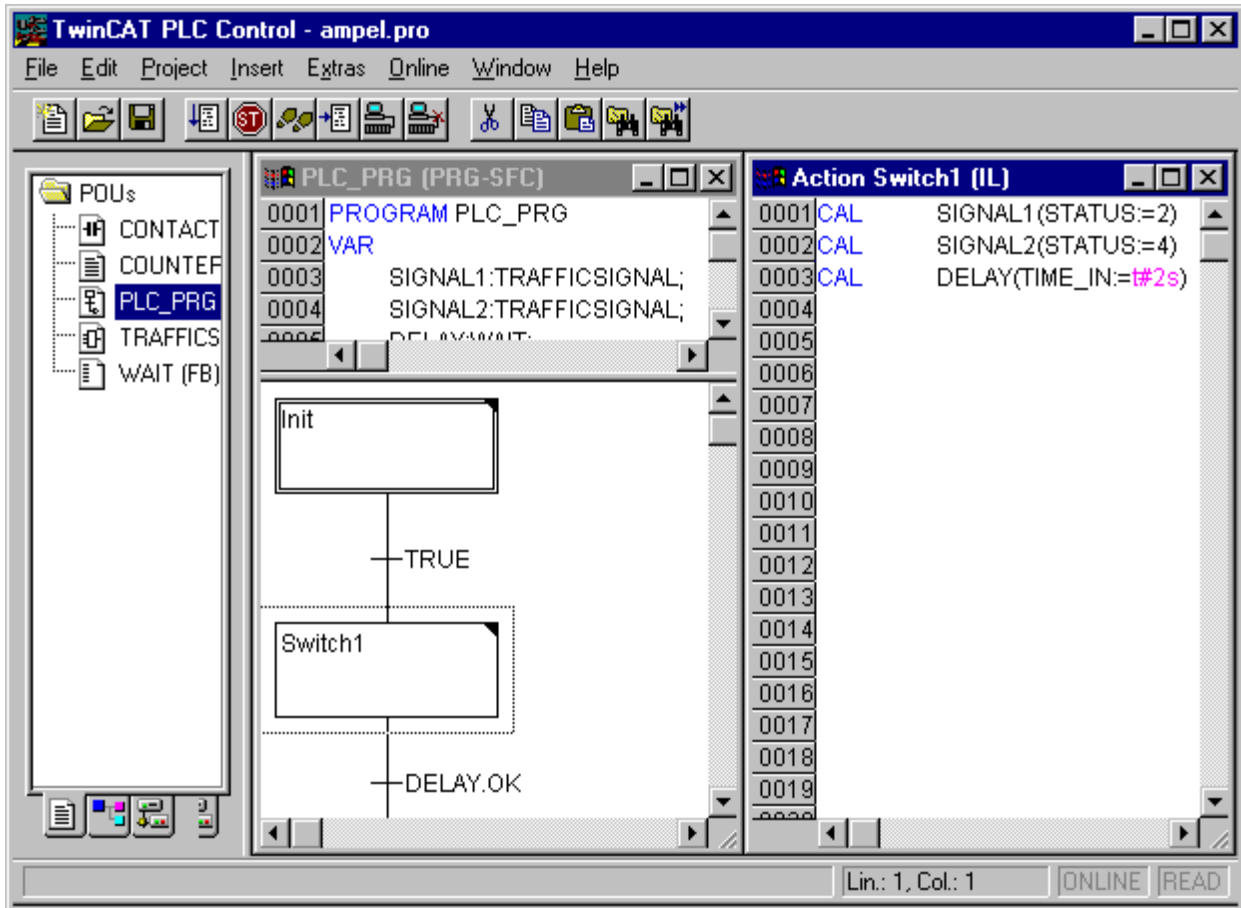
当使用“Step over”（单步跳出）时，总是跳转到下一个带有较高顺序索引的图形元素上。如果当前的图形元素是一个宏或一个 POU，则当“Step in”（单步进入）有效时，进入其实现分支。如果从那里执行“Step over”（单步跳出），则跳转到跟随在这个宏顺序索引之后的图形元素上。

“Extras”（附加）→“Zoom”（缩放）快捷键：〈Alt〉+〈Enter〉

使用该命令，如果选择了一个 POU，则打开这个 POU 的实体。

4.10 顺序功能图编辑器

这是说明写入在 SFC 中的一个 POU 是如何出现在 TwinCAT PLC Control 编辑器内的示例：



用于 POU 的所有编辑器都由一个声明部分和一个本体组成。这两个部分以一个屏幕分隔器隔开。

顺序功能图编辑器是一种图形编辑器。最重要的命令可以在上下文菜单（鼠标右键）中找到。工具提示条显示“Offline”（脱机）和“Online”（联机）模式时，以及在缩放状态时的各个步、转换、跳转、跳转标号、修饰符或相关动作的全名或表达式。

有关顺序功能图资料，可参阅有关“Sequential Function Chart”（顺序功能图）编辑器的内容。

用于顺序功能图的编辑器必须符合 SFC 的特定情况。为此可参考以下使用的菜单项。

SFC 中的标记块

标记块由一系列的、在一个虚线矩形内的 SFC 图形元素组成。（在上面示例的某个地方，步使用 Shift1 标记）

通过将鼠标指向这个图形元素并按鼠标左键，或者你可以使用方向键来选择一个图形元素（步、转换，或跳转）。为了标记一组中的几个图形元素，对已经标记的图形元素块，按〈Shift〉（换挡）键，并选择该组中的左下角或右下角图形元素。选择的最终结果是将包括的这些图形元素聚合为一组最小的图形元素。

要注意的是，仅当选出的图形元素不与该语言的规则相矛盾时，才能执行所有的命令。

“Insert”（插入）→ “Step Transition (before)”（步转换（前向））〈Ctrl〉+ 〈T〉

该命令在 SFC 编辑器中的标记块之前插入一个步，并在其后跟随一个转换。

“Insert” (插入) → “Step Transition (after)” (步转换 (后向)) (Ctrl) + (E)

该命令在 SFC 编辑器中的标记块第一个转换之后插入一个步，并在其后跟随一个转换。

“Insert” (插入) → “Alternative Branch (right)” (选择分支 (右向)) (Ctrl) + (A)

该命令在 SFC 编辑器中插入一个选择分支，并作为标记块的右向分支。为此，标记块必须在开始和结束处都有一个转换。这样，一个转换才能形成这个新分支。

“Insert” (插入) → “Alternative Branch (left)” (选择分支 (左向))

该命令在 SFC 编辑器中插入一个选择分支，并作为标记块的左向分支。为此，标记块必须在开始和结束处都有一个转换。这样，一个转换才能形成这个新分支。

“Insert” (插入) → “Parallel Branch (right)” (并联分支 (右向)) (Ctrl) + (L)

该命令在 SFC 编辑器中插入一个并联分支，并作为标记块的右向分支。为此，标记块必须在开始和结束处都有一个步。这样，这个新分支才能形成一个步。为了能跳转到已创建的并联分支上，必须为这些并联分支提供一个跳转标号。

“Insert” (插入) → “Parallel Branch (left)” (并联分支 (左向))

该命令在 SFC 编辑器中插入一个并联分支，并作为标记块的左向分支。为此，标记块必须在开始和结束处都有一个步。这样，这个新分支才能形成一个步。为了能跳转到已创建的并联分支上，必须为这些并联分支提供一个跳转标号（参见“Extras”（附加）→“Add label to parallel branch”（将标号添加到并联分支））。

“Insert” (插入) → “Jump” (跳转) (Ctrl) + (U)

该命令在 SFC 编辑器中属于标记块的分支结束处插入一个跳转。为此，该分支必须是一个选择分支。在插入的跳转中，插入的文本字符串“Step”可以被选择，并可以使用步名称或需要跳转的并联分支的跳转标号所替代。

“Insert” (插入) → “Transition-Jump” (转换-跳转)

该命令在 SFC 编辑器中插入一个转换，并在所选择的分支结束处跟随一个跳转。为此，这个分支必须是一个并联分支。在插入的跳转中，插入的文本字符串“Step”可以被选择，并可以使用步名称或需要跳转的一个并联分支的跳转标号所替代。

“Insert” (插入) → “Add Entry-Action” (添加进入-动作)

使用该命令，你可以将一个进入动作添加到一个步。一个进入-动作只能执行一次，在此之后，该步将变为有效步。进入-动作可以使用你所选择的语言实现。

带有进入-动作的步命名为“E”，标示在左下角。不能给 IEC 步定义进入-动作。

“Insert” (插入) → “Add Exit-Action” (添加退出-动作)

使用该命令，你可将一个退出动作添加到一个步。一个退出-动作只能在该步变为失效之前执行一次。退出-动作可以使用你所选择的语言实现。

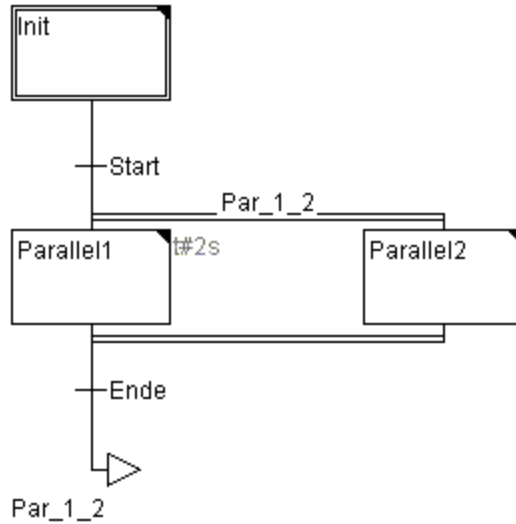
带有退出-动作的步命名为“X”，标示在右下角。不能给 IEC 步定义退出-动作。

“Extras” (附加) → “Paste Parallel Branch (right)” (粘贴并联分支 (右向))

该命令粘贴剪贴板的内容，并作为标记块的一个右向并联分支。为此，标记块必须在开始和结束处都有一个步。类似地，剪贴板的内容必须是一个开始和结束处都有一个步的 SFC 块。

“Extras” (附加) → “Add label to parallel branch” (将标号添加到并联分支)

为了提供一个新插入的并联分支，并使其带有一个跳转标号，则必须标记并联分支之前出现的转换，并且必须执行命令“将标号添加到并联分支”。此时，该并联分支将由一个“Parallel”（并联）和一个附加的序列号组成的标准名称命名，并可以按照标识符名称规则对它进行编辑。在以下的例子中，“Parallel”（并联）以“Par_1_2”替换，并用跳转到转换“End”取代这个跳转标号。



删除一个标号

通过删除标号名可以删除一个跳转标号。

“Extras”（附加）→ “Paste After”（后向粘贴）

该命令将剪贴板中的 SFC 块粘贴在标记块的第一个步或第一个转换之后（普通复制是将其粘贴在标记块的前面。）。现在，如果按照语言准则，最终的 SFC 结构是正确的，则将执行这一命令。

“Extras”（附加）→ “Zoom Action/Transition”（缩放动作/转换）快捷键：<Alt>+<Enter>

标记块的第一个步的动作或标记块的第一个转换的转换本体被装入到已由相应语言写入的编辑器内。如果动作或转换本体是空的，则必须选择将要写入的语言。

“Extras”（附加）→ “Clear Action/Transition”（清除动作/转换）

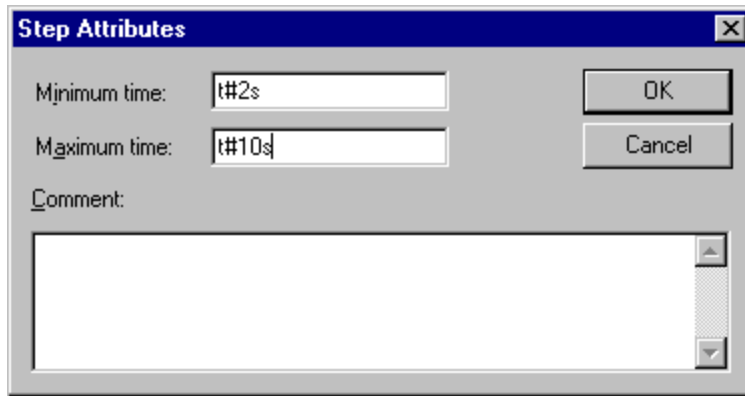
使用该命令，你可以删除标记块的第一个步的动作，或第一个转换的转换本体。

如果是在一个步中，你也许只是实现了动作，进入-动作，或是退出-动作，则使用该命令，它们将被同样删除。否则，出现一个对话框，你可从中选择哪个动作或哪些动作要被删除。

如果光标放置在一个 IEC 步的动作上，则只会删除相关的动作。如果选中了带有相关动作的 IEC 步，则将删除这些相关的动作。当一个 IEC 步带有多个动作时，将出现一个选择对话框。

“Extras”（附加）→ “Step Attributes”（步属性）

使用该命令，你可以打开一个对话框，并对标记步的属性进行编辑。

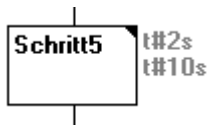


编辑步属性的对话框

你可以使用步属性对话框中三个不同的输入项。在“**Minimum Time**”（最小时间）下，你可以输入处理该步所需的最短时间长度。在“**Maximum Time**”（最大时间）下，你可以输入处理该步所需要的最大时间长度。应注意，输入项都是 **TIME** 类型，因此，你需要使用一个 **TIME** 常数（例如 **T#3s**）或一个 **TIME** 类型变量。

在“**Comment**”（注释）下，你可以将注释插入到步内。在“**Extras**”（附加）→“**Options**”（选项）下打开“顺序功能图选项”对话框中，从而使你可以确定在 **SFC** 编辑器的步中是否显示注释或时间设定值。在右侧，即步的旁边，显示的是注释或时间设定值。

如果超过最大时间，则用户可以查询置位的 **SFC** 标志。



该示例表示，一个步的执行时间应至少持续 2 秒，最大为 10 秒。在“**Online mode**”（联机模式），除了这两个时间外，还会显示该步已经有效的时间是多少。

SFC-标志

在 **SFC** 中，如果一个步的有效大于其属性状态，一些特殊标志将被置位。也有一些变量被置位，以便在顺序功能图中控制程序流。要使用这些标志，则必须将它们声明为全局的或本地的输入或输出变量。

SFCEnableLimit: (SFC 允许极限)

该变量为 **BOOL** 型。当其值为 **TRUE** 时，该步的超时将记录在 **SFCError** 中。忽略其它超时。

SFCInit: (SFC 初始化)

当这个变量值为 **TRUE** 时，顺序功能图返回到 **Init**（初始）步。其它 **SFC** 标志也被复位（初始化）。**Init** 步保持有效，只要该变量的值为 **TRUE**，**Init** 步就不执行。只有当 **SFCInit** 再次被设置为 **FALSE** 时，该块才能被正常地处理。

SFCReset: (SFC 复位)

该变量为 **BOOL** 型，其特性与 **SFCInit** 类似。然而与后者不同的是，在 **Init** 步初始化后将进行进一步的处理。因此，例如 **SFCReset** 标志可以在 **Init** 步中复位为 **FALSE**。

SFCQuitError (SFC 退出出错):

只要这个布尔变量值为 **TRUE**，并且变量 **SFCError** 中可能的超时被复位，则 **SFC** 程序的执行就被停止。当该变量被重新设置为 **FALSE** 时，有效步中所有以前的时间均被复位。

SFCPause (SFC 暂停)

只要这个布尔变量值为 TRUE, SFC 程序的执行就被停止。

SFCError (SFC 错误):

当一个 SFC 程序出现超时, 这个布尔变量值为 TRUE。如果在程序出现第一次超时而后又出现了其它超时, 则该事件将不会被记录, 除非变量 SFCError 被首先复位。

SFCTrans: (SFC 转换)

当一个转换有效时, 这个布尔变量值为 TRUE。

SFCErrorStep: (SFC 出错步)

该变量是 STRING 类型。如果 SFCError 记录了一个超时, 则在这个变量中保存了导致超时的步名称。

SFCErrorPOU: (SFC 出错 POU)

该变量是 STRING 类型, 它包含着发生超时的块名称。

SFCCurrentStep: (SFC 当前步)

这个变量是 STRING 类型。有效的步名称保存在这个变量中, 与时间监视无关。在同时执行的顺序情况下, 该步保存的是外侧的右部分支。

如果出现了超时, 而且变量 SFCError 没有被再次复位时, 则不再记录其它的超时。

SFCErrorAnalyzation: (SFC 错误分析)

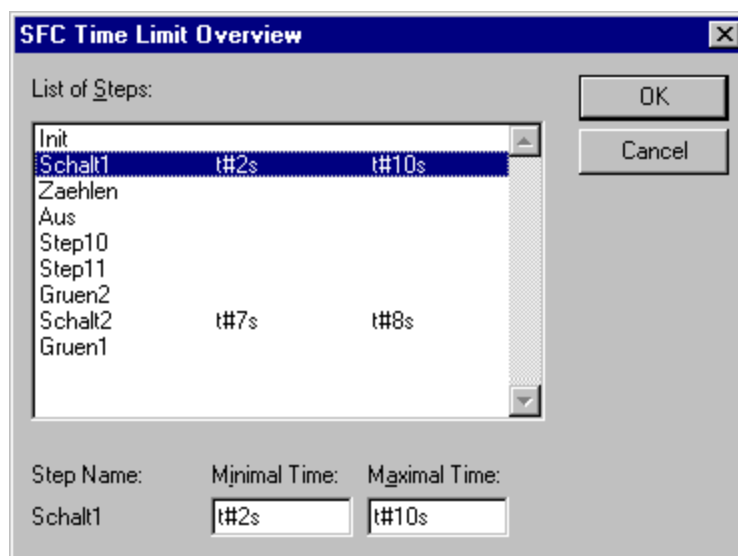
该变量是 STRING 类型, 它提供转换表达式以及集合表达式中的每个变量取值 FALSE 时, 从而导致转换和因此在以前的步中产生超时。为此, 它需要声明用于记录超时的 SFCError 标志。SFCErrorAnalyzation 参考 TcSystem.Lib 库中称为 AppedErrorString 的一个回调功能。输出字符串使用符号 “|” 分隔多项成员。

SFCtip, SFCtipMode:

该变量为 BOOL 类型, 并允许 SFC 为渐进模式。当通过 SFCtipMode=TRUE 允许这个变量时, 如果 SFCtip 设置为 TRUE, 才有可能跳过下一步。只要 SFCtipMode 设置为 FALSE, 甚至有可能跳过转换。

“Extras” (附加) → “Time Overview” (时间总览)

使用该命令, 你可以打开一个窗口, 并在其中编辑 SFC 步的时间设置:



SFC POU 的时间限定总览

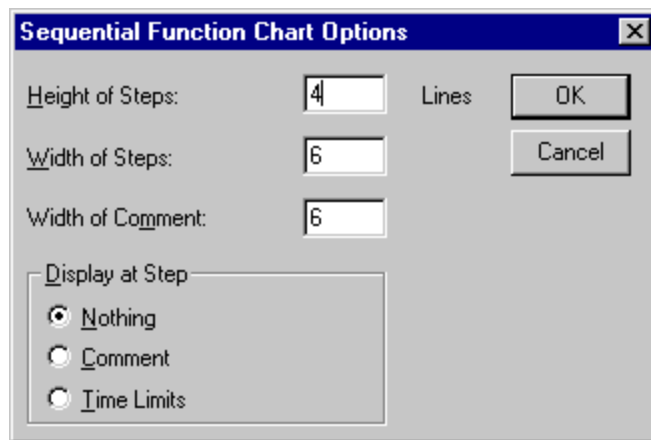
在时间限定总览中，显示你的 SFC POU 的所有步。如果你已经为一个步输入了时间限定，则时间限定将显示在该步的右边（首先是下限定，然后是上限定）。你还可以编辑时间限定。为此，在总览内点击所需要的步。则会在窗口的下方显示步的名称（**name of the step**）。进入到“**Minimum Time**”（最小时间）字段或“**Maximum Time**”（最大时间）字段，并在那里输入所需要的时间限定。

注意，输入项是 TIME 类型，因此，你需要使用一个 TIME 常数（例如 T#3s）或一个 TIME 类型的变量。如果你以 OK 按钮关闭窗口，则将保存所有的更改。

在这个示例中，步 2 和步 6 有时间限定。Shift1 至少保持 2 秒，最大为 10 秒。Shift2 至少保持 7 秒，最大为 8 秒。

“Extras”（附加）→“Options”（选项）

使用该命令打开一个对话框，在这个对话框内，可以为你的 SFC POU 设置不同的选项。



在 SFC 选项对话框中，你可以采用五个输入项。在“**Height of Steps**”（步高度），你可以输入 SFC 编辑器中一个 SFC 步应有多少行高。在这里，4 是标准的设置值。在“**Width of Step**”（步宽度），你可以输入一个 SFC 步应有多少栏宽。这里，标准设置值是 6。如果你需要显示步的注释。“**Width of Comment**”（注释宽度）定义显示的栏数。

可以预置“**Display at Step**”（步显示）。使用这个选项，你有三种选择：你或是显示“**Nothing**”（没有）或显示“**Comment**”（注释），或显示“**Time Limits**”（时间限定）。后两个显示需要将它们在“Extras”（附加）→“**Step Attributes**”（步属性）选项中进行设置。

“Extras”（附加）→“Associate Action”（关联动作）

使用该命令，动作和布尔变量可以和 IEC 步关联。在右侧，或紧邻 IEC 步，连接着一个附加的分配框，用于关联动作。并在左侧字段内预置了限定符“N”和名称“Action”（动作）。这两个预置都可以改变。为此，你可以使用“Input Assistant”（输入助手）。

使用命令“**Project**”（项目）→“**Add Action**”（添加动作），可以在 SFC POU 的对象管理器中创建 IEC 步的新动作。

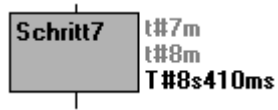
“Extras”（附加）→“Use IEC-Steps”（使用 IEC 步）

如果该命令有效（菜单项前有一个对勾（√）以及工具条中有一个打印符号），则插入的 IEC 步将替代插入步转换和并联分支时的简化型步。

如果该选项被允许，当你创建一个新的 SFC POU 时，Init 步将被设置为一个 IEC 步。

联机模式的顺序功能图

在“Online mode”（联机模式）的顺序功能图编辑器中，当前的有效步将显示为蓝色的步（示例中以黑色表示）。如果你已经在“Extras”（附加）→“Options”（选项）下进行了设置，则各个步的旁边描绘的是时间管理。在你已经设置的下限定和上限定的下方，出现的第三个时间指示符，从中你可以读出有效步的时间是多长。



在上图中，所描绘的步已经有效 8 秒 410 毫秒。也就是说，在该步将要失效前，至少还应有 7 分钟时间是有效的。

通过“Online”（联机）→“Toggle Breakpoint”（翻转断点），断点可以设置在一个步，或在所用语言允许的位置上的一个动作内。因此，可以在执行这个步之前或在程序中的动作位置之前停止处理。在步或程序位置的断点设置使用浅蓝色标记。

如果在一个并联的分支中有几个步同时有效，则将要处理的下一个有效步的动作以红色显示。

如果已经使用了 IEC 步，则在“Online”（联机）模式中的所有有效动作将以蓝色显示。

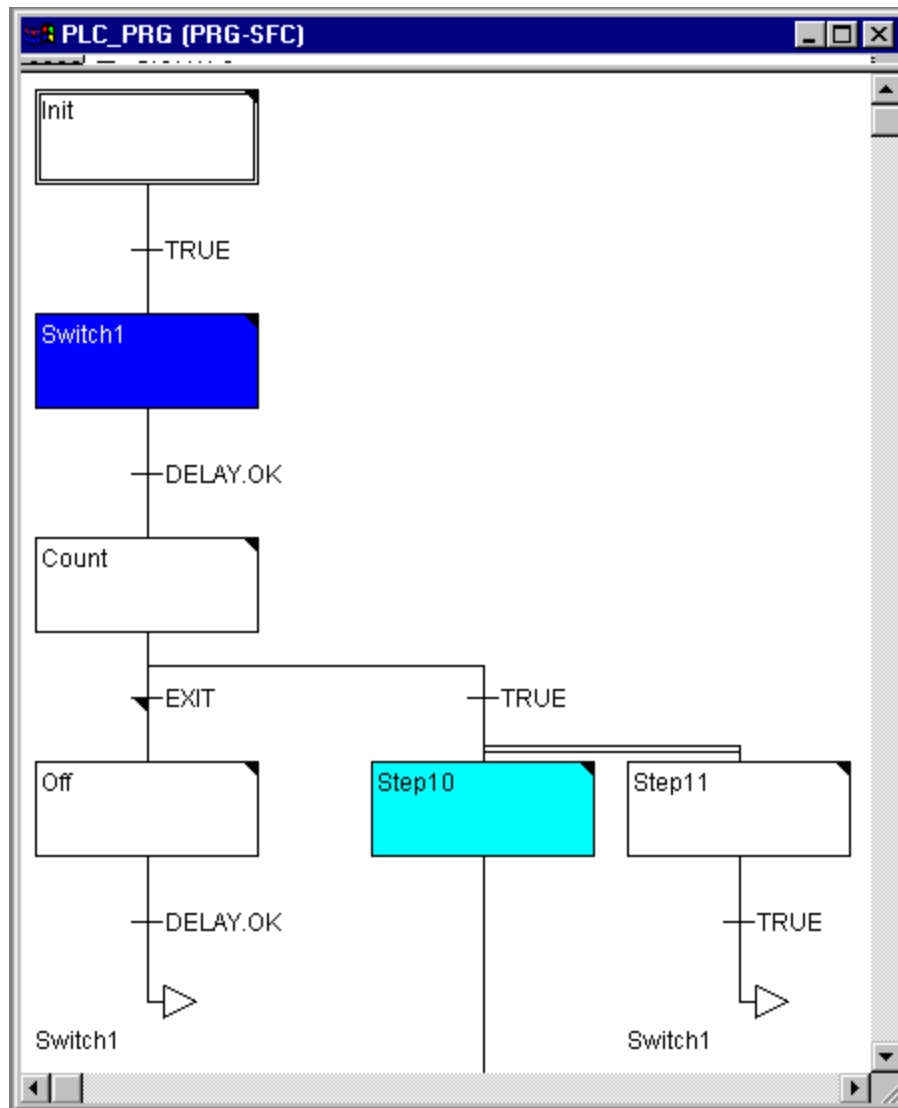
使用命令“Online”（联机）→“Step over”（步进跳出），则总是步进到要执行动作的下一个步。

如果当前的位置是：

- 一个 POU 线性处理中的一个步，或一个 POU 最右侧并联分支中的一个步，执行从 SFC POU 返回到调用程序。如果该 POU 是主程序，则开始下一个循环。
- 除最右侧的并联分支外的并联分支中的一个步，执行跳转到下一个并联分支中的有效步。
- 在一个 3S 动作内的最后断点位置，执行跳转到 SFC 的调用程序。
- 在一个 IEC 动作内的最后断点位置，执行跳转到 SFC 的调用程序。
- 在输入动作或输出动作内的最后断点位置，执行跳转到下一个有效步。

使用“Online”（联机）→“Step in”（步进进入），即使是动作也可以步进进入。如果需要跳转到一个输入，输出或 IEC 动作，则必须在这些位置设置一个断点。在动作范围内，相应编辑器的所有调试功能都可以提供给用户。

在声明编辑器中，如果你将鼠标光标停留在一个变量上一段时间，将会在工具提示条中显示该变量的类型，地址和注释。



在“Online”（联机）模式，顺序功能图中的一个有效步（Shift1）和一个断点（步 10）。



请注意：当这个步有效时，如果你重新命名了一个步并完成“Online Change”（联机更改），则程序将以未定义状态停止！

注：

顺序表中图形元素的处理顺序：

1. 首先，顺序中用于 IEC 动作的所有“Action Control Block”（动作控制块）标志都被复位（但是，动作中被调用的 IEC 动作标志不复位）。
2. 所有的步依次在设定的顺序中进行测试（从上到下，从左到右），以便确定输出动作的执行条件是否满足，如果条件成立，则执行之。
3. 所有的步依次在设定的顺序中进行测试，以便确定输入动作的条件是否满足，如果条件成立，则执行之。

4. 对于所有步，按照它们设定的顺序依次作出以下动作：
 - 若可以应用，已执行的时间被复制到相应的步变量中。
 - 若可以应用，任何超时都会被测试，并且 SFC 的出错标志将按照需求进行服务。
 - 对于非 IEC 步而言，相关的动作正被执行。
5. 顺序中使用的 IEC 动作以字母次序执行。通过动作表它可以在二次传递中完成。在第一次传递时，执行当前循环中失效的所有 IEC 动作。在第二次传递时，执行当前循环中有效的所有 IEC 动作。
6. 评估转换：如果当前循环中的步是有效的，而且随后的转换返回 TRUE（若可以应用，已经历时最小的有效时间），则随后的步变为有效。

实现有关动作时必须注意以下事项：

有可能出现一个动作在一个循环中被多次执行，因为它与多个顺序相关联。（例如，一个 SFC 可以有两个 IEC 动作 A 和 B，两者都在 SFC 中实现，而且二者都调用 IEC 的动作 C；这样在 IEC 动作中，A 和 B 在相同的循环内都是有效的，而且在二者的动作中，IEC 动作 C 可以是有效的；从而使动作 C 被调用二次）。

如果相同的 IEC 动作同时作用于 SFC 的不同层次，由于存在上面所描述的处理顺序，它可能会导致出现并非是你所需的效果。为此，在这种情况下，系统会发出一个错误消息。

在处理用较早的 TwinCAT PLC 所创建的项目过程中，可能会出现这种情况。

注：

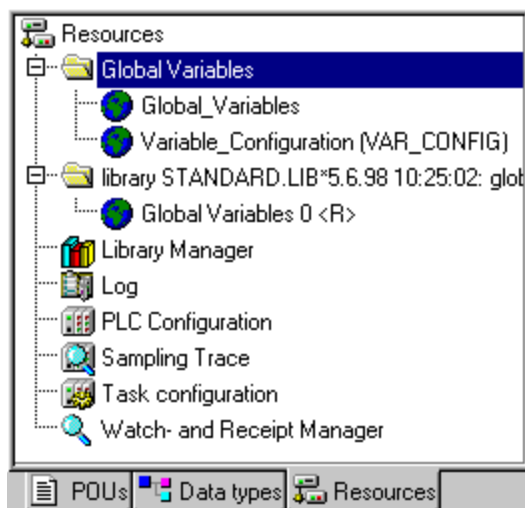
在监视转换的表达式（例如 A AND B）时，只显示该转换的“Total value”（总值）。

5 资源

5.1 概述

在对象管理器的“Resources”（资源）属性页中，有用来配置和组织你的项目，并保持对变量值跟踪的对象：

- 全局变量，可用于整个项目
- 库管理器，用来管理包括在项目内的所有库。
- 日志，用来保存按时间顺序的动作，这些动作是在“Online”（联机）期间出现的事件。
- PLC 配置，用来配置你的硬件。
- 任务配置，用来通过任务调度你的程序控制。
- 抽样跟踪，用于变量值的图形记录。
- 监视和接收管理器，用来指示和预设变量值。



资源

5.2 全局变量

在“Object Organizer (对象管理器)”中，你可以在三个属性页之中的“Resources” (资源) 内找到“Global Variables” (全局变量) 文件夹 (对象的默认名在括号内)。

- 全局变量表
- 变量配置

这些对象中定义的所有变量可以被整个项目识别。若全局变量文件夹未打开 (在文件夹前面有加号)，你可双击该行的 (Enter) 来打开文件夹。选择相应的对象。“Object Open” (对象打开) 命令打开一个以前定义的全局变量的窗口。编辑器在这方面的运行与声明编辑器的工作方式相同。

“Several Variables Lists” (多变量表)

全局变量，全局网络变量 (VAR_GLOBAL)，以及变量配置 (VAR_CONFIG) 必须在单独的对象内定义。如果你已经声明了很大数量的全局变量，现在你想构建结构更好的全局变量表，那么你可创建其它的变量表。在“Object Organizer (对象管理器)”内，选择“Global Variables” (全局变量) 文件夹，或已有的全局变量表。然后执行“Project” (项目) → “Object Add” (对象添加) 命令。命名对话框中出现的相应对象名称。则会在关键字 VAR_GLOBAL 中使用这个名称创建一个附加对象。如果你需要一个变量配置对象，则可将相应的关键字改为 VAR_CONFIG。

“Global Variables” (全局变量)

在整个项目中已知的“Normal” (标准) 变量，常数或保持型变量可作为全局变量加以声明。

“Create Global Variable List” (创建全局变量表)

要创建一个“Global Variables List” (全局变量表)，打开“Object Organizer (对象管理器)”中的“Resources” (资源) 属性页并选择“Global Variables” (全局变量) 条目或选择一个已有的变量表。然后选择命令“Project” (项目) → “Object” (对象) → “Insert” (插入)，打开全局变量列表对话框。

该对话框也可以通过命令“Project” (项目) → “Object” (对象) → “Properties” (属性) 打开，如果已有的全局变量列表在对象管理器中进行了标记。

编辑保持型“Global Variables List” (全局变量表)

若它们受运行系统的支持，保持型变量可以被处理。

有二种类型的保持型全局变量：

在运行系统的一次受控停机 (TwinCAT 停机) 或一次“Online” (联机) → “Cold reset” (冷复位) 或一次装载后，持久保持型变量和一般保持型变量均保持不变。

持久保持型变量不像一般保持型变量那样是自动赋值的！

保持型变量由附加的关键字 RETAIN 或 PERSISTENT 定义。

语法：

```
VAR_GLOBAL RETAIN
    (*变量声明*)
END_VAR
VAR_GLOBAL PERSISTENT
    (*变量声明*)
END_VAR
```

全局常数

全局常数由附加的关键字 CONSTANT 定义。

语法:

```
VAR_GLOBAL CONSTANT
    (*变量声明*)
END_VAR
```

“Variable Configuration” (变量配置)

在功能块中，有可能在关键字 `VAR` 和 `END_VAR` 之间定义变量，以指定未完整定义的输入和输出的地址。未完整定义的地址用一个星号标识。

示例:

```
FUNCTION_BLOCK locio
VAR
    loci AT %I*: BOOL := TRUE;
    loco AT %Q*: BOOL;
END_VAR
```

这里定义了两个本地 I/O 变量，即一个本地输入 (%I*) 和一个本地输出 (%Q*)。

如果你要配置本地 I/O，可在资源管理器的“Resources”（资源）属性页中配置变量，通常利用对象“Variable_Configuration”（变量配置）。对象可以重新命名，并可以创建其它的变量配置对象。

用于变量配置功能的编辑器类似于声明编辑器。用于本地 I/O 配置的变量必须放置在关键字 `VAR_CONFIG` 和 `END_VAR` 之间。这样一种变量的名称由一个完全的实例路径组成，通过该路径，各个 POU 和实例名称彼此用点号分隔开。声明必须包含一个地址，其类别“Input/output”（输入/输出）应与功能块中未完整定义的地址 (%I*, %Q*) 相对应。同时，数据类型也必须和功能块中的声明相匹配。

由于实例不存在，配置变量的实例路径无效时，也作为错误指出。另一方面，如果不存在实例变量的配置，也报告一个错误。为了接收所有相关配置变量的列表，可以使用“Insert”（插入）菜单中的“**All Instance Paths**”（所有实例路径）菜单项。

示例:

假设在程序中定义了下面的功能块:

```
PROGRAM PLC_PRG
VAR
    Hugo: locio;
    Otto: locio;
END_VAR
```

则正确的变量配置如下:

```
VAR_CONFIG
    PLC_PRG.Hugo.loci AT %IX1.0 : BOOL;
    PLC_PRG.Hugo.loco AT %QX0.0 : BOOL;
    PLC_PRG.Otto.loci AT %IX1.0 : BOOL;
    PLC_PRG.Otto.loco AT %QX0.3 : BOOL;
END_VAR
```

这里定义了两个本地 I/O 变量，即一个本地输入 (%I*) 和一个本地输出 (%Q*)。

注:

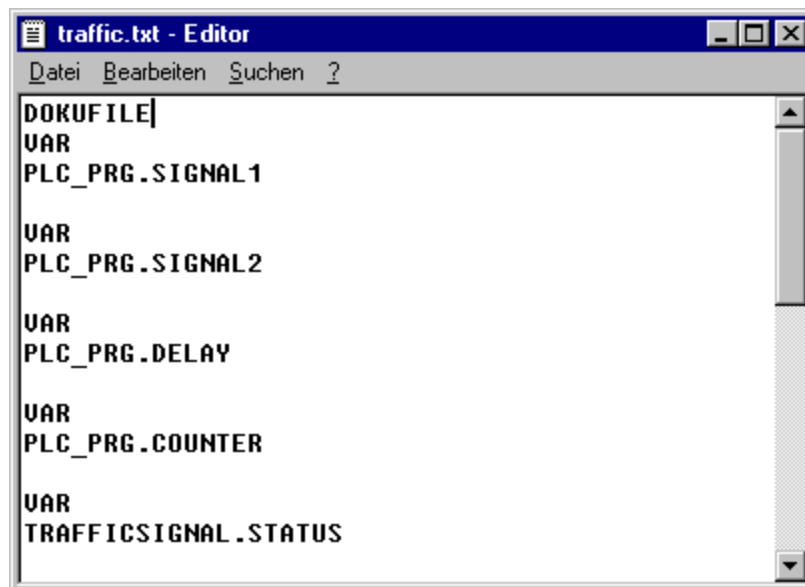
要注意，不要使用已用于变量配置的输出，以及在项目内或用变量的输出（AT 声明）。这些都不再予以说明。

“Insert”（插入）→ “All Instance Paths”（所有实例路径）

使用该命令，生成一个 VAR_CONFIG - END_VAR 块，它包含项目中可供使用的所有实例路径。手头已有的声明不必重新插入，以防止包含已有的地址。若项目已完成编译，可在变量配置窗口内找到这个菜单项（“Project”（项目）→ “Rebuild All”（全部重建））。

“Document Frame”（文档框架）

若一个项目要接收多文档，或许有德文和英文注释，或者你要将几个使用相同变量名的类似项目存档，则可以通过使用 “Extras”（附加）→ “Make Docuframe File”（建立文档框架文件）命令建立文档框架，使你可以节省许多工作。创建的文件可以装入到所需要的文本编辑器内并可进行编辑。该文件有一个起始行 **DOKUFILE**。然后列出项目变量表，后跟为每个变量分配三行的排列：一个 VAR 行显示当前出现的新变量；接着是变量名行，最后是一空行。你可以用一条变量注释来替换该行。也可以简单地删除不想归档的任何变量。如果你愿意，也可为项目建立几个文档框架。



有文档框架的 Windows 编辑器

为了使用文档框架，可选择 “Extras”（附加）→ “Link Docu Frame”（链接文档框架）命令。现在，如果要将整个项目进行归档，或打印项目的一部分，则在程序文本中，会将文档框架中所产生的注释插入到所有变量内。该注释只在打印输出时出现！

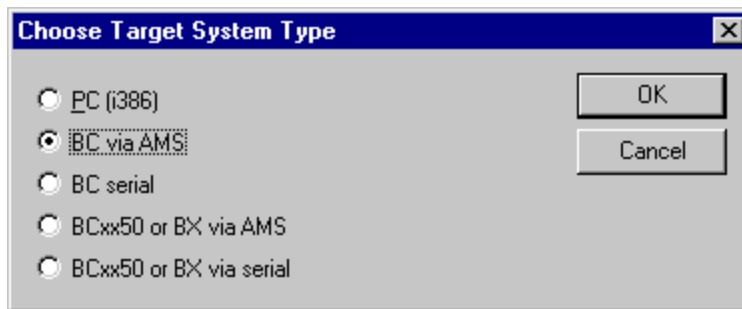
“Extras”（附加）→ “Make Docuframe File”（建立文档框架文件）

使用该命令，建立一个文档框架。每当你从全局变量中选择一个对象时可任意使用该命令。为保存这个新文件，会打开一个对话框。在文件名字段内，已输入了 *.txt 扩展名。选择所需要的名称。现在，已建立了一个文本文件，其中列出了项目中的所有变量。

“Extras”（附加）→ “Link Docu File”（链接文档文件）

使用该命令，可选择一个文档框架。并显示打开文件对话框。选择所需要的文档框架并按 OK。现在，如果你要对整个项目进行存档，或打印项目的一部分，则在程序文本中，将文档框架中所产生的注释插入到所有的变量内。该注释只在打印输出时出现！为了建立一个文档框架，可以使用 “Extras”（附加）→ “Make Docuframe File”（建立文档框架文件）命令。

5.3 PLC 配置



PLC 配置取决于相应的配置硬件。TwinCAT 支持三种不同的硬件平台：

- PC (i386)：PLC 程序在 PC 上运行。
- BCxxxx：PLC 程序在总线控制器上运行。
- BCxx50 或 BXxxxx

如你原意将 PLC 程序在一个 BCxxxx/BCxx50 或 BXxxxx 总线端子 PLC 上运行，可以选择是否需要装入程序。

- 通过 AMS（取决于现场总线）
- 通过串行接口

5.4 任务配置

一个任务是处理一个 IEC 程序的一个时间单元。它通过名称，优先级和确定在何种条件下触发任务的类型进行定义。该条件只能由时间“cyclic”（周期）定义。

对于每个任务而言，你可以指定一系列由任务启动的程序。

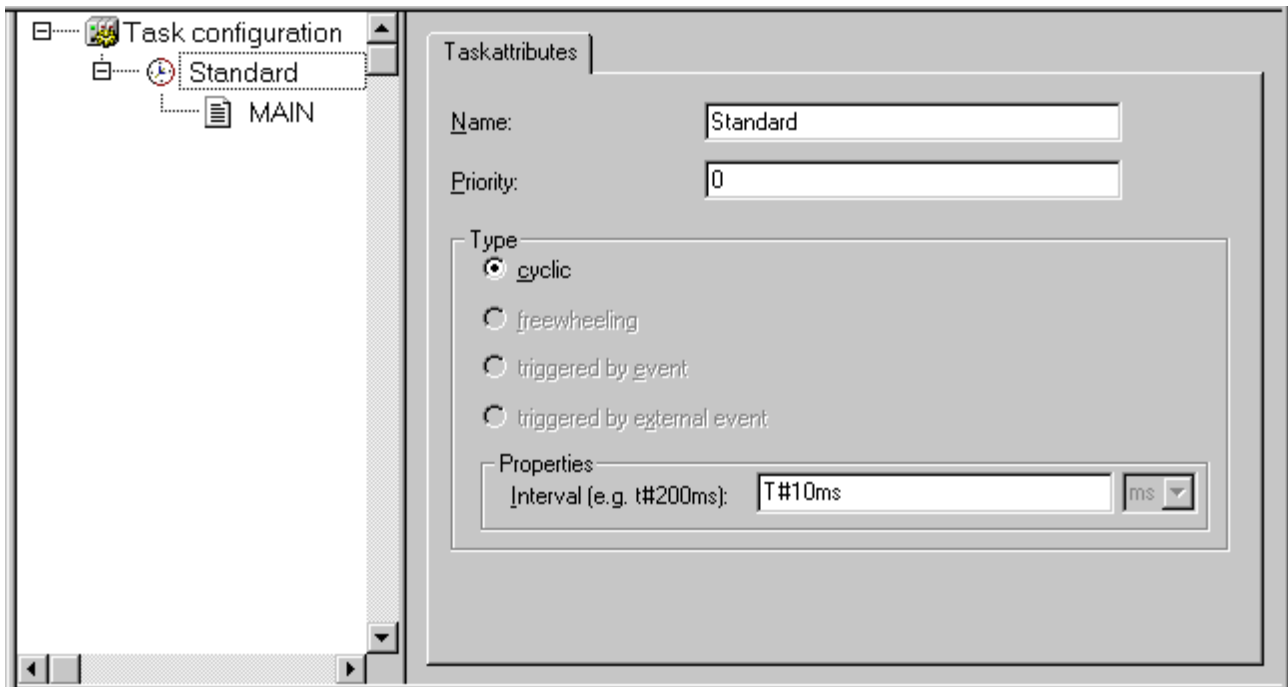
优先级与条件的组合将确定任务以何种时间顺序执行。

对于每个任务而言，你可配置一个看门狗监视器“time control”（时间控制）。

在“Online”（联机模式），可以使用图形监视任务处理。

此外，还有可能将系统事件（例如启动，停止，复位）直接与项目中执行的 POU 相链接。

在对象管理器的“Resources”（资源）属性页中可以找到作为对象的“Task Configuration”（任务配置），“Task”（任务）编辑器在一个对半分开的窗口内被打开。



在窗口的左半部分，以树形结构表示任务配置。在其顶部，总能找到“Taskconfiguration”（任务配置）条目。其下是全部特定任务项，分别以任务名表示。在每个任务项目下是插入分配的程序调用。

在窗口的右半部分，显示一个对话框，它属于树形结构配置中当前标出的项目。在这里，你可以配置任务和程序调用。



注：请不要在几个任务中使用相同的字符串功能，因为，它会引起由于改写而出现程序错误。

与“Task Configuration”（任务配置）一起工作

你可以在上下文菜单中找到最重要的命令（鼠标右键）。

任务配置的标题是“Task Configuration”（任务配置）。如果在其之前出现一个加号，表明顺序表是关闭的。通过双击该表或按〈Enter〉键，可将该表打开。随即出现一个减号。再次双击该表可重新关闭。

- 对每个任务而言，都有一个链接的程序调用表。类似地，你可用同样方法打开和关闭这个

表。

- 使用“Insert”（插入）→“Insert Task”（插入任务）命令，可插入一个任务。
- 使用“Insert”（插入）→“Append Task”（添加任务）命令，可在配置树的结尾处插入一个任务。
- 使用“Insert”（插入）→“Insert Program Call”（插入程序调用）命令，可插入一个程序调用。

随后，在窗口右侧，配置树的每个配置项目都会显示一个对应的对话框。可以在此分别进行有效/失效选择。还可以生成输入的编辑字段。取决于所选择的配置树项目，会出现定义“Task Attributes”（任务属性）或定义“Program Call”（程序调用）的对话框。只要光标再次落到配置树上，对话框中所作出的设置就会被配置树接收。

也可以在配置树上编辑一个任务名或程序名。为此，用鼠标点击该任务名，或选择该项目并按（Space）（空格）键，打开一个编辑框。

你可用箭头键选择配置树中的上一个或下一个项目。

“Insert”（插入）→“Insert Task”（插入任务）或“Insert”（插入）→“Append Task”（添加任务）

使用该命令，你可以将一个新任务插入到任务配置中。如果选中一个任务，则“Insert Task”（插入任务）命令就可由你任意支配。新任务将插入到所选择的任务之后。如果选择“Task Configuration”（任务配置），则可以使用“Append Task”（添加任务）命令，新任务就会添加到已有任务表的末端。并为你打开设置任务属性的对话框。

插入所需要的属性：

- “Name”（名称）：用于任务的一个名称：该名称在配置树中表示任务；用鼠标点击该项时，或在选择该项时按（Space）（空格）键，可在配置树中编辑该任务名称。
- “Priority (0-3)”（优先级 (0-3)）：（0 到 31 之间的一个数字；0 是最高优先级，31 是最低优先级）。
- “Type”（类型）：
- **Cycle (循环)**：按照“间隔”字段中定义的时间，该任务将以循环方式处理。

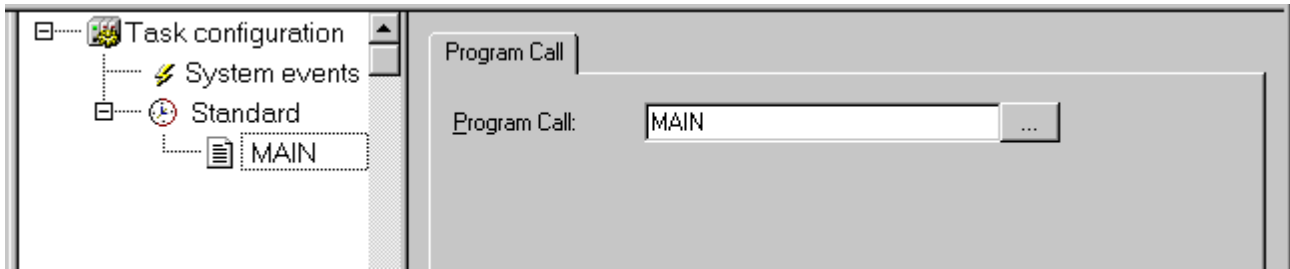
自由轮询，由事件触发，或由外部事件触发；目前均不支持这些任务类型！

“Properties”（属性）：

- 间隔（用于“循环”类型）：时间周期，经过这段时间后，任务将被重新启动。如果你输入一个数，则在编辑字段后的选择框中，你可选择所需要的单位：毫秒 [ms] 或微秒 [μ s]。一旦窗口被重新刷新，以 [ms] 为单位的格式输入将立即显示为 TIME 格式（例如“t#200ms”）；但是你也可以直接输入 TIME 格式。以 [ms] 为单位的输入总是以一个纯数据进行显示（例如“300”）。

“Insert”（插入）→“插入程序调用”（Insert Program Call）或“Insert”（插入）→“Append Program Call”（添加程序调用）

使用这些命令，你可以打开一个对话框，并为任务配置中的任务输入程序调用。使用“Insert Program Call”（插入程序调用）命令，新的程序调用插入到光标前，而使用“Append Program Call”（添加程序调用）命令，程序调用将被添加到已有程序调用列表的末端。



在程序字段中，可以为你的项目指定一个有效的程序名，或按...按钮打开“输入助手”，或用 (F2) 选择一个有效的程序名。程序名也可在配置树中进行编辑，为此，可以用鼠标点击该程序名，或选择该项并按 (Space) (空格) 键，打开一个编辑框。如果选出的程序需要输入变量，则可以按照声明类型的常规形式输入 (例如，prg(invar:=17))。

“Extras” (附加) → “Set Debug Task” (设置调试任务)

使用该命令，可以在“Online” (联机) 模式时，在任务配置中设置调试任务。任务设置后，将出现文本 [DEBUG]。

因此，调试功能只对该任务有效。换句话说，如果该程序通过设置任务运行，该程序只在一个断点处停止。

“Debug Task” (调试任务) 的设置被保存在项目中，并自动在系统录入/装载时重新进行设置。

“Extras” (附加) → “Display Callstack” (显示调用栈)

如在调试过程中程序停在一个断点处，则该命令可用来显示相应 POU 的调用栈。为此，必须在任务配置树中选择调试任务。窗口“Callstack of task <task name>” ((任务名) 的任务调用栈) 将被打开。在此窗口中，你可以获得 POU 的名称和断点位置 (例如 "prog_x(2)", 表示 POU prog_x 的第 2 行)。在完整的调用栈下部显示的是回调顺序。如果你按“GO TO”按钮，光标就会跳转到 POU 中当前在调用栈中标记的位置。

5.5 抽样跟踪

抽样跟踪表示在一定的时间内，对变量值的进程进行跟踪。这些值被写入到一个环形缓冲区（跟踪缓冲区）内。若内存已满，则内存起始处最早的值将被覆盖。

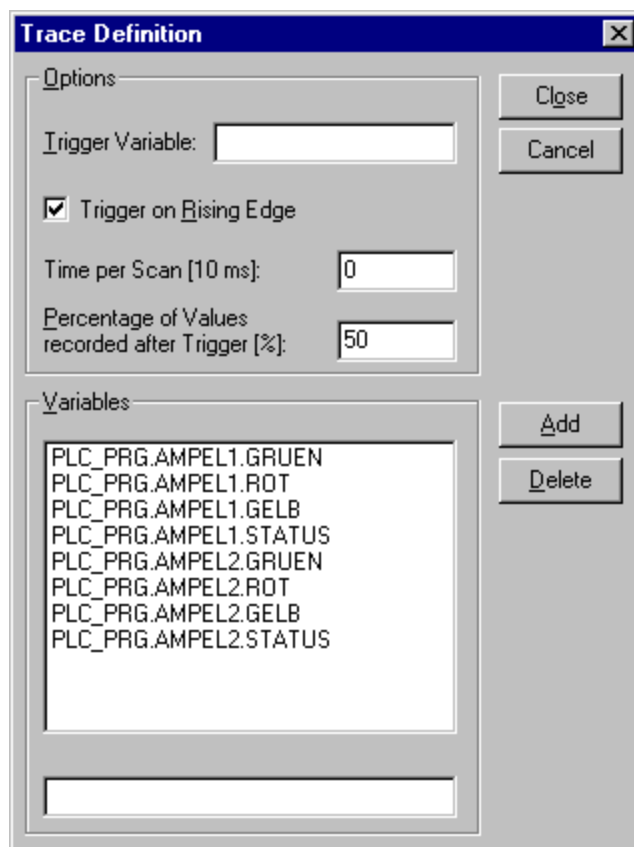
作为最大值，可同时跟踪 20 个变量。由于 PLC 中跟踪缓冲区的容量有限，当出现变量很多或变量长度很大（DWORD）的情况时，只能跟踪较少数量的变量值。

为了完成跟踪，应打开对象管理器“Resources”（资源）属性页中的“Sampling Trace”（抽样跟踪）对象。之后，你必须输入需要跟踪的跟踪变量。（参见“Extras”（附加）“Trace Configuration”（跟踪配置）。）

当你已经完成将“Define Trace”（定义跟踪）的配置发送给 PLC 之后，并在 PLC 中启动了跟踪（“Start Trace”）（启动跟踪），则变量值才会被跟踪。使用“Read Trace”（读跟踪）命令，可读出最终的跟踪值并以曲线图形进行显示。

“Extras”（附加）→“Trace Configuration”（跟踪配置）

使用该命令，你可以在打开的对话框中输入将要跟踪的变量，以及抽样跟踪所需要的相关跟踪参数。



用于跟踪配置的对话框

要跟踪的“Variables”（变量）表被初始化为空白。为了添加一个变量，必须在列表字段中输入变量。之后，你可以使用 ADD(加)按钮或(Enter)(回车)键将变量添加到列表中。你也可以使用“Input Assistant”（输入助手）。

当你选择一个变量然后按“Delete”（删除）按钮时，也可从列表中删除该变量。

布尔值变量或模拟值变量可以输入到字段“**Trigger Variable**”（触发变量）中。这里也可以使用输入助手。触发变量描述了跟踪结束的条件。在“**Trigger Level**”（电平触发）中，你可以输入一个模拟值触发变量的电平（当到达该电平时触发事件）。当选择上升沿“**Trigger edge**”（脉冲触发）时，在布尔值触发变量产生上升沿后，或者当一个模拟值变量产生了从电平触发值开始的由低电平到高电平的跳变时将触发事件。而（**Negative**）（下降沿触发），则当布尔值变量产生下降沿后，或者当模拟值变量产生了从电平触发值开始的由高电平到低电平的跳变时触发事件。（**Both**）（边沿触发），表示当出现上升沿或下降沿时，或者出现正向脉冲或负向脉冲时触发事件，而（**none**）则表示根本就不触发事件。

“**Trigger Position**”（触发位置）用来设置在触发事件发生前记录测量值的百分率。例如，如果你在这里输入 25，则当触发事件发生时，其之前显示的是测量值的 25% 的数据，其之后显示的是测量值的 75% 的数据，然后跟踪终止。字段的抽样率则使用“**Time per Scan**”（每次扫描时间）字段，你可以使用毫秒为单位指定二次扫描之间的时间。默认设定值“0”表示每次循环扫描一次过程。

选择记录值的再调用方式：使用“**Single**”（单次），（**Number of the defined samples**）（指定的采样数量）只显示一次。使用“**Continuous**”（连续），则每次重新启动测量值指定的记录数量。例如，如果你输入数量为“35”，则第一次显示的是包含第一次测量值 1 到 35 的采样值，之后的 35 个测量值（36 - 70）的记录会自动被读出。依此类推。（**Manual**）（手动）选择用来读出在“**Extras**”（附加）→“**Read trace**”（读跟踪）所指定的跟踪记录。

再调用方式功能与是否设置了触发变量无关。如果没有设置触发变量，跟踪缓冲区将用指定的测量值数量填充，而缓冲区的内容将在再调用时被读出和显示。

“**Save**”（保存）按钮用来保存已在一个文件中建立的跟踪配置。为此，“**File save as**”（文件另存为）标准窗口将被打开。

“**Load**”（装入）按钮用来回放保存的跟踪配置。为此，“**File open**”（文件打开）标准窗口将被打开。

注：

请注意，**Save** 和 **Load** 在配置对话框中只与配置有关，而与跟踪记录值无关（对照命令“**Extras**”（附加）→“**Save trace**”（保存跟踪）和“**Extras**”（附加）→“**Load trace**”（装入跟踪））。

如果字段“**Trigger Variable**”（触发变量）为空，跟踪记录将不断运行并可以使用“**Extras**”（附加）→“**Stop Trace**”（停止跟踪）命令使其停止。

“Extra”（附加）→“Start Trace”（启动跟踪）

使用该命令，跟踪配置传输到 PLC，并启动 PLC 的跟踪采样。

“Extra”（附加）→“Read Trace”（读跟踪）

使用该命令，从 PLC 中读出当前的跟踪缓冲区，并显示所选择的变量值。

“Extra”（附加）→“Auto Read”（自动读出）

使用该命令，从 PLC 自动读出当前的跟踪缓冲区，并连续显示值。如果跟踪缓冲区是自动读出的，则在菜单项之前有一个对勾（✓）。

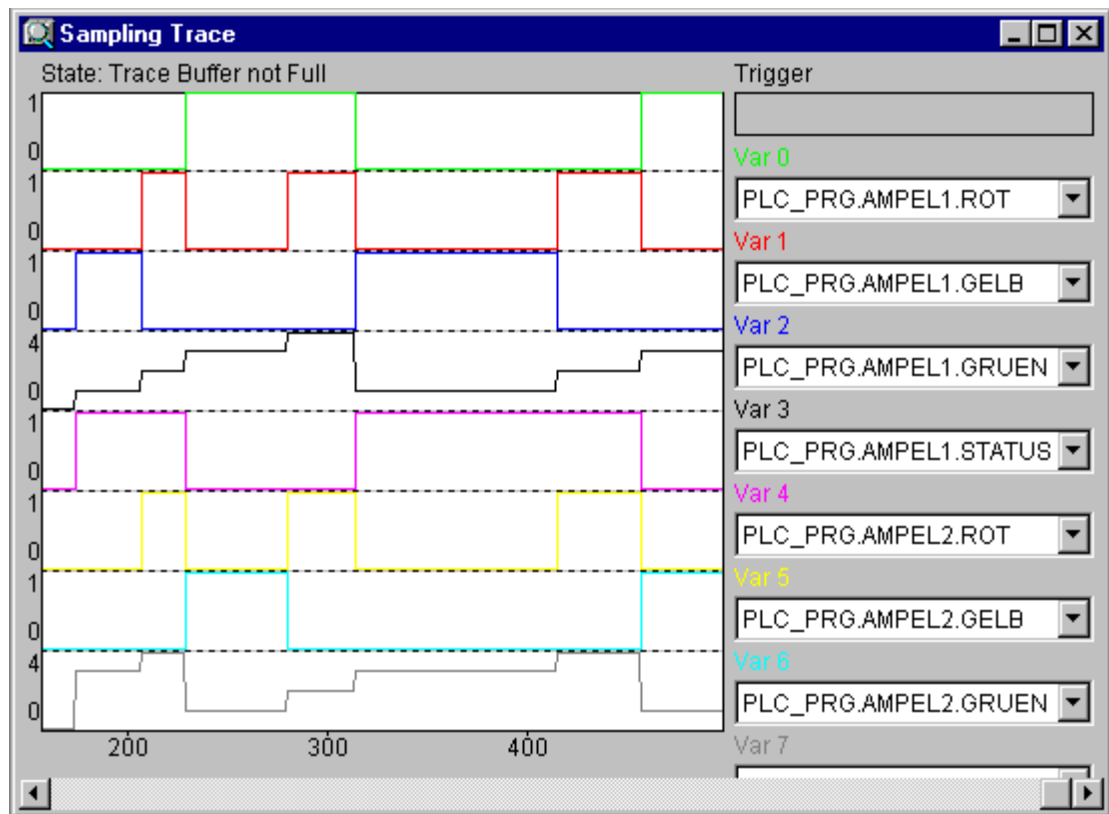
“Extra”（附加）→“Stop Trace”（停止跟踪）

该命令停止 PLC 中的“**Sampling Trace**”（采样跟踪）。在一次新的跟踪之前，必须装入跟踪定义，而且必须重新运行跟踪过程。

“Selection of the Variables to be Displayed”（选择要显示的变量）

右侧的组合格，紧接着跟踪配置定义的跟踪变量曲线显示窗。若从列表中选择一个变量，当跟踪缓冲区被读出后，该变量将以相应的颜色（Var 0 绿色，等等）显示。即使曲线已经显示，也可以选择变量。

在跟踪窗内，最多可同时观察 8 个变量。



无触发器的八个不同变量的采样跟踪

一旦跟踪缓冲区被装入，将要显示的所有变量值均会被读出并且进行显示。如果尚未设置扫描频率，则 X 轴将输出跟踪值的连续序号。

跟踪窗口的状态指示器（第一行）将指示跟踪缓冲区是否被填满，以及何时完成跟踪。如果指定了扫描频率值，则 X 轴将显示指定的时间值。该时间值对应最早的跟踪值。在这个例子中，例如，显示值是最后 25 秒钟时间所对应的跟踪值。

Y 轴输出整数值。标定的方法是使最小值和最大值都能放置到观察区中。在这个例子中，Var5 的最小值为 6，最大值为 11：因而标定设置在左边缘。

如果遇到触发请求，则在出现触发请求的前后值之间接口处，会显示一条垂直虚线。

已经被读出的内存将被保留，直到你改变项目或离开系统时为止。

“Extras”（附加）→ “Cursor Mode”（光标模式）

在监视区域内设置一个光标的最简单方法是在那里点击鼠标左键。会出现一个光标并可被鼠标移动。在监视窗口的顶部将显示当前光标的 X 位置。在 'Var 0'、'Var 1'、...、'Var n' 旁边的字段中，显示的是相应变量的值。

另一种方法是命令 “Extras”（附加）→ “Cursor mode”（光标方式）。使用该命令，在 “Sampling Trace”（采样跟踪）中，将出现二条垂直线。起初，它们彼此重叠。使用方向键可将其中的任何一条线向左或向右移动。通过按 <Ctrl>+<left> 或 <Ctrl>+<right> 按钮，可增加移动速度。

如果鼠标指针位于图形窗口内，并且鼠标左键被按下，则光标也会以类似的方式显示。

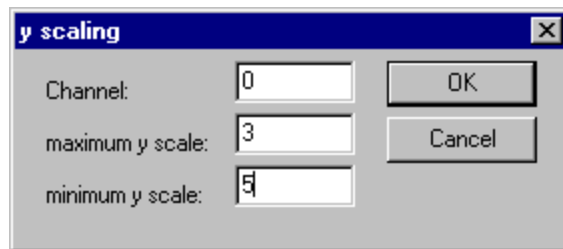
如果同时按下 <Shift> 键，则可移动第二条线，以显示与第一条线的不同。

“Extras”（附加）→ “Multi Channel”（多通道）

使用该命令，你可在采样跟踪的单通道和多通道显示之间进行变换。在多通道显示情况下，菜单项的前面有一个对勾（√）。多通道显示已预先设置。此时，显示窗口被分割成同时显示 8 条曲线。对于每条曲线而言，其边缘显示的是其最大值和最小值。在单通道显示时，所有的曲线都以相同的标定因子显示，而且是重叠的。当显示的曲线反常时该特性是有用的。

“Extras”（附加）Y → “Scaling”（Y 标定）

使用该命令，你可以改变跟踪显示曲线的标定预设置值 Y。在对话框中指定所需要的曲线数量“Channel”（通道）以及 Y 轴上新的最大值“maximum y scale”（最大 Y 标定）和新的最小值“minimum y scale”（最小 Y 标定）。双击曲线，你也可以得到该对话框，通道数和以前的值是预置的。



设置 Y 标定的对话框

“Extras”（附加）→ “Stretch”（扩展）

使用该命令，你可以扩展（缩放）所显示的采样跟踪值。其起始位置是通过有水平图形的调节工具条进行设置。通过一次又一次的重复扩展动作，窗口中显示的跟踪部分将越来越收缩其尺寸。该命令与“Extras”（附加）“Compress”（压缩）命令作用相反。

“Extras”（附加）→ “Compress”（压缩）

使用该命令，显示的采样跟踪值被压缩；亦即，在该命令后，你可在一个更大的时间范围内观察跟踪变量的进展。该命令可被多次执行。该命令与“Extras”（附加）→“Stretch”（伸展）命令作用相反。

“Extras”（附加）→ “Save Trace”（保存跟踪）

使用该命令，你可保存一个采样跟踪。保存文件对话框被打开。选择所需要的文件扩展名“*.trc”。用“Extras”（附加）→“Load Trace”（装入跟踪）命令可以将保存的采样跟踪重新装入。

“Extras”（附加）→ “Load Trace”（装入跟踪）

使用该命令，保存的采样跟踪可以被重新装入。“打开文件”对话框被打开。选择所需要的文件扩展名“*.trc”。用“Extras”（附加）→“Save Trace”（保存跟踪）命令，你可以保存一个采样跟踪。

“Extras”（附加）→ “Trace in ASCII-file” 实现（ASCII）文件跟踪

使用该命令，你可以将采样跟踪保存在一个 ASCII 文件中。“保存文件”对话框被打开。文件的扩展名为“*.txt”。并在文件中按以下格式储存采样跟踪值：

TwinCAT PLC Control Trace

D:\TWINCAT PLC CONTROL\PROJECTS\TRAFFICSIGNAL.PRO

Cycle PLC_PRG.COUNTER PLC_PRG.LIGHT1

0 2 1

1 2 1

2 2 1

.....

如果在跟踪配置中没有设置扫描频率，则将周期放置到第一栏；这意味着，每个周期都有一个值在任何给定的时刻被记录。另一方面，该点在时间轴上以 **ms** 为单位，其对应的变量值是自从采样跟踪运行以来已经保存的记录值。

在随后的栏内，是跟踪变量相应的保存值。在任何给定的时间内，用一个空格将这些值彼此分开。

按照顺序（PLC_PRG.COUNTER, PLC_PRG.LIGHT1），从第二栏开始，是逐个显示的附属变量名。

5.6 监视和接收管理器

借助于“Watch and Receipt Manager”（监视和接收管理器），你可以观察所选出的变量值。‘监视和接收管理器’也使预设置变量的确定值成为可能，并可将这些变量作为一个组传输到 PLC（“Write Receipt”（写接收））。以相同的方式，当前的 PLC 值可以被读出并保存在“Watch and Receipt Manager”（监视和接收管理器）内（“Read Receipt”（读接收））。这些功能是很有帮助的，例如，用来设置和输入控制参数。

所有创建的监视表（“Insert”（插入）→“New Watch List”（新的监视表））都在“Watch and Receipt Manager.”（监视和接收管理器）的左侧栏内显示。这些监视表可以使用鼠标点击或使用一个方向键来进行选择。在“Watch and Receipt Manager”（监视和接收管理器）的右侧，显示的是在任何给定的时间内可以使用的变量。

为了使“Watch and Receipt Manager”（监视和接收管理器）能够运行，可以打开对象管理器“Resources”（资源）属性页中“Watch and Receipt Manager”（监视和接收管理器）对象。

在“Offline Mode”（脱机模式）时，使用“Insert”（插入）“New Watch List”（新的监视表）命令，你可以在“Watch and Receipt Manager”（监视和接收管理器）中创建若干个监视表。

为了使输入的变量能够被监视，你可以使用“Input Assistant”（输入助手）打开所有变量的列表，或者你可以使用键盘，并按照以下的符号表示法输入变量：

<POUName>.<Variable Name>.

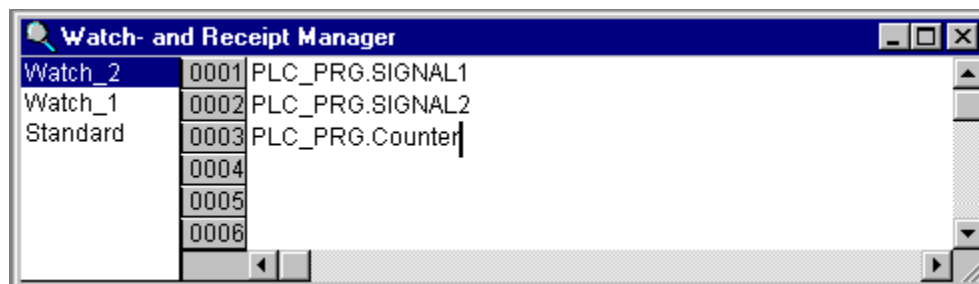
使用全局变量时，POU 名可以被省略，你可以使用一个点号作为开始。变量名可以再次包含多个层次。地址可以直接输入。

多层次变量示例：

PLC_PRG.实例 1.实例 2.结构.成员名

全局变量示例：

.全局 1.成员 1



脱机模式时的监视和接收管理器

监视表中的变量可以使用常值进行预设置。这意味着，在“Online”（联机）模式时，你可以使用“Extras”（附加）→“Write Receipt”（写接收）命令将这些值写入到变量中去。为此必须使用 := 将常数赋值给变量：

示例：

PLC_PRG.COUNTER:=6

在这个示例中，PLC_PRG.COUNTER 变量被预设置为值 6。

“Insert”（插入）→“New Watch List”（新的监视表）

使用该命令，可将一个新的监视表插入到“Watch and Receipt Manager”（监视和接收管理器）内。在出现的对话框中输入所需要的监视表名称。

“Extras”（附加）→“Rename Watch List”（重新命名监视表）

使用该命令，你可以改变“Watch and Receipt Manager”（监视和接收管理器）中监视表的名称。在出现的对话框中，输入新的监视表名称。

“Extras”（附加）→“Save Watch List”（保存监视表）

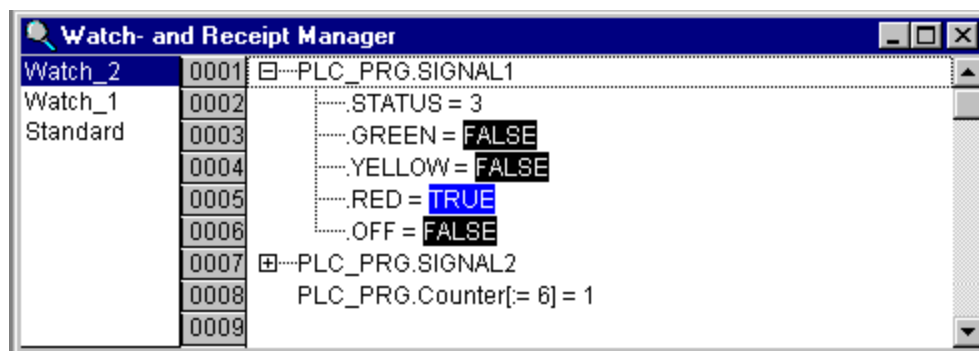
使用该命令，你可以保存监视表。保存文件的对话框被打开。文件名是预设置的监视表名称，其扩展名为“*.wtc”。使用“Extras”（附加）→“Load Watch List”（装入监视表）命令可以重新装入保存的监视表。

“Extras”（附加）→“Load Watch List”（装入监视表）

使用该命令，你可以重新装入保存的监视表。打开文件的对话框被打开。选择所需要的扩展名为“*.wtc”的文件。在出现的对话框中，你可以给监视表命名一个新的名称。该文件名被预置为不带扩展名。使用“Extras”（附加）“Save Watch List”（保存监视表）命令，你可以保存监视表。

联机模式的“Manager in the Online Mode”（监视和接收管理器）

在“Online”（联机）模式，显示的是输入的变量值。结构化变量（数组，结构或功能块实例）在其标识符前面使用一个加号进行标记。通过鼠标点击加号，或按（Enter）键可以展开或闭合该变量。为了输入新的变量，你可以使用“Extras”（附加）“Active Monitoring”（激活监视）命令，关闭显示。在完成输入变量后，你可以再次使用该命令来激活显示值。



联机模式的监视和接收管理器

“Offline Mode”（脱机模式）时，你可以使用常数预置变量（在变量后输入:=<值>）。在“Online Mode”（联机模式）时，使用“Extras”（附加）→“Write Receipt”（写接收）命令，现在可以将这些值写入到变量中去。使用“Extras”（附加）→“Read Receipt”（读接收）命令，你可以使用变量的当前值替换变量的预置值。

注：

这些值只有在“Watch and Receipt Manager”（监视和接收管理器）中将选择的监视表装入时才有效！

“Extra”（附加）→“Monitoring Active”（监视激活）

“Online mode”（联机模式）时，在“Watch and Receipt Manager”（监视和接收管理器）中使用该命令，显示可以被打开或关闭。如果显示是激活的，则在菜单项前面会出现一个对勾（✓）。为了输入新的变量或预置一个值（见“脱机方式”），应通过该命令将显示关闭。当完成输入变量后，你可以再次使用该命令来激活变量值显示。

“Extras”（附加）→ “Write Receipt”（写接收）

“Online Mode”（联机模式）时，在“Watch and Receipt Manager”（监视和接收管理器）中使用这个命令，你可以将预置值（见“脱机方式”）写入到变量中去。

“Extras”（附加）→ “Read Receipt”（读接收）

“Online Mode”（联机模式）时，在“Watch and Receipt Manager”（监视和接收管理器）中使用该命令，你可以使用当前的变量值替换变量的预置值（见“脱机模式”）。

示例：

```
PLC_PRG.Counter [:= <present value>] = <present value>
```

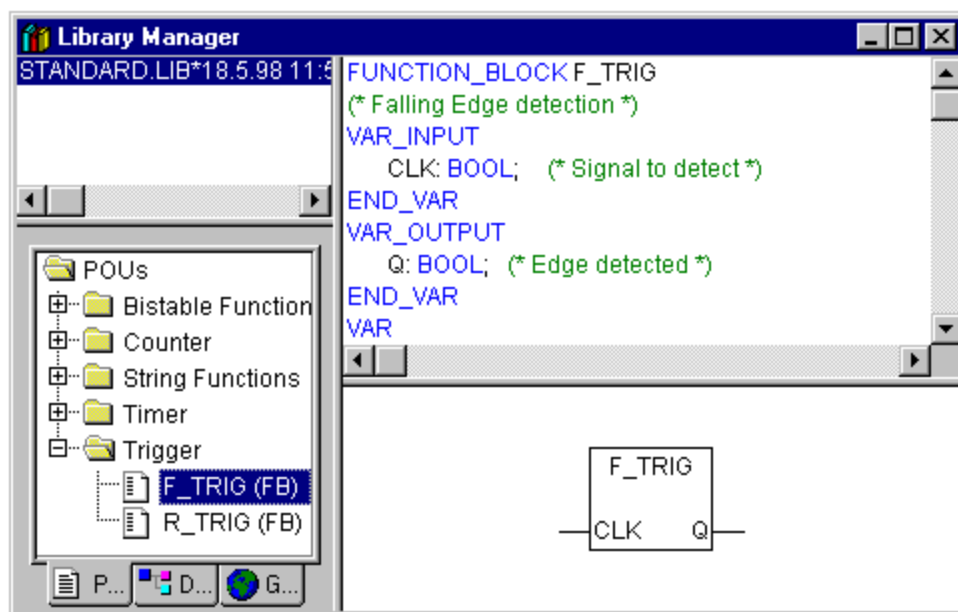
在“Watch and Receipt Manager”（监视和接收管理器）中，你还可以使用“Force values”（强制变量值）和“Write values”（写入变量值）。如果你分别点击变量值，将打开一个对话框，你可以在此框内为变量输入一个新的值。改变后的变量值将在“Watch and Receipt Manager”（监视和接收管理器）中用红色表示。

6 库管理

6.1 库管理器

库管理器显示与当前项目有关的所有库。库的 POU、数据类型和全局变量，都可以像用户定义的 POU、数据类型和全局变量那样以相同的方式使用。

库管理器可以使用“**Window**”（窗口）→“**Library Manager**”（库管理器）命令打开，包括库在内的有关信息和项目一起进行保存。



“Using the Library Manager”（使用库管理器）

库管理器窗口被屏幕分配器分割成三个或四个区域。和项目相关联的库在左上方区域列出。该区域的下面，取决于选择了哪个属性页，这里所列出的是在上面区域中选择的库的 POU、数据类型、或全局变量。通过双击行或按下（Enter）键可以打开和关闭文件夹。在关闭的文件夹前面有一个加号，在打开的文件夹前面有一个减号。如果一个 POU 被鼠标点击或使用方向键而选中，则 POU 的声明将出现在库管理器的右上方区域；而右下方区域则是以图形方式显示的带有输入和输出的黑匣子。数据类型和全局变量的声明在库管理器的右侧区域显示。

“Standard Library”（标准库）

“standard.lib”库总是可以使用的。它包括 IEC 编程系统、IEC61131-3 所要求的、作为标准 POU 的所有功能和功能块。标准功能与操作符的差别在于操作符是被编程系统隐含识别的，而标准 POU 必须依赖于项目（标准 .lib）。

“User-defined Libraries”（用户定义库）

如果一个项目是以实体编译的，而且没有错误，则可以使用“File”（文件）菜单中的“Save as”（另存为）命令，将它保存为一个库。项目本身将保持不变。随后，你可以在项目中输入该名称进行操作，就像使用标准库那样。

“Insert”（插入）→“Additional Library”（附加库）

使用该命令，你可以将一个附加库连接到你的项目中。在打开文件的对话框中，选择所需要的带有 “.lib” 扩展名的库。现在，该库列表在库管理器内，你可以像用户定义的对象那样使用库中的对象。

“Remove Library”（删除库）

使用 “Edit”（编辑）→ “Delete”（删除）命令，你可以从一个项目的库管理器中删除一个库。

“Features”（特性）

该命令将打开 “Informations about internal (resp. external)”（有关内部（相应外部）库信息）对话框。对内部库而言，你可以在那里找到已经插入在 “Project Info”（项目信息）的所有数据。这些数据是在该库创建时建立的。对于外部库而言，将显示库的名称和库的路径。

7 工程接口 (ENI)

7.1 概述

ENI 接口 (“Engineering Interface”) (工程接口) 允许将 “Programming system” (编程系统) 连接到一个外部数据库。在那里可以保存在自动化项目建立过程中所需要的数据。外部数据库的使用保证了数据的一致性, 从而可以将这些数据为多个用户、项目和程序共享。此外, 由于有可能进行以下项目而扩展系统的功能:

- **“Revision control”** (再版控制) 用于项目和相关资源 (共享对象): 如果一个对象在数据库中已经完成检验、修改和重新进行登记, 则在数据库中将创建该对象的一个新版本, 但是, 旧版本仍将保留, 并在需要时可被再次调用。对每个对象和整个项目而言, 版本的历史数据将被记录。两个版本可被检查出其差别。(该特性对于使用本地数据系统而言是不能实现的, 与数据库的使用情况不同。)
- **“Multi-User Operation”** (多用户操作): 对象样本的最新版本, (例如项目的 POU), 可以被一个组的用户操作。因此, 当该对象目前正在被某个用户检验时将被标记为 “in the works” (正在使用中), 其它用户则不能编辑。这样, 多个用户可在相同的项目中并行工作, 而不会有彼此覆盖版本的危险。
- **“Access by external tools”** (外部工具存取): 除了 TwinCAT PLC Control 以外, 其它工具也可以使用 ENI 特性访问公用数据库。例如这些工具可以是: 外部可视化系统, ECAD 系统等, 它们需要使用 TwinCAT PLC Control 产生的数据或者它们自身也生成数据。

ENI 由客户端和服务端部分组成。因此, 将数据库装在能够满足多用户操作需求的一台远程计算机上是可能的。TwinCAT PLC Control 和其它应用程序一样是独立于 ENI 服务器进程的客户端, 它需要访问数据库 (请参阅有关 ENI 服务器的单独文档资料)。

目前, ENI 支持数据库系统 “Visual SourceSafe 5.0”、“Visual SourceSafe 6.0”、“MKS Source Integrity” 和本地文件系统。对象可在那里使用不同的 “folders” (文件夹) (有不同访问权限的数据库类别) 保存。对象可经检验后用于编辑, 从而对其它用户而言是锁定状态。对象的最新版本可从数据库调用。而且, 在并行工作状态时, 你可以像通常无源代码控制的项目那样在项目中保存本地的任何对象。*.pro 文件是由数据库管理的一个项目本地工作拷贝。

与 ENI 项目数据库一起工作的先决条件

如果你想要在 TwinCAT PLC Control 中使用 ENI, 以便管理外部数据中的项目对象, 则必须满足以下一些先决条件:



请注意: 有关安装和使用 ENI 服务器的指南, 请参阅有关的服务器文档和联机帮助。在那里你还可以找到快速入门指南。另外, 还要考虑使用 ENI 浏览器的可能性, 即能够完成数据库的相关功能而又独立于当前使用的数据库系统。

- TwinCAT PLC Control 和 ENI 服务器之间的通讯需要使用 TCP/IP 协议, 这是因为 ENI 服务器使用 HTTP 协议。

- ENI 服务器 (“ENI 服务器套件”) 必须在本地或远程计算机上安装和启动。需要许可证才能使用有标准数据库驱动 (它与服务器一起安装) 的 ENI 服务器。正是由于使用了这个本地文件系统的驱动, 才使无许可证的 ENI 服务器版本能够正常运行。
- 在 ENI 服务器管理工具 (ENI Admin) 中, 必须进行以下配置:
 - 用户必须登录, 并且具备存取权限 (“用户管理”)
 - 必须正确设置有关数据库文件夹的存取权限 (“存取权限”)
 - 建议: 存取 ENIAdmin 和 ENI Control 的管理员密码应在安装后立即指定
- 在 ENI 服务器服务控制工具 (ENI Control) 中, 必须正确配置与所需数据库的连接 (数据库)。在安装过程中, 系统会自动要求你这样做, 但是, 在 ENI Control 中, 你也可以随时修改设置。
- 必须安装可供使用的、支持项目数据库的 ENI 驱动。当 ENI 服务器正在运行时, 这就是为什么要在相同计算机上安装驱动的原因。作为一种选择, 也可以使用本地文件系统, 因为它是系统缺省方式提供的驱动。
- 在数据库管理中, 不管是可能的用户 (客户), 还是 ENI 服务器都必须作为拥有存取权限的有效用户进行登录。无论如何, 这是 “Visual SourceSafe” (虚拟资源安全) 所要求的, 如果你使用的是其它数据库系统, 请参阅有关用户配置信息方面的文档。
- 对当前项目而言, ENI 接口必须激活 (在 TwinCAT PLC Control 中, 使用 “Project” (项目) → “Options” (选项) → “Project data base” (项目数据库) 对话框, 完成这项工作。)
- 对当前项目而言, 必须正确配置数据库连接; 在 TwinCAT PLC Control 中, 使用 “Project” (项目) → “Options” (选项) → “Project data base” (项目数据库) 对话框, 完成这项工作。
- 在当前项目中, 用户必须使用用户名和密码登录 ENI 服务器; 这是在 “login” (登录) 对话框中作出的, 可以使用命令 “Project” (项目) → “Data Base Link” (数据库链接) → “Login” (登录) 显式地打开这个对话框, 相应地, 当你试图进入数据库而事先尚未登录时也会自动打开这个对话框。

与 ENI 项目数据库一起工作

ENI 项目数据库中用于管理项目对象的数据库命令 (“Get Latest Version” (获取最新版), “Check Out” (检验), “Check In” (登记), “Version History” (版本历史), “Label Version” (标记版本) 等), 只要与数据库的连接已经被激活并且正确配置, 就可以在当前项目中使用。为此可参阅与 ENI 项目数据库一起工作的先决条件。该命令在上下文菜单或 “Project” (项目) 菜单的子菜单 “Data Base Link” (数据库链接) 中是可配置的, 它取决于对象管理器中当前标记的对象。当前分配给数据库类别的对象在 “Object Properties” (对象属性) 中显示, 并可在那里修改。数据库类别属性 (通讯参数, 存取权限, 登记/检验行为) 可以在项目数据库的选项 (“Project” (项目) → “Project” (选项) → “Project Source Control” (项目资源控制)) 对话框中修改。

与项目数据库有关的对象类别

有关项目资源控制的 “CoDeSys” 项目被划分为四个对象类别:

- ENI 区分三种在项目数据库中管理的对象类别 (“ENI 对象类别”): “Project objects” (项目对象), “Shared objects” (共享对象), “Compile files” (编译文件)。
- 如果一个对象不必保存到数据库中, 则将它分配到 “Local” (本地) 类别, 这意味着, 它

将被项目作为已知的对象进行处理，不受任何资源控制。

这样，在编程系统中，一个对象可以被分配为以下类别之一：“Project objects”（项目对象）“Shared objects”（共享对象）或“Local”（本地）；“Compile files”（编译文件）尚未作为项目中的对象存在。

在创建对象时自动将一个对象分配给一个类别，因为这是在项目选项对话框“Project source control”（项目资源控制）中定义的，但是，它可以随时在“Object Properties”（对象属性）对话框中修改。

在“ENI 设置”（“Project”（项目）→“Options”（选项）→“category project data base”（项目数据库类别））的设置中，将分别配置每个 ENI 对象类别。这意味着，每个类别都应获取与数据库通讯的特定参数（目录，端口，存取权限，用户存取数据等）以及有关调用最新版本，登记和检验时的特性。这些设置将作用于属于该类别的所有对象。因此，你应该分别登录（用户名，密码）每个类别的数据库；可以通过“Login”（登录）对话框（“Project”（项目）→“data base agssignment”（数据库分配）→“Login”（登录））完成。

在数据库中，为每个类别对象创建单独的文件夹是合理的，但是，在相同的文件夹中保存所有的对象也是可能的。（“category”（类别）是对象的属性，不是文件夹的属性。）

参见以下的三种 ENI 类别对象：

	说明
项目	包含项目指定资源信息的对象，例如，多用户操作共享的 POU。“Get all latest versions”（获取所有最新版本）命令将自动从数据库调用该类别的所有对象到本地项目；甚至是那些到目前为止仍然没有完成的对象也是如此。
共享对象	不是项目指定的对象，例如，POU 库，通常情况下可被多个项目共享。 注意： “Get all Latest Versions”（获得所有最新版本）命令只将该项目文件夹中的这些类别对象复制到本地项目中，这些对象已经是项目的一部分！
编译文件	由 TwinCAT PLC Control 创建的编译信息（例如符号文件），用于当前项目，该信息也许对其它程序也是需要的。 例如：第三方的可视化系统不仅需要项目变量，而且还需要分配的地址。后者只有在项目被编译后，才能知道。



请注意：

作为一种选择，项目的任何对象都可以不包括在项目的资源控制中，可将它们分配到“Local”（本地）类别，即它们可以像通常情况下用于项目的对象那样，随项目进行保存，而无须任何的资源控制。

8 附录

8.1 数据类型

8.1.1 概述

当编程时，你可以使用标准类型和用户定义的数据类型，分配给数据类型的每个标识符将指示保留多少存储器空间和保存什么类型的数值。

注：

不同的数据覆盖不同的数值范围。如果使用的类型转换是从较大类型到较小类型的转换，则可能丢失信息。

标准数据类型

BOOL

BYTE

WORD

DWORD

SINT

USINT

INT

UINT

DINT

UDINT

LINT (64 位整型 TwinCAT 目前不支持)

ULINT (无符号 64 位整型, TwinCAT 目前不支持)

REAL

LREAL

STRING

TIME

TIME_OF_DAY (TOD)

DATE

DATE_AND_TIME (DT)

用户定义的数据类型

ARRAY

POINTER

ENUM (枚举的数据类型)

STRUCT (结构)

ALIAS (派生的数据类型)

子范围数据类型

8.1.2 标准数据类型

8.1.2.1 BOOL

BOOL 类型变量的值为 TRUE 和 FALSE。

类型	占用内存
BOOL	8 位

注:

如果在内存中的最低位被置位（例如 2#00000001），则 BOOL 类型变量为“true”（真）。如果内存中没有被置位的位，则 BOOL 变量为 FALSE（例如 2#00000000）。所有其它值都不能被正确地进行转换，并显示为 (**INVALID: 16#xy**，联机监视时)。类似的问题是可能出现的，例如，在 PLC 程序中使用了重叠的内存范围。

示例:

布尔型变量和字节型变量在内存中占据相同的空间。

```
PROGRAM MAIN
VAR
  bBool AT%MB0      : BOOL;
  nByte AT%MB0      : BYTE := 3;
  bIsTRUE           : BOOL;
END_VAR

IF bBool THEN
  bIsTRUE := TRUE;
ELSE
  bIsTRUE := FALSE;
END_IF
```

程序启动后的联机显示。

```
bBool (%MB0) = INVALID: 16#03
nByte (%MB0) = 3
bIsTRUE = TRUE
```

8.1.2.2 BYTE

整型数据类型。

类型	下界限	上界限	占用内存
BYTE	0	255	8 位

8.1.2.3 WORD

整型数据类型。

类型	下界限	上界限	占用内存
WORD	0	65535	16 位

8.1.2.4 DWORD

整型数据类型。

类型	下界限	上界限	占用内存
DWORD	0	4294967295	32 位

8.1.2.5 SINT

有符号（短）整型数据类型。

类型	下界限	上界限	占用内存
SINT	-128	127	8 位

8.1.2.6 USINT

无符号（短）整型数据类型。

类型	下界限	上界限	占用内存
USINT	0	255	8 位

8.1.2.7 INT

有符号整型数据类型。

类型	下界限	上界限	占用内存
INT	-32768	32767	16 位

8.1.2.8 UINT

无符号整型数据类型。

类型	下界限	上界限	占用内存
UINT	0	65535	16 位

8.1.2.9 DINT

有符号的整型数据类型。

类型	下界限	上界限	使用的保存位
DINT	-2147483648	2147483647	32 位

8.1.2.10 UDINT

无符号整型数据类型。

类型	下界限	上界限	占用内存
UDINT	0	4294967295	32 位

8.1.2.11 REAL

32 位浮点型数据类型。需要表示成有理数。

类型	下界限	上界限	占用内存
REAL	$\sim -3.402823 \times 10^{38}$	$\sim 3.402823 \times 10^{38}$	32 位

8.1.2.12 LREAL

64 位浮点型数据类型。需要表示成有理数。

类型	下界限	上界限	占用内存
LREAL	$\sim -1.79769313486231E308$	$\sim 1.79769313486232E308$	64 位

8.1.2.13 STRING

一个字符串型的变量可以包含任何字符串。在声明中有关其大小的输入项确定了该变量应保留多少内存空间，它对应于字符串中的字符个数，并可置于括弧或方括号内。

有 35 个字符的字符串声明示例：

```
str:STRING(35):='This is a String';
```

类型	占用内存
STRING	<ul style="list-style-type: none"> ● 如果没有指定大小，则使用默认值，即 80 个字符：内存使用[字节数] = 80 + 1 个用于结束字符串的 Null（空）字符； ● 如果指定了大小：内存使用[字节数] = 字符串大小 + 1 个用于结束字符串的 Null（空）字符；

8.1.2.14 TIME

时间以 ms 表示，并在内部作为 DWORD 进行处理。

类型	下界限	上界限	占用内存
TIME	T#0ms	T#71582m47s295ms	32 位

8.1.2.15 TIME_OF_DAY

TOD

一天中的时间，以秒表示，并在内部作为 DWORD 进行处理。

类型	下界限	上界限	占用内存
TIME_OF_DAY TOD	TOD#00:00	TOD#1193:02:47.295	32 位

8.1.2.16 DATE

日期在内部作为 DWORD 进行处理，最高位表示 1 秒。

类型	下界限	上界限	占用内存
DATE	D#1970-01-01	D#2106-02-06	32 位

8.1.2.17 DATE_AND_TIME

DT

日期和时间。最高位表示 1 秒。数据类型在内部作为 DWORD 进行处理

类型	下界限	上界限	占用内存
DATE_AND_TIME DT	DT#1970-01-01-00:00	DT#2106-02-06-06:28:15	32 位

8.1.3. 用户数据类型

8.1.3.1 数组

一维、二维和三维字段（数组）是被作为所支持的基本数据类型。数组可以在 POU 的声明部分和全局变量表中定义。

语法:

```
<Field_Name>:ARRAY [<LowLim1>..UpLim1>, <LowLim2>..UpLim2>] OF <elem. Type>
```

LowLim1, *LowLim2* 标识字段范围的下限; *UpLim1* 和 *UpLim2* 标识字段范围的上限。这些范围值必须是整型数。

示例:

```
Card_game: ARRAY [1..13, 1..4] OF INT;
```

初始化数组

你可以初始化数组中的所有成员，或者全部都不进行初始化。

数组初始化的示例:

```
arr1 : ARRAY [1..5] OF INT := 1,2,3,4,5;
arr2 : ARRAY [1..2,3..4] OF INT := 1,3(7); (*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; (* 0,0,4,4,4,4,2,3 的缩写形式* )
```

结构中的数组初始化示例:

```
TYPE STRUCT1
STRUCT
p1:int;
p2:int;
p3:dword;
END_STRUCT

ARRAY[1..3] OF STRUCT1:= (p1:=1,p2:=10,p3:=4723), (p1:=2,p2:=0,p3:=299),
(p1:=14,p2:=5,p3:=112);
```

数组的部分初始化示例:

```
arr1 : ARRAY [1..10] OF INT := 1,2;
```

没有预置的数组成员，则使用其基本类型的默认初始值进行初始化。在上例中，数组成员 `arr1[3]` 到 `arr1[10]` 均被初始化为 0。

二维数组的数组成员使用以下语法:

```
<Field_Name>[Index1,Index2]
```

示例:

```
Card_game[9,2]
```

注:

如果你在项目中使用 **CheckBounds** 定义了一个功能，则可以自动检查出数组的超范围错误，这个功能名称是固定的，并且只能这样命名。在下面的章节中，你可以找到实现该功能的示例。

8.1.3.2 指针

当程序运行时，变量或功能块的地址保存在指针中，使用下面的语法声明指针：

```
<Identifier>:POINTER TO <Datatype/Functionblock>;
```

指针可以指向任何数据类型，或功能块，甚至是用户定义的数据类型。**Address Operator**（地址运算符）**ADR** 的功能是将一个变量或功能块的地址分配给一个指针。

通过在指针标识符后添加内容运算符 **"^"**，可以提取指针内容值。借助于 **SIZEOF** 运算符可以对指针进行增量运算。

示例:

```
pt:POINTER TO INT;
var_int1:INT := 5;
var_int2:INT;
```

```
pt := ADR(var_int1);
var_int2:= pt^; (* 现在 var_int2 是 5)
```

示例 2（指针增量）：

```
ptByCurrDataOffs : POINTER TO BYTE;
udiAddress        : UDINT;
```

```
(*--- 指针增量 ---*)
udiAddress := ptByCurrDataOffs;
udiAddress := udiAddress + SIZEOF(ptByCurrDataOffs^);
ptByCurrDataOffs := udiAddress;
(* -- 指针增量结束 ---*)
```

8.1.3.3 枚举（ENUM）

枚举是用户定义的数据类型，它由一定数量的字符串常数组成。这些常数作为枚举值。在整个项目中均可识别枚举值，即使它们是在一个 **POU** 中作为本地声明。最好在对象管理器（**Object Organizer**）的数据类型（**Data types**）属性页中，将枚举值作为对象创建。它们以关键字 **TYPE** 开始，并以 **END_TYPE** 结束。

语法:

```
TYPE <Identifier>:(<Enum_0> ,<Enum_1>, ..., <Enum_n>);END_TYPE
```

<Identifier> 可以采用枚举值中的一个，并被初始化为第一个枚举值，这些值和所有数据兼容，也就是说，当你使用这些值完成操作时，就像使用 INT 数据时所完成的操作一样。你可以给 <Identifier> 赋值一个数据 x。如果枚举值还没有进行初始化，则初始化值将从 0 开始依次递增，当初始化时，应该清楚这些初始化值是逐渐增加的。当运行时，将验证这些数据值的有效性。

示例：

```
TRAFFIC_SIGNAL: (Red, Yellow, Green:=10); (* 每种颜色的初始化值是红 0、黄 1、绿 10 *)
TRAFFIC_SIGNAL:=0; (* 交通信号的值是红*)
FOR i:= Red TO Green DO
    i := i + 1;
END_FOR;
```

相同的枚举值不能被多次使用。

示例：

```
TRAFFIC_SIGNAL: (red, yellow, green);
COLOR: (blue, white, red);
```

错误：red 不能同时用于 TRAFFIC_SIGNAL 和 COLOR。

8.1.3.4 结构 (STRUCT)

在对象管理器 (Object Organizer) 的数据类型 (Data Types) 属性页中，结构作为对象进行创建。并使用关键字 TYPE 开始，END_TYPE 结束。结构声明的语法如下：

```
TYPE <Structurename>:
STRUCT
    <Declaration of Variables 1>
    .
    .
    <Declaration of Variables n>
END_STRUCT
END_TYPE
```

<Structurename> (结构名) 是在整个项目中识别的一种类型，并可以像标准数据类型一样使用。允许结构内部连锁。唯一的限制是变量不能放置地址符 (不允许使用 AT 声明！)。

定义 Polygonline (多边形线) 的结构示例：

```
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
```

使用以下语法，可以操作结构成员：

```
<Structure_Name>.<Componentname>
```

例如，如果你有一个命名为“**Week**”的结构，该结构包含一个命名为“**Monday**”的成员，则你可以按照以下方式对其进行操作：**Week.Monday**。

8.1.3.5 参考类型（别名）

你可以使用由用户定义的派生数据类型，为变量、常数或功能块创建一个可供选择的名称。可以在对象管理器的数据类型属性页中，将你的参考类型值作为对象进行创建。该参考类型值以关键字 **TYPE** 开始，以 **END_TYPE** 结束。

语法：

```
TYPE <Identifier>: <Assignment term>;  
END_TYPE
```

示例：

```
TYPE message:STRING[50];  
END_TYPE;
```

8.1.3.6 子范围类型

子范围数据类型，是一种数据类型，其数值范围只是其基本类型的一个子集。它可以在数据类型属性页中进行声明，但一个变量也可以使用子范围数据类型直接声明。

属性页中的“**Data types**”（数据类型）声明语法：

```
TYPE <Name> : <Inttype> (<ug>..<>og>) END_TYPE;
```

类型	说明
<Name>	必须是一个有效的 IEC 标识符
<Inttype>	是数据类型 SINT, USINT, INT, UINT, DINT, UDINT, BYTE, WORD, DWORD (LINT, ULINT, LWORD) 之一。
<ug>	这是一个常数，必须兼容于基本类型，并作为范围类型的下边界。下边界本身包含在基本类型中。
<og>	这是一个常数，必须兼容于基本类型，并作为范围类型的上边界。上边界本身包含在基本类型中。

示例：

```
TYPE
SubInt : INT (-4095..4095);
END_TYPE
```

子范围类型直接声明的变量：

```
VAR
  i1 : INT (-4095..4095);
  i2 : INT (5..10) :=5;
  ui : UINT (0..10000);
END_VAR
```

如果一个常数被赋值为一个子范围类型（在声明或实现中），但这个常数并不在该子范围类型的有效范围内（例如 `1 :=5000`），系统将发布一个错误消息。

为了在运行时检验是否遵守范围的边界，必须引入功能“**CheckRangeSigned**”或“**CheckRangeUnsigned**”。

8.2 操作符

8.2.1 概述

TwinCAT PLC Control 支持所有的 IEC 操作符。与标准功能不同，这些操作符在整个项目内是被隐式识别的。在 POU 中，操作符和功能（function）的使用方法相同。

8.2.2 IEC 操作符概述

下表给出 ST 和 IL 中的操作符，并提供在 IL 中的修饰符。

注意，对 IL 操作符而言：只显示使用了操作符的行。一个先决条件是（首先）要求在先行（例如 LD in）已成功地装载了操作数。IL 的修饰符一栏表示在 IL（指令表）中可能的修饰符：

C	只有先行表达式的结果是 TRUE 时，才能执行这类命令。
N	对于 JMP, CALC 和 RETC 命令而言，只有先行表达式的结果是 FALSE 时，才能执行这类命令。
N	否则：将操作数取反（不适用于累加器）
(操作符包含在括号中：只有在到达闭合括号后，才能执行位于括号前的动作。

关于用法的详细说明，请参阅附录。

操作符 ST	操作符 AWL	修饰符 AWL	含义
'			串定界符（例如 “string1”）
[..]			数组范围的大小（例如 ARRAY[0..3] OF INT）
:			声明时，操作符和类型之间的分界符（例如 var1: INT;）
;			终结指令（例如 a: =var1;）
^			提取指针（例如 pointer1^）
	LD var1	N	在缓冲器中装载 var1 值
:=	ST var1	N	保存实际结果到 var1
	S boolvar		当实际结果是 TRUE 时，准确的设置布尔操作数 boolvar 为 TRUE
	R boolvar		当实际结果是 TRUE 时，准确的设置布尔操作数 boolvar 为 FALSE
	JMP marke	CN	跳转到标号
<Programmname>	CAL prog1	CN	调用程序 prog1
<Instanzname>	CAL inst1	CN	调用功能块实例 inst1
<Fktname>(vx,vy,..)	<Fktname> vx,vy,..	CN	调用功能 fctname 和传送变量 vx, vy

RETURN	RET	CN	离开 POU 并返回到调用程序
	(紧随括号的值作为操作数处理, 在表达式没有由右括号闭合之前, 不执行动作。
)		在右括号闭合后, 现在可执行动作。
AND	AND	N, (按位与 AND
或者	或者	N, (按位或 OR
XOR	XOR	N, (按位异或 XOR
NOT	NOT		按位取反 NOT
+	ADD	(加
-	SUB	(减
*	MUL	(乘
/	DIV	(除
>	GT	(大于
>=	GE	(大于或等于
=	EQ	(等于
<	LT	(小于
<>	NE	(不等于
<=	LE	(小于或等于
in1 MOD in2	MOD		“取模”运算
INDEXOF(in)	INDEXOF		POU in1 的内部索引; [INT]
SIZEOF(in)	SIZEOF		数据类型 in 所需要的字节数
SHL(K,in)	SHL		操作数 in 按位左移 K 次
SHR(in, K)	SHR		操作数 in 按位右移 K 次
ROL(K,in)	ROL		操作数 in 按位循环左移 K 次
ROR(K,in)	ROR		操作数 in 按位循环右移 K 次
SEL(G,in0,in1)	SEL		在2个操作数 in0 (G 是 FALSE) 和 in1 (G 是 TRUE) 之间的二进制选择
MAX(in0,in1)	MAX		返回两个值之间的大者
MIN(in0,in1)	MIN		返回两个值 in0和in1之间的小者
LIMIT(Min,in,Max)	LIMIT		限制值范围(在超过该范围时, in 返回到 MIN 或 MAX)
MUX(K, in0,... in_n)	MUX		从数值组中 (in0 到 in_n) 选出第 K 个值
ADR(in)	ADR		[DWORD]操作数地址
BOOL_TO_<type>(in)	BOOL_TO_<type>		布尔操作数的类型转换
<type>_TO_BOOL(in)	<type>_TO_BOOL		转换为布尔的类型转换
INT_TO_<type>(in)	INT_TO_<type>		INT 操作数, 转换类型为另一个基本类型
REAL_TO_<type>(in)	REAL_TO_<type>		REAL 操作数, 转换类型为另一个基本类型

LREAL_TO_<type>(in)	LREAL_TO_<type>	LREAL 操作数, 转换类型为另一个基本类型
TIME_TO_<type>(in)	TIME_TO_<type>	TIME 操作数, 转换类型为另一个基本类型
TOD_TO_<type>(in)	TOD_TO_<type>	TOD 操作数, 转换类型为另一个基本类型
DATE_TO_<type>(in)	DATE_TO_<type>	DATE 操作数, 转换类型为另一个基本类型
DT_TO_<type>(in)	DT_TO_<type>	DT 操作数, 转换类型为另一个基本类型
STRING_TO_<type>(in)	STRING_TO_<type>	STRING 操作数, 转换类型为另一个基本类型
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 的指数
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	带 expt 的操作数 in 的指数
LEN(in)	LEN	操作数 in 的串长度
LEFT(str, size)	LEFT	串 standard.lib 给定的左起始串
RIGHT(str, size)	RIGHT	串 standard.lib 给定长度的右起始串
MID(str, size)	MID	串 str 中给定长度的部分串
CONCAT(str1, str2)	CONCAT	两个子串 standard.lib 的合并
INSERT(str1, str2, pos)	INSERT	在串 standard.lib 位置插入串 str1
DELETE(str1, len, pos)	DELETE	删去部分串(长度为 len), 于 str1 的 pos standard.lib 位置开始
REPLACE(str1, str2, len, pos)	REPLACE	由 str2 替换部分的串(长度为 len), 在 str1 的 pos 位置开始
FIND(str1, str2)	FIND	在 str1 中搜索串 str2 部分
SR	SR	以稳态功能块, 复位优先
RS	RS	双稳态功能块, 置位优先
SEMA	SEMA	FB:Semaphor 软件(可中断的)
R_TRIG	R_TRIG	FB:检测上升沿
F_TRIG	F_TRIG	FB:检测下降沿

CTU	CTU	FB:加计数
CTD	CTD	FB:减计数
CTUD	CTUD	FB:加/减计数
TP	TP	FB:触发器
TON	TON	FB:接通延迟定时器
TOF	TOF	FB:断开延迟定时器

8.2.3 数值操作符

8.2.3.1 ABS



返回一个数的绝对值。例如 **ABS** (-2) 等于 2。

以下的输入和输出变量类型组合是允许的。

输入

INT

REAL

BYTE

WORD

DWORD

SINT

USINT

UINT

DINT

UDINT

输出

INT, REAL, WORD, DWORD, DINT

REAL

INT, REAL, BYTE, WORD, DWORD, DINT

INT, REAL, WORD, DWORD, DINT

REAL, DWORD, DINT

REAL

REAL

INT, REAL, WORD, DWORD, DINT, UDINT, UINT

REAL, DWORD, DINT

REAL, DWORD, DINT, UDINT

8.2.3.2 ACOS



返回一个数的反余弦（反余弦函数）。

输入可以是类型 **BYTE**, **WORD**, **DWORD**, **INT**, **DINT**, **REAL**, **SINT**, **USINT**, **UINT**, **UDINT**。输出必须是类型 **REAL**。

8.2.3.3 ASIN



返回一个数的反正弦（反正弦函数）

输入可以是类型 BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT。输出必须是类型 REAL。

8.2.3.4 ATAN



返回一个数的反正切（反正切函数）

输入可以是类型 BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT。输出必须是类型 REAL。

8.2.3.5 COS



返回一个数的余弦。

输入可以是类型 BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT。输出必须是类型 REAL。

8.2.3.6 EXP



返回指数函数。

输入可以是类型 BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT。输出必须是类型 REAL。

8.2.3.7 EXPT



一个变量，带另一个变量的指数： $OUT = IN1^{IN2}$ 。

输入 1 和输入 2 可以是类型 BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT，输出必须是类型 REAL。

IL 语言的示例:

```
LD 7
EXPT 2
ST var1 (* Result is 49 *)
```

ST 语言的示例:

```
var1 := EXPT(7,2);
```

8.2.3.8 LN



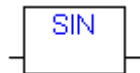
返回一个数的自然对数，输入可以是类型 BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT。输出必须是类型 REAL。

8.2.3.9 LOG



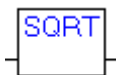
返回一个以 10 为底的对数，输入可以是类型 BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT，输出必须是类型 REAL。

8.2.3.10 SIN



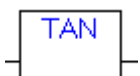
返回一个数的正弦。输入可以是类型 BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT。输出必须是类型 REAL。

8.2.3.11 SQRT



返回一个数的平方根。输入可以是类型 BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT。输出必须是类型 REAL。

8.2.3.12 TAN



返回一个数的正切。输入可以是类型 BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT。输出必须是类型 REAL。

8.2.4 算术操作符

8.2.4.1 ADD

变量类型为: BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL 和 LREAL, 的加法, 两个时间变量也可以相加, 结果是另一个时间变量。

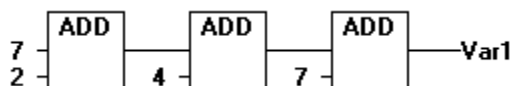
IL 语言的示例:

```
LD 7
ADD 2,4,7
ST var1
```

ST 语言的示例:

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

FBD 语言的示例:



8.2.4.2 MUL

变量类型为: BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL 和 LREAL 的乘法。

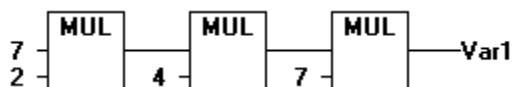
IL 语言的示例:

```
LD 7
MUL 2,4,7
ST var1
```

ST 语言的示例:

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

FBD 语言的示例:



8.2.4.3 SUB

一个变量减去另一个变量的减法, 两者的类型均为: BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL 和 LREAL。亦可以从一个时间变量减去另一个时间变量, 结果是第三个 TIME 类型变量。注意, 负的时间值是没有意义的。

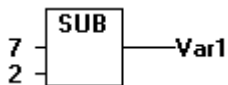
IL 语言的示例:

```
LD 7
SUB 8
ST var1
```

ST 语言的示例:

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

FBD 语言的示例:



8.2.4.4 DIV

一个变量除以另一个变量的除法，两者均为类型：BYTE，WORD，DWORD，SINT，USINT，INT，UINT，DINT，UDINT，REAL 和 LREAL。

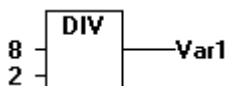
IL 语言的示例:

```
LD 8  
DIV 2  
ST var1
```

ST 语言的示例:

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

FBD 语言的示例:



注:

如果在你的项目中以“CheckDivByte”、“CheckDivWord”、“CheckDivDWordand”和“CheckDivReal”为名定义功能。并且使用操作符 DIV，则可以通过这些功能检验除数值，例如，避免除以 0，功能必须是上述列出的名字。

8.2.4.5 MOD

一个变量以另一个变量为模的模式运算除法，两者的数据类型均为：BYTE，WORD，DWORD，SINT，USINT，INT，UINT，DINT，UDINT，REAL 和 LREAL。这个功能的结果是除法运算的余数。其结果是一个整数。

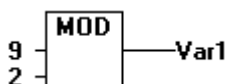
IL 语言的示例:

```
LD 9  
MOD 2  
ST var1 (* Result is 1 *)
```

ST 语言的示例:

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

FBD 语言的示例:



8.2.5 位串操作符

8.2.5.1 AND

按位“与”操作。操作数必须是类型 **BOOL**, **BYTE**, **WORD** 或 **DWORD**。

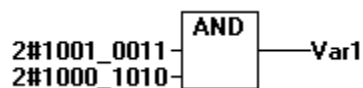
IL 语言的示例:

```
var1 :BYTE;  
LD 2#1001_0011  
AND 2#1000_1010  
ST var1 (* Result is 2#1000_0010 *)
```

ST 语言的示例:

```
var1 := 2#1001_0011 AND 2#1000_1010
```

FBD 语言的示例:



8.2.5.2 OR

按位“OR”（或）操作。操作数必须是类型 **BOOL**, **BYTE**, **WORD** 或 **DWORD**。

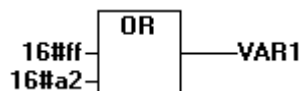
IL 语言的示例:

```
var1 :BYTE;  
LD 2#1001_0011  
OR 2#1000_1010  
ST var1 (* Result is 2#1001_1011 *)
```

ST 语言的示例:

```
Var1 := 2#1001_0011 OR 2#1000_1010
```

FBD 语言的示例:



8.2.5.3 XOR

按位“XOR”（异或）操作。操作数必须是类型 **BOOL**, **BYTE**, **WORD** 或 **DWORD**。

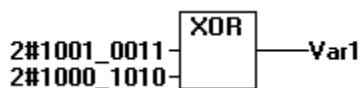
IL 语言的示例:

```
Var1 :BYTE;  
LD 2#1001_0011  
XOR 2#1000_1010  
ST Var1 (* Result is 2#0001_1001 *)
```

ST 语言的示例:

```
Var1 := 2#1001_0011 XOR 2#1000_1010
```

FBD 语言的示例:



注:

请注意, 扩展形式 XOR POU (输入多于 2 个)没有采用标准的形式实现。输入成对地进行“异或”→“运算”, 然后相应的结果彼此进行比较。

8.2.5.4 NOT

按位“NOT”（非）操作。操作数必须是类型 BOOL, BYTE, WORD 或 DWORD。

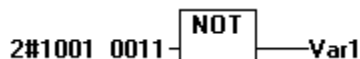
IL 语言的示例:

```
Var1 :BYTE;
LD 2#1001_0011
NOT
ST Var1 (* Result is 2#0110_1100 *)
```

ST 语言的示例:

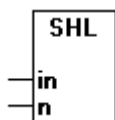
```
Var1 := NOT 2#1001_0011;
```

FBD 语言的示例:



8.2.6 位移操作符

8.2.6.1 SHL



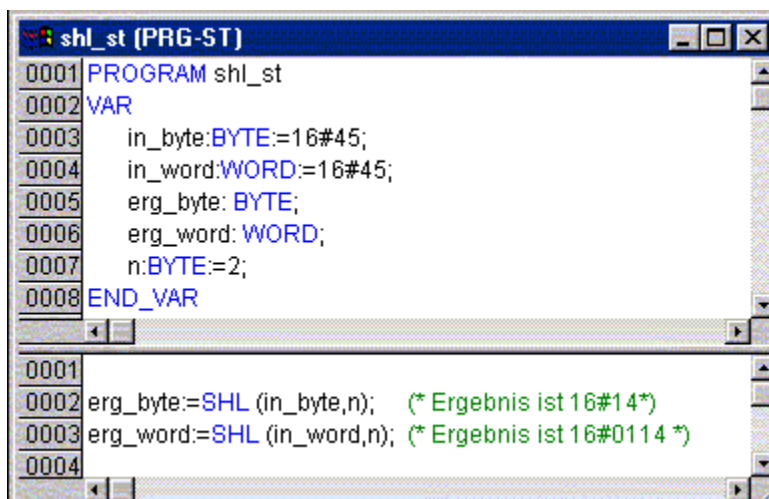
操作数的按位左移: $A := SHL(IN, N)A$, 其中 IN 和 N 必须是类型 BYTE, WORD, 或 DWORD。IN 左移以 0 填充。

注:

- TwinCAT 2.7 Build < 519 和 TwinCAT 2.8 Build < 735.: 用于 C16x (BC-控制器)目标系统类型的代码生成器执行 (IN, N = 模 16) 算术移位运算。
- 请注意, 关于算术运算位的数量实际上和输入变量的数据类型有关! 如输入变量是一个常数, 应考虑可能的最小数据类型, 输出变量对算术运算绝不会有影响。

参阅以下 16 进制的示例, 对 erg_byte 和 erg_word 而言, 其运算结果是不同的, 这取决于输入变量的数据类型 (BYTE 或 WORD), 虽然输入变量 in_byte 和 in_word 的值是相同的。

ST 语言的示例:



```

0001 PROGRAM shl_st
0002 VAR
0003   in_byte:BYTE:=16#45;
0004   in_word:WORD:=16#45;
0005   erg_byte: BYTE;
0006   erg_word: WORD;
0007   n:BYTE:=2;
0008 END_VAR

0001
0002 erg_byte:=SHL (in_byte,n); (* Ergebnis ist 16#14*)
0003 erg_word:=SHL (in_word,n); (* Ergebnis ist 16#0114 *)
0004

```

IL 语言的示例:

```

LD 1
SHL 1
ST Var1 (* Result is 2 *)

```

8.2.6.2 SHR

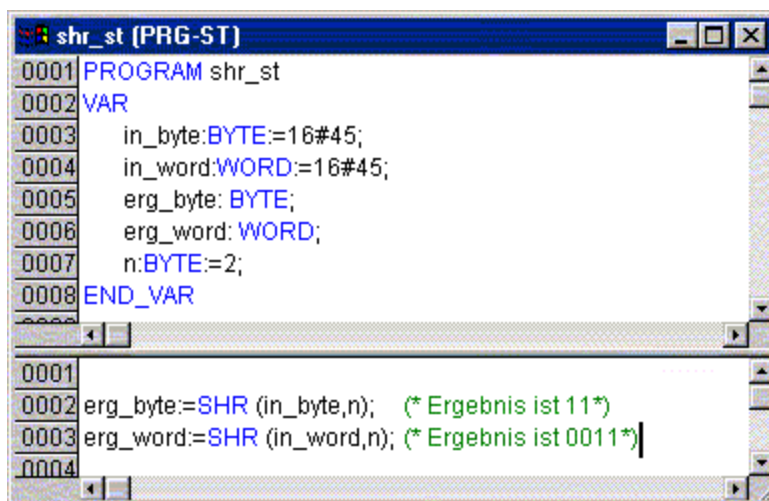


操作数的按位右移: $A := \text{SHR}(\text{IN}, N)A$, 其中 IN 和 N 必须是类型 BYTE, WORD 或 DWORD。IN 右移 N 位, 其左边以 0 填充。

注:

- TwinCAT 2.7 Build < 519 和 TwinCAT 2.8 Build < 735: 用于 C16x (BC-控制器)目标系统类型的代码生成器执行 (IN, N = 模 16) 算术移位运算。
- 请注意, 关于算术运算位的数量实际上和输入变量的数据类型有关! 如输入变量是一个常数, 应考虑可能的最小数据类型, 输出变量对算术运算绝不会有影响。

ST 语言的示例:



```

shr_st (PRG-ST)
0001 PROGRAM shr_st
0002 VAR
0003   in_byte:BYTE:=16#45;
0004   in_word:WORD:=16#45;
0005   erg_byte: BYTE;
0006   erg_word: WORD;
0007   n:BYTE:=2;
0008 END_VAR

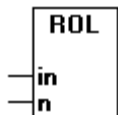
0001
0002 erg_byte:=SHR (in_byte,n); (* Ergebnis ist 11*)
0003 erg_word:=SHR (in_word,n); (* Ergebnis ist 0011*)
0004

```

IL 语言的示例:

```
LD 32
SHR 2
ST Var1 (* Result is 8 *)
```

8.2.6.3 ROL



操作数按位循环左移: $A := \text{ROL}(\text{IN}, N)A$, 其中 IN 和 N 必须是类型 BYTE, WORD 或 DWORD。IN 按位左移 N 次而最左位再从右边插入。

注:

- 请注意, 关于算术运算位的数量实际上和输入变量的数据类型有关! 如输入变量是一个常数, 应考虑可能的最小数据类型, 输出变量对算术运算绝不会有影响。

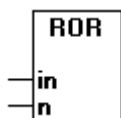
ST 语言的示例:

```
rol_st (PRG-ST)
0001 PROGRAM rol_st
0002 VAR
0003   in_byte:BYTE:=16#45;
0004   in_word:WORD:=16#45;
0005   erg_byte: BYTE;
0006   erg_word: WORD;
0007   n:BYTE:=2;
0008 END_VAR
0002 erg_byte:= ROL (in_byte,n); (* Ergebnis ist 16#15 *)
0003 erg_word:=ROL (in_word,n); (* Ergebnis ist 16#0114 *)
```

IL 语言的示例:

```
Var1 :BYTE;
LD 2#1001_0011
ROL 3
ST Var1 (* 结果是 2#1001_1100 *)
```

8.2.6.4 ROR



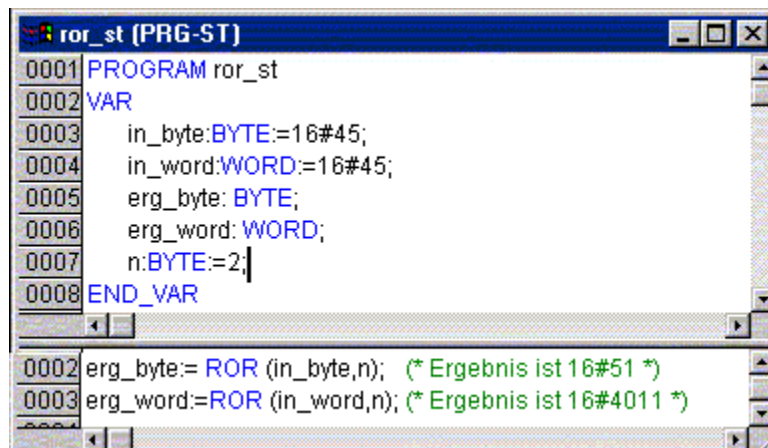
操作数按位循环右移: $A := \text{ROR}(\text{IN}, N)A$, 其中 IN 和 N 必须是类型 BYTE, WORD 或 DWORD。IN 按位右移 N 次, 而最右位再从左边插入。

注:

请注意, 关于算术运算位的数量实际上和输入变量的数据类型有关! 如输入变量是一个常数, 应考虑可能的最小数据类型, 输出变量对算术运算绝不会有影响。

参阅以下 16 进制的示例，对 `erg_byte` 和 `erg_word` 而言，其运算结果是不同的，这取决于输入变量的数据类型，（`BYTE` 或 `WORD`），虽然输入变量 `in_byte` 和 `in_word` 的值是相同的。

ST 语言的示例：



```
0001 PROGRAM ror_st
0002 VAR
0003     in_byte:BYTE:=16#45;
0004     in_word:WORD:=16#45;
0005     erg_byte: BYTE;
0006     erg_word: WORD;
0007     n:BYTE:=2;
0008 END_VAR

0002 erg_byte:=ROR(in_byte,n); (* Ergebnis ist 16#51 *)
0003 erg_word:=ROR(in_word,n); (* Ergebnis ist 16#4011 *)
```

IL 语言的示例：

```
Var1 :BYTE;
LD 2#1001_0011
ROR 3
ST Var1 (* 结果是 2#0111_0010 *)
```

8.2.7 选择操作符

8.2.7.1 SEL

二值选择。

```
OUT := SEL(G, IN0, IN1)
```

含义:

```
OUT := IN0 if G=FALSE;OUT := IN1 if G=TRUE.
```

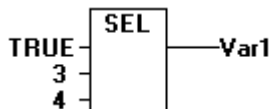
IN0、IN1 和 OUT 可以是任何类型的变量,G 必须是类型 BOOL。如 G 是 FALSE,则选择的结果是 IN0;如果 G 是 TRUE,则选择的结果是 IN1。

IL 语言的示例:

```
LD TRUE
SEL 3,4
ST Var1 (* 结果是 4 *)

LD FALSE
SEL 3,4
ST Var1 (* 结果是 3 *)
```

FBD 语言的示例:



注:

为了优化运行之目的,可进行如下处理:附加在 IN0 上的表达式,只有当 G 是 FALSE 时才进行计算。附加在 IN1 上的表达式,只有当 G 是 TRUE 时才进行计算。

8.2.7.2 MAX

最大值功能 返回两个数值中较大的数。

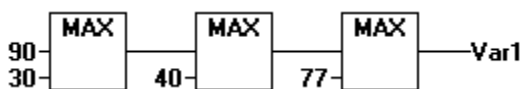
```
OUT := MAX(IN0, IN1)
```

IN0, IN1 和 OUT 可以是任何类型的变量。

IL 语言的示例:

```
LD 90
MAX 30
MAX 40
MAX 77
ST Var1 (* 结果是 90 *)
```

FBD 语言的示例:



8.2.7.3 MIN

最小值功能。返回两个数值中较小的数。

```
OUT := MIN(IN0, IN1)
```

IN0, IN1 和 OUT 可以是任何类型的变量。

IL 语言的示例:

```
LD 90
MIN 30
MIN 40
MIN 77
ST Var1 (* 结果是 30 *)
```

FBD 语言的示例:



8.2.7.4 LIMIT

极限值

```
OUT := LIMIT(Min, IN, Max)
```

含义:

```
OUT := MIN (MAX (IN, Min), Max)
```

对结果而言, 上限位是 Max, 下限位是 Min。IN 的值超过 Max, 则返回 Max。IN 的值低于 Min, 则返回 Min。

IN 和 OUT 可以是任何类型的变量。

IL 语言的示例:

```
LD 90
LIMIT 30,80
ST Var1 (* 结果是 80 *)
```

8.2.7.5 MUX

多值选择器

```
OUT := MUX(K, IN0, ..., INn)
```

含义:

```
OUT := INK.
```

IN0, ..., INn 和 OUT 可以是任何类型的变量。K 必须是 BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT 或 UDINT。MUX 从一组数值中选择第 K 个值。

IL 语言的示例:

```
LD 0
MUX 30,40,50,60,70,80
ST Var1 (* 结果是 30 *)
```

注:

注意: 一个表达式是在一个输入之前而不是在 INK 之前发生变化, 并且该处理过程不会节省运行时间。

8.2.8 比较操作符

8.2.8.1 GT

大于

当第一个操作数的值大于第二个操作数的值时，则布尔操作符返回值 **TRUE**。操作数可以是 **BOOL, BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME** 和 **STRING**。

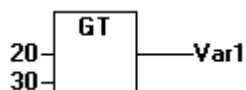
IL 语言的示例:

```
LD 20
GT 30
ST Var1 (* 结果是 FALSE *)
```

ST 语言的示例:

```
VAR1 := 20 > 30 > 40 > 50 > 60 > 70;
```

FBD 语言的示例:



8.2.8.2 LT

小于

当第一个操作数的值小于第二个操作数的值时，则布尔操作符返回值 **TRUE**。操作数可以是 **BOOL, BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME** 和 **STRING**。

IL 语言的示例:

```
LD 20
LT 30
ST Var1 (* 结果是 TRUE *)
```

ST 语言的示例:

```
VAR1 := 20 < 30;
```

FBD 语言的示例:



8.2.8.3 LE

小于或等于

当第一个操作数的值小于或等于第二个操作数的值时，则布尔操作符返回值 **TRUE**。操作数可以是 **BOOL, BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME** 和 **STRING**。

IL 语言的示例:

```
LD 20
LE 30
ST Var1 (* 结果是 TRUE *)
```

ST 语言的示例:

```
VAR1 := 20 <= 30;
```

FBD 语言的示例:



8.2.8.4 GE

大于或等于

当第一个操作数的值大于或等于第二个操作数的值, 则布尔操作符返回值 TRUE。操作数可以是 BOOL, BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME 和 STRING。

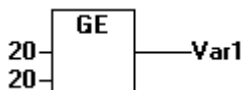
IL 语言的示例:

```
LD 60
GE 40
ST Var1 (* 结果是 TRUE *)
```

ST 语言的示例:

```
VAR1 := 60 >= 40;
```

FBD 语言的示例:



8.2.8.5 EQ

等于

当第一个操作数的值等于第二个操作数的值, 则布尔操作符返回值 TRUE。操作数可以是 BOOL, BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME 和 STRING。

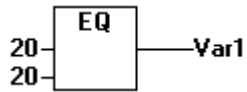
IL 语言的示例:

```
LD 40
EQ 40
ST Var1 (* 结果是 TRUE *)
```

ST 语言的示例:

```
VAR1 := 40;
```

FBD 语言的示例:



8.2.8.6 NE

不等于

当操作数不相等时，则布尔操作符返回值 TRUE。操作数可以是 BOOL, BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME 和 STRING。

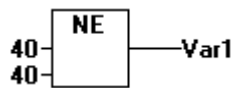
IL 语言的示例:

```
LD 40
NE 40
ST Var1 (* 结果是 FALSE *)
```

ST 语言的示例:

```
VAR1 := 40 <> 40;
```

FBD 语言的示例:



8.2.9 选择不同的操作符

8.2.9.1 INDEXOF

执行这个功能以找出用于一个 POU 的内部索引。

ST 语言的示例:

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

8.2.9.2 SIZEOF

执行这个功能以确定给出的数据类型所需要的字节数量。

IL 语言的示例:

```
arr1:ARRAY[0..4] OF INT;  
var1:INT;  
LD arr1  
SIZEOF  
ST var1 (* 结果是 10 *)
```

8.2.9.3 ADR (地址操作符)

地址功能

ADR 在一个 DWORD 中返回其变量的地址。这个地址可以被传送到与制造商相关的功能中，并作为一个指针进行处理，或者它可以分配给项目中的一个指针。

IL 语言的示例:

```
LD var1  
ADR  
ST var2
```

8.2.9.4 ^ (内容操作符)

通过在指针标识符后添加内容操作符 "^"，则可以提取指针内容。

ST 语言的示例:

```
pt:POINTER TO INT;  
var_int1:INT;  
var_int2:INT;  
pt := ADR(var_int1);  
var_int2:=pt^;
```

8.2.9.5 CAL (调用操作符)

调用一个功能块

在 IL 语言中通过 CAL 调用一个功能块实例。输入变量应放在功能块实例名称右边的圆括号内。

示例: 一个功能块的调用实例 Inst, 输入变量 Par1 和 Par2 分别是 0 和 TRUE。

```
CAL INST(PAR1 := 0, PAR2 := TRUE)
```

8.2.9.6 BITADR

BITADR 返回分配变量的位地址。

ST 语言的示例:

```
bOFF AT%X10.1 : BOOL;
```

```
iBitAdr :BYTE ;
```

```
iBitAdr := BITADR( bOFF ); (* 返回 81 *)
```

8.2.10 类型转换操作符

8.2.10.1 BOOL_TO 转换

从 **BOOL** 类型转换为其它类型的变量:对于数值型变量而言,当操作数是 **TRUE** 时结果是 1,当操作数是 **FALSE** 时结果是 0。如果是 **STRING** 类型变量,结果分则是"TRUE" 或 "FALSE"。

ST 语言的示例:

```
i:=BOOL_TO_INT(TRUE); (* 结果是 1 *)
str:=BOOL_TO_STRING(TRUE); (* 结果是 'TRUE' *)
t:=BOOL_TO_TIME(TRUE); (* 结果是 T#1ms *)
tof:=BOOL_TO_TOD(TRUE); (* 结果是 TOD#00:00:00.001 *)
dat:=BOOL_TO_DATE(FALSE); (* 结果是 D#1970-01-01 *)
dandt:=BOOL_TO_DT(TRUE); (* 结果是 DT#1970-01-01-00:00:01 *)
```

8.2.10.2 TO_BOOL 转换

从另一个变量类型到布尔变量类型的转换:当操作数不等于 0 时,结果是 **TRUE**。操作数等于 0 时,结果是 **FALSE**。对字符串类型,当操作数是 "TRUE" 时,结果是 **TRUE**,否则结果是 **FALSE**。

ST 语言的示例:

```
b := BYTE_TO_BOOL(2#11010101); (* 结果是 TRUE *)
b := INT_TO_BOOL(0); (* 结果是 FALSE *)
b := TIME_TO_BOOL(T#5ms); (* 结果是 TRUE *)
b := STRING_TO_BOOL('TRUE'); (* 结果是 TRUE *)
```

8.2.10.3 STRING_TO 转换

从变量类型 **STRING** 转换为一个不同的变量类型:该变量必须在目标变量类型中包含一个有效的值,否则其结果是 0。

ST 语言的示例:

```
b :=STRING_TO_BOOL('TRUE'); (* 结果是 TRUE *)
w :=STRING_TO_WORD('abc34'); (* 结果是 0 *)
t :=STRING_TO_TIME('T#127ms'); (* 结果是 T#127ms *)
```

8.2.10.4 TO_STRING 转换

从不同变量类型转换为 **STRING**。

ST 语言的示例:

```
str:=BOOL_TO_STRING(TRUE); (* 结果是 'TRUE' *)
str :=TIME_TO_STRING(T#12ms); (* 结果是 'T#12ms' *)
str :=DATE_TO_STRING(D#2002-08-18); (* 结果是 'D#2002-08-18' *)
str:=TOD_TO_STRING(TOD#14:01:05.123); (* 结果是 'TOD#14:01:05.123' *)
str:=DT_TO_STRING(DT#1998-02-13-14:20); (* 结果是 'DT#1998-02-13-14:20' *)
k := LREAL_TO_STRING(1.4); (* 结果是 '1.4' *)
```

8.2.10.5 TIME_TO 转换

从变量类型 **TIME** 转换为一个不同的变量类型：时间在内部保存为 **DWORD** 格式，单位[ms]。这个值将被转换。当你将一个较大的类型转换为一个较小的类型时，则有丢失部分信息的风险。对 **STRING** 类型变量而言，结果是一个时间常数。

ST 语言的示例：

```
str :=TIME_TO_STRING(T#12ms); (* 结果是 'T#12ms' *)
dw:=TIME_TO_DWORD(T#5m); (* 结果是 300000 *)
```

8.2.10.6 DATE_TO 转换

从 **DATE** 变量类型转换为不同的变量类型：日期在内部保存为 **DWORD** 格式，单位为秒，其起始值是 1970 年 1 月 1 日。当从一个较大的类型转换为一个较小的类型时，则有丢失部分信息的风险。对 **STRING** 类型变量而言，结果是一个日期常数。

ST 语言的示例：

```
b :=DATE_TO_BOOL(D#1970-01-01); (* 结果是 FALSE *)
i :=DATE_TO_INT(D#1970-01-15); (* 结果是 29952 *)
str :=DATE_TO_STRING(D#2002-08-18); (* 结果是 'D#2002-08-18' *)
vdt:=DATE_TO_DT(D#2002-08-18); (* 结果是 DT#2002-08-18-00:00 *)
udw:=DATE_TO_DWORD(D#2002-08-18); (* 结果是 16#3D5EE380 *)
```

8.2.10.7 TOD_TO 转换

从 **TIME_OF_DAY** 变量类型转换为不同的变量类型：时间在内部保存为 **DWORD** 格式，单位为毫秒（**TIME_OF_DAY** 变量的计时从上午 12:00 开始）。这个值将被转换。当从一个较大的类型转换为一个较小的类型时，则有丢失部分信息的风险。对 **STRING** 类型变量而言，结果是一个时间常数。

ST 语言的示例：

```
si:=TOD_TO_SINT(TOD#00:00:00.012); (* 结果是 12 *)
str:=TOD_TO_STRING(TOD#14:01:05.123); (* 结果是 'TOD#14:01:05.123' *)
tm:= TOD_TO_TIME(TOD#14:01:05.123); (* 结果是 T#841m5s123ms *)
udi:= TOD_TO_UDINT(TOD#14:01:05.123); (* 结果是 16#03020963 *)
```

8.2.10.8 “DT_TO 转换

从 **DATE_AND_TIME** 变量类型转换为不同的变量类型：变量在内部保存为 **DWORD** 格式，单位为秒，其起始值是 1970 年 1 月 1 日。当从一个较大的类型转换为一个较小的类型时，则有丢失部分信息的风险。对 **STRING** 类型变量而言，结果是一个 **DATE_AND_TIME** 常数。

ST 语言的示例：

```
byt :=DT_TO_BYTE(DT#1970-01-15-05:05:05); (* 结果是 129 *)
str:=DT_TO_STRING(DT#1998-02-13-14:20); (* 结果是 'DT#1998-02-13-14:20' *)
vtod:=DT_TO_TOD(DT#1998-02-13-14:20); (* 结果是 TOD#14:20 *)
vdate:=DT_TO_DATE(DT#1998-02-13-14:20); (* 结果是 D#1998-02-13 *)
vdw:=DT_TO_DWORD(DT#1998-02-13-14:20); (* 结果是 16#34E45690 *)
```

8.2.10.9 REAL_TO-/LREAL_TO 转换

从 REAL 或 LREAL 变量类型转换为不同的变量类型:

这个值将被四舍五入到最接近的整数并转换为新的变量类型。但对于变量类型 STRING, BOOL, REAL 和 LREAL 则例外。当从一个较大的类型转换为一个较小的类型时, 则有丢失部分信息的风险。

请注意在转换为 STRING 类型时全部数据个数只限于 16 个。假如 (L) REAL 类型数据有更多个数, 则第 16 个数将被四舍五入。假如这个 STRING 类型长度定义的较短, 它将从右端开始切除。

ST 语言的示例:

```
i := REAL_TO_INT(1.5); (* 结果是 2 *)
j := REAL_TO_INT(1.4); (* 结果是 1 *)
k := LREAL_TO_STRING(); (* 结果是 '1.4' *)
```

IL 语言的示例:

```
LD 2.7
REAL_TO_INT
GE %MW8
```

8.2.10.10 整型数据类型之间的转换

从一种整型数据类型转换为另一种数据类型: 当从一个较大的类型转换为一个较小的类型时, 则有丢失部分信息的风险。假如你需要转换的数据超过了限制范围, 则该数据的第一个字节将被忽略。

ST 语言的示例:

```
si := INT_TO_SINT(4223); (* 结果是 127 *)
```

假如你将整数 4223(在 16 进制中表示为 16#107f)作为一个 SINT 类型变量进行保存, 它将显示为 127 (在 16 进制中表示为 16#7f)。

IL 语言的示例:

```
LD 2
INT_TO_REAL
MUL 3.5
```

8.2.10.11 TRUNC



从 REAL 类型转换为 INT 类型。只采用该数据的整数部分。当从一个较大的类型转换为一个较小的类型时, 则有丢失部分信息的风险。

ST 语言的示例:

```
i:=TRUNC(1.9); (* 结果是 1 *).
```

```
i:=TRUNC(-1.4); (* 结果是 -1 *).
```

IL 语言的示例:

```
LD 2.7
TRUNC
GE %MW8
```

8.3 操作数

8.3.1 常数

8.3.1.1 BOOL 常数

BOOL 常数为逻辑值 TRUE（真）和逻辑值 FALSE（假）。

8.3.1.2 TIME 常数

TIME 常数可以在 TwinCAT PLC Control 中声明。这些常数一般用于标准库的定时器操作。一个 TIME 常数总是由一个起始符“t”或“T”（或者用全部拼出来的“time”或“TIME”）和一个数字标识符“#”组成。然后，是跟随的实际时间声明，包括天数（“d”标识）、小时（“h”标识）、分钟（“m”标识）、秒（“s”标识）和毫秒（“ms”标识）。请注意时间各项必须根据时间长度单位顺序进行设置（d 在 h 之前，h 在 m 之前，m 在 s 之前，s 在 ms 之前），但无须包含所有的时间长度单位。

在 ST 语言的赋值语句中，正确使用时间常数的示例：

```
TIME1 := T#14ms;  
TIME1 := T#100S12ms; (*最高单位的值可以超出其限制*)  
TIME1 := t#12h34m15s;
```

下面的示例不正确：

```
TIME1 := t#5m68s; (*最低单位的值溢出*)  
TIME1 := 15ms; (*没有 T# *)  
TIME1 := t#4ms13d; (*输入的顺序错误*)
```

8.3.1.3 DATE 常数

这些常数可以用来输入日期。声明一个 DATE 常数时，起始符为“d”，“D”，“DATE”或“date”后跟随一个“#”号。然后你就可以按照年-月-日的格式输入任何日期。

示例：

```
DATE#1996-05-06  
d#1972-03-29
```

8.3.1.4 TIME_OF_DAY 常数

使用这种类型常数可以保存一天中的不同时间。一个 TIME_OF_DAY 常数的声明使用起始符“tod#”，“TOD#”，“TIME_OF_DAY#”或“time_of_day#”，后跟随一个时间格式为：小时：分：秒的时间。秒值可以是实数也可以是小数。

示例：

```
TIME_OF_DAY#15:36:30.123  
tod#00:00:00
```

8.3.1.5 DATE_AND_TIME 常数

日期常数和一天中的时间常数可以合并起来构成一个所谓的 DATE_AND_TIME 常数。DATE_AND_TIME 常数的起始符为“dt#”，“DT#”，“DATE_AND_TIME#”或“date_and_time#”。在日期之后用 (-) 字符连接时间。

示例:

```
DATE_AND_TIME#1996-05-06-15:36:30
dt#1972-03-29-00:00:00
```

8.3.1.6 数值常数

数值可以用二进制数、八进制数、十进制数和十六进制数表示。假如一个整数不是十进制数，你必须在该整数常数之前写出它的基数并加上数字符号(#)。在十六进制中，数值 10-15 通常由字母 A-F 表示。你可以在数值中使用下划线。

示例:

```
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)
```

这些数值可以是变量类型 BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL 或 LREAL。不允许隐式地将较大的变量类型转换为较小的变量类型。这就意味着，一个 DINT 变量不能简单地作为 INT 变量。你必须使用类型转换功能才能做到这种转换。

8.3.1.7 REAL/LREAL 常数

REAL 和 LREAL 常数可以使用十进制小数和指数形式表示。使用带小数点的美国格式表示实数 (REAL/LREAL)。

示例:

```
7.4 取代 7,4
1.64e+009 取代 1,64e+009
```

8.3.1.8 STRING 常数

一个字符串是一个字符队列。STRING (字符串) 常数使用一个单引号作为其前缀和后缀。也可以输入空格和专用字符 (例如音符)。这些字符将同所有其它字符一样进行处理。在字符对列中，美元符号 (\$) 和后面跟随的两个十六进制数的组合被解释为八位字符码的十六进制表示。此外，以美元符号作为起始符的两个字符的组合，其含义如下表所示，并以字母的顺序排列:

字符	说明
\$\$	美元符号
\$'	单引号
\$L 或 \$l	换行
\$N 或 \$n	新行
\$P 或 \$p	换页
\$R 或 \$r	行中止

\$T or \$t

制表符

示例:

```
'w1WU??'  
'Susi und Claus'  
' :-)'
```

8.3.1.9 类型符

通常, 对 IEC 常数, 有可能使用最小的数据类型。如果必需使用另一种数据类型。则可借助于类型符而不需要显式地声明常数。为此, 常数可以使用一个前缀表示, 该前缀决定了其类型。

其格式为: <Type>#<Literal>

<Type> 指定所要求的数据类型, 可能的数据类型是: BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, LREAL。类型必须使用大写字母。

<Literal>指定常数。输入的数据必须与<Type>下指定的数据类型相匹配。

示例:

```
var1:=DINT#34;
```

如果常数不能保证在不丢失数据的情况下转换为目标类型, 系统将发出一个出错消息。

类型符可以用于一般常数。

8.3.2 变量

8.3.2.1 变量

变量可以在一个 POU 的声明部分作为本地变量进行声明或在全局变量列表中进行声明。变量标识符不能包含空格或专用字符。同一变量不能进行多次声明，也不能与任何关键字同名。不区分大小写，也就是说，VAR1、Var1 和 var1 表示相同的变量。下划线字符可以用来区分不同的标识符（例如，“A_BCD”和“AB_CD”被认为是两个不同的标识符）。不能在一行标识符中多次使用下划线字符。标识符的前 32 个字符是有意义的。变量可以在声明类型允许它们的任何地方使用。通过输入助手（Input Assistant）可以操作已有的变量。

8.3.2.2 地址

通过使用专用的字符序列，可以直接显示不同的内存定位。这个序列采用百分号 "%", 一个范围前缀，一个大小范围前缀，以及一个或多个由空格分隔的自然数组合。支持以下的范围前缀：

前缀	说明
I	输入
Q	输出
M	内存定位

支持以下大小范围的前缀：

前缀	说明
X	单个位
无	单个位
B	字节（8 位）
W	字（16 位）
D	双字（32 位）

示例：

%QX75.1 %Q75 输出位 75

%IW215 输入字 215

%QB7 输出字节 7

%MD48 内存定位中的第 48 个双字的位置

注：

如果没有显式地指定单个位的地址，则布尔值将按字节定位。示例：变量值 varbool1 AT %QW0 的变化，将影响 QX0.0 到 QX0.7。

内存定位

你可以使用任何支持的大小范围来访问内存定位。例如，地址 `%MD48` 对应内存定位区中的地址字节为 192, 193, 194, 和 195 ($48 * 4 = 192$)，第 1 个字节的数是 0。你可以使用相同的方法访问字，字节甚至是位：地址 `%MX5.0` 允许你访问第 5 个字的第 1 位（位通常使用字格式进行保存）。

8.3.2.3 存取数组、结构和 POU 变量

二维数组的成员可以使用以下语法进行存取：

```
<Fieldname>[Index1, Index2]
```

结构变量可以使用以下语法进行存取：

```
<Structurename>.<Variablename>
```

功能块和程序变量可以使用以下语法进行存取：

```
<Functionblockname>.<Variablename>
```

8.3.2.4 变量的位寻址

对整型变量，可以存取其中的每一个位。为此，可以在这个变量后添加将要处理的位索引，并使用一个点号 (.) 分隔。位索引可以用任何常数表示。索引使用 0 为其基值。

示例：

```
a : INT;  
b : BOOL;  
...  
a.2 := b;
```

变量 **a** 的第三位设定为变量 **b** 的值。

如果索引值大于变量的位宽度，系统发出以下的出错信息：索引 '`<n>`' 在变量 '`<var>`' 的有效范围之外！

以下变量类型可以进行位寻址：SINT, INT, DINT, USINT, UINT, UDINT, BYTE, WORD, DWORD。

如果变量类型不允许，系统发出以下的出错信息：“用于直接索引的 '`<type>`' 是无效的数据类型”

不能将位存取分配给一个 VAR_IN_OUT 变量。

8.3.2.5 功能

在 ST 语言中，功能调用可以作为操作数出现。

示例：

```
Result := Fct(7) + 3;
```

8.3.2.6 系统标志

8.3.2.6.1 概述

系统标志是隐式声明的变量，对每个专用的 PLC 来说是各不相同的。使用命令“Insert”（插入）→“Operand”（操作数），可以查找系统可用的系统标志。在“Input Assistant”（输入助手）对话框中，选择“System Variable”（系统变量）类别。

8.3.2.6.2 SYSTEMINFO

```
VAR_GLOBAL
  SystemInfo    AT%MB32768 (* 实际地址可能不同!*) : SYSTEMINFOTYPE;
END_VAR
```

系统标志是隐式声明的变量，对每个专用的 PLC 来说是各不相同的。类型 “SYSTEMINFOTYPE” 在系统库中声明，因此，该库必须包含在库管理器内。

开发环境	目标系统类型	包含的 PLC 库
TwinCAT v2.7.0	PC (i386)	PLCSystem.Lib
TwinCAT v2.8.0	PC (i386)	TcSystem.Lib

8.3.2.6.3 SYSTEMTASKINFOARR

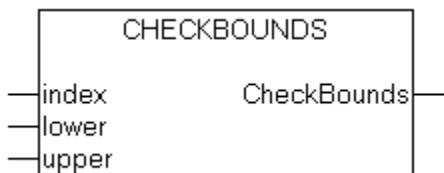
```
VAR_GLOBAL
  SystemTaskInfoArr  AT%MB32832 (* 实际地址可能不同!*) : ARRAY[1..4] OF
SYSTEMTASKINFOTYPE;
END_VAR
```

系统标志是隐式声明的变量，对每个专用的 PLC 来说是各不相同的。类型 “SYSTEMINFOTYPE” 在系统库中声明，因此，这个库必须包含在库管理器内。该数组的索引是 `task-Id`，当你在任务中调用功能块 `GETCURTASKINDEX` 时可以得到这个 `task-Id`。

开发环境	目标系统类型	包含的 PLC 库
TwinCAT v2.7.0	PC (i386)	PLCSystem.Lib
TwinCAT v2.8.0	PC (i386)	TcSystem.Lib

8.4 系统功能

8.4.1 CheckBounds 功能



如果你在项目中以 **CheckBounds** 为名定义一个功能，则可以在该项目中检验范围溢出！该功能名的定义是固定的，并且只能使用该标识符。关于如何实现这个功能的示例，请参见下例：

注：

CheckBound 功能可能会导致增加系统的装载量。**CheckBounds** 仅用于测试目的。

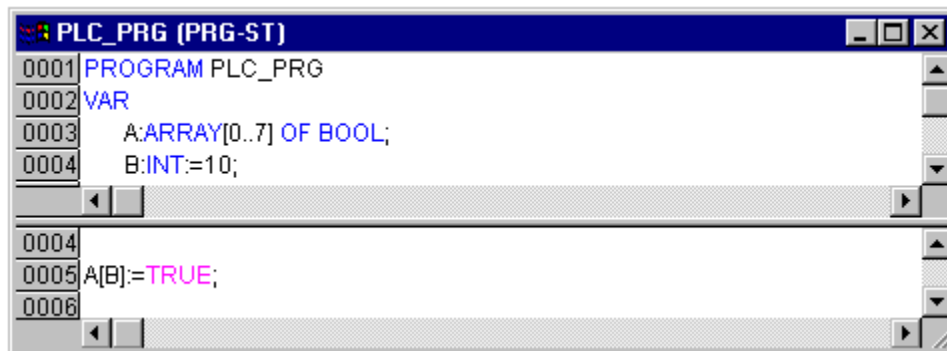
CheckBounds 功能：DINT

```
VAR_INPUT
  index, lower, upper : DINT;
END_VAR
```

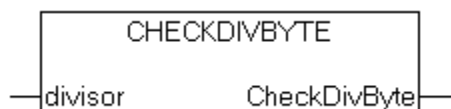
实现 **CheckBounds** 功能示例：

```
IF index<lower THEN
  CheckBounds := lower;
ELSIF index>upper THEN
  CheckBounds := upper;
ELSE
  CheckBounds := index;
END_IF
```

下面是测试 **CheckBounds** 功能的典型程序，它可以对一个定义数组进行超出边界检验。**CheckBounds** 功能可以确认 **TRUE** 不能分配给 **A[10]**，而是将其分配给仍然有效的上边界 **A[7]**。因此，**CheckBounds** 功能可用于纠正超出数组边界的扩展。



8.4.2 CheckDivByte 功能



如果在项目中以 `CheckDivByte` 为名定义一个功能，并且使用了操作符 `DIV`，则可以通过该功能检查除数的值，例如，避免被 0 除，该功能名的定义是固定的，并且只能使用该标识符。

注：

`CheckDivByte` 功能可能会导致增加系统的装载量。`CheckDivByte` 仅用于测试目的。

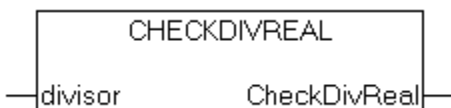
CheckDivByte 功能：BYTE

```
VAR_INPUT
    divisor :BYTE ;
END_VAR
```

实现 `CheckDivByte` 功能的示例：

```
IF divisor = 0 THEN
    CheckDivByte := 1; (**)
ELSE
    CheckDivByte := divisor;
END_IF
```

8.4.3 CheckDivReal 功能



如果你在项目中以 `CheckDivReal` 为名定义一个功能，并且使用了操作符 `DIV`，则可以通过该功能检查除数的值，例如，避免被 0 除，该功能名的定义是固定的，并且只能使用该标识符。

注：

`CheckDivReal` 功能可能会导致增加系统的装载量。`CheckDivReal` 仅用于测试目的。

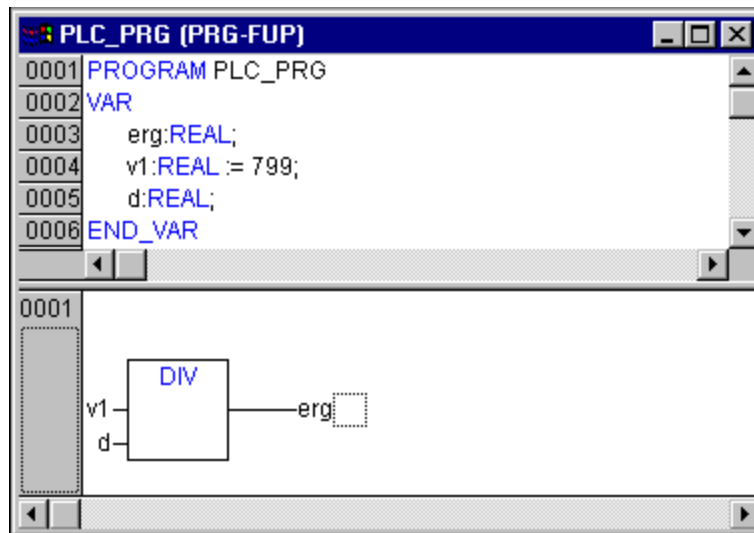
CheckDivReal 功能：REAL

```
VAR_INPUT
    divisor :REAL;
END_VAR
```

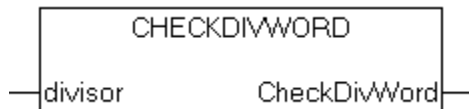
实现 `CheckDivReal` 的示例：

```
IF divisor = 0 THEN
    CheckDivReal := 1;
ELSE
    CheckDivReal := divisor;
END_IF
```

操作符 `DIV` 使用 `CheckDivReal` 的输出作为除数。下面的程序示例可以避免被 0 除，除数 (`d`) 设定为从 0 到 1，因此其结果是 799。



8.4.4 CheckDivWord 功能



如果你在项目中以 `CheckDivWord` 为名定义一个功能，并且使用了操作符 `DIV`，则可以通过该功能检查除数的值，例如，避免被 0 除，该功能名的定义是固定的，并且只能使用该标识符。

注：

`CheckDivWord` 功能可能会导致增加系统的装载量。`CheckDivWord` 仅用于测试目的。

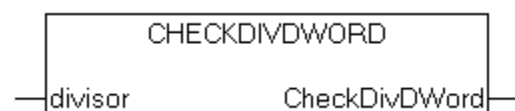
CheckDivWord 功能：WORD

```
VAR_INPUT
  divisor :WORD;
END_VAR
```

实现 `CheckDivWord` 功能的示例：

```
IF divisor = 0 THEN
  CheckDivWord := 1;
ELSE
  CheckDivWord := divisor;
END_IF
```

8.4.5 CheckDivDWord 功能



如果你在项目中以 `CheckDivDWord` 为名定义一个功能，并且使用了操作符 `DIV`，则可以通过该功能检查除数的值，例如，避免被 0 除，该功能名的定义是固定的，并且只能使用该标识符。

注：

CheckDivDWord 功能可能会导致增加系统的装载量。CheckDivDWord 仅用于测试目的。

CheckDivDWord 功能: DWORD

```
VAR_INPUT
```

```
  divisor :DWORD;
```

```
END_VAR
```

实现 CheckDivDWord 功能的示例:

```
IF divisor = 0 THEN
```

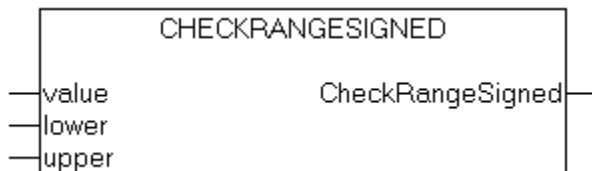
```
  CheckDivDWord := 1;
```

```
ELSE
```

```
  CheckDivDWord := divisor;
```

```
END_IF
```

8.4.6 CheckRangeSigned 功能



为了在运行时检查是否遵守范围边界，必须引入 **CheckRangeSigned** 功能。可以通过合适的方法和手段来捕捉违反边界的事件（例如，可以切断数值或设置一个出错标志）。只要变量属于一个子范围类型（该子范围类型由一个有符号类型构成），系统就可以隐式地调用该功能。

CheckRangeSigned 功能: DINT

```
VAR_INPUT
```

```
  value, lower, upper: DINT;
```

```
END_VAR
```

示例:

当变量属于一个有符号的子范围类型时，**CheckRangeSigned** 功能被调用；可以通过下面的程序来修正该值在允许的范围内：

```
FUNCTION CheckRangeSigned : DINT
```

```
VAR_INPUT
```

```
  value, lower, upper: DINT;
```

```
END_VAR
```

```
IF (value < lower) THEN
```

```
  CheckRangeSigned := lower;
```

```
ELSIF (value > upper) THEN
```

```
  CheckRangeSigned := upper;
```

```
ELSE
```

```
  CheckRangeSigned := value;
```

```
END_IF
```

当自动调用该功能时，`CheckRangeSigned` 的功能名是有约束的，这是由于接口规范的原因所致：返回数据和三个参数都是 `DINT` 类型。当调用时，该功能进行如下参数化：

<code>value</code>	分配到范围类型的数值
<code>lower</code>	范围的下边界
<code>upper</code>	范围的上边界
返回值	实际分配给范围类型的数值

在这个示例中，系统隐式地分配 `l := 10 * y` 的结果为：

`i := CheckRangeSigned(10 * y, -4095, 4095)`；即使在这个示例中，`Y` 的值为 1000，但经过系统处理的分配后，`l` 的值仍然只能是 4095。

注：

- 如果既没有定义 `CheckRangeSigned` 功能，也没有定义 `CheckRangeUnsigned` 功能，则在运行时不进行子范围类型的输入检查！变量 `i` 可以介于 `-32768` 和 `32767` 之间！
- 如果按照上例实现了 `CheckRangeSigned` 功能或 `CheckRangeUnsigned` 功能，则在 `FOR` 循环中可以使用子范围类型实现一个连续循环。如果 `FOR` 循环区和子范围类型一样大或大于子范围类型，则该功能将会起作用！

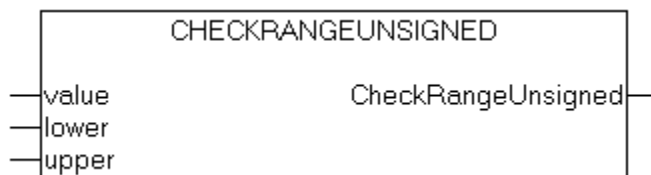
示例：

```
VAR
    ui : UINT (0..10000);
END_VAR

FOR ui:=0 TO 10000 DO
    ...
END_FOR
```

由于 `ui` 不会大于 10000，因此，`FOR` 循环不会出现遗漏。类似地，在 `FOR` 循环中使用增量值时，应该考虑 `CheckRange` 功能的内容。

8.4.7 CheckRangeUnsigned 功能



为了在运行时检查是否遵守范围边界，必须引入 `CheckRangeUnSigned` 功能。可以通过合适的方法和手段来捕捉违反边界的事件（例如，可以切断数值或设置一个出错标志）。只要变量属于一个子范围类型（该子范围类型由一个有符号类型构成），系统就可以隐式地调用该功能。

CheckRangeUnsigned 功能 UDINT

```
VAR_INPUT
    value, lower, upper: UDINT;
END_VAR
```

示例：

当变量属于一个有符号的子范围类型时，**CheckRangeUnsigned** 功能被调用；可以通过下面的程序来修正该值在允许的范围内：

```
FUNCTION CheckRangeUnsigned : UDINT
VAR_INPUT
    value, lower, upper: UDINT;
END_VAR

IF (value < lower) THEN
    CheckRangeUnsigned := lower;
ELSIF (value > upper) THEN
    CheckRangeUnsigned := upper;
ELSE
    CheckRangeUnsigned := value;
END_IF
```

当自动调用该功能时，**CheckRangeUnsigned** 的功能名是有约束的，这是由于接口规范的原因所致：返回数据和三个参数都是 **DINT** 类型。当调用时，该功能进行如下参数化：

当调用时，该功能如下进行参数化：

value	分配到范围类型的数值
lower	范围的下边界
upper	范围的上边界
返回值	实际分配给范围类型的数值

在这个示例中，系统隐式地分配 $i := 10 * y$ 的结果为：

$i := \text{CheckRangeSigned}(10 * y, 0, 4095)$ ；即使在这个示例中， Y 的值为 1000，但经过系统处理的分配后， i 的值仍然只能是 4095。

注：

- 如果既没有定义 **CheckRangeSigned** 功能，也没有定义 **CheckRangeUnsigned** 功能，则在运行时不进行子范围类型的输入检查！变量 i 可以介于 -32768 和 32767 之间！
- 如果按照上例实现了 **CheckRangeSigned** 功能或 **CheckRangeUnsigned** 功能，则在 **FOR** 循环中可以使用子范围类型实现一个连续循环。如果 **FOR** 循环区和子范围类型一样大或大于子范围类型，则该功能将会起作用！

示例：

```
VAR
    ui : UINT (0..10000);
END_VAR

FOR ui:=0 TO 10000 DO
    ...
END_FOR
```

由于 ui 不会大于 10000，因此，**FOR** 循环不会出现遗漏。类似地，在 **FOR** 循环中使用增量值时，应该考虑 **CheckRange** 功能的内容。

8.5 使用键盘

如果只想使用键盘运行 TwinCAT PLC Control，可能会使用到菜单中没有的一些命令。

- 使用功能键 <F6>，可以在打开的 POU 中，在声明部分和指令部分之间进行切换。
- 使用 <Alt>+<F6>，可以从一个打开的对象移动到对象管理器，如果信息窗被打开，则从那儿移动到信息窗。如果搜索框被打开，则使用 <Alt>+<F6>，可从对象管理器切换到搜索框，并从那儿返回到对象。
- 按 <Tab>，可以在对话框中的输入字段之间移动，包括对话框中的按钮。
- 使用箭头键，可以在属性页以及对象管理器和库管理器的对象之间进行移动。
- 所有其它动作都可以使用菜单命令或菜单命令后面的快捷键完成。使用 <Shift>+<F10>，可以打开上下文菜单，此菜单中包含选择对象或激活编辑器最常用的命令。

组合键

下面是所有组合键和功能键的一览表：

通用功能	
在 POU 的声明部分和指令部分之间移动	<F6>
在对象管理器、对象和信息窗之间移动	<Alt>+<F6>
上下文菜单	<Shift>+<F10>
用于说明的快捷模式	<Ctrl>+<Enter>
从信息窗的一个信息行返回到编辑器中的原始位置	<Enter>
打开和关闭多层变量	<Enter>
打开和关闭文件夹	<Enter>
切换对象管理器和库管理器中的属性页	<箭头键>
移动到对话框中的下一字段	<Tab>
上下文相关联的帮助	<F1>

通用命令	
“File”（文件）→ “Save”（保存）	<Ctrl>+<S>
“File”（文件）→ “Print”（打印）	<Ctrl>+<P>
“File”（文件）→ “Exit”（退出）	<Alt>+<F4>
“Project”（项目）→ “Delete Object”（删除对象）	
“Project”（项目）→ “Add Object”（添加对象）	<Ins>
“Project”（项目）→ “Object Rename” （对象重新命名）	<Spacebar>
“Project”（项目）→ “Edit Object”（编辑对象）	<Enter>
“Edit”（编辑）→ “Undo”（取消）	<Ctrl>+<Z>
“Edit”（编辑）→ “Redo”（重复执行）	<Ctrl>+<Y>
“Edit”（编辑）→ “Cut”（剪切）	<Ctrl>+<X> 或 <Shift>+
“Edit”（编辑）→ “Copy”（复制）	<Ctrl>+<C>

“Edit”（编辑）→ “Paste”（粘贴）	<Ctrl>+<V>
“Edit”（编辑）→ “Delete”（删除）	
“Edit”（编辑）→ “Find Next”（查找下一个）	<F3>
“Edit”（编辑）→ “Input Assistant”（输入助手）	<F2>
“Edit”（编辑）→ “Next Error”（下一个错误）	<F4>
“Edit”（编辑）→ “Previous Error”（前一个错误）	<Shift>+<F4>
“Online”（联机）→ “Run”（运行）	<F5>
“Online”（联机）→ “Toggle Breakpoint” （切换断点）	<F9>
“Online”（联机）→ “Step Over”（步进跳出）	<F10>
“Online”（联机）→ “Step In”（步进进入）	<F8>
“Online”（联机）→ “Single Cycle”（单循环）	<Ctrl>+<F5>
“Online”（联机）→ “Write Values”（写入值）	<Ctrl>+<F7>
“Online”（联机）→ “Force values”（强制值）	<F7>
“Online”（联机）→ “Release Force”（解除强制）	<Ctrl><Shift>+<F7>
“Online”（联机）→ “Write/Force Dialog” （写入/强制对话框）	<Shift>+<F7>
“Window”（窗口）→ “Messages”（信息）	<Shift>+<Esc>

FBD 编辑器命令	
“Insert”（插入）→ “Network (after)” （网络（后向））	<Shift>+<T>
“Insert”（插入）→ “Assign”（赋值）	<Ctrl> + <A>
“Insert”（插入）→ “Jump”（跳转）	<Ctrl> + <L>
“Insert”（插入）→ “Return”（返回）	<Ctrl> + <R>
“Insert”（插入）→ “Operator”（操作符）	<Ctrl> + <O>
“Insert”（插入）→ “Function”（功能）	<Ctrl>+<F>
“Insert”（插入）→ “Function Block”（功能块）	<Ctrl> +
“Insert”（插入）→ “Input”（输入）	<Ctrl> + <U>
“Extras”（附加）→ “Negate”（取反）	<Ctrl> + <N>
“Extras”（附加）→ “Zoom”（缩放）	<Alt>+<Enter>

CFC 编辑器命令	
“Insert”（插入）→ “POU”	<Ctrl> +
“Insert”（插入）→ “Input”（输入）	<Ctrl> + <E>
“Insert”（插入）→ “Output”（输出）	<Ctrl> + <A>
“Insert”（插入）→ “Jump”（跳转）	<Ctrl>+<G>
“Insert”（插入）→ “Label”（标号）	<Ctrl> + <L>

“Insert”（插入）→ “Return”（返回）	<Ctrl> + <R>
“Insert”（插入）→ “Comment”（注释）	<Ctrl> + <K>
“Insert”（插入）→ “POU input”（POU 输入）	<Ctrl> + <U>
“Extras”（附加）→ “Negate”（取反）	<Ctrl> + <N>
“Extras”（附加）→ “Set/Reset”（置位/复位）	<Ctrl>+<T>
“Extras”（附加）→ “Connection”（连接）	<Ctrl>+<M>
“Extras”（附加）→ “EN/ENO”	<Ctrl> + <O>
“Extras”（附加）→ “Zoom”（缩放）	<Alt>+<Enter>

LD 编辑器命令	
“Insert”（插入）→ “Network (after)”（网络（后向））	<Shift>+<T>
“Insert”（插入）→ “Contact”（接点）	<Ctrl> + <O>
“Insert”（插入）→ “Parallel Contact”（并联接点）	<Ctrl> + <R>
“Insert”（插入）→ “Function Block”（功能块）	<Ctrl> +
“Insert”（插入）→ “Coil”（线圈）	<Ctrl> + <L>
“Extras”（附加）→ “Paste Below”（下部粘贴）	<Ctrl> + <U>
“Extras”（附加）→ “Negate”（取反）	<Ctrl> + <N>

SFC 编辑器命令	
“Insert”（插入）→ “Step Transition (before)”（步转换（前向））	<Ctrl>+<T>
“Insert”（插入）→ “Step Transition (after)”（步转换（后向））	<Ctrl> + <E>
“Insert”（插入）→ “Alternative Branch (right)”（选择分支（右向））	<Ctrl> + <A>
“Insert”（插入）→ “Parallel Branch (right)”（并联分支（右向））	<Ctrl> + <L>
“Insert”（插入）→ “Jump”（跳转）	<Ctrl> + <U>
“Extras”（附加）→ “Zoom Action/Transition”（缩放动作/转换）	<Alt>+<Enter>
从 SFC 总貌移回到编辑器	<Enter>

PLC 配置	
打开和关闭管理器成员	<Enter>
在名字周围放置一个编辑控制框	<Spacebar>
“Extras”（附加）→ “Edit Entry”（编辑输入项）	<Enter>

任务配置	
在任务或程序名称周围放置一个编辑控制框	<Spacebar>

8.6 创建错误表

编辑器错误

这里，你可以找到语法分析（斜体字）的错误消息及其可能的错误原因。

1500, 3100, 3200, 3400, 3500, 3600, 3700, 3800, 3900, 4000, 4100, 4200, 4300, 4400, 4500

报警

编号	错误消息	可能的原因
1100	未知库中的功能名'<name>'	使用一个外部库。请检查，在 hex 文件中定义的所有功能是否也定义在 lib 文件内。
1101	不能识别的符号'<Symbol>'	代码生成器期望有一个名称为<Symbol>的 POU。它没有在目标项目中定义。并使用这个名称定义了一个功能/程序。
1102	对符号'<Symbol>'的无效接口	代码生成器期望有一个名为<Symbol>的功能和恰好是一个标量的输入，或一个名为<Symbol>和无输入或输出的程序。
1103	在代码地址'<address>'的常数'<name>'改写一个 16K 页边界！	超过 16K 边界的一串常数。系统不能处理这种边界。这取决于运行时系统，通过目标文件中的一个登录项是否能解决这个问题。请与 PLC 制造商联系。
1200	任务 '%s'，调用 '% 在参数表中的存取变量未更新。	在任务配置时，只用于一个功能块调用的变量不列入在交叉引用表内。
1300	文件未找到'<name>'	全局变量对象所指向的文件不存在。请检查文件路径。
1301	找不到分析库！未生成用于分析的代码。	使用了分析功能，但丢失库分析 .lib。将它加入到库管理程序内。
1302	新的外部引用功能插入。从而不再可能进行“Online Change”（联机改变）！	由于最后装载，你链接的一个库它所包含的功能尚未在运行时系统中引用。为此理由，你应装载整个项目。
1400	忽略未知附注'<name>'！	这个附注不受编译程序支持。受支持的命令见关键字的附注。
1401	结构'<name>'不包含任何图形元素。	有名<name>的结构不包含任何图形元素。但是这种类型的变量将使用存储器的 1 个字节。
1500	表达式不包含赋值。不生成代码。	没有块这个表达式的结果。为此理由，不会对整个表达式生成代码。
1501	串常数作为'VAR_IN_OUT'!'<name>'不需重写！	该常数也许未写入到 POU 内，这是因为在那里不可能进行容量检验。
1502	变量'<name>'与一个 POU 有相同名。不能调用 POU！	使用一个与 POU 相同名的变量。 例子： PROGRAM a ... VAR_GLOBAL a:INT; END_VAR ... a; (*不调用 POU 而是装入变量 a .. *)
1503	POU '<name>'没有输出。框结果设置为'TRUE'。	无输出的一个 POU 输出插针连接在 FBD 或 KOP。通过自动赋值得到值 TRUE。
1504	'<name>' ('<number>')：由于逻辑表达式计算，不能执行语句。	最终并非所有的逻辑表达式的分支都可执行。例子： IF a AND funct(TRUE) THEN

		若 a 是 funct, 则不能调用功能。
1505	'<name>'的副作用! 几乎不能执行分支!	POU 的第一个输入是 FALSE, 为此理由, 不执行在第二个输入进入的侧分支。
1506	变量'%s'与一个局部动作有相同的名。不能调用该动作!	重新命名该变量或该动作。
1600	打开的 DB 不清楚 (生成的代码也许赋值)。	未告知原始的 Siemens 程序, 在该程序中, POU 是打开的。
1700	输入框没有赋值。	CFC 中所用的一个输入框没有赋值。为此, 不生成代码。
1800	<name>(图形元素 #<图形元素号>): 无效的监视表达式 '%s'	可视化图形元素含一个不能监视的表达式。检查变量名和替换的占位符。
1801	'<name> (number): 表达式 '<name>'可能无输入。	在配置现场输入的可视化对象时, 使用了一个组合的表达式。应用一个单独的变量替换这个表达式。
1900	在库中, 没有提供 POU '<name>' (主例行程序)。	当项目作为库使用时, 不能利用“启动 POU”(例如 PLC_PRG)。
1901	存取变量和变量配置没有保存在库内!	存取变量和变量配置没有保存在库内。
1902	'<name>': 不是为当前的机器类型用的库!	库的对象文件是为其它设备而生成的。
1903	<name>: 不是有效的库。	文件没有实际目标所需要的格式。

编辑器错误

号	出错消息	可能的原因
3100	代码太长。最大容量: '<number>' 字节 (<number>K)	超过最大程序规模。减小项目规模。
3101	总数据太大。最大容量: '<number>' 字节 (<number>K)	超过保存量。减少应用程序的数据应用。
3110	库文件中的错误 '<name>'。	.hex 文件不是以 INTEL Hex 格式。
3111	库 '<name>' 太大。最大容量: 64K	.hex 文件超过设定的最大容量。
3112	库中的不可重定位的指令。	.hex 文件包含有一个不可重定位的指令。不能链接库代码。
3113	库代码覆盖了功能表。	代码和功能表的范围重叠。
3114	库使用了不止一个字段。	.hex 文件中的表和代码利用不止一个字段。
3115	不能将常数分配给 VAR_IN_OUT。数据类型不兼容。	由于数据设置在“near” (近的), 但是串常数设置在“huge” (大的) 或“far” (远的), 因而串常数的内部指针格式不能转换成 VAR_IN_OUT 的内部指针格式。如有可能, 改变这些目标设置。
3120	当前的代码段超过 64K。	当前生成的代码大于 64K。最终建立太多的初始化代码。
3121	POU 太大。一个 POU 的规模不应超过 64K。	一个 POU 的规模不应超过 64K。
3122	初始化太大。最大容量: 64K	用于一个功能或一个结构化 POU 的初始化代码不应超过 64K。
3130	用户堆栈太小: '<number>' DWORD 可利用 '<number>' DWORD。	POU 调用的嵌套深度太大。在目标设置中输入一个更大的堆栈尺寸, 或编译没有任选项“Debug” (可在对话框“Project” (项目) “ (Options) ”任选项“Build” (建立) 中设置) 的建立项目。
3131	用户堆栈太小: '<number>' WORD, 可利用 '<number>' WORD。	请与 PLC 制造商联系。

3132	系统堆栈太小: '<number>' WORD, 可利用 '<number>' WORD。	请与 PLC 制造商联系。
3150	功能 '<name>' 的参数 <number>: 不能将一个 IEC 功能的结果作为串参数传送给一个 C 功能。	使用一个中间变量, 将 IEC 功能的结果赋值给该中间变量。
3160	不能打开库文件 '<name>'。	一个库 <name> (名) 包括在用于这个项目的库管理程序内。但是库文件不在给定的路径处。
3161	库 '<name>' 不包含代码段。	一个程序库的一个 .obj 文件至少应包含一个 C 功能。将一个哑功能插入到 .lib 文件内, 在 .lib 文件内尚未定义这个哑功能。
3162	不能分辨库中的引用 '<name>' Symbol' (符号 <name>', 级别 '<name>', 类型 '<name>')	.obj 文件包含一个对其它符号不能分辨的引用。请检查 C 编译程序的设置。
3163	库中求知的引用类型 '<name>' (符号 '<name>', 级别 '<name>', 类型 '<name>')	.obj 文件包含一个引用类型, 它不能被代码发生器所分辨。请检查 C 编译程序的设置。
3200	"%s (%d): 布尔表达式太复杂。	目标系统的暂时存储器容量对表达式的规模来说显得不够。将表达式分成几个部分表达式, 为此可使用对几个中间变量进行赋值。
3201	<name> (<network>): 一个网络的结果不应大于 512 字节的代码。	不能分辨内部跳转。启动选项“利用 16 位跳转补偿”中的 68K 目标设置。
3202	用于嵌套的字符串/数组/结构功能调用的堆栈超限。	使用一嵌套的功能调用 CONCAT(x, f(i))。这会导致数据丢失。将调用分成两个表达式。
3203	表达式太复杂 (所用的地址寄存器太多)。	将赋值分成几个表达式。
3204	一个跳转超过 32K 字节。	跳转距离不要大于 32767 字节。
3205	内部错误: 常数串太多。在一个 POU 内可用的最大常数串为 3000 个。	在一个 POU 内, 可以使用的最大常数为 3000 个。
3206	功能块数据超过最大容量。	一个功能块可以产生最大 32767 字节代码。
3207	数组优化	由于在索引计算过程中出现一个功能调用。使数组存取优化失败。
3208	尚未实现的转换。	使用的转换功能, 不是为实际代码发生器而实现的。
3209	未实现的操作符。	使用的操作符, 不是为这种数据类型和实际的代码发生器而实现的。MIN (字符串 1, 字符串 2)
3210	找不到功能 '<name>'。	调用的功能不能用于项目内。
3211	超过最大串使用。	类型串的一个变量在一个表达式中最多可用 10 次。
3250	实际不支持 8 位控制器。	目标当前未得到支持。
3251	日类型的日期不支持 8 位控制器。	目标当前未得到支持。
3252	堆栈规模超过 <number> 字节。	目标当前未得到支持。
3253	不能找到十六进制文件: “ <name> ”	目标当前未得到支持。
3254	不能分辨对外部库功能的调用。	目标当前未得到支持。

3400	在输入存取变量期间出现一个错误。	.exp 文件包含一个不正确的存取变量段。
3401	在输入变量配置期间出现一个错误。	.exp 文件包含一个不正确的配置变量段。
3402	在输入全局变量期间出现一个错误。	.exp 文件包含一个不正确的全局变量段。
3403	不能输入<name>	.exp 文件中的对象<name>段不正确。
3404	在输入任务配置期间出现一个错误。	.exp 文件中的任务配置段不正确。
3405	在输入 PLC 配置期间出现一个错误。	.exp 文件中的 PLC 配置段不正确。
3406	名'<name>'有二步。未输入第二步。	.exp 文件中的 SFC POU 段包含有相同名的两个步。在输出文件中重新命名其中的一步。
3407	未找到先行步'<name>'。	.exp 文件中丢失步<name>。
3408	未找到后继步 '<name>'。	.exp 文件中丢失步<name>。
3409	没有后继转换的步'<' name>'。	.exp 文件中, 丢失转换名, 它需要步<name>作为先行步。
3410	没有传输'<name>'的顺序步。	.exp 文件中, 丢失一个步, 它需要转换名<name>作为先行条件。
3411	步'<name>'不能从初始步达到。	.exp 文件中, 丢失步<name>与初始步之间的连接。
3450	PDO'<PDO-name>': 丢失 COB-Id!	为此模块, 点出 PLC 配置对话框中的“Properties”(属性), 并输入用于 PDO <PDO 名>的一个 COB ID。
3451	装入过程中的错误: EDS 文件 '<name>'找不到。但在硬件配置中引用!	CAN 配置所需的设备文件不在正确的目录内。检查“Project”(项目)“Options”(选项)“目录”中的配置文件的目录设置。
3452	不能建立模块'<name>'!	用于模块<name>的设备文件不适应于当前的配置。自建立配置以来已作了修改, 或者它已损坏。
3453	不能建立通道'<name>'!	用于通道<name>的设备文件不适应于当前的配置。自建立配置以来已作了修改, 或者它已损坏。
3454	地址'<name>'指向一个已用的存储器!	在对话框PLC 配置的“Settings”(设置)中, 已启动选项“Check for overlapping addresses”(检查重叠地址), 并已检测到一个重叠。注意, 区域检查是基于尺寸, 它由模块的数据类型有关。在配置文件中, 通过登录项“size”(尺寸), 给出区域尺寸。
3455	装入过程中的错误: 找不到 GSD 文件'<name>', 但在硬件配置中被引用!	Profibus 配置所需的设备文件不在正确的目录内。检查“Project”(项目)“Options”(选项)“目录”中的配置文件的目录设置。
3456	不能建立 profibus 设备 '<name>'!	用于模块<name>的设备文件不适应于当前的配置。自建立配置以来已作了修改, 或者它已损坏。
3457	模块描述中的错误!	请检查这个模块的设备文件。
3500	没有用于'VAR_CONFIG' 的 '<name>'	将用于变量的说明插入到全局变量表内, 该表包含“Variable_Configuration”(变量_配置)。
3501	没有用于'VAR_CONFIG'用的地址'<name>'。	将这个变量的一个地址分配给全局变量表, 该表包含“Variable_Configuration”(变量_配置)。
3502	'VAR_CONFIG'中的错误的数据类型'<name>'。	在包含“Variable_Configuration”(变量_配置)的全局变量表中, 变量以不同于 POU 中的数据类型说明。
3503	'VAR_CONFIG'中的错误的数据类型'<name>'。	在包含“Variable_Configuration”(变量_配置)的全局变量表中, 变量以不同于 POU 中的数据类型说明。
3504	初始值不支持'VAR_CONFIG	“Variable Configuration”(变量_配置)的一个变量是以地址

		和初始值说明的。但是，一个初始值只能为没有地址分配的输入变量而定义的。
3505	'<name>'没有有效的实例路径	“Variable Configuration”（变量_配置）包括一个存在的变量。
3506	期望的存取路径	在“Access Variables”（存取变量）的全局变量表中，一个变量的存取路径是不正确的。正确的应该是：<标识符>: '<存取路径>': <类型> <存取方式>
3507	对'VAR_ACCESS'变量无地址规范	“Access Variables”（存取变量）的全局变量表包含用于一个变量的地址分配。这是不允许的。有效的变量定义是：<标识符>: '<存取路径>': <类型> <存取方式>
3550	标识符'<name>'的重复定义	使用一个相同的名称来定义两个任务。重新命名其中的一个。
3551	任务'<name>'至少必需包含一个程序调用	插入一个程序调用或删除任务。
3552	没有定义在任务'%s'中的事件变量'<name>'	在任务属性的“Single”（单）区域对话框中设置的一个事件变量，在项目未经全局说明。利用其它变量或全局定义该变量。
3553	任务'%s'中的“事件变量”'<name>'必须是类型 'BOOL'	在任务属性的“Single”（单）区域对话框中，使用一个类型 BOOL 的变量作为事件变量。
3554	任务登录项 '<name>' 必须是一个程序或全局功能块实例	在该字段中，输入“Program call”（程序调用）一个功能或一个不定义的 POU。输入一个有效的程序名。
3555	任务登录项'<name>'包含无效的程序	在该字段中，所用的“Append program call”（附加程序）参数不符合程序 POU 的说明。
3600	找不到隐含的变量！	使用命令“Rebuild All”（重建所有）。如依然得到出错消息，请与 PLC 制造商联系。
3601	<name>是一个保持的变量名	给出的变量在项目中说明，虽然它是保持用于代码发生器。重新命名该变量。
3610	不支持的'<name>'	给出的属性不被编译系统的当前版本支持。
3611	给出的编译目录'<name>'是无效的	用于编译文件的“Project”（项目）→“Options”（选项）→“Directories”（目录）中给出的目录是无效的。
3612	超过 POU (<number>) 的最大数！编译被删除。	在项目中所用的 POU 和数据类型太多。在“Target Settings / Memory Layout”（目标设置/存储器布局）中，修改 POU 的最大数。
3613	建立被删除	编译过程被用户删除。
3614	项目必须包含一个 POU 命名的'<name>'（主例程序）或一个任务配置	建立一个类型“Program”（程序）的初始 POU（例如 PLC_PRG），或建立一个任务配置。
3615	<name>（主例程）必须是类型“program”（程序）	在项目中所用的一个初始 POU（例如 PLC_PRG）不是“Program”（程序）类型。
3616	程序不必在外部库中实现	应作为外部库保存的项目包含一个程序。当使用该库时，不能使用这个程序。
3617	超出存储器容量	增加你的计算机的虚拟存储器的容量。
3618	在当前的代码发生器中，位存取不受支持！	当前设置的目标系统的代码发生器，不支持对变量的位存取。
3700	POU 名'<name>'早已在库'<name>'内	一个 POU 名已用于项目内，这一名早已用于一个库 POU。重新命名 POU。
3701	用于接口的名与 POU 名不同	使用命令“Project”（项目）→“Rename object”（重新命名对象），重新命名“Object Organizer（对象组织程序）”中的 POU 或在说明窗内改变 POU 的名。在那里，POU 名必须放置在邻近关键字 PROGRAM, FUNCTION 或 FUNCTIONBLOCK 中

		的一个。
3702	标识符表溢出	最大为 100 个标识符可输入到一个变量说明内。
3703	标识符'<name>'的重复定义	要注意, 在 POU 的说明部分中, 只允许一个标识符有给定的名。
3905	数据递归: <POU 0> -> <POU 1> -> ..-> <POU 0>	使用了需要自身的 FB 实例。
3720	期望在'AT'后的地址	在关键字 AT 后面加一个有效地址, 或修改关键字。
3721	只有 'VAR' 和 'VAR_GLOBAL'可以装入到地址	将说明放置在一个 VAR 或 VAR_GLOBAL 说明区。
3722	只有'BOOL'变量允许有位地址	修改地址或修改分配地址的变量类型。
3729	在地址'<name>'处的无效类型: “ <name> ”	这种类型的变量不能放置在给定地址上。例: 对于以'alignment 2'工作的一个目标系统, 以下的说明是无效的: var1 AT %IB1:WORD;
3740	无效类型: “ <name> ”	在一个变量说明中, 使用无效的数据类型。
3741	预期的类型规范	使用一个关键字或一个操作符以替代一个有效类型标识符。
3742	预期的枚举值	在枚举类型定义中, 在打开的括号后, 或在括号之间的逗号后面, 丢失一个标识符。
3743	预期的整数	枚举只能以类型 INT 的数初始化。
3744	早已定义了枚举常数 '<name>'	检查一下, 你是否遵循了以下的枚举值定义规则: - 在一个枚举定义中, 所有的值均应是唯一的。 - 在所有的全局枚举定义中, 所有的值均应是唯一的。 - 在所有的局部枚举定义中, 所有的值均应是唯一的。
3745	在整数上只允许子范围!	子范围类型只能在整数数据类型上定义。
3746	子范围'<name>'不与类型 '<name>'兼容	子范围类型的范围极限设定中, 有一个极限设定超出其基本类型的有效范围。
3747	串的长度未知: “ <name> ”	没有一个有效的常数用于串长的定义。
3748	数组的维数不允许大于三维	在一个数据的定义中, 给出允许的维数即大于三维。若可应用的话, 使用“ARRAY OF ARRAY” (数组的数组)。
3749	未规定下限'<name>'	使用一个未定义的常数来定义一个子范围或数组类型的下限。
3750	未规定上限'<name>'	使用一个未定义的常数来定义一个子范围或数组类型的上限。
3760	初始值错误	使用一个与类型定义相当的初始值。为了更改说明, 你可使用变量的说明对话框 (Shift/F2 或 “Edit” (编辑) “Autodeclare” (自动说明))。
3761	'VAR_IN_OUT' 变量不必有一个初始值	在 VAR_IN_OUT 变量的说明部分, 去除初始化。
3780	期望的 VAR, VAR_INPUT, VAR_OUTPUT 或 VAR_IN_OUT	POU 名后的第一行必须包含这些关键字中的一个关键字。
3781	期望的 "'END_VAR' 或标识符	在说明窗内的给出行的起始位置输入一个 END_VAR 的有效标识符。
3782	未期望的结束	在说明性编辑器中: 在说明部分结束处加上关键字 END_VAR。 在编译部分的文本编辑器中: 加上一个指令, 它终止最后的指令顺序 (例如 END_IF)。
3783	期望的 END_STRUCT' 或标识符	应确实弄清楚, 类型说明已正确地终止。

3800	全局变量需要太多的保存量。在项目选项中增加可供利用的存储器	增加在对话框“Project”（项目）→“Options”（选项）→“Build”（建立）的设定中所给出的程序段数。
3801	变量'<name>'太大（<Size>字节）	变量使用大于 1 个数据段的类型。段尺寸是一个目标特定的参数并可在目标设定/存储器布局中修改。如果你在当前的目标设定中找不到它，请与你的 PLC 制造商联系。
3802	超出保持的保存量。变量'<name>', <number> 字节	可供保变量用的内存空间已耗尽。在目标设定/存储器布局中可设定目标专用的内保存区域尺寸。如果你在对话框中找不到设定字段，请与你的 PLC 制造商联系。（请注：若保变量用于一个功能块实例，完整的实例 POU 将保存在保持内存区域！）
3803	超出全局数据保存量。变量'<name>', <number>'字节	可供全局变量用的内存空间已耗尽。在目标设定/存储器布局中可设定目标专用的内保存区域尺寸。如果你在对话框中找不到设定字段，请与你的 PLC 制造商联系。
3820	不允许在功能中有"VAR_OUTPUT" 和"VAR_IN_OUT"	在一个功能中，可以定义无输出或输入_输出变量。
3821	功能至少需要一个输入	为这种功能至少附加一个输入参数。
3840	未知的全局变量'<name>'!	在 POU 中，使用一个 VAR_EXTERNAL 变量，对它未说明其全局变量。
3841	'<name>' 的说明与全局说明不匹配!	在 VAR_EXTERNAL 变量说明中给出的类型与全局说明中的类型不匹配。
3900	标识符中的多重下划线	在标识符名下除去多重下划线。
3901	最多允许 4 个数字域地址	有一种表达式允许直接赋值给一个地址，它不止四层的数字域（例如 %QB0.1.1.0.1）。
3902	关键字必须大写	对关键字应使用大写字母，或启动选项，“Project”（项目）→“Options”（选项）中的“Autoformat”（自动格式化）。
3903	无效的持续常数	常数的记数法不符合 IEC61131-3 格式。
3904	持续常数溢出	用于时间常数值不能以内部格式表示。可表示的最大值为 t#49d17h2m47s295ms。
3905	无效的数据常数	常数的记数法不符合 IEC61131-3 格式。
3906	无效的日常数时间	常数的记数法不符合 IEC61131-3 格式。
3907	无效的日期和时间常数	常数的记数法不符合 IEC61131-3 格式。
3908	无效的串常数	串常数包含一个无效的字符。
4000	期望的标识符	在这个位置输入一个有效的标识符。
4001	未说明变量'<name>'	说明变量是局部或全局。
4010	类型失配: 不能将“<name>”转换成“<name>”	检查一下，操作符期望的是何种数据类型(为操作符名而“Browse Online Help”（浏览联机帮助）)，并改变引起错误的变量类型，或选择其它变量。
4011	在'<name>'的参数'<name>'中，类型失配: 不能将“<name>”转换成“<name>”	实际参数的数据类型不能自动转换成格式化参数的类型。使用某种类型转换方式或使用其它变量类型。
4012	在'<name>'的参数'<name>'中，类型失配: 不能将“<name>”转换成“<name>”	将一个无效类型<类型 2>变量分配给输入变量'<name>'。将该变量或常数以一个类型<类型 1>变量代之，或采用一种类型转换。相应地，一个带有类型前缀的常数。
4013	在“<name>”输出“<name>”中，类型失配: 不能将“<name>”转换成“<name>”	将一个无效类型<类型 2>变量分配给输出变量'<name>'。将该变量或常数以一个类型<类型 1>变量代之，或采用一种类型转换。相应地，一个带有类型前缀的常数。

4014	打印的文字: 不能将“〈name〉”转换成“〈name〉”	常数类型与前缀类型不兼容。例: SINT#255
4015	数据类型“〈name〉”对直接位存取是非法的	直接位编址只允许用于整数和“位串”数据类型。你在位存取 <var1>.<bit>中, 使用 REAL/LREAL 类型的变量 var1, 或一个常数。
4016	位索引'<number>'超出类型“〈name〉”的变量范围	你正式试图存取的一位, 它不是为变量的数据类型而规定的。
4017	'MOD'不是为'REAL'规定的	操作符 MOD 只可用于整数和位串数据类型。
4020	'ST', 'STN', 'S', 'R'需要直接地址或具有写存取的变量	使用一个具有写存取的变量取代第一个操作数。
4021	不允许对变量“%S”进行写存取	使用一个具有写存取的变量取代该变量。
4022	期望的操作数	在注释后面加上一个操作数。
4023	“+”或“-”后面的期望数	输入一个数字。
4024	〈name〉前面期望的〈operator 0〉或〈operator 1〉或…	在命名的位置处, 输入一个有效的操作数。
4025	〈name〉前面期望的“:=”或“=>”	在命名的位置处, 输入二种操作符中的一种。
4026	“BITADR”期望一个位地址或一个位地址上的变量	使用一个有效的位地址(例如, %1×0.1)
4027	期望的整数或符号常数	输入一个整数或一个有效常数的标识符。
4028	“INI”操作符需要功能块实例或数据单位类型实例	检查 INI 操作符所使用的变量的数据类型。
4029	相同功能的嵌套调用是不可能的。	在不是可重入的目标系统和在仿方式时, 一个功能调用并不包含作为参数的自身调用。例: fun1 (a, fun1 (b, c, d), e); 可使用一个中间表。
4030	表达式和常数不允许作为“ADR”的操作数	使用一个变量或一个直接地址取代常数或表达式。
4031	在位上不允许“ADR”! 用“BITADR”取代。	使用 BITADR。请注: BITADR 功能不返回一个物理的内存地址
4032	用于<name>的<number>操作数太多。至少需要〈number〉	检查一下, 命名的操作符需要多少操作数, 并添加缺少的操作数。
4033	用于<name>的<number>操作数太多。至少需要〈number〉	检查一下, 命名的操作符需要多少操作数, 并除去多余的操作数。
4034	以 0 除	你正在一个常数表达式中以0作为除数。如果你要引起一个运行时错误, 使用—如可以应用的话—一个有值为0的变量。
4035	若启动“replaced constants”(替代的常数)不必将 ADR 应用于 VAR CONSTANT	使用直接值的一个常数地址存取是不可能的。如可应用的话, 取消“Project”(项目)→“Options”(选项)→“Build”(建立)对话框中的选项“Replace Constants”(取代常数)。
4040	未定义的标记〈name〉	使用名〈标记名〉定义一个标记, 或将名〈标记名〉改变成一个定义的标记名。
4041	重复定义的标记〈name〉	在 POU 中, 标记〈name〉是重复定义的。重新命名标记或从重复定义中除去一个名。
4042	允许在顺序中不多于 %d 标记	跳转标记数限制在“〈Anzahl〉”。插入一个哑指令。
4043	Format of label invalid. 一个标记必须是一个任选的名, 后跟一	标记名是无效的, 或在表达式中丢失了冒号。

	个冒号。	
4050	未定义 POU “%s”	利用命令“Project”（项目）→“Add Object”（附加对象）来定义有名〈name〉的一个 POU，或将〈name〉改变成一个已定义的 POU 名。
4051	“%s”没有功能	替代〈name〉，使用一个在项目或在库中定义的功能名。
4052	“%s”必须是一个 FB “%s”的说明实例	使用一个在项目中定义的数据类型〈name〉的实例，或将〈实例名〉类型改为〈name〉。
4053	“%s”是无效框或无效操作符	使用一个 POU 名或一个在项目定义的操作符来取代〈name〉。
4054	期望的 POU 名作为“INDEXOF”参数	给出的参数不是一个有效的 POU 名。
4060	“VAR_IN_OUT”的参数“%s”需要具有写存取变量作为输入。	具有写存取的变量必须移交给 VAR_IN_OUT 参数，这是因为，一个 VAR_IN_OUT 可在 POU 内修改。
4061	必须使用“%s”的参数“%s”	VAR_IN_OUT 参数必须有移交的具有写存取的变量，这是因为，一个 VAR_IN_OUT 可在 POU 内修改。
4062	对“%s”的参数“%s”不必以位地址使用。	VAR_IN_OUT 参数只能在 POU 内写入或读取，这是因为它们是由引用转交的。
4063	“%s”的参数“%s”不必以位地址使用。	一个位地址不是一个有效的物理地址。转交一个变量或一个直接的非位地址。
4064	“VAR_IN_OUT”不必在局部动作调用内改写！	在局部动作调用中，删除用于 VAR_IN_OUT 变量的参数集。
4070	POU 包含一个太复杂的表达式	通过将表达式分成几个表达式来降低嵌套深度。为此，可使用中间变量。
4071	网络太复杂	将网络分成几个网络。
4100	需要一个指示符类型	你正在试图将一个不作为指示符说明的变量非关联化。
4110	[〈index〉]需要的数组变量	[〈index〉]用于一个变量，它不是作为有 ARRAY OF 的一个数组加以说明的。
4111	一个数组的索引表达式必须是类型“INT”	使用正确类型的表达式，或类型转换。
4112	数组的索引太多	检查索引数（1，2或3），这些索引是说明数组的，除去多余的索引。
4113	数组的索引太少	检查索引数（1，2或3），这些索引是说明数组的；添加缺少的索引。
4114	常数索引中的一个索引不在数组范围内	确实弄清楚，所用的索引是在数组界限之内。
4120	“.”、“.”需要结构化“variable”（变量）	句点左边的标识符必须是一个类型 STRUCT 或 FUNCTION_BLOCK 的变量，或是名为一个 FUNCTION 或一个 PROGRAM 的变量。
4121	“〈name〉”不是〈object name〉的一个成分	成分“〈name〉”不包括在对象〈object name〉的定义内。
4122	“%s”不是被调用的功能块的一个输入变量	检查一下，被调用的功能块的输入变量，并将“〈name〉”改成这些变量中的一个变量。
4200	期望的“LD”	在 IL 编辑器中，在跳转符号后面，至少应插入一个 LD 指令。
4201	期望的 IL 操作符	每个 IL 指令必须用一个操作符或一个跳转符开始。
4202	在括号内不期望的文本结束	在文本后插入一个关闭括号。

4203	在括号 <name> 是不允许的	操作符 <name> 在一个 IL 括号表达式中是无效的。(无效的是: “JMP”, “RET”, “CAL”, “LDN”, “LD”, “TIME”)
4204	关闭括号没有相应的打开括号	插入一个打开括号, 或除去这个关闭括号。
4205)' 后面不允许有逗号	在关闭括号后面除去逗号。
4206	括号内不允许有符号	移去跳转符号, 使其在括号外面。
4207	“N” 修改符需要类型为 “BOOL”, “BYTE”, “WORD” 或 “DWORD” 的操作数	N 修改符需要可执行布尔求反的数据类型。
4208	条件操作符需要类型 “BOOL”	确实弄清楚, 表达式给出布尔结果, 或应用类型转换。
4209	这里不允许有功能名	使用一个变量或一个常数来替换这个功能。
4210	“CAL”, “CALC” 和 “CALN” 需要一个功能块实例作为操作 数	说明一个你要调用的功能块的实例。
4211	允许在行结束处出现指令表(IL)	将注释移到行的结束处, 或移到附加行。
4212	条件语句前的累加器是无效的	未定义这个累加器。如果一个指令是领先的, 它未提交一个结果 (例如 “CAL”) 的话, 就会出现这种情况。
4213	“S” 和 “R” 需要 “BOOL” 操 作数	在这个位置使用一个布尔变量。
4250	期望另一个 “ST” 语句或 POU 的结束	该行不是以一个有效的 ST 指令开始。
4251	功能 “%s” 中的参数太多	给出的参数多于功能定义中所说明的参数。
4252	功能 “%s” 中的参数太少	给出的参数少于功能定义中所说明的参数。
4253	“IF” 或 “ELSIF” 需要 “BOOL” 表达式作为条件	确实弄清楚, IF 或 ELSIF 的条件是一个布尔表达式。
4254	“WHILE” 需要 “BOOL” 表达 式作为条件	确实弄清楚, “WHILE” 后的条件是一个布尔表达式。
4255	“WHILE” 需要 “BOOL” 表达 式作为条件	确实弄清楚, “UNTIL” 后的条件是一个布尔表达式。
4256	“NOT” 需要 “BOOL” 表达 式作为条件	确实弄清楚, “NOT” 后的条件是一个布尔表达式。
4257	“FOR” 语句的变量必须是类型 “INT”	确实弄清楚, 计数器变量是一个整数或位串数据类型 (例如 DINT, DWORD)。
4258	“FOR” 语句中的表达式没有带 写存储的变量	使用一个有写存储的变量来替换计数器变量。
4259	“FOR” 语句中的结束值必须是 类型 “INT”	“FOR” 指令中的起始值必须与计数器变量类型相兼容。
4260	“FOR” 语句的结束值必须是类 型 “INT”	“FOR” 指令中的结束值必须与计数器变量类型相兼容。
4261	“FOR” 语句的增量值必须是类 型 “INT”	“FOR” 指令中的起始值必须与计数器变量类型相兼容。
4262	“EXIT” 在一个循环之外	只能在 “FOR”, “WHILE” 或 “UNTIL” 指令内使用 “EXIT”。
4263	期望的数, “ELSE” 或 “END-CASE”	在一个 “CASE” 表达式内, 你只能使用一个数或一个 “ELSE” 指令, 或结束指令 “END_CASE”。
4264	“CASE” 要求一种整数类型的 选择器。	确实弄清楚, 选择器是一种整数或位串数据类型的 (例如 DINT, DWORD)。

4265	“ , ” 号后预期的数	在 CASE 选择器的枚举时, 在逗号后面必须插入一个其它选择器。
4266	至少需要一个语句	插入一个指令, 至一个分号。
4267	功能块调用需要功能块实例	功能块调用中的标识符没有实例。说明所需要的功能块的一个实例, 或利用一个早已定义的实例。
4268	预期的表达式	在这里插入一个表达式。
4269	“ELSE”分支后面预 “END_CASE”	使用一个“END_CASE”来终止“ELSE”部分后面的“CASE”指令。
4270	早已使用了“CASE”常数“%ld”	一个“CASE”选择器在一个“CASE”指令内只能使用一次。
4271	范围的下边界大于其上边界	修改选择器的区域边界, 使其下部边界不大于上部边界。
4272	在调用“%s”时, 预期的参数 “%s”在位置%d!	你可以这样的方式编辑一个功能调用, 使它还包含参数名, 而不仅包含参数值。但是, 无论如何, 参数的位置(顺序)必须与功能定义中相同。
4273	“CASE”范围“%ld…%ld”部分 早已用于范围“%ld…%ld”	确实型清楚, 用于 CASE 指令中的选择器区域不出现重叠。
4274	“CASE”语句中的多重“ELSE” 分支	一个 CASE 指令不能包含多于一个“ELSE”指令。
4300	跳转需要“BOOL”作为输入类 型	确实型清楚, 相应于 RETURN 指令的跳转输入是一个布尔表达式。
4301	POU “%s”需要精确的 %d 输 入	输入数不对应于在 POU 定义中给出的 VAR_INPUT 和 VAR_IN_OUT 变量数。
4302	POU “%s”需要精确的 %d 输 出	输出数不对应于在 POU 定义中给出的 VAR_OUTPUT 变量数。
4303	“%s”无操作符	使用一个有效的操作符来替换 <name>。
4320	与接点一起使用时, 没有布尔表 达式“<name>”	用于一个接点的开关信号必须是一个布尔表达式。
4321	与线圈一起用时, 没有布尔表 达式<name>	一个线圈的输出变量必须是类型 BOOL。
4330	在框<name>的输入“EN”处 的预期表达式	将一个输入或一个表达式分配给 POU “<name>”的输入 EN。
4331	在框“<name>”的输入 “<number>”预期的表达式	没有分配操作符 POU 的输入 <number>
4332	在框“<name>”的输入<name> 处的预期表达式	POU 的输入是类型 VAR_IN_OUT 而且没有分配。
4333	预期在跳转时的标识符	给出的跳转标记不是一个有效的标识符。
4334	在跳转的输入处预期的表达式	将一个布尔表达式分配给 RETURN 指令的输入。如这是 TRUE, 则将执行这个跳转。
4335	在返回的输入处预期的表达式	将一个布尔表达式分配给 RETURN 指令的输入。如这是 TRUE, 则将执行这个跳转。
4336	在输出的输入处预期的表达式	将一个合适的表达式分配给输出框
4337	预期的输入标识符	在输入框中插入一个有效的表达式或标识符。
4338	框“%s”没有输入	对操作符 POU <name> 输入分配有效的表达式。
4339	在输出处的类型失配: 不能将 “<name>”转换成“<name>”	输出框中的表达式类型与应分配给它的表达式类型不兼容。

4340	跳转需要“BOOL”作为输入类型	确实弄清楚，跳转的输入是一个布尔表达式。
4341	返回需要“BOOL”作为输入类型	确实型清楚，用于 RETURN 指令的输入是一个布尔表达式。
4342	在框〈name〉的输入“EN”处的预期的表达式	将一个有效的布尔表达式分配给框的 EN 输入。
4343	常数值：“〈name〉”	框“〈name〉”的输入“〈name〉”作为 VAR_INPUT CONSTANT 说明。但是，在对话框“编辑参数”中，已将一个类型不兼容的表确定式发配给这个 POU 框
4344	“S”和“R”需要“BOOL”操作数	在相应的“复位”指令的设定后面，插入一个有效的布尔表达式。
4345	无效的“〈name〉”的参数“〈name〉”类型：不能将“〈type〉”转换为“〈type〉”	分配给 POU 框“〈name〉”的输入“〈name〉”的一个表达式，类型是不兼容。
4346	不允许将一个常数作为一个输出	你只能将一个输出分配给一个变量或一个有写存取的直接地址。
4347	“VAR_IN_OUT”参数需要带有写存取的变量作为输入	只有有写存取的变量才可移交给 VAR_IN_OUT 参数，这是因为这些变量可在 POU 内修改。
4350	不能从外部存取一个 SFC 动作！	SFC 动作只能在它们被定义的 SFC POU 内才能调用。
4351	步名无标识符：“〈name〉”	重新命名步，或选择一个有效的标识符作为步名。
4352	附加字符后跟有效步名：“〈name〉”	在步名中除去无效的字符。
4353	步名重复：“〈name〉”	重新命名一个步名。
4354	跳转到未定义的步：“〈name〉”	选择一个现有的步名作为相应的跳转的目标。插入一个有名的步，“〈name〉”
4355	一次转换不应有任何副作用（分配，FB 调用等）	一次转换必须是一个布尔表达式。
4356	跳转没有有效步名：“〈name〉”	使用一个有效标识符作为跳转目标（标记）。
4357	找不到 IEC 库	检查一下，库 iecsf.lib 是否插入在库管理程序内，以及在“Project”（项目）→“Options”（选项）→“Paths”（路径）中规定的库路径是否正确。
4358	来说明动作：“%s”	确实弄清楚，在对象组织程序内，IEC 步的动作是在 SFC POU 下面插入的，而且在编辑器中，动作名插入在限定符右边的框内。
4359	无效的限定符：“%s”	在动作名左边的框内，输入一个用于 IEC 动作的限定符。
4360	限定符“%s”后预期的时间常数	紧邻动作名左侧的框，在一个限定符后输入一个时间常数。
4361	“%s”不是一个动作的名	紧邻限定线右侧的框，输入一个动作名，或在项目中定义的一个变量名。
4362	非布尔表达式用于动作：“%s”	插入一个布尔变量或一个有效的动作名。
4363	IEC—步名早已用于变量：“〈name〉”	请重新命名步或变量。
4364	一个转换必须是一个布尔表达式。	转换表达式的结果必须是类型 BOOL。
4365	限定符“〈name〉”后预期的时间常数	打开步“〈name〉”的对话框“步属性”，并输入一个有效的的时间变量或时间常数。

4366	并行分支的标记是无效的标识符：“〈name〉”	在三角形符号旁输入一个有效的标识符，它标记跳转符。
4367	早已使用过的标记“〈name〉”	早已有一个跳转符或一个步使用过这个名。请相应地重新命名。
4368	动作“〈name〉”用于多重步链，在步链中，动作名彼此包含！	动作“〈name〉”用于 POU 以及 POU 的一个或几个动作。
4369	准确地说一个网络需要一个转换	对于一个转换使用了几个 FBD 相应的 LD 网络。请减少到1个网络。
4370	在正确的 IL 转换后，发现附加的线路。	在转换结束处除去不必要的线路。
4371	有效表达式后的无效的字符：“〈name〉”	在转换结束处除去不必要的字符。
4400	POU “%s” 的输入/转换包含错误，相关部分不完全。	POU 不能完全转换到 IEC61131-3，例如 MAS 那样的 CPU 命令。
4401	S5 时间常数%lu 秒太长（最大9990s）	在累加器内没有有效的 BCD 编码时间。
4402	只允许对 I/O 的直接存取	确实弄清楚，你只存取定义为输入或输出的变量。
4403	STEP5/7 指令无效或不能转换成 IEC61131-3	有些 STEP5/7 命令不能转换到 IEC61131-3，相应地丢失一个操作数。
4404	STEP5/7 操作数无效或不能转换成 IEC61131-3	有些 STEP5/7 操作数不能转换到 IEC 61131-3，相应地丢失一个操作数。
4405	一个 STEP5/7 定时器的复位，不能转变成 IEC 61131-3	相应的 IEC 定时器没有复位输入。
4406	STEP5/7 计数器常数超出范围（最大999）	累加器中没有有效的 BCD 编码的计数器常数。
4407	STEP5 指令，不能转换成 IEC61131-3	有些 STEP5/7 指令不能转换成 IEC61131-3，例如 DUF。
4408	定时器或计数器的位存取不能转换成 IEC61131-3	专用的定时器/计数器命令不能转换成 IEC61131-3
4409	未规定 ACCU1 或 ACCU2 的内容，不能转换成 IEC 61131-3	与两个累加器相连接的一个命令是不能转换的，这是因为未规定累加器的值。
4410	被调用的 POU 不在项目内	输入被调用的 POU。
4411	全局变量表内有错误	请检查 SEQ 文件。
4412	内部错误 no 11	请与 PLC 制造商联系。
4413	数据块中的行格式错误	应输入的代码中有一个错误的日期。
4414	FB/FX 名丢失	在需始的 S5D 文件中，丢失一个（扩展的）POU 的符号名。
4415	不允许块结束后的指令	不能输入一个受保护的 POU。
4416	无效的命令	S5/S7 命令不能被取消。
4417	注释未关闭	使用“*”) 来关闭注释。
4418	FB/FX 名太长（最大8字符）	一个（扩展的）POU 的符号名太长。
4419	期望的行格式““（*名：〈FB/FX-Name〉””	校正相应的行。
4420	丢失 FB/FX 参数	检查 POU。
4421	FB/FX 参数类型无效	检查 POU。

4422	丢失 FB/FX 参数类型	检查 POU。
4423	无效的 FB/FX 调用参数	检查 POU 的接口。
4424	报警: FB/FX 调用, 或是丢失, 或是参数无效或有“0”参数	被调用的 POU 尚未输入, 或不正确, 或无参数 (在最后一场合, 你可忽略出错消息)。
4425	丢失标记的定义	未定义跳转的目标 (标记)。
4426	POU 没有一个有效的 STEP 5 块名, 例如 PB10	修改 POU 名。
4427	未说明定时器类型	在全局变量表中加入一个定时器的说明。
4428	超过打开的 STEP5 括号的 最大数	不允许使用多于七个的开括号。
4429	形式参数名有误	参数名不能超过四个字符。
4430	形式参数的类型不是 IEC 可 转换的	在 IEC 61131-3 中, 定时器, 计数器和 POU 不能转换为形式参数。
4431	对于 STEP5 STL 中的一个调 用, VAR_OUTPUT 的参数太 多。	一个 POU 不能包含多于16个形式参数作为输出。
4432	表达式中的标记是不允许的。	在 IEC 61131-3 中, 跳转标记不能插入在任何需要的位置。
4434	太多的标记	一个 POU 不能包含多于100个标记。
4435	跳转/调用后, 必须开始一个 新的表达式	跳转或调用后, 必须后随一个“Load” (装入) 命令 LD。
4436	未定义位结果, 不能转换到 IEC 61131-3	VKE 使用的命令不能转换, 这是因为, VKE 的值是未知的。
4437	指令和操作数的类型不兼容	一个值命令用于一个字操作数 (或反过来也如此)。
4438	没有打开数据块 (以前插入指令 C DB)	插入一个“ADB”
4500	无法识别的变量或地址	在项目内未说明监视变量。通过按〈F2〉按钮, 你得到列出说明变量的输入辅助。
4501	附加字符后随有效的监视表 达式	除去多余的记号。
4520	附注中的错误: 期望的标记在 〈name〉!	附注不正确。检查一下, “〈name〉” 是否是一个有效的标记。
4521	附注中的错误: 不期望的图形 元素 〈name〉	检查一下, 附注是否正确编写。
4522	期望的“flag off” (除去标 记) 附注!	丢失断开附注, 添加一个“flag off” (除去标记) 指令。
4550	索引超出规定范围: 变量 OD 〈number〉, Line 〈line number〉	保证索引是在目标设定/网络功能度中所规定的区域内。
4551	子索引超出规定范围: 变量 OD 〈number〉, Line 〈line number〉	保证子索引是在目标设定/网络功能度中所规定的区域内。
4552	索引超出规定范围: 参数 OD 〈number〉, Line 〈line number〉	保证索引是在目标设定/网络功能度中所规定的区域内。
4553	子索引超出规定范围: 参数 OD 〈number〉, Line 〈line number〉	保证子索引是在目标设定/网络功能度中所规定的区域内。
4554	变量名无效: 变量 OD 〈number〉, Line 〈line number〉	保证在字段“variable” (变量) 中的一个有效项目变量。使用相应于全局变量〈variable name〉的语法〈POU 名〉〈变量名〉

4555	空表登录项, 输入不任选: 参数 OD <number>, Line <line number>	你必须在这个字段作出一个登录项。
4556	空表登录项, 输入不任选: 变量 OD <number>, Line <number>	你必须在这个字段作出一个登录项。

8.7 命令行命令

命令行/命令文件命令

命令行命令

当启动 TwinCAT PLC Control 时，你可以将命令加入到命令行中，在程序执行过程中将这类命令插入在程序内。这些命令以一个“/”符号开始。不考虑使用大小写。命令的执行顺序是从左到右。

命令	说明
/调试	
/联机	
/运行	
/显示…	可作出 TwinCAT PLC Control 帧窗口的设置。
/显示隐藏	窗口不显示，它也不在任务菜单内表示。
/显示图标	窗口在显示时将最小化。
/显示最大	窗口在显示时将最大化。
/显示正常	窗口将显示它在上次关闭期间相同的状态。
/超出 (超出文件)	所有消息都在消息窗口内显示，而且写入在文件 (超出文件) 内。
/cmd (cmd文件)	启动 (cmd文件) 命令后，执行这项命令。

一个命令行的输入，其结构有如下的形式：

“ <Path ofTwinCAT PLC Control-Exe file> ” “ <Path of the project> ” / <Command1> / <Command2> …

一个命令行的例子：

“D:\dir1 TwinCAT PLC Control”、“C:\projects\ampel.pro” /show hide /cmd command.cmd

项目 ampel.pro 打开，但没有打开的窗口。将执行包括在命令文件 command.cmd 内的各个命令。

命令文件 (cmdfile) 命令

参阅以下的命令表，它可以用于一个命令文件 (<cmdfile>) 内。通过命令行 (见上面) 调用命令文件。不考虑使用大小写字母书写。命令行作为一个内显示，可在消息文件中给出 (见下面)。附加于命令的是一个前缀“@”。忽略分号 (;) 后面的符号 (注释)。

联机菜单命令：

命令	说明
联机登入	使用装入的项目进行登入 (“Online Login” (联机登入))
联机登出	登出 (“Online” (联机) → “Logout” (登出))
联机运行	启动应用程序 (“Online” (联机) → “Run” (运行))
联机仿真	接通仿真方式 (“Online” (联机) → “Simulation” (仿真))
联机断开	断开仿真模式 (“Online” (联机) → “Simulation” (仿真))

文件菜单的命令:

命令	说明
文件新	建立一个新项目 (“File” (文件) → “New” (新))。
文件打开 (项目文件)	将装入项目 (项目文件) (“File” (文件) → “Open” (打开))。
文件关闭	将关闭当前的项目 (“File” (文件) → “Close” (关闭))。
文件保存	将保存当前的项目 (“File” (文件) → “Save” (保存))。
文件另存为 (项目文件)	当前的项目将应用文件名 (项目文件) (“File” (文件) → “Save as” (另存为)) 保存。
文件退出	将关闭 TwinCAT PLC Control (“File” (文件) “Exit” (退出))。

项目菜单的命令:

命令	说明
项目编译	使用 “Rebuild All” (全部重建) → (“Project” (项目) → “Rebuild All” (全部重建)) 编译当前的项目。
项目检查	检查当前的项目 (“Project” (项目) → “Check” (检查))。
项目建立	建立当前的项目 (“Project” (项目) → “Build” (建立))。
项目输入 (文件1) … (文件N)	将文件 (文件1) … (文件N) 输入到当前项目内 (“Project” (项目) → “Import” (输入))。
项目输出 (输出文件)	将当前的项目输出到文件 (expfile) 内 (“Project” (项目) → “Export” (输出))。
项目 expmul (输出文件)	将当前项目的每个对象输出到一个自用文件内。文件使用该项目的名称。

用于消息文件控制的命令:

命令	说明
out 打开 (msg 文件)	文件 (msgfile) 作为消息文件打开。将附加有新的消息。
out 关闭	将关闭当时显示的消息文件。
out 清除	将删除当前打开消息文件的所有消息。

用于消息控制的命令

命令	说明
回送开放	命令行将作为消息显示
回送关闭	命令行将不作为消息显示。
回送 (文本)	将在消息窗内显示 (文本)。

相应于输入, 输出, 替代文件控制的用于对象替代控制的命令:

命令	说明
替代 OK 替代 yes	替代
替代 no	不替代
替代 noall	一个也不替代
替代 yesall	替代所有的

对话框默认参数控制命令：

命令	说明
查询 接通	显示对话框并需要用户输入。
查询 断开 OK	所有对话框如同用户已在“OK”按钮上点击那样响应。
查询 断开 NO	所有对话框如同用户已在“No”按钮上点击那样响应。
查询 断开 取消	所有对话框如同用户已在“Cancel”（删除）按钮上点击地那样响应。

调试命令：

命令	说明
调试	相当于命令行中的命令“/debug”（/调试）

调用子程序的命令文件命令：

命令	说明
〈参数1〉 … 〈参数10〉	调用作为子程序的命令文件。最多可提交10个参数。在调用的子例程中，你可用\$0-\$9存取这些参数。

库设定命令：

命令	说明
dir lib 〈libdir〉	〈libdir〉设置作为库目录
dir compile 〈compiledir〉	〈compiledir〉设置作为编译文件的目录。

设定与 CMDFILE 执行有关的延迟时间命令：

命令	说明
延时 5000	等待5秒

监视和接收管理器控制命令：

命令	说明
装载监视表 〈文件〉	将装载保存在〈文件〉内的监视文件，并打开相应的窗口（“Extras”（附加）→“Load Watch List”（装载监视表））。
保存监视表 〈文件〉	将当前的监视表保存在〈文件〉内（“Extras”（附加）→“Save Watch List”（保存监视表））。
设定监视表 〈文本〉	对以前装载的监视表改名〈文本〉（“Extras”（附加）→“Rename Watch List”（重新命名监视表））。
读监视表	更新监视变量值（“Extras”（附加）→“Read Receipt”

	(读接收))。
写监视表	将监视表的值写入到监视变量 (“Extras” (附加) → “Write Receipt” (写接收))

链接库命令:

命令	说明
附加库 <库文件1> <库文件2> ... <库文件 N>	将特定的库文件附加到当前打开的项目库表。如果文件路径是一种相对路径, 则输入到项目中的库目录就作为路径的根源。
删除库 <库1> <库2> ... <库 N>	删除特定的库, 或从当前打开的项目库表中全部删除库 (如没有规定库名的话)。

复制对象命令:

命令	说明
对象复制 <源项目文件>、<源路径>、<目标路径>	将对象从源项目文件的规定路径复制到已打开的项目目标路径。如果源路径是一个对象名, 则复制这个名。若它是一个文件夹, 则复制这个文件夹下的所有对象。在某种场合, 复制源文件夹下的文件夹结构。如尚不存在目标路径, 则建立这个路径。

调用系统命令:

命令	说明
系统 <命令>	完成规定的系统命令。

联机方式命令:

命令	说明
复位	如果你已用一个特定值将变量初始化, 该命令就会将变量复位到该初始值。
全部复位	该命令将包括保持型变量在内的所有变量复位到它们的初始化值, 并擦除控制器上的用户程序。
建立引导项目	该命令在控制器上建立编译项目。
选择运行系统 <文本>	可选择一個未用的 “Runtime System” (运行系统)。对于 <文本> 使用 Net-Id (网络标识符) 并以 “: ” 分隔端口号。 例子: 选择运行系统 172.16.77.23.1.1: 811

命令文件举例:

如下所示的命令文件将打开项目文件 `ampel.pro`, 然后装入一个作为 `w.wtc` 保存的监视表, 接着启动应用程序并将变量值 (经一秒钟延时后) 写入到监视表 `watch.wtc` (该表将被保存), 最后关闭项目。

```
打开文件 C:\work\projects\ampel.pro  
查询断开 OK  
装载监视表 c:\work\w.wtc  
联机登入  
联机运行  
延时 1000  
建立引导项目  
读取监视表  
保存监视表 c:\work\watch.wtc  
联机登出  
文件关闭
```