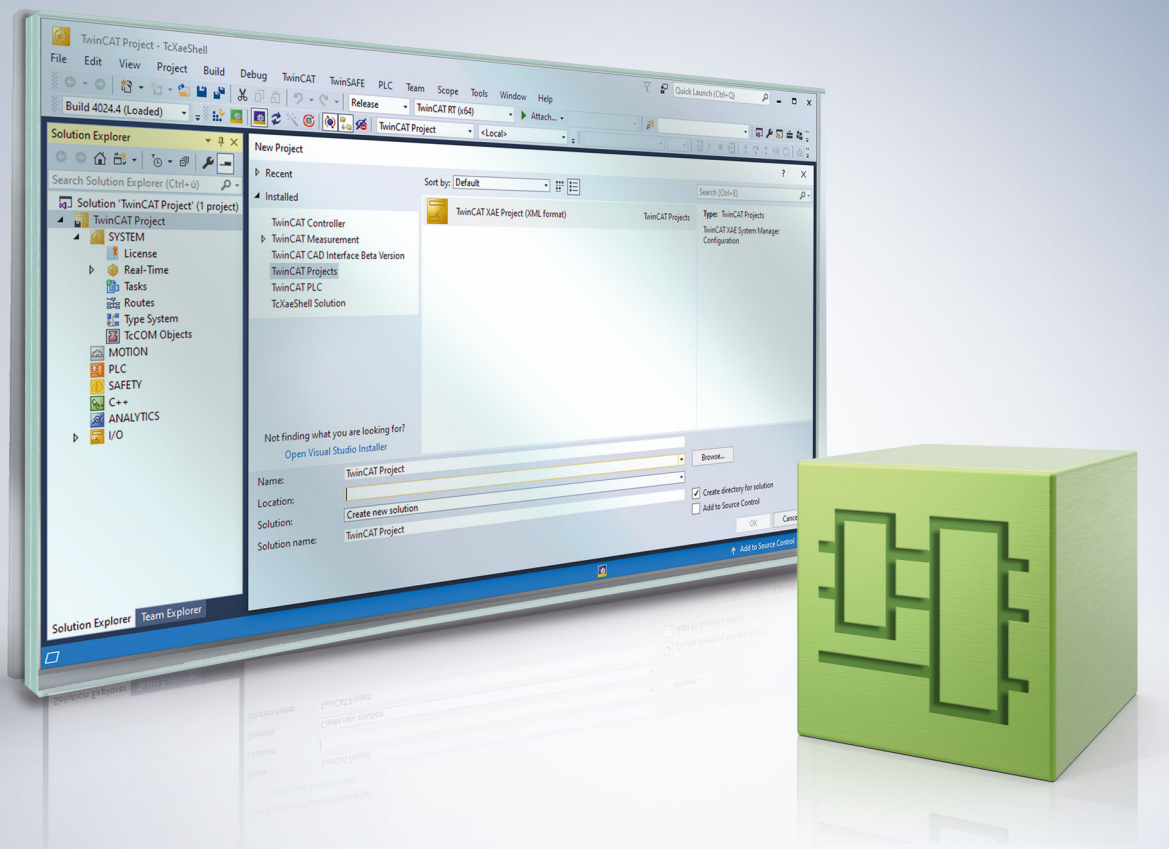
BECKHOFF New Automation Technology

手册 | ZH

TE1000

TwinCAT 3 | PLC



2024-11-29 | 版本： 3.1.0

目录

1 前言

1.1 文档说明

本说明仅适用于熟悉国家标准且经过培训的控制和自动化工程专家。
在安装和调试组件时，必须遵循文档和以下说明及解释。
操作人员应具备相关资质，并始终使用最新的生效文档。

相关负责人员必须确保所述产品的应用或使用符合所有安全要求，包括所有相关法律、法规、准则和标准。

免责声明

尽管本文档经过精心编制， 然而，所述产品正在不断开发中。
我们保留随时修订和更改本文档的权利，恕不另行通知。
不得依据本文档中的数据、图表和说明对已供货产品的修改提出赔偿。

商标

Beckhoff®、TwinCAT®、TwinCAT/BSD®、TC/BSD®、EtherCAT®、EtherCAT G®、EtherCAT G10®、EtherCAT P®、Safety over EtherCAT®、TwinSAFE®、XFC®、XTS® 和 XPlanar® 是德国倍福自动化有限公司的注册商标并已获得授权。
本文档中所使用的其它名称可能是商标名称，任何第三方为其自身目的而引用，都可能触犯商标所有者的权利。

正在申请的专利

涵盖 EtherCAT 技术，包括但不限于以下专利申请和专利：
EP1590927、EP1789857、EP1456722、EP2137893、DE102015105702
并在多个其他国家进行了相应的专利申请或注册。

EtherCAT® 是注册商标和专利技术，由德国倍福自动化有限公司授权使用。

版权所有

© 德国倍福自动化有限公司。
未经明确授权，不得复制、分发、使用和传播本文档内容。
违者将被追究赔偿责任。德国倍福自动化有限公司保留所有发明、实用新型和外观设计专利权。

1.2 安全信息

安全规范

为了确保您的使用安全，请务必仔细阅读
并遵守本文档中每个产品的安全使用说明。

责任免除

所有组件在供货时都配有适合应用的特定硬件和软件配置。严禁未按文档所述修改硬件或软件配置，否则，德国倍福自动化有限公司对由此产生的后果不承担责任。

人员资格

本说明仅供熟悉适用国家标准的控制、自动化和驱动工程专家使用。

警示性词语

文档中使用的警示信号词分类如下。为避免人身伤害和财产损失，请阅读并遵守安全和警告注意事项。

人身伤害警告

⚠ 危险

存在死亡或重伤的高度风险。

⚠ 警告

存在死亡或重伤的中度风险。

⚠ 谨慎

存在可能导致中度或轻度伤害的低度风险。

财产或环境损害警告

注意

可能会损坏环境、设备或数据。

操作产品的信息

这些信息包括：
有关产品的操作、帮助或进一步信息的建议。

1.3 信息安全说明

Beckhoff Automation GmbH & Co.KG（简称 Beckhoff）的产品，只要可以在线访问，都配备了安全功能，支持工厂、系统、机器和网络的安全运行。尽管配备了安全功能，但为了保护相应的工厂、系统、机器和网络免受网络威胁，必须建立、实施和不断更新整个操作安全概念。Beckhoff 所销售的产品只是整个安全概念的一部分。客户有责任防止第三方未经授权访问其设备、系统、机器和网络。它们只有在采取了适当的保护措施的情况下，方可与公司网络或互联网连接。

此外，还应遵守 Beckhoff 关于采取适当保护措施的建议。关于信息安全和工业安全的更多信息，请访问本公司网站 https://www.beckhoff.com/secguide。

Beckhoff 的产品和解决方案持续进行改进。这也适用于安全功能。鉴于持续进行改进，Beckhoff 明确建议始终保持产品的最新状态，并在产品更新可用后马上进行安装。使用过时的或不支持的产品版本可能会增加网络威胁的风险。

如需了解 Beckhoff 产品信息安全的信息，请订阅 https://www.beckhoff.com/secinfo 上的 RSS 源。

2 快速入门

系统概览

TwinCAT 3 PLC 在 1 台 CPU 上采用 IEC 61131-3 第 3 版国际标准实现 1 个或多个 PLC。标准中描述的所有编程语言都可用于编程。PROGRAM 类型的块可以与实时任务相关联。各种方便的调试选项使得故障查找和调试变得简单。随时可以对程序进行任何规模的在线修改，即在 PLC 正在运行时。所有变量都可以通过 ADS 以符号方式使用，并且可以在相应的客户端中读写。

功能

TwinCAT 3 PLC 为您的开发工作提供多样化且方便的工程功能：

	另请参见本在线帮助：
通过向导进行项目配置	创建和配置项目 [▶ 48]
界面定制	定制界面
根据 IEC 61131-3 标准生成具有多种标准功能的专业控制程序	对 PLC 项目进行编程 [▶ 60]
使用鼠标和键盘以所有 IEC 61131-3 语言轻松编程 FBD、LD、IL、ST、SFC 以及 CFC 变体和扩展 CFC 的相应编辑器	编程语言及其编辑器 [▶ 569]
针对多种数据的输入和配置的输入支持	使用输入向导 [▶ 123]
支持面向对象的编程 符合 61131-3 第 3 版标准的真正面向对象的编程，可使用所有 IEC 61131-3 语言，无需额外工具 将程序块继承到类似的程序部分，以减少开发时间和错误 不一定要使用面向对象的编程：可以根据需要使用和混合使用函数式编程或面向对象的编程	面向对象的编程 [▶ 158]
全面的项目比较，也适用于图形编辑器	比较项目
方便重复使用程序代码的库概念	使用库 [▶ 246]
调试和在线功能可优化程序代码，并加快测试和调试的速度	测试和调试 [▶ 195]
适用于各种 CPU 平台的集成编译器	项目属性 - 类别编译 [▶ 844]
保护源代码和操作控制器的安全属性	保护和保存项目 加密 PLC 项目

使用帮助

每个帮助组件都由概念部分和参考部分组成。概念部分详细解释了与创建、管理和执行 TwinCAT 3 PLC 项目相关的所有主题。在进行描述的同时，还提供了实现预期效果的逐步说明。参考部分提供了有关 TwinCAT 3 PLC 用户界面和编程的完整参考资料。另请参见“TC3 用户界面”文档。

2.1 与 TwinCAT 2 的区别

库版本

- 在同一个项目中，1 个库可以有多个版本。通过指定命名空间，可确保明确的访问。
- 在存储库（带有不同版本库的数据库）中安装
- 自动更新
- 可进行调试
- 库配置文件和占位符便于实现版本兼容性

与其他文件格式的项目兼容

- 命令 **Add Existing Item …**（添加现有项目……）可用于将不同格式的项目自动转换为 TwinCAT 3 格式。您可以规定处理集成库的方式。
- TwinCAT 2 格式转换器是标准编程系统的一部分。但是，如果您想要在 TwinCAT 3 中使用 TwinCAT 2 项目，则必须考虑一些前提条件和限制。

数据类型

新增了以下数据类型：

- any_type
- UNION
- LTIME
- BIT
- 引用
- 枚举：可指定基本数据类型。
- 不允许 di : DINT := DINT#16#FFFFFFFF :

编辑器

ST 编辑器：

- 括号、断句、代码补全、内联监控、内联设置/重置赋值

FBD/LD/IL 编辑器：

- FBD、LD 和 IL 可以相互转换，它们有 1 个通用的编辑器。
- 作为表格编辑器的 IL 编辑器
- 在具有多输出的功能块中可以设置主输出。
- 分支和“网络中的网络”

SFC 编辑器：

- 只有 1 种步骤类型，宏，独立元素的多项选择，编辑期间没有语法检查

可视化概念

- 在默认情况下，可视化编辑器与工具箱和元素属性编辑器配合使用。
- 部分可视化功能根据 IEC 61131 标准实现，因此可通过库提供。内部运行时系统执行主要的可视化功能。
- 文本列表和图像池用于管理文本和图像文件。
- 可视化管理器可处理不同的可视化客户端（如网络客户端、独立客户端……），这些客户端可灵活使用，并可轻松用于各种自定义 PLC 项目。
- 可视化配置文件可定义当前可用的库版本和元素。
- 可视化样式便于调整可视化的“外观”。

运算符和变量

- 新的有效范围运算符，扩展的命名空间
- Init 方法取代 INI 运算符
- 退出方法
- 函数和方法调用中的输出变量
- VAR_TEMP，VAR_STAT
- 变量初始化的任何表达式
- 赋值可作为表达式
- 使用指针和字符串的索引访问

面向对象

- 功能块扩展：属性、界面、方法、继承、方法调用

其它参数

- 可配置菜单、工具栏和键盘操作
- 支持 Unicode
- 单行注释：// 注释
- 在循环中继续
- 条件编译
- 条件断点
- 调试：运行到光标处，跳出

从 TwinCAT 2 PLC 起的变更：

- FUNCTIONBLOCK 不再是替代 FUNCTION_BLOCK 的功能块的有效关键字
- TYPE（结构声明）后必须带有 1 个“:”。
- ARRAY 初始化必须用括号括起来。
- 不再支持 INI，必须将其替换为 Init 方法。
- 在函数调用中，不再可能将显式参数赋值与隐式赋值混用。因此，参数输入赋值的顺序可以更改：

```
fun(formal1 := actual1, actual2); // → Fehlermeldung
fun(formal2 := actual2, formal1 := actual1); // gleiche Semantik wie ...
fun(formal1 := actual1, formal2 := actual2);
```

- 编译指示（尚未实现 TwinCAT 2 编译指示的导入）
- TRUNC 运算符现在可以转换为数据类型 DINT，而不是 INT；在 TwinCAT 2 导入过程中会自动添加相应的类型转换。
- 已分配变量的直接寻址：

已分配变量的直接寻址 [▶ 695]已改变。无论数据类型如何，TwinCAT 2的起点始终相同，，而在 TwinCAT 3 中则有所区别：

TwinCAT 2：W0 包含 B0 和 B1，W1 包含 B1 和 B2，W100 包含 B100 和 B101

TwinCAT 3：W0 包含 B0 和 B1，W1 包含 B2 和 B3，W100 包含 B200 和 B201

2.2 您的第 1 个 TwinCAT 3 PLC 项目

您的第 1 个项目的内容

在本教程中，您将对 1 个简单的冰箱控制器进行编程。

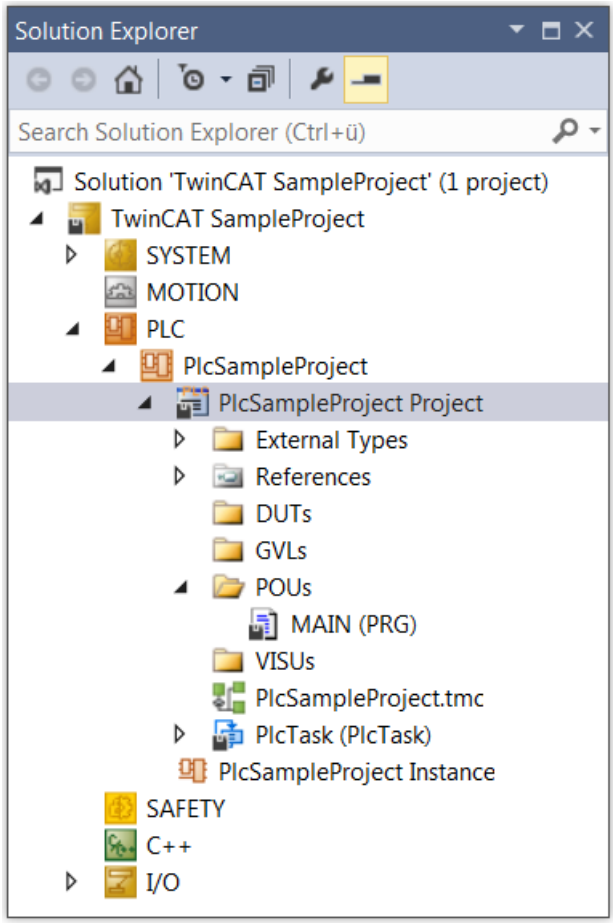
- 与传统冰箱一样，用户可通过控制旋钮设置设定温度。
- 冰箱通过传感器检测实际温度。如果实际温度过高，冰箱会通过可调延迟时间来启动压缩机。
- 压缩机冷却至设定温度减去 1 度滞后值的状态。滞后的目的是防止实际温度在设定温度附近波动过大，并防止压缩机连续开关。
- 当冰箱门打开时，冰箱内的灯会亮起。
- 如果冰箱门打开的时间过长，就会发出定时鸣响信号。
- 如果压缩机在电机未启用的情况下长时间达不到设定温度，蜂鸣器就会发出持续的鸣响信号。

项目规划：

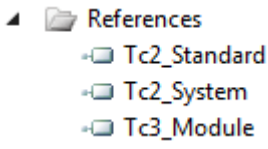
在 PLC 项目的主程序中控制冷却活动，信号管理则在另一个程序块中进行。在 Tc2_Standard 库中可提供所需的标准功能块。由于在本示例项目中没有连接真实的温度传感器和执行器，因此，您还将编写 1 个模拟温度升降的程序。这样，您就可以在在线模式下监控冰箱控制单元的运行状况。您可以在全局变量列表中定义所有功能块所使用的变量。

创建 PLC 项目

1. 在 File（文件）菜单中，选择 New > Project（新建 > 项目），创建 1 个新的 TwinCAT 项目文件。
   ⇨ 在 Solution Explorer（解决方案资源管理器）中会打开 1 个带有 TwinCAT 项目树的新解决方案。
2. 在 Solution Explorer（解决方案资源管理器）中，选择 PLC 节点和命令 Add New Item（添加新项目），以便在 TwinCAT 项目中添加 1 个 PLC 项目。
   ⇨ 对话框 Add New Item - TwinCAT <project name>（添加新项目 - TwinCAT <project name>）会打开。
3. 在类别 Plc Templates（Plc 模板）中，选择 Standard PLC project （标准 PLC 项目）模板。
4. 为项目输入名称和存储位置，然后点击 Add（添加）按钮。
   ⇨ 所选模板会自动创建 1 个 MAIN 程序，由任务调用。系统会自动将“结构化文本（ST）”选为编程语言。



在 References（引用）下，系统会自动选择带有一些重要默认库的库管理器。Tc2_Standard 库包含 IEC 61131-3 标准所描述的所有功能和功能块。



声明全局变量

首先，声明您想要在整个 PLC 项目中使用的变量。为此，您需要创建 1 个全局变量列表：

1. 在 PLC 项目树中，选择子文件夹 GVLs。
2. 在上下文菜单中，选择命令 Add > Global Variable List（添加 > 全局变量列表）。
3. 将自动输入的名称“GVL”改为“GVL_Var”。
4. 点击 Open（打开）进行确认。

⇨ 在子文件夹 **GVLs** 的 PLC 项目树中会显示对象“GVL_Var”（  ）。将打开GVL 编辑器。

⇨ 当文本视图出现时，关键字 VAR_GLOBAL 和 END_VAR 已经包含在内。

5. 点击编辑器的右侧边栏中的  按钮，激活示例的表格视图。

   ⇨ 出现 1 个空行。光标位于 **Name**（名称）列中。

6. 在 **Name**（名称）字段中输入“fTempActual”。

   ⇨ 同时，系统会在该行中自动输入范围 VAR_GLOBAL 和数据类型 BOOL。

7. 双击 **Data type**（数据类型）列中的字段。

   ⇨ 现在，该字段可供编辑，同时按钮  出现。

8. 点击该按钮并选择 **Input Assistant**（输入助手）。

   ⇨ **将打开Input Assistant**（输入助手）对话框。

9. 选择数据类型 REAL，然后点击 **OK**（确定）。

10. 在 **Initialization**（初始化）列中输入 1 个数值，例如，“8.0”。

⇨ 以同样的方式声明以下变量：

名称	数据类型	初始化	注释
fTempActual	REAL	1.0	实际温度
fTempSet	REAL	8.0	设定温度
bDoorOpen	BOOL	FALSE	门状态
tImAlarmThreshold	TIME	T#30s	压缩机运行时间，超过该时间会发出警报。
tDoorOpenThreshold	TIME	T#10s	门打开后的时间，超过该时间会发出警报。
xCompressor	BOOL	FALSE	控制信号
xSignal	BOOL	FALSE	控制信号
xLamp	BOOL	FALSE	状态消息

在 CFC 编辑器中创建冷却控制主程序

在默认创建的 MAIN 程序块中，您可以描述 PLC 程序的主要功能：当实际温度高于设定温度加上滞后值的数值时，压缩机开始工作并制冷。当实际温度低于设定温度减去滞后值的数值时，压缩机关闭。

执行以下步骤，使用实现语言“连续功能图（CFC）”描述该功能：

✓ 由于自动创建的 MAIN 程序块默认使用实现语言“结构化文本（ST）”进行创建，因此，您必须首先删除该程序。通过上下文菜单命令 **Add** > **POU...**（添加 > POU……），您可以使用实现语言“连续功能图（CFC）”创建 1 个新的 MAIN 程序。

1. 然后，双击 PLC 项目树（子文件夹 **POUs**）中的 **MAIN** 程序。

   ⇨ 通过 **MAIN** 选项卡打开 CFC 编辑器。在图形编辑器区域上方显示文本或表格形式的声明编辑器。右侧是 **Toolbox**（工具箱）视图。如果工具箱没有出现，您可以使用 **View**（视图）菜单中的命令 **Toolbox**（工具箱），将其放在桌面上。

2. 在 **Toolbox**（工具箱）视图中，点击 **Input**（输入）元素，然后，使用鼠标将其拖至 CFC 编辑器中的某个位置。

   ⇨ 无名输入 **???** 已插入。

3. 在 CFC 编辑器中，点击输入 **???**，并通过点击  打开 **Input Assistant**（输入助手）。
4. 从 **Project** > **GVLs**（项目 > GVLs）中的类别 **Variables**（变量）中，选择变量 fTempActual。
5. 选择 **OK**（确定）来确认对话框，引用全局变量 fTempActual。
6. 与步骤 3 一样，使用全局变量 rTempSet 的名称，再创建 1 个输入。
7. 创建另一个输入。
8. 点击 **???**，然后将其替换为名称 fHysteresis。

   ⇨ 由于这不是已知变量的名称，因此会出现 **Auto Declare**（自动声明）对话框。名称已包含在对话框中。

9. 使用数据类型 REAL 和初始化值“1”完成 **Auto Declare**（自动声明）对话框中的字段。
10. 点击 **OK**（确定）按钮。

    ⇨ 变量 fHysteresis 将出现在声明编辑器中。
11. 现在，添加 1 个加法功能块：在 **Toolbox**（工具箱）视图中，点击 **Function block**（功能块）元素，然后，使用鼠标将其拖至 CFC 编辑器中的某个位置。

    ⇨ 功能块将出现在 CFC 编辑器中。
12. 将 **???** 替换为 ADD。

    ⇨ ADD（加法）功能块对与之连接的所有输入端进行加法运算。
13. 连接输入端 GVL_Var.fTempSet 与 ADD 功能块：为此，点击输入端的输出连接器，并将其拖至 ADD 功能块的上部输入端。
14. 以同样的方式连接 fHysteresis 输入端与 ADD 功能块的下部输入端。

    ⇨ 现在，2 个输入端 fHysteresis 和 fTempSet 通过 ADD 相加。
15. 如果您想要移动编辑器中的元素，请点击元素中的自由空间或边框，以便选中元素（红色边框，红色阴影）。
16. 按住鼠标按钮，将元素拖至所需位置。
17. 在 ADD 功能块的右侧，创建另一个功能块。

    ⇨ 此操作旨在将“GVL_Var.fTempActual”与“GVL_Var.fTempSet”和 fHysteresis 之和进行比较。
18. 将函数 GT（大于）分配给功能块。

    ⇨ GT 功能块的工作原理如下：
    IF (oberer Eingang > unterer Eingang) THEN Ausgang := TRUE;
19. 连接输入端“GVL_Var.fTempActual”与 GT 功能块的上部输入端。
20. 连接 ADD 功能块的输出端与 GT 功能块的下部输入端。
21. 现在，在 GT 功能块的右侧，创建 1 个功能块，该功能块可根据输入条件（设置 - 重置）启动或关闭冷却压缩机。
22. 在 **???** 字段中输入名称“SR”。
23. 按 Enter 键，关闭功能块（SR_0）上方的开放输入字段。

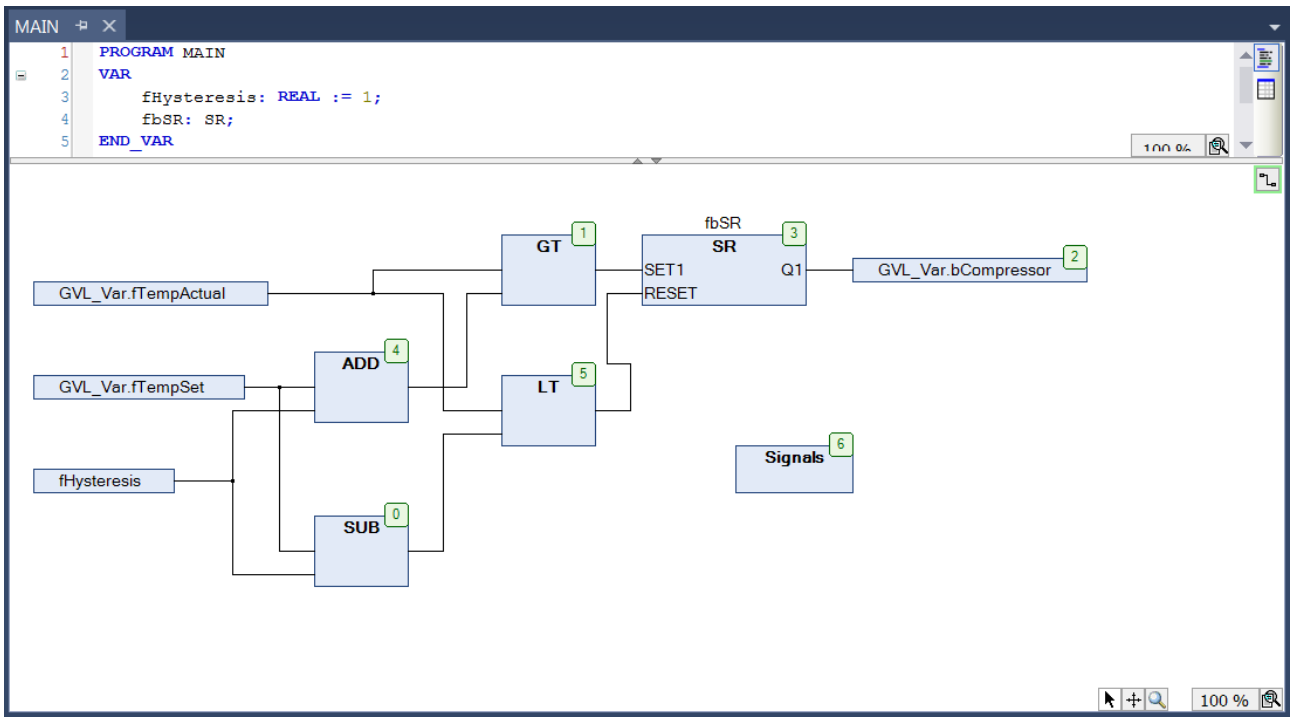
    ⇨ **Auto Declare**（自动声明）对话框就会出现。
24. 声明名称为“fbSR”且数据类型为 SR 的变量。
25. 点击 **OK**（确定）按钮。

    ⇨ SR 功能块也在 Tc2_Standard 库中进行定义，它可以决定 GT 功能块输出端的 THEN。输入端 SET1 和 RESET 会出现。
26. 连接 GT 功能块右侧的输出端连接与 fbSR 功能块的 SET1 输入端。

    ⇨ SR 可以将 1 个 Boolean 变量从 FALSE 重置为 TRUE。如果条件适用于输入端 SET1，则 Boolean 变量会被设置为 TRUE。如果条件适用于 RESET，则会重置变量。在我们的示例中，Boolean（全局）变量是“GVL_Var.bCompressor”。
27. 创建 1 个 **Output**（输出）元素，并将全局变量“GVL_Var.bCompressor”分配给它。在“GVL_Var.bCompressor”与 Q1 至 SR 的输出端连接之间画 1 条连接线。

现在，输入压缩机应再次关闭的条件，即 SR 功能块的 RESET 输入端收到 TRUE 信号的条件。为此，应提出与上述内容相反的条件。为了这个目的，可使用功能块 SUB（减法）和 LT（小于）。

创建以下 CFC 计划：



在梯形图编辑器中创建信号管理的程序块

现在，在另一个程序块中实现针对警报蜂鸣器和开关灯的信号管理。实现语言“梯形图（LD）”非常适合此用途。

为以下信号使用单独的网络：

- 如果压缩机因温度过高而长时间运行，则会发出持续的鸣响信号。
- 如果开门时间过长，则会发出脉冲信号。
- 只要门打开，灯就会亮。

1. 在 PLC 项目树（子文件夹 **POUs**）中创建 1 个**程序**类型的 POU 对象，实现语言为“梯形图（LD）”。
2. 将 POU 对象命名为“信号”。
   ⇨ “信号”会在 MAIN 下方的 PLC 项目树中出现。通过 **Signals** （信号）选项卡打开梯形图编辑器。声明编辑器会在屏幕上部出现，**Toolbox**（工具箱）视图位于右侧。LD 包含 1 个空网络。
3. 在该网络中，您可以这样编程：如果制冷压缩机运行时间过长而没有达到设定温度，则发出鸣响信号。为此，插入 1 个 TON 计时器功能块。
   ⇨ 经过指定的时间之后，它会将 Boolean TRUE 信号切换为 TRUE。
4. 在 **Toolbox**（工具箱）视图中，在 **Function blocks**（功能块）下选择 1 个 TON，然后将其拖至空网络中显示的 **Start here**（从此处开始）矩形中。
5. 当该区域变绿时，松开鼠标按钮。
   ⇨ 功能块显示为矩形，带有输入端和输出端，并会自动指定实例名称 TON_0。行编辑器会打开，光标闪烁。
6. 按 Enter 键，确认实例名称。
   ⇨ **Auto Declare**（自动声明）对话框会打开。
7. 声明名称为“fbTimer1”且数据类型为 TON 的变量。
8. 点击 **OK**（确定）按钮。

如果您想要阅读功能块的帮助信息，请用光标标出该功能块的完整名称并按 [F1]。

9. 如要实现这样的编程效果：在冷却压缩机开始运行后，立即激活功能块，可将功能块上部输入端的触点命名为“GVL_Var.bCompressor”。
   ⇨ 您在全局变量列表“GVL_Var”中已定义此 Boolean 变量。

当您在输入位置开始输入变量名时，您总是会自动收到 1 个列表，该列表会列出名称以所输入的字符开头且在此时可以使用的所有变量。该功能是 TwinCAT 选项中关于智能编码的默认设置。

10. 插入您想要激活的信号：为此，从 **Toolbox**（工具箱）视图的类别 **Ladder Diagram elements**（梯形图元素）中，将 1 个 **Coil**（线圈）拖至 TON 功能块的输出端 Q。将线圈命名为“GVL_Var.bSignal”。
11. 将变量“GVL_Var.tImAlarmThreshold”添加到“fbTimer1”的 PT 输入端，以定义从激活 TON 设备开始的时间，在该时间之后应发出鸣响信号。为此，点击输入端连接右侧带边框的矩形。
12. 输入变量名。
13. 点击 TON 功能块，然后，在上下文菜单中，选择命令 **Remove unused FB call parameters**（删除未使用的 FB 调用参数）。

    ⇨ 未使用的输出 ET 已被删除。
14. 在第 2 个 LD 网络中，您可以这样编程：当开门时间过长时，就会间歇性发出鸣响信号。
15. 首先，在 PLC 项目树的子文件夹 **POUs** 中，以“功能块图（FBD）”实现语言创建 1 个**功能块**类型的新 POU 对象。
16. 将其命名为“FB_Blinker”。

    ⇨ 功能块 FB_Blinker 会在 MAIN 上方的 PLC 项目树中出现。通过 **FB_Blinker** 选项卡打开 FBD 编辑器。声明编辑器会在屏幕上部出现，**Toolbox**（工具箱）视图位于右侧。FBD 包含 1 个空网络。

    ⇨ 闪烁器由 1 个 AND 运算符和 2 个 TON 功能块实现。
17. 在 **Toolbox**（工具箱）视图中，在 **General**（常规）下选择 1 个**功能块**，然后将其拖至空网络中显示的 **Start here**（从此处开始）矩形中。
18. 当该区域变绿时，松开鼠标按钮。

    ⇨ 功能块显示为矩形，带有输入端和输出端。
19. 点击功能块中的 **???**，然后在字段中输入关键字 AND，该字段现在可供编辑。
20. 按 Enter 键进行确认。

    ⇨ 由于它是 1 个函数，因此不需要实例化。
21. 在 **Toolbox**（工具箱）视图中，在 **Function blocks**（功能块）下选择 1 个 TON 功能块，然后将其拖至网络中的 AND 功能块的输出端。
22. 当该区域变绿时，松开鼠标按钮。

    ⇨ 功能块显示为矩形，带有输入端和输出端，并会自动指定实例名称 TON_0。
23. 按 Enter 键，关闭功能块（TON_0）上方的开放输入字段。

    ⇨ **Auto Declare**（自动声明）对话框会打开。
24. 声明名称为“fbTimerA”且数据类型为 TON 的变量。
25. 点击 **OK**（确定）按钮。
26. 在 **Toolbox**（工具箱）视图中，在 **Function blocks**（功能块）下再选择 1 个 TON 功能块，然后将其拖至网络中的 TON 功能块“fbTimerA”的输出端。
27. 当该区域变绿时，松开鼠标按钮。

    ⇨ 功能块显示为矩形，带有输入端和输出端，并会自动指定实例名称“TON_0”。
28. 按 Enter 键，关闭功能块（TON_0）上方的开放输入字段。

    ⇨ **Auto Declare**（自动声明）对话框会打开。
29. 声明名称为“fbTimerB”且数据类型为 TON 的变量。点击 **OK**（确定）按钮。

    ⇨ AND 功能块的第 1 个输入端要与 Boolean 变量“bStart”连接。
30. 点击 **???**，输入变量名。
31. 按 Enter 键进行确认。

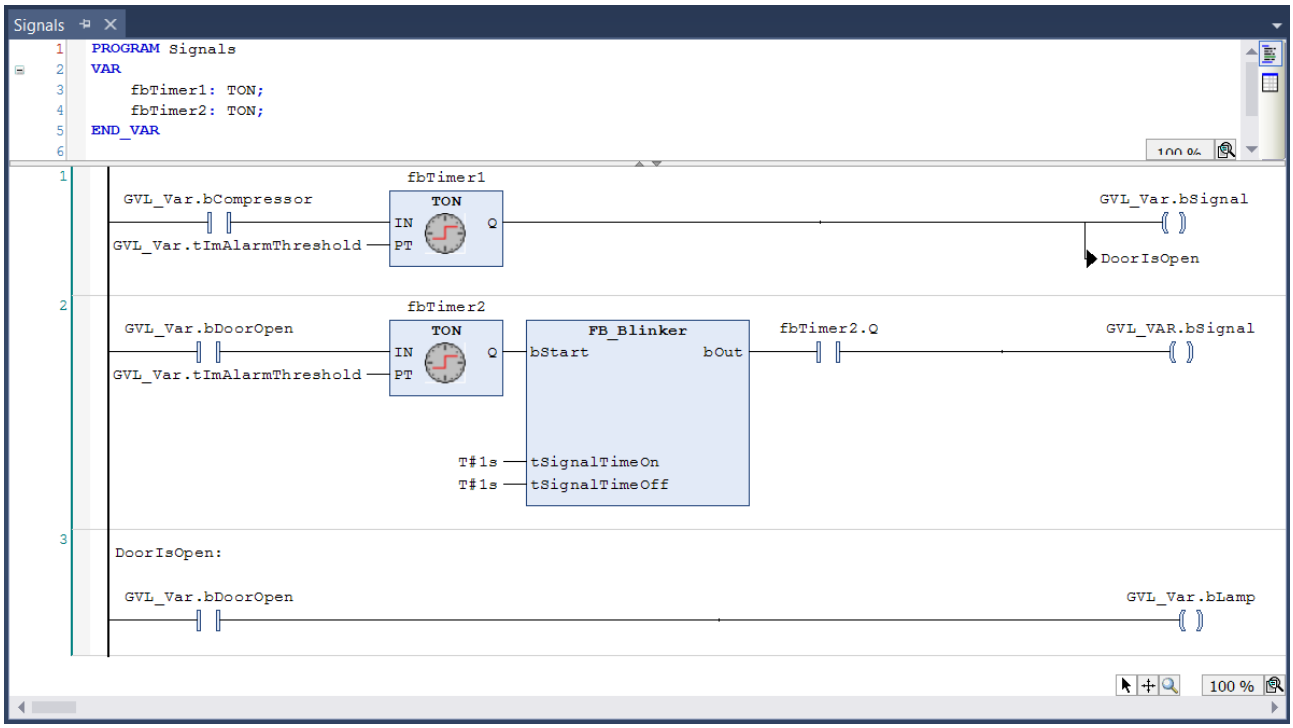
    ⇨ **Auto Declare**（自动声明）对话框会打开。系统会自动识别名称和数据类型。
32. 选择条目 VAR_INPUT 作为**范围**。
33. 选择 **OK**（确定），确认对话框。

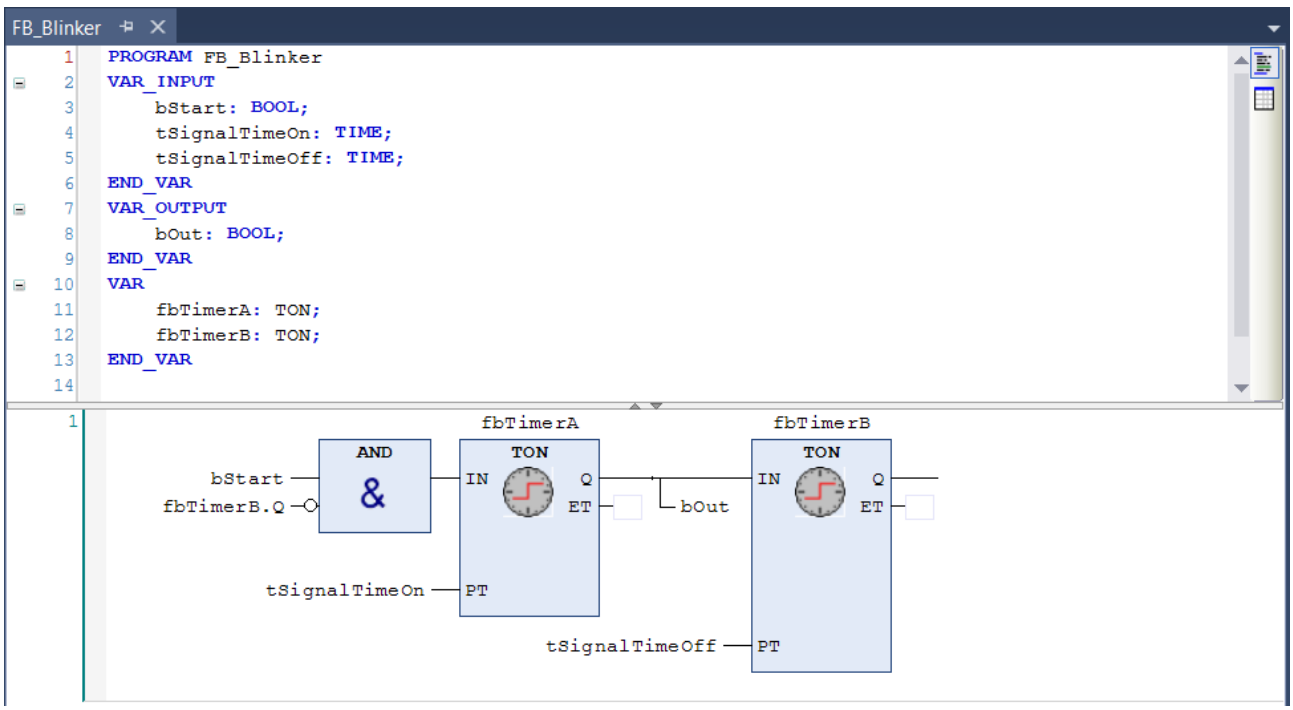
    ⇨ AND 功能块的第 2 个输入端要与第 2 个 TON 功能块“fbTimerB”的输出端 Q 连接。
34. 点击 **???**，通过 … 打开 **Input Assistant**（输入助手）。

35. 在类别 **Variables**（变量）中，选择功能块“fbTimerB”和输出端 Q。
36. 选择 **OK**（确定），确认对话框。
    ⇨ 在第 2 个输入端添加变量“fbTimerB.Q”。
37. 选择第 2 个输入端，点击鼠标右键打开上下文菜单。
38. 选择命令 **Negation**（否定），否定输入。
    ⇨ 在相应的输入端会出现 1 个圆圈。
39. 设置通过 TON 功能块的 PT 输入端设置输出端 Q 之前的时间。
40. 通过 **Auto Declare**（自动声明）对话框，为 TON 功能块“fbTimerA”和“fbTimerB”声明输入变量“tSignalTimeOn”和“tSignalTimeoff”。
41. 选择条目 VAR_INPUT 作为**范围**。
    ⇨ 在 TON 功能块“fbTimerA”的输出端 Q，输出生成的时钟脉冲。
42. 为此，在 **Toolbox** （工具箱）视图中，在 **General**（常规）下选择 1 个**赋值**，然后将其拖至网络中的 TON 功能块“fbTimerA”的输出端。
43. 当该区域变绿时，松开鼠标按钮。
    ⇨ 在功能块“fbTimerA”和“fbTimerB”之间添加赋值。
44. 点击 **???**，输入变量名“bOut”。
45. 按 Enter 键进行确认。
    ⇨ **Auto Declare**（自动声明）对话框会打开。
46. 选择**范围** VAR_OUTPUT 和数据类型 BOOL。
47. 确认对话框。
48. 最后，删除功能块未使用的输入端和输出端上的 **???**。
    ⇨ 现在可以将已完成的功能块 FB_Blinker 实例化并进行调用。
49. 在 FBD/LD/IL 编辑器中，打开“信号”程序。
50. 点击编辑器窗口中的第 1 个网络的下方。
51. 在上下文菜单中，选择命令 **Insert network**（插入网络）。
    ⇨ 出现 1 个编号为 2 的空网络。
52. 与第 1 个网络一样，执行 1 个 TON 功能块，对信号的激活进行时序控制，这次由输入端 IN 处的全局变量“GVL_Var.bDoorOpen”触发。
53. 在输入端 PT 处添加全局变量“GVL_Var.tImDoorOpenThreshold”。
54. 此外，将功能块 FB_Blinker 添加到该网络中 TON 功能块的输出端 Q 上。
    ⇨ 功能块 FB_Blinker 为信号转发 Q 计时，因此为“GVL_Var.bSignal”计时。
55. 为此，从 **Toolbox**（工具箱）视图中，将 1 个 **Contact**（触点）元素拖至功能块的 OUT 输出端。
56. 将变量“fbTimer2.Q”分配给该触点。
57. 在该触点后插入 1 个 **Coil** （线圈）元素，并将全局变量“GVL_Var.bSignal”分配给它。
58. 将值 T#1s 分配给功能块 FB_Blinker 的 2 个输入变量“tSignalTimeOn”和“tSignalTimeOff”。
    ⇨ 然后，TRUE 的循环时间为 1 秒，FALSE 的循环时间为 1 秒。
59. 点击 TON 功能块。
60. 在上下文菜单中，选择命令 **Remove unused FB call parameters**（删除未使用的 FB 调用参数）。
    ⇨ 未使用的输出 ET 已被删除。
    ⇨ 在 LD 的第 3 个网络中，编程让灯在门打开时亮起。
61. 为此，添加另一个网络，并在左侧添加 1 个触点“GVL_Var.bDoorOpen”，该触点直接通向插入的线圈“GVL_Var.bLamp”。
    ⇨ TwinCAT 可连续处理 1 个 LD 的网络。
62. 为确保仅执行网络 1 或网络 2，可在网络 1 的末尾添加跳转到网络 3。
63. 点击进入网络或带有网络编号的字段，选择网络 3。
64. 从上下文菜单中，选择命令 **Insert label** （插入标签）。
65. 用“DoorIsOpen”替换网络左上部分中的标签的 **Label:**（标签：）文本。

66. 选择网络 1。
67. 从 **Toolbox**（工具箱）视图的 **General**（常规）类别中，将 **Jump**（跳转）元素拖至网络中。
68. 将其放置在显示的 **Output**（输出）矩形上，或在 **Insert jump here**（在此处插入跳转）处。
    ⇨ 出现跳转元素。跳转目标仍显示为 **???**。
69. 选择 **???** 并点击 ... 。
70. 从可能的标签标识符中选择“DoorIsOpen”。
71. 点击 **OK**（确定）进行确认。
    ⇨ 网络 3 的标签已执行。

⇨ 现在的 LD 程序如下所示：





在主程序中调用“信号”程序

在我们的示例程序中，MAIN 程序应调用“信号”程序进行信号处理。

1. 在 PLC 项目树中，双击 MAIN 程序。
   ⇨ 在编辑器中打开 MAIN 程序。
2. 从 **Toolbox**（工具箱）视图中，将 1 个 **Box**（方框）元素拖至 MAIN 编辑器中。
3. 使用 **Input Assistant**（输入助手），从 **Module Calls**（模块调用）类别中将“信号”程序调用添加至此功能块中。

为仿真创建 ST 程序块

由于该示例项目没有连接真实的传感器和执行器，因此，您要编写 1 个模拟温度升降的程序。这样，您就可以在在线模式下监控冰箱控制单元的运行状况。

使用“结构化文本（ST）”创建仿真程序。

该程序会使温度升高，直至 MAIN 程序检测到已超过设定温度并激活冷却压缩机。然后，仿真程序再次降低温度，直至主程序停用压缩机。

1. 在 PLC 项目树中，添加 1 个名为“仿真”的**程序**类型的 POU 功能块，并以“结构化文本（ST）”实现语言进行编写。
2. 在 ST 编辑器中执行以下操作：

```
PROGRAM Simulation
VAR
        fbT1                                    : TON;                          //
The temperature is decreased on a time delay, when the compressor has been activated
        tCooling                                : TIME := T#500MS;
        bReduceTemp                             : BOOL;                         //Signal for dereasing the temperature
        fbT2                                    : TON;                          //
The temperature is increased on a time delay, when the compressor has been activated
        tEnvironment                            : TIME := T#2S;         //Delay time when the door is closed
        tEnvironmentDoorOpen : TIME := T#1s;        //Delay time when the door is open
        bRaiseTemp                              : BOOL;                         //Signal for increasing the temperature
        tImTemp                                 : TIME;                         //Delay time
        nCounter                                : INT;
END_VAR

// After the compressor has been activated due to fTempActual being too high, the temperature decreas
es.
// The temperature is decremented by 0.1° C per cycle after a delay of P_Cooling
IF GVL_VAR.bCompressor THEN
        fbT1(IN:= GVL_Var.bCompressor, PT:= tCooling, Q=>bReduceTemp);
        IF bReduceTemp THEN
                GVL_Var.fTempActual := GVL_Var.fTempActual-0.1;
                fbT1(IN:=FALSE);
        END_IF
END_IF

//If the door is open, the warming will occur faster; SEL selects tEnvironmentDoorOpen
tImTemp:=SEL(GVL_Var.bDoorOpen, tEnvironment, tEnvironmentDoorOpen);

//If the compressor is not in operation, then the cooling chamber will become warmer.
//The    temperature is incremented by 0.1° C per cycle after a delay of tImTemp
fbT2(IN:= TRUE, PT:= tImTemp, Q=>bRaiseTemp);
IF bRaiseTemp THEN
        GVL_Var.fTempActual := GVL_Var.fTempActual + 0.1;
        fbT2(IN:=FALSE);
END_IF

nCounter := nCounter+1; // No function, just for demonstration purposes.
```

可视化

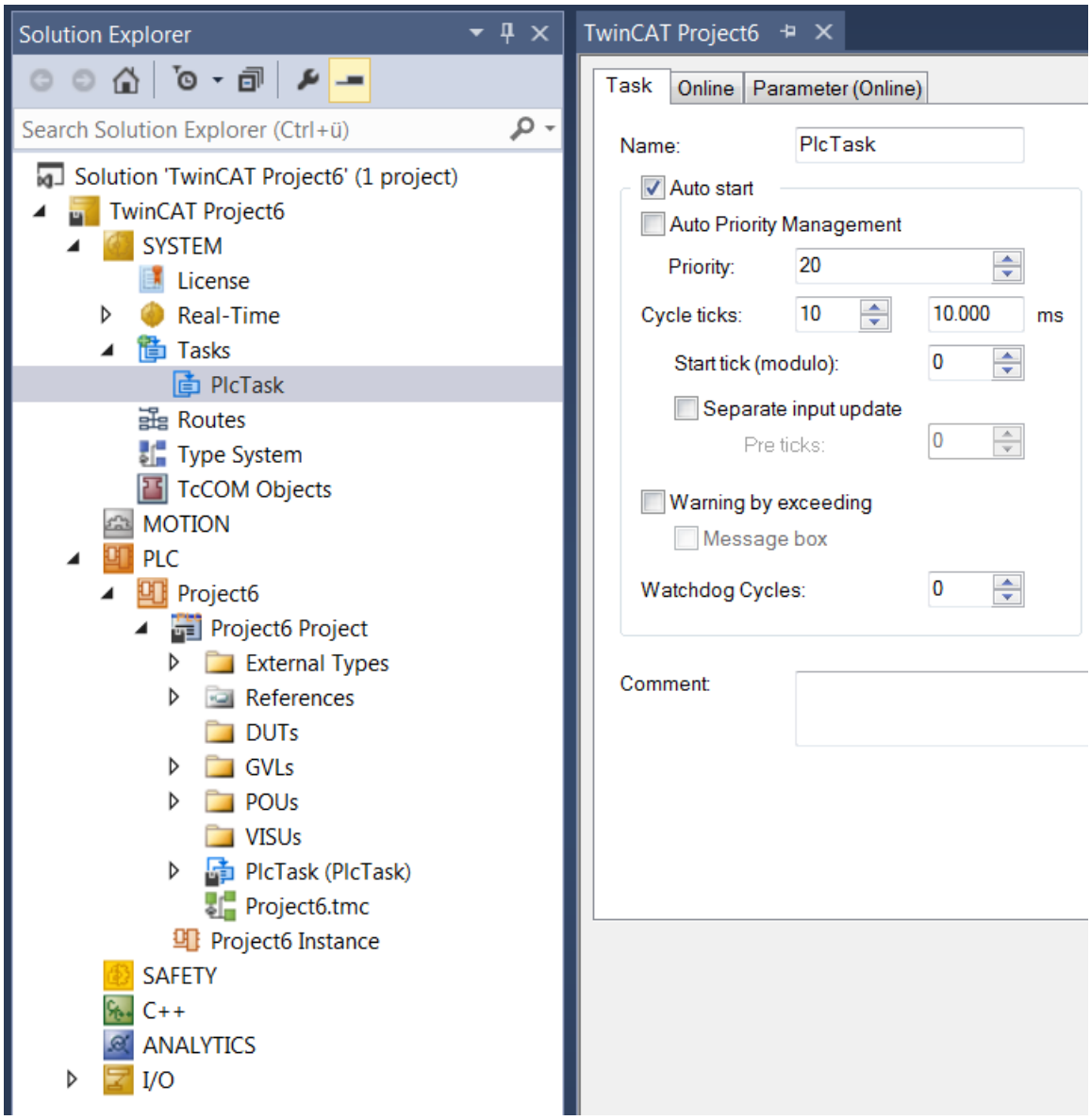
为了便于操作和监控整个控制程序，您可以使用可视化系统来显示冰箱以及仿真程序的运行情况。在 TwinCAT 中，您还可以在 PLC 区域对可视化进行编程。在控制器上启动项目时，您无需输入任何内容即可开始可视化。根据编程的不同，您可以通过点击打开/关闭开关等方式触发关门和开门，也可以通过控制旋钮的指针调节温度预选。这里不介绍创建可视化系统的过程。

在任务配置中定义要执行的程序

预设任务配置包含 MAIN 程序的调用。对于我们的示例项目，您必须添加“仿真”程序的调用。

1. 在 PLC 项目树中，将“仿真”条目拖至任务引用（PlcTask）中。
   ⇨ “仿真”程序已添加至任务配置中。
2. 如要查看任务配置，可双击 PLC 项目树中的 **PlcTask** 条目。
   ⇨ 在 **Solution Explorer** （解决方案资源管理器）中，在 **SYSTEM** > **Tasks**（系统 > 任务）下已启用引用任务（**PlcTask**）。
3. 双击任务可在编辑器中打开任务配置。

⇨ 在 **PlcTask** 的 PLC 项目树中，您可以看到该任务调用的 POU：MAIN（默认输入）和仿真。SYSTEM 区域中的 **PlcTask** 节点的编辑器会显示所引用 PlcTask 的相应循环时间。在本示例中，时间间隔为 10 毫秒。在在线模式下，该任务将在每个周期对这 2 个功能块进行 1 次处理。



定义活动的 PLC 项目

1 个 TwinCAT 控制器可以运行多个 PLC 项目。**TwinCAT PLC Toolbar Options**（TwinCAT PLC 工具栏选项）中的 **Active PLC Project**（活动的 PLC 项目）下拉列表的第 1 个条目会显示当前活动的 PLC 项目。如果存在多个 PLC 项目，您可通过下拉列表选择 1 个 PLC 项目。

检查 PLC 项目是否有错误

在输入代码时，如果出现语法错误，TwinCAT 会立即用红色波浪线提醒您。

1. 如要对整个 PLC 项目进行语法检查，可选择 PLC 项目对象“<PLC project name> Project”（<PLC project name> 项目）。
2. 从上下文菜单或 **Build**（生成）菜单中，选择命令 **Check all objects**（检查所有对象）。
   ⇨ 检查结果显示在 **Error List**（错误列表）视图中。
3. 如有必要，可使用 **View**（视图）菜单中的命令 **Error List**（错误列表），打开 **Error List**（错误列表）视图。
4. 然后，您可以双击该消息，以跳转到相应的代码位置。

您可以使用命令 **Check all objects**（检查所有对象）来检查和编译 PLC 节点中的所有功能块。如果您在编译的过程中发现树中的功能块无法编译，例如，这些功能块可能只是为了测试而被包含在内，则会生成 1 个错误。因此，建议运行命令 **Check all objects**（检查所有对象），尤其是在检查库功能块时。

您可以使用命令 **Build**（生成解决方案）或 **Rebuild**（重新生成解决方案）将检查和编译限制在 PLC 项目中实际使用的功能块上。

在将 PLC 项目加载到控制器上时，可对其执行进一步检查。

只有准确无误的 PLC 项目才能加载到控制器上。

编译 PLC 模块

您可以使用命令 **Build**（生成解决方案）或 **Rebuild**（重新生成解决方案）来编译在 PLC 项目中使用的代码，并检查语法准确性。

选择目标系统

现在，从 **TwinCAT XAE Base Toolbar Options**（TwinCAT XAE 基础工具栏选项）中的 **Choose Target System**（选择目标系统）下拉列表中，为您的控制程序选择目标设备：

- 选择 **<Local>** 可将控制代码直接加载到您的编程设备的本地运行时中。（为当前示例选择此选项）。
- 如果您想要选择其他目标设备，请从下拉列表中选择 **Choose Target System**（选择目标系统）。然后，选择预配置的目标设备，或者浏览网络查找目标设备，对其进行配置，然后再选择该目标设备。

激活配置

1. 点击 **TwinCAT XAE Base Toolbar Options**（TwinCAT XAE 基础工具栏选项）中的  。
   ⇨ 此时会出现 1 个对话框，询问您是否想要激活配置。
2. 点击 **OK**（确定）。
   ⇨ 此时会出现 1 个对话框，询问是否应该在运行模式下重新启动 TwinCAT。
3. 点击 **OK**（确定）。

⇨ 配置已被激活，TwinCAT 已被设置为运行模式。在任务栏中显示当前状态：  。激活还可将 PLC 项目传输至控制器。

将 PLC 项目加载到 PLC 上

✓ PLC 项目编译准确无误。请参见 检查 PLC 项目是否有错误 [▶ 32]，

1. 在 **PLC** 菜单中选择 **Login**（登录）命令，或者在 **TwinCAT PLC Toolbar Options**（TwinCAT PLC 工具栏选项）中点击  。
   ⇨ 此时会出现 1 个对话框，询问是否应该创建并加载应用程序。
2. 点击 **Yes**（是）。

⇨ PLC 项目已加载到控制器上。现在，开发环境已进入在线模式。PLC 模块尚未进入运行模式。在 **Solution Explorer**（解决方案资源管理器）中，在 PLC 项目对象前会出现以下符号：  。在加载过程中，**Error List**（错误列表）视图会显示关于生成代码的大小、全局数据的大小以及由此产生的控制器内存需求的信息。

启动程序

如果您已经按照整个教程学习到这里，您现在就可以在 PLC 设备上使用 PLC 项目了。

1. 在 **PLC** 菜单中选择 **Start**（开始）命令，或者在 **TwinCAT PLC Toolbar Options**（TwinCAT PLC 工具栏选项）中点击 （**[F5]**）。

⇨ 程序正在运行。PLC 模块进入运行模式。在 **Solution Explorer**（解决方案资源管理器）中，在 PLC 项目对象前会出现以下符号： 。

在运行时进行监控并写入 1 次变量值

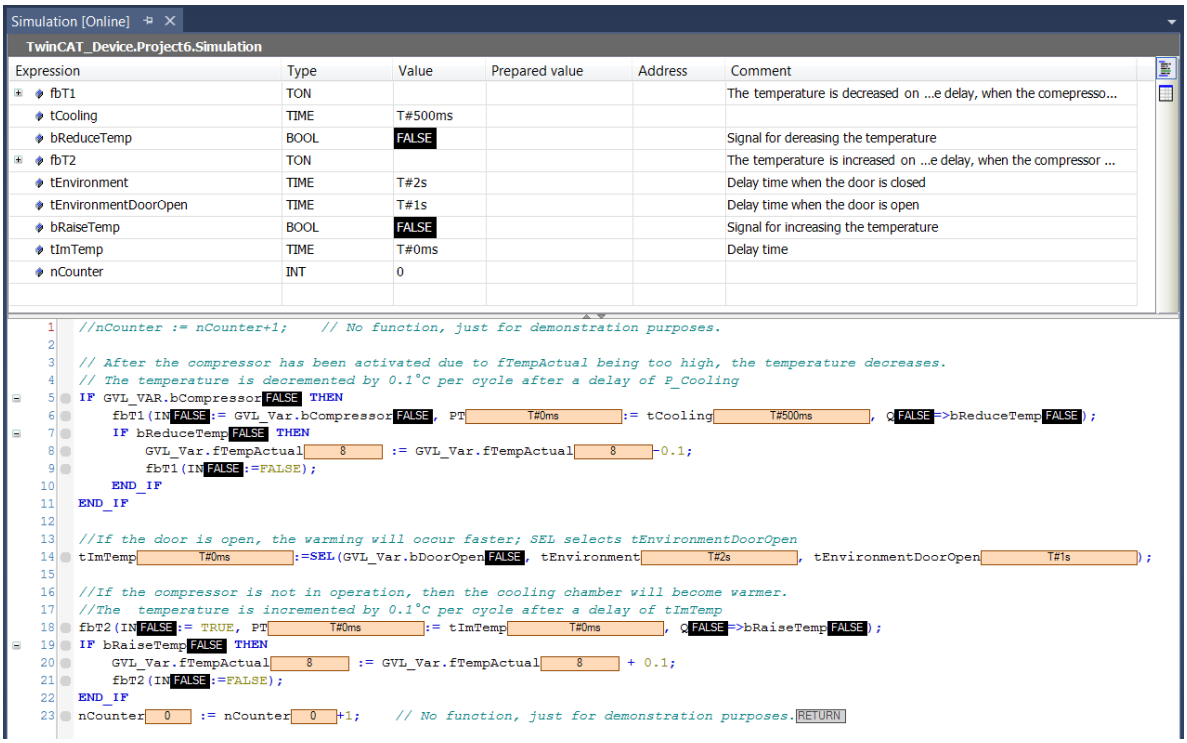
以下部分将说明如何监控各个程序块中的变量值，请您通过 TwinCAT 在控制器上设置 1 次变量值。

在功能块编辑器的在线视图或监视列表中可以看到程序变量的实际值。本示例有意限制在功能块编辑器中进行监控。

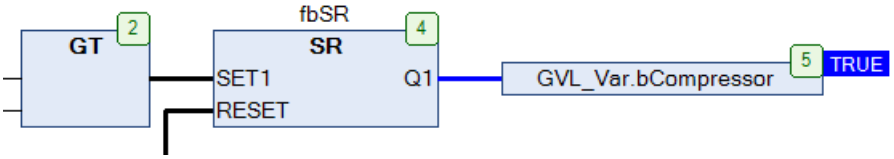
✓ 在控制器上运行 PLC 程序。

1. 在 PLC 项目树中，双击对象 MAIN、“Signals”（信号）、“Simulation”（仿真）和“GVL_Var”，以打开编辑器的在线视图。

⇨ 在每个视图的声明部分，**Value**（值）列中的表达式表格显示了控制器上的变量的实际值。



⇨ 实现部分的监控取决于实现语言：对于非 Boolean 变量，其值始终位于标识符右侧的矩形区域内。在 ST 编辑器中，这也适用于 Boolean 变量。该显示名为“Inline Monitoring”（内联监控）。在图形编辑器中，Boolean 变量的值用输出链接线的颜色表示：黑色表示 FALSE，蓝色表示 TRUE：



⇨ 注意不同功能块中的变量值是如何变化的。例如，全局变量列表“GVL_Var”显示了“fTempActual”和“bCompressor”的值在执行仿真程序时如何发生变化。

在控制器上一次性设置变量值：

1. 将焦点设置为全局变量列表“GVL_Var”的在线视图。
2. 如要指定新的设定点，可双击表达式“fTempSet”下的 **Prepared value**（准备值）列。

⇨ 输入字段打开。

3. 输入值“9”并退出输入字段。
4. 如要将门设置为打开，可在“bDoorOpen”表达式的 **Prepared value**（准备值）字段中点击 1 次。

   ⇨ 值 TRUE 已输入。
5. 再点击 3 次，您就会发现可以用它将准备值设置为 FALSE，然后再次设置为空，最后再次设置为 TRUE。
6. 如要将准备值 TRUE 写入变量 1 次，可在 **PLC** 菜单中选择命令 **Write values**（写入值），或者在 **TwinCAT PLC Toolbar Options**（TwinCAT PLC 工具栏选项）中点击  。

⇨ 这 2 个值将被传输至 **Value**（值）列。变量“bDoorOpen”的值不再变化，设定温度现在为 9 度。变量“tImTemp”的值变为 1 秒，因为冰箱门现在是“打开”的，因此，仿真加热的速度应该比之前（2 秒）更快。

在运行时设置断点和逐步执行

调试：为了排除故障，您想要检查某些代码位置的变量值。为此，您可以为处理过程定义断点并启动指令的逐步执行。

✓ PLC 程序已加载到控制器上并处于运行状态。

1. 双击“Simulation”（仿真）对象，以在编辑器中打开程序。
2. 将光标置于代码行 nCounter:=nCoutner+1; 并按 **[F9]**。

   ⇨ 在代码行前出现符号  。这表示在该行已设置断点。该符号立即变为  。黄色箭头始终指向要执行的下一条指令。
3. 在内联监控或 **Simulation**（仿真）程序的声明部分，考虑变量“nCounter”的值。

   ⇨ 变量值不再变化。在断点处停止执行。
4. 按 **[F5]** 重新启动执行。

   ⇨ 经过 1 个周期后，程序在断点处再次停止。“nCounter”递增 1。
5. 按 **[F11]** 执行下一步

   ⇨ RETURN 在行尾 nCounter:=nCounter+1; 处，指令以黄色突出显示
6. 再次按 **[F11]** 执行下一步。

   ⇨ 执行跳转到 MAIN 的编辑器。反复按 **[F11]** 可以显示程序如何逐步执行。再次以黄色箭头标记要执行的指令。
7. 如要禁用断点并返回正常执行，可再次将光标移至代码行并按 **[F9]**。然后，按 **[F5]** 重新启动程序执行。

⇨ 您可以逐步运行程序，并检查某些代码位置的变量值。

在运行时执行单周期

✓ PLC 程序已加载到控制器上并处于运行状态。

1. 再次观察 **Simulation**（仿真）程序中的行 nCounter:=nCounter+1;。
2. 点击 **TwinCAT PLC Toolbar Options**（TwinCAT PLC 工具栏选项）中的  ，执行单周期。

   ⇨ 执行过程经过 1 个周期，在断点处再次停止。“nCounter”递增 1。
3. 再点击 3 次按钮，即可看到各个周期。然后，再次按 **[F5]**。

⇨ 程序再次运行而不会停止，也没有强制值。变量“tImTemp”的值再次变为 1 秒。

3 技巧和窍门

在以下几页中，您将会看到一些有用的信息，这些信息可能对您规划 TwinCAT 3 项目有所帮助。这些技巧和窍门包括以下几点：

- 在 项目交付 [▶ 36] 之前建议的步骤
- TwinCAT 3 版本的 新属性和功能 [▶ 39]

3.1 项目交付

一旦 PLC 项目开发完成，在项目交付之前，我们建议检查以下步骤，如果这些步骤与项目相关，则应执行这些步骤。

命令/步骤	在哪里可以找到	目的	更多信息
激活“Pin Version”（引脚版本）选项	TwinCAT 3 项目 > 双击 SYSTEM 节点 > ”General“（常规）选项卡	在打开项目时，如果工程计算机上存在项目中定义的 TwinCAT 3 开发环境版本，则会自动使用该版本。 为了避免在稍后激活/登录时因使用不同的开发环境版本而导致源代码发生变化	TE1000 XAE\技术\远程管理器\\引脚版本项目设置
将 PLC 应用程序项目的编译器版本设为“newest”（最新）	PLC 项目 > 右键点击 PLC 项目对象 > 选择“Properties”（属性）命令 > 打开“Compile”（编译）类别 > 将“Compiler version”（编译器版本）设置为“newest”（最新）	然后，应用程序项目使用的编译器版本将自动与远程管理器中加载的开发环境版本相匹配（请参见第 1 点：引脚版本）。这是编译器版本的预期使用方法。	类别编译 [▶ 844]
将 PLC 库引用设置为固定版本（不使用“always newest”（始终最新）/“*”）	PLC 项目 只需 1 条命令，即可同时引用所有库/占位符： > 右键点击引用节点 > 选择命令“Use effective version”（使用有效版本） 或者对于单个或多个库/占位符（通过多项选择）： > 右键点击引用节点下的库引用 > 选择命令“Use effective version”（使用有效版本） 或者对于单个库/占位符： > 标记引用节点下的库引用 > 在 Properties（属性）窗口中选择 1 个固定版本 或者对于单个占位符： > 右键点击引用节点 > 打开占位符对话框 > 在“Library”（库）列中选择 1 个固定版本	为了避免在稍后登录时因自动使用较新版本的库而触发在线更改查询（如果已触发“always newest”（始终最新）/“*”的库引用，且该库的库存储库中有较新版本的库）	建议和说明 [▶ 248]

命令/步骤	在哪里可以找到	目的	更多信息
检查目标存档的设置 注意： 如果您应该更改设置，则必须重新激活配置或启动项目，才能接受这些设置。	PLC 项目 > 双击 PLC 项目 > “Settings”（设置）选项卡	为了避免因在目标系统上保存而在无意中传递项目源或库 • 如果项目源在目标上，则具有潜在优点： ◦ 在 XAE 中打开的 PLC 项目可与目标上的版本进行详细比较（“Compare Project with Target”（比较项目与目标）命令）。 ◦ 可从目标加载并打开 PLC 项目（“Open Project from Target”（从目标打开项目）命令）。 • 如果项目源在目标上，则具有潜在缺点：如果具有目标系统的访问权限，就可以读取/复制源。	设置选项卡 [▶ 859]

对于标准设备，使用/激活以下选项通常会有所帮助。
此外，如果要在文件级别进行完整的设备更新，这 2 个选项必不可少。

选项	在哪里可以找到	目的	更多信息
路由设置中的“Use relative NetIds”（使用相对 NetIds）选项	TwinCAT 3 项目 > 展开 SYSTEM 节点 > 双击路由节点 > “NetId Management”（NetId 管理）选项卡	如果已激活选项“Use relative NetIds”（使用相对 NetIds），则前 4 位数字与占位符的项目 NetId 相对应，并根据目标系统的实际 NetId 进行设置。这样就可以在另一个相同的目标系统上使用项目或配置，而无需手动调整项目 NetId。 示例： • 具有 AMS NetId 172.17.35.70.1.1 的目标系统 • 第 1 台设备（例如 EtherCAT 主站）的 NetId ◦ 如果未启用选项“Use relative NetIds”（使用相对 NetIds）：172.17.35.70.2.1 ◦ 如果已启用选项“Use relative NetIds”（使用相对 NetIds）：[172.17.35.70].2.1	TE1000 XAE\系统\系统节点\系统子节点\路由 和 TE1000 XAE\系统\文件级设备更新\执行完整的设备更新
所有网络和 USB 设备的适配器设置中的选项“Virtual device names”（虚拟设备名称）	TwinCAT 3 项目 > 展开 I/O 节点 > 展开设备节点 > 双击设备，例如 EtherCAT 主站 > “Adapter”（适配器）选项卡	如果已激活选项“Virtual device names”（虚拟设备名称），则使用口语名称或显示名称作为设备的引用。然后，系统会使用设备名称，而不是 MAC 地址。	TE1000 XAE\I/O\EtherCAT\EtherCAT 主站\适配器 和 TE1000 XAE\系统\文件级设备更新\执行完整的设备更新

3.2 新属性和功能

3.2.1 TwinCAT 3.1 Build 4024

TwinCAT 3.1 Build 4024 提供以下新属性和功能。

类别	功能	更多信息	此构建版本及以上
开发支持	“Go to definition”（转至定义）命令在 PLC 过程映像中可用	命令转至定义 [▶ 816]	4024.0
	用于取消注释或撤销取消注释的菜单命令和热键	命令注释选择 [▶ 817] 命令取消注释选择 [▶ 818]	4024.0
	确定未链接的已分配变量（AT%I、AT%Q）	见下文	4024.10
存储格式	可选存储格式 Base64	命令属性（对象） [▶ 836]	4024.0
编译器	在声明编辑器中也提供了条件编译（除实现编辑器之外）	条件编译指示 [▶ 775]	4024.0
面向对象的编程	对接口变量进行扩展监控： 在监控区域（声明编辑器、监控列表）的接口变量下会显示当前分配的 FB 实例的符号路径和在线数据。	对象接口 [▶ 94]	4024.0
	迷你图标显示访问修饰符： 项目树中的对象符号上的迷你图标会显示对象是私有的、公开的，还是受保护的。	对象方法 [▶ 82]	4024.0
	ABSTRACT 关键字	ABSTRACT 概念 [▶ 185]	4024.0
在线功能	枚举的 {attribute 'to_string'}： 该属性用于通过 TO_STRING/TO_WSTRING 以文本方式查询枚举元素的名称。	属性“to string” [▶ 774]	4024.0
	通过 __TRY/__CATCH 处理异常（适用于 32 位运行时系统）	__TRY、__CATCH、__FINALLY、__ENDTRY [▶ 679]	4024.0
	在线更改的内存保留	为在线更改配置内存保留 [▶ 122]	4024.0
	在登录已更改的 PLC 项目时，消息窗口中的详细信息按钮	更新 PLC 上的 PLC 项目 [▶ 231]	4024.0
	命令“Go to instance”（转至实例）	命令转至实例 [▶ 816]	4024.0
	利用核心转储进行错误分析	利用核心转储进行错误分析 [▶ 227]	4024.11
其它	选项“Generate tpy file”（生成 tpy 文件）	类别编译 [▶ 844]	4024.0

确定未链接的已分配变量（AT%I、AT%Q）

借助下文所述的摘要，您可以获得未链接的已分配变量（AT%I、AT%Q）的概览。因此，您可以获得链接或项目状态的概览等信息。

程序：

1. 双击解决方案资源管理器中的 PLC 任务的输入或输出过程映像（例如，双击对象“PlcTask Inputs”（PlcTask 输入）或“PlcTask Outputs”（PlcTask 输出））。
   ⇨ 1 个摘要会打开，显示此 PLC 过程映像的分配输入或输出（见下文，第 1 张截图）。
2. 点击第 2 个表格列的标题（列标题：“[X]”）。
   ⇨ 只有未链接的已分配输入或输出仍会显示（列标题变为：“[ ]”（见下文，第 2 张截图）。
3. 再次点击第 2 个表格列的标题。
   ⇨ 所有已分配的输入或输出会再次显示（列标题变为“[X]”）。

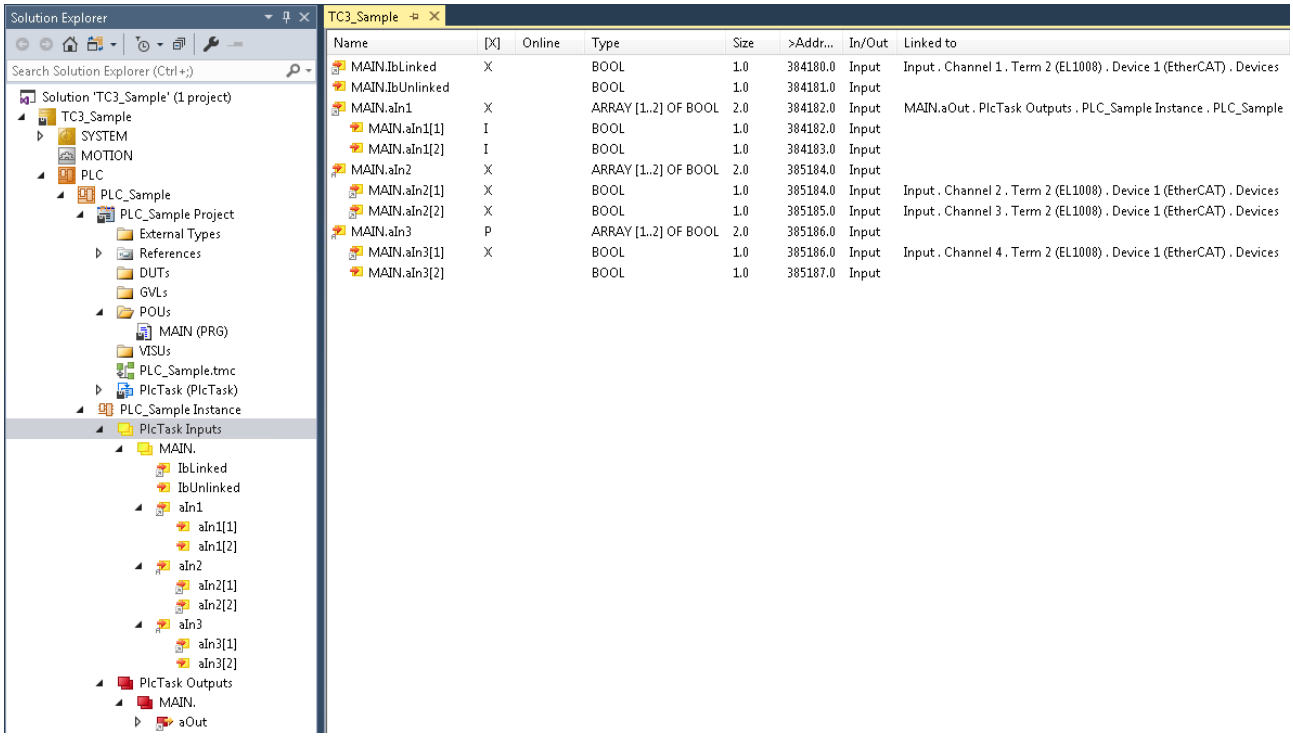
注意：

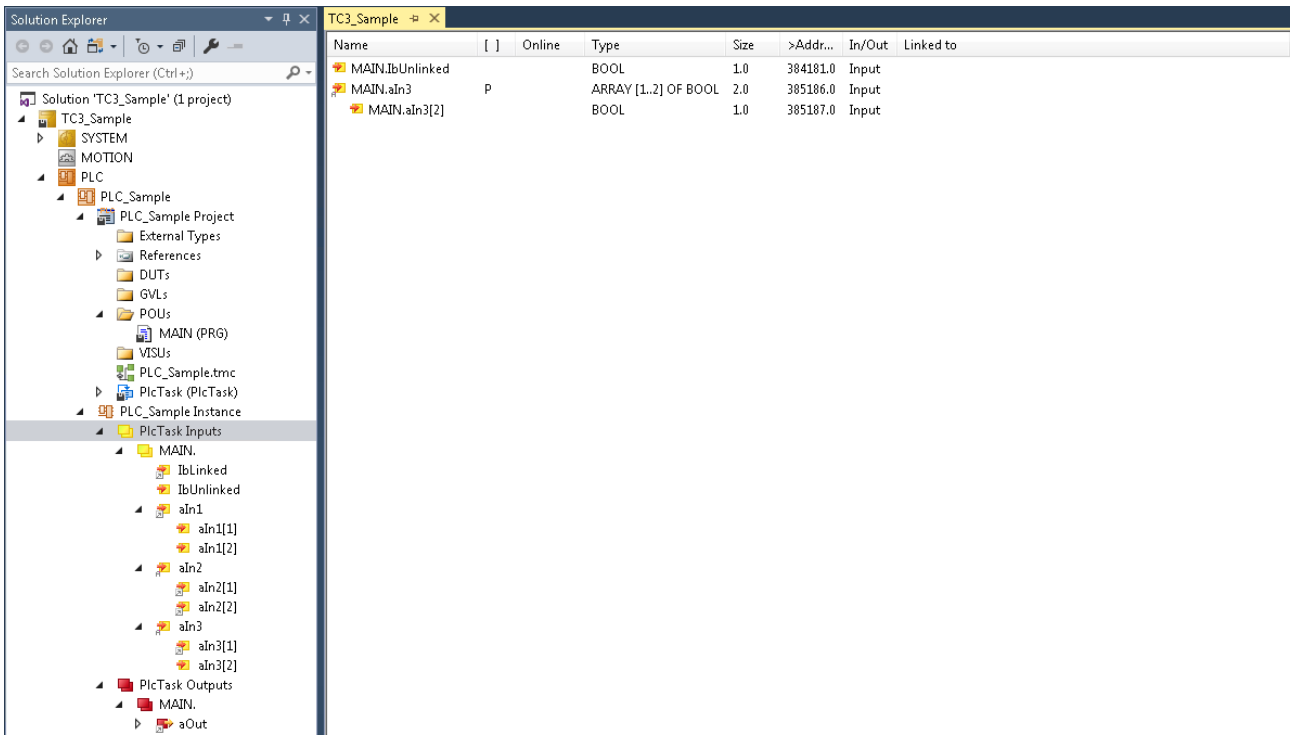
- 借助该摘要，您可以创建新链接或更改现有链接。为此，可右键点击变量/行并选择命令 **Change link**（更改链接）。
- 如要显示结构化变量或数组的子元素，可点击 **TwinCAT XAE Base** 工具栏内的 **Show sub-elements**（显示子元素）按钮。

“[X]”或“[ ]”列的值：

列值	含义	示例（见 *）	截图中的变量示例（见下文）
空	未链接		IbUnlinked
X	已链接	单独链接的变量	IbLinked MAIN.aIn3[1]
		数组本身已链接的数组	MAIN.aIn1
		数组中的所有元素均已单独链接的数组	MAIN.aIn2
P	部分链接	数组中至少有 1 个元素（并非所有元素）已单独链接的数组	MAIN.aIn3
I	间接链接（通过父类变量）	数组本身已链接的数组的元素，因此数组元素也已间接链接	MAIN.aIn1[1]

* 已命名数组示例也可被传输至结构变量（例如，结构实例）及其子元素中。





3.2.2 TwinCAT 3.1 Build 4026

TwinCAT 3.1 Build 4026 提供以下新属性和功能。

类别	功能	更多信息	此构建版本及以上
开发支持	交叉引用列表：新的“Context”（上下文）列	命令：交叉引用表 [▶ 875]	4026.0
	列出组件：扩展功能，可对输入的字符进行分类和突出显示。	使用输入向导 [▶ 123]	4026.0
	智能标签功能：从实现部分简单声明变量。	使用输入向导 [▶ 123]	4026.0
	PLC 书签	设置和使用书签 [▶ 146]	4026.0
ST 编辑器	深色主题（离线）	对话框选项 - 文本编辑器 [▶ 914]	4026.0
	突出显示光标所在变量的所有使用位置。	ST 编辑器 [▶ 571]	4026.0
	增量搜索	ST 编辑器 [▶ 571]	4026.0
	矩形选择	ST 编辑器 [▶ 571]	4026.0
CFC 编辑器	默认的执行顺序设置：自动数据流模式	命令属性（对象） [▶ 840]	4026.0
	临时显示执行顺序	命令显示执行顺序 [▶ 941]	4026.0
	更容易从工具箱中插入多个元素	CFC 编辑器 [▶ 607]	4026.0
	从声明部分和解决方案资源管理器拖放变量和对象	CFC 编辑器 [▶ 607]	4026.0
	通过拖放改变引脚排列	CFC 编辑器 [▶ 607]	4026.0
面向对象的编程	对于带有初始值的输入参数，在调用时无需传递变量。	方法调用 [▶ 184]	4026.0
	附加 FB_init 参数的 IntelliSense	FB_init [▶ 787]	4026.0
在线功能	继承变量的结构化表示法	命令继承的显示 - 简单、结构化 [▶ 895]	4026.0
	“Go to implementation”（转至实现）和“Go to reference”（转至引用）命令	命令转至实现 [▶ 816] 命令转至引用 [▶ 816]	4026.0
	“Select online status”（选择在线状态）对话框中的筛选选项	编程对象中的监控 [▶ 205]	4026.0
库	使用 PLC 项目作为引用库。	使用 PLC 项目作为引用库	4026.0
	PLC 项目属性中的新增库选项：“Allow implicit checks for compiled libraries”（允许对已编译的库进行隐式检查）和“Force Qualified_only for library access”（强制对库访问进行 Qualified_only）。	类别：通用 [▶ 841]	4026.0
	通过链接改进库管理器的导航功能。	库管理器 [▶ 254]	4026.0
	能够在库管理器中激活和停用存储库。	库存储库 [▶ 251]	4026.0
	区分添加占位符和库的命令。	添加无占位符解析的库命令 [▶ 330]	4026.0
	“Add Library”（添加库）对话框中的多项选择	命令添加库 [▶ 329]	4026.0
	将参数列表的值导出为 .csv 文件	对象参数列表 [▶ 67]	4026.0
运算符和变量	使用 __POUNAME 运算符确定本地对象的名称，使用 __POSITION 运算符确定行号。	__POUNAME [▶ 685] __POSITION [▶ 685]	4026.0
	通用常量	通用常量变量 - VAR_GENERIC CONSTANT [▶ 629]	4026.0

类别	功能	更多信息	此构建版本及以上
其它	类型系统的表示法： 库 Tc3_Global_Types ExternalTypes.tmc，而不是外部类型文件夹	使用全局数据类型 [▶ 54]	4026.0
	将 PLC 项目另存为可重复使用的项目模板。		4026.0

3.3 更多有用的属性和功能

除了特定 TC3.1 版本提供的属性和功能外，工程环境还包含以下有用的属性和功能。

类别	功能	更多信息
开发支持	热键： 许多功能都有热键，这些热键可根据需要进行单独调整。	热键 [▶ 46]
	选项“Activate folder view”（激活文件夹视图）： 在 PLC 过程映像和链接对话框中可选择文件夹视图	见下文
	重构： 方便后续更改程序	重构 [▶ 148]
	面向对象的编程选项： 可用于创建易于读取和扩展的软件	面向对象的编程 [▶ 158]
定向和导航	查找使用交叉引用列表的位置	使用交叉引用列表查找出现 [▶ 145]
	查找声明	查找声明 [▶ 146]
在线功能	隐式监控函数： 例如，在运行时检查数组的限值	使用功能块进行隐式检查 [▶ 151]
	流程控制： 跟踪程序执行情况	流程控制 [▶ 202]
	监控列表： 明确监控变量值	使用监视列表 [▶ 209]
	断点和逐步执行程序： 查找程序中的错误	使用断点 [▶ 195] 逐步处理程序（步进） [▶ 197]
库	使用库： 创建和使用可重复使用的对象集合	使用库 [▶ 246]
	库占位符： 便于为 PLC 项目中直接或间接引用的占位符选择特定的库版本	库占位符 [▶ 257]
	命令“Use effective version”（使用有效版本）： 定义对特定库版本的库引用	命令设置为有效版本 [▶ 334] 项目交付 [▶ 36]

PLC 过程映像和链接对话框中的可选文件夹视图

选项：复选框 **Activate folder view**（激活文件夹视图）

- 已激活：PLC 过程映像和链接对话框的 PLC 区域中的变量会显示为可扩展的文件夹级别。
- 已停用：PLC 过程映像和链接对话框的 PLC 区域中的变量会显示为平面列表。

对话框：双击 **Solution Explorer**（解决方案资源管理器） > **PLC settings**（PLC 设置）选项卡中的 PLC 节点

示例：

功能块 FB_Sample

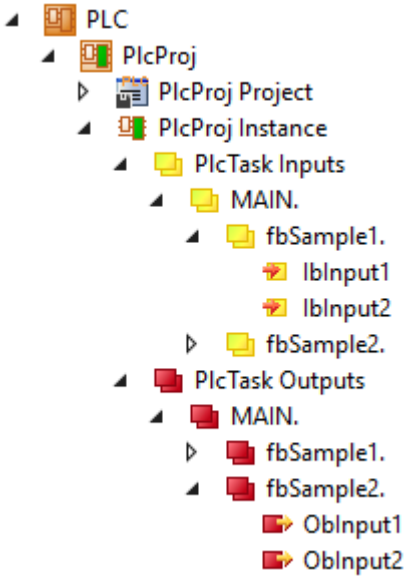
```
FUNCTION_BLOCK FB_Sample
VAR
    IbInput1  AT%I* : BOOL;
    IbInput2  AT%I* : BOOL;
    ObInput1  AT%Q* : BOOL;
    ObInput2  AT%Q* : BOOL;
END_VAR
```

MAIN 程序

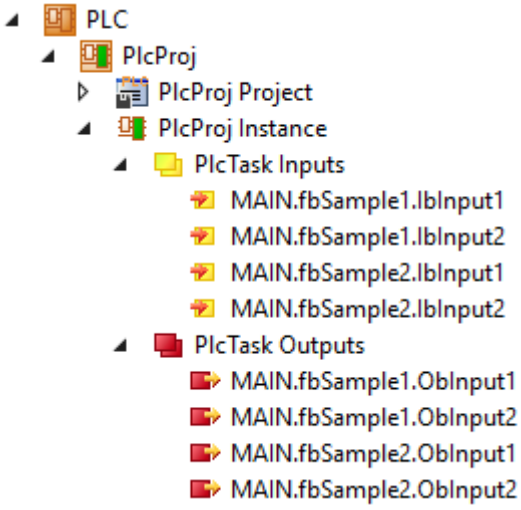
```
PROGRAM MAIN
VAR
    fbSample1       : FB_Sample;
    fbSample2       : FB_Sample;
END_VAR
```

PLC 过程映像：

- 情况 1：**Activate folder view**（激活文件夹视图）复选框已激活。

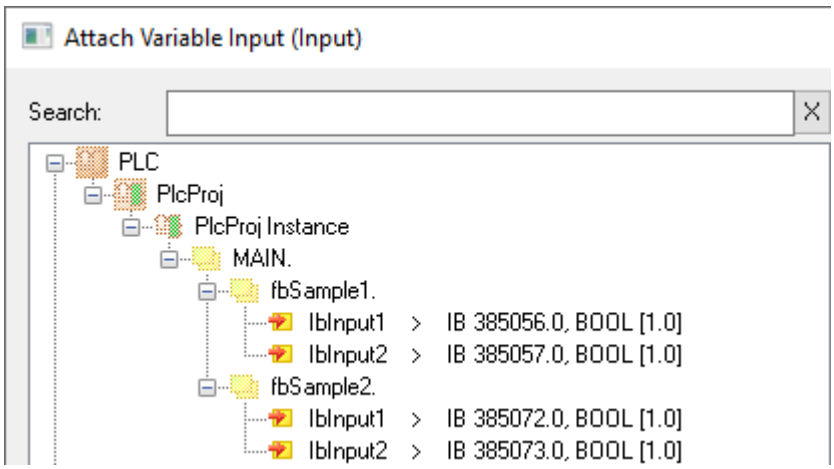


- 情况 2：**Activate folder view**（激活文件夹视图）复选框已停用。

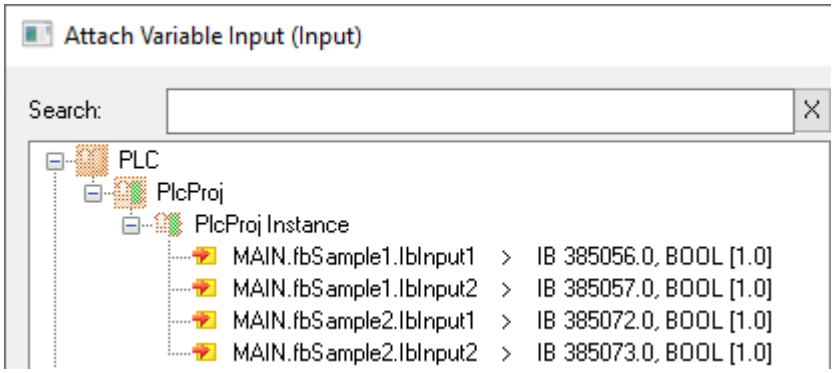


链接对话框：

- 情况 1：**Activate folder view**（激活文件夹视图）复选框已激活



- 情况 2: **Activate folder view**（激活文件夹视图）复选框已停用



3.3.1 热键

在 TwinCAT 3 开发环境中，通过热键可以执行一些命令。您可以在下文中看到这些命令及其默认热键的摘要。如有必要，您可以调整和/或扩展热键的配置。

类别	命令	默认 热键	更多 信息
标准命令	撤销	Ctrl+Z	标准命令 [▶ 809]
	重做	Ctrl+Y	
	复制	Ctrl+C	
	剪切	Ctrl+X	
	粘贴	Ctrl+V	
	全选	Ctrl+A	命令全选 [▶ 810]
	删除	Delete	命令删除 [▶ 809]
开发支持	转至定义	F12	命令转至定义 [▶ 816]
	搜索引用	Shift+F12	命令查找所有引用 [▶ 817]
	输入助手	F2	命令：输入助手 [▶ 810]
	在声明和实现编辑器之间切换焦点	F6	
注释	取消注释	[Ctrl+K] + [Ctrl+C]	命令注释选择 [▶ 817]
	取消取消注释	[Ctrl+K] + [Ctrl+U]	命令取消注释选择 [▶ 818]
在线命令	开始	F5	命令：开始 [▶ 890]
	停止	Shift+F5	命令：停止 [▶ 890]

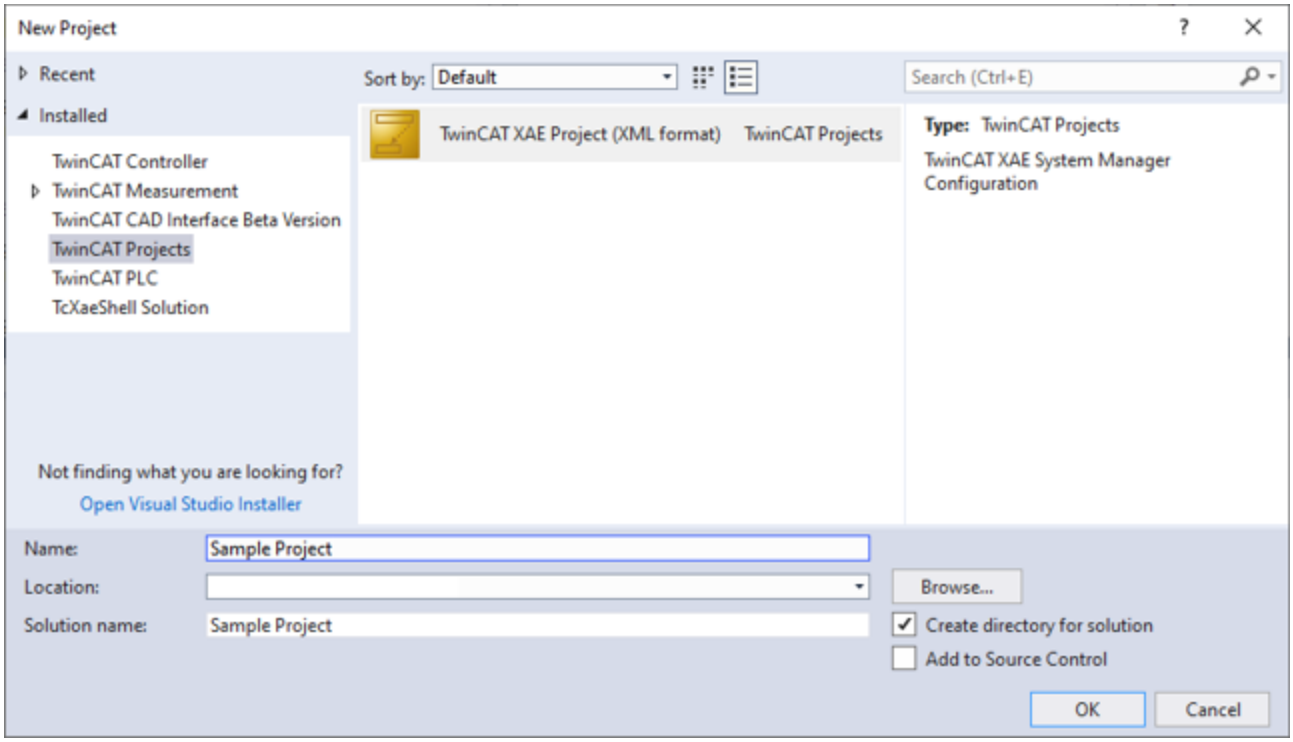
3.4 重命名项目

情景

1 个现有项目要重新用于新设备。为此，要以新设备的名称为该项目命名。

挑战

在创建项目时，系统会询问您是否应该创建解决方案目录。该选项为默认行为。如果选择该选项，则将在该目录下创建 TwinCAT 3 项目。



项目的解决方案文件（*.sln）包含对所有子项目的引用。如果重命名 TwinCAT 3 项目，则只重命名项目文件，而不重命名文件夹。因此，在重命名项目后，TwinCAT 3 XAE 项目的旧文件夹名称仍将存在于硬盘上，尽管它不在解决方案中。

解决方案选项

- 选择不引用项目的通用文件夹名称。
- 禁用选项 **Create directory for solution**（为解决方案创建目录），然后，解决方案文件会保存在 TwinCAT 3 XAE 项目的项目文件旁边。
- 如果解决方案文件只包含 1 个 TwinCAT 3 项目，则可以将 TwinCAT 3 项目手动复制到另一个文件夹中，该文件夹不带 *.sln 文件和保存 TwinCAT 3 项目的文件夹。双击 *.tsproj 文件还可以打开 TwinCAT 3 XAE Shell。您可以在 XAE Shell 中相应地重命名项目。在保存项目时，系统会自动询问您是否应该创建新的解决方案文件。
- 复制整个目录并手动调整子项目的文件夹名称。随后，您还须在 *.sln 文件中手动替换这些文件夹名称。

4 创建和配置 PLC 项目

什么是 PLC 项目？

- 项目包含创建控制器程序所需的对象：
  - 纯编程块，例如程序、功能块、函数、GVL。
  - 在 PLC 上运行程序所需的附加对象。例如，引用任务、库管理器和可视化。
- 在 1 个 TwinCAT 项目中，您可以对多个 PLC 项目进行编程，并在 1 个目标设备上运行它们。
- TwinCAT 在 PLC 节点下的解决方案资源管理器视图中管理特定于项目的功能块。
- 一些模板包含用于创建项目的特定对象。
- 在项目设置和项目信息中可以定义项目的基本配置和信息。例如：
  - 编译器设置
  - 作者
- 您可以将项目另存为文件系统中的 1 个文件。您还可以选择将其与项目相关文件和信息一起打包到项目存档中。您也可以将其存储在 Microsoft Team Foundation Server（TFS）或 SVN 等源代码管理系统中。
- 每个项目都包含创建时使用的 TwinCAT 版本信息。如果您在不同的版本中打开它，TwinCAT 将通知您可能或必要的更新。
- 您可以比较项目并导出和导入单个对象。
- 您可以保护项目不被更改，甚至完全不被读取。通过用户管理功能，您可以有目的地控制用户对项目的访问，甚至对项目中的单个对象的访问。

4.1 创建标准项目

1. 在菜单 **File**（文件）中，选择命令 **New > Project**（新建 > 项目）。

   ⇨ 对话框 **New Project**（新建项目）会打开。

2. 选择模板 **TwinCAT Projects > TwinCAT XAE Project**（TwinCAT 项目 > TwinCAT XAE 项目）并输入 1 个名称（例如，此处为“Sample Project”（示例项目））和文件系统中的存储位置。
3. 选择 **OK**（确定），退出对话框。

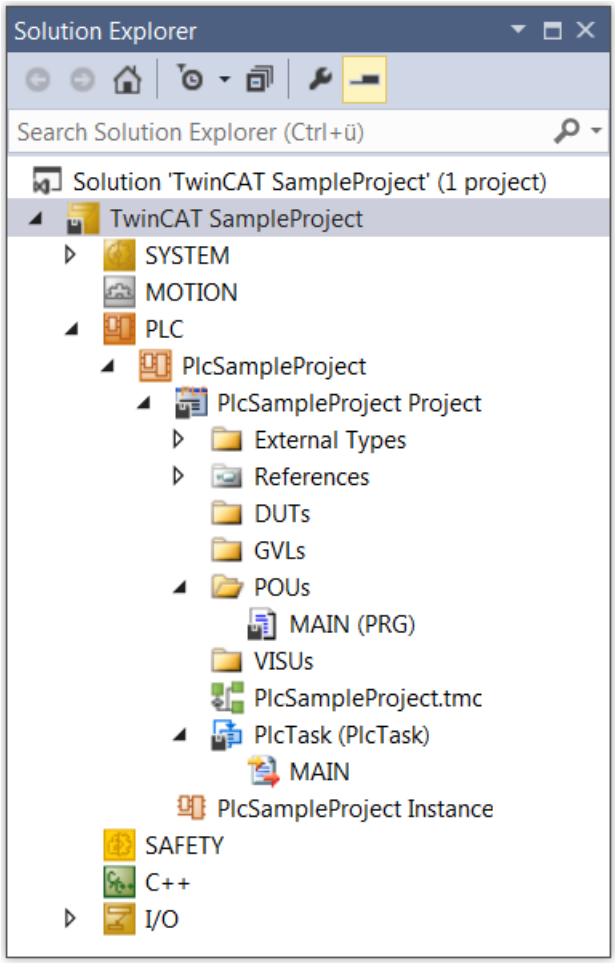
   ⇨ 在 **Solution Explorer**（解决方案资源管理器）中会打开 1 个新的解决方案。在主窗口的标题栏中会显示项目名称“TwinCAT Sample Project”（TwinCAT 示例项目）。

4. 在项目树中标记 PLC 对象，并在菜单 **Project**（项目）或上下文菜单中选择命令 **Add New Item...**（添加新项目……）。

   ⇨ 对话框 **Add New Item <TwinCAT project name>**（添加新项目 <TwinCAT project name>）会打开。

5. 在类别 **Plc Templates**（Plc 模板）中，选择 **Standard PLC Project**（标准 PLC 项目）并输入 1 个名称（此处仍为“Sample Project”（示例项目））。
6. 选择 **Add**（添加），退出对话框。

⇨ 在视图 **Solution Explorer**（解决方案资源管理器）中已创建以下结构。



⇨ 在选择模板之后，在 PLC 项目对象（**PlcSampleProject**）下会自动出现以下基本对象：1 个 PLC 项目（**PlcSampleProject project**）（PlcSampleProject 项目）和 1 个项目实例（**PlcSampleProject instance**）（PlcSampleProject 实例）。PLC 项目包含 1 个库管理器（**References**）（引用）、标准程序块 **MAIN** 和 1 个任务引用（**PlcTask**）。其中，引用的任务（**PlcTask**）定义了程序块 MAIN 的执行。此外，结构文件夹 **External Types**（外部类型）、**DUTS**、**GVLs**、**POUS** 和 **VISUs** 也会自动出现。
库管理器已包含带有基本块的标准库，例如，计数器、计时器和字符串函数，这些基本块随后可以用于编程。
如果您现在使用无误码填充 MAIN，则可将其加载到控制器上并运行，而无需进一步的编程对象。

另请参见：

- 对 PLC 项目进行编程 [▶ 60]
- 使用库 [▶ 246]

4.2 添加对象

以下说明介绍了在 PLC 项目中创建对象的一些可能性。

✓ 1 个 PLC 项目已打开。

1. 在 PLC 项目树中，选择 1 个条目，例如，文件夹 **POUs**。
2. 在上下文菜单中，选择命令 **Add**（添加）。
   ⇨ 根据在树中选择的条目，TwinCAT 会提供合适的对象以供选择。以下对象可供您使用：

- 对象 POU [▶ 71]
  - 对象方法 [▶ 82]
  - 对象属性 [▶ 88]
  - 对象动作 [▶ 78]
  - 对象转换 [▶ 79]

- 对象用于隐式检查的 POU [▶ 151]
- 对象 DUT [▶ 68]
- 对象全局变量列表 [▶ 66]
- 对象引用任务 [▶ 120]
- 对象配方管理器 [▶ 210]
  - 对象配方定义 [▶ 214]
- 对象图像池 [▶ 135]
- 对象接口 [▶ 94]
- 对象参数列表 [▶ 67]
- 对象“Text list” [▶ 133]（文本列表）
- 对象“Class Diagram [▶ 122]”（类图）
- 对象“Visualization Manager”（可视化管理器） [▶ 354]
- 对象“Visualization” [▶ 367]（可视化）

3. 例如，选择对象 **POU**，并在随后出现的对话框 **Add POU**（添加 POU）中选择类型 **Program**（程序），实现语言为“结构化文本（ST）”，名称为“Prog”。点击 **Open**（打开）。
   ⇨ 在所选条目下方的 PLC 项目树中，TwinCAT 会添加 1 个程序对象 **Prog**。
4. 在树中选择 1 个对象，并在上下文菜单中选择命令 **Properties**（属性）（对象）。
   ⇨ 包含对象相关类别的 **Properties**（属性）视图处于活动状态。例如，如果您使用用户管理功能，您可以在此处限制对对象的访问。
5. 在树中选择 1 个您想要在其下方创建文件夹以便收集其中对象的条目。
6. 在上下文菜单中，选择命令 **Add** > **New Folder** （添加 > 新建文件夹）。
   ⇨ 在 PLC 项目树中会出现文件夹。
7. 为该文件夹命名并确认。
8. 在 PLC 项目树中选择 1 个对象，然后用鼠标在项目树内部拖动该对象，将其移动到不同的位置，例如移动到文件夹中。

另请参见：

- TC3 用户界面文档：命令：添加现有项（对象）
- TC3 用户界面文档：命令属性（对象） [▶ 836]
- TC3 用户界面文档：命令：新文件夹

4.3 更改编译器版本

生成在目标设备上使用的当前项目代码所使用的编译器版本可以在项目属性中进行定义。

编译器版本与 TwinCAT 版本无关。因此，如果设置了相同的编译器版本，即使是从不同版本的 TwinCAT 进行设置，也会从源代码中生成恒定的程序代码。

如果您打开 1 个未设置最新编译器版本的项目，则会出现对话框 **Project Environment**（项目环境），其中包含相应的信息和直接更新的选项。

✓ 1 个 PLC 项目已打开。

1. 标记 PLC 项目，并在菜单 **Project**（项目）或上下文菜单中选择命令 **Properties**（属性）。
   ⇨ PLC 项目属性会在编辑器窗口中打开。
2. 选择类别 **Compile**（编译）。
3. 在组字段 **Solution options**（解决方案选项）中，选择所需的固定版本。

⇨ 变更立即生效。

另请参见：

- TC3 用户界面文档：命令属性（项目） > 类别编译 [▶ 844]

4.4 打开 TwinCAT 3 PLC 项目

✓ TwinCAT XAE 已启动。1 个 TwinCAT 项目已打开。

1. 在视图 **Solution Explorer**（解决方案资源管理器）中标记 PLC 对象，并在菜单 **Project** （项目）或上下文菜单中选择命令 **Add Existing Item** （添加现有项目）。
2. 在对话框 **Open**（打开）中，从文件系统中选择所需的 TwinCAT 3 PLC 项目或项目存档或库。如要进行搜索，您可以在对话框的右下角设置文件筛选器。
3. 点击 **Open**（打开）。

⇨ 可能出现以下情况：

- 您已选择 1 个使用较新的 TwinCAT 版本保存的项目：
  这样的项目可能包含无法加载的数据。您仍然可以打开该项目，但需要考虑以下情况：由于无法得到完全解释，该项目可能会以意想不到的方式运行。TwinCAT 无法完全加载或根本无法加载的对象会在视图 **Solution Explorer**（解决方案资源管理器）中被标记为红色，并标注文本“[unknown]”（[未知]）或“[incomplete]”（[不完整]）。TwinCAT 无法在编辑器中显示在当前版本中未知的对象。TwinCAT 会在编辑器中显示不完整的对象以及关于显示内容可能与原始内容不符的警告。TwinCAT 无法以原始名称保存不完整的项目。右上角的写保护说明会指明这一点。您能够以不同的名称保存文件。
- 您选择的项目在上次更改后未正确终止 TwinCAT，但项目选项 **Auto Save**（自动保存）已启用：
  现在，对话框 **Auto Save Backup**（自动保存备份）将会打开，以处理备份副本。

另请参见：

- 打开 TwinCAT 2 PLC 项目 [▶ 51]
- TC3 用户界面文档：命令项目/解决方案（打开项目/解决方案） [▶ 799]
- TC3 用户界面文档：命令添加现有项目（项目） [▶ 800]

4.5 打开 TwinCAT 2 PLC 项目

✓ TwinCAT XAE 已启动。1 个 TwinCAT 项目已打开。您应该了解以下指南中所述的限制条件。

1. 在 **Solution Explorer**（解决方案资源管理器）视图中标记 PLC 对象，并在 **Project**（项目）菜单或上下文菜单中选择 **Add Existing Item**（添加现有项目）命令。
2. 在 **Open**（打开）对话框中，从文件系统中选择所需的 Plc 2.x 项目或库。如要进行搜索，您可以在对话框的右下角设置文件筛选器。

   ⇨ 之后，TwinCAT 2.x 转换器会自动启动。
3. TwinCAT 2.x 转换器会检查是否可以对项目进行准确无误地编译。如果是这样的话，它会自动处理该项目。
   如果项目包含转换器无法解析的带有占位符变量的可视化对象，则会直接将相应的可视化对象集成为 1 个分组，以代替可视化引用。
4. 库转换：如果在要打开的项目中引用了尚未定义转换规则的库，则会出现对话框 **Converting a library reference**（转换库引用）。在此处可以定义转换器是否应该以及如何用最新的库引用替换先前的库引用。如果您在此过程中选择 1 个缺少项目信息的库，则会出现 **Project information**（项目信息）对话框，您必须填写该对话框。

⇨ 转换器会加载经过调整后的项目。

在 TwinCAT 3.1 中重复使用 TwinCAT 2.x 项目时的限制条件

自动语法调整仅适用于先前的编译功能

TwinCAT 2（TC2）和 TwinCAT 3（TC3）的语法在某些方面（例如，在初始化数组时）存在差异。请注意，如果要转换的 TC2 项目可以在 TC2 端进行编译，则转换器仅会在这些代码位置调整语法。

具体来说，这意味着：

在转换过程中，TC3 中包含的转换器会首先使用 TC2 编译器对选定的 TC2 项目进行编译。如果这种 TC2 端编译成功的话，才会在创建 TC3 SPS 项目期间对 TC2 代码进行语法调整。TC2 端编译成功的前提条件是，转换器可以使用所有必要的 TC2 库。例如，通过将 TC2 项目中的引用库插入到文件夹 C:\TwinCAT\\3.1\Components\\Plc\Converter\\Lib 中，或者通过显示的转换对话框向转换器通报 TC2 库的位置，都可以实现这一点。

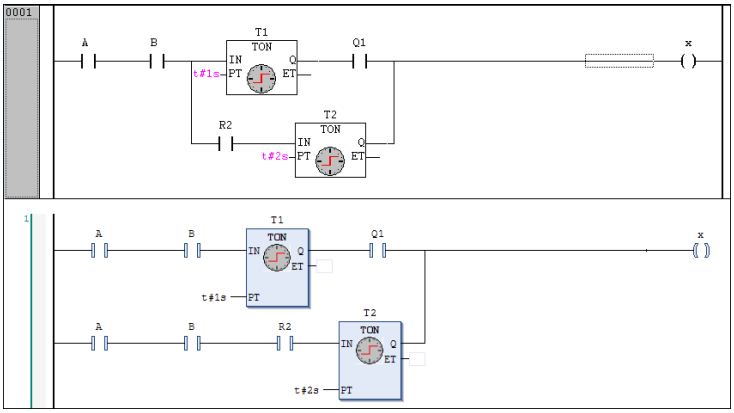
编译	务必能够在 TwinCAT 2.x PLC Control 中编译项目，而不会出现编译错误。尽管如此，TwinCAT 还是会在编译过程中发出警告。隐式转换可能会导致信息丢失（例如，由于符号的改变），从而引发这些问题。 TwinCAT 3.1 针对开关变量测试情况语句：在 TwinCAT 2.x 中未检查 CASE USINT OF INT，但在导入 TwinCAT 3.1 时会显示错误消息。
库	在库中使用的所有变量和常量也必须在此库中声明。务必能够在 TwinCAT 2.x 中编译库，而不会出现错误。

句法和语义限制	• FUNCTIONBLOCK 不再是替代 FUNCTION_BLOCK 的有效关键字
	• TYPE（结构声明）后必须带有 1 个“:”。
	• ARRAY 初始化必须用括号括起来。
	• 除了在 TYPE 中之外，不再可能实现枚举的局部声明：在 TwinCAT 2.x 导入的情况下，局部枚举声明会自动转换为显式类型定义。
	• 不再支持 INI（您必须在代码中将其替换为 Init 方法）。
	• 在函数调用中，不再可能将显式参数赋值与隐式赋值混用。因此，参数输入赋值的顺序可以更改：
	fun(formal1 := actual1, actual2); // → Fehlermeldung
	fun(formal2 := actual2, formal1 := actual1); // gleiche Semantik wie folgende Zeile:
	fun(formal1 := actual1, formal2 := actual2);
	• TwinCAT 2.x 编译指示不会转换。它们会在 TwinCAT 3.1 中生成警告。
	• TRUNC 运算符现在可以转换为数据类型 DINT，而不是 INT；在 TwinCAT 2.x 导入的情况下，TwinCAT 3.1 会自动添加适当的类型转换。

内存	• 与 TwinCAT 2 相比，TwinCAT 3.1 采用 8 字节对齐方式。有关更多详细信息和示例，请参见引用编程 > 对齐方式 [▶ 731]。 • 在 TwinCAT 2 中，数据类型 STRING 的实例通过将整个内存区清零的方式进行预初始化。不过，在 TwinCAT 3.1 中，只有此类实例的第 1 个字节会被清零。由于清零终止，实例也是空的。

可视化

占位符及其替换	占位符	VAR_INPUT	使用	替换
	MAIN. $LocalVar$.aArr[0]	localVar: MyStruct;	localVar.aArr[0]	localVar := MAIN.myStructVar
	Var.aArr[0]	Var : MyStruct;	Var.aArr[0]	Var := MAIN.myStructVar
	MAIN.myStructVar.a Arr[$Index$]	Index : INT;	MAIN.myStructVar.a Arr[Index]	Index := 0
有问题的占位符	• 文本内的占位符： 文本：$axle$-Axis 校正： localVar : STRING; 文本：%s-Axis TextVariable : localVar • 占位符仅描述变量名的一部分： axis$axis$spur$spur$.fActPosition 校正： 只为占位符 axis$axis$spur$spur$ 定义 1 个占位符。 axis_spur : MyFunctionBlock; 然后，直接传输功能块的相应实例。 axis_spur := MAIN.axis1spur2; • 占位符由表达式替换： $Expression$ → MAIN.var1 + MAIN.var2 校正： 您必须将表达式传输至帮助变量，然后将此帮助变量作为实例进行传输。 • 占位符描述程序名称：$Program$.bToggle → MAIN.bToggle D 转换器无法将这种形式的占位符传输至 TwinCAT 3.1。然而，在实践中却很少使用。 • 占位符由各种类型替换： Var → 替换 1：MAIN.n (INT) → 替换 2：MAIN.st (STRING) 校正： 在界面中针对此情况定义 2 个不同的占位符。 • 可视化位于 1 个库内。稍后，您从任何使用可视化的项目中替换占位符。 校正： 您必须在此处手动替换 TYPE_NONE 数据类型。不过，您也可以将库集成到 1 个项目中，并正确地替换占位符。如果您现在导入此项目，也要在库中正确确定数据类型。			
不可导入的元素	趋势、ActiveX - 无法导入，因为实现方式迥然不同。在 TwinCAT 3.1 中会发出相应的警告，并需要进行相应的手动复制。			

编程语言	ST、IL、FBD	没有限制
	LD	TwinCAT 3.1 导入的功能块具有平行分支，因此每个分支都会重复分支前的部分。这与 TwinCAT 2.x 为平行分支生成的代码相对应。 
	SFC	• 用户明确声明的步骤变量必须在 SFC 编辑器中进行局部声明。您不得将它们声明为 VAR_INPUT、VAR_OUTPUT 或 VAR_INOUT，因为 TwinCAT 3.1 无法自动调整调用。说明：在 TwinCAT 3.1 中，步骤不再使用 Boolean 变量来管理内部状态，而是使用 SFCStepType 类型的结构。 • 标识符：以下标识符不得以下划线开头： ◦ 树中的 IEC 动作的名称 ◦ 在 IEC 关联列表中调用的变量 ◦ 已编入程序的转换的名称 说明：在 TwinCAT 3.1 中，TwinCAT 3.1 为动作创建的隐式变量均以下划线作为前缀。这将产生带有双下划线的无效标识符。
	CFC	• 大型功能块：大型功能块的布局可能会因导入而造成质量损失；功能块框可能会显著重叠。 • 宏：无法导入宏。

另请参见：

- TC3 用户界面文档：命令添加现有项目（项目） [▶ 800]

4.6 配置 PLC 项目

您可以在以下对话框和视图中配置 TwinCAT PLC 项目：

- TwinCAT 选项：针对编辑器行为的与项目无关的常规设置
- PLC 项目属性：与编译器等有关的项目相关属性、信息和设置
- PLC 项目设置：常规项目设置

另请参见：

- TC3 用户界面文档：命令属性（PLC 项目） [▶ 841]
- TC3 用户界面文档：命令：选项 [▶ 897]
- TC3 用户界面文档：PLC 项目设置 [▶ 858]

4.7 使用全局数据类型

如果不仅要在 PLC 项目内部使用数据类型，还要跨 PLC 项目使用数据类型，则可将这些数据类型转换为全局数据类型。在系统管理器的类型系统中可以管理，并将全局数据类型自动提供给 PLC 项目。

在 build 4026 之前，使用“超全局”标志可以将全局数据类型传递给 PLC 项目。这样，它们不仅可以在 PLC 项目本身中使用，也可以在其中使用的库中使用。

使用 build 4026，可以自动创建虚拟数据类型库并将其添加到 PLC 项目以及在其中使用的库中。顶层是 Tc3_GlobalTypes，它可以将实际的数据类型库置于其下。所有带有命名空间的数据类型都会被分组至带有该命名空间的相应库中，所有其他数据类型都会被添加至 Tc3_GlobalType_Global 库下方。

5 导出和传输 PLC 项目

您可以使用导出和导入功能将 TwinCAT 项目中的数据与其他程序进行交换。

TwinCAT 开发系统之间的 TwinCAT 项目交换通过项目文件（*.project）或项目存档（*.projectarchive）的副本进行。

5.1 导出和导入 PLC 项目

TwinCAT 提供将对象导入或导出文件的命令。此处提供一些选项：

- 导出至 ZIP 文件（*.zip）或从该文件导入
  您可以使用此格式导出和导入多个文件（包括非 TwinCAT 项目文件），包括路径。
- 导出至项目存档（*.zip）
  您可以使用此格式将整个项目（包括使用的库）导出至 ZIP 存档。
- 导出至 PLCopen 格式（*.xml）的 XML 文件或从该文件导入
  您可以使用此格式与其他程序（例如，程序编辑器或文档工具）交换信息。PLCopen XML 定义了 TwinCAT 中已知元素的子集。因此，无法确保 100% 的兼容性。

ZIP 导出或导入的方式与此处所述的 PLCopenXml 导出或导入相同。

导出项目

✓ 在 TwinCAT 中已打开 1 个项目。

1. 标记您想要导出的 PLC 对象或 PLC 项目。
2. 在上下文菜单中，选择命令 **Export PLCopenXML...**（导出 PLCopenXML……）
3. 在对话框 **Export PLCopenXML file**（导出 PLCopenXML 文件）中，选择您想要保存导出文件的内存位置或文件名。
4. 点击 **Save**（保存）进行确认。

⇨ 导出的文件可供您在所选的内存位置使用。

导入项目

✓ 在 TwinCAT 中已打开 1 个项目。

1. 标记要将文件导入的 PLC 项目或 PLC 项目中的文件夹。
2. 在上下文菜单中，选择命令 **Import PLCopenXML...**（导入 PLCopenXML……）
3. 在对话框 **Import PLCopenXML file**（导入 PLCopenXML 文件）中，选择您想要导入的文件。
   ⇨ 此时会打开 1 个对话框，以树状结构显示可插入的对象。
4. 选择对象并点击 **Open**（打开）。

⇨ 对象会被插入现有项目树中。

另请参见：

- TC3 用户界面文档：命令：导出至 ZIP
- TC3 用户界面文档：命令：从 ZIP 导入
- TC3 用户界面文档：命令：导出 PLCopenXML
- TC3 用户界面文档：命令：导入 PLCopenXML
- TC3 用户界面文档：对话框选项 - PLCopenXML [▶ 906]
- TC3 用户界面文档：对话框选项 - ZIP 导出/导入 [▶ 923]

5.2 传输 PLC 项目

如果您将项目传输至另一台计算机并希望从该计算机上与同一个 PLC 连接，而无需在线更改或下载，请注意以下几点：

- 确保项目只需要固定版本的库（例外：接口库）、可视化配置文件和编译器。
- 确保启动项目是最新的。

创建 1 个项目存档，然后在另一台计算机上将其解压，或者在源代码管理系统中检查该项目。

将项目传输至不同的系统

✓ 在计算机“PC1”上，打开您希望将其传输至另一台计算机“PC2”的项目，而且，您希望在此处再次与同一个控制器连接。

✓ 只有版本固定的库才会集成到项目中（例外：纯接口库）。如要检查这一点，打开库管理器。如果 1 个库条目旁边不是固定版本 ID，而是 1 个“*”，则表示该库未与固定版本集成。（请参见“使用库 [▶ 246]”部分）

✓ 打开的 PLC 项目与当前在 PLC 上使用的项目相同。这意味着“启动项目”与编程系统中的项目完全相同。如果 PLC 项目树中的 PLC 项目和 PLC 对象的软磁盘符号为红色，这意味着该项目或 PLC 对象已更改，但尚未保存。在这种情况下，PLC 项目和启动项目可能不匹配。保存更改并生成新的启动项目。如要明确生成启动项目，可在 PLC 节点的上下文菜单中选择命令 **Activate Boot Project**（激活启动项目）。使用命令 **Login**（登录）可登录 PLC，使用命令 **Start**（开始）可开始执行。现在，项目已在您希望与 PC2 上的同一个项目再次连接的 PLC 上运行。

1. 在 PLC 项目的上下文菜单中，选择命令 **Save <TwinCAT project name> as Archive...**（将 <TwinCAT project name> 另存为存档……），以生成项目存档或在您的源代码管理系统中检查整个项目。
2. 将项目存档传输至 PC2 并将其提取出来，或将您的源代码管理系统中的最新版本加载到 PC2 中。
3. 现在，打开 TwinCAT 项目或将 PLC 集成到 1 个新的 TwinCAT 3 项目中。
4. 编译项目。
5. 再次登录 PLC。

⇨ TwinCAT 不要求在线更改或下载。项目运行。

另请参见：

- TC3 用户界面文档：命令：激活启动项目
- TC3 用户界面文档：命令登录 [▶ 889]
- TC3 用户界面文档：命令将 <TwinCAT project name> 另存为存档…… [▶ 988]
- TC3 用户界面文档：命令：开始 [▶ 890]

6 本地化 PLC 项目

如果您创建并集成本地化文件，您可以用不同的语言显示项目。本地化文件与 GNU“gettext”系统的文件相对应。本地化模板文件是 *pot 文件（可移植对象模板），在翻译后，将根据这些文件创建 *.po 格式（可移植对象）的本地化文件。

虽然项目可以用不同的语言呈现，但仅可在原始版本中进行编辑。

您可以配置您想要本地化的项目中包含的文本信息类别。然后，您可以将这些文本导出到翻译模板中。该模板是 1 个格式为 pot 的文件（例如，“project_1.pot”）。该模板可作为使用合适的外部翻译工具创建或使用简单的文本编辑器手动创建 po 格式的本地化文件（例如，“de.po”、“en.po”、“es.po”）的基础。然后，您可以将 po 文件导回到 TwinCAT 中并将其用于本地化。在菜单 **PLC > Project Localization** （PLC > 项目本地化）菜单中可以找到管理项目本地化的命令。

生成本地化模板

✓ 1 个项目已打开。

1. 在菜单 **PLC > Project Localization**（PLC > 项目本地化）中，选择命令 **Create Localization Template**（创建本地化模板）。

   ⇨ **Generating a localization template…**（生成本地化模板……）对话框会打开。

2. 选择您想要包含在本地化模板中的文本信息类别。
3. 模板中还可以包含“Position information”（位置信息）。对于每个要翻译的文本，您可以在项目中为其指定位置。您可以在此处选择是否在翻译模板中只显示找到的文本的第 1 个实例、所有实例或不显示任何实例。
4. 点击 **Create**（创建）。

⇨ 用于以 pot 格式保存文件的对话框打开。保存本地化模板。然后，您可以在翻译工具中编辑该文件，并以所需语言创建 <*Language*>.po 本地化文件。

本地化模板的格式（文件 *.pot）

第 1 行显示在创建模板时已选择哪些文本类别用于翻译：

示例：

#: Content:Comments|Identifiers|Names|Strings：所有 4 个类别均被选中。

对于每个要翻译的文本，后面都有 1 个部分，具体如下方示例所示：

示例：

```
#: D:\Projects\p1.project\Project_Settings:1
msgid "Project Settings"
msgstr ""
```

第 1 行：作为源代码引用的位置信息：只有在生成翻译文件时已进行配置的情况下，才会显示该信息。

第 2 行：作为条目 msgid 的未翻译的文本。示例：msgid ″Project Settings″。

第 3 行：译文的占位符：msgstr ″″。然后，务必在 po 文件的引号之间插入相应语言的译文。

本地化文件的格式（文件 *-<Language>.po）

您可以使用翻译工具创建 po 文件，也可以根据 pot 文件使用中性文本编辑器手动创建该文件。您可以将 pot 文件重命名为 po 文件，然后按照标准 po 格式对其进行编辑。请务必在文件的元数据中以常用的语言标签形式指定语言。

例如：″Language: de″ 代表德语。

将单个文本的译文输入引号之间的 msgstr ″″ 条目中。

示例：

```
"Language: de\n"
#: Content:Names
#: D:\projects\p1.project\Project_Settings:1
msgid "Project Settings"
msgstr "Projekteinstellungen"
```

导入本地化文件（本地化项目）

✓ 根据 *.pot 本地化模板，为您的项目已创建本地化文件 *<language>.po*。项目已打开。

1. 在菜单 **PLC** > **Project Localization**（PLC > 项目本地化）中，选择命令 **Manage Localization…**（管理本地化……）
2. 点击 **Add**（添加）。
   ⇨ 用于从文件系统中选择 po 文件的 **Open Localization File**（打开本地化文件）对话框会出现。
3. 从本地化文件中选择 1 个，例如，“<project name>-de.po”。
4. 对话框会关闭，受影响的文本将以相应的语言出现在项目中。例如，如果您在德语本地化文件中为功能块名称 MAIN 输入了译文 msgstr "Hauptprogramm"，那么，在 PLC 项目树中就会出现对象名称 **Hauptprogramm**。
5. 用同样的方法，导入存在译文的其他语言的本地化文件。

更改本地化（添加和删除本地化文件）

✓ 在导入相应的 po 文件后，所有需要的语言都会存储在项目中。

✓ 项目已打开。

1. 在菜单 **Project** > **Project Localization**（项目 > 项目本地化）中，选择命令 **Manage Localization…**（管理本地化……）。
   ⇨ **Manage Localization**（管理本地化）对话框会打开。在 **Files**（文件）下会出现所有存储的本地化文件 **-<language>.po* 和条目 **<original version>**。
2. 选择所需的语言并点击 **Switch Localization**（切换本地化）。

⇨ 项目以所选语言显示。如果您选择 **<original version>**，项目会显示为非本地化的原始版本，您可以再次编辑。

可选：指定标准本地化（切换本地化）

从可用的本地化中选择 1 个，并启用选项 **Standard Localization**（标准本地化）。

使用菜单 **PLC** > **Project Localization**（PLC > 项目本地化）中的命令 **Toggle Localization**（切换本地化），在标准本地化和原始版本之间切换本地化。默认情况下，通过工具栏中的  按钮也可以使用该命令。

另请参见：

- TC3 用户界面文档：命令：创建本地化模板 [▶ 895]
- TC3 用户界面文档：命令：管理本地化 [▶ 896]
- TC3 用户界面文档：命令：切换本地化 [▶ 896]

7 对 PLC 项目进行编程

如要创建可在控制器上运行的应用程序，可在 POU 中填入声明和实现代码（源代码），将控制器的 I/O 与程序变量连接起来，并配置任务分配。经过检查和调试后，编译器会创建可加载到控制器上的程序代码。

项目块（POU）的编程由编程语言编辑器和某些进一步的功能（例如，编译指示和重构）以及 TwinCAT 3 PLC 库中现成功能块的应用提供支持。

现有的工具可用于语法检查和代码分析、创建数据持久性以及对加载到控制器上的程序代码进行加密。

另请参见：

- 您的第 1 个 TwinCAT 3 PLC 项目 [▶ 23]

7.1 分配标识符

标识符是变量和编程对象（例如，程序、功能块、方法等）的名称以及其他 PLC 项目对象的名称。在分配标识符时，您必须遵守特定的规则。此外，还有一些建议有助于使标识符保持一致并具有意义。

变量标识符在变量声明阶段进行分配。您可以在编程对象的声明部分更改这些标识符。编程和其他对象的标识符在添加相应的对象时在对话框中进行分配。您可以在对象的属性窗口中更改现有 PLC 项目对象的标识符。您无法修改对象的标识符，此类标识符仅在每个 PLC 项目中存在 1 次。这包括 References 和 RecipeManager 标识符。

另请参见：

- 引用编程 > 标识符 [▶ 782]
- TwinCAT 3 编程规范 >

7.2 声明变量

变量声明

您可以在以下位置声明变量：

- 编程对象的声明部分
- GVL 编辑器

对话框 **Auto Declare**（自动声明）支持您进行变量声明。

语法

```
( <pragma> )*
<scope> ( <type qualifier> )?
      <identifier> (AT <address> )? : <data type> ( := <initial value> )? ;
END_VAR
```

<pragma>（可选）	编译指示（完全没有、1 次或多次） 通过添加编译指示可以影响 1 个或多个变量的属性。	另请参见 • 使用编译指示 [▶ 126]
<scope>	范围 • VAR • VAR_CONFIG • VAR_EXTERNAL • VAR_GLOBAL • VAR_INPUT • VAR_INST • VAR_IN_OUT • VAR_OUTPUT • VAR_STAT • VAR_TEMP	另请参见 • 变量 [▶ 623]
<type qualifier>（可选）	类型限定符 • CONSTANT • RETAIN • PERSISTENT	另请参见 • 变量类型 - 属性关键字 [▶ 635]
<identifier>	标识符，变量名 在分配标识符时，务必遵循在“标识符”部分所列的规则。在“标识符/名称”部分，您可以看到更多统一规范。	另请参见 • 标识符 [▶ 782] •
AT <address>（可选）	在输入、输出或标志内存区（I、Q 或 M）中分配地址 示例：AT %I*，AT %Q*	另请参见 • AT 声明 [▶ 63] • 地址 [▶ 695]
<data type>	数据类型 • <elementary data type> • <user defined data type> • <function block>	另请参见 • 数据类型 [▶ 697]
<initial value>（可选）	初始值 • <literal value> • <identifier> • <expression>	另请参见 • 操作数 [▶ 686] • ST 表达式 [▶ 572]
(...)?	可选	
(…)*	可选重复	

变量初始化

所有声明的标准初始化值都是 0。在声明部分，您可以为每个变量和每个数据类型指定用户定义的初始化值。

用户定义的初始化从分配运算符 := 开始，由编程语言 ST（结构化文本）中的有效表达式组成。因此，您可以借助常量、其他变量或函数来定义初始化值。如果您使用变量，您还必须对其进行初始化。

示例 1：

```
VAR
    nVar1   : INT := 12;                    // initialization value 12
    nVar2   : INT := 13 + 8;                //
initialization value defined by an expression of constants
    nVar3   : INT := nVar2 + F_Fun(4);      //
initialization value defined by an expression that contains a function call; notice the order!
    pSample : POINTER TO INT := ADR(nVar1); //
```

```
not described in the standard IEC61131-3: initialization value defined by an adress function; Notice
: the pointer will not be initialized during an Online Change
END_VAR
```

示例 2：

在下面的示例中，1 个输入变量和 1 个属性由 1 个功能块初始化，该功能块的 FB_init 方法 [▶ 787]带有 1 个附加参数。

功能块 FB_Sample：

```
FUNCTION_BLOCK FB_Sample
VAR_INPUT
    nInput           : INT;
END_VAR
VAR
    nLocalInitParam  : INT;
    nLocalProp       : INT;
END_VAR
```

方法 FB_Sample.FB_init：

```
METHOD FB_init : BOOL
VAR_INPUT
    bInitRetains : BOOL;  // if TRUE, the retain variables are initialized (warm start / cold start)
    bInCopyCode  : BOOL;  // if TRUE, the instance afterwards gets moved into the copy code (online
change)
    nInitParam   : INT;
END_VAR
```

```
nLocalInitParam := nInitParam;
```

属性 FB_Sample.nMyProperty 和相关的 Set 函数：

```
PROPERTY nMyProperty : INT
```

```
nLocalProp := nMyProperty;
```

MAIN 程序：

```
PROGRAM MAIN
VAR
    fbSample  : FB_Sample(nInitParam := 1) := (nInput := 2, nMyProperty := 3);
    aSample   : ARRAY[1..2] OF FB_Sample[(nInitParam := 4), (nInitParam := 7)]
                            := [(nInput := 5, nMyProperty := 6), (nInput := 8, nMyProperty := 9)];
END_VAR
```

初始化结果：

- fbSample
  - nInput = 2
  - nLocalInitParam = 1
  - nLocalProp = 3
- aSample[1]
  - nInput = 5
  - nLocalInitParam = 4
  - nLocalProp = 6
- aSample[2]
  - nInput = 8
  - nLocalInitParam = 7
  - nLocalProp = 9

另请参见：

- 引用编程 > 数据类型 [▶ 697]
- 引用编程 > 变量类型和特殊变量 [▶ 623]
- 引用编程 > 操作数 > 地址 [▶ 695]
- 引用编程 > 标识符 [▶ 782]
- TwinCAT 3 编程规范 >

7.2.1 AT 声明

如要将项目变量绑定到灵活（*）或直接地址，您可以在声明变量时指定地址。通过使用合适的变量名，您可以给地址起 1 个有意义的名字。

自动寻址

建议不要对已分配变量使用直接寻址，而应使用 * 占位符。
如果使用占位符 *（%I*、%Q* 或 %M*），TwinCAT 将自动执行灵活的优化寻址。

语法：

```
<identifier> AT <address> : <data type>;
```

如果已经指定地址，则可以通过特殊字符串表示内存中的位置和大小。地址用百分号 % 标记，然后是内存区前缀、可选大小前缀以及内存位置。

```
%<memory area prefix> ( <size prefix> )? <memory position>
```

```
<memory area prefix> : I | Q | M
<size prefix>        : X | B | W | D
<memory position>    : * | <number> ( .<number> )*
```

内存区前缀

I	用于输入的输入内存区 通过输入驱动器（“传感器”）进行物理输入
Q	用于输出的输出内存区 通过输出驱动器（“执行器”）进行物理输出
M	标志内存区

大小前缀

X	单个位
B	字节（8 位）
W	字（16 位）
D	双字（32 位）

示例：

```
IbSensor1 AT%I*    : BOOL;
IbSensor2 AT%IX7.5 : BOOL;
```

如果您没有明确指定单个位地址，则会逐字节分配 Boolean 变量。示例：bVar AT %QB0 的值变化涉及从 QX0.0 至 QX0.7 的区域。

如果将 1 个变量分配给 1 个地址，您必须遵守以下规定：

- 在实现编辑器中，您无法写入访问通过直接寻址分配给输入的变量。这会导致编译器出错。
- 如果您使用 AT 声明并直接寻址到结构或功能块组件，则所有实例都会使用相同的内存。这与以经典编程语言（例如“C”）使用静态变量 [▶ 627]的情况相对应。
- 结构的内存布局同样取决于目标系统。

有效地址

关键字 AT 必须跟在有效地址之后。有关此方面的更多信息，请参见“引用编程 > 操作数 > 地址 [▶ 695]”部分。请注意在字节寻址模式下可能出现的重叠。

示例

变量声明：

IbSensor AT%I* : BOOL;	在指定地址时，指定的是占位符 *，而不是内存位置。这使得 TwinCAT 能够自动执行灵活的优化寻址。
InInput AT%IW0 : WORD;	变量声明，1 个输入字的地址指定
ObActuator AT%QB0 : BOOL;	Boolean 变量声明 注意：对于 Boolean 变量，如果没有指定单个位地址，则会在内部分配 1 个字节。因此，ObActuator 的值变化会影响从 QX0.0 至 QX0.7 的范围。
IbSensor AT%IX7.5 : BOOL;	Boolean 变量声明，明确指定单个位地址。在访问时，只读取输入位 7.5。

其他地址：

%QX7.5 %Q7.5	输出位 7.5 的单个位地址
%IW215	输入字 215 的字地址
%QB7	输出字节 7 的字节地址
%MD48	标志区中内存位置 48 的双字地址
%IW2.5.7.1	解释取决于当前控制器配置（见下文）

另请参见：

- 引用编程 > 操作数 > 地址 [▶ 695]

7.2.2 使用声明编辑器

声明编辑器用于在变量列表和 POU 中声明变量。

如果声明编辑器与编程语言编辑器一起使用，则会在 POU 窗口的上部显示为声明部分。

声明编辑器提供 2 种可能的视图：文本视图（ ）或表格视图（ ）。在对话框 **Tools > Options > TwinCAT > PLC Environment > Declaration editor**（工具 > 选项 > TwinCAT > PLC 环境 > 声明编辑器）中，您可以定义是否仅提供文本视图或表格视图，或者用户是否可以通过编辑器窗口右侧的按钮在 2 种视图之间进行选择。

另请参见：

- 引用编程：声明编辑器 [▶ 569]

通过文本声明编辑器进行声明

✓ 1 个项目的编程对象（POU 或 GVL）已打开。焦点在于文本声明编辑器。

1. 用正确的语法输入变量声明。使用 **[F2]** 或上下文菜单中的 **Input Assistant**（输入助手）选项，打开 **Input Assistant**（输入助手）对话框，以选择数据类型或关键字。使用上下文菜单中的 **Auto Declare**（自动声明）命令，打开 **Auto Declare**（自动声明）对话框。

⇨ 在声明变量时，关键字会自动校正并突出显示。

另请参见：

- 使用自动声明对话框 [▶ 65]
- TC3 用户界面文档：命令：输入助手 [▶ 810]
- TC3 用户界面文档：命令：自动声明 [▶ 811]

通过表格声明编辑器进行声明

✓ 1 个项目的编程对象（POU 或 GVL）已打开。焦点在于表格声明编辑器。

1. 点击声明标题中的按钮 或在上下文菜单中选择 **Auto Declare**（自动声明）命令，打开 **Auto Declare**（自动声明）对话框。

   ⇨ TwinCAT 为变量声明插入新行，并打开变量名的输入字段。

2. 输入 1 个有效的变量标识符。

3. 根据需要，通过双击打开声明行的其他字段，从选择列表中或者通过显示的对话框选择所需的信息。

⇨ 在声明变量时，会自动使用正确的语法。

另请参见：

- TC3 用户界面文档：命令：自动声明 [▶ 811]

7.2.3 使用自动声明对话框

✓ 1 个项目的编程对象（POU 或 GVL）已打开。

1. 在编辑器的 **Edit**（编辑）菜单或上下文菜单中，选择 **Auto Declare**（自动声明）命令。
   ⇨ 对话框 **Auto Declare**（自动声明）会打开。
2. 为 **Scope**（范围）选择列表中的变量选择所需的有效范围。
3. 在 **Name**（名称）字段中，输入变量名。
4. 从 **Type**（类型）选择列表中，选择所需的数据类型。
5. 如要使用不同于标准初始化值的初始化值，可为变量输入 1 个初始化值。
6. 点击 **OK**（确定）完成输入。
   ⇨ TwinCAT 会在您的编程对象的声明部分中列出新声明的变量。

您可以在声明部分中使用编译指示，以影响编译器对声明的处理。（请参见“使用编译指示 [▶ 126]”部分）

另请参见：

- TC3 用户界面文档：命令：自动声明 [▶ 811]

7.2.4 声明数组

✓ 1 个项目的编程对象（POU 或 GVL）已打开。

1. 在编辑器的 **Edit**（编辑）菜单或上下文菜单中，选择 **Auto Declare**（自动声明）命令。
   ⇨ **Auto Declare**（自动声明）对话框会打开。
2. 为 **Scope**（范围）选择列表中的数组选择所需的范围。
3. 在 **Name**（名称）输入字段中，输入数组的标识符。
4. 点击 **Type**（类型）输入字段旁边的  ，并从选择菜单中选择条目 **Array Wizard**（数组向导）。
5. 在 **Dimension 1**（维度 1）输入字段中，为数组的第 1 个维度输入索引下限和索引上限，例如：1 和 3。
   ⇨ **Result**（结果）字段会显示数组的第 1 个维度，例如：ARRAY [1..3] OF ?。
6. 在输入字段 **Base Type**（基本类型）中，直接或者通过 **Input Assistant**（输入助手）或 **Array Wizard**（数组向导）（按钮  ）输入数组的数据类型，例如：DINT。
   ⇨ **Result**（结果）字段会显示数组的数据类型，例如：ARRAY [1..3] OF DINT。
7. 根据步骤 5 和 6 定义数组的维度 2 和 3，例如：维度 2：1 和 4，维度 3：1 和 2。
   ⇨ **Result**（结果）字段会显示带有定义维度的数组：Array [1..3, 1..4, 1..2] OF DINT。数组由 3 * 4 * 2 = 24 个元素组成。

对于可变长度的数组，用星号占位符 * 声明维度边界。仅可在功能块、方法或函数的 VAR_IN_OUT 声明中使用可变长度的数组。

可变长度的二维数组示例：aVariableLength : ARRAY [* , *] OF INT;

8. 点击 **OK**（确定）。
   ⇨ 在 **Auto Declare**（自动声明）对话框中，**Type**（类型）字段会显示数组。
9. 如要更改数组的初始化值，可点击 **Initialization Value**（初始化值）输入字段旁边的  。

⇨ **Initialization Value**（初始化值）对话框会打开。

10. 选择您想要更改初始化值的数组元素的行。示例：选择数组元素 [1, 1, 1]。
11. 在列表下方的输入字段中输入所需的初始化值，并点击 **Apply value to selected lines**（将值应用于所选行），例如：“值 4”。

    ⇨ TwinCAT 会显示所选行的修改后的初始化值。
12. 点击 **OK**（确定）。

    ⇨ TwinCAT 会在 **Auto Declare**（自动声明）对话框的 **Initialization**（初始化）字段中显示数组的初始化值，例如：[4, 23(0)]。
13. 如果需要，您可以在输入字段中输入注释（可选）。
14. 点击 **OK**（确定）完成数组声明。

⇨ TwinCAT 会将数组声明添加到编程对象的声明部分。

另请参见：

- TC3 用户界面文档：命令：自动声明 [▶ 811]
- 引用编程 > 数组 [▶ 714]

7.2.5 声明全局变量

全局变量列表用于声明和编辑全局变量。参数列表用于在库中声明全局常量。

另请参见：

- 引用编程 > 剩余变量 - RETAIN, PERSISTENT [▶ 630]

7.2.5.1 对象全局变量列表

符号： 

全局变量列表用于声明、编辑和显示全局变量。如果您将 GVL 添加到项目中，则变量适用于整个项目。

对象创建全局变量列表

1. 在 PLC 项目树的 **Solution Explorer**（解决方案资源管理器）中，选择文件夹 **GVLs**。
2. 在上下文菜单中，选择命令 **Add** > **Global Variable List**（添加 > 全局变量列表）。
3. 输入名称并点击 **Open**（打开）。

   ⇨ TwinCAT 会将 GVL 添加到 PLC 项目树中，并在编辑器中打开它。您可以在关键字 VAR_GLOBAL 和 END_VAR 之间定义全局变量。

定义全局变量

✓ 1 个 GVL 已在编辑器中打开。

1. 用正确的语法输入变量声明，或者在编辑器的 **Edit**（编辑）菜单或上下文菜单中选择 **Auto Declare**（自动声明）命令。

   ⇨ **Auto Declare**（自动声明）对话框会打开。在 **Scope**（范围）选择列表中，选择条目 VAR_GLOBAL。
2. 在 **Name**（名称）字段中，输入全局变量的名称。
3. 在 **Type**（类型）选择列表中，选择数据类型。
4. 如果变量的初始化值与标准初始化值不同，可点击初始化值字段旁边的  。

   ⇨ **Initialization Value**（初始化值）对话框会打开。
5. 双击您的变量的 **Initialization**（初始化）单元格，并输入所需的有效值。
6. 点击 **OK**（确定）。

   ⇨ 在 **Auto Declare**（自动声明）对话框中会显示初始化值。
7. 如果需要，激活其中 1 个标志。
8. 点击 **OK**（确定），确认您的输入。

⇨ TwinCAT 会在 GVL 中添加已声明的变量。在整个 PLC 项目中可以使用该全局变量。

另请参见：

- TC3 用户界面文档：命令：自动声明 [▶ 811]

7.2.5.2 对象参数列表

符号：

如果您想要在 PLC 项目的后期阶段配置由库提供的全局常量，则可在参数列表中定义这些常量。参数列表是 1 种特殊类型的全局变量列表。

创建对象参数列表

✓ 1 个库项目已打开。

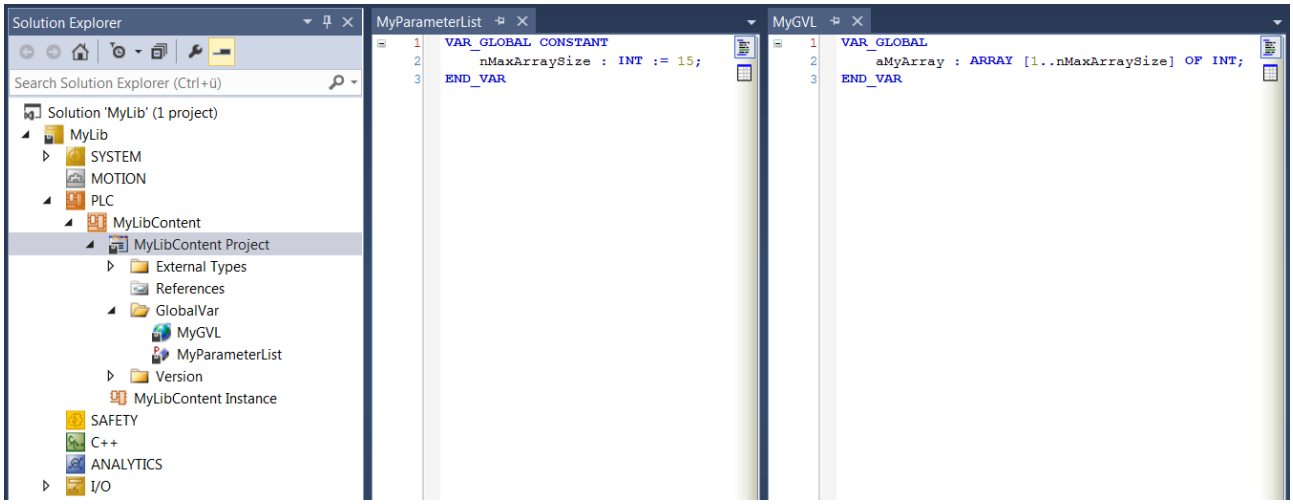
1. 在 **Solution Explorer**（解决方案资源管理器）中，在 PLC 项目树中选择 PLC 项目 <Projectname.project>。
2. 在上下文菜单中，选择命令 **Add** > **Parameter List…**（添加 > 参数列表……）
3. 输入名称并点击 **Open**（打开）。

⇨ TwinCAT 会将参数列表添加到 PLC 项目树中，并在编辑器中打开它。您可以在关键字 VAR_GLOBAL CONSTANT 和 END_VAR 之间定义全局常量。

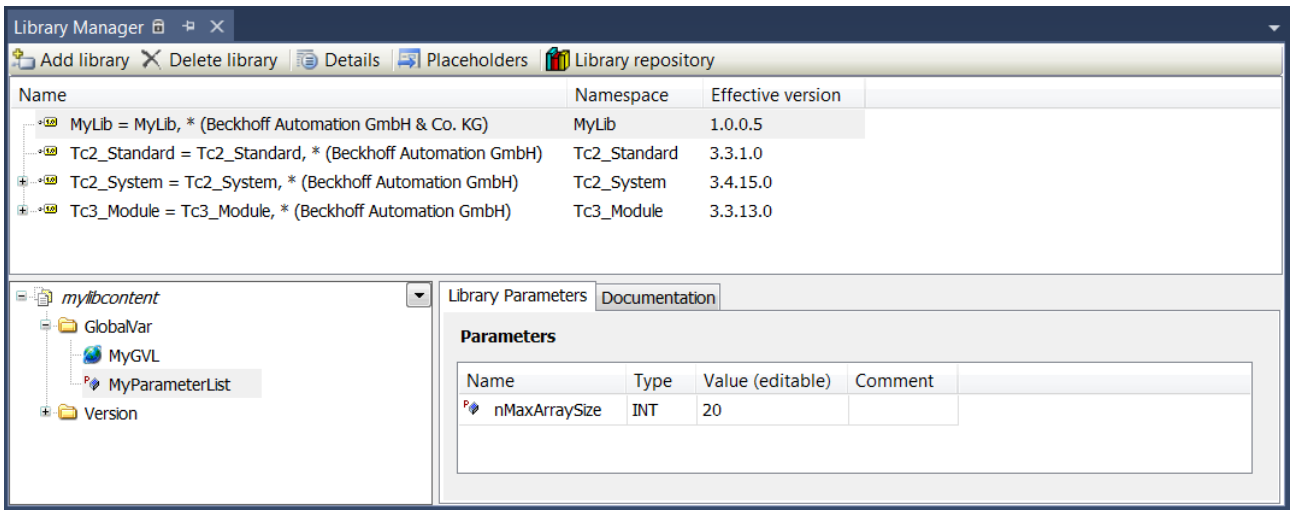
示例：

MyLib 库提供了 1 个 ARRAY 变量 aMyArray，其大小由全局常量 nMaxArraySize 定义。该库已集成到各种 PLC 项目中。PLC 项目使用不同的数组大小，库中的全局常量应由项目特定值覆盖。

在创建 MyLib 库时，在参数列表中可以定义全局常量 nMaxArraySize。首先，通过上下文菜单中的 **Add**（添加）命令将名称为 MyParameterList 的 **Parameter List**（参数列表）对象添加到 PLC 项目中。在编辑器中，为对象声明变量 nMaxArraySize。



库 MyLib 已集成到 PLC 项目中。打开库管理器，将全局常量的值替换为项目特定值。在上部选择库。在下部会显示带参数列表 MyParameterList 的功能块树。参数列表已被选中。在右下角，**Library Parameters**（库参数）选项卡会打开，其中包含参数列表中的声明。在 **Value (editable)**（值（可编辑））列中，要编辑的全局常量 nMaxArraySize 的值已被选中。按空格键可以打开 1 个输入字段，您可以为 nArraySize 输入所需的新值。在关闭输入字段时，该值将应用于库的局部范围。



导出和导入参数

TwinCAT 3.1 Build 4026 及以上可用

导出参数：

✓ 在库管理中，1 个带有参数列表的库已被选中。

1. 在库管理中，选择参数列表。
2. 从上下文菜单中，选择命令 **Export Library Parameters…**（导出库参数……）
3. 选择 1 个文件夹，输入文件名并点击 **Save**（保存）。

⇨ 参数列表中的编辑值已保存在 csv 文件中。

导入参数：

✓ 在库管理中，1 个带有参数列表的库已被选中。

1. 在库管理中，选择参数列表。
2. 从上下文菜单中，选择命令 **Import Library Parameters…**（导入库参数……）
3. 选择 1 个带有已保存参数的 csv 文件，并点击 **Open**（打开）。

⇨ 在 csv 文件中存储的值会被插入参数列表。

7.3 创建用户特定数据类型

除标准数据类型外，您还可以定义自己的数据类型，例如，结构、枚举、引用和联合。这些数据类型会被创建为数据类型对象（DUT = 数据单元类型）。

7.3.1 对象 DUT

符号：

- 表示不带文本列表支持的 DUT
- 表示带有文本列表支持的枚举类型的 DUT

DUT（数据单元类型）描述的是 1 种用户特定数据类型。

创建对象 DUT

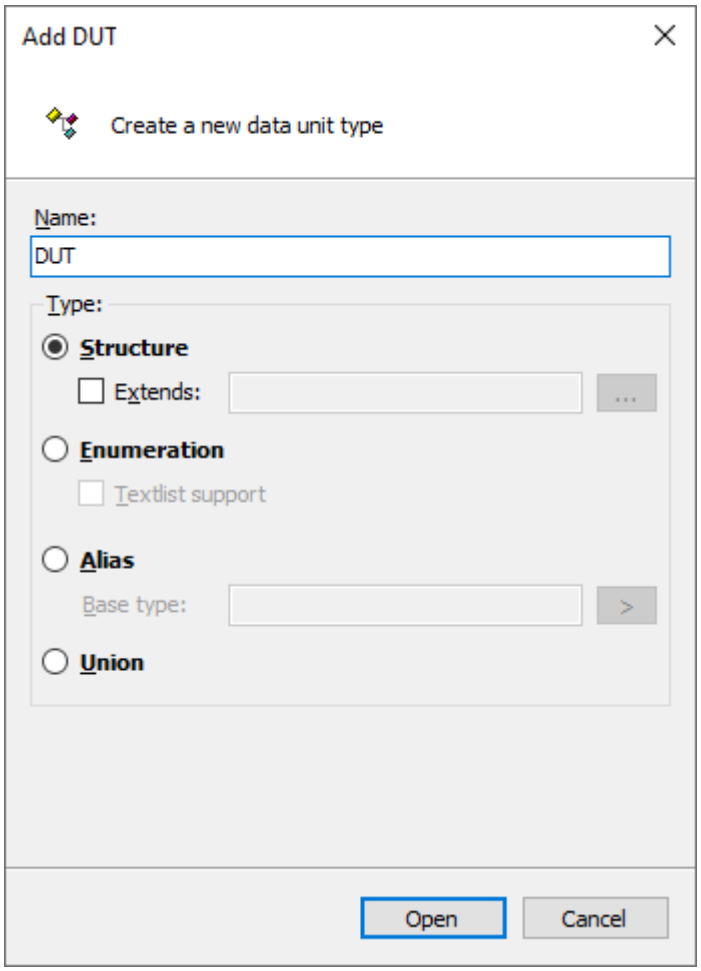
1. 在 PLC 项目树的 **Solution Explorer**（解决方案资源管理器）中，选择 1 个文件夹。
2. 在上下文菜单中，选择命令 **Add > DUT...**（添加 > DUT……）

⇨ 对话框 **Add DUT**（添加 DUT）会打开。

3. 输入名称并选择数据类型。
4. 点击 **Open**（打开）。

⇨ DUT 被添加到 PLC 项目树中，并在编辑器中打开。

对话框添加 DUT



名称	新 DUT 的名称

数据类型

结构	创建 1 个声明结构的对象，该结构将多个不同数据类型的变量组合成 1 个逻辑单元。 在结构中声明的变量被称为组件。 ☑ **高级**：该结构用附加组件扩展了现有结构。在旁边的输入字段中指定 1 个现有结构。现有结构的组件会自动出现在新结构中。 （另请参见：结构 [▶ 719]）
枚举	创建 1 个声明枚举的对象，该枚举将多个整数常量组合成 1 个逻辑单元 在枚举中声明的常量也被称为枚举值。 ☐ **文本列表支持**：1 个不带文本列表支持的枚举已创建。DUT 对象在 Solution Explorer（解决方案资源管理器）的 PLC 项目树中出现，符号如下：  ☑ **文本列表支持**：文本列表允许您查找枚举值的名称。DUT 对象在 Solution Explorer（解决方案资源管理器）的 PLC 项目树中出现，符号如下：  例如，您可以在可视化中输出本地化文本。然后，在可视化元素的文本输出中，符号枚举值会以当前语言显示，而不是数字枚举值。在可视化元素的 Text variable（文本变量）属性中输入支持文本列表的枚举变量时，会添加补充 <enumeration name>。 编辑器右边的按钮可用于在文本视图（ ）和本地化视图（文本列表）（ ）之间切换。 示例：使用 E_myEnum 类型的变量 MAIN.eVar。E_myEnum 是 1 个支持文本列表的 DUT。然后，属性编辑器中的条目如下所示：MAIN.eVar <E_myEnum>。如果在 PLC 项目中更改枚举名称，则会出现提示，询问 TwinCAT 是否应该相应更新受影响的可视化。 在现有枚举对象上，您之后随时可以添加或删除文本列表支持：为此，您可以使用对象的上下文菜单中的命令 **Add text list support**（添加文本列表支持）或 **Remove text list support**（删除文本列表支持）。 （另请参见：枚举 [▶ 721]）
别名	创建 1 个声明别名的对象，该别名可用于声明基本类型、数据类型或功能块的替代名称。 您可以直接输入基本类型，也可以通过输入助手或数组向导进行选择。 （另请参见：别名 [▶ 724]）
联合	创建 1 个声明联合的对象，该联合将多个组件（通常是不同数据类型的组件）组合成单个逻辑单元。 所有组件都有相同的偏移量，因此它们占据相同的存储空间。联合的内存占用取决于其“最大”组件的内存占用。 （另请参见：UNION [▶ 725]）

声明 DUT

语法：

```
TYPE <identifier> : <data type declaration with optional initialization>
END_TYPE
```

数据类型声明的语法具体取决于所选的数据类型（例如，结构或枚举）。

示例：

结构的声明

以下部分将展示 2 个 DUT，它们定义了结构 ST_Struct1 和 ST_Struct2。结构 ST_Struct2 扩展了结构 ST_Struct1，这意味着 ST_Struct2.nVar1 可用于访问变量 nVar1。

```
TYPE ST_Struct1 :
STRUCT
    nVar1 : INT;
    bVar2 : BOOL;
END_STRUCT
END_TYPE
```

```
TYPE ST_Struct2 EXTENDS ST_Struct1 :
STRUCT
    nVar3 : DWORD;
    sVar4 : STRING;
END_STRUCT
END_TYPE
```

枚举的声明

```
TYPE E_TrafficSignal :
(
    eRed,
    eYellow,
    eGreen := 10
);
END_TYPE
```

别名的声明

```
TYPE T_Message : STRING[50];
END_TYPE
```

不同数据类型组件的联合的声明

```
TYPE U_Name :
UNION
    fA : LREAL;
    nB : LINT;
    nC : WORD;
END_ UNION
END_TYPE
```

7.4 创建编程对象

您可以向您的 PLC 项目添加各种编程对象，您可以在其中编写和构建控制程序的源代码。

7.4.1 对象 POU

符号： 

根据 IEC 61131-3 标准的定义，**POU** 类型的对象是 TwinCAT PLC 项目中的 1 个程序组织单元。

您可以向 PLC 项目添加以下 POU 类型：

- 函数 [▶ 73]
- 功能块 [▶ 76]
- 程序 [▶ 77]

此外，您还可以为这些对象添加以下编程对象：

- 动作 [▶ 78]
- 转换 [▶ 79]
- 方法 [▶ 82]
- 属性 [▶ 88]

某些 POU 可以调用其他 POU。不允许递归。

通过命名空间调用 POU 时，TwinCAT 会按照以下顺序在项目中搜索调用的 POU：

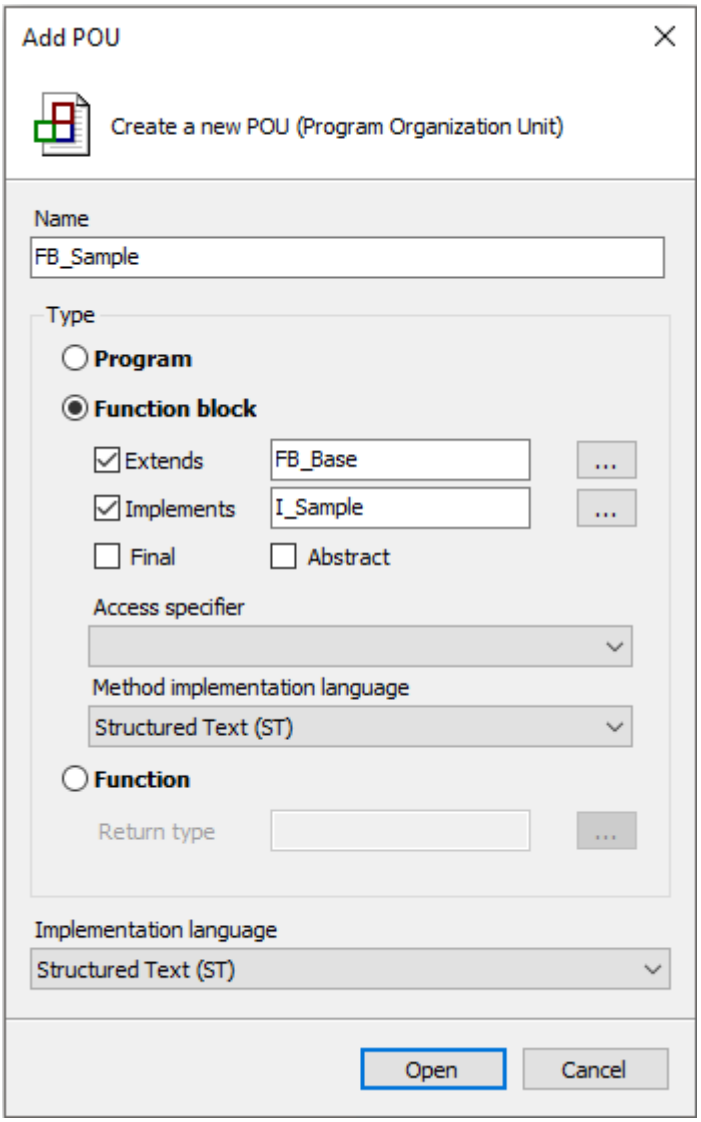
1. 当前应用程序
2. 当前应用程序的库管理器

创建对象 POU

1. 在 PLC 项目树的 **Solution Explorer**（解决方案资源管理器）中，选择 1 个文件夹。
2. 在上下文菜单中，选择命令 **Add** > **POU...**（添加 > POU……）
   ⇨ **Add POU**（添加 POU）对话框会打开。
3. 输入名称并选择类型和实现语言。
4. 点击 **Open**（打开）。

⇨ POU 被添加到 PLC 项目树中，并在编辑器中打开。编辑器由顶部的声明编辑器和底部的实现部分组成。根据实现语言的不同，**Toolbox**（工具箱）视图可能会自动进入活动状态，其中提供了合适的元素、运算符和功能块。

对话框添加 POU



名称	POU 的名称
实现语言	POU 的实现语言的选择列表

类型

程序	
功能块	• ☑ **高级**：输入或选择 1 个面向对象编程的预定义功能块。在功能块声明中，这可以用关键字 EXTENDS 进行指定。 • ☑ **已实现**：输入或选择 1 个面向对象编程的接口。在功能块声明中，这可以用关键字 IMPLEMENTS 进行指定。在创建 POU 时，将会创建通过接口定义的所有方法。 另请参见功能块中的接口元素的自动创建 [▶ 76] • ☑ **最终**：不允许派生访问。这意味着您不能用另一个功能块扩展该功能块。这让您能够检查是否应该允许进一步派生。这样可以优化代码生成。 • ☑ **抽象**：表示功能块的实现缺失或不完整，无法实例化。抽象功能块只作为基本功能块，通常在派生功能块中实现。如果您已创建 1 个非抽象功能块，而该功能块又扩展了 1 个抽象功能块，则抽象基本功能块的所有方法都会作为（非抽象）方法添加到新功能块中。 另请参见 ABSTRACT 概念 [▶ 185] • **访问修饰符** ◦ 公开：对应于没有访问修饰符的规范 ◦ 内部：对功能块的访问仅限于命名空间（库）。 • **方法实现语言**：如果您已选择选项 **Implements**（实现），您可以使用该选项为所有方法对象选择 1 种实现语言，TwinCAT 会通过实现接口生成这些方法对象。 方法实现语言与功能块的实现语言无关。
函数	如果在实现语言的选择列表中选择顺序功能图（SFC）语言，则不可用。 **返回类型**：返回值的数据类型的选择列表

7.4.1.1 对象函数

函数是 1 种 POU，在执行时会精确返回 1 个数据元素，其调用可作为表达式中的运算符出现在基于文本的语言中。数据元素也可以是数组或结构。

在 PLC 项目树中，函数 POU 具有后缀（FUN）。函数的编辑器由声明部分和实现部分组成。

函数的所有数据都是临时的，只有在执行函数时才有效（堆栈变量）。这意味着在每次调用函数时，TwinCAT 都会重新初始化您在函数中声明的所有变量。

用相同的输入变量值调用函数，总是会返回相同的输出值。因此，函数不可以使用全局变量和地址！

声明部分的第 1 行包含以下声明：

```
FUNCTION <function> : <data type>
```

输入变量和函数变量声明如下。

函数的输出变量是函数名。

如果您在函数中将局部变量声明为 RETAIN，则无效。在这种情况下，TwinCAT 会提示编译器出错。

在 TwinCAT 3 中，您不能在函数调用中混合显式参数赋值和隐式参数赋值。这意味着您应该在函数调用中仅使用显式参数赋值或仅使用隐式参数赋值。函数调用中的参数赋值的顺序无关紧要。

调用函数

在 ST 中，您可以在表达式中将函数调用用作操作数。

在 SFC 中，您只能在步骤动作或转换中使用函数调用。

示例：

带有声明部分和 1 行实现代码的函数：

F_Sample

```
1  FUNCTION F_Sample : INT
2  VAR_INPUT
3      nVar1 : INT;
4      nVar2 : INT;
5      nVar3 : INT;
6  END_VAR
7  VAR
8  END_VAR
```

```
1  F_Sample := nVar1 + nVar2 * nVar3;
```

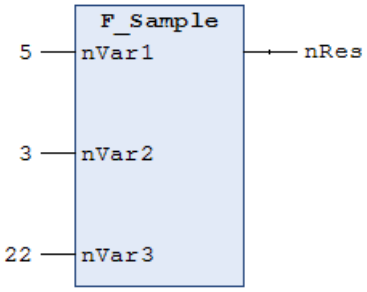
ST：

```
nRes := F_Sample(5,3,22)
```

IL：

```
1   LD          5
    F_Sample    3
                22
    ST          nRes
```

FBD：



带有附加输出的函数

根据 IEC 61131-3 标准，函数可以有附加输出。您可以在关键字 VAR_OUTPUT 和 END_VAR 之间声明函数中的附加输出。根据以下语法进行函数调用：

<function> (<function output variable1> => <output variable 1>, <function output variable n> => <output variable n>)

示例：

使用 2 个输入变量 nIn1 和 nIn2 对函数 F_Fun 进行定义。函数 F_Fun 的输出变量被写入局部声明的输出变量 nLoc1 和 nLoc2 中。

```
F_Fun(nIn1 := 1, nIn2 := 2, nOut1 => nLoc1, nOut2 => nLoc2);
```

在方法/函数/属性调用期间，访问结构化返回类型的单个元素

在调用方法、函数或属性时，可使用以下实现直接访问方法/函数/属性返回的结构化数据类型的单个元素。例如，结构化数据类型是 1 个结构或 1 个功能块。

1. 方法/函数/属性的返回类型被定义为“REFERENCE TO <structured type>”（而不只是“<structured type>”）。
2. 请注意，对于这种返回类型 - 例如，如果要返回结构化数据类型的 FB-local实例 - 则必须使用引用运算符 REF=，而不是“正常”赋值运算符 :=。

本节中的声明和示例涉及属性的调用。不过，它们同样适用于其他提供返回值的调用（例如，方法或函数）。

示例

结构 ST_Sample（结构化数据类型）的声明：

```
TYPE ST_Sample :
STRUCT
    bVar  : BOOL;
    nVar  : INT;
END_STRUCT
END_TYPE
```

功能块 FB_Sample 的声明：

```
FUNCTION_BLOCK FB_Sample
VAR
    stLocal     : ST_Sample;
END_VAR
```

返回类型为“REFERENCE TO ST_Sample”的属性 FB_Sample.MyProp 的声明：

```
PROPERTY MyProp : REFERENCE TO ST_Sample
```

属性 FB_Sample.MyProp 的 Get 方法的实现：

```
MyProp REF= stLocal;
```

属性 FB_Sample.MyProp 的 Set 方法的实现：

```
stLocal := MyProp;
```

在主程序 MAIN 中，调用 Get 和 Set 方法：

```
PROGRAM MAIN
VAR
    fbSample    : FB_Sample;
    nSingleGet  : INT;
    stGet       : ST_Sample;
    bSet        : BOOL;
    stSet       : ST_Sample;
END_VAR
```

```
// Get - single member and complete structure possible
nSingleGet := fbSample.MyProp.nVar;
stGet      := fbSample.MyProp;

// Set - only complete structure possible
IF bSet THEN
    fbSample.MyProp REF= stSet;
    bSet            := FALSE;
END_IF
```

通过将属性 MyProp 的返回类型声明为“REFERENCE TO ST_Sample”，并在该属性的 Get 方法中使用引用运算符 REF=，在调用该属性时可以直接访问返回结构化数据类型的单个元素。

```
VAR
    fbSample    : FB_Sample;
    nSingleGet  : INT;
END_VAR
```

```
nSingleGet := fbSample.MyProp.nVar;
```

如果仅将返回类型声明为“ST_Sample”，则首先必须将属性返回的结构分配给局部结构实例。然后，根据局部结构实例可以查询单个结构元素。

```
VAR
    fbSample    : FB_Sample;
    stGet       : ST_Sample;
    nSingleGet  : INT;
END_VAR
```

```
stGet      := fbSample.MyProp;
nSingleGet := stGet.nVar;
```

7.4.1.2 对象功能块

功能块是 1 个 POU [▶ 71]，它在执行时会返回 1 个或多个值。在下一次执行之前，输出变量和内部变量的值将会保留。这意味着，如果使用相同的输入变量重复调用该功能块，它可能不会返回相同的输出值。

在 PLC 项目树中，功能块 POU 具有后缀（FB）。功能块的编辑器由声明部分和实现部分组成。

功能块总是通过实例调用，实例是功能块的副本。

除了在 IEC 61131-3 中描述的功能之外，在 TwinCAT 中，功能块还可用于以下面向对象的编程功能：

- 扩展功能块（扩展功能块 [▶ 177]）
- 实现接口（实现接口 [▶ 183]）
- 方法（对象方法 [▶ 82]）
- 属性（对象属性 [▶ 88]）

声明部分的第 1 行包含以下声明：

```
FUNCTION_BLOCK <access specifier> <function block> | EXTENDS <function block> |
  IMPLEMENTS <comma-separated list of interfaces>
```

8 字节对齐

TwinCAT 3 引入了 8 字节对齐方式。如果数据作为整个内存块与其他控制器或软件组件进行交换，请确保对齐方式适当（请参见 对齐方式 [▶ 731]）。

在功能块中自动创建接口元素

有 2 种方法可以在此功能块中自动生成实现功能块的接口元素。

1. 如果您在创建新功能块时在对话框 **Add**（添加）中的字段 **Implements**（实现）中指定了 1 个接口，则 TwinCAT 也会自动将该接口的方法和属性添加到该功能块中。
2. 如果现有功能块实现 1 个接口，则您可以使用命令 **Implement Interface**（实现接口），以便在功能块中生成接口元素。在项目树中的功能块的上下文菜单中可以找到命令 **Implement Interface**（实现接口）。

另请参见：

- TC3 用户界面文档：命令：实现界面 [▶ 987]
- 接口的实现 [▶ 183]

调用功能块

调用总是通过功能块的实例进行。如果调用 1 个功能块，则只有相应实例的值会发生变化。

实例的声明：

```
<instance> : <function block>;
```

在实现部分中，功能块的变量的访问方式如下：

```
<instance>.<variable>
```

- 从功能块实例的外部，您只能访问功能块的输入和输出变量，而不能访问内部变量。
- 除非您已在全局范围内声明实例，否则对功能块实例的访问仅限于声明实例的 POU。
- 您可以在调用实例时为功能块变量分配所需的值。

示例：

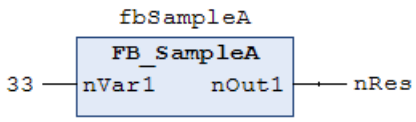
功能块 FB_SampleA 有 INT 类型的输入变量 nVar1 和输出变量 nOut1。在下面的示例中，从 MAIN 程序可以调用变量 nVar1。

ST：

```
PROGRAM MAIN
VAR
    fbSampleA : FB_SampleA;
END_VAR
```

```
fbSampleA.nVar1 := 33; (* FB_SampleA is called and the value 33 is assigned to the variable nVar1 *)
fbSampleA(); (* FB_SampleA is called, that's necessary for the following access to the output variab
le *)
nRes := fbSampleA.nOut1 (* the output variable nOut1 of the FB1 is read *)
```

FBD:



在调用过程中分配变量值:

在基于文本的语言 IL 和 ST 中，您在调用功能块时可以直接为输入变量和/或输出变量赋值。

使用 := 为输入变量赋值

使用 => 为输出变量赋值

示例:

在调用计时器功能块的实例 fbTimer 时，需要对输入变量 IN 和 PT 进行赋值。然后，将计时器的输出变量 Q 分配给变量 bVarA

```
PROGRAM MAIN
VAR
    fbTimer : TOF;
    bIn     : BOOL;
    bVarA   : BOOL;
END_VAR
```

```
fbTimer(IN := bIn, PT := t#300ms);
bVarA := fbTimer.Q;
```

如果您通过输入助手添加功能块实例，并且在 **Input Assistant**（输入助手）对话框中已启用选项 **Insert with arguments**（带参数插入），则 TwinCAT 会添加带有所有输入和输出变量的调用。您只需添加所需的赋值。在上述示例中，TwinCAT 会添加如下调用：CMD_TMR (IN:= , PT:= , Q=>)。

使用局部变量上的属性 **“is_connected”**，您可以在功能块实例中调用时确定特定输入是否会从外部接收赋值。

7.4.1.3 对象程序

程序是 1 个 POU，它在执行时会返回 1 个或多个值。在程序执行之后，所有值将会在再次执行程序之前保留。在任务对象中可以定义 PLC 项目中的程序的调用顺序。

在 PLC 项目树中，程序 POU 具有后缀（PRG）。程序的编辑器由声明部分和实现部分组成。

声明部分的第 1 行包含以下声明:

```
PROGRAM <program>
```

调用程序

程序和功能块可以调用程序。在函数中不允许进行程序调用。程序没有实例。

如果 POU 调用 1 个程序，而调用会导致程序的值发生变化，那么，这些变化将会在下一次程序调用之前保留。如果下一次调用来自另一个 POU，则程序的值也会被保留。这与功能块调用不同。当调用 1 个功能块时，只有相应功能块实例的值会发生变化。只有当 POU 再次调用同一个实例时，才会发生相关的变化。

或者，也可以直接通过调用来设置程序的输入和/或输出参数。

语法:

<program>(<input variable> := <value>, <output value> => <value>):

如果您通过输入助手添加程序调用，并且在 **Input assistant**（输入助手）中已启用选项 **Insert with arguments**（带参数插入），则 TwinCAT 会根据相应的语法将输入和/或输出参数添加到程序调用中。

示例：

IL：

```
1  CAL          SampleProg(
          nIn1:= 2)
   LD           SampleProg.nOut2
   ST           nRes
```

带参数分配：

```
1  CAL          SampleProg(
          nIn1:= 2
         nOut2=> nRes)
```

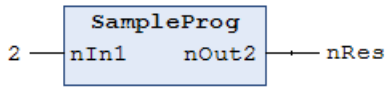
ST：

```
SampleProg();
nRes := SampleProg.nOut2;
```

带参数分配：

```
SampleProg(nIn1 := 2, nOut2 => nRes);
```

FBD：



7.4.2 对象动作

符号： 

动作可用于执行更多程序代码。您可以用不同于基本实现的语言来实现该程序代码。基本实现是您插入动作的功能块或程序。

动作没有自己的声明，而是使用基本实现的数据。这意味着动作使用基本实现的输入/输出和局部变量。

创建对象动作

1. 在 PLC 项目树的 **Solution Explorer**（解决方案资源管理器）中，选择功能块或程序。
2. 在上下文菜单中，选择命令 **Add** > **Action…**（添加 > 动作…）
   ⇨ 对话框 **Add Action**（添加动作）会打开。
3. 输入名称并选择实现语言。
4. 点击 **Open**（打开）。

⇨ 对象被添加到 PLC 项目树中，并在编辑器中打开。

对话框添加动作

名称	动作的名称
实现语言	实现语言的复选框

调用动作

语法：

<program>.<action> 或 <FB-instance>.<action>

如要仅在基本实现中调用 1 个动作，只需指定动作名称即可。

示例：

从另一个 POU 调用重置动作。换句话说，该调用并非在基本实现中进行。

声明：

```
PROGRAM MAIN
VAR
    fbCounterA : FB_Counter;
END_VAR
```

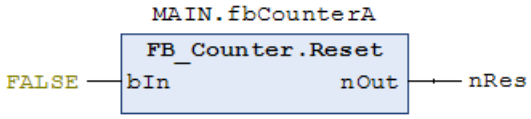
从 IL POU 调用重置动作：

```
1   CAL         fbCounterA.Reset(
            bIn:= FALSE)
    LD          fbCounterA.nOut
    ST          nRes
```

从 ST POU 调用重置动作：

```
fbCounterA.Reset(In := FALSE);
nRes := fbCounterA.nOut;
```

从 FBD POU 调用重置动作：



动作在实现语言 SFC 中很常见。（SFC 元素动作 [▶ 588]）

另请参见：

- 面向对象的编程 [▶ 158]
- 扩展功能块 [▶ 177]

7.4.3 对象转换

符号：

转换用于指定 1 个条件，在此条件下，后续步骤将进入活动状态。转换条件可以有值 TRUE 或 FALSE。如果为 TRUE，则执行下一步。通过以下 2 种方式可以指定转换条件：

- (1) 直接（“内联条件”）：用 Boolean 变量名、Boolean 地址、Boolean 常量或带 Boolean 结果的语句（例如，(i<100) AND b）替换标准转换名称。不得指定任何程序、功能块或赋值。
- (2) 使用单独的转换或属性对象（“多用途条件”）：用转换或属性对象的名称替换标准转换名称。这些对象可以通过上下文菜单中的命令 **Add > Transition…**（添加 > 转换…）创建（请参见“创建转换的对象 [▶ 79]“部分）。这样可以多次使用转换。与”内联条件“一样，该对象可以包含 1 个 Boolean 变量、1 个地址、1 个常量或 1 条语句，此外，它还可以包含多条带有任何代码的语句。

请按照”访问功能块的 VAR_IN_OUT 变量 [▶ 81]“部分中的说明操作。

创建对象转换

1. 在 PLC 项目树的 **Solution Explorer**（解决方案资源管理器）中，选择功能块或程序。
2. 在上下文菜单中，选择命令 **Add > Transition…**（添加 > 转换…）
   ⇨ **Add Transition**（添加转换）对话框会打开。
3. 输入名称并选择实现语言。

4. 点击 **Open**（打开）。

⇨ 对象被添加到 PLC 项目树中，并在编辑器中打开。

对话框添加转换

名称	转换的名称
实现语言	实现语言的复选框

调用转换

语法：

与 TwinCAT 2 PLC 控制相比，转换条件被视为方法调用。根据以下语法进行输入：

<transition name> := <transition condition>

或

<transition condition>

如果转换包含多个语句，您必须将所需的表达式分配给转换变量（语法的第 1 个任意变体）。

示例：

在 ST POU 中调用转换 Trans1：

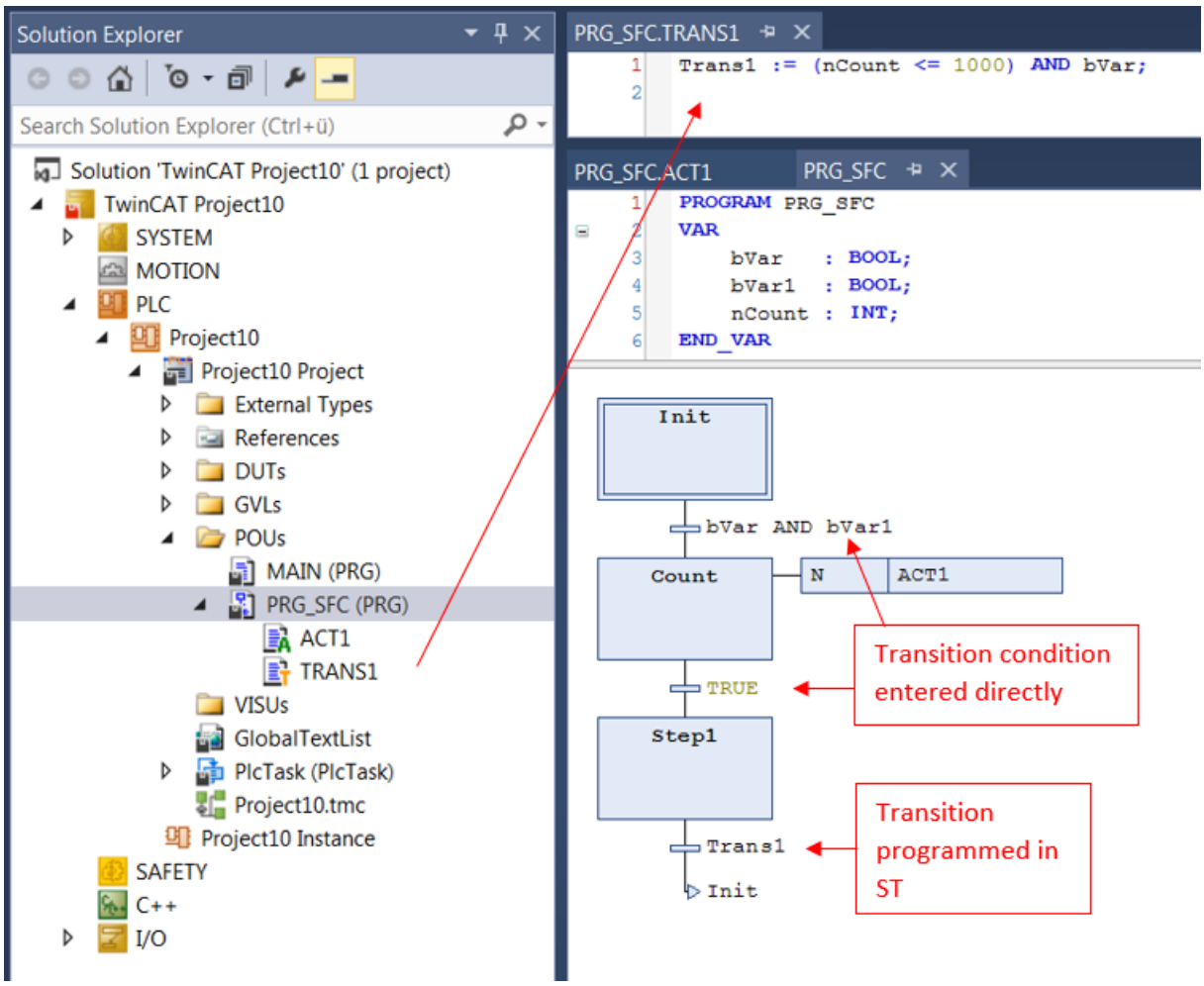
PRG_SFC.Trans1：

```
Trans1 := (nCount<=100);
```

PRG_SFC：

```
nCount := nCount+1;
IF Trans1 = TRUE THEN // IF nCount<=100 THEN …
    nVar:=1;
    ELSE
        nVar:=2;
END_IF
```

在 SFC POU 中调用转换 Trans1：



在方法/转换/属性中访问功能块的 VAR_IN_OUT 变量

原则上，在功能块的方法、转换或属性中可以访问功能块的 VAR_IN_OUT 变量。对于这种类型的访问，请注意以下几点：

- 如果从 FB 外部调用功能块的主体或某个动作，编译器应确保功能块的 VAR_IN_OUT 变量与该调用一起分配。
- 如果调用功能块的方法、转换或属性，则不是这种情况，因为 FB 的 VAR_IN_OUT 变量不能在方法、转换或属性调用中分配。因此，在 VAR_IN_OUT 变量被分配给有效引用之前，通过调用方法/转换/属性可能会导致对 VAR_IN_OUT 变量的访问。由于这将意味着运行时的无效访问，因此，在方法、转换或属性中访问 FB 的 VAR_IN_OUT 变量具有潜在风险。

因此，如果在方法、转换或属性中访问 FB 的 VAR_IN_OUT 变量，就会发出带有 ID C0371 的以下警告：

“警告：从外部环境 <Method/Transition/Property> 访问在 <POU> 中声明的 VAR_IN_OUT <Var>”

对此项警告的适当回应可以是，在访问之前，检查方法/转换/属性中的 VAR_IN_OUT 变量。运算符 __ISVALIDREF 可用于此项检查，以确定引用是否指向 1 个有效值。如果此项检查已启用，则可以认为用户已经意识到在方法/转换/属性中访问 FB 的 VAR_IN_OUT 变量时可能存在的风险。检查引用被认为是对这一风险的适当处理。因此，通过属性“禁用警告”可以抑制相应的警告。

方法实现示例如下所示。

功能块 FB_Sample：

```
FUNCTION_BLOCK FB_Sample
VAR_IN_OUT
    bInOut : BOOL;
END_Var
```

方法 FB_Sample.MyMethod：

```
METHOD MyMethod
VAR_INPUT
END_VAR
```

```
// The warning can be disabled here as the user is aware of the risk that the reference may not be
valid by checking its validity
{warning disable C0371}

// Checking the VAR_IN_OUT reference, leave the current method in case of invalid reference
IF NOT __ISVALIDREF(bInOut) THEN
     RETURN;
END_IF

// Access to VAR_IN_OUT reference (only if the reference was confirmed as valid before)
bInOut := NOT bInOut;

// The warning may be restored at the end of the access area
{warning restore C0371}
```

另请参见：

- 面向对象的编程 [▶ 158]
- 扩展功能块 [▶ 177]
- 引用编程：SFC 元素步骤和转换 [▶ 588]

7.4.4 对象方法

符号： 

方法是对 IEC 61131-3 标准的扩展，也是面向对象的编程中用于数据封装的一种手段。1 个方法包含 1 个声明和 1 个实现。但是，与函数不同的是，方法不是一个独立的程序块，而是从属于某一功能块或程序。方法可以访问上级编程块的所有有效变量。

可以在程序或功能块下面添加方法。请使用“**Project > Add object > Method**”（项目 > 添加对象 > 方法）命令打开“**Add method**”（添加方法）对话框。

您可以使用接口来组织方法。有关更多信息，请参见：接口的实现 [▶ 183]

如果将编程块下面的方法复制并粘贴到某一接口下面，或将该方法移动至该接口下面，那么这个方法中的实现内容会自动被移除。

方法说明

- 方法的所有数据均为临时数据，仅在执行方法时有效（堆栈变量）。这意味着每次调用方法时，TwinCAT 都会重新初始化您在该方法中声明的所有变量和功能块。
- 与函数一样，方法也可以返回 1 个返回值。
- 根据 IEC 61131-3 标准，方法可以像正常函数一样有附加输入和输出。您可以在调用方法时分配输入和输出。
  - **TwinCAT 3.1.4026 版本开始：**没有明确指定初始值的输入在调用方法时必须赋值。有明确初始值的输入在调用方法时可以选择赋值或忽略。
- 在方法的实现部分，允许访问功能块实例或程序变量。
- 使用 THIS 指针指向自己的实例。
- 无法访问方法中功能块的 VAR_TEMP 变量。
- 您可以声明 VAR_INST 变量，当再次调用方法时这些变量不会重新初始化（另请参见“实例变量”一章）。
- 通过使用返回类型“REFERENCE TO <structured type>”，您可以在调用方法时直接访问该方法返回的结构体数据类型的单个元素。有关更多信息，请参见“在方法/函数/属性调用期间，访问结构化返回类型的单个元素 [▶ 86]”部分。
- 原则上，在方法中可以访问功能块的 VAR_IN_OUT 变量。由于此类访问存在潜在风险，应谨慎使用。有关更多信息，请参见“在方法/转换中访问功能块的 VAR_IN_OUT 变量 [▶ 87]”部分。
- 在接口中定义的方法只能声明输入、输出和 VAR_IN_OUT 变量，但不得包含具体的实现代码。

示例：

以下示例中的代码会使 TwinCAT 将方法的返回值和输出写入局部声明的变量中。

功能块 FB_Sample 的方法“Method1”的声明部分：

```
METHOD Method1 : BOOL
VAR_INPUT
    nIn1  : INT;
    bIn2  : BOOL;
END_VAR
VAR_OUTPUT
    fOut1 : REAL;
    sOut2 : STRING;
END_VAR
```

```
// <method implementation code>
```

MAIN 程序：

```
PROGRAM MAIN
VAR
    fbSample      : FB_Sample;
    bReturnValue  : BOOL;
    nLocalInput1  : INT;
    bLocalInput2  : BOOL;
    fLocalOutput1 : REAL;
    sLocalOutput2 : STRING;
END_VAR
```

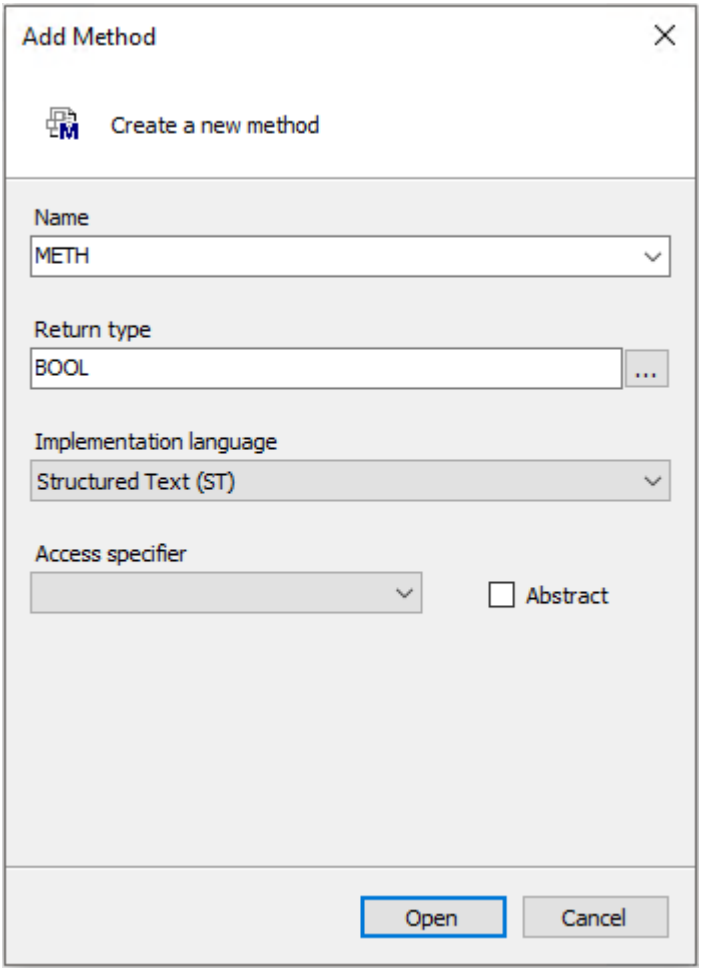
```
bReturnValue := fbSample.Method1(nIn1  := nLocalInput1,
                                 bIn2  := bLocalInput2,
                                 fOut1 => fLocalOutput1,
                                 sOut2 => sLocalOutput2);
```

创建对象方法

1. 在 PLC 项目树的 **Solution Explorer**（解决方案资源管理器）中，选择功能块或程序。
2. 在上下文菜单中，选择命令 **Add** > **Method...**（添加 > 方法……）
   ⇨ 对话框 **Add Method**（添加方法）会打开。
3. 输入名称并选择返回类型、实现语言，还可选择访问修饰符
4. 点击 **Open**（打开）。

⇨ 对象被添加到 PLC 项目树中，并在编辑器中打开。编辑器由顶部的声明编辑器和底部的实现部分组成。

对话框添加方法



名称	方法的名称 如果在功能块下方尚未插入标准方法 FB_Init 和 FB_Exit，则在选择列表中会提供这 2 种方法。如果是派生功能块，选择列表还会提供基本功能块的所有方法。
返回类型	返回值的类型
实现语言	实现语言选择列表

访问修饰符

访问修饰符	规范数据访问 • **公开**：访问不受限制（相当于没有指定访问修饰符）。 • **私有**：对方法的访问分别限制在功能块或程序中。 • **受保护**：对方法的访问分别限制在程序或功能块及其导数中。 • **内部**：对方法的访问限制在命名空间（库）中。 除了这些访问修饰符外，您还可以为方法手动添加 FINAL 修饰符： • **最终**：不允许覆盖功能块导数中的方法。这意味着在可能存在的子类中不能覆盖/扩展该方法。
抽象	☑：表示方法没有实现，实现由派生 FB 提供。 有关 ABSTRACT 关键字的背景信息，请参见 ABSTRACT 概念 [▶ 185] 下方内容。

在 PLC 项目树的解决方案资源管理器中，访问修饰符与 PUBLIC 不同的方法会用信号符号标记。

访问修饰符	对象图标	信号符号
私有		 （锁）
受保护		 （星）
内部		 （心）

如果您将 POU 中的方法复制或移动到接口中，TwinCAT 会自动删除所包含的实现。

功能块的特殊方法

FB_init	默认为隐式声明。也可以提供显式声明。 包含功能块的初始化代码，如在功能块的声明部分所定义。 （方法 FB_init、FB_reinit 和 FB_exit [▶ 786]）
FB_reinit	需要显式声明。当功能块实例被复制时（例如，在在线更改期间）进行调用。它会重新初始化新的实例模块。 （方法 FB_init、FB_reinit 和 FB_exit [▶ 786]）
FB_exit	需要显式声明。 在再次下载、重置之前或者在线更改所有移动或删除的实例期间，调用功能块的每个实例。 （方法 FB_init、FB_reinit 和 FB_exit [▶ 786]）
属性和接口属性	它们都包含 1 个 Set 和/或 1 个 Get 访问器方法。 （对象接口属性 [▶ 98]，对象属性 [▶ 88]）

调用方法

语法：

```
<return value variable> := <POU name>.<method name>(<method input name> := <variable name> (,
<further method input name> := <variable name> )* );
```

当您调用方法时，您可以将传输参数分配给方法的输入变量。在此过程中，请遵守声明。只需指定输入变量的名称即可，无需考虑它们在声明中的顺序。

示例：

以下示例中的代码会使 TwinCAT 将方法的返回值和输出写入局部声明的变量中。

功能块 FB_Sample 的“Method1”的声明部分：

```
METHOD Method1 : BOOL
VAR_INPUT
    nIn1  : INT;
    bIn2  : BOOL;
END_VAR
VAR_OUTPUT
    fOut1 : REAL;
    sOut2 : STRING;
END_VAR
```

```
// <method implementation code>
```

MAIN 程序：

```
PROGRAM MAIN
VAR
    fbSample      : FB_Sample;
```

```
    bReturnValue  : BOOL;
    nLocalInput1  : INT;
    bLocalInput2  : BOOL;
    fLocalOutput1 : REAL;
    sLocalOutput2 : STRING;
END_VAR
```

```
bReturnValue := fbSample.Method1(nIn1  := nLocalInput1,
                                 bIn2  := bLocalInput2,
                                 fOut1 => fLocalOutput1,
                                 sOut2 => sLocalOutput2);
```

递归方法调用

在实现过程中，方法可以直接通过THIS指针直接调用自身，也可以通过功能块索引间接调用自身。

- 借助 THIS 指针直接调用相关功能块实例：
  <Variable für Rückgabewert> := THIS^.<Methodenname> (<Parameter>);
- 通过一个局部变量调用方法，该变量会临时实例化受影响的功能块。
  <Variable für Rückgabewert> := <FB-Instanz>.<Methodenname> (<Parameter>);

如果方法中有递归调用，会出现编译器警告。如果使用特定的编译指示 {attribute 'estimated-stack-usage' := '<estimated stack size in bytes>'} ，编译器警告将受到抑制。

实现示例请参见“属性 “estimated-stack-usage” [▶ 740]”。

若要以递归的方式调用方法，仅指定方法名称是不够的。如果仅指定了方法名称，则会输出编译器错误（需要指定“程序名称、函数或功能块实例，而不是.....”）。

在方法/函数/属性调用期间，访问结构化返回类型的单个元素

在调用方法、函数或属性时，可使用以下实现直接访问方法/函数/属性返回的结构化数据类型的单个元素。例如，结构化数据类型是 1 个结构或 1 个功能块。

1. 方法/函数/属性的返回类型被定义为“REFERENCE TO <structured type>”（而不只是“<structured type>”）。
2. 请注意，对于这种返回类型 - 例如，如果要返回结构化数据类型的 FB-local实例 - 则必须使用引用运算符 REF=，而不是“正常”赋值运算符 :=。

本节中的声明和示例涉及属性的调用。不过，它们同样适用于其他提供返回值的调用（例如，方法或函数）。

示例

结构 ST_Sample（结构化数据类型）的声明：

```
TYPE ST_Sample :
STRUCT
    bVar  : BOOL;
    nVar  : INT;
END_STRUCT
END_TYPE
```

功能块 FB_Sample 的声明：

```
FUNCTION_BLOCK FB_Sample
VAR
    stLocal     : ST_Sample;
END_VAR
```

返回类型为“REFERENCE TO ST_Sample”的属性 FB_Sample.MyProp 的声明：

```
PROPERTY MyProp : REFERENCE TO ST_Sample
```

属性 FB_Sample.MyProp 的 Get 方法的实现：

```
MyProp REF= stLocal;
```

属性 FB_Sample.MyProp 的 Set 方法的实现：

```
stLocal := MyProp;
```

在主程序 MAIN 中，调用 Get 和 Set 方法：

```
PROGRAM MAIN
VAR
    fbSample    : FB_Sample;
    nSingleGet  : INT;
    stGet       : ST_Sample;
    bSet        : BOOL;
    stSet       : ST_Sample;
END_VAR
```

```
// Get - single member and complete structure possible
nSingleGet := fbSample.MyProp.nVar;
stGet      := fbSample.MyProp;

// Set - only complete structure possible
IF bSet THEN
    fbSample.MyProp REF= stSet;
    bSet            := FALSE;
END_IF
```

通过将属性 MyProp 的返回类型声明为“REFERENCE TO ST_Sample”，并在该属性的 Get 方法中使用引用运算符 REF=，在调用该属性时可以直接访问返回结构化数据类型的单个元素。

```
VAR
    fbSample    : FB_Sample;
    nSingleGet  : INT;
END_VAR
```

```
nSingleGet := fbSample.MyProp.nVar;
```

如果仅将返回类型声明为“ST_Sample”，则首先必须将属性返回的结构分配给局部结构实例。然后，根据局部结构实例可以查询单个结构元素。

```
VAR
    fbSample    : FB_Sample;
    stGet       : ST_Sample;
    nSingleGet  : INT;
END_VAR
```

```
stGet      := fbSample.MyProp;
nSingleGet := stGet.nVar;
```

在方法/转换/属性中访问功能块的 VAR_IN_OUT 变量

原则上，在功能块的方法、转换或属性中可以访问功能块的 VAR_IN_OUT 变量。对于这种类型的访问，请注意以下几点：

- 如果从 FB 外部调用功能块的主体或某个动作，编译器应确保功能块的 VAR_IN_OUT 变量与该调用一起分配。
- 如果调用功能块的方法、转换或属性，则不是这种情况，因为 FB 的 VAR_IN_OUT 变量不能在方法、转换或属性调用中分配。因此，在 VAR_IN_OUT 变量被分配给有效引用之前，通过调用方法/转换/属性可能会导致对 VAR_IN_OUT 变量的访问。由于这将意味着运行时的无效访问，因此，在方法、转换或属性中访问 FB 的 VAR_IN_OUT 变量具有潜在风险。

因此，如果在方法、转换或属性中访问 FB 的 VAR_IN_OUT 变量，就会发出带有 ID C0371 的以下警告：

“警告：从外部环境 <Method/Transition/Property> 访问在 <POU> 中声明的 VAR_IN_OUT <Var>”

对此项警告的适当回应可以是，在访问之前，检查方法/转换/属性中的 VAR_IN_OUT 变量。运算符 __ISVALIDREF 可用于此项检查，以确定引用是否指向 1 个有效值。如果此项检查已启用，则可以认为用户已经意识到在方法/转换/属性中访问 FB 的 VAR_IN_OUT 变量时可能存在的风险。检查引用被认为是对这一风险的适当处理。因此，通过属性“禁用警告”可以抑制相应的警告。

方法实现示例如下所示。

功能块 FB_Sample：

```
FUNCTION_BLOCK FB_Sample
VAR_IN_OUT
    bInOut : BOOL;
END_Var
```

方法 FB_Sample.MyMethod：

```
METHOD MyMethod
VAR_INPUT
END_VAR
```

```
// The warning can be disabled here as the user is aware of the risk that the reference may not be
valid by checking its validity
{warning disable C0371}

// Checking the VAR_IN_OUT reference, leave the current method in case of invalid reference
IF NOT __ISVALIDREF(bInOut) THEN
     RETURN;
END_IF

// Access to VAR_IN_OUT reference (only if the reference was confirmed as valid before)
bInOut := NOT bInOut;

// The warning may be restored at the end of the access area
{warning restore C0371}
```

另请参见：

- 面向对象的编程 [▶ 158]
- 对象接口 [▶ 94]
- 接口的实现 [▶ 183]
- 扩展功能块 [▶ 177]
- 方法调用 [▶ 184]
- ABSTRACT 概念 [▶ 185]
- 引用编程 > 实例变量 - VAR INST [▶ 628]

7.4.5 对象属性

符号： 

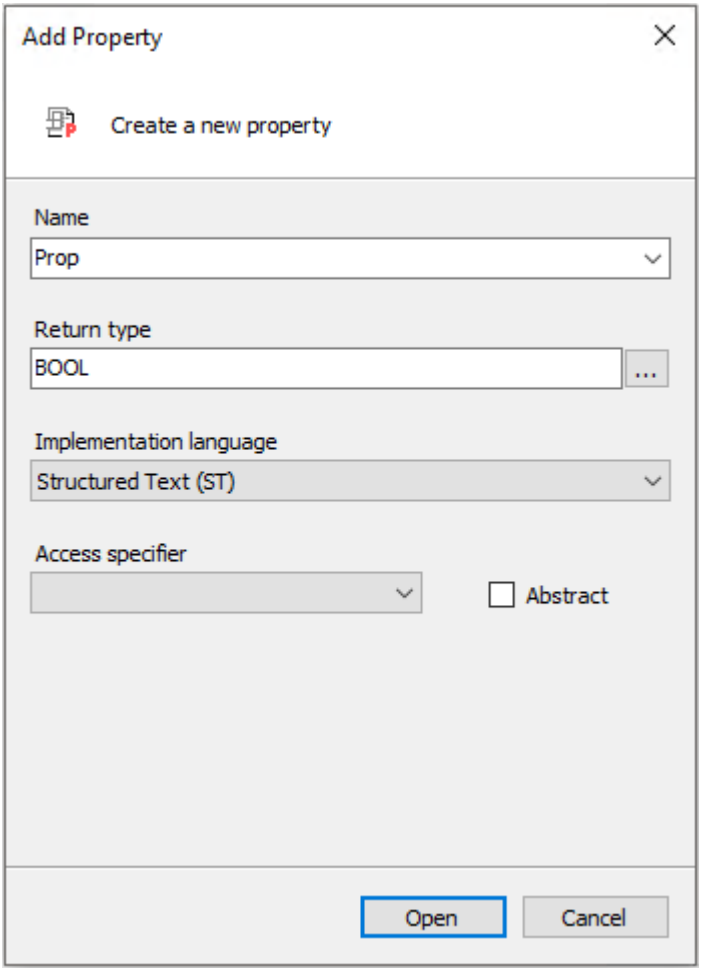
属性是 IEC 61131-3 标准的扩展，是 1 种面向对象的编程的手段。它由访问器方法 Get 和 Set 组成。当对实现该属性的功能块进行读取或写入访问时，TwinCAT 会自动调用这些方法。

创建属性的对象

1. 在 PLC 项目树的 **Solution Explorer**（解决方案资源管理器）中，选择功能块或程序。
2. 在上下文菜单中，选择命令 **Add > Property...**（添加 > 属性……）
   ⇨ **Add Property**（添加属性）对话框会打开。
3. 输入名称并选择返回类型、实现语言，还可选择访问修饰符。
4. 点击 **Open**（打开）。

⇨ 对象被添加到 PLC 项目树中，并在编辑器中打开。

对话框添加属性



名称	属性的名称
返回类型	返回值的类型（默认类型或结构化类型）
实现语言	实现语言选择列表

访问修饰符

访问说明符	规范数据访问 • **公开**：访问不受限制（相当于没有指定访问修饰符）。 • **私有**：对属性的访问分别限制在功能块或程序中。 • **受保护**：对属性的访问分别限制在程序或功能块及其导数中。 • **内部**：对属性的访问限制在命名空间（库）中。 除了这些访问修饰符外，您还可以为属性手动添加 FINAL 修饰符： **最终**：不允许覆盖功能块导数中的属性。这意味着在可能存在的子类中不能覆盖/扩展该属性。
抽象	☑：表示属性没有实现，实现由派生 FB 提供。 有关 ABSTRACT 关键字的背景信息，请参见 ABSTRACT 概念 [▶ 185] 下方内容。

在 PLC 项目树的解决方案资源管理器中，访问修饰符与 PUBLIC 不同的属性会用信号符号标记。

访问修饰符	对象图标	信号符号
私有		 （锁）
受保护		 （星）
内部		 （心）

此外，属性还可以包含局部变量。但是，属性不能包含附加输入。与函数或方法相比，它也不能包含附加输出。

如果您将 POU 中的属性复制或移动到接口中，TwinCAT 会自动删除所包含的实现。

Get 和 Set 访问器

TwinCAT 会在 PLC 项目树中的属性对象下方自动添加 Get 和 Set 访问器方法。您可以使用命令 **Add**（添加）明确添加它们。

当对属性进行写入访问时（即您将属性名称用作输入参数时），TwinCAT 会调用 Set 访问器。

当对属性进行读取访问时（即您将属性名称用作输出参数时），TwinCAT 会调用 Get 访问器。

请注意，您必须实现访问器方法才能访问该属性。

实现示例

功能块 FB_Sample 的声明

```
FUNCTION_BLOCK FB_Sample
VAR
    nVar : INT;
END_VAR
```

```
nVar := nVar + 1;
```

属性 nValue 的声明

```
PROPERTY PUBLIC nValue : INT
```

访问器方法 FB_Sample.nValue.Set 的实现

```
nVar := nValue;
```

访问器方法 FB_Sample.nValue.Get 的实现

```
nValue := nVar;
```

调用属性 nValue

```
PROGRAM MAIN
VAR
    fbSample : FB_Sample;
END_VAR
```

```
fbSample();
If fbSample.nValue > 500 THEN
    fbSample.nValue := 0;
END_IF;
```

如果您只想将属性用于读取访问或写入访问，您可以删除未使用的访问器。

您可以在以下位置为访问器方法添加访问说明符：

- 通过手动输入访问器的声明部分。
- 在 **Add 'get' accessor**（添加“get”访问器）或 **Add 'set' accessor**（添加“set”访问器）对话框中，当您使用 **Add**（添加）命令明确添加访问器时。

针对 REFERENCE 类型属性的兼容性警告

在 TC3.1 Build 4022 中，使用返回类型“REFERENCE TO <…>”定义的属性的调用行为发生了变化。

对于使用“:=”的写入访问，在版本 < 3.1.4022.0 时，会调用 set 访问器，以便写入引用。不过，在版本 >= 3.1.4022.0 时，会调用 get 访问器，以便写入值。如要在版本 >= 3.1.4022.0 时分配引用，务必使用引用赋值运算符“REF= [▶ 575]”。

在线模式下的属性监控

编译指示可用于在线模式下的属性监控，应该将其添加到定义属性的顶部（属性“monitoring” [▶ 749]）：

- {attribute 'monitoring':= 'variable'}：每次访问该属性时，TwinCAT 都会将实际值存储到 1 个变量中，并显示该变量的值。如果代码不再访问该属性，该值可能会过时。
- {attribute 'monitoring' := 'call'}：每次显示值时，TwinCAT 都会调用 Get 访问器的代码。如果该代码包含副作用，副作用将由监控执行。

您可以使用以下功能监控属性：

- 内联监控
  要求：在类别 **TwinCAT > PLC Programming Environment > Text Editor**（TwinCAT > PLC 编程环境 > 文本编辑器）中的 TwinCAT 选项中，选项卡 **Monitoring**（监控）中的选项 **Enable Inline-Monitoring**（启用内联监控）已被激活。
- 监视列表（使用监视列表 [▶ 209]）

在方法/函数/属性调用期间，访问结构化返回类型的单个元素

在调用方法、函数或属性时，可使用以下实现直接访问方法/函数/属性返回的结构化数据类型的单个元素。例如，结构化数据类型是 1 个结构或 1 个功能块。

1. 方法/函数/属性的返回类型被定义为“REFERENCE TO <structured type>”（而不只是“<structured type>”）。
2. 请注意，对于这种返回类型 - 例如，如果要返回结构化数据类型的 FB-local实例 - 则必须使用引用运算符 REF=，而不是“正常”赋值运算符 :=。

本节中的声明和示例涉及属性的调用。不过，它们同样适用于其他提供返回值的调用（例如，方法或函数）。

示例

结构 ST_Sample（结构化数据类型）的声明：

```
TYPE ST_Sample :
STRUCT
    bVar  : BOOL;
    nVar  : INT;
END_STRUCT
END_TYPE
```

功能块 FB_Sample 的声明：

```
FUNCTION_BLOCK FB_Sample
VAR
    stLocal     : ST_Sample;
END_VAR
```

返回类型为“REFERENCE TO ST_Sample”的属性 FB_Sample.MyProp 的声明：

```
PROPERTY MyProp : REFERENCE TO ST_Sample
```

属性 FB_Sample.MyProp 的 Get 方法的实现：

```
MyProp REF= stLocal;
```

属性 FB_Sample.MyProp 的 Set 方法的实现：

```
stLocal := MyProp;
```

在主程序 MAIN 中，调用 Get 和 Set 方法：

```
PROGRAM MAIN
VAR
    fbSample    : FB_Sample;
    nSingleGet  : INT;
    stGet       : ST_Sample;
    bSet        : BOOL;
    stSet       : ST_Sample;
END_VAR
```

```
// Get - single member and complete structure possible
nSingleGet := fbSample.MyProp.nVar;
stGet      := fbSample.MyProp;

// Set - only complete structure possible
IF bSet THEN
    fbSample.MyProp REF= stSet;
    bSet            := FALSE;
END_IF
```

通过将属性 MyProp 的返回类型声明为“REFERENCE TO ST_Sample”，并在该属性的 Get 方法中使用引用运算符 REF=，在调用该属性时可以直接访问返回结构化数据类型的单个元素。

```
VAR
    fbSample    : FB_Sample;
    nSingleGet  : INT;
END_VAR
```

```
nSingleGet := fbSample.MyProp.nVar;
```

如果仅将返回类型声明为“ST_Sample”，则首先必须将属性返回的结构分配给局部结构实例。然后，根据局部结构实例可以查询单个结构元素。

```
VAR
    fbSample    : FB_Sample;
    stGet       : ST_Sample;
    nSingleGet  : INT;
END_VAR
```

```
stGet      := fbSample.MyProp;
nSingleGet := stGet.nVar;
```

在方法/转换/属性中访问功能块的 VAR_IN_OUT 变量

原则上，在功能块的方法、转换或属性中可以访问功能块的 VAR_IN_OUT 变量。对于这种类型的访问，请注意以下几点：

- 如果从 FB 外部调用功能块的主体或某个动作，编译器应确保功能块的 VAR_IN_OUT 变量与该调用一起分配。
- 如果调用功能块的方法、转换或属性，则不是这种情况，因为 FB 的 VAR_IN_OUT 变量不能在方法、转换或属性调用中分配。因此，在 VAR_IN_OUT 变量被分配给有效引用之前，通过调用方法/转换/属性可能会导致对 VAR_IN_OUT 变量的访问。由于这将意味着运行时的无效访问，因此，在方法、转换或属性中访问 FB 的 VAR_IN_OUT 变量具有潜在风险。

因此，如果在方法、转换或属性中访问 FB 的 VAR_IN_OUT 变量，就会发出带有 ID C0371 的以下警告：

“警告：从外部环境 <Method/Transition/Property> 访问在 <POU> 中声明的 VAR_IN_OUT <Var>”

对此项警告的适当回应可以是，在访问之前，检查方法/转换/属性中的 VAR_IN_OUT 变量。运算符 __ISVALIDREF 可用于此项检查，以确定引用是否指向 1 个有效值。如果此项检查已启用，则可以认为用户已经意识到在方法/转换/属性中访问 FB 的 VAR_IN_OUT 变量时可能存在的风险。检查引用被认为是对这一风险的适当处理。因此，通过属性“禁用警告”可以抑制相应的警告。

方法实现示例如下所示。

功能块 FB_Sample：

```
FUNCTION_BLOCK FB_Sample
VAR_IN_OUT
    bInOut : BOOL;
END_Var
```

方法 FB_Sample.MyMethod：

```
METHOD MyMethod
VAR_INPUT
END_VAR
```

```
// The warning can be disabled here as the user is aware of the risk that the reference may not be
valid by checking its validity
{warning disable C0371}

// Checking the VAR_IN_OUT reference, leave the current method in case of invalid reference
IF NOT __ISVALIDREF(bInOut) THEN
    RETURN;
END_IF

// Access to VAR_IN_OUT reference (only if the reference was confirmed as valid before)
bInOut := NOT bInOut;

// The warning may be restored at the end of the access area
{warning restore C0371}
```

初始化示例：

在下面的示例中，1 个输入变量和 1 个属性由 1 个功能块初始化，该功能块的 FB_init 方法 [▶ 787]带有 1 个附加参数。

功能块 FB_Sample：

```
FUNCTION_BLOCK FB_Sample
VAR_INPUT
    nInput           : INT;
END_VAR
VAR
    nLocalInitParam  : INT;
    nLocalProp       : INT;
END_VAR
```

方法 FB_Sample.FB_init：

```
METHOD FB_init : BOOL
VAR_INPUT
    bInitRetains : BOOL;  // if TRUE, the retain variables are initialized (warm start / cold start)
    bInCopyCode  : BOOL;  // if TRUE, the instance afterwards gets moved into the copy code (online
change)
    nInitParam   : INT;
END_VAR
```

```
nLocalInitParam := nInitParam;
```

属性 FB_Sample.nMyProperty 和相关的 Set 函数：

```
PROPERTY nMyProperty : INT
```

```
nLocalProp := nMyProperty;
```

MAIN 程序：

```
PROGRAM MAIN
VAR
    fbSample  : FB_Sample(nInitParam := 1) := (nInput := 2, nMyProperty := 3);
    aSample   : ARRAY[1..2] OF FB_Sample[(nInitParam := 4), (nInitParam := 7)]
                            := [(nInput := 5, nMyProperty := 6), (nInput := 8, nMyProperty := 9)];
END_VAR
```

初始化结果：

- fbSample
  - nInput = 2
  - nLocalInitParam = 1
  - nLocalProp = 3
- aSample[1]
  - nInput = 5
  - nLocalInitParam = 4
  - nLocalProp = 6
- aSample[2]
  - nInput = 8

- nLocalInitParam = 7
- nLocalProp = 9

另请参见：

- 对象接口属性 [▶ 98]
- 面向对象的编程 [▶ 158]
- 扩展功能块 [▶ 177]
- 引用编程 > 方法 FB_init、FB_reinit 和 FB_exit [▶ 786]

7.4.6 对象接口

符号： 

关键字：接口

接口是 1 种面向对象的编程的工具。对象 **Interface**（接口）描述了 1 组方法和属性原型。这里的原型是指方法和属性只包含声明，而不包含实现。

这样，您可以通过相同的方式使用具有共同属性的各种功能块。

您可以在 **Interface**（接口）对象中添加对象 **Interface property**（接口属性）和 **Interface method**（接口方法）。

创建对象接口

1. 在 PLC 项目树的 **Solution Explorer**（解决方案资源管理器）中，选择 1 个文件夹。
2. 在上下文菜单中，选择命令 **Add > Interface...**（添加 > 接口……）
   ⇨ **Add Interface**（添加接口）对话框会打开。
3. 输入名称，还可选择要扩展的接口。
4. 点击 **Open**（打开）。

⇨ 接口被添加到 PLC 项目树中，并在编辑器中打开。

对话框添加接口

名称	接口名称

继承

高级	 ：扩展在输入字段中输入或通过输入助手  选择的接口。这意味着新接口扩展的所有接口方法在新接口中也可用。

接口应用

1. 通过编译器检查接口声明

- 在 1 个接口（例如，I_Sample）中，您可以声明与该接口相关的方法和属性（包括返回类型、输入等）。
- 在与该接口相对应并应提供相应的方法和属性的功能块中，您可以实现接口 I_Sample。
  - 功能块在其声明部分的 IMPLEMENTS 列表中包含接口（例如，FB_Sample IMPLEMENTS I_Sample）。
  - 1 个功能块可以实现 1 个或多个接口（例如，FB_Sample IMPLEMENTS I_Sample1、I_Sample2）。
- 实现接口的功能块必须包含在该接口中定义的所有方法和属性（接口方法和属性）。方法和属性的声明必须与接口中的声明完全一致（名称、返回类型、输入、输出）。
- 功能块在接口方法和接口属性中添加功能块特定代码。如果 1 个接口由多个功能块实现，您可以在不同的功能块中使用带有参数和不同实现代码的相同方法。
- 对于实现 1 个或多个接口的功能块，编译器会检查该功能块是否符合相应的接口声明。如果接口中的元素声明与功能块中的元素声明不同，或者如果接口中包含更多元素，而功能块中没有包含这些元素，则编译器会报错。

2. 通过接口变量调用功能块实例的方法和属性

除了通过编译器自动验证接口声明之外，您还可以使用接口来通过接口变量调用功能块实例的接口方法或接口属性。

- 首先，将接口（例如 iSample : I_Sample;）和正确实现接口的功能块实例化（请参见用例 1：通过编译器检查接口声明）。
- 然后，您可以将接口变量分配给正确实现接口的功能块的每个实例。如果接口变量尚未分配，则该变量在在线模式下包含值 0。
- 在最后一步中，您可以通过接口变量调用接口方法或属性。为接口所引用的功能块调用该方法或属性。
- 这种实现可以通过接口变量以一致的方式使用不同但相似的功能块。例如，根据项目状态，您可以为接口变量分配 1 个特定的功能块实例，这样，尽管根据项目状态使用的是不同的功能块实例，但对接口方法和属性的调用是相同的。

在通过接口变量调用方法或属性之前，务必将接口类型变量分配给功能块实例。

注意

- 您不能在接口中声明变量。接口没有实现部分，也没有动作。仅可定义方法和属性的集合，其中包含声明代码，但没有实现代码。
- 接口类型变量是对功能块实例的引用。TwinCAT 总是将定义为接口类型的变量视为引用。

接口引用和在线更改

- 接口引用从 TwinCAT 3.0 build 3100 开始自动重定向，因此，即使是在线更改，也能始终引用正确的接口。这需要额外的代码和时间，根据受影响对象的数量，可能会引发问题。因此，在执行在线更改之前，程序员会看到受影响的变量和接口引用的数量，以便决定是继续执行在线更改还是中止更改。

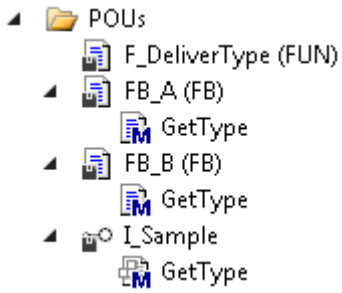
检查接口变量

- 运算符 __ISVALIDREF 仅可用于 REFERENCE TO 类型的操作数。此运算符不能用于检查接口变量。如要检查接口变量是否已分配给 1 个功能块实例，您可以检查接口变量是否等于 0（IF iSample <> 0 THEN …）。

对接口变量进行扩展监控

- 从 TwinCAT 3.1 Build 4024 版本开始提供用于监控/调试接口变量的高级选项。监控区域（声明编辑器、监视列表）中的接口变量可以扩展。然后，在接口变量下会显示当前分配的 FB 实例的符号路径和在线数据。

示例 1



接口声明：

- 您已在您的项目中添加接口 I_Sample。在接口中添加返回类型为 STRING 的方法 GetType。
- I_Sample 和 GetType 不包含实现代码。方法 GetType 只包含所需的（变量）声明（例如，返回类型）。稍后，您可以在实现接口 I_Sample 的功能块中编出方法 GetType。

```
INTERFACE I_Sample
```

方法 I_Sample.GetType：

```
METHOD GetType : STRING
```

接口实现：

- 如果您随后在项目中添加功能块并在对话框 **Add**（添加）的字段 **Implements**（实现）中输入接口 I_Sample，TwinCAT 也会将方法 GetType 自动添加到该功能块中。在此处，您可以在方法中实现功能块特定代码。
- 功能块 FB_A 和 FB_B 都可以实现接口 I_Sample：

```
FUNCTION_BLOCK FB_A IMPLEMENTS I_Sample
```

```
FUNCTION_BLOCK FB_B IMPLEMENTS I_Sample
```

- 因此，这 2 个功能块都必须包含 1 个名称为 GetType 且返回类型为 STRING 的方法。否则，编译器会报错（请参见接口应用 [▶ 94]部分中的用例 1）。

方法 FB_A.GetType：

```
METHOD GetType : STRING
```

```
GetType := 'FB_A';
```

方法 FB_B.GetType：

```
METHOD GetType : STRING
```

```
GetType := 'FB_B';
```

接口应用：

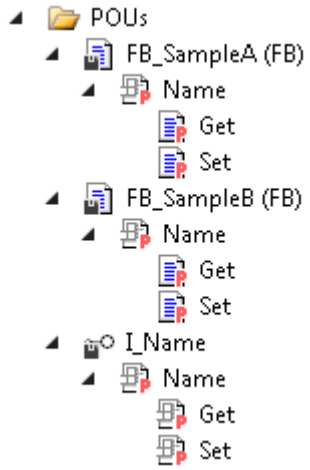
- 函数 F_DeliverType 包含接口 I_Sample 类型的输入变量的声明。在该函数中，通过接口变量 iSample 可以调用接口方法 GetType。在这种情况下，调用 FB_A.GetType 还是 FB_B.GetType 取决于传输的功能块类型（请参见接口应用 [▶ 94]部分中的应用 2）。

```
FUNCTION F_DeliverType : STRING
VAR_INPUT
    iSample : I_Sample;
END_VAR

F_DeliverType := iSample.GetType();
```

- 您可以将实现接口 I_Sample 的功能块实例（例如，FB_A 和 FB_B）分配给函数 F_DeliverType 的输入变量。
- 函数调用示例：
  - 如果将功能块实例 fbA 传输到函数 F_DeliverType，由于接口变量 iSample 指向功能块实例 fbA，因此，在函数内部将会调用方法 fbA.GetType。该方法调用会产生返回值“FB_A”，而该返回值又会由函数 F_DeliverType 返回，并在主程序中被分配给变量 sResultA。
  - 由于在函数 F_DeliverType 内部调用方法 fbB.GetType，因此 sResultB 会接收到值“FB_B”。

```
PROGRAM MAIN
VAR
    fbA       : FB_A;
    fbB       : FB_B;
    sResultA  : STRING;
    sResultB  : STRING;
END_VAR

sResultA := F_DeliverType(iSample := fbA); // call with instance of type FB_A
sResultB := F_DeliverType(iSample := fbB); // call with instance of type FB_B
```

示例 2



接口声明：

- 您已在您的项目中添加接口 I_Name。在接口中添加返回类型为 STRING 的属性 Name。该属性有 Get 和 Set 访问器方法。Get 访问器可用于从实现接口的功能块中读取任何对象的名称。Set 访问器可用于将名称写入该功能块。
- I_Name 和 Name 不包含实现代码。属性 Name 只包含所需的声明（返回类型）。因此，您不能在接口定义中处理 Get 和 Set 方法，但您可以在稍后实现接口 I_Name 的功能块中进行此类处理。

```
INTERFACE I_Name
```

属性 I_Name.Name：

```
PROPERTY Name : STRING
```

接口实现：

- 功能块 FB_SampleA 和 FB_SampleB 可实现接口 I_Name。
- 如果已经指定接口，例如，在对话框 **Add**（添加）中创建功能块时，TwinCAT 会在功能块 FB_SampleA 和 FB_SampleB 下自动添加带有 Get 和 Set 方法的属性 Name。
- 例如，您可以编辑功能块下方的访问器方法，以便读取变量 sVar1，从而获得对象的名称。在实现相同接口 I_Name 的 FB_SampleB 中，您可以实现 Get 方法代码，然后该代码会返回另一个对象的名称。Set 方法可用于将 MAIN 程序提供的名称（“abc”）写入功能块 FB_SampleB 中。
- 功能块 FB_SampleA 和 FB_SampleB 分别可实现接口 I_Name：

```
FUNCTION_BLOCK FB_SampleA IMPLEMENTS I_Name
VAR
    sVar1 : STRING := 'My name is A.';
END_VAR

FUNCTION_BLOCK FB_SampleB IMPLEMENTS I_Name
VAR
    sVar2 : STRING := 'My name is B.';
END_VAR
```

- 因此，这 2 个功能块都必须包含 1 个名称为 Name 且返回类型为 STRING 的属性。属性必须有 Get 和 Set 方法。否则，编译器会报错（请参见接口应用 [▶ 94]部分中的用例 1）。

FB_SampleA.Name.Get：

```
Name := sVar1;
```

FB_SampleA.Name.Set：

```
sVar1 := Name;
```

FB_SampleB.Name.Get：

```
Name := sVar2;
```

FB_SampleB.Name.Set：

```
sVar2 := Name;
```

接口应用：

- 功能块的属性既可以通过相应的功能块实例进行访问，也可以通过类型 I_Name 的接口变量进行访问。通过接口变量进行访问的前提条件是，该变量事先已被分配给 1 个实现接口 I_Name 的特定功能块实例。

```
PROGRAM MAIN
VAR
    iName     : I_Name;

    fbSampleA : FB_SampleA;
    sNameA    : STRING;       // will be 'My name is A.'

    fbSampleB : FB_SampleB;
    sNameB    : STRING;       // will be 'My name is B.' after first cycle
                              // and will be 'New name' afterwards
END_VAR

// assign FB instance fbSample1 to interface variable
iName := fbSampleA;

// access to name property of fbSample1 via interface variable (Get)
sNameA := iName.Name;

// access to name property of fbSample2 via FB instance (Get and Set)
sNameB := fbSampleB.Name;
fbSampleB.Name := 'New name';
```

另请参见：

- 实现接口 [▶ 183]
- 扩展接口 [▶ 182]

7.4.6.1 对象界面方法

符号： 

接口方法是 1 种面向对象的编程的手段。通过命令 **Add > Method...**（添加 > 方法……）可以将 **Interface method**（接口方法）对象添加到接口中

如果已在接口下添加方法，您只能在该方法中添加变量声明（输入、输出和输入/输出变量）并将其实例化。

只有当 1 个功能块“实现”了属于该方法的接口后，程序代码才能被添加到方法中。然后，TwinCAT 会将该方法添加到功能块下。

另请参见：

- 对象接口 [▶ 94]
- 对象方法 [▶ 82]
- 接口的实现 [▶ 183]

7.4.6.2 对象接口属性

符号： 

接口属性是 1 种面向对象的编程的工具。通过命令 **Add > Property...**（添加 > 属性……）可以将对象 **Interface property**（接口属性）添加到接口中，以便使用 Get 和/或 Set 访问器方法扩展对接口的描述。接口属性中的访问器方法不包含实现代码。如果您删除 Set 访问器，则该属性只能进行读取访问，而不能进行写入访问。

Get 访问器可用于对属性进行读取访问。
Set 访问器可用于对属性进行写入访问。

如果属性没有 Get 和/或 Set，则可以使用 **Add**（添加）命令将该访问器添加到接口属性中。

如果您使用包含属性的接口来扩展功能块或程序，则 TwinCAT 会在 POU 下的 PLC 项目树中自动添加该接口以及相应的 Get 和/或 Set 访问器。然后，您可以在 Get 和/或 Set 访问器中实现代码。

另请参见：

- 对象接口 [▶ 94]
- 对象属性 [▶ 88]
- 接口的实现 [▶ 183]

7.5 在 IEC 中创建源代码

源代码

“源代码”表示实现代码，您可以借助相应的编程语言编辑器将其添加到编程块中。

编程语言

当您创建编程块时，您可以决定使用哪种实现语言进行编程。除 IEC 语言之外，CFC 也可供使用。

编程语言编辑器

双击 PLC 项目树中的对象，可在相应的编程语言编辑器中打开编程块进行编辑。因此，功能块既可以出现在基于文本的 ST 编辑器中，也可以出现在 FBD/LD/IL、SFC 或 CFC 的图形编辑器中。每个编辑器均由 2 个窗口组成：上部窗口可用于输入声明，根据设置的不同，声明可以采用文本形式或表格形式。下部窗口可用于实现代码。在 TwinCAT 选项的相应选项卡中可以为整个项目配置每个编辑器的显示和行为。

请注意，即使应用程序处于在线模式，也可以在编辑器中选择在离线模式下打开编程块。（命令：编辑对象（脱机）[▶ 822]）

另请参见：

- 引用编程 > 编程语言及其编辑器 [▶ 569]

7.5.1 FBD/LD/IL

通过组合编辑器可以使用 FBD（功能块图）、LD（梯形图）和 IL（指令表）语言进行编程。

FBD 和 LD 编程的基本单元是网络。每个网络都包含 1 个结构，该结构可以表示以下内容：逻辑或算术表达式、编程块（函数、功能块、程序等）的调用、跳转或返回语句。IL 实际上不需要网络。然而，在 TwinCAT 中，1 个 IL 程序也至少包括 1 个网络，以支持转换为 FBD 或 LD。因此，IL 程序的结构也应考虑到网络。

另请参见：

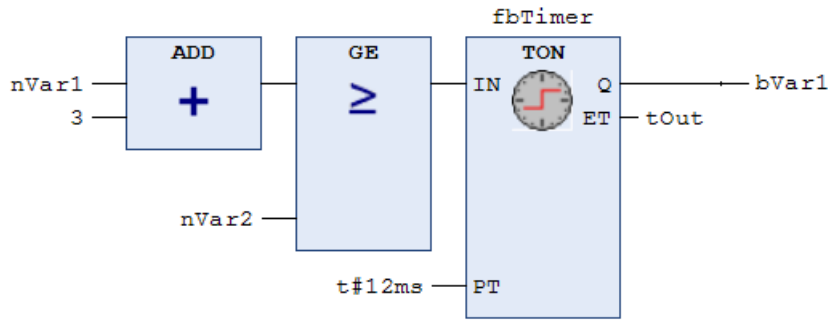
- 引用编程：功能块图/梯形图/指令表（FBD/LD/IL） [▶ 594]
- TC3 用户界面文档：FBD/LD/IL [▶ 951]

功能块图（FBD）

功能块图是 1 种图形化的 IEC 61131 编程语言。它使用 1 个网络列表。每个网络都包含 1 个结构，该结构可能包含逻辑和算术表达式、功能块调用、跳转或返回语句。

所使用的功能块与 Boolean 代数很相似。功能块和变量通过连接线相连接。网络中的信号流从左向右流动。编辑器中的信号流从上至下流动，从网络 1 开始。

示例：



梯形图（LD）

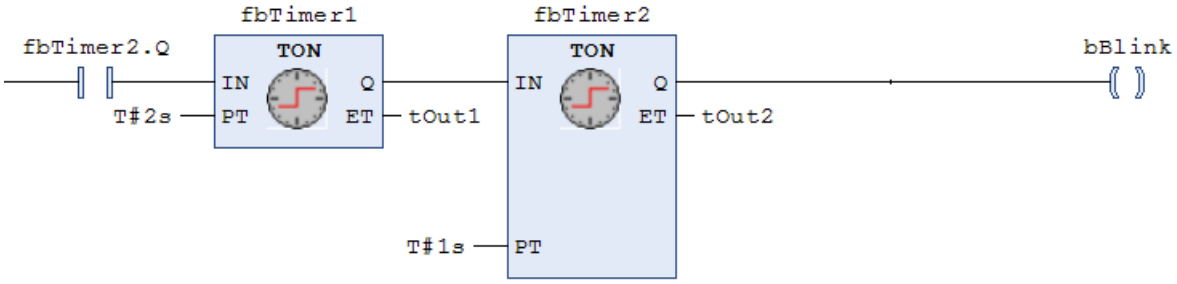
梯形图（LD）是 1 种图形化编程语言，它以电路图的原理为基础。

梯形图适用于配置逻辑控制电路，也可用于创建网络，类似于 FBD。因此，LD 非常适合用于控制其他程序块的调用。

梯形图由一系列网络组成。网络在左侧由 1 条垂直线（母线）限定。网络包含由触点、线圈、可选功能块（POU）和连接线组成的电路图。网络的左侧有 1 个触点或一系列触点，从左向右传递状态 ON 或 OFF，与 Boolean 值 TRUE 和 FALSE 相对应。每个触点都与 1 个 Boolean 变量相关联。如果该变量为 TRUE，则通过连接线从左向右传递该状态。否则，传递的状态为 OFF。这样就会为网络右侧部分的线圈分配左侧部分中的值 ON 或 OFF。相应地，值 TRUE 或 FALSE 会被写入分配给它们的 Boolean 变量中。

如果元素串联连接，则相当于逻辑 AND 链路。如果它们并联连接，则相当于逻辑 OR 链路。通过元素的直线表示该元素的否定。输入或输出的否定以圆圈符号表示。

示例：



IEC 61131-3 定义了由各种类型的触点和线圈组成的完整的 LD 指令集。触点从左向右（根据其类型）传导电流。线圈存储传入值。为触点和线圈分配 Boolean 变量。它们可以通过跳转、向后跳转、标记和注释来补充 LD 网络。

指令表（IL）

指令表是 1 种符合 IEC 61131 标准的编程语言，与汇编语言类似。它支持基于累加器的编程。

指令表（IL）由一系列语句组成。每条指令从新的 1 行开始，根据操作类型的不同，包含 1 个运算符以及 1 个或多个用逗号分隔的操作数。指令前面可以是标记，后面是冒号。标记用于识别指令，可用作跳转目标。注释必须是 1 行中的最后一个元素。在指令之间可以添加空行。

支持所有 IEC 61131-3 运算符以及多输入、多输出、否定、注释、设置/重置输出和条件/非条件跳转。

每条指令主要基于将值加载到累加器中（LD 指令）。然后，使用累加器中的参数执行相应的操作。操作结果被写回累加器，通过 ST 指令从累加器存储到特定的位置。

指令表支持比较运算符（EQ、GT、LT、GE、LE、NE）和跳转，用于编程条件执行或循环。跳转可以是无条件的（JMP），也可以是有条件的（JMPC / JMPCN）。对于条件跳转，程序会检查累加器中的值是 TRUE 还是 FALSE。

示例：

```
PROGRAM IL
VAR
    fbTimer1 : TON;
    fbTimer2 : TON;
    bVar : BOOL;
    nVar : INT;
    tIn1 : TIME;
    tOut1 : TIME;
END_VAR
```

```
1   LD      bVar
    ST      fbTimer1.IN     starts timer with rising edge, resets time…
    JMP     mark1
    CAL     fbTimer1(
            PT:=tIn1,
            ET:=>tOut1)
    LD      fbTimer1.Q      gets TRUE, delay time (PT) after a rising…
    ST      fbTimer2.IN     starts timer with rising edge, resets time…
2   mark1:
    LD      nVar
    ADD     230
```

7.5.1.1 编程功能块图（FBD）

用功能块图实现语言（FBD）创建 POU

1. 在 PLC 项目树的 **Solution Explorer**（解决方案资源管理器）中，选择 1 个文件夹。
2. 在上下文菜单中，选择命令 **Add** > **POU...**（添加 > POU……）
   ⇨ **Add POU**（添加 POU）对话框会打开。
3. 输入名称并选择实现语言“功能块图（FBD）”。
4. 点击 **Open**（打开）。

⇨ POU 被添加到 PLC 项目树中，并在编辑器中打开。它由上部的声明编辑器和下部带空网络的实现部分组成。**Toolbox**（工具箱）视图会自动打开，为 FBD 编程提供合适的元素、运算符和功能块。

编程网络

1. 点击进入实现部分中自动添加的空网络。
   ⇨ 网络以黄色突出显示，左边缘带有网络编号的部分以红色突出显示。
2. 右键点击可打开上下文菜单。
   ⇨ 它为此时可以添加的元素提供了添加命令。
3. 通过菜单命令或从 **Toolbox**（工具箱）视图中拖动元素，添加所需的编程元素。
4. 例如，选择命令 **Insert Assignment**（插入赋值）。
   ⇨ 赋值行已添加。3 个问号代表赋值源和赋值目标。
5. 选择问号，并用所需的变量替换它们。输入助手可供使用。
6. 将光标移至赋值行上。
   ⇨ 其他元素的可能插入位置显示为灰色哈希字符。点击哈希字符，选择位置。合适的插入指令再次可供使用。
7. 或者，您也可以用鼠标将元素从 **Toolbox**（工具箱）视图拖到网络中。例如，在 **Toolbox**（工具箱）视图中，点击功能块元素，按住鼠标按钮并将光标移至网络上。
   ⇨ 所有可能的插入位置均显示为绿色。
8. 松开鼠标按钮，插入功能块。
   ⇨ 在网络中会显示功能块。方框（如果使用功能块，则需要使用它）上方的内部功能块类型和实例名称仍用 3 个问号表示。
9. 选择方框内的 3 个问号，并用功能块名称替换它们。输入助手可供使用。

⇨ 所选功能块的输入和输出会显示出来。它们仍然用问号表示，对于功能块也用实例名称表示。

另请参见：

- TC3 用户界面文档：命令：插入赋值 [▶ 953]

编程分支（子网络）

1. 通过 **FBD/LD/IL** 菜单中的命令 **Insert Network**（插入网络）或者从上下文菜单或从 **Toolbox**（工具箱）视图中，在您的 POU 的实现部分添加新网络。
2. 例如，将 ADD 运算符拖入空网络，并用数据类型为 INT 的 2 个变量替换 3 个问号。
3. 将元素 **Branch**（分支）从 **Toolbox**（工具箱）视图拖入您的实现中，在运算符输出端的绿色插入位置直接松开鼠标按钮。

⇨ 线路分支将运算符方框输出端的处理线分成 2 个子网络。2 个子网络中的每个子网络都可以添加更多的 FBD 元素或线路分支。

另请参见：

- TC3 用户界面文档：命令：插入网络 [▶ 952]

7.5.1.2 编程梯形图（LD）

用梯形图实现语言（LD）创建 POU

1. 在 PLC 项目树的 **Solution Explorer**（解决方案资源管理器）中，选择 1 个文件夹。
2. 在上下文菜单中，选择命令 **Add** > **POU...**（添加 > POU……）

   ⇨ **Add POU**（添加 POU）对话框会打开。
3. 输入名称并选择实现语言梯形图（LD）。
4. 点击 **Open**（打开）。

⇨ TwinCAT 会将 POU 添加到 PLC 项目树中，并在编辑器中打开它。在实现部分中添加 1 个空网络。在左侧，该空网络由 1 条垂直线限定，该线代表母线。**Toolbox**（工具箱）视图会自动打开，为 LD 编程提供合适的元素、运算符和功能块。

添加触点和功能块（TON）

✓ 在编辑器中打开 1 个实现语言为 LD 的 POU，并添加 1 个空网络。

1. 点击 **Toolbox**（工具箱）视图中的类别 **Ladder elements**（梯形元素）。
2. 点击 **Contact**（触点）元素，将其拖入您的网络，然后在插入位置 **Start here**（从此处开始）松开鼠标按钮。

   ⇨ 触点直接被添加到网络左侧的垂直线上。
3. 点击 **???**，输入 Boolean 变量的名称。输入助手可用于此目的。
4. 在 **Toolbox**（工具箱）视图中，点击功能块类别，并将功能块 TON 拖到已添加触点右侧的连接线上的插入位置。

   ⇨ TwinCAT 在触点右侧添加功能块 TON。触点与 TON 功能块的输入端 IN 相连。
5. 在输入端 PT 输入时间常量，例如 T#3s。

⇨ 如果您的触点的变量变为 TRUE，则 TON 功能块的输入端 IN 也会变为 TRUE。例如，TON 功能块将值 TRU 传递给输出端 Q，接通延迟时间为 T#3s。

插入封闭分支

✓ 在编辑器中打开 1 个实现语言为 LD 的 POU，并添加 1 个空网络。

1. 点击进入空网络，并在菜单 **FBD/LD/IL** 中选择命令 **Insert Contact**（插入触点）。
2. 选择触点左侧的连接线，并在菜单 **FBD/LD/IL** 中选择命令 **Set Branch Start Point**（设置分支起点）。

   ⇨ 红色矩形表示连接线上的起点。TwinCAT 用蓝色矩形表示分支的所有可能终点。
3. 点击蓝色矩形，设置您的封闭线路分支的终点。

⇨ TwinCAT 会在起点和终点之间添加线路分支。程序将通过这 2 个分支运行到终点。
如果您在功能块而不是触点处插入线路分支，则只有在其他分支均不为 TRUE 时才会调用该功能块。

另请参见：

- TC3 用户界面文档：命令：插入接触点 [▶ 956]
- TC3 用户界面文档：命令：设置分支开始点 [▶ 961]

7.5.1.3 编程指令表（IL）

如果需要，可通过 TwinCAT 选项启用 IL。（工具 > 选项 > TwinCAT > PLC 编程环境 > FBD、LD 和 IL > IL）

在指令表实现语言（IL）中创建 POU

1. 在 PLC 项目树的 **Solution Explorer**（解决方案资源管理器）中，选择 1 个文件夹。
2. 在上下文菜单中，选择命令 **Add > POU...**（添加 > POU……）
   ⇨ **Add POU**（添加 POU）对话框会打开。
3. 输入名称并选择实现语言指令表（IL）。
4. 点击 **Open**（打开）。

⇨ TwinCAT 会将 POU 添加到 PLC 项目树中，并在编辑器中打开它。在实现部分中已经添加 1 个网络。

编程网络（例如，针对 ADD 操作）

✓ 在编辑器中打开 1 个带有空网络的 POU（IL）。

1. 点击突出显示行的第 1 列，并输入运算符 LD。
2. 按 **[Tab]** 键。
   ⇨ 光标跳转到第 2 列。
3. 输入您的 ADD 操作的第 1 个求和值，例如，“6”。
4. 按 **[Ctrl] + [Enter]** 或在菜单 **FBD/LD/IL** 中选择命令 **Insert IL line below**（在下面插入 IL 行）。
   ⇨ TwinCAT 会在底部添加 1 个新的语句行。焦点位于该行的第 1 列。
5. 输入 ADD 并按 **[Tab]**。
6. 输入您的 ADD 操作的第 2 个被加数，例如 "12"。
7. 按 **[Ctrl] + [Enter]**
8. 输入运算符 ST 并按 **[Tab]**。
9. 输入数据类型为 INT 的变量，例如，“nVar”。

⇨ 在示例“16”中，结果存储在 nVar 中。

另请参见：

- TC3 用户界面文档：命令：在下方插入 IL 行 [▶ 958]

调用功能块

✓ 在编辑器中打开 1 个带有空网络的 POU（IL）。在声明部分中声明数据类型为 <function block> 的变量，例如，fbSample : CTU;.

1. 点击进入突出显示行的第 1 列，并在菜单 **FBD/LD/IL** 中选择命令 **Insert Box**（插入方框）。
   ⇨ **Input Assistant**（输入助手）会打开。
2. 在类别 **Function blocks**（功能块）或 **Module calls**（模块调用）中，选择所需的功能块，例如，库 Tc2_Standard 中的计数器 CTU，然后点击 **OK**（确定）。
3. TwinCAT 会添加所选的功能块 CTU，如下所示：

```
CAL     ???(
    CU:= ???,
  RESET:= ???,
    PV:= ???,
     Q=> ???,
    CV=> ???)
```

4. 用变量名和功能块输入/输出的值或变量替换 ??? 字符串。
5. 除了通过输入助手添加功能块之外，您还可以直接在编辑器中输入调用。

另请参见：

- TC3 用户界面文档：命令：插入框 [▶ 953]

7.5.2 连续功能图（CFC）

连续功能图（CFC）实现语言是 1 种图形化编程语言，扩展了 IEC 61131-3 标准语言。

您可以使用 CFC 在编程块中插入并自由定位元素。通过连接，您可以将各元素互相连接成网络。您还可以插入反馈。结果是 1 个清晰的功能图，您可以像阅读电路图或块状图一样阅读。

功能图的执行顺序以数据流为基础。1 个编程块可以处理多个数据流。这样，数据流没有公用数据，编辑器中有几个网络，而它们之间没有连接。FBD、LD 或 IL 中的编程块采用基于网络的执行顺序。

面向页面的实现语言连续功能图（CFC）也是 1 种图形化编程语言，扩展了 IEC 61131-3 的标准语言。

您可以使用这种语言对耗费空间的复杂功能图进行图形编程。与连续功能图（CFC）中相同的元素和命令可用于此目的。此外，您还可以将分发的代码安排在任何数量的页面上。这样，您就可以创建大量的功能图，这些功能图打印起来也很方便。在每个页面的边框区域，您可以将输入端和汇端连接标记安排在左边，并将输出端和源端连接标记安排在右边。在插入连接线时，这将为您提供支持并帮助您更好地了解全局。

您无法在面向页面的连续功能图（CFC）和连续功能图（CFC）实现语言之间切换编程块。

另请参见：

- 引用编程：连续功能图（CFC）和面向页面的 CFC [▶ 607]
- TC3 用户界面文档：CFC [▶ 938]

7.5.2.1 CFC 语言编程

您可以在 CFC 编辑器中将编程块布线，并创建描述性块状图。

编辑器可为您提供以下支持：

- 用元素和连接线编程
- 将实例和变量拖到编辑区中
- 连接线的自动路由
- 自动连接
- 通过控制点固定连接线
- 碰撞检测
- 连接标记的输入支持
- 在线模式下的强制和写入值
- 使用箭头键移动选区
- 对没有未连接的连接的功能块的简化表示

用实现语言连续功能图（CFC）创建 POU

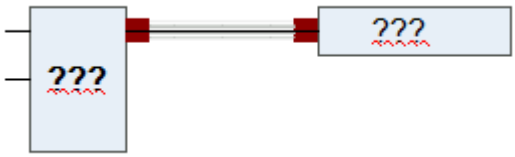
1. 在 PLC 项目树的 **Solution Explorer**（解决方案资源管理器）中，选择 1 个文件夹。
2. 在上下文菜单中，选择命令 **Add > POU...**（添加 > POU……）
   ⇨ **Add POU**（添加 POU）对话框会打开。
3. 输入名称并选择实现语言 **Continuous Function Chart (CFC)**（连续功能图（CFC））。
4. 点击 **Open**（打开）。

⇨ TwinCAT 会将 POU 添加到 PLC 项目树中，并在编辑器中打开它。

插入元素并用连接线将它们互相连接

1. 在编辑器中定位 **Function block**（功能块）元素和 **Output**（输出）元素。用鼠标将这 2 个元素从 **Toolbox**（工具箱）视图拖到编辑器中。
2. 点击 **Function block**（功能块）元素的输出端。
   ⇨ 输出用红色正方形标记。
3. 按住鼠标按钮，从 **Function block**（功能块）元素的输出端到 **Output**（输出）元素的输入端画 1 条连接线。
   ⇨ 当到达输入引脚时，光标会改变符号。
4. 松开鼠标按钮。

⇨ 功能块的输出引脚与输出端的输入引脚连接。



或者，您也可以在按下 **[Ctrl]** 键的同时选择这 2 个引脚，然后在 **CFC** 菜单或上下文菜单中选择命令 **Connect Selected Pins**（连接所选引脚）。

调用实例

1. 创建新项目并添加默认 PLC 项目
2. 使用实现语言 ST 创建带有输入端和输出端的功能块 FB_Sample：

```
FUNCTION_BLOCK FB_Sample
VAR_INPUT
        nIn1 : INT;
        nIn2 : INT;
        sIn1 : STRING := 'Name';
        bIn1 : BOOL;
END_VAR
VAR_OUTPUT
        nOut : INT;
        sOut : STRING;
        bOut : BOOL;
END_VAR
VAR
END_VAR
```

```
nOut := nIn1 + nIn2;

sResult := sIn1;

IF bIn1 THEN
bOut := TRUE;
END_IF
```

3. 使用实现语言 ST 创建程序 PRG_First。
4. 将功能块实例化并声明变量：

```
PROGRAM PRG_First
VAR
        nCounter    : INT;
        fbSample1   : FB_Sample;
        fbSample2   : FB_Sample;
        nResult     : INT;
        sResult     : STRING;
        bResult     : BOOL;
END_VAR
```

5. 从 **Toolbox**（工具箱）视图中，将 1 个 **Box**（方框）元素拖至编辑器中。
6. 点击 **???** 字段并键入 ADD。
   ⇨ 该功能块类型为 ADD。该功能块可充当加法器。
7. 点击声明编辑器中的第 3 行。
   ⇨ nCounter 的声明行已选中。
8. 点击进入选区，并将选中的变量拖至实现中。将 ADD 功能块的输入端集中在这里。

⇨ 创建并声明输入端，然后将其与功能块连接。

9. 再次点击进入选区，并将变量拖至 ADD 功能块的输出端。

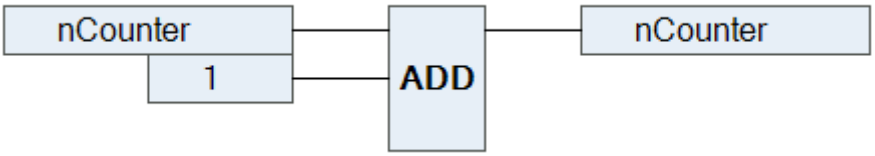
⇨ 创建并声明输出端，然后将其与功能块连接。

10. 从 **Toolbox**（工具箱）视图中，将 1 个 **Input**（输入）元素拖至实现中。

11. 点击其字段 **???** 并输入 1。

12. 连接输入端 1 与 ADD 功能块上的输入端。

⇨ 网络已编程。在运行时，网络会对周期进行计数，并将结果存储在 nCounter 中。

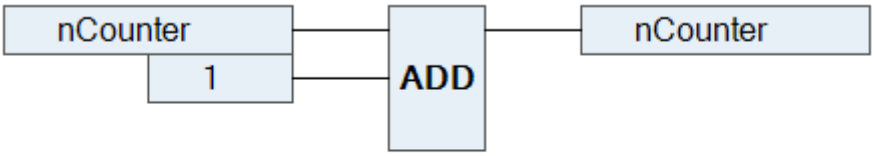


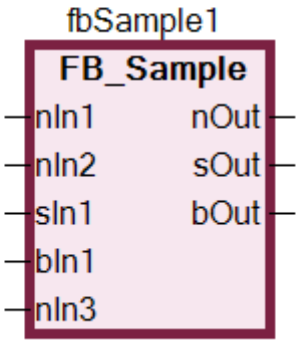
13. 点击声明编辑器中的第 4 行。

⇨ 带有 fbSample1 的行已选中。

14. 点击进入选区，并将选中的实例拖至实现中。

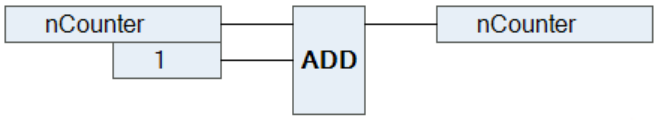
⇨ 该实例在编辑器中显示为 1 个功能块。类型、名称和功能块引脚会相应地显示。

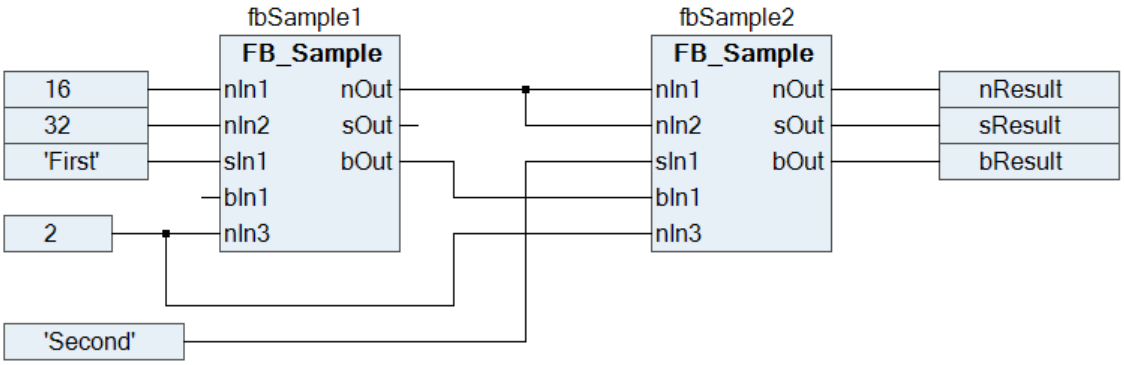




15. 将 fbSample2 实例拖入编辑器中。实例之间以及实例与输入端和输出端之间相互连接。

⇨ 示例：





使用 ST 且具有相同功能的程序可以是这样的：

```
PROGRAM PRG_First_ST
VAR
    nCounter  : INT;
    fbSample1 : FB_Sample;
    fbSample2 : FB_Sample;
    nResult   : INT;
    sResult   : STRING;
    bResult   : BOOL;
END_VAR
```

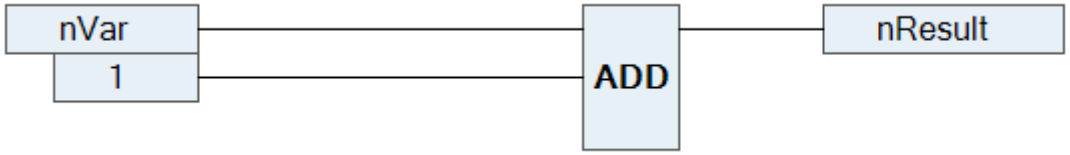
```
nCounter := nCounter + 1;
fbSample1(nIn1 := 16, nIn2 := 32, sIn1 := 'First', xItem := TRUE, nIn3 := 2, nOut => fbSample2.nIn1,
bOut => fbSample2.bIn1);
fbSample2(nIn2 := fbSample1.nOut, sIn1 := 'Second', nIn3 := 2, nOut => nResult, sOut => sResult,
bOut => bResult);
```

创建连接标记

✓ 您有 1 个带有连接元素的 CFC 编程块。

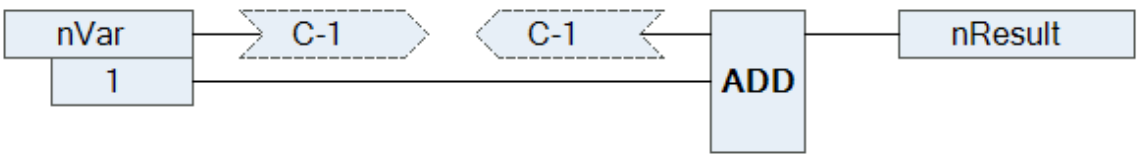
1. 在 2 个元素之间选择 1 条连接线。

   ⇨ 连接线已选中，用红色正方形标记元素的输入端或输出端 。



2. 在上下文菜单或 CFC 菜单中，选择命令 **Connection Mark**（连接标记）。

   ⇨ 连接已断开，并替换为 1 个**连接标记 - 源端**和 1 个**连接标记 - 汇端**。标记名称已自动生成。

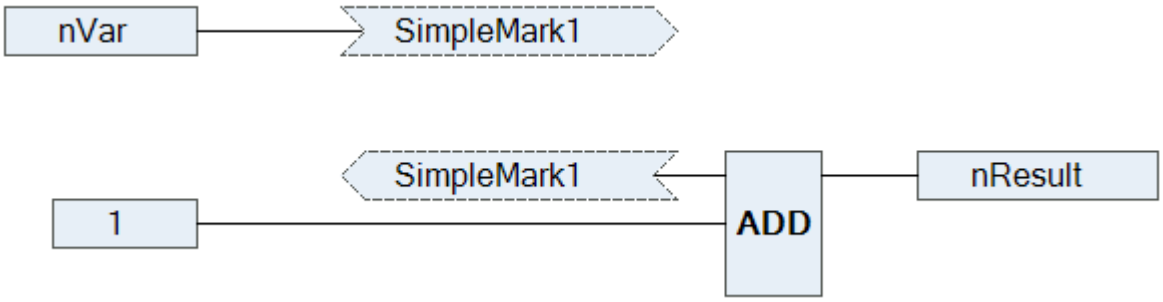


3. 点击进入连接标记源端。

   ⇨ 名称可以进行编辑。

4. 输入源端连接标记的名称。

⇨ 源端连接标记和汇端连接标记的名称相同。



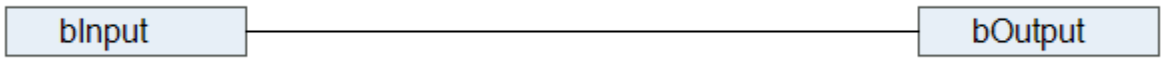
另请参见：

- TC3 用户界面文档：命令：连接标记 [▶ 948]

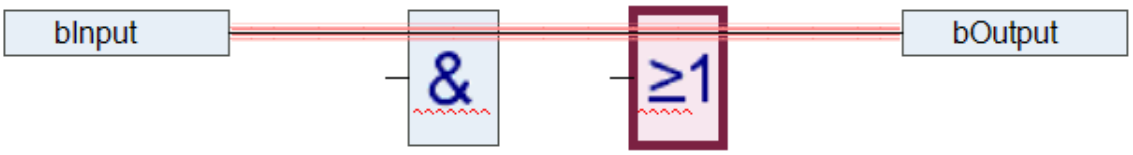
通过控制点解决碰撞和固定连接线

下面的示例说明了命令 **Route All Connections**（路由全部连接）的应用以及控制点的应用。

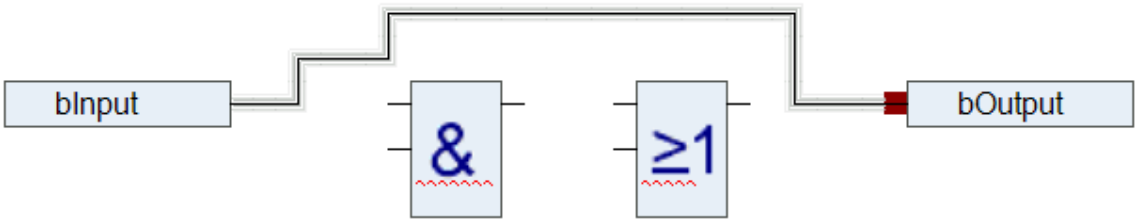
1. 定位元素**输入端**和**输出端**，并将这 2 个元素连接起来。



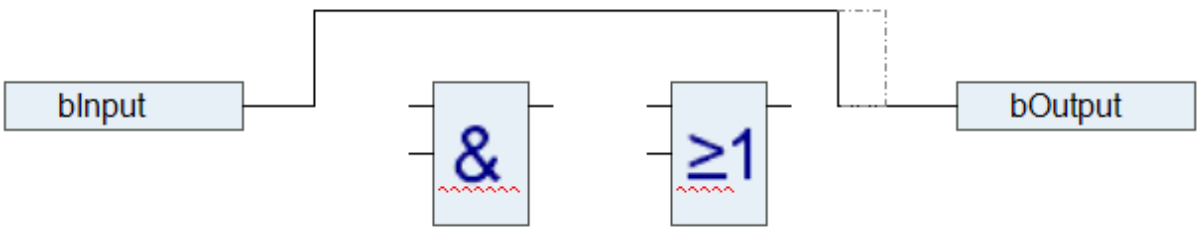
2. 在线上定位 2 个**功能块**元素。
   ⇨ 由于碰撞，连接线和功能块显示为红色。



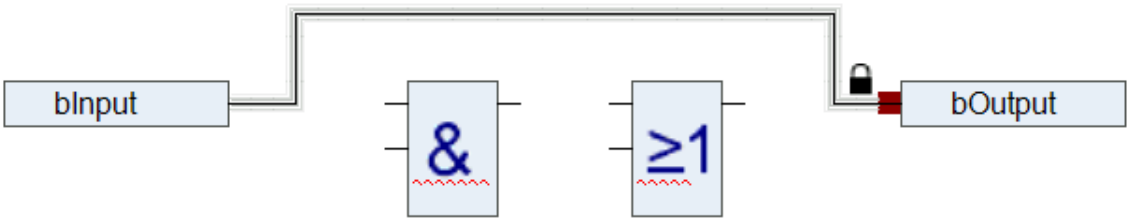
3. 在 **CFC** > **Routing**（CFC > 路由）菜单中，选择命令 **Route All Connections**（路由全部连接）。
   ⇨ 碰撞已解决。



4. 逐步改变连接线。

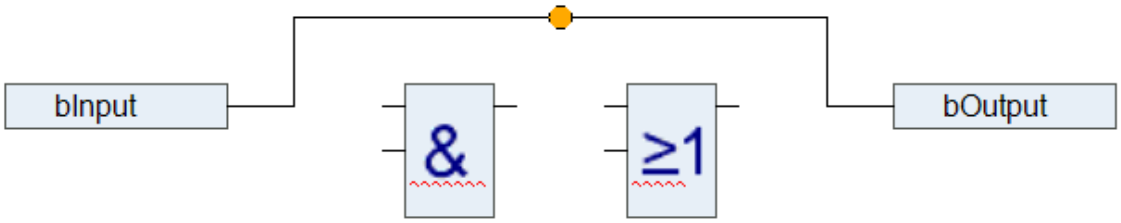


⇨ 连接线已手动更改，现已锁定为自动路由。在连接的末端，用 1 个挂锁表示这种情况。

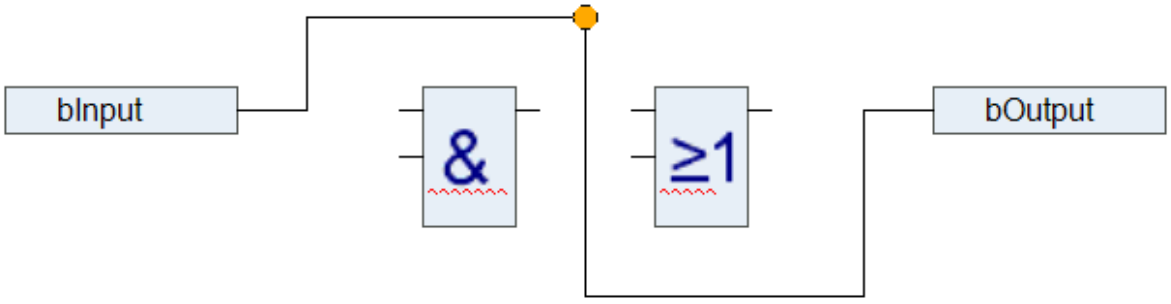


5. 选择连接线，并在 **CFC > Routing**（CFC > 路由）菜单中选择命令 **Create Control Point**（创建控制点）。或者，您也可以将控制点从 **Toolbox** （工具箱）视图拖至线上。

⇨ 在连接线上创建 1 个控制点。连接线固定在控制点上。



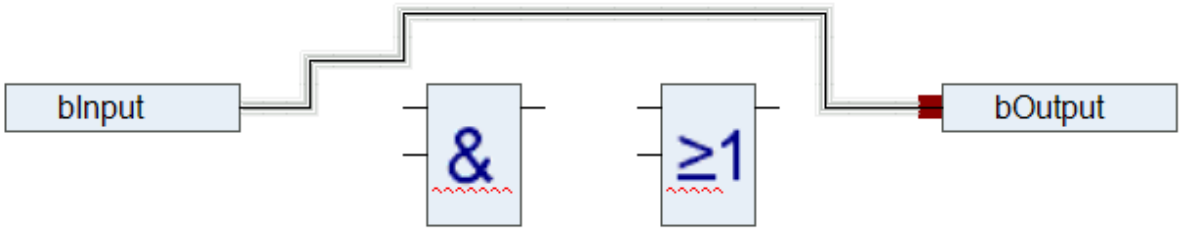
6. 按照下面的示例更改连接。



⇨ 控制点让您能够根据需要修改连接线。您可以设置任意数量的控制点。

7. 在 **CFC > Routing**（CFC > 路由）菜单中，使用命令 **Remove Control Point**（删除控制点）可以删除控制点。
8. 使用命令 **Unlock Connection**（解锁连接）或通过点击挂锁符号可以解锁连接。
9. 选择连接线，并选择命令 **Route All Connections**（路由全部连接）。

⇨ 如步骤 3 所示，连接线会自动绘制。



组内的连接不会自动路由。

另请参见：

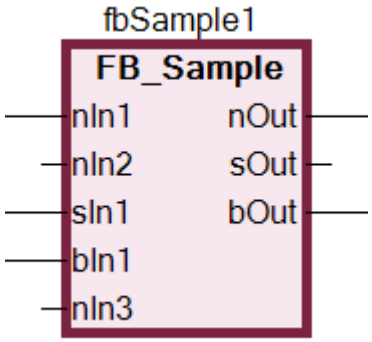
- TC3 用户界面文档：命令：引导所有连接 [▶ 947]

- TC3 用户界面文档：命令：删除控制点 [▶ 948]
- TC3 用户界面文档：命令：解锁连接 [▶ 945]

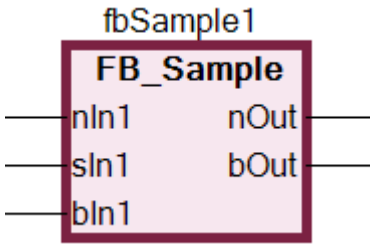
简化功能块表示

要求：1 个 CFC 编程块已打开。在编辑器中，其功能块与所有已声明的连接一起显示。

1. 选择 1 个连接部分未连接的功能块。
   ⇨ 示例：FB_Sample



   ⇨ 该功能块需要为所有连接留出空间。
2. 从菜单 **CFC** > **Routing**（CFC > 路由）中，选择 **Remove Unused Pins**（删除未使用的引脚）。
   ⇨ 功能块需要的空间较小，现在只显示与功能相关的连接。



7.5.2.2 按数据流自动执行顺序

在基于文本和基于网络的编辑器中，编程块的执行顺序是唯一确定的。但是，在 CFC 编辑器中，您可以自由定位元素，因此执行顺序最初并不是唯一的。因此，它是由 TwinCAT 根据数据流确定的，而且，在多个网络的情况下，则是根据元素的拓扑位置确定的。元素从上到下、从左到右排序。因此，在处理编程块时，可以按时间和周期进行优化。

您可以获取关于表中元素的时间顺序信息，并简要显示执行顺序。当您对带有反馈的网络进行编程时，您可以将某个元素定义为反馈循环的起点。

如有必要，您还可以在 CFC 对象中明确编辑处理顺序。为此，可将 **Explicit Execution Order**（显式执行顺序）属性的值从默认的 False 改为 True。在显式执行顺序模式下，您可以使用菜单命令编辑执行顺序。

数据流

通常，人们将编程对象在何时以及如何读取或写入数据的时间顺序称为数据流。1 个编程块可以处理许多不同的数据流，这些数据流也可以相互独立地执行。

显示执行顺序

在默认激活的自动数据流模式下，CFC 对象的执行顺序是自动确定的。您可以在 CFC 编辑器中简要显示执行顺序。

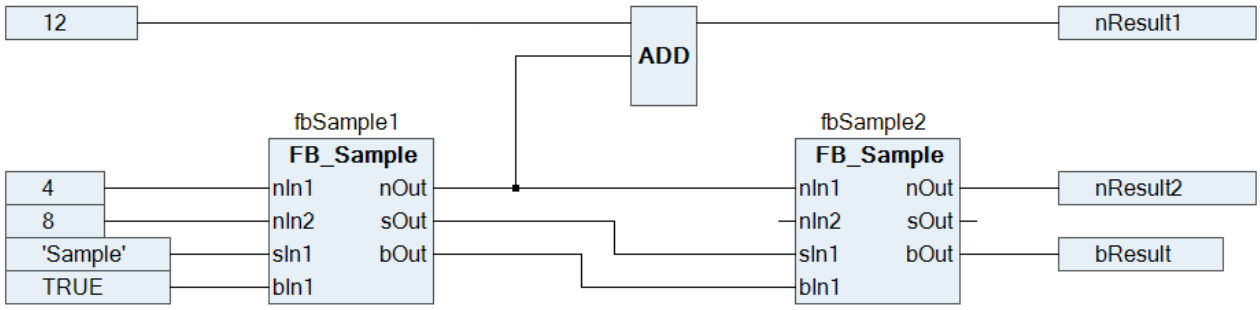
1. 创建新项目并添加默认 PLC 项目。
2. 使用 ST 实现语言创建带有输入端和输出端的功能块 FB_Sample。

```
FUNCTION_BLOCK FB_Sample
VAR_INPUT
        nIn1 : INT;
        nIn2 : INT;
        sIn1 : STRING := 'Name';
        bIn1 : BOOL;
END_VAR
VAR_OUTPUT
        nOut : INT;
        sOut : STRING;
        bOut : BOOL;
END_VAR
VAR
END_VAR
```

```
nOut := nIn1 + nIn2;

sResult := sIn1;

IF bIn1 THEN
bOut := TRUE;
END_IF
```

3. 使用 CFC 实现语言创建 Execute_CFC 程序。

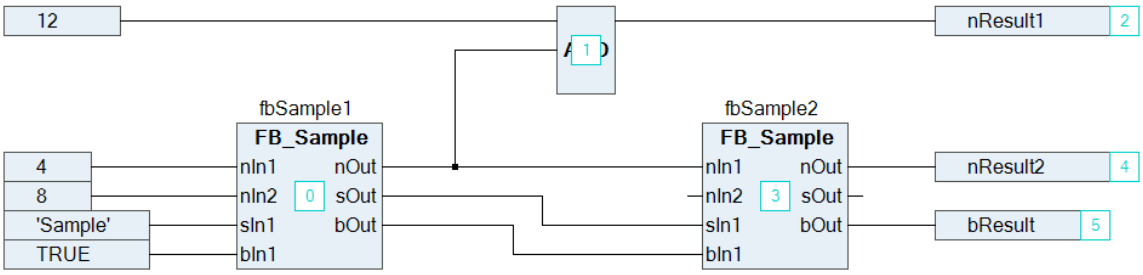
```
PROGRAM Execute_CFC
VAR
        fbSample1  : FB_Sample;
        fbSample2  : FB_Sample;
        nResult1    : INT;
        nResult2    : INT;
        bResult     : BOOL;
END_VAR
```



在 CFC 中新建的编程对象已激活自动数据流模式。编程对象的执行顺序由内部优化定义。

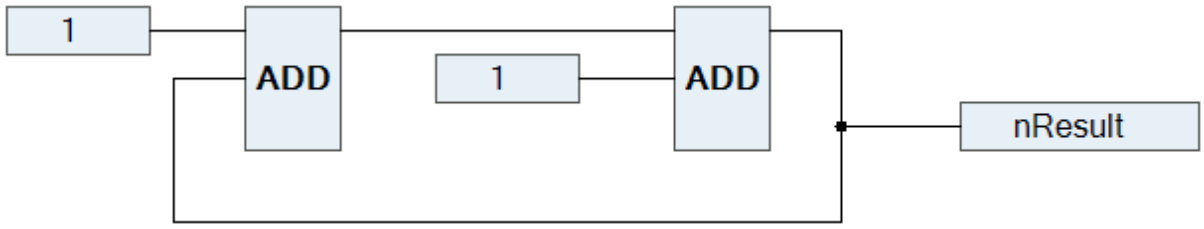
4. 在 **CFC** 菜单中，选择命令 **Execution Order** > **Display Execution Order**（执行顺序 > 显示执行顺序）。

   ⇨ 对象的执行顺序已显示。方框和输出端都有相应的编号，并反映按时间的处理顺序。只要您再次点击进入 CFC 编辑器，编号就会隐藏。

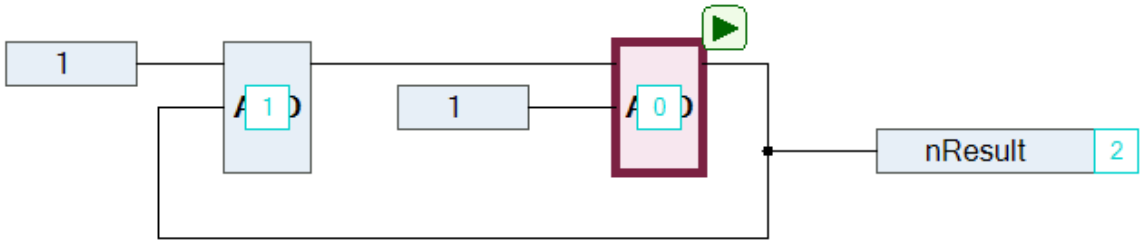


确定反馈网络中的执行顺序

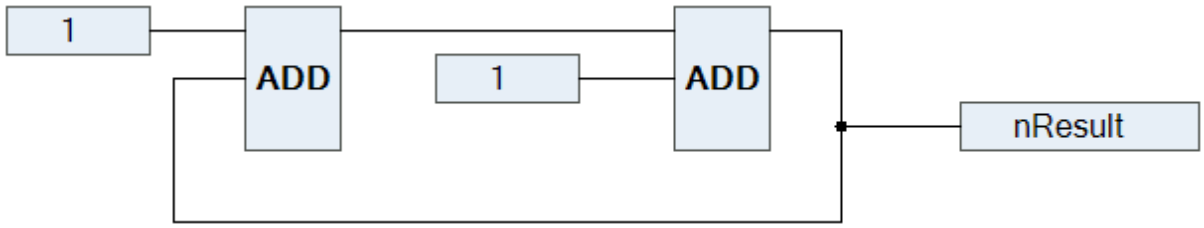
1. 创建 1 个带有反馈的 CFC 程序。



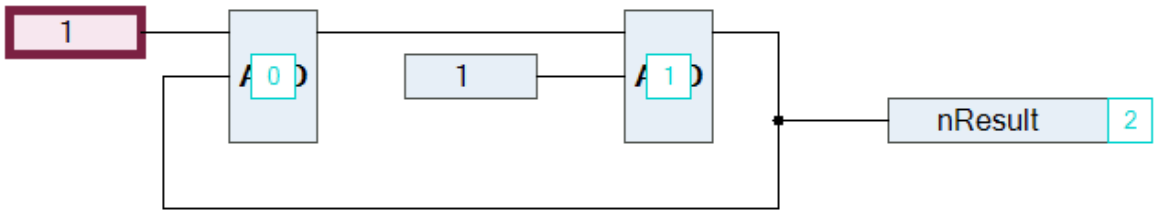
2. 在反馈中选择 1 个元素。
   ⇨ 所选元素以红色突出显示。
3. 在 CFC 菜单中，选择命令 **Execution Order > Set Start of Feedback**（执行顺序 > 设置反馈开始）。
   ⇨ 在运行时，首先处理该功能块。反馈的开始功能块由  符号定义和装饰。执行顺序会重新排序，所选元素会得到编号 0（通常是反馈中的最低数字）。



4. 再次选择启动功能块。
5. 在 CFC 菜单中，选择命令 **Execution Order > Set Start of Feedback**（执行顺序 > 设置反馈开始）。
6. 取消选择功能块作为开始功能块。执行顺序由内部定义。



7. 在 CFC 菜单中，选择命令 **Execution Order > Display Execution Order**（执行顺序 > 显示执行顺序）。
   ⇨ 数据流的执行顺序已显示。



明确定义执行顺序

通过数据流自动定义的执行顺序可以优化编程块的执行时间和周期。因此，在开发过程中，您不需要任何有关内部管理的执行顺序的信息。

在显式执行顺序模式下，您需要自行调整执行顺序并自行评估后果和影响。这也是为什么总是显示执行顺序的另一个原因。

如果您已启用显式执行顺序属性，则可以明确更改 CFC 对象的自动定义的执行顺序。

1. 在项目树中选择 1 个 CFC 对象。
2. 在上下文菜单中，点击 **Properties**（属性）。
3. 将 CFC 下的 **Excplicit Execution Order**（显式执行顺序）选项设置为 True。
   ⇨ 显式执行顺序模式已启用。在 CFC 编辑器中，网络已编号。在 **CFC** 菜单的 **Execution order**（执行顺序）中提供了关于编辑执行顺序的各种命令。
4. 打开 CFC 对象。
5. 选择 1 个编号元素，并选择命令 **Execution Order** > **Move to Beginning**（执行顺序 > 移动到开始）。
   ⇨ 执行顺序会重新排列，所选元素会得到编号 0。

另请参见：

- TC3 用户界面文档：命令：向前移动一位 [▶ 943]

7.5.3 结构化文本（ST）、扩展结构化文本（ExST）

ST 编辑器可用于使用 IEC-61131-3 编程语言“结构化文本（ST）”或“扩展结构化文本”对 POU 进行编程。与标准 IEC 61131-3 语言相比，“扩展结构化文本”可以提供某些附加功能。

结构化文本是 1 种可与其他高级语言（例如，C 或 PASCAL）相媲美的编程语言，它能够开发复杂的算法。程序代码由表达式和指令组合而成，它可以是条件式（IF...THEN...ELSE），也可以是循环式（WHILE...DO）。

表达式是 1 种在求值后返回值的结构。表达式也可以是运算符和操作数的组合。赋值也可以用作表达式。操作数可以是常量、变量、函数调用或其他表达式。

指令可控制表达式的处理方式。

在 **Tools**（工具）菜单的 **Options**（选项）和 **Customize**（自定义）对话框中，可以对该文本编辑器的行为、外观和菜单进行各种设置。各种熟悉的 Windows 功能（例如 IntelliMouse）也可用于该编辑器。

ExST - 扩展结构化文本

“扩展结构化文本（ExST）”是在 IEC 61131-3 标准中定义的“结构化文本（ST）”的 TwinCAT 特定扩展。

另请参见：

- TC3 用户界面文档：命令：自定义 [▶ 923]
- TC3 用户界面文档：命令：选项 [▶ 897]
- 引用编程：结构化文本和扩展结构化文本（ExST） [▶ 571]

7.5.3.1 编程结构化文本（ST）

原则

ST 编辑器可用于使用编程语言结构化文本和扩展结构化文本进行编程。程序代码由表达式和指令组合而成，它可以是条件式，也可以是循环式。每条指令必须以 1 个 ; 结束

变量在声明编辑器中声明。

另请参见：

- 使用声明编辑器 [▶ 64]

用实现语言结构化文本（ST）创建 POU

1. 在 PLC 项目的 **Solution Explorer**（解决方案资源管理器）中，选择 1 个文件夹。
2. 在上下文菜单中，选择命令 **Add > POU...**（添加 > POU……）
   ⇨ **Add POU**（添加 POU）对话框会打开。
3. 输入名称并选择实现语言“结构化文本（ST）”。
4. 点击 **Open**（打开）。

⇨ POU 被添加到 PLC 项目树中，并在编辑器中打开。现在，在 POU 的上部添加变量声明，在 POU 的下部添加 ST 程序代码。

7.5.4 顺序功能图（SFC）

使用 SFC 编辑器，以 IEC 61131-3 编程语言顺序功能图对 POU 进行编程。顺序功能图（SFC）是 1 种图形语言，便于描述程序中各个动作的时间顺序。为此，作为独立编程对象的动作被分配给步骤元素。步骤处理顺序由转换元素控制。

另请参见：

- 引用编程：顺序功能图（SFC） [▶ 580]
- TC3 用户界面文档：SFC [▶ 929]

7.5.4.1 用顺序功能图（SFC）编程

用 SFC 实现语言创建编程块

1. 在 PLC 项目的 **Solution Explorer**（解决方案资源管理器）中，选择 1 个文件夹。
2. 在上下文菜单中，选择命令 **Add > POU...**（添加 > POU……）
   ⇨ **Add POU**（添加 POU）对话框会打开。
3. 输入名称并选择实现语言顺序功能图（SFC）。
4. 点击 **Open**（打开）。

⇨ TwinCAT 会将编程块添加到 PLC 项目树中，并在编辑器中打开它。

添加步骤-转换

1. 选择 Init 步骤后的转换。
   ⇨ 该转换显示为红色。
2. 在 **SFC** 菜单或上下文菜单中，选择命令 **Insert step-transition after**（在……后插入步骤-转换）。
   ⇨ TwinCAT 会添加步骤 Step0 和转换 Trans0。
3. 选择转换 Trans0，然后，在 **SFC** 菜单或上下文菜单中，选择命令 **Insert step transition**（插入步骤转换）。

⇨ TwinCAT 会在 Trans0 之前添加转换 Trans1 和步骤 Step1。

或者，您也可以通过拖放将元素 **Step**（步骤）和 **Transition**（转换）从**Toolbox**（工具箱）视图移到图中。

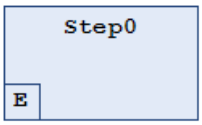
另请参见：

- TC3 用户界面文档：命令：在...后插入步骤-转换 [▶ 929]
- TC3 用户界面文档：命令：插入步骤转换 [▶ 929]

添加进入动作

1. 选择 Step0。
2. 在 **SFC** 菜单或上下文菜单中，选择命令 **Add entry action**（添加进入动作）。
   ⇨ 默认情况下，系统会出现提示，用于指定步骤动作的复制模式。这可用于指定是否将引用信息复制到现有的步骤动作对象中，或者是否应该在将来复制步骤时“嵌入”这些对象。如果选择嵌入，则在复制步骤时会创建新的步骤动作对象。复制模式在步骤属性 **Duplicate or Copy**（复制或拷贝）中进行定义。只要禁用该属性，复制的步骤就会调用与当前步骤相同的动作。

3. 在此示例中，保留 **Copy reference**（复制引用）默认设置并点击 **OK**（确定）进行确认。
   ⇨ **Add entry action**（添加进入动作）对话框会打开。
4. 输入“Step0_entry”作为名称，并选择实现语言“结构化文本（ST）”。点击 **Add**（添加）。
   ⇨ TwinCAT 在 PLC 项目树的功能块下添加动作 Step0_entry，并在编辑器中打开该动作。现在，您可以在进入动作 Step0_entry 中对指令进行编程，这些指令将在步骤 Step0 启用时执行 1 次。
5. 关闭 Step0_entry 的编辑器。
   ⇨ 现在，步骤 Step0 的左下角标有 1 个 E。双击 E 可打开编辑器。



   ⇨ 现在，进入动作 Step0_entry 出现在 **Entry action**（进入动作）下的步骤属性中。如果需要，您还可以在此处选择其他动作。
6. 选择 Step0。按 **[Ctrl] + [C]** 可复制步骤。

⇨ 添加的步骤副本包含与上文相同的进入动作。换句话说，新步骤会调用相同的动作。

SFC 编辑器选项

在 **SFC editor**（SFC 编辑器）类别的 TwinCAT 选项中，您可以指定在添加步骤动作时是否始终出现指定复制模式的提示，或者您可以指定复制模式为标准模式。

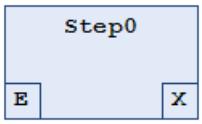
另请参见：

- 引用编程：SFC 元素属性 [▶ 593]
- TC3 用户界面文档：命令：添加输入操作 [▶ 935]
- TC3 用户界面文档：对话框选项 - SFC 编辑器 [▶ 907]

添加退出动作

1. 选择 Step0。
2. 在 **SFC** 菜单或上下文菜单中，选择命令 **Add exit action**（添加退出动作）。
   ⇨ 默认情况下，系统会出现提示，用于指定步骤动作的复制模式。请参见“添加输入选项”和“SFC 元素属性”部分的中说明。
   ⇨ **Add exit action**（添加退出动作）对话框会打开。
3. 输入“Step0_exit”作为名称，并选择实现语言“结构化文本（ST）”。点击 **Add**（添加）。
   ⇨ TwinCAT 在 PLC 项目树的功能块下添加动作 Step0_exit，并在编辑器中打开该动作。现在，您可以在退出动作 Step0_exit 中对指令进行编程，这些指令将在步骤 Step0 被禁用之前执行 1 次。
4. 关闭 Step0_exit 的编辑器。

⇨ 现在，步骤 Step0 的右下角标有 1 个 X。双击 E 可打开编辑器。



⇨ 您可以在退出动作下的步骤属性中定义退出动作。如果需要，您还可以在此处选择其他动作。

另请参见：

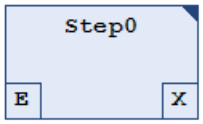
- TC3 用户界面文档：命令：添加退出操作 [▶ 936]

添加动作

1. 双击 Step0。
   ⇨ 默认情况下，系统会出现提示，用于指定步骤动作的复制模式。请参见“添加输入选项”和“SFC 元素属性”部分的中说明。**Add Action**（添加动作）对话框会打开。
2. 输入“Step0_active”作为名称，并选择实现语言“结构化文本（ST）”。点击 **Add**（添加）。
   ⇨ TwinCAT 在 PLC 项目树的功能块下添加动作 Step0_active，并在编辑器中打开该动作。现在，您可以在步骤动作 Step0_active 中对指令进行编程，只要步骤处于活动状态，这些指令就会被执行。

3. 关闭 Step0_active 的编辑器。

⇨ 现在，Step0 的右上角标有 1 个黑色三角形。



您可以在步骤动作下的步骤属性中定义动作。您还可以在此处选择其他动作。

添加替代分支

1. 选择 Step1。
2. 在 **SFC** 菜单或上下文菜单中，选择命令 **Insert branch right**（在右侧插入分支）。

   ⇨ TwinCAT 在 Step1 的右侧添加步骤 Step2。这些步骤就像双线平行分支一样链接在一起。
3. 从 2 条双线中选择 1 条。

   ⇨ 双线显示为红色。
4. 在上下文菜单或 **SFC** 菜单中，选择命令 **Alternative**（替代）。

⇨ TwinCAT 将该分支转换为替代分支。双线变为单线。

使用 **Parallel**（平行）命令可以将替代分支转换为平行分支。

另请参见：

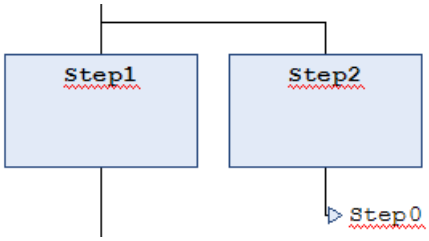
- TC3 用户界面文档：命令：在右侧插入分支 [▶ 931]
- TC3 用户界面文档：命令：替代 [▶ 930]

添加跳转

1. 选择 Step2。
2. 在 **SFC** 菜单中，选择命令 **Insert jump after**（在……后插入跳转）。

   ⇨ TwinCAT 在步骤 Step2 之后添加步骤跳转。
3. 点击跳转的跳转目标 **Step**（步骤）。

⇨ 现在，您可以手动输入跳转目标，或通过输入助手 [...] 进行选择。选择 Step0。



另请参见：

- TC3 用户界面文档：命令：在...后插入跳转 [▶ 933]

添加宏

1. 选择 Step1。
2. 在 **SFC** 菜单或上下文菜单中，选择命令 **Insert macro after**（在……后插入宏）。

   ⇨ TwinCAT 在步骤 Step1 之后添加宏 Macro0。
3. 双击元素 Macro0。

   ⇨ 在编辑器的实现部分会打开宏。标题显示名称 Macro0。
4. 在 **SFC** 菜单或上下文菜单中，选择命令 **Insert step transition**（插入步骤转换）。

   ⇨ TwinCAT 插入步骤转换组合。
5. 在 **SFC** 菜单或上下文菜单中，选择命令 **Exit macro**（退出宏）。

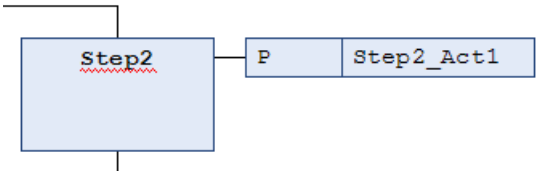
⇨ 实现部分再次显示主图。

另请参见：

- TC3 用户界面文档：命令：在...后插入宏 [▶ 934]
- TC3 用户界面文档：命令：插入步骤转换 [▶ 929]
- TC3 用户界面文档：命令：退出宏 [▶ 934]

添加关联

1. 选择 Step2。
2. 在 **SFC** 菜单或上下文菜单中，选择命令 **Insert action association**（插入动作关联）。
   ⇨ TwinCAT 在步骤 Step2 的右边添加 1 个关联。
3. 点击进入关联的左侧字段，选择限定符。
   ⇨ 您可以手动输入限定符，或通过输入助手 [...] 进行选择。选择“P”。
4. 点击进入关联的右侧字段，选择动作。
   ⇨ 您可以手动输入动作，或通过输入助手 [...] 进行选择。



另请参见：

- TC3 用户界面文档：命令：插入动作关联 [▶ 932]

使用 AnalyzeExpression 分析表达式

Tc2_System 库中的 AnalyzeExpression 功能块可对表达式进行分析。例如，在 SFC 图中，它可用于检查 SFCError 标志的结果，而该标志用于监控流程图中的超时。

另请参见：

- 文档 Tc2_System：
- 引用编程：SFC 标志 [▶ 584]

7.5.5 梯形图（LD）（测试版）

TC3.1 Build 4026 及以上的测试版本可提供新的梯形图编辑器。

梯形图编辑器是 1 种基于网络的编辑器，适用于 IEC 编程语言梯形图（LD）。它是仍在使用的 FBD/LD 编辑器的后续版本。它与 FBD/LD 编辑器的主要区别在于编辑功能有所简化。以前需要从工具箱中插入单独元素的元素变体，现在只需选择插入位置或随后切换修饰符即可进行创建。在 FBD/LD 编辑器中创建的编程块可通过菜单命令转换为梯形图格式。

另请参见：

- 引用编程：梯形图（LD）（测试版） [▶ 620] 章节
- 用户界面文档：梯形图编辑器 [▶ 964]

7.5.5.1 在梯形图编辑器中编程

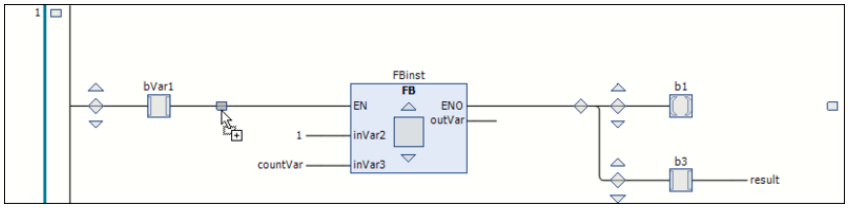
使用梯形图编程的前提条件：
您已在您的项目中使用梯形图实现语言创建了 1 个编程块。您已在梯形图编辑器中打开此方框进行编辑。

以下说明仅描述在编辑器中工作时的个别动作。没有创建特定的程序。如有需要，另请参见下一章：梯形图编辑器中的导航 [▶ 622]

在网络中插入、重新定位和修改元素

将元素插入网络：

1. 例如，在编辑器的工具箱（工具窗口）中，选择 Contact（触点）元素。
2. 使用鼠标将其拖到网络上，然后按住鼠标按钮。
   ⇨ 当您在网络中拖动时，以下符号会指明可能的插入位置：
   现有元素符号内带灰色背景的正方形
   连接线上的菱形
   指向下方或上方的三角形，用于在上方或下方插入
3. 当鼠标光标出现加号 时，您可以插入触点元素。



4. 在此位置松开鼠标按钮。

⇨ 触点已插入网络的处理线。

插入方框：

1. 在网络中添加元素 Box（方框）。
2. 将元素方框中的 3 个问号 ??? 替换为项目或库中所需方框的名称。
3. 为此，双击问号，最好通过点击 按钮使用输入助手。

⇨ 方框已插入并被赋予适当的名称。

更新方框：

例如，如果您在梯形图中调用您的项目中的功能块，您在更改基本 FB 后也必须在梯形图中进行更新。

目前还没有相应的“更新”命令。

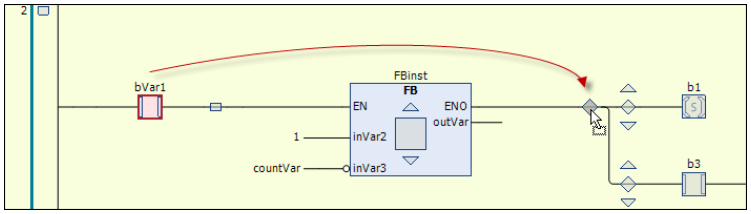
插入输入端：

1. 在方框前插入 1 个**输入端**。
2. 如要分配变量，可将元素处的 3 个问号 ??? 替换为变量名。

⇨ 如有必要，打开标准对话框 Auto Declare, Select, Reposition（自动声明、选择、重新定位）。

重新定位网络中的元素：

1. 选择 1 个元素，使其显示为红色背景。如果是方框，方框图标内的正方形必须显示为红色背景。
2. 将元素拖动到另一个可能的插入位置，直到光标符号出现正方形，然后松开鼠标按钮。



⇨ 元素会相应定位。

替换元素：

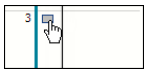
1. 如要用另一个元素替换现有元素，可将新元素拖到您想要替换的元素上。
2. 当要替换的元素本身亮起绿色插入点时，松开鼠标按钮。

⇨ 元素已替换。

添加新网络：

1. 将网络元素拖动到实现部分。

⇨ 您将在显示网络编号的左侧空白处看到矩形插入标记。新网络在所选插入点的上方插入。

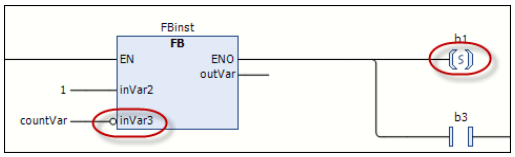


修改网络：

在带有修饰符的网络中提供元素：

1. 选择方框的输入引脚。
2. 右键点击可打开上下文菜单。
3. 选择 Negate [▶ 964]（否定）命令否定输入端，或选择 Set/Reset [▶ 965]（设置/重置）或 Edge Detect（边缘检测）命令之一（命令：边缘检测：上升沿 [▶ 965]，命令：边缘检测：下降沿 [▶ 965]）。

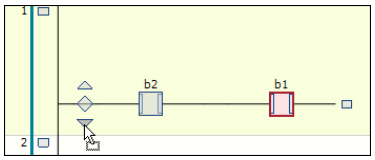
⇨ 为输入端分配相应的符号。
示例：方框上否定的输入端，设置线圈



创建和编辑线路分支

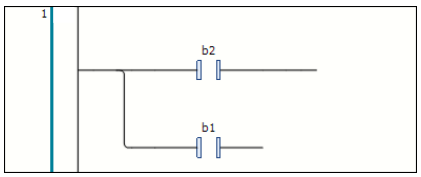
与 LD/FBD 编辑器不同，在梯形图编辑器中没有用于线路分支的元素。您只需定位或重新定位元素，即可创建和编辑分支。示例：

1. 创建以下网络：



2. 将第 2 个触点拖动到第 1 个触点前面标有向下箭头的插入位置。

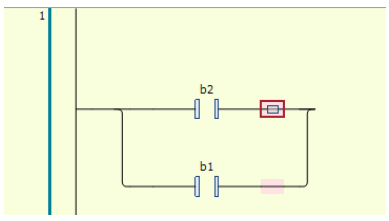
⇨ 这会形成 1 个向后方开放的平行分支。每个分支包含 1 个触点。



如要从开放线路分支创建 1 个封闭线路分支，即对 1 个 OR 结构编程，可执行以下步骤：

1. 选择触点元素后面的 2 个分支（多项选择）。
2. 线上带红色背景的小正方形表示选择。
3. 然后选择 Close parallel branch [▶ 965]（关闭平行分支）命令。

⇨ 2 条平行分支在末端打开，成为 1 个封闭分支。



有 2 种方法可以重新打开封闭分支：

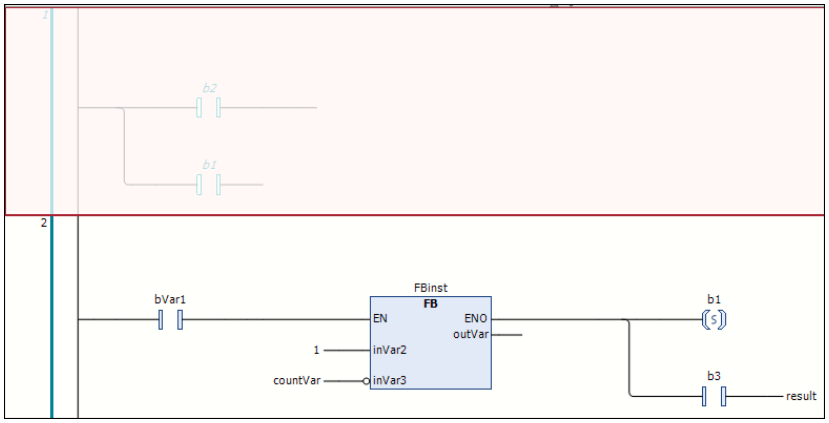
- 您选择 2 个分支中的 1 个，然后将选择框拖到另一个分支的选择框上。
- 选择 2 个分支，然后选择命令 Open parallel branch [▶ 965]（打开平行分支）。

如果封闭线路分支有多个平行分支，则 **Open parallel branch**（打开平行分支）命令将始终打开所有分支。

注释掉网络

1. 选择 1 个网络，使其完全显示为红色背景。
2. 从上下文菜单中，选择 Commented out [▶ 964]（注释掉）命令。

⇨ 网络内容以灰色显示，文本以斜体显示。在处理过程中，不会考虑网络。

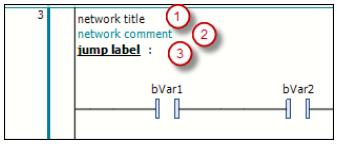


添加网络标题、网络注释、跳转标签

您可以为网络添加网络标题 (1)、网络注释 (2) 和/或跳转标签 (3)。

1. 为此，可点击进入网络左上角的第 1 行、第 2 行或第 3 行。在插入跳转 [▶ 968]元素时，跳转标签可作为目标。

⇨ 网络标题和网络注释的显示在 TwinCAT 选项的梯形图 [▶ 912]类别中进行定义。



7.6 创建引用任务

您可以在 1 个 PLC 项目中集成 1 个或多个任务，以定义程序块的处理。为了能够使用任务，您必须创建 1 个任务引用。

7.6.1 对象引用任务

符号：

通过任务引用，您可以定义要在任务中执行的程序块。

在创建标准 PLC 项目时，系统会自动创建 1 个任务引用 **PlcTask**，它定义了 MAIN 程序块的处理。

创建对象引用任务

1. 在 PLC 项目树的 Solution Explorer（解决方案资源管理器）中，选择 PLC project（PLC 项目）（<Project name> 项目）。
2. 在上下文菜单中，选择命令 **Add** > **Referenced Task**（添加 > 引用任务）。

⇨ 对话框 **Add Referenced Task**（添加引用任务）会打开。在 **Available Tasks**（可用任务）列表框中显示了现有任务。列表框 **Used Tasks**（已用任务）显示了已引用的任务。

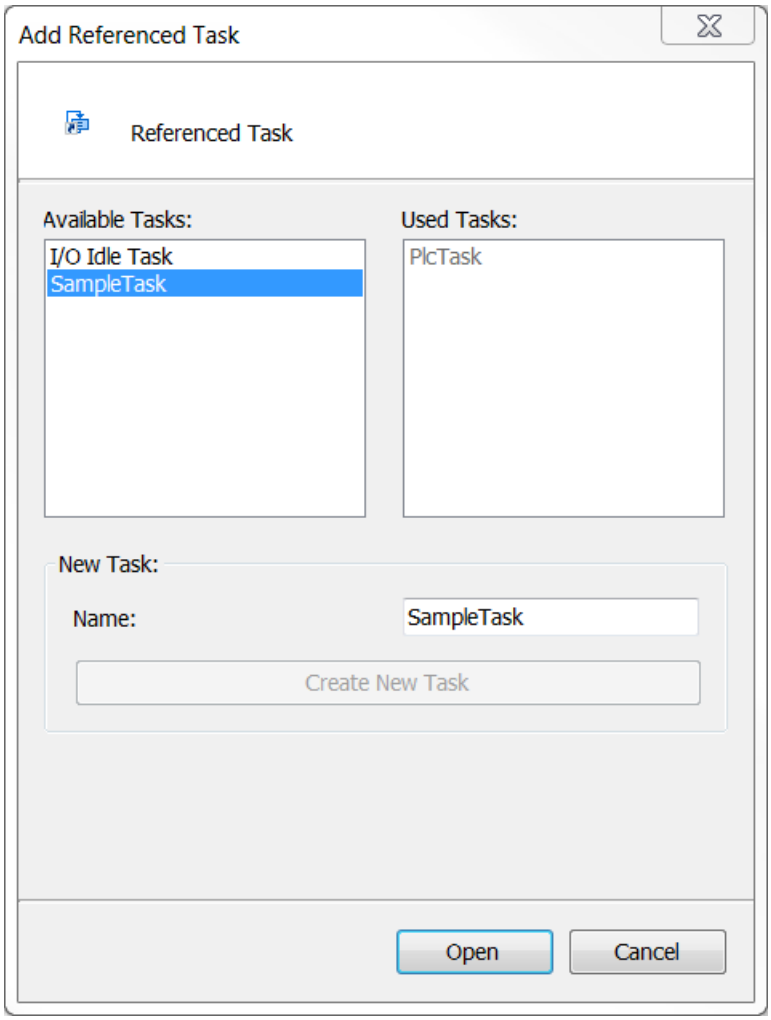
3. 如要生成并集成 1 个新任务，可在 **Name**（名称）字段中输入名称并点击 **Create New Task**（创建新任务）。

   ⇨ 在 **Available Tasks**（可用任务）列表框中显示了新任务。同时，新任务会出现在 **Task**（任务）节点下的 **SYSTEM** 部分中的 TwinCAT 项目树中。

4. 点击 **Open**（打开）。
5. 如果您想要集成的任务已经存在，则可在 **Available Tasks**（可用任务）列表框中选择该任务并点击 **Open**（打开）。

⇨ 任务引用已添加到 PLC 项目树中。当您再次打开对话框 **Add Referenced Tasks**（添加引用任务）时，任务将显示在 **Used Tasks**（已用任务）列表框中。

对话框添加引用任务



可用任务	在 PLC 项目中未引用的任务。
已用任务	在 PLC 项目中引用的任务。

新任务

名称	要创建的新任务的名称
创建新任务	创建 1 个新任务

在任务中指定程序块

1. 在 PLC 项目树中，选择任务引用。在上下文菜单中，选择命令 **Add** > **Existing Item…**（添加 > 现有项目……）

   ⇨ **Input Assistant**（输入助手）对话框打开。

2. 选择要在任务中执行的程序块，并点击 **OK**（确定），确认对话框。
3. 或者，您也可以使用鼠标将 PLC 项目树中的程序块直接拖到任务引用中。

   ⇨ 该程序块会在任务引用下的 PLC 项目树中出现。
4. 如要查看任务配置，可双击 PLC 项目树中的任务引用。在 Solution Explorer（解决方案资源管理器）中，在 **SYSTEM** > **Tasks**（SYSTEM > 任务）部分的 TwinCAT 项目树中已启用相应的任务。再次双击任务，可在编辑器中打开任务配置。

⇨ 在 PLC 项目树中，任务调用的程序块会在任务引用下显示。该任务可在任务编辑器的 **SYSTEM** 部分中进行配置。

关于多任务系统的重要说明

多核系统能够独立执行多个命令，这可能会导致优先级反转。

示例：

低优先级任务调用的程序块可被分配给 1 个独立的核心。在这种情况下，优先级较低的任务可能会在优先级较高的任务之前执行完毕。

7.7 创建类图

有关更多信息，请参见“”部分中的文档“TF1910 TC3 UML”。

7.8 为在线更改配置内存保留

您可以为功能块的在线更改配置内存保留。这意味着，只要内存保留足够大，在随后的在线更改中对功能块的声明进行更改后，功能块的实例无需复制到新的存储空间。这主要是指在线更改，即在功能块中添加 1 个或多个新的变量。如果功能块实例不必因为内存保留而复制到新的存储空间，那么在线更改的速度就会更快，出现的问题也会更少。当内存保留用完时，在执行在线更改之前会出现 1 条信息。

首次下载前的内存保留配置

首次将 PLC 项目下载到控制器之前，最好先配置功能块的内存保留。如果仅在 PLC 项目已在控制器上时配置内存保留，则需要进行复杂的在线更改。

替代配置选项

除了 **Online Change Memory Reserve Settings**（在线更改内存保留设置）窗口之外，还可在属性窗口中配置每个功能块的内存保留。为此，务必在项目树中选择功能块。

为在线更改的功能块配置内存保留

✓ 在未来，将对项目的功能块进行重大更改。这需要在在线更改时将功能块实例复制到其他存储位置。

✓ 理想情况下，打开的项目还不在控制器上。

1. 从 **PLC** 菜单中，选择 **Window** > **Online Change Memory Reserve Settings**（窗口 > 在线更改内存保留设置）。

   ⇨ **Online Change Memory Reserve**（在线更改内存保留）视图会打开。
2. 从选择列表中选择 PLC 项目。
3. 从 **Build**（构建）菜单中，选择 **Build**（构建）。
4. 点击 **Browse application**（浏览应用程序）按钮。
5. 在“Function blocks”（功能块）区域中，选择“All”（全部）条目。

   ⇨ 在视图中会显示 PLC 项目的所有功能块。
6. 选择您想要配置内存保留的功能块。

   ⇨ 如果应用程序尚未安装在控制器上，则可以编辑输入字段“Memory reserve (in bytes)”（内存保留（以字节为单位））。
7. 如果控制器上已有 PLC 项目，可点击“Allow editing”（允许编辑）区域中的 **Allow**（允许）按钮。请注意，如果您更改控制器上已存在的 PLC 项目的内存保留，则必须在内存中复制所有受影响功能块的实例。

8. 输入以字节为单位的内存保留大小，并点击 **Apply to selection**（应用于选择）。
   ⇨ 在 **Memory reserve**（内存保留）字段的表格中会显示输入的字节数。
9. 从 **Build**（构建）菜单中，选择 **Build**（构建）。
10. 点击 **Browse application**（浏览应用程序）按钮。
    ⇨ 在已配置功能块的功能块列表中，“大小”、“实例数”、“所有实例的额外内存”和“剩余内存保留大小”等信息均会更新。
11. 将 PLC 项目加载到控制器上。

⇨ 功能块实例会占用当前所需的内存和额外的内存保留。因此，未来对功能块的重大更改可以通过在线更改加载到控制器上，而无需在内存中重新复制功能块的所有实例。

另请参见：

- 文档 TC3 用户界面：参考用户界面 > PLC > 窗口 > 命令在线更改内存保留设置 [▶ 881]
- 执行在线更改 [▶ 231]

7.9 调用具有外部实现的功能块、函数或方法

仅适用于特殊情况

只有在特殊的系列情况中才能使用该功能。通常情况下，您可以忽略外部实现选项。

运行时系统可能包含功能块、函数或方法的实现，例如，来自 1 个库。如果您在 PLC 项目中创建了 1 个同名的 POU，其属性为 **External implementation without implementation**（没有实现的外部实现），那么，您可以执行现有实现。确保您使用外部功能块来声明局部变量。外部函数或方法可能不包含局部变量。

在项目下载过程中，TwinCAT 会在运行时系统中为每个外部 POU 找到相应的实现并进行链接。

创建具有外部实现的 POU

1. 在 PLC 项目树的上下文菜单中，选择命令 **Add** > **POU...**（添加 > POU……）
2. 启用功能块或函数，并输入相应的运行时系统实现的名称。选择 **Open**（打开），退出对话框。
   ⇨ 名称为运行时系统 POU 的 POU 已创建。
3. 选择该 POU 并启用 **Properties**（属性）视图。
4. 启用选项 **External implementation**（外部实现）（在运行时系统中进行后期链接）。

⇨ POU 已声明，您可以实现对 POU 的调用。

创建具有外部实现的方法

1. 在 **Solution Explorer**（解决方案资源管理器）的 PLC 项目树中，选择 1 个功能块。
2. 在上下文菜单中，选择命令 **Add** > **Method**（添加 > 方法），并输入相应的运行时系统实现的名称。选择 **Open**（打开），退出对话框。
   ⇨ 方法已创建。
3. 选择该方法并启用 **Properties**（属性）视图。
4. 启用选项 **External implementation**（外部实现）（在运行时系统中进行后期链接）。

⇨ 方法已声明，您可以实现对方法的调用。

7.10 使用输入向导

TwinCAT 提供各种功能和向导，便于在编程过程中输入代码。

对话框输入助手

对话框提供所有编程元素，您可以将其添加到当前的光标位置。使用 **Edit**（编辑）菜单或上下文菜单中的命令 **Input Assistant**（输入助手），或者使用键盘快捷键 **[F2]**，打开 **Input Assistant**（输入助手）对话框。

另请参见：

- TC3 用户界面文档：命令：输入助手 [▶ 810]

自动声明对话框

对话框支持声明变量。使用 **Edit**（编辑）菜单中的 **Auto Declare**（自动声明）命令或通过上下文菜单，打开 **Auto Declare**（自动声明）对话框。

另请参见：

- TC3 用户界面文档：命令：自动声明 [▶ 811]

列出组件

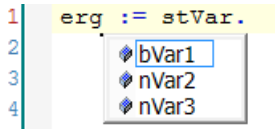
List components（列出组件）功能是文本编辑器中的 1 项输入支持功能，可帮助输入有效的标识符。通过以下方法可以启用该功能：

1. 在 **Tools**（工具）菜单中选择命令 **Options**（选项），然后选择类别 **TwinCAT > PLC Environment > Smart coding**（TwinCAT > PLC 环境 > 智能编码）。
2. 启用选项 **List components after typing a dot (.)**（键入点（.）后列出组件）。
   - 如果您输入的是点而不是全局变量，则会出现 1 个包含所有可用全局变量的选择列表。双击选择列表中的 1 个变量，或按 **[Enter]** 键，在点后添加所选变量。
   - 如果您输入的是点而不是全局变量，或者您在功能块实例变量或结构变量后输入点，则 TwinCAT 会显示 1 个包含所有全局变量、功能块的所有输入和输出变量或所有结构组件的相应的选择列表。
     双击选择列表中的 1 个变量，或按 **[Enter]** 键，在点后添加所选变量。
     如果您还想查看功能块实例的局部变量，可在用于**智能编码**的 TwinCAT 选项中启用选项 **List all instance variables in the input assistant**（在输入助手中列出所有实例变量）。
   - 如果已对选择列表进行组件访问（带点），则在下一次组件访问时将会预选上一次所选的条目。
   - 如果您输入任意字符串，然后按 **[Ctrl] + [space bar]**，就会出现 1 个包含所有可用 POU 和全局变量的选择列表。该列表中以之前输入的字符串开头的第 1 个元素会被自动选中，通过双击或按 **[Enter]** 键，您可以在编辑器中输入该元素。
     与输入的字符串匹配的元素会在选择列表中以黄色突出显示。当输入的字符串发生变化时，显示的选择列表也会更新。
   - 在 ST 编辑器中，您可以根据有效性区域对显示的选择列表进行筛选：
     根据每种情况下显示的选择列表的不同，您可以使用 **[right arrow]** 和 **[left arrow]** 键在下列选择列表之间切换：
     - 所有条目
     - 关键字
     - 全局声明
     - 局部声明
   - 如果您调用 1 个功能块、1 个方法或 1 个函数，并输入左括号用于输入 POU 参数，则 TwinCAT 会显示 1 个工具提示。该工具提示包含在 POU 中声明的参数信息。在您点击鼠标或将焦点置于当前视图之外来关闭工具提示之前，该工具提示始终可见。如果您意外关闭工具提示，您可以用 **[Ctrl] + [shift key] + [space bar]** 重新打开该工具提示。

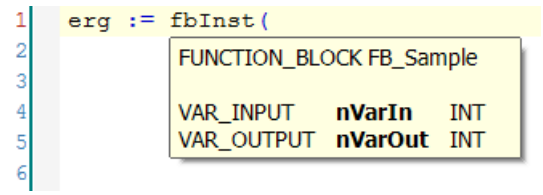
使用编译指示属性“隐藏”可将变量排除在 **List components**（列出组件）功能之外。（属性 “hide” [▶ 744]）

示例：

输入结构变量：



调用功能块：



快捷方式模式

快捷方式模式可以在声明编辑器和文本编辑器中输入变量声明的快捷方式，在文本编辑器中可以进行变量声明。您可以使用快捷键 [Ctrl] + [Enter] 结束声明行，进而可以启用该模式。

TwinCAT 支持以下快捷方式：

- 除 1 行中最后一个标识符外的所有标识符都可以成为声明的变量标识符。
- 声明的数据类型由该行中最后一个标识符决定。以下规则适用：
  - B 或 BOOL 的结果是 BOOL
  - I 或 INT 的结果是 INT
  - R 或 REAL 的结果是 REAL
  - S 或 STRING 的结果是 STRING
- 如果没有根据这些规则指定数据类型，则数据类型将被自动设置为 BOOL，最后一个标识符不用作数据类型（请参见示例 1）。
- 输入的每个常量都会被解释为初始化或字符串长度声明，具体取决于声明类型（请参见示例 2 和 3）。
- 地址（例如在 %MD12 中）会自动使用 AT 属性进行扩展（请参见示例 4）
- 分号“;”之后的文本被解释为注释（请参见示例 3）。
- 行中的所有其他字符都会被忽略（请参见示例 5 中的感叹号）。

示例：

示例	快捷方式	结果声明
1	bA	bA: BOOL;
2	nA nB I 2	nA, nB: INT := 2;
3	sC S 2; C 字符串	sC: STRING(2); // C 字符串
4	X %MD12 R 5	X AT %MD12: REAL := 5.0;
5	bE !	bE: BOOL;

智能标签功能

TC3.1 Build 4026 及以上可用

智能标签通过直接在编程元素上提供合适的选择命令，为程序代码的创建提供便利。如果您将光标置于可使用智能标签功能的编程元素上，则会出现符号  。点击  将会显示您可以选择的命令。可用的智能标签：

- 对于 ST 编辑器的实现部分中的未声明变量，智能标签功能提供了命令 **Auto Declare**（自动声明）。

另请参见：

- 声明变量 [▶ 60]
- TC3 用户界面文档：对话框选项 - 智能编码 [▶ 911]

7.11 使用编译指示

TwinCAT 中的编译指示

编译指示是应用程序源代码中用大括号括起来的文本。编译指示可用于在代码中插入特殊指令，编译器可以对其进行评估。这使得 1 个编译指示可以影响 1 个或多个变量在预编译或编译（代码生成）方面的属性。编译器不知道的编译指示会像注释一样被读取。

编译指示的指令字符串也可以是多行的。有关语法的详细信息，请参见关于各个编译指示的描述。

不同的效果有不同的编译指示：变量的初始化、变量的监控、编译过程中的强制消息输出、变量在特定条件下的行为等。

务必要区分大小写。

示例：

```
{warning 'This is not allowed'}
{attribute 'obsolete' := 'datatype FB_Sample not valid!'}
```

可能的插入位置

TwinCAT 中的编译指示并不是 C 预处理器指令的一对一实现。编译指示必须像标准指令一样定位。在表达式中不能使用编译指示。

您可以在以下位置插入可供编译器评估的编译指示：

- 在编程块的声明部分：
  - 在文本声明编辑器中，您可以直接以行的形式输入编译指示，它可以在编程块的开头，也可以在变量声明之前。
  - 在表格编辑器中，您可以在 **Attributes**（属性）对话框中添加编译指示，它必须位于第 1 行声明的上方。双击 **Attributes**（属性）列。
- 在全局变量列表中
- 在编程块的实现部分：
  - 务必将编译指示放在“指令位置”，即在编程块开头的单独 1 行中，或在“;”或 END_IF、END_WHILE 等之后。
  - FBD/LD/IL 编辑器：在 FBD/LD/IL 编辑器的网络中像输入标签一样输入编译指示：
    为此，可选择命令 **FBD/LD/IL > Insert label**（FBD/LD/IL > 插入标签），然后，使用相应的编译指示指令替换标签文本字段中的标准文本 **Label:**（标签：）。如果您想在标签之外使用编译指示，可首先输入编译指示，然后再输入标签。

条件编译指示的不正确和正确定位：

不正确：	正确：

{IF defined(abc)}
IF x = abc THEN
{ELSE}
IF x = 12 THEN
{END_IF}
y := {IF defined(cde)} 12;
{ELSE} 13;
{END_IF}
END_IF
	{IF defined(abc)}
	IF x = abc THEN
	{IF defined(cde)}
	y := 12;
	{ELSE}
	y := 13;
	{END_IF}
	END_IF
	{ELSE}
	IF x = 12 THEN
	{IF defined(cde)}
	y := 12;
	{ELSE}
	y := 13;
	{END_IF}
	END_IF
	{END_IF}

在 POU **Properties**（属性）的 **Advanced**（高级）类别中，您可以输入可在编译指示中查询的 **Defines**（定义）。

范围：

根据编译指示的类型和内容，它会对以下内容产生影响：

- 后续声明
- 特别是后续指令
- 所有后续指令，直到它被相应的编译指示取消。
- 所有后续指令，直到使用其他参数执行相同的编译指示或代码结束。这里的“代码”是指：声明部分、实现部分、全局变量列表、类型声明。因此，1 个单独出现在声明部分第 1 行且未被另一个编译指示取代或取消的编译指示对整个对象都是有效的。

编译指示类别

TwinCAT 编译指示分为以下几类：

- 属性编译指示：影响编译和预编译（属性编译指示 [▶ 734]）
- 消息编译指示：在编译过程中输出由用户定义的消息（消息编译指示 [▶ 733]）
- 条件编译指示：影响代码生成（条件编译指示 [▶ 775]）
- 用户定义的编译指示（用户定义的属性 [▶ 734]）

另请参见：

- 使用自动声明对话框 [▶ 65]

7.12 管理文本列表中的文本

文本列表可用于提供多种语言的可视化文本。您可以输入 Unicode 格式的文本，以便使用所有语言和字符。文本列表可以导出和导入，以便在当前项目之外编译文本。

TwinCAT 对静态文本和动态文本进行区分，前者在“GlobalTextList”对象中进行管理，后者在“Textlist”类型的对象中进行管理。

静态文本是可视化中的文本，只能在运行时改变语言。文本 ID 保持不变。

动态文本可通过包含文本 ID 的 IEC 变量进行控制。这样您就可以在运行时在可视化元素中显示不同的文本。例如，您可以对文本字段进行配置，使其输出带有错误编号的错误文本。

这 2 种类型的文本列表都包含 1 个带有文本条目的表格。条目由用于识别的 ID、源文本及其翻译组成。在文本列表或全局文本列表中，您可以将源文本翻译成许多不同的语言。翻译是可视化中的语言选择和语言切换的基础。

文本列表文件的目录

在项目属性的 **Visualization**（可视化）类别中可以指定目录，它可以提供可视化文本列表。

另请参见：

- 使用输入向导 [▶ 123]
- TC3 用户界面文档：文本列表 [▶ 970]

添加语言和翻译文本

✓ 1 个带有文本列表或全局文本列表的项目已打开。

1. 在 PLC 项目树中，双击 **Textlist** 或 **GlobalTextList** 类型的对象。
   ⇨ 在菜单栏中会出现 **Textlist** 菜单，在编辑器中会打开文本列表。
2. 在 **Textlist** 菜单或上下文菜单中，选择命令 **Add Language**（添加语言）。
3. 输入语言名称，例如，“en-US”。选择 **OK**（确定），退出对话框。

⇨ 1 个标题为 **en-US** 的列会出现。

4. 在此列中输入源文本的译文。

⇨ 您可以添加许多不同的语言。

鉴于 **Default**（默认值）列必须包含文本，您不必在语言列中输入译文。如果某个文本条目没有找到译文，系统会自动显示默认文本。

另请参见：

- TC3 用户界面文档：命令：添加语言 [▶ 970]

导出文本列表

✓ 1 个带有文本列表或全局文本列表的项目已打开。

1. 在 PLC 项目树中，双击 **Textlist** 或 **GlobalTextList** 类型的对象。
   ⇨ 在菜单栏中会出现 **Textlist** 菜单，在编辑器中会打开文本列表。
2. 在 **Textlist** 菜单或上下文菜单中，选择命令 **Import/Export Text Lists**（导入/导出文本列表）。
   ⇨ **Import/Export**（导入/导出）对话框打开。
3. 在 **Choose export file**（选择导出文件）下，点击 …，选择目录并输入文件名，例如，“Text_lists_exported”。
4. 启用选项 **Export**（导出）。
5. 选择 **OK**（确定），退出 **Import/Export**（导入/导出）对话框。

⇨ TwinCAT 将项目中所有文本列表的文本列表条目导出到 .csv 文件中。表格包含 1 列文本列表名称。

您可以使用 **Export All**（全部导出）命令，将文本列表条目导出为 .txt 文件。为每个文本列表创建 1 个文件。在项目属性中可以定义目录，它可以自动存储导出文件。

示例：

文件 Text_lists_exported 的内容

```
TextList        Id    Default         en-us
TextList_A      A     Information A   Information A_en
TextList_A      B     Information B   Information B_en: Ok
TextList_A      C     Information C   Information C_en
AlarmGroup      1     Warnung 1
AlarmGroup      2     Warnung 2
GlobalTextList        Information G   Information G_en
GlobalTextList        Information H   Information H_en
GlobalTextList        Umschalten      Switch
GlobalTextList        Zähler: %i      Counter: %i
```

另请参见：

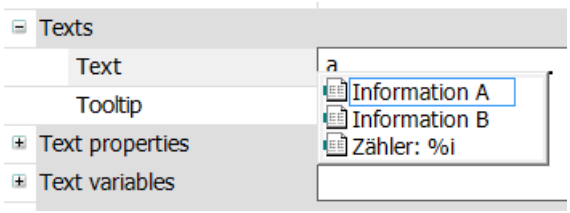
- TC3 用户界面文档：命令：导入/导出文本列表 [▶ 971]

为输入助手提供导出文件

✓ 您已通过命令 **Import/Export Text Lists**（导入/导出文本列表）创建 1 个文件，例如，Text_lists_exported。它包含项目文本列表的文本。

1. 在 **Tools**（工具）菜单中，点击命令 **Options**（选项），类别为 **TwinCAT > PLC Environment > Visualization**（TwinCAT > PLC 环境 > 可视化），选项卡为 **File options**（文件选项）。
2. 在**文本“IntelliSensed”的文本文件**中，点击 … 并选择 1 个文件，例如，Text_lists_exported。选择 **OK**（确定），退出对话框。

⇨ 如果您在可视化元素的 **Texts**（文本）属性下输入静态文本，TwinCAT 会在您输入第 1 个字母后立即提供文件中包含的源文本供您选择。



另请参见：

- 使用输入向导 [▶ 123]

导入包含文本列表条目的文件

可导入文件是指格式为 .csv 的文件。第 1 行是标题，例如，“TextList Id Default en_US”。其他行包含文本列表条目。当您将项目的文本列表导出到文件时，您就会得到这样 1 个文件。您可以在 TwinCAT 外部编辑文本列表条目，然后重新导入文件。在导入过程中，TwinCAT 会对 GlobalTextList 和动态文本列表的文本列表条目进行不同的处理。

GlobalTextList：

- 如果 ID 未知，TwinCAT 不会创建和新建文本列表条目。
- TwinCAT 会忽略对 ID 或源文本有影响的修改。
- TwinCAT 接受对翻译的更改。

Textlist：

- 对于新的 ID，TwinCAT 会用相应的文本列表条目来补充相应的文本列表。
- 对于源文本不同的现有 ID，文本列表中的源文本将被文件中包含的源文本覆盖。
- TwinCAT 接受对翻译的更改。

导入文件：

✓ 1 个带有文本列表或全局文本列表的项目已打开。

1. 在 PLC 项目树中，双击 **Textlist** 或 **GlobalTextList** 类型的对象。
   ⇨ 在菜单栏中会出现 **Textlist** 菜单，在编辑器中会打开文本列表。
2. 在 **Textlist** 菜单或上下文菜单中，选择命令 **Import/Export Text Lists**（导入/导出文本列表）。
   ⇨ **Import/Export**（导入/导出）对话框打开。
3. 在 **Choose file to compare or to import**（选择要比较或导入的文件）下，点击 … 并选择目录和文件名，例如，“Text_lists_corrected.csv”。
4. 启用选项 **Import**（导入）。
5. 选择 **OK**（确定），退出对话框。

⇨ TwinCAT 将文件中的文本列表条目导入到相应的文本列表中。

示例：

文件 Text_lists_corrected.csv 的内容

```
TextList        Id    Default             en-us
TextList_A      A     Information A       Information A_en
TextList_A      B     Information B: Ok   Information B_en: Ok
TextList_A      C     Information C       Information C_en
TextList_A      D     Information D       Information D_en
AlarmGroup      1     Warnung 1           Warning 1
AlarmGroup      2     Warnung 2
GlobalTextList        Information G       Information G_en: Ok
GlobalTextList        Information HH      Information H_en
GlobalTextList        Information I       Information I_en
GlobalTextList        Umschalten          Switch
GlobalTextList        Zähler: %i          Counter: %i
```

这些内容会被传输到项目中相应的文本列表中。

```
TextList        Id   Default             en-us
TextList_A      A    Information A       Information A_en
TextList_A      B    Information B: Ok   Information B_en:  Ok
TextList_A      C    Information C       Information C_en
TextList_A      D    Information D       Information D_en
AlarmGroup      1    Warnung 1           Warning 1
AlarmGroup      2    Warnung 2
GlobalTextList       Information G       Information G_en: Ok
GlobalTextList       Information H       Information H_en
GlobalTextList       Umschalten          Switch
GlobalTextList       Zähler:  %i         Counter:  %i
```

比较文本列表与文件和导出差异

✓ 1 个带有文本列表或全局文本列表的项目已打开。

1. 在 PLC 项目树中，双击 **TextList** 或 **GlobalTextList** 类型的对象。
   ⇨ 在菜单栏中会出现 **Textlist** 菜单，在编辑器中会打开文本列表。
2. 在 **Textlist** 菜单或上下文菜单中，选择命令 **Import/Export Text Lists**（导入/导出文本列表）。
   ⇨ **Import/Export**（导入/导出）对话框打开。
3. 在 **Choose file to compare or to import**（选择要比较或导入的文件）下，点击 ... 并选择比较文件的目录和文件名，例如，“Text_lists_corrected.csv”。
4. 在 **Choose export file**（选择导出文件）下，点击 ...，选择目录并输入包含比较结果的文件的名称。
5. 启用选项 **Export only text differences**（仅导出文本差异）。
6. 选择 **OK**（确定），退出对话框。

⇨ TwinCAT 读取导入文件并比较具有相同 ID 的文本列表条目。如果存在差异，TwinCAT 会将文本列表的文本列表条目写入导出文件。对于全局文本列表，TwinCAT 会将译文与源文本进行比较。如果存在差异，TwinCAT 会将文本列表条目写入导出文件。

示例：

```
TextList        Id   Default          en-us
TextList_A      B    Information B    Information B_en: Ok
AlarmGroup      1    Warnung 1        Warning 1
GlobalTextList       Information G    Information G_en
GlobalTextList       Information H    Information H_en
```

另请参见：

- TC3 用户界面文档：命令：导入/导出文本列表 [▶ 971]

7.12.1 管理全局文本列表中的静态文本

全局文本列表是项目中文本的中心位置，这些文本将在可视化中输出。

如果在可视化元素中首次出现文本，TwinCAT 会自动创建 1 个全局文本列表。它作为 1 个对象被添加到 PLC 项目树中，并且只存在 1 次。您双击编辑器中的对象，即可打开全局文本列表。

全局文本列表包含 1 个表格，该表格将显示您在项目可视化中编写的所有静态文本。如果您在 **Texts**（文本）属性下的可视化元素中进一步写入文本，TwinCAT 会自动修正该表格。从 0 开始，TwinCAT 会将 ID 作为连续的整数进行分配。

您可以根据可视化的静态文本来检查、更新和同步全局文本列表。此处，选项仅限于编辑和翻译现有文本，即您无法编写新文本。您不能直接在表格中编辑源文本或 ID，不过，您可以创建替换文件并将其导入，从而用另一个文本替换源文本。菜单命令可用于此目的。

TwinCAT 提供以下用于合并 GlobalTextList 的命令：

- 检查可视化文本 ID
- 更新可视化文本 ID
- 删除未使用的文本列表记录

另请参见：

- TC3 用户界面文档：文本列表 [▶ 970]

全局文本列表的结构

符号： 

GlobalTextList

ID	Default	en-us
0	Information G	Information G en
1	Information H	Information H en
2	Umschalten	Switch
3	Zähler: %i	Counter: %i

ID	唯一文本标识符
标准	源文本作为字符串，最多包含 1 个格式规范，例如，信息 A：%i 选项。如果语言列中没有写入译文，TwinCAT 将使用该文本。 双击字段可编辑文本。
该表格包含您可能已经添加的许多不同的语言列。语言列使用语言代码标识，该代码是您在使用 Add Language（添加语言）命令创建该列时指定的。	
<language code>	语言代码形式的语言名称，例如，en-US。该列包含在 Default（默认值）下所编写文本的译文。 如果在可视化管理器中选择语言代码，则可视化将在操作过程中发布译文。在操作过程中，可视化可根据用户的要求切换到另一种语言。 双击字段可编辑文本。

配置包含静态文本的可视化元素

GlobalTextList 中的文本可能包含格式规范。

✓ 1 个可视化项目已打开。**GlobalTextList** 对象包含在项目可视化中定义的文本。

1. 在 PLC 项目树中，双击可视化。
   ⇨ 编辑器会打开。
2. 选择 1 个具有 **Text**（文本）属性的元素，例如，文本字段。
3. 在 **Text**（文本）属性中输入文本，例如，“静态信息 A”。

⇨ TwinCAT 用新文本扩展 POU 视图中的全局文本列表。

检查全局文本列表

✓ 1 个可视化项目已打开。**GlobalTextList** 对象包含在项目可视化中定义的文本。

1. 在 PLC 项目树中，双击 **GlobalTextList** 对象。
   ⇨ 包含静态文本作为列表条目的表格会打开。
2. 在编辑器的 **Textlist** 菜单或上下文菜单中，选择命令 **Check Visualization Text Ids**（检查可视化文本 ID）。

⇨ 如果文本列表中的源文本与通过 ID 识别的静态文本不同，TwinCAT 将给出提示。全局文本列表中的源文本与具有相同 ID 的可视化文本不匹配。

另请参见：

- TC3 用户界面文档：命令：检查图形文本 ID [▶ 973]

更新全局文本列表中的 ID

✓ 1 个可视化项目已打开。**GlobalTextList** 对象包含在项目可视化中定义的文本。

1. 在 PLC 项目树中，双击 **GlobalTextList** 对象。
   ⇨ 包含静态文本作为列表条目的表格会打开。
2. 在编辑器的 **Textlist** 菜单或上下文菜单中，选择命令 **Update Visualization Text Ids**（更新可视化文本 ID）。

⇨ 如果静态文本属性中的文本与项目可视化中的源文本不匹配，TwinCAT 会补充全局文本列表。

另请参见：

- TC3 用户界面文档：命令：更新可视化文本 ID [▶ 973]

从全局文本列表中删除 ID

✓ 1 个可视化项目已打开。GlobalTextList 对象包含在项目可视化中定义的文本。

1. 在 PLC 项目树中，双击 GlobalTextList 对象。
   ⇨ 1 个带有文本的表格已打开。
2. 在编辑器的 Textlist 菜单或上下文菜单中，选择命令 Remove Unused Text List Records（删除未使用的文本列表记录）。

⇨ TwinCAT 会删除文本列表条目，这些条目的 ID 未在项目可视化中引用。

另请参见：

- TC3 用户界面文档：命令：删除未使用的文本列表记录 [▶ 973]

使用替换文件编辑全局文本列表

替换文件的格式为 .csv。第 1 行是标题：defaultold defaultnew REPLACE。其他行包含旧的源代码文本、新的源代码文本和命令 REPLACE。允许使用制表符、逗号或分号作为分隔符。文件中不允许混合使用分隔符。

示例（制表符作为分隔符）：

```
defaultalt     defaultneu      REPLACE
Information A  Information A1  REPLACE
```

如果您导入替换文件，TwinCAT 会逐行处理该文件，并在 GlobalTextList 中实现指定的替换。此外，TwinCAT 还会用替换文本在可视化中替换之前的文本。如果替换文本已经作为静态文本提供，TwinCAT 会识别这一点，协调静态文本并仅留下 1 个文本列表条目。

✓ 1 个带有文本列表或全局文本列表的项目已打开。

1. 在 PLC 项目树中，双击 GlobalTextList 对象。
   ⇨ 该对象会打开。
2. 在编辑器的 Textlist 菜单或上下文菜单中，选择命令 Import/Export Text Lists（导入/导出文本列表）。
   ⇨ Import/Export（导入/导出）对话框打开。
3. 在 Choose file to compare or to import（选择要比较或导入的文件）下，点击 … 并选择目录和文件名，例如，“ReplaceGlobalTextList.csv”。
4. 启用选项 Import replacement file（导入替换文件）。
5. 选择 OK（确定），退出对话框。

⇨ 文本列表和可视化中的文本会被替换。

示例：

全局文本列表包含以下源文本：

```
GlobalTextList  Counter:%i
GlobalTextList  Information A
GlobalTextList  Information Aa
GlobalTextList  Umschalten
GlobalTextList  Zähler:%i
```

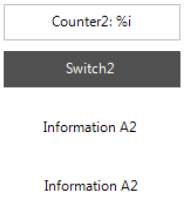
替换文件包含以下替换内容：

```
defaultalt       defaultneu      REPLACE
Counter:%i       Counter2:%i     REPLACE
Information A    Information A2  REPLACE
Information Aa   Information A2  REPLACE
Umschalten       Switch          REPLACE
Zähler:%i        Counter2:%i     REPLACE
```

TwinCAT 会检测到重复的文本列表条目并删除其中 1 个。这样，全局文本列表会包含以下条目：

5	Switch2
4	Counter2: %i
3	Information A2

可视化中的文本已替换。



另请参见：

- TC3 用户界面文档：命令：导入/导出文本列表 [▶ 971]

7.12.2 管理文本列表中的动态文本

在动态文本的文本列表中，您可以管理、创建和翻译文本。在 **Dynamic texts**（动态文本）属性中的可视化元素中，您在此处编写的文本可供选择。在操作过程中，可视化会以所选语言动态输出这些文本。

在 PLC 项目树中，创建文本列表作为对象。文本列表包含 1 个带有文本列表条目的表格，您可以对其进行编辑和扩展。文本列表条目由用于识别的 ID、源文本及其翻译组成。您可以在文本列表中添加新的文本列表条目。菜单命令可用于此目的。

另请参见：

- TC3 用户界面文档：文本列表 [▶ 970]

创建对象文本列表

1. 在 PLC 项目树的 **Solution Explorer**（解决方案资源管理器）中，选择 1 个文件夹。
2. 在上下文菜单中，选择命令 **Add** > **Text List...**（添加 > 文本列表……）
   ⇨ **Add Text List**（添加文本列表）对话框会打开。
3. 输入名称。
4. 点击 **Open**（打开）。

⇨ 文本列表被添加到 PLC 项目树中，并在编辑器中打开。

文本列表的结构

符号：

TextList_A

ID	Default	en-us
A	Information A	Information A_en
B	Information B	Information B_en: OK
C	Information C	Information C_en
D	Information D	Information D_en

ID	唯一文本标识符
标准	源文本作为字符串，例如，信息 A 双击字段可编辑文本。
该表格包含您可能已经添加的许多不同的语言列。语言列使用语言代码标识，该代码是您在使用 Add Language（添加语言）命令创建该列时指定的。	
<language code>	语言代码形式的语言名称，例如，en-US。该列包含在 Default（默认值）下所编写文本的译文。只要在可视化管理器中选择语言代码，可视化就会在操作过程中输出翻译文本。如果没有输入翻译，TwinCAT 将使用 Default（默认值）下的文本。在操作过程中，可视化可根据用户的要求切换语言。
空行	您可以编辑该行，添加自己的文本。

为动态文本输出创建文本列表

✓ 1 个可视化项目已打开。

1. 在 **Solution Explorer**（解决方案资源管理器）中，选择 PLC 项目树中的 PLC 项目对象或其下的文件夹。
2. 在上下文菜单中，选择命令 **Add** > **Text List**（添加 > 文本列表）。
3. 输入名称，例如，“TextList_A”，然后选择 Open（打开），退出对话框。
   ⇨ 创建 1 个 **Text List**（文本列表）类型的对象。
4. 点击 **Default**（默认值）列下，打开输入字段。输入文本，例如，“信息 A”。
   ⇨ 源文本已创建。它可以作为表格中的关键和翻译的源文本。
5. 在 **ID** 列中，输入任意字符串，例如，“A”。
6. 双击 Default（默认值）下表格末尾的空行，输入更多文本列表条目。
   ⇨ 带有源文本和 ID 的文本列表条目已定义。如果您为可视化中的元素配置属性 **Dynamic texts**（动态文本），那么，举例而言，您现在可以选择文本列表“TextList_A”，并分配 ID“A”。

动态文本输出

在可视化中，您可以通过配置元素的 **Dynamic texts**（动态文本）属性来配置在文本列表中编写的文本的动态输出。您可以直接分配文本列表和 ID，也可以使用 IEC 变量，通过编程设置值。

✓ 1 个可视化项目已打开，在 PLC 项目树中有 1 个文本列表。

1. 在编辑器中打开文本列表，例如，“TextList_A”。
2. 双击可视化对象。
   ⇨ 编辑器会打开。
3. 将 1 个元素（例如 **Textfield** 类型的元素）拖到可视化中。
4. 配置其属性 **Dynamic texts**（动态文本），方法是从 **Text list**（文本列表）属性中选择 1 个元素（例如，“TextList_A”），并在 **Text index**（文本索引）中输入文本列表中的 1 个 ID（例如，“A”）。注意引号。您也可以为文本列表名称和 ID 分配 1 个 STRING 类型的 IEC 变量。
   ⇨ IEC 变量可实现对文本列表中的文本的编程访问。
5. 编译应用程序，将其上传到控制器并启动它。

⇨ 可视化会将文本列表中的文本输出到文本字段中：信息 A。

7.13 使用图像池

图像池是 1 个图像文件表。通过指定图像池的 ID 和名称，当您在项目中（例如，在可视化中）使用图像文件时，TwinCAT 可以明确地引用该图像文件。1 个项目可以包含多个图像池。您可以在解决方案资源管理器的 PLC 项目树中添加图像池。在库项目中，您可以通过对象属性分配 1 个图像池作为可视化的符号库。

我们建议您在集成图像文件时尽可能减小其大小。这样可以优化所有可视化变体（TargetVisu、WebVisu 和编程系统）中可视化的加载时间。

当您在可视化中插入图像元素并在元素属性中输入 ID（静态 ID）时，TwinCAT 会自动创建 1 个全局图像池。TwinCAT 使用默认名称“GlobalImagePool”。

如果图像文件的 ID 存在于多个图像池中，请注意以下几点：

- 搜索顺序：如果您选择 1 个在 GlobalImagePool 中管理的图像，则您无需指定图像池的名称。图像文件的搜索顺序如下：
  - 1. GlobalImagePool
  - 2. 分配给当前活动 PLC 项目的图像池
  - 3. 与 GlobalImagePool 一起存在于 **Solution Explorer**（解决方案资源管理器）中的图像池
  - 4. 库中的图像池
- 唯一访问：根据语法 <collection name>.<image ID>，您可以在 ID 之前加上图像池的名称，从而直接且明确地处理所需的图像。

7.13.1 对象图像池

符号： 

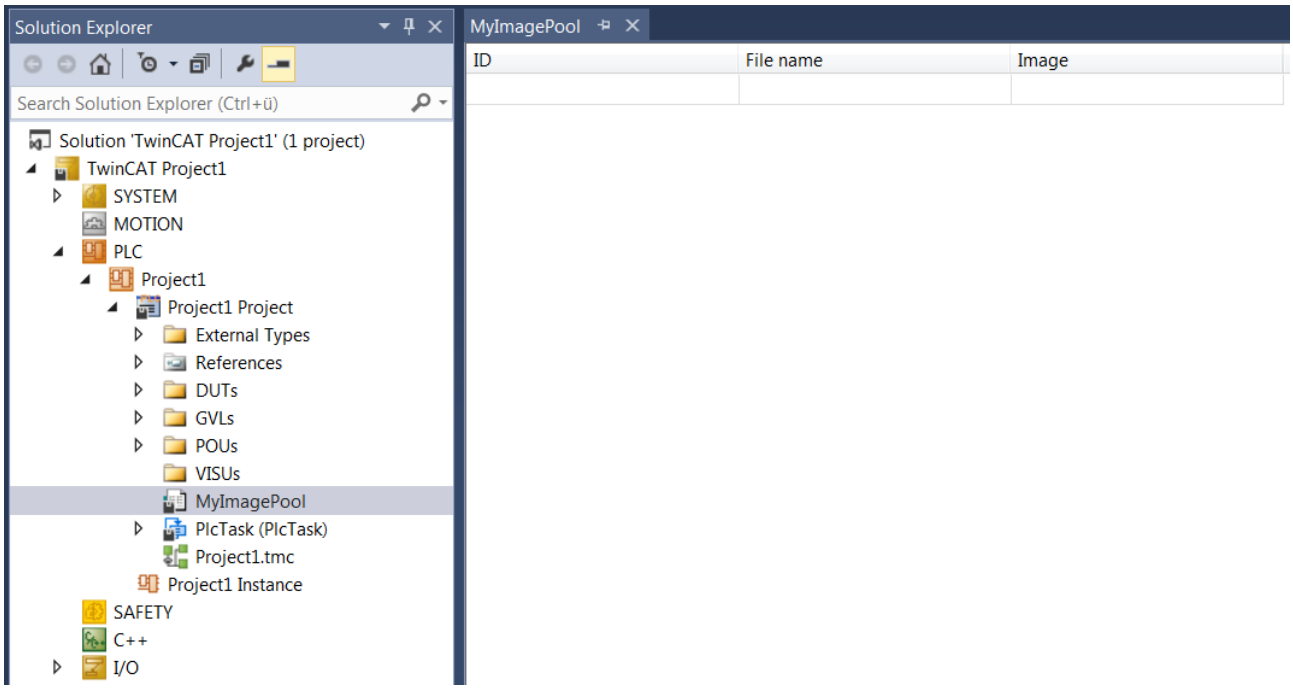
Image Pool（图像池）对象包含 1 个表格，在该表格中为图像分配 ID。

创建对象图像池

1. 在 PLC 项目树的 Solution Explorer（解决方案资源管理器）中，选择 1 个文件夹。
2. 在上下文菜单中，选择命令 Add > Image Pool（添加 > 图像池）。
   ⇨ Add Image Pool（添加图像池）对话框会打开。
3. 输入名称。
4. 点击 Open（打开）。

⇨ 图像池被添加到 PLC 项目树中，并在编辑器中打开。

图像池的结构



ID	图像 ID。例如，该 ID 可用于在可视化中引用图像。
文件名	图像文件的文件路径。当您点击 … 按钮时，标准图像选择对话框会打开。
图像	显示图像的缩略图视图。

7.13.2 创建图像池

创建图像池

1. 在 Solution Explorer（解决方案资源管理器）中，选择 PLC 项目树中的 PLC 项目对象或其下的文件夹。在上下文菜单中，选择命令 Add > Image Pool（添加 > 图像池）。
   ⇨ Add Image Pool（添加图像池）对话框会打开。
2. 输入图像池的名称（例如，“Images1”），并点击 Open（打开）按钮，确认对话框。
   ⇨ 图像池被添加到 PLC 项目树中，并在编辑器中打开。
3. 双击 ID 字段并分配 1 个合适的 ID（例如，“Icon1”）。
4. 双击 File name（文件名）列。点击 … 。
   ⇨ 标准的 Open（打开）对话框会打开。

5. 选择图像文件。
6. 在 **ID** 列中输入 ID。如果您不输入 ID，系统会自动插入文件名。

⇨ 在 **Image**（图像）列中会显示图像文件的缩略图视图。在 **File name**（文件名）中会显示路径，在 **ID** 字段中会显示您指定的文件名或 ID。现在，通过名称“Images1.Icon1”可以引用图像文件。

在图像可视化元素中使用图像文件

当您在可视化中插入图像元素时，您可以设置图像类型：

- 静态图像：在元素配置（属性 **Static ID**（静态 ID））中，输入图像文件的 ID 或图像池名称 + ID。请注意上文提供的有关搜索顺序和访问的信息。
- 动态图像：在元素配置（属性 **Bitmap ID variable**（位图 ID 变量））中，输入定义图像文件 ID 的变量，例如，MAIN.imagevar。在在线模式下，可根据变量替换动态元素

使用图像文件作为可视化背景

您可以在可视化的背景定义中指定 1 个图像。该图像可以通过图像池名称加上文件名来定义，如上文对可视化元素的描述。

7.14 检查语法和分析代码

TwinCAT 提供一些有用的功能来协助编程和故障排除。语法检查会突出显示程序输入过程中的错误，并发出相应的消息。

TwinCAT 中的静态代码分析为遵循指定的编码准则和识别易于出错的结构提供了额外的帮助。

7.14.1 检查语法

当您输入代码时，TwinCAT 预编译器会执行一些基本检查。在编辑器中，任何错误都会用红色波浪线突出显示。

在编程完成之后，您必须编译 PLC 项目。这可以通过 **Project**（项目）菜单中的 **Build**（构建）命令来完成。编译器会检查程序，并在 **Error List**（错误列表）中列出任何错误。

在将 PLC 项目下载到控制器之前，TwinCAT 会根据在开发系统中编写的源代码自动生成程序代码。在生成程序代码之前，系统会检查赋值、数据类型和库的可用性。此外，在生成程序代码时会分配内存地址。

通过 **Build**（构建）菜单中的命令 **Check all objects**（检查所有对象）也可以明确启动检查。这还能检测当前 PLC 项目中未使用的对象中的错误。

TwinCAT 会在 **Error List**（错误列表）中列出所有错误和警告。双击错误信息可打开受影响的 POU。故障位置会突出显示。或者，您也可以通过错误消息的上下文菜单导航到故障位置。

另请参见 **Smart Coding**（智能编码）下 TwinCAT 选项中的设置。

另请参见：

- TC3 用户界面文档：对话框选项 - 智能编码 [▶ 911]

7.14.2 代码分析（静态分析）

在将项目加载到目标系统之前，TwinCAT 3 PLC 会使用“Static Code Analysis”（静态代码分析）来检查源代码是否遵循指定的编码准则。

免授权版本的静态代码分析被称为“Static Analysis Light”（轻量静态分析）。每当生成代码时，就会自动执行在分析功能中配置的检查。有关更多信息，请参见“轻量静态分析 [▶ 139]”部分。

静态代码分析的授权版本是“Static Analysis”（静态分析），与轻量版本相比，它的功能和配置的范围均大大扩展。静态代码分析可手动触发或在生成代码的过程中自动执行。有关该扩展的更多信息，请参见文档“TE1200 | TC3 PLC 静态分析”。

轻量静态分析与 全量静态分析

下文概述了静态分析软件免授权版本和授权管理版本的不同功能。

功能方面	轻量静态分析 （无 TE1200 授权）	全量静态分析 （有 TE1200 授权）
需要授权使用	否，可免费使用	是，需要 TE1200 授权使用
保存/导出和加载/导入（规则）配置	不支持，与 PLC 项目属性耦合	支持 （使用 中的 **Load/Save**（加载/保存）按钮）
执行与编译过程耦合	是，不可配置	可配置 （使用 中的 **Perform static analysis automatically**（自动执行静态分析）选项； 借助命令 手动执行）
检查未使用的对象（例如，库项目中的对象）	不支持	支持 （借助命令 ）
报告错误的最大数量	500（不可配置） （有关 500 作为错误的最大数量之意义的更多信息，请参见 ）	可配置 （使用 中的设置 **Maximum number of errors**（错误的最大数量））
报告警告的最大数量	不支持警告输出（见下一行）	可配置 （使用 中的设置 **Maximum number of warnings**（警告的最大数量））
规则：激活选项	• 激活并作为错误输出 • 未激活	• 激活并作为错误输出 • 激活并作为警告输出 • 未激活
规则：范围	7 条编码规则 • SA0033：未使用的变量 • SA0028：内存区重叠 • SA0006：对多个任务进行写入访问 • SA0004：在输出端进行多次写入访问 • SA0027：多次使用名称 • SA0167：报告临时功能块实例 • SA0175：字符串上的可疑操作	超过 100 条编码规则
规则：预编译波浪下划线，QuickFix	不可用	可用
	不可用	可用
	不可用	可用
	不可用	可用
临时禁用规则的编译指示和属性	是，在轻量范围中可用： • 编译指示 {analysis ...} • 属性 {attribute 'no-analysis'} • 属性 {attribute 'analysis' := '...'}	是，在全量范围中可用： • 编译指示 {analysis ...} • 属性 {attribute 'no-analysis'} • 属性 {attribute 'analysis' := '...'} • 属性 {attribute 'naming' := '...'} • 属性 {attribute 'nameprefix' := '...'} • 属性 {attribute 'analysis:report-multiple-instance-calls'}

7.14.2.1 轻量静态分析

在将项目加载到目标系统之前，TwinCAT 3 PLC 会使用“Static Code Analysis”（静态代码分析）来检查源代码是否遵循指定的编码准则。

免授权版本的静态代码分析被称为“Static Analysis Light”（轻量静态分析）。每次成功生成代码后，都会自动执行在此处配置的检查。在 **Static Analysis**（静态分析）下的 PLC 项目属性中定义了所需的规则集。

偏离规则的情况会以错误消息的形式出现在 **Error List**（错误列表）中。每条规则都有 1 个唯一的编号。如果在静态分析过程中检测到违反规则的情况，则会在错误列表中输出规则编号以及基于以下语法的错误描述。缩写“SA”代表“静态分析”。

语法：“SA<rule number>: <rule description>”

第 33 条规则示例（未使用的变量）：“SA0033：未使用：变量‘bSample’”

TwinCAT 只分析当前项目的程序代码，而不分析库。

请注意，在编译成功后会自动执行轻量静态分析。另一方面，如果未能成功生成代码，即如果编译器检测到编译错误，则不会执行轻量静态分析。

您可以使用编译指示来禁用某些代码部分的检查（见下文）。

配置规则集

PLC 项目属性的 **Static Analysis**（静态分析）类别定义了轻量版静态代码分析在生成代码时执行的检查。

静态分析配置的范围

您在 PLC 项目属性的 **Static Analysis**（静态分析）类别中设置的参数被称为 **Solution options**（解决方案选项），因此，这些参数不仅仅会影响您当前编辑属性的 PLC 项目。配置的规则集可用于开发环境中的所有 PLC 项目。

轻量静态分析中的可用规则：

- SA0033：未使用的变量
- SA0028：内存区重叠
- SA0006：多个任务中的写入访问
- SA0004：在输出端进行多次写入访问
- SA0027：多次使用名称
- SA0167：报告临时功能块实例
- SA0175：字符串上的可疑操作

SA0033：未使用的变量

功能	确定已声明但未在编译后的程序代码中使用的变量。
原因	未使用的变量会降低程序的易读性和可维护性。未使用的变量会占用不必要的内存空间，并在初始化过程中占用不必要的运行时。
重要性	中等
PLCopen 规则	CP22/CP24

SA0028：内存区重叠

功能	确定 2 个或多个变量占用相同存储空间的点。
原因	如果 2 个变量占用相同的存储空间，则代码可能会表现出非常出人意料的行为。在任何情况下都必须避免这种情况。如果无法避免在不同的解释中使用 1 个值，例如，1 次作为 DINT，1 次作为 REAL，您应定义 1 个 UNION。此外，通过指针您可以访问以其他方式键入的值，而无需转换该值。
重要性	高

示例：

在下面的示例中，2 个变量都使用字节 21，即变量的内存区重叠。

```
PROGRAM MAIN
VAR
    nVar1 AT%QB21  : INT;         // => SA0028
    nVar2 AT%QD5   : DWORD;       // => SA0028
END_VAR
```

SA0006：多个任务中的写入访问

功能	确定可从多个任务中写入访问的变量。
原因	在某些情况下，在多个任务中写入的变量可能会意外地改变其值。这样可能会导致混乱的情况。如果同时在 2 个任务中写入变量，那么，字符串变量以及某些 32 位系统上的 64 位整数变量甚至可能会出现不一致的状态。
例外	在某些情况下，可能需要为多个任务写入 1 个变量。例如，通过使用信号确保访问不会导致不一致的状态。
重要性	高
PLCopen 规则	CP10

另请参见规则 SA0103。

调用对应于写入访问

请注意，调用被视为写入操作。例如，调用功能块实例的方法会被视为对功能块实例的写入访问。不过，当涉及到虚拟调用（例如指针和接口）时，进行更详细的访问和调用分析是比较困难的。

如要对某个变量（例如，功能块实例）停用规则 SA0006，可在变量声明上方插入以下属性：
{attribute 'analysis' := '-6'}

示例：

2 个全局变量 nVar 和 bVar 由 2 个任务写入。

全局变量列表：

```
VAR_GLOBAL
    nVar  : INT;
    bVar  : BOOL;
END_VAR
```

任务 PlcTaskFast 调用的程序 MAIN_Fast：

```
nVar := nVar + 1;                 // => SA0006
bVar := (nVar > 10);              // => SA0006
```

任务 PlcTaskSlow 调用的程序 MAIN_Slow：

```
nVar := nVar + 2;                 // => SA0006
bVar := (nVar < -50);             // => SA0006
```

SA0004：对输出的多次写入访问

功能	确定在多个位置写入的输出。
原因	如果在代码中的多个地方对一个输出变量做写操作，可维护性会受到影响。这样，我们便无法分清到底是哪种写入访问影响了进程。最好将输出变量的计算结果存储在辅助变量中，并在周期结束时将计算值分配给输出变量。
例外	如果在 IF 或 CASE 语句的不同分支中写入输出变量，不会出现错误。
重要性	高
PLCopen 规则	CP12

不能通过在代码中写pragma或attribute禁用该规则！
有关attribute的更多信息，请参见“”。

Sample：

全局变量列表：

```
VAR_GLOBAL
    bVar     AT%QX0.0 : BOOL;
    nSample  AT%QW5   : INT;
END_VAR
```

MAIN 程序：

```
PROGRAM MAIN
VAR
    nCondition        : INT;
END_VAR
```

```
IF nCondition < INT#0 THEN
    bVar    := TRUE;             // => SA0004
    nSample := INT#12;           // => SA0004
END_IF

CASE nCondition OF
    INT#1:
        bVar := FALSE;           // => SA0004

    INT#2:
        nSample := INT#11;       // => SA0004

ELSE
     bVar    := TRUE;            // => SA0004
     nSample := INT#9;           // => SA0004
END_CASE
```

SA0027：多次使用名称

功能	确定在项目范围内多次使用变量名/标识符或对象名称（POU）的情况。包括以下情况： • 枚举常量的名称与应用程序内或所含库中另一个枚举的名称相同。 • 变量的名称与应用程序中或所含库中另一个对象的名称相同。 • 变量的名称与应用程序中或所含库中枚举常量的名称相同。 • 对象的名称与应用程序中或所含库中另一个对象的名称相同。
原因	在读取代码时，相同的名称可能会引起混淆。如果在无意中访问了错误的对象，它们可能会导致错误。因此，应定义并遵循命名规范，以避免出现此类情况。
例外	使用“qualified_only”属性声明的枚举可免于 SA0027 检查，因为其元素只能以限定的方式访问。
重要性	中等

示例：

由于在项目中引用了提供功能块 TON 的 Tc2_Standard 库，以下示例会产生错误/警告 SA0027。

```
PROGRAM MAIN
VAR
    ton  : INT;                 // => SA0027
END_VAR
```

SA0167：报告临时功能块实例

功能	识别声明为临时变量的功能块实例。它适用于在方法中、在函数中或作为 VAR_TEMP 声明的实例，这些实例在每个处理周期或每次功能块调用中都会重新初始化。
原因	• 功能块的状态通常会保留几个 PLC 周期。堆栈上的实例仅在函数调用期间存在。因此，只有在极少数情况下才需要创建 1 个实例作为临时变量。 • 其次，功能块实例通常很大，需要大量的堆栈空间（而控制器上的空间通常有限）。 • 第三，功能块的初始化和调度往往也会占用大量时间。
重要性	中等

示例：

方法 FB_Sample.SampleMethod：

```
METHOD SampleMethod : INT
VAR_INPUT
END_VAR
VAR
    fbTrigger : R_TRIG;         // => SA0167
END_VAR
```

Function F_Sample：

```
FUNCTION F_Sample : INT
VAR_INPUT
END_VAR
VAR
    fbSample  : FB_Sample;      // => SA0167
END_VAR
```

MAIN 程序：

```
PROGRAM MAIN
VAR_TEMP
    fbSample  : FB_Sample;      // => SA0167
    nReturn   : INT;
END_VAR
```

```
nReturn := F_Sample();
```

SA0175：字符串上的可疑操作

功能	识别代码中可能不符合UTF-8编码规则或可能导致编码错误的位置。
捕获的结构	1. 对单字节字符串的索引访问 ◦ 示例：sVar[2] ◦ 消息：字符串上的可疑操作：索引访问“<expression>” 2. 对单字节字符串的地址访问 ◦ 示例：ADR(sVar) ◦ 消息：字符串上的可疑操作：可能的索引访问“<expression>” 3. 调用 Tc2_Standard 库的字符串函数，CONCAT 和 LEN 除外 ◦ 示例：FIND(sVar, 'a'); ◦ 消息：字符串上的可疑操作：可能的索引访问“<expression>” 4. 包含非 ASCII 字符的单字节字面值 ◦ 示例： sVar := '99€'; sVar := 'Ä'; ◦ 消息：字符串上的可疑操作：字面量“<literal>”包含非 ASCII 字符
重要性	中等

示例：

```
VAR
    sVar  : STRING;
    pVar  : POINTER TO STRING;
    nVar  : INT;
END_VAR
```

```
// 1) SA0175: Suspicious operation on string: Index access
sVar[2];                        // => SA0175

// 2) SA0175: Suspicious operation on string: Possible index access
pVar := ADR(sVar);              // => SA0175

// 3) SA0175: Suspicious operation on string: Possible index access
nVar := FIND(sVar, 'a');        // => SA0175

// 4) SA0175: Suspicious operation on string: Literal '<...>' contains Non-ASCII character
sVar := '99€';                  // => SA0175
sVar := 'Ä';                    // => SA0175
```

轻量静态分析的编译指示和属性

编译指示或属性可用于将代码的某些部分排除在检查之外。使用编译指示 {analysis ...} [▶ 143] 可关闭实现部分的编码规则，使用属性 {attribute 'analysis' := '...'} [▶ 144] 可关闭声明部分的编码规则。

要求：您已在项目属性中激活规则。

此外，您还可以使用属性 {attribute 'no-analysis'} [▶ 145] 将编程对象排除在静态分析之外。

在项目属性中禁用的规则不能通过编译指示或属性激活。

规则 SA0004（在输出端进行多次写入访问）不能通过编译指示禁用。

编译指示 {analysis ...}

您可以在编程块的实现部分使用编译指示 {analysis -/+<rule number>}，以忽略后续代码行的个别编码规则。停用编码规则的方法是在规则编号前加上减号（“-”）。激活时，在前面会加上 1 个加号（“+”）。借助逗号分隔，您可以在编译指示中指定许多不同的规则。

插入位置：

- 停用规则：在使用 {analysis - ...} 禁用代码分析的第 1 行代码的实现部分。
- 激活规则：在使用 {analysis + ...} 停用的最后一行后。
- 对于规则 SA0164，也可以在注释之前的声明部分插入编译指示。

语法：

- 停用规则：
  - 1 条规则：{analysis -<rule number>}
  - 多条规则：{analysis -<rule number>、-<further rule number>、-<further rule number>}
- 激活规则：
  - 1 条规则：{analysis +<rule number>}
  - 多条规则：{analysis +<rule number>、+<further rule number>、+<further rule number>}

示例：

对 1 行禁用规则 24（只允许键入字面量）（即在这些行中，不必写入“nTest := DINT#99”），然后再次启用该规则：

```
{analysis -24}
nTest := 99;
{analysis +24}
nVar := INT#2;
```

指定多条规则：

```
{analysis -10, -24, -18}
```

属性 {attribute 'analysis' := '...'}

您可以使用属性 {attribute 'analysis' := '-<rule number>'} 关闭单个声明或整个编程对象的某些规则。通过指定前面带减号的规则编号，可停用代码规则。您可以在属性中指定许多不同的规则。

插入位置：

在编程对象声明的上方，或在变量声明的上一行

语法：

- 1 条规则：{attribute 'analysis' := '-<rule number>'}
- 多条规则：{attribute 'analysis' := '-<rule number>、-<further rule number>、-<further rule number>'}

示例：

对于结构的所有变量，规则 33（未使用的变量）将被禁用。

```
{attribute 'analysis' := '-33'}
TYPE ST_Sample :
STRUCT
    bMember  : BOOL;
    nMember  : INT;
END_STRUCT
END_TYPE
```

对于变量 nVar1，规则 28（内存区重叠）和规则 33（未使用的变量）的检查将被禁用。

```
PROGRAM MAIN
VAR
    {attribute 'analysis' := '-28, -33'}
    nVar1 AT%QB21   : INT;
    nVar2 AT%QD5    : DWORD;

    nVar3 AT%QB41   : INT;
    nVar4 AT%QD10   : DWORD;
END_VAR
```

对于全局变量，规则 6（并发访问）将被禁用，这样，如果从多个任务对变量进行写入访问，就不会生成错误消息。

```
VAR_GLOBAL
    {attribute 'analysis' := '-6'}
    nVar   : INT;
    bVar   : BOOL;
END_VAR
```

属性 {attribute 'no-analysis'}

您可以使用 {attribute 'no-analysis'} 属性将整个编程对象排除在静态分析检查之外。对于该编程对象，不会对编码规则、命名规范和禁用符号执行检查。

插入位置：

在程序对象的声明上方

语法：

```
{attribute 'no-analysis'}
```

示例：

```
{attribute 'qualified_only'}
{attribute 'no-analysis'}
VAR_GLOBAL
    …
END_VAR
```

```
{attribute 'no-analysis'}
PROGRAM MAIN
VAR
    …
END_VAR
```

7.15 定向和导航

7.15.1 使用交叉引用列表查找出现

您可以将变量或 POU（程序、功能块、函数）的位置输出到所谓的 **Cross Reference List**（交叉引用列表）中，您可以从该列表跳转到项目中的相应位置。

✓ 1 个 POU 已在编辑器中打开。

1. 将光标置于声明或实现中的变量或 POU 的名称上。
2. 在菜单 **PLC** > **Window**（PLC > 窗口）中，选择命令 **Cross Reference List**（交叉引用列表），或者在编辑器的上下文菜单中，选择命令 **Find All References**（查找所有引用）。

   ⇨ **Cross Reference List**（交叉引用列表）视图将会打开并显示变量或 POU 的位置。它总是会显示声明位置，并在其下缩进显示在项目中使用声明的位置。如果您搜索结构化变量或功能块名称，则会在声明位置下方缩进显示成员或功能块实例的位置。
3. 双击交叉引用列表的位置。

⇨ 相应的对象会在编辑器中打开，并突出显示该位置。

如果 **Cross Reference List**（交叉引用列表）视图已打开，您还可以按照如下步骤指定变量、POU 或 DUT，以查找其位置：

- 在 **TwinCAT** > **PLC Environment** > **Smart Coding**（TwinCAT > PLC 环境 > 智能编码）下的菜单 **Tools** > **Options**（工具 > 选项）中，启用选项 **Automatically list selection in cross reference view**（在交叉引用视图中自动列出选择）。然后，将光标置于 POU 编辑器窗口中的变量/POU/DUT 的名称上。
- 在 **Name**（名称）字段中，手动输入变量名。

您可以使用占位符“*”（许多不同的字符）或“?”（只有 1 个字符）与变量标识符的子字符串的组合。

另请参见：

- TC3 用户界面文档：命令查找所有引用 [▶ 817]
- TC3 用户界面文档：命令：交叉引用表 [▶ 875]
- TC3 用户界面文档：对话框选项 - 智能编码 [▶ 911]

7.15.2 查找声明

TwinCAT 提供在整个项目中搜索变量或函数的定义位置的选项。在编辑器中打开包含定义的功能块，并选择声明。

查找变量的声明

✓ 1 个 POU 已在编辑器中打开。

1. 将光标置于实现中的标识符上。
2. 在编辑器的上下文菜单中，选择命令 **Go To Definition**（转至定义）。

⇨ 在编辑器中打开包含定义的 POU，并选择变量定义。如果定义位于“编译”库中，则会在库管理器中打开相应的功能块。

您可以在离线和在线模式下使用该命令。

示例：

下面的功能块包含功能块定义（fbInst）、程序调用（SampleProg()）和功能块调用（fbInst.nOut）：

```
VAR
    fbInst : FB_Sample;
    nVar : INT;
    nRes : INT;
END_VAR

SampleProg();
nVar := SampleProg.nVar1;
nRes := fbInst.nOut;
```

如果您将光标置于 SampleProg 上，命令会在编辑器中打开 SampleProg 程序。

如果您将光标置于 fbInst 上，命令就会将焦点放在 fbInst : FBSample; 行中的声明窗口上

如果您将光标置于 nOut 上，命令会在编辑器中打开功能块 FB_Sample。

另请参见：

- TC3 用户界面文档：命令转至定义 [▶ 816]

7.15.3 设置和使用书签

您可以使用书签，以便浏览较长的程序。除了 SFC（顺序功能图）之外，书签可用于所有编程语言编辑器。这些命令可用于直接导航到选定的程序位置。

默认情况下，书签存储在 Visual Studio 项目的用户选项文件（.suo）中。该文件是用户特定文件，因此与源代码控制管理系统不兼容。因此，书签只对本地项目有效。

为了跨用户保存书签，您可以选择将书签另外保存在 1 个单独文件中的选项 Write Bookmarks to File [▶ 853]（将书签写入文件），然后将其作为 TwinCAT 存档选项的一部分。该文件也与源码控制管理系统不兼容。

设置/清除书签

✓ 1 个编程块已在编辑器中打开。

1. 将光标置于任何程序行中。

2. 从菜单 **PLC** > **PLC Bookmark**（PLC > PLC 书签）中，选择命令 **Enable/disable bookmarks**（启用/禁用书签）。

   ⇨ 此时，在程序中会设置 1 个书签。此情况用书签图标 表示。

3. 在程序中的不同点设置多个书签。
4. 如要删除书签，可将光标置于有书签的程序行中。
5. 从菜单 **PLC** > **PLC Bookmark**（PLC > PLC 书签）中，选择命令 **Enable/disable bookmarks**（启用/禁用书签）。

⇨ 书签再次被删除。书签图标 已被删除。

或者，您也可以使用按钮 在 **PLC Bookmarks**（PLC 书签）窗口中删除 1 个或多个书签。为此，您必须选择相应的书签。

在菜单 **PLC** > **PLC Bookmark**（PLC > PLC 书签）中，选择命令 **Delete all bookmarks (active editor)**（删除全部书签（活动编辑器）），可删除活动编程块的所有书签。

在菜单 **PLC** > **PLC Bookmarks**（PLC > PLC 书签）中，选择命令 **Delete all bookmarks**（删除全部书签），可删除项目的所有书签。

另请参见：

- 命令启用/禁用书签 [▶ 885]
- 命令清除所有书签（活动编辑器） [▶ 887]
- 命令清除所有书签 [▶ 886]

跳转到编程块内的书签

✓ 1 个编程块已在编辑器中打开。多个书签已设置。

1. 从菜单 **PLC** > **PLC Bookmark**（PLC > PLC 书签）中，选择命令 **Next Bookmark (active editor)**（下一个书签（活动编辑器））。

   ⇨ 根据当前光标位置，光标会向下跳转到下一个书签。

2. 从菜单 **PLC** > **PLC Bookmark**（PLC > PLC 书签）中，选择命令 **Previous Bookmark (active editor)**（上一个书签（活动编辑器））。

⇨ 根据当前光标位置，光标会向上跳转到上一个书签。

另请参见：

- 命令下一个书签（活动编辑器） [▶ 886]
- 命令上一个书签（活动编辑器） [▶ 887]

跳转到项目的不同编程块的书签

✓ 1 个包含多个编程块的项目已打开。在不同的编程块中已设置多个书签。

1. 从菜单 **PLC** > **Window**（PLC > 窗口）中，选择命令 **PLC Bookmarks**（PLC 书签）

   ⇨ **Bookmarks**（书签）视图已打开。
   项目的所有书签均以表格形式在视图中列出。

2. 点击 **Next Bookmark**（下一个书签） 按钮。

   ⇨ 在 **Bookmark**（书签）视图中，当前所选书签下方行中的书签已被选中。
   带有表格中新选书签的编程块已在编辑器中打开，并且，带有书签的行已被选中。

3. 根据步骤 2，您可以使用按钮 **Previous Bookmark**（上一个书签） 跳转到在上一行中的 **Bookmark**（书签）视图中显示的项目书签。

另请参见：

- 命令 PLC 书签 [▶ 883]
- 命令下一个书签 [▶ 885]

- 命令上一个书签 [▶ 886]

7.16 在整个项目中查找和替换

在 TwinCAT 中，您可以在单个对象或整个项目中搜索字符串，并根据需要用另一个字符串替换它们。

1. 在菜单 **Edit** > **Find and Replace**（编辑 > 查找和替换）中，启用命令 **Quick Find**（快速查找）。
   ⇨ **Find and Replace**（查找和替换）对话框会打开。
2. 在 **Find what** （查找内容）字段中，输入所需的字符串。
3. 在 **Find in**（在……中查找）选择列表中，指定要搜索的对象。
4. 选择 **search options**（搜索选项）。
5. 选择 **result options**（结果选项）。
6. 点击 **Find Next**（查找下一个）。
   ⇨ 在编辑器中会显示第 1 个查询结果。
7. 点击 **Find all**（查找全部）可获得所有查询结果的概览。
   ⇨ **Search results**（搜索结果）屏幕打开，显示查询结果列表。
8. 点击 **Replace in files**（在文件中替换），用另一个字符串替换搜索字符串。
9. 在 **Replace with**（替换为）字段中，输入要替换原始字符串的字符串。
10. 点击 **Replace**（替换）。
11. 点击 **Replace All**（全部替换）可替换所有字符串。

另请参见：

- TC3 用户界面文档：命令：快速替换 [▶ 820]
- TC3 用户界面文档：命令：快速查找 [▶ 818]

7.17 重构

一般来说，重构是 1 种在不改变现有软件行为的前提下改进其设计的方法。

在 TwinCAT 中，重构提供了重命名对象和变量名以及更新功能块连接的功能。您可以显示重命名对象和变量显示的所有位置，然后对它们进行单独或集体重命名。此外，在 TwinCAT 选项 **Tools** > **Options**（工具 > 选项）中的 **TwinCAT** > **PLC Environment** > **Refactoring**（TwinCAT > PLC 环境 > 重构）类别下，您可以配置 TwinCAT 是否以及在何处自动发出重构提示。

重命名全局变量

在整个项目中重命名全局变量：

✓ 您已打开 1 个至少包含 1 个功能块 FB_Sample 和 1 个全局变量列表 GVL 的项目。全局变量列表已在您的编辑器中打开，其中包含变量的声明，例如，nGlob1。FB_Sample 使用 nGlob1。

1. 选择全局变量的名称，例如，“nGlob1”。
2. 在上下文菜单中，选择命令 **Refactoring** > **Rename 'nGlob1'**（重构 > 重命名“nGlob1”）。
3. 在 **Rename**（重命名）对话框中，在 **New name**（新名称）输入字段中输入新名称，例如，“nGlobNew”，然后点击 **OK**（确定）。
   ⇨ **Refactoring**（重构）对话框会打开。在左侧的项目树中，对象 GVL 和 FB_Sample 显示为红色，背景为黄色。在右侧的窗口中，FB_Sample 已在编辑器中打开，nGlob1 已重命名为 nGlobNew。
4. 点击 **OK**（确定）。

⇨ 在您的项目中，全局变量 nGlob1 的所有实例均已重命名为 nGlobNew。

在整个项目中重命名全局变量，但 POU 除外：

1. 选择全局变量的名称，例如，“nGlob1”。
2. 在上下文菜单中，选择命令 **Refactoring** > **Rename 'nGlob1'**（重构 > 重命名“nGlob1”）。
3. 在 **Rename**（重命名）对话框中，在 **New name**（新名称）输入字段中输入新名称，例如，“nGlobNew”，然后点击 **OK**（确定）。

⇨ **Refactoring**（重构）对话框会打开。在左侧的项目树中，对象 GVL 和 FB_Sample 显示为红色，背景为黄色。在右侧窗口中，功能块 FB_Sample 已在编辑器中打开。列出的不是 nGlob1，而是 nGlobNew。

4. 将光标置于右侧的窗口，打开上下文菜单。
5. 选择命令 **Reject this object**（拒绝此对象），然后点击 **OK**（确定）。

⇨ 您的项目在 FB_Sample 中包含全局变量 nGlob1。否则，变量 nGlobNew 将在出现变量的对象中指定。消息窗口会显示 1 条错误消息，说明未定义标识符 nGlob1。

另请参见：

- TC3 用户界面文档：命令：重命名 '<variable>' [▶ 822]

添加和删除输入变量

在重构命令的功能块声明部分，您可以添加或删除输入或输出变量。TwinCAT 会相应地更新调用/使用功能块的位置；您可以接受或放弃对每个位置的更新。**Refactoring**（重构）对话框可用于此目的。

✓ 您已在编辑器中打开函数 F_Sample。该函数已有输入变量 nInput1、nInput2 和 nInputx。它在 MAIN 和 POU 程序中被调用。

1. 将焦点放在函数 F_Sample 的声明部分。
2. 在上下文菜单中，选择命令 **Refactoring** > **Add variable**（重构 > 添加变量）。

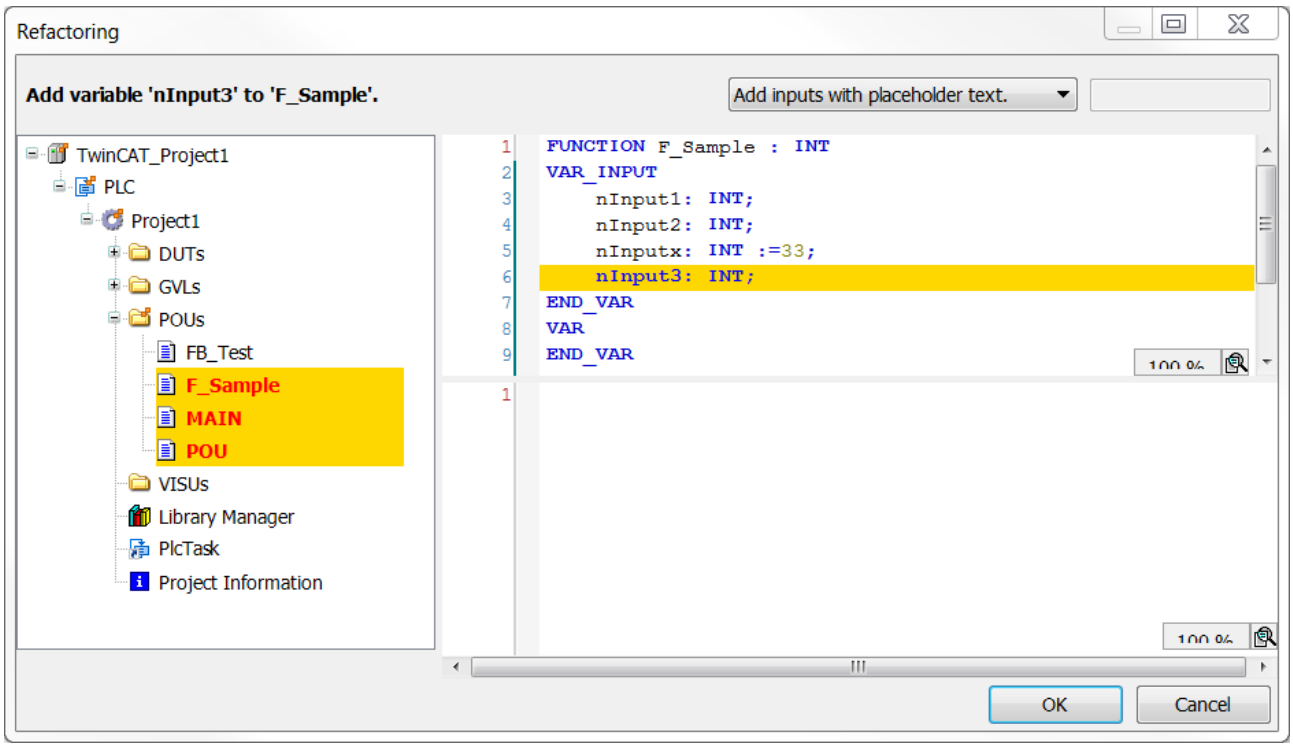
   ⇨ 用于声明变量的标准对话框会出现。
3. 声明变量 nInput3，范围为 VAR_INPUT，数据类型为 INT。选择 **OK**（确定），关闭对话框。

   ⇨ **Refactoring**（重构）对话框会出现（见下图）。受影响的区域被标记为黄色。
4. 在右上角选择选项 **Add inputs with placeholder text"**（添加带占位符文本的输入）。
5. 在左侧的窗口中，将光标置于 1 个以黄色突出显示的对象上，例如 MAIN。在上下文菜单中，选择命令 **Accept whole project**（接受整个项目），以便在整个项目中使用 F_Sample 的位置添加新变量。

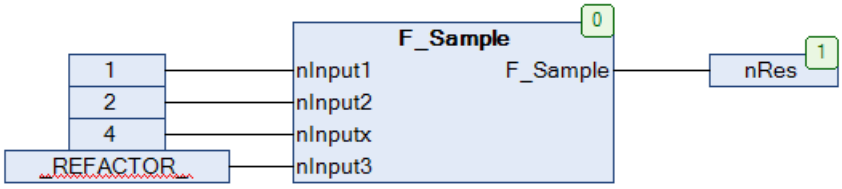
   ⇨ 右侧的窗口显示 MAIN 的实现部分中的变化：占位符 _REFACTOR_ 会在插入新变量的位置出现。
6. 选择 **OK**（确定），关闭 **Refactoring**（重构）对话框。

⇨ 选择命令 **Edit** > **Find and Replace**（编辑 > 查找和替换）。在项目中查找“_REFACTOR_”，以便检查受影响的位置，并根据需要进行编辑。

或者，您也可以直接添加新变量，并自选 1 个初始化值，而不用从占位符开始。在这种情况下，选择步骤 4 下的选项“Add inputs with the following value”（添加具有以下值的输入），并输入右侧的值。



在 CFC 功能块中使用占位符文本的新变量示例：



请注意，重构也可用于删除变量。

另请参见：

- TC3 用户界面文档：命令：删除 '<variable>' [▶ 825]
- TC3 用户界面文档：命令：添加 '<variable>' [▶ 823]

在声明中重新排列变量

您可以使用重构来更改功能块声明部分中的声明顺序。这适用于有效范围为 VAR_INPUT、VAR_OUTPUT 或 VAR_IN_OUT 的声明。

```
VAR_INPUT
    nInVar2 : INT;
    nInVar1 : INT;
    in      : DUT;
    bVar    : BOOL;
    nInVar3 : INT;
END_VAR
```

✓ 例如，您已打开 POU 的声明部分，其中可能包含上文所示的声明。

1. 将光标置于该声明块上，然后按鼠标右键打开上下文菜单。
2. 在上下文菜单中，选择命令 **Refactoring** > **Reorder variables**（重构 > 重新排序变量）。
   ⇨ **Reorder**（重新排序）对话框将显示 VAR_INPUT 变量列表。
3. 例如，选择条目 nInVar1 : INT;，然后用鼠标将其拖到条目 nInVar2 : INT; 之前。
   ⇨ 现在，nInVar1 的声明位于顶部。
4. 选择 **OK**（确定），关闭对话框。
   ⇨ **Refactoring**（重构）对话框会出现。受影响的区域被标记为黄色（见上图）。
5. 点击 **OK**（确定）进行确认。

⇨ 新顺序已传输到功能块。

另请参见：

- TC3 用户界面文档：命令：重新排序变量 [▶ 825]

更改变量声明并自动应用重构

重构支持在声明中重命名变量（使用自动声明功能）。

✓ 您已在编辑器中打开功能块 FB_Sample。该功能块有 1 个输入变量 "nVAR"。

✓ 在 **TwinCAT** > **PLC Environment** > **Refactoring**（TwinCAT > PLC 环境 > 重构）类别中的 TwinCAT 选项 **Tools** > **Options**（工具 > 选项）中，选项 **On renaming variables**（重命名变量）已被激活。

1. 在 FB_Sample 的声明中，选择变量 nVar。或者，您也可以将光标置于变量之前或变量中。
2. 在 **Edit**（编辑）菜单或上下文菜单中，选择 **Auto Declare**（自动声明）命令。

   ⇨ **Auto Declare**（自动声明）对话框会打开。对话框包含“nVar”的设置。
3. 将名称从“nVar”改为“nCounterVar”。
4. **Apply changes using refactoring**（通过重构应用更改）选项会出现并启用。在更改变量名时，该选项的显示与 TwinCAT 选项中的设置无关。不过，只有在先决条件中提到的 TwinCAT 重构选项被激活的情况下，它才会被自动激活。
5. 点击 **OK**（确定）。

   ⇨ 对话框 **Refactoring**（重构）会打开。所有受变量重命名影响的位置都会在这里标注出来，并可以更改。

另请参见：

- TC3 用户界面文档：命令：自动声明 [▶ 811]
- TC3 用户界面文档：对话框选项 - 重构 [▶ 909]

7.18 数据持久性

当重新加载 PLC 项目时，持久数据会保留其值。在失控终止（断电）、下载或冷启动后，可恢复持久数据的值。

除了 PERSISTENT 变量外，RETAIN 变量也是 1 种保留剩余数据的方法。在出现失控终止（断电）、下载或冷启动的情况时，RETAIN 变量同样可以保留其值。

另请参见：

- 重置 PLC 项目 [▶ 201]
- 引用编程：剩余变量 - PERSISTENT, RETAIN [▶ 630]
-

7.19 使用功能块进行隐式检查

TwinCAT 提供特殊的 POU，可实现隐式监控函数。您可以将这些特殊的 POU 添加到应用程序中，使隐式提供的监控功能可用。在运行时，这些函数会检查数组或子范围类型的边界、指针地址的有效性或除以 0 的情况。请注意，如果此类测试功能块由特定隐式库提供，则可能会在设备上禁用此功能。

7.19.1 对象用于隐式检查的 POU

对象创建用于隐式检查的 POU

1. 在 PLC 项目树中选择 1 个文件夹。
2. 在上下文菜单中，选择命令 **Add** > **POU for implicit checks...**（添加 > 用于隐式检查的 POU……）

   ⇨ 对话框 **Add POU for implicit checks**（添加用于隐式检查的 POU）会打开。
3. 激活所需的功能。

4. 点击 **Open**（打开）按钮。
   ⇨ 在 PLC 项目树中插入选定的 POU。
5. 在编辑器中打开 POU。
6. 根据您的要求调整实现建议。

为了防止多重集成，在 **Add POU for implicit checks**（添加用于隐式检查的 POU）对话框中，可能已经集成的监控函数不再可供选择。

不要更改声明部分

为了保持监控函数的正常运行，不得修改声明部分。唯一的例外是添加局部变量。

不可进行在线更改

从项目中删除隐式监控函数（例如，CheckBounds）之后，便无法再进行在线更改，只能进行下载。系统会发出相应的消息。

对库中的功能块进行隐式检查

TwinCAT 不会对库中的功能块执行隐式检查。不过，您可以使用编译器定义“checks_in_libs”，以将检查扩展到库中的功能块。为此，可在 **Compile**（编译）类别的 **Compiler defines**（编译器定义）字段中输入 **PLC project properties**（PLC 项目属性）中的编译器定义“checks_in_libs”。这会影响源库（*.library）。

从 TC3.1 Build 4026 开始，该检查还可扩展到受保护库（*.compiled-libraries）中的功能块。为此，您必须激活 **PLC project properties**（PLC 项目属性）中 **Common**（通用）类别中的“Allow implicit checks for compiled libraries”（允许对已编译的库进行隐式检查）选项，然后才能将其保存并安装为库。此外，您还必须为受保护的库输入编译器定义“checks_in_libs”。

将单个 POU 排除在检查之外

使用 attribute 'no check' [▶ 751]（属性“no_check”），您可以在项目中禁用对特殊 POU 的检查。

对话框添加用于隐式检查的 POU

可用函数

监控函数	类型
CheckBounds	边界检查： 适当处理违反字段边界的情况（例如，设置错误标志或更改字段索引）。
CheckDivDInt	除法检查： 监控除数值，以避免除以 0 的情况。
CheckDivLInt	
CheckDivReal	
CheckDivLReal	
CheckRangeSigned	范围检查： 在运行时监控子范围类型的范围边界。适用于数据类型 DINT/UDINT
CheckRangeUnsigned	
CheckLRangeSigned	L 范围检查： 在运行时监控子范围类型的范围边界。适用于数据类型 LINT/ULINT
CheckLRangeUnsigned	
CheckPointer	指针检查： 对于该函数，您必须自己完成整个实现代码。请参见 POU CheckPointer 的帮助页面。该函数的目的是监控传输的指针是否指向有效的内存地址，以及引用的内存区的方向是否与指针指向的变量类型相匹配。如果 2 个条件都满足，则返回指针本身。否则，CheckPointer 应执行适当的故障排除。同样，CheckPointer 也可以监控 REFERENCE TO 类型的变量

另请参见：

- 引用编程：属性“no_check” [▶ 751]

7.19.1.1 边界检查（POU CheckBounds）

用于检查数组边界的函数：CheckBounds

该监控函数的目的是妥善处理数组越界的情况。例如，响应越界可能是设置一个错误标志或更改数组索引。这个检查仅在数组索引是变量时进行。如果数组索引一直有问题，会导致编译错误。一旦为数组变量赋值，TwinCAT 就会隐式调用该函数。

在插入该函数后，自动生成的代码就会在声明部分和实现部分出现。

不要更改声明部分

为了保持监控函数的正常运行，不得修改声明部分。唯一的例外是添加局部变量。

标准实现

声明部分：

```
// Implicitly generated code : DO NOT EDIT
FUNCTION CheckBounds : DINT
VAR_INPUT
    index, lower, upper: DINT;
END_VAR
```

实现部分：

```
// Implicitly generated code : Only an Implementation suggestion
{noflow}
IF index < lower THEN
    CheckBounds := lower;
ELSIF index > upper THEN
    CheckBounds := upper;
ELSE
    CheckBounds := index;
END_IF
{flow}
```

在调用该函数时，会向其分配以下输入参数：

- 索引：数组元素的索引
- 下限：数组范围的下限
- 上限：数组范围的上限

返回值是数组元素的索引，但要在有效范围内。否则，TwinCAT 将根据超出的界限返回上限或下限。

示例实现

声明部分：

```
// Implicitly generated code : DO NOT EDIT
FUNCTION CheckBounds : DINT
VAR_INPUT
    index, lower, upper: DINT;
END_VAR
// User defined local variables
VAR
    sMessageLow   : STRING := 'CheckBounds: Index too low (%d)';
    sMessageHigh  : STRING := 'CheckBounds: Index too high (%d)';
END_VAR
```

实现部分：

```
{noflow}
// Index too low
IF index < lower THEN
    CheckBounds := lower;
    // Increase global counter
    GVL_CheckBounds.nTooLow := GVL_CheckBounds.nTooLow + 1;
    // Log message
    ADSLOGDINT(msgCtrlMask := ADSLOG_MSGTYPE_WARN,
               msgFmtStr   := sMessageLow,
               dintArg     := index);
```

```
// Index too high
ELSIF index > upper THEN
    CheckBounds := upper;
    // Increase global counter
    GVL_CheckBounds.nTooHigh := GVL_CheckBounds.nTooHigh + 1;
    // Log message
    ADSLOGDINT(msgCtrlMask := ADSLOG_MSGTYPE_WARN,
               msgFmtStr   := sMessageHigh,
               dintArg     := index);

// Index OK
ELSE
    CheckBounds := index;
END_IF
{flow}
```

与标准实现一样，在此示例实现中，数组越界会通过返回上限或下限来纠正。此外，如果数组访问超出了定义的范围，全局计数器会增加。因此，全局计数器可表示整个项目中上限或下限越界的次数。此外，Tc2_System 库中的函数 ADSLOGDINT 可用于在消息窗口中发出数组越界的警告。

应用示例

在下面列出的程序中，索引超过了aSample 数组的定义上限。CheckBounds 函数会纠正这个超出数组边界的访问。

```
PROGRAM MAIN
VAR
    aSample : ARRAY[0..7] OF BOOL;
    nIndex  : INT := 10;
END_VAR
```

```
aSample[nIndex] := TRUE;
```

在本示例中，CheckBounds 函数使索引 10 变为 aSample (7) 的数组范围的上限。然后，aSample[7] 这个元素被赋值为TRUE。这样，函数就纠正了超出有效范围的数组访问。但是，务必同时校正造成错误的原因，即访问位置 10 的元素。为了避免这种自动校正的错误访问一直未被发现，在本示例中，会在消息窗口中显示警告，如上文所述。

错误/异常分类的实现示例

全局变量列表“GVL_Exc”：

```
VAR_GLOBAL
    nCheckBounds               : INT;
    nCheckBounds_OutOfBounds   : INT;
    myException                : __SYSTEM.ExceptionCode;
END_VAR
```

“CheckBounds”h函数：

```
// Implicitly generated code : DO NOT EDIT
FUNCTION CheckBounds : DINT
VAR_INPUT
    index, lower, upper        : DINT;
END_VAR
```

```
// Only an implementation suggestion
{noflow}
GVL_Exc.nCheckBounds := GVL_Exc.nCheckBounds + 1;

// Index too low
IF index < lower THEN
    CheckBounds                      := lower;
    GVL_Exc.myException              := __SYSTEM.ExceptionCode.RTSEXCPT_ARRAYBOUNDS;
    GVL_Exc.nCheckBounds_OutOfBounds := GVL_Exc.nCheckBounds_OutOfBounds + 1;

// Index too high
ELSIF index > upper THEN
    CheckBounds                      := upper;
    GVL_Exc.myException              := __SYSTEM.ExceptionCode.RTSEXCPT_ARRAYBOUNDS;
    GVL_Exc.nCheckBounds_OutOfBounds := GVL_Exc.nCheckBounds_OutOfBounds + 1;

// Index OK
ELSE
    CheckBounds := index;
END_IF
```

另请参见：

- 使用功能块进行隐式检查 [▶ 151]
- 对象用于隐式检查的 POU [▶ 151]
- 数据类型 __SYSTEM.ExceptionCode [▶ 681]

7.19.1.2 除法检查（POU：CheckDivDInt、CheckDivLInt、CheckDivReal、CheckDivLReal）

避免除数值为“0”的函数：CheckDivDInt、CheckDivLInt、CheckDivReal 和 CheckDivLReal

函数 CheckDivDInt、CheckDivLInt、CheckDivReal 和 CheckDivLReal 可用于避免除以 0 的情况。如果您将这些函数集成到 PLC 项目中，则会在代码中出现的每个除法之前调用它们。

不要更改声明部分

为了保持监控函数的正常运行，不得修改声明部分。唯一的例外是添加局部变量。

函数 CheckDivReal 的标准实现

声明部分：

```
// Implicitly generated code : DO NOT EDIT
FUNCTION CheckDivReal : REAL
VAR_INPUT
    divisor : REAL;
END_VAR
```

实现部分：

```
// Implicitly generated code : Only an Implementation suggestion
{noflow}
IF divisor = 0 THEN
    CheckDivReal := 1;
ELSE
    CheckDivReal := divisor;
END_IF
{flow}
```

DIV 运算符使用函数 CheckDivReal 中的输出作为除数。在下面的示例程序中，在执行除法之前，通过函数 CheckDivReal 将除数“fDivisor”的值从隐式启动时的“0”改为“1”，从而避免了除以 0 的情况。因此，除法的结果是 799。

```
PROGRAM MAIN
VAR
    fResult   : REAL;
    fDivident : REAL := 799;
    fDivisor  : REAL := 0;
END_VAR
fResult := fDivident / fDivisor;
```

7.19.1.3 范围/LRange 检查（POU：CheckRangeSigned、CheckRangeUnsigned、CheckLRangeSigned、CheckLRangeUnsigned）

用于监控子范围类型的范围边界的函数

- CheckRangeSigned 检查 SINT、INT 和 DINT 类型的子范围类型。
- CheckRangeUnsigned 检查 BYTE、WORD、DWORD、USINT、UINT、UDINT 类型的子范围类型。
- CheckLRangeSigned 检查 LINT 类型的子范围类型。
- CheckLRangeUnsigned 检查 LWORD、ULINT 类型的子范围类型。

这些监控函数的目的是适当处理违反范围边界的情况。举例而言，对违规行为的响应可能是设置错误标志或更改值。当将子范围类型的值分配给变量时，函数会被隐式调用。

不要更改声明部分

为了保持监控函数的正常运行，不得修改声明部分。唯一的例外是添加局部变量。

在调用该函数时，会将以下输入参数传输给它：

- 值：要分配给子范围类型的变量的值。
- 下限：范围的下限
- 上限：范围的上限

返回值是赋值本身，前提是它在有效范围内。否则，将返回上限或下限，具体取决于违反下限的情况。

赋值 i := 10*y 现在被隐式替换为 i := CheckRangeSigned(10*y, -4095, 4095);

例如，如果 y 的值为“1000”，变量 i 的赋值不是在原始代码中指定的“10*1000=10000”，而是范围上限的值，即“4095”。

这同样适用于函数 CheckRangeUnsigned、CheckLRangeSigned 和 CheckLRangeUnsigned。

如果没有可用的范围监控函数，则在运行时不会使用相应的变量进行子范围验证！在这种情况下，您可以为子范围类型为 DINT/UDINT 的变量分配介于 -2147483648 和 +2147483648 之间或介于 0 和 4294967295 之间的任意值。然后，您可以为子范围类型为 LINT/ULINT 的变量分配介于 9223372036854775808 和 +9223372036854775807 之间或介于 0 和 18446744073709551615 之间的任意值。

无限循环

当您集成字段监控函数时，可能会出现无限循环。例如，如果 FOR 循环的计数器变量属于子范围类型，而循环范围离开了定义的子范围，就会出现这种情况。

无限循环的示例：

```
VAR
    nVar : DINT(0..10000);
    ...
END_VAR
```

```
FOR nVar := 0 TO 10000 DO
    ...
END_FOR
```

程序永远不会离开 FOR 循环，因为监控函数 CheckRangeSigned 可以防止 nVar 被设为大于 10000 的值。

函数 CheckRangeSigned 的标准实现

如果给有符号子范围类型的 DINT 变量赋值，则必须自动调用函数 CheckRangeSigned。该函数将赋值限制在变量声明中指定的子范围内，在 ST 中的默认实现方式如下：

声明部分：

```
// Implicitly generated code : DO NOT EDIT
FUNCTION CheckRangeSigned : DINT
VAR_INPUT
    value, lower, upper : DINT;
END_VAR
```

实现：

```
// Implicitly generated code : Only an Implementation suggestion
{noflow}
IF (value < lower) THEN
    CheckRangeSigned := lower;
ELSEIF(value > upper) THEN
    CheckRangeSigned := upper;
ELSE
    CheckRangeSigned := value;
END_IF
{flow}
```

7.19.1.4 指针检查（POU CheckPointer）

指针的监控函数 CheckPointer

使用此函数可在运行时监控指针的内存访问。与其他监控函数不同，CheckPointer 没有默认实现。用户必须自己执行实现！

CheckPointer 函数的目的是检查传输的指针是否指向有效的内存地址，以及引用的内存区的方向是否与指针指向的变量类型相匹配。如果 2 个条件都满足，则返回指针本身。否则，函数应执行适当的错误处理。

不要更改声明部分

为了保持监控函数的正常运行，不得修改声明部分。唯一的例外是添加局部变量。

对于 THIS 指针和 SUPER 指针，不会隐式调用监控函数。

函数 CheckPointer 对 REFERENCE 类型变量的作用与指针变量相同。

模板

声明：

```
// Implicitly generated code : DO NOT EDIT
FUNCTION CheckPointer : POINTER TO BYTE
VAR_INPUT
    ptToTest : POINTER TO BYTE;
    iSize    : DINT;
    iGran    : DINT;
    bWrite   : BOOL;
END_VAR
```

实现（不完整！）：

```
// No standard way of implementation. Fill your own code here
{noflow}
CheckPointer := ptToTest;
{flow}
```

在调用该函数时，TwinCAT 会将以下输入参数传输给它：

- ptToTest：指针的目标地址
- iSize：引用变量的大小（以字节为单位）
- iGran：引用变量的粒度（以字节为单位），即引用变量中包含的最大非结构化数据类型。
- bWrite：访问的类型（TRUE = 写入访问，FALSE = 读取访问）

如果检查结果为正，则返回的输入指针将保持不变（ptToTest）。

示例：

在下面的实现示例中，一旦识别到无效指针，就会在 TwinCAT 输出窗口中显示 1 条消息。该实现可识别各种类型的无效指针。不过，它无法识别所有无效指针。

```
// Implicitly generated code : DO NOT EDIT
FUNCTION CheckPointer : POINTER TO BYTE
VAR_INPUT
    ptToTest : POINTER TO BYTE;
    iSize    : DINT;
    iGran    : DINT;
    bWrite   : BOOL;
END_VAR
```

```
IF ptToTest=0 THEN
    ADSLOGSTR(ADSLOG_MSGTYPE_ERROR OR ADSLOG_MSGTYPE_STRING,'CheckPointer failed due to invalid dest
ination address.','');
ELSIF iSize<=0 THEN
    ADSLOGSTR(ADSLOG_MSGTYPE_ERROR OR ADSLOG_MSGTYPE_STRING,'CheckPointer failed due to invalid size
.','');
ELSIF iGran<=0 THEN
    ADSLOGSTR(ADSLOG_MSGTYPE_ERROR OR ADSLOG_MSGTYPE_STRING,'CheckPointer failed due to invalid gran
ularity.','');
// -> Please note that the following memory area check is time consuming:
//ELSIF F_CheckMemoryArea(pData:=ptToTest,nSize:=DINT_TO_UDINT(iSize)) = E_TcMemoryArea.Unknown THEN
//    ADSLOGSTR(ADSLOG_MSGTYPE_ERROR OR ADSLOG_MSGTYPE_STRING,'CheckPointer failed due to unknown me
```

```
mory area.','');
END_IF
CheckPointer := ptToTest;
```

⚠ 谨慎

无效指针的后果

无效指针的后果通常是，一旦通过该指针进行任何访问，运行时就会停止。CheckPointer 函数通常无法避免这种情况。实际上，该监控函数的目的是进行有效的原因诊断。

7.20 面向对象的编程

TwinCAT 3 支持面向对象的编程（OOP），并为此提供以下功能和对象：

- 功能块（对象功能块 [▶ 159]）
- 方法（方法对象 [▶ 161]）
- 属性（属性对象 [▶ 166]）
- 接口（对象接口 [▶ 172]，接口的实现 [▶ 183]）
- 继承
  - 将功能块定义为其他功能块的扩展 [▶ 177]
  - 将结构定义为其他结构的扩展 [▶ 182]
  - 将接口定义为其他接口的扩展 [▶ 182]
- 方法调用 [▶ 184]
- ABSTRACT 概念 [▶ 185]

面向对象的编程的基本思路

在面向对象的编程中，软件被划分为多个对象。与 1 个对象相关的所有描述都合并在 1 个元素中（例如，功能块）。描述包括与对象相关的数据和程序。此外，还可以通过方法和属性来定义对象的访问接口。

因此，这种编程方法开发的对象可以自主重复使用，与特定条件无关。这些元素可用于 1 种或多种应用程序，而无需改动。

面向对象的编程的优点

面向对象的编程方法有很多优点。

通过将软件划分为多个对象，可以开发出**清晰**且**结构**良好的应用程序。因此，应用程序和各个元素均易于**理解**且易于**扩展**。编程对象的**可重用性**能够为应用程序的开发和维护节约**时间**和**成本**。

一般说明/建议

- 在使用面向对象的编程时，务必注意为各个应用程序找到正确的“面向对象的程度”。
  - 借助 OOP，软件被划分为多个对象，这样就可以开发出清晰、结构良好且可重复使用的软件。
  - 但是，如果划分的对象过于详细，则会影响程序的可理解性。
  - 示例：使用继承功能可以将声明和实现传递给子类，从而供子类重复使用。此外，将编程对象划分为一般类和特殊类也有助于理解各个编程对象。如果对“基本”和“扩展”的划分是一致的，那么继承就属于这种情况。现实世界中的 1 个例子是基本电机和具有附加功能的更加特殊的电机。但是，如果使用大量的继承级别，而且，各个继承级别的功能变得不明确或不精确，则应用程序可能会变得难以理解和维护。
- 如果方法可以提供返回值，则这些返回值应在方法中进行设置，并在调用方法时进行评估。
- 如果方法、函数或属性会返回结构化返回类型（例如，结构或功能块），则应将这些返回类型声明为“REFERENCE TO <structured type>”。
  - 将结构化数据作为直接返回类型（“<structured type>”）返回的效率很低，因为数据会被复制多次。
  - 如果改用“REFERENCE TO <structured type>”，一方面，由于数据不会复制，效率变得更高；另一方面，在调用方法/函数/属性时可以直接访问结构化数据类型的单个元素。
  - 有关更多信息和示例，请参见对象方法 [▶ 161]描述。

- 面向对象的实现的编程应基于事件。
  - 不应无故循环调用程序元素。
  - 通常情况下，当某个事件发生时调用程序元素是合理的。
- 通过未分配的接口变量进行的方法或属性调用与 NullPointer 调用的效果相同。在使用接口变量之前，您必须为其分配 1 个相应的功能块实例（实现接口的功能块的实例）。
  - 为了确保接口变量不对应 NullPointer，可在使用接口变量前检查其是否不等于 0（例如，IF (iSample <> 0) THEN iSample.METH (); END_IF）。

OOP 和 UML

是否必须同时使用面向对象的编程（OOP）和 UML？

- 联合使用 OOP 和 UML 有很多好处（见下一个问题），不过，这并不是强制性用法。应用程序的面向对象的编程也可以不使用 UML。同样，UML 也可用于不基于面向对象编程的 PLC 项目。

同时使用 OOP 和 UML 有哪些好处？

- 为了充分发挥 OOP 的作用，应在实现之前设计和创建面向对象软件的结构（例如，哪些类可用、它们之间的关系如何、它们提供哪些功能等）。在编程之前、期间和之后，文档有助于理解、分析和维护软件。
- 作为软件的分析、设计和文档工具，UML 为规划、创建和记录应用程序提供了多种选择。UML 尤其适合面向对象的实现，因为模块化软件特别适合借助图形语言来表示。
- 例如，类图可用于分析和创建程序结构。软件结构的模块化程度越高，类图的使用就越便捷且高效（例如，独立功能块的图形表示，用单独的方法提供功能等）。
- 状态图可用于指定具有离散事件的系统的顺序。软件结构的面向对象和面向事件的一致性越强，状态机的设计就越透明且有效（例如，模块/系统的行为基于具有状态（例如，启动、生产、暂停）的状态模型；在状态中调用封装在方法中的相应功能（例如，启动、执行、暂停）等）。

另请参见：

- 文档资料：TF1910 TC3 UML
- 文档示例：示例：控制分拣设备的面向对象的程序

7.20.1 对象功能块

功能块是 1 个 POU [▶ 71]，它在执行时会返回 1 个或多个值。在下一次执行之前，输出变量和内部变量的值将会保留。这意味着，如果使用相同的输入变量重复调用该功能块，它可能不会返回相同的输出值。

在 PLC 项目树中，功能块 POU 具有后缀（FB）。功能块的编辑器由声明部分和实现部分组成。

功能块总是通过实例调用，实例是功能块的副本。

除了在 IEC 61131-3 中描述的功能之外，在 TwinCAT 中，功能块还可用于以下面向对象的编程功能：

- 扩展功能块（扩展功能块 [▶ 177]）
- 实现接口（实现接口 [▶ 183]）
- 方法（对象方法 [▶ 161]）
- 属性（对象属性 [▶ 166]）

声明部分的第 1 行包含以下声明：

```
FUNCTION_BLOCK <access specifier> <function block> | EXTENDS <function block> |
  IMPLEMENTS <comma-separated list of interfaces>
```

8 字节对齐

TwinCAT 3 引入了 8 字节对齐方式。如果数据作为整个内存块与其他控制器或软件组件进行交换，请确保对齐方式适当（请参见 对齐方式 [▶ 731]）。

在功能块中自动创建接口元素

有 2 种方法可以在此功能块中自动生成实现功能块的接口元素。

1. 如果您在创建新功能块时在对话框 **Add**（添加）中的字段 **Implements**（实现）中指定了 1 个接口，则 TwinCAT 也会自动将该接口的方法和属性添加到该功能块中。
2. 如果现有功能块实现 1 个接口，则您可以使用命令 **Implement Interface**（实现接口），以便在功能块中生成接口元素。在项目树中的功能块的上下文菜单中可以找到命令 **Implement Interface**（实现接口）。

另请参见：

- TC3 用户界面文档：命令：实现界面 [▶ 987]
- 接口的实现 [▶ 183]

调用功能块

调用总是通过功能块的实例进行。如果调用 1 个功能块，则只有相应实例的值会发生变化。

实例的声明：

<instance> : <function block>;

在实现部分中，功能块的变量的访问方式如下：

<instance>.<variable>

- 从功能块实例的外部，您只能访问功能块的输入和输出变量，而不能访问内部变量。
- 除非您已在全局范围内声明实例，否则对功能块实例的访问仅限于声明实例的 POU。
- 您可以在调用实例时为功能块变量分配所需的值。

示例：

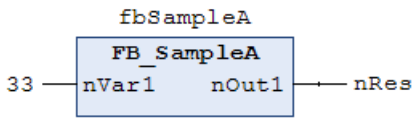
功能块 FB_SampleA 有 INT 类型的输入变量 nVar1 和输出变量 nOut1。在下面的示例中，从 MAIN 程序可以调用变量 nVar1。

ST：

```
PROGRAM MAIN
VAR
    fbSampleA : FB_SampleA;
END_VAR
```

```
fbSampleA.nVar1 := 33; (* FB_SampleA is called and the value 33 is assigned to the variable nVar1 *)
fbSampleA(); (* FB_SampleA is called, that's necessary for the following access to the output variab
le *)
nRes := fbSampleA.nOut1 (* the output variable nOut1 of the FB1 is read *)
```

FBD：



在调用过程中分配变量值：

在基于文本的语言 IL 和 ST 中，您在调用功能块时可以直接为输入变量和/或输出变量赋值。

使用 := 为输入变量赋值

使用 => 为输出变量赋值

示例：

在调用计时器功能块的实例 fbTimer 时，需要对输入变量 IN 和 PT 进行赋值。然后，将计时器的输出变量 Q 分配给变量 bVarA

```
PROGRAM MAIN
VAR
    fbTimer : TOF;
    bIn     : BOOL;
    bVarA   : BOOL;
END_VAR
```

```
fbTimer(IN := bIn, PT := t#300ms);
bVarA := fbTimer.Q;
```

如果您通过输入助手添加功能块实例，并且在 **Input Assistant**（输入助手）对话框中已启用选项 **Insert with arguments**（带参数插入），则 TwinCAT 会添加带有所有输入和输出变量的调用。您只需添加所需的赋值。在上述示例中，TwinCAT 会添加如下调用：CMD_TMR (IN:= , PT:= , Q=>)。

使用局部变量上的属性 **“is_connected”**，您可以在功能块实例中调用时确定特定输入是否会从外部接收赋值。

7.20.2 对象方法

符号： 

方法是对 IEC 61131-3 标准的扩展，也是面向对象的编程中用于数据封装的一种手段。1 个方法包含 1 个声明和 1 个实现。但是，与函数不同的是，方法不是一个独立的程序块，而是从属于某一功能块或程序。方法可以访问上级编程块的所有有效变量。

可以在程序或功能块下面添加方法。请使用 **“Project > Add object > Method”**（项目 > 添加对象 > 方法）命令打开 **“Add method ”**（添加方法）对话框。

您可以使用接口来组织方法。有关更多信息，请参见：接口的实现 [▶ 183]

如果将编程块下面的方法复制并粘贴到某一接口下面，或将该方法移动至该接口下面，那么这个方法中的实现内容会自动被移除。

方法说明

- 方法的所有数据均为临时数据，仅在执行方法时有效（堆栈变量）。这意味着每次调用方法时，TwinCAT 都会重新初始化您在该方法中声明的所有变量和功能块。
- 与函数一样，方法也可以返回 1 个返回值。
- 根据 IEC 61131-3 标准，方法可以像正常函数一样有附加输入和输出。您可以在调用方法时分配输入和输出。
  - **TwinCAT 3.1.4026 版本开始：**没有明确指定初始值的输入在调用方法时必须赋值。有明确初始值的输入在调用方法时可以选择赋值或忽略。
- 在方法的实现部分，允许访问功能块实例或程序变量。
- 使用 THIS 指针指向自己的实例。
- 无法访问方法中功能块的 VAR_TEMP 变量。
- 您可以声明 VAR_INST 变量，当再次调用方法时这些变量不会重新初始化（另请参见“实例变量”一章）。
- 通过使用返回类型“REFERENCE TO <structured type>”，您可以在调用方法时直接访问该方法返回的结构体数据类型的单个元素。有关更多信息，请参见“在方法/函数/属性调用期间，访问结构化返回类型的单个元素 [▶ 164]”部分。
- 原则上，在方法中可以访问功能块的 VAR_IN_OUT 变量。由于此类访问存在潜在风险，应谨慎使用。有关更多信息，请参见“在方法/转换中访问功能块的 VAR_IN_OUT 变量 [▶ 165]”部分。
- 在接口中定义的方法只能声明输入、输出和 VAR_IN_OUT 变量，但不得包含具体的实现代码。

示例：

以下示例中的代码会使 TwinCAT 将方法的返回值和输出写入局部声明的变量中。

功能块 FB_Sample 的方法“Method1”的声明部分：

```
METHOD Method1 : BOOL
VAR_INPUT
    nIn1  : INT;
    bIn2  : BOOL;
```

```
END_VAR
VAR_OUTPUT
    fOut1 : REAL;
    sOut2 : STRING;
END_VAR
```

```
// <method implementation code>
```

MAIN 程序：

```
PROGRAM MAIN
VAR
    fbSample      : FB_Sample;
    bReturnValue  : BOOL;
    nLocalInput1  : INT;
    bLocalInput2  : BOOL;
    fLocalOutput1 : REAL;
    sLocalOutput2 : STRING;
END_VAR
```

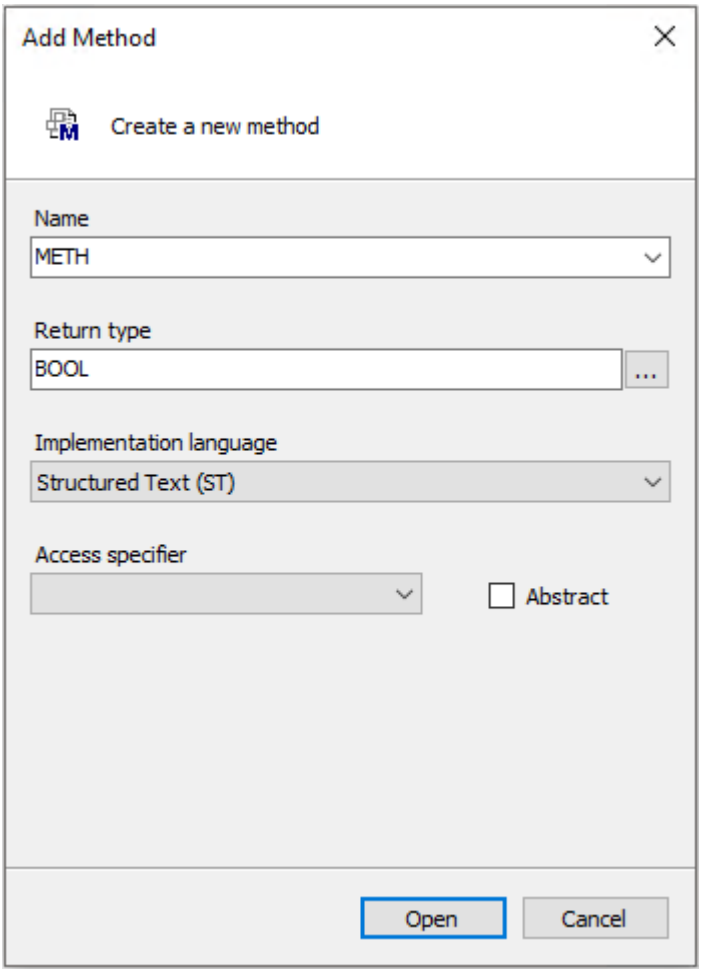
```
bReturnValue := fbSample.Method1(nIn1  := nLocalInput1,
                                 bIn2  := bLocalInput2,
                                 fOut1 => fLocalOutput1,
                                 sOut2 => sLocalOutput2);
```

创建对象方法

1. 在 PLC 项目树的 **Solution Explorer**（解决方案资源管理器）中，选择功能块或程序。
2. 在上下文菜单中，选择命令 **Add** > **Method...**（添加 > 方法……）
   ⇨ 对话框 **Add Method**（添加方法）会打开。
3. 输入名称并选择返回类型、实现语言，还可选择访问修饰符
4. 点击 **Open**（打开）。

⇨ 对象被添加到 PLC 项目树中，并在编辑器中打开。编辑器由顶部的声明编辑器和底部的实现部分组成。

对话框添加方法



名称	方法的名称 如果在功能块下方尚未插入标准方法 FB_Init 和 FB_Exit，则在选择列表中会提供这 2 种方法。如果是派生功能块，选择列表还会提供基本功能块的所有方法。
返回类型	返回值的类型
实现语言	实现语言选择列表

访问修饰符

访问修饰符	规范数据访问 • **公开**：访问不受限制（相当于没有指定访问修饰符）。 • **私有**：对方法的访问分别限制在功能块或程序中。 • **受保护**：对方法的访问分别限制在程序或功能块及其导数中。 • **内部**：对方法的访问限制在命名空间（库）中。 除了这些访问修饰符外，您还可以为方法手动添加 FINAL 修饰符： • **最终**：不允许覆盖功能块导数中的方法。这意味着在可能存在的子类中不能覆盖/扩展该方法。
抽象	☑：表示方法没有实现，实现由派生 FB 提供。 有关 ABSTRACT 关键字的背景信息，请参见 ABSTRACT 概念 [▶ 185] 下方内容。

在 PLC 项目树的解决方案资源管理器中，访问修饰符与 PUBLIC 不同的方法会用信号符号标记。

访问修饰符	对象图标	信号符号
私有		（锁）
受保护		（星）
内部		（心）

如果您将 POU 中的方法复制或移动到接口中，TwinCAT 会自动删除所包含的实现。

功能块的特殊方法

FB_init	默认为隐式声明。也可以提供显式声明。 包含功能块的初始化代码，如在功能块的声明部分所定义。 （方法 FB_init、FB_reinit 和 FB_exit [▶ 786]）
FB_reinit	需要显式声明。当功能块实例被复制时（例如，在在线更改期间）进行调用。它会重新初始化新的实例模块。 （方法 FB_init、FB_reinit 和 FB_exit [▶ 786]）
FB_exit	需要显式声明。 在再次下载、重置之前或者在线更改所有移动或删除的实例期间，调用功能块的每个实例。 （方法 FB_init、FB_reinit 和 FB_exit [▶ 786]）
属性和接口属性	它们都包含 1 个 Set 和/或 1 个 Get 访问器方法。 （对象接口属性 [▶ 176]，对象属性 [▶ 166]）

调用方法

语法：

<return value variable> := <POU name>.<method name>(<method input name> := <variable name> (, <further method input name> := <variable name>)*);

当您调用方法时，您可以将传输参数分配给方法的输入变量。在此过程中，请遵守声明。只需指定输入变量的名称即可，无需考虑它们在声明中的顺序。

示例：

以下示例中的代码会使 TwinCAT 将方法的返回值和输出写入局部声明的变量中。

功能块 FB_Sample 的“Method1”的声明部分：

```
METHOD Method1 : BOOL
VAR_INPUT
    nIn1  : INT;
    bIn2  : BOOL;
END_VAR
VAR_OUTPUT
    fOut1 : REAL;
    sOut2 : STRING;
END_VAR
```

```
// <method implementation code>
```

MAIN 程序：

```
PROGRAM MAIN
VAR
    fbSample      : FB_Sample;
    bReturnValue  : BOOL;
    nLocalInput1  : INT;
    bLocalInput2  : BOOL;
    fLocalOutput1 : REAL;
    sLocalOutput2 : STRING;
END_VAR
```

```
bReturnValue := fbSample.Method1(nIn1  := nLocalInput1,
                                 bIn2  := bLocalInput2,
                                 fOut1 => fLocalOutput1,
                                 sOut2 => sLocalOutput2);
```

递归方法调用

在实现过程中，方法可以直接通过THIS指针直接调用自身，也可以通过功能块索引间接调用自身。

- 借助 THIS 指针直接调用相关功能块实例：
  <Variable für Rückgabewert> := THIS^.<Methodenname> (<Parameter>);
- 通过一个局部变量调用方法，该变量会临时实例化受影响的功能块。
  <Variable für Rückgabewert> := <FB-Instanz>.<Methodenname> (<Parameter>);

如果方法中有递归调用，会出现编译器警告。如果使用特定的编译指示 {attribute 'estimated-stack-usage' := '<estimated stack size in bytes>'} ，编译器警告将受到抑制。

实现示例请参见“属性“estimated-stack-usage” [▶ 740]”。

若要以递归的方式调用方法，仅指定方法名称是不够的。如果仅指定了方法名称，则会输出编译器错误（需要指定“程序名称、函数或功能块实例，而不是.....”）。

在方法/函数/属性调用期间，访问结构化返回类型的单个元素

在调用方法、函数或属性时，可使用以下实现直接访问方法/函数/属性返回的结构化数据类型的单个元素。例如，结构化数据类型是 1 个结构或 1 个功能块。

1. 方法/函数/属性的返回类型被定义为“REFERENCE TO <structured type>”（而不只是“<structured type>”）。
2. 请注意，对于这种返回类型 – 例如，如果要返回结构化数据类型的 FB-local实例 – 则必须使用引用运算符 REF=，而不是“正常”赋值运算符 :=。

本节中的声明和示例涉及属性的调用。不过，它们同样适用于其他提供返回值的调用（例如，方法或函数）。

示例

结构 ST_Sample（结构化数据类型）的声明：

```
TYPE ST_Sample :
STRUCT
    bVar  : BOOL;
    nVar  : INT;
END_STRUCT
END_TYPE
```

功能块 FB_Sample 的声明：

```
FUNCTION_BLOCK FB_Sample
VAR
    stLocal     : ST_Sample;
END_VAR
```

返回类型为“REFERENCE TO ST_Sample”的属性 FB_Sample.MyProp 的声明：

```
PROPERTY MyProp : REFERENCE TO ST_Sample
```

属性 FB_Sample.MyProp 的 Get 方法的实现：

```
MyProp REF= stLocal;
```

属性 FB_Sample.MyProp 的 Set 方法的实现：

```
stLocal := MyProp;
```

在主程序 MAIN 中，调用 Get 和 Set 方法：

```
PROGRAM MAIN
VAR
    fbSample    : FB_Sample;
    nSingleGet  : INT;
    stGet       : ST_Sample;
    bSet        : BOOL;
    stSet       : ST_Sample;
END_VAR
```

```
// Get - single member and complete structure possible
nSingleGet := fbSample.MyProp.nVar;
stGet      := fbSample.MyProp;

// Set - only complete structure possible
IF bSet THEN
    fbSample.MyProp REF= stSet;
    bSet            := FALSE;
END_IF
```

通过将属性 MyProp 的返回类型声明为“REFERENCE TO ST_Sample”，并在该属性的 Get 方法中使用引用运算符 REF=，在调用该属性时可以直接访问返回结构化数据类型的单个元素。

```
VAR
    fbSample    : FB_Sample;
    nSingleGet  : INT;
END_VAR
```

```
nSingleGet := fbSample.MyProp.nVar;
```

如果仅将返回类型声明为“ST_Sample”，则首先必须将属性返回的结构分配给局部结构实例。然后，根据局部结构实例可以查询单个结构元素。

```
VAR
    fbSample    : FB_Sample;
    stGet       : ST_Sample;
    nSingleGet  : INT;
END_VAR
```

```
stGet      := fbSample.MyProp;
nSingleGet := stGet.nVar;
```

在方法/转换/属性中访问功能块的 VAR_IN_OUT 变量

原则上，在功能块的方法、转换或属性中可以访问功能块的 VAR_IN_OUT 变量。对于这种类型的访问，请注意以下几点：

- 如果从 FB 外部调用功能块的主体或某个动作，编译器应确保功能块的 VAR_IN_OUT 变量与该调用一起分配。
- 如果调用功能块的方法、转换或属性，则不是这种情况，因为 FB 的 VAR_IN_OUT 变量不能在方法、转换或属性调用中分配。因此，在 VAR_IN_OUT 变量被分配给有效引用之前，通过调用方法/转换/属性可能会导致对 VAR_IN_OUT 变量的访问。由于这将意味着运行时的无效访问，因此，在方法、转换或属性中访问 FB 的 VAR_IN_OUT 变量具有潜在风险。

因此，如果在方法、转换或属性中访问 FB 的 VAR_IN_OUT 变量，就会发出带有 ID C0371 的以下警告：

“警告：从外部环境 <Method/Transition/Property> 访问在 <POU> 中声明的 VAR_IN_OUT <Var>”

对此项警告的适当回应可以是，在访问之前，检查方法/转换/属性中的 VAR_IN_OUT 变量。运算符 __ISVALIDREF 可用于此项检查，以确定引用是否指向 1 个有效值。如果此项检查已启用，则可以认为用户已经意识到在方法/转换/属性中访问 FB 的 VAR_IN_OUT 变量时可能存在的风险。检查引用被认为是对这一风险的适当处理。因此，通过属性“禁用警告”可以抑制相应的警告。

方法实现示例如下所示。

功能块 FB_Sample：

```
FUNCTION_BLOCK FB_Sample
VAR_IN_OUT
    bInOut : BOOL;
END_Var
```

方法 FB_Sample.MyMethod：

```
METHOD MyMethod
VAR_INPUT
END_VAR
```

```
// The warning can be disabled here as the user is aware of the risk that the reference may not be
valid by checking its validity
{warning disable C0371}

// Checking the VAR_IN_OUT reference, leave the current method in case of invalid reference
IF NOT __ISVALIDREF(bInOut) THEN
    RETURN;
END_IF

// Access to VAR_IN_OUT reference (only if the reference was confirmed as valid before)
bInOut := NOT bInOut;

// The warning may be restored at the end of the access area
{warning restore C0371}
```

另请参见：

- 面向对象的编程 [▶ 158]
- 对象接口 [▶ 172]
- 接口的实现 [▶ 183]
- 扩展功能块 [▶ 177]
- 方法调用 [▶ 184]
- ABSTRACT 概念 [▶ 185]
- 引用编程 > 实例变量 - VAR INST [▶ 628]

7.20.3 对象属性

符号： 

属性是 IEC 61131-3 标准的扩展，是 1 种面向对象的编程的手段。它由访问器方法 Get 和 Set 组成。当对实现该属性的功能块进行读取或写入访问时，TwinCAT 会自动调用这些方法。

创建属性的对象

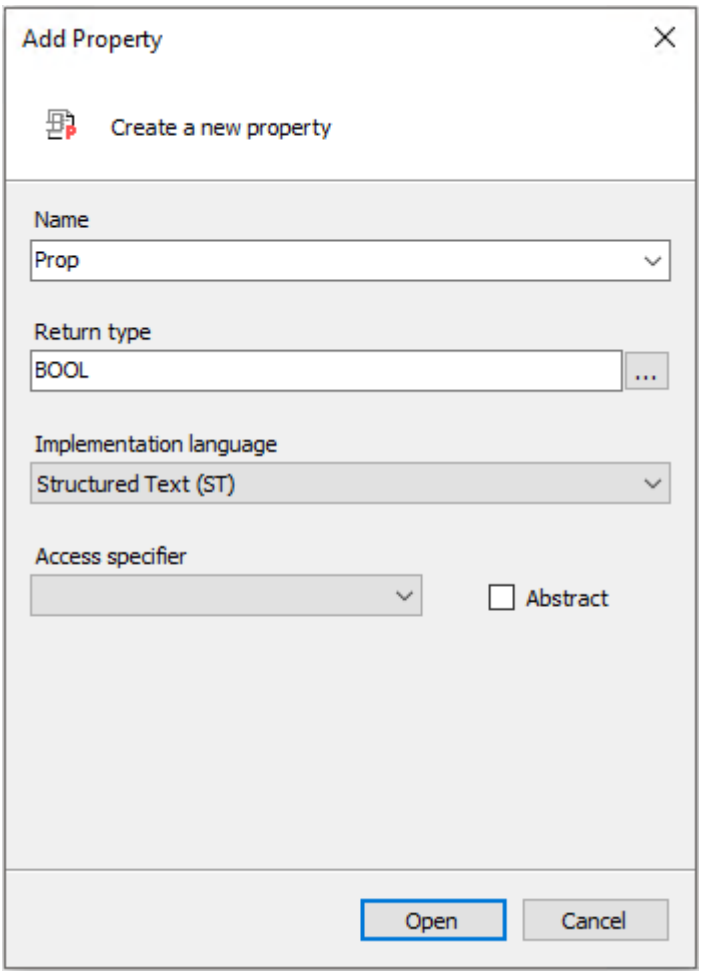
1. 在 PLC 项目树的 **Solution Explorer**（解决方案资源管理器）中，选择功能块或程序。
2. 在上下文菜单中，选择命令 **Add** > **Property...**（添加 > 属性……）

⇨ **Add Property**（添加属性）对话框会打开。

3. 输入名称并选择返回类型、实现语言，还可选择访问修饰符。
4. 点击 **Open**（打开）。

⇨ 对象被添加到 PLC 项目树中，并在编辑器中打开。

对话框添加属性



名称	属性的名称
返回类型	返回值的类型（默认类型或结构化类型）
实现语言	实现语言选择列表

访问修饰符

访问说明符	规范数据访问 • **公开**：访问不受限制（相当于没有指定访问修饰符）。 • **私有**：对属性的访问分别限制在功能块或程序中。 • **受保护**：对属性的访问分别限制在程序或功能块及其导数中。 • **内部**：对属性的访问限制在命名空间（库）中。 除了这些访问修饰符外，您还可以为属性手动添加 FINAL 修饰符： **最终**：不允许覆盖功能块导数中的属性。这意味着在可能存在的子类中不能覆盖/扩展该属性。
抽象	☑：表示属性没有实现，实现由派生 FB 提供。 有关 ABSTRACT 关键字的背景信息，请参见 ABSTRACT 概念 [▶ 185] 下方内容。

在 PLC 项目树的解决方案资源管理器中，访问修饰符与 PUBLIC 不同的属性会用信号符号标记。

访问修饰符	对象图标	信号符号
私有		 （锁）
受保护		 （星）
内部		 （心）

此外，属性还可以包含局部变量。但是，属性不能包含附加输入。与函数或方法相比，它也不能包含附加输出。

如果您将 POU 中的属性复制或移动到接口中，TwinCAT 会自动删除所包含的实现。

Get 和 Set 访问器

TwinCAT 会在 PLC 项目树中的属性对象下方自动添加 Get 和 Set 访问器方法。您可以使用命令 **Add**（添加）明确添加它们。

当对属性进行写入访问时（即您将属性名称用作输入参数时），TwinCAT 会调用 Set 访问器。

当对属性进行读取访问时（即您将属性名称用作输出参数时），TwinCAT 会调用 Get 访问器。

请注意，您必须实现访问器方法才能访问该属性。

实现示例

功能块 FB_Sample 的声明

```
FUNCTION_BLOCK FB_Sample
VAR
    nVar : INT;
END_VAR
```

```
nVar := nVar + 1;
```

属性 nValue 的声明

```
PROPERTY PUBLIC nValue : INT
```

访问器方法 FB_Sample.nValue.Set 的实现

```
nVar := nValue;
```

访问器方法 FB_Sample.nValue.Get 的实现

```
nValue := nVar;
```

调用属性 nValue

```
PROGRAM MAIN
VAR
    fbSample : FB_Sample;
END_VAR
```

```
fbSample();
If fbSample.nValue > 500 THEN
    fbSample.nValue := 0;
END_IF;
```

如果您只想将属性用于读取访问或写入访问，您可以删除未使用的访问器。

您可以在以下位置为访问器方法添加访问说明符：

- 通过手动输入访问器的声明部分。
- 在 **Add 'get' accessor**（添加“get”访问器）或 **Add 'set' accessor**（添加“set”访问器）对话框中，当您使用 **Add**（添加）命令明确添加访问器时。

针对 REFERENCE 类型属性的兼容性警告

在 TC3.1 Build 4022 中，使用返回类型“REFERENCE TO <…>”定义的属性的调用行为发生了变化。

对于使用“:=”的写入访问，在版本 < 3.1.4022.0 时，会调用 set 访问器，以便写入引用。不过，在版本 >= 3.1.4022.0 时，会调用 get 访问器，以便写入值。如要在版本 >= 3.1.4022.0 时分配引用，务必使用引用赋值运算符“REF= [▶ 575]”。

在线模式下的属性监控

编译指示可用于在线模式下的属性监控，应该将其添加到定义属性的顶部（属性“monitoring” [▶ 749]）：

- {attribute 'monitoring':= 'variable'}：每次访问该属性时，TwinCAT 都会将实际值存储到 1 个变量中，并显示该变量的值。如果代码不再访问该属性，该值可能会过时。
- {attribute 'monitoring' := 'call'}：每次显示值时，TwinCAT 都会调用 Get 访问器的代码。如果该代码包含副作用，副作用将由监控执行。

您可以使用以下功能监控属性：

- 内联监控
  要求：在类别 **TwinCAT > PLC Programming Environment > Text Editor**（TwinCAT > PLC 编程环境 > 文本编辑器）中的 TwinCAT 选项中，选项卡 **Monitoring**（监控）中的选项 **Enable Inline-Monitoring**（启用内联监控）已被激活。
- 监视列表（使用监视列表 [▶ 209]）

在方法/函数/属性调用期间，访问结构化返回类型的单个元素

在调用方法、函数或属性时，可使用以下实现直接访问方法/函数/属性返回的结构化数据类型的单个元素。例如，结构化数据类型是 1 个结构或 1 个功能块。

1. 方法/函数/属性的返回类型被定义为“REFERENCE TO <structured type>”（而不只是“<structured type>”）。
2. 请注意，对于这种返回类型 - 例如，如果要返回结构化数据类型的 FB-local实例 - 则必须使用引用运算符 REF=，而不是“正常”赋值运算符 :=。

本节中的声明和示例涉及属性的调用。不过，它们同样适用于其他提供返回值的调用（例如，方法或函数）。

示例

结构 ST_Sample（结构化数据类型）的声明：

```
TYPE ST_Sample :
STRUCT
    bVar  : BOOL;
    nVar  : INT;
END_STRUCT
END_TYPE
```

功能块 FB_Sample 的声明：

```
FUNCTION_BLOCK FB_Sample
VAR
    stLocal     : ST_Sample;
END_VAR
```

返回类型为“REFERENCE TO ST_Sample”的属性 FB_Sample.MyProp 的声明：

```
PROPERTY MyProp : REFERENCE TO ST_Sample
```

属性 FB_Sample.MyProp 的 Get 方法的实现：

```
MyProp REF= stLocal;
```

属性 FB_Sample.MyProp 的 Set 方法的实现：

```
stLocal := MyProp;
```

在主程序 MAIN 中，调用 Get 和 Set 方法：

```
PROGRAM MAIN
VAR
    fbSample    : FB_Sample;
    nSingleGet  : INT;
    stGet       : ST_Sample;
    bSet        : BOOL;
    stSet       : ST_Sample;
END_VAR
```

```
// Get - single member and complete structure possible
nSingleGet := fbSample.MyProp.nVar;
stGet      := fbSample.MyProp;

// Set - only complete structure possible
IF bSet THEN
    fbSample.MyProp REF= stSet;
    bSet            := FALSE;
END_IF
```

通过将属性 MyProp 的返回类型声明为“REFERENCE TO ST_Sample”，并在该属性的 Get 方法中使用引用运算符 REF=，在调用该属性时可以直接访问返回结构化数据类型的单个元素。

```
VAR
    fbSample    : FB_Sample;
    nSingleGet  : INT;
END_VAR
```

```
nSingleGet := fbSample.MyProp.nVar;
```

如果仅将返回类型声明为“ST_Sample”，则首先必须将属性返回的结构分配给局部结构实例。然后，根据局部结构实例可以查询单个结构元素。

```
VAR
    fbSample    : FB_Sample;
    stGet       : ST_Sample;
    nSingleGet  : INT;
END_VAR
```

```
stGet      := fbSample.MyProp;
nSingleGet := stGet.nVar;
```

在方法/转换/属性中访问功能块的 VAR_IN_OUT 变量

原则上，在功能块的方法、转换或属性中可以访问功能块的 VAR_IN_OUT 变量。对于这种类型的访问，请注意以下几点：

- 如果从 FB 外部调用功能块的主体或某个动作，编译器应确保功能块的 VAR_IN_OUT 变量与该调用一起分配。
- 如果调用功能块的方法、转换或属性，则不是这种情况，因为 FB 的 VAR_IN_OUT 变量不能在方法、转换或属性调用中分配。因此，在 VAR_IN_OUT 变量被分配给有效引用之前，通过调用方法/转换/属性可能会导致对 VAR_IN_OUT 变量的访问。由于这将意味着运行时的无效访问，因此，在方法、转换或属性中访问 FB 的 VAR_IN_OUT 变量具有潜在风险。

因此，如果在方法、转换或属性中访问 FB 的 VAR_IN_OUT 变量，就会发出带有 ID C0371 的以下警告：

“警告：从外部环境 <Method/Transition/Property> 访问在 <POU> 中声明的 VAR_IN_OUT <Var>”

对此项警告的适当回应可以是，在访问之前，检查方法/转换/属性中的 VAR_IN_OUT 变量。运算符 __ISVALIDREF 可用于此项检查，以确定引用是否指向 1 个有效值。如果此项检查已启用，则可以认为用户已经意识到在方法/转换/属性中访问 FB 的 VAR_IN_OUT 变量时可能存在的风险。检查引用被认为是对这一风险的适当处理。因此，通过属性“禁用警告”可以抑制相应的警告。

方法实现示例如下所示。

功能块 FB_Sample：

```
FUNCTION_BLOCK FB_Sample
VAR_IN_OUT
    bInOut : BOOL;
END_Var
```

方法 FB_Sample.MyMethod：

```
METHOD MyMethod
VAR_INPUT
END_VAR
```

```
// The warning can be disabled here as the user is aware of the risk that the reference may not be
valid by checking its validity
{warning disable C0371}

// Checking the VAR_IN_OUT reference, leave the current method in case of invalid reference
IF NOT __ISVALIDREF(bInOut) THEN
    RETURN;
END_IF

// Access to VAR_IN_OUT reference (only if the reference was confirmed as valid before)
bInOut := NOT bInOut;

// The warning may be restored at the end of the access area
{warning restore C0371}
```

初始化示例：

在下面的示例中，1 个输入变量和 1 个属性由 1 个功能块初始化，该功能块的 FB_init 方法 [▶ 787]带有 1 个附加参数。

功能块 FB_Sample：

```
FUNCTION_BLOCK FB_Sample
VAR_INPUT
    nInput           : INT;
END_VAR
VAR
    nLocalInitParam  : INT;
    nLocalProp       : INT;
END_VAR
```

方法 FB_Sample.FB_init：

```
METHOD FB_init : BOOL
VAR_INPUT
    bInitRetains : BOOL;  // if TRUE, the retain variables are initialized (warm start / cold start)
    bInCopyCode  : BOOL;  // if TRUE, the instance afterwards gets moved into the copy code (online
change)
    nInitParam   : INT;
END_VAR
```

```
nLocalInitParam := nInitParam;
```

属性 FB_Sample.nMyProperty 和相关的 Set 函数：

```
PROPERTY nMyProperty : INT
```

```
nLocalProp := nMyProperty;
```

MAIN 程序：

```
PROGRAM MAIN
VAR
    fbSample  : FB_Sample(nInitParam := 1) := (nInput := 2, nMyProperty := 3);
    aSample   : ARRAY[1..2] OF FB_Sample[(nInitParam := 4), (nInitParam := 7)]
                            := [(nInput := 5, nMyProperty := 6), (nInput := 8, nMyProperty := 9)];
END_VAR
```

初始化结果：

- fbSample
  - nInput = 2
  - nLocalInitParam = 1
  - nLocalProp = 3
- aSample[1]
  - nInput = 5
  - nLocalInitParam = 4
  - nLocalProp = 6
- aSample[2]
  - nInput = 8

- nLocalInitParam = 7
- nLocalProp = 9

另请参见：

- 对象接口属性 [▶ 176]
- 面向对象的编程 [▶ 158]
- 扩展功能块 [▶ 177]
- 引用编程 > 方法 FB_init、FB_reinit 和 FB_exit [▶ 786]

7.20.4 对象接口

符号： 

关键字：接口

接口是 1 种面向对象的编程的工具。对象 **Interface**（接口）描述了 1 组方法和属性原型。这里的原型是指方法和属性只包含声明，而不包含实现。

这样，您可以通过相同的方式使用具有共同属性的各种功能块。

您可以在 **Interface**（接口）对象中添加对象 **Interface property**（接口属性）和 **Interface method**（接口方法）。

创建对象接口

1. 在 PLC 项目树的 **Solution Explorer**（解决方案资源管理器）中，选择 1 个文件夹。
2. 在上下文菜单中，选择命令 **Add > Interface...**（添加 > 接口……）
   ⇨ **Add Interface**（添加接口）对话框会打开。
3. 输入名称，还可选择要扩展的接口。
4. 点击 **Open**（打开）。

⇨ 接口被添加到 PLC 项目树中，并在编辑器中打开。

对话框添加接口

名称	接口名称

继承

高级	 ：扩展在输入字段中输入或通过输入助手  选择的接口。这意味着新接口扩展的所有接口方法在新接口中也可用。

接口应用

1. 通过编译器检查接口声明

- 在 1 个接口（例如，I_Sample）中，您可以声明与该接口相关的方法和属性（包括返回类型、输入等）。
- 在与该接口相对应并应提供相应的方法和属性的功能块中，您可以实现接口 I_Sample。
  - 功能块在其声明部分的 IMPLEMENTS 列表中包含接口（例如，FB_Sample IMPLEMENTS I_Sample）。
  - 1 个功能块可以实现 1 个或多个接口（例如，FB_Sample IMPLEMENTS I_Sample1、I_Sample2）。
- 实现接口的功能块必须包含在该接口中定义的所有方法和属性（接口方法和属性）。方法和属性的声明必须与接口中的声明完全一致（名称、返回类型、输入、输出）。
- 功能块在接口方法和接口属性中添加功能块特定代码。如果 1 个接口由多个功能块实现，您可以在不同的功能块中使用带有参数和不同实现代码的相同方法。
- 对于实现 1 个或多个接口的功能块，编译器会检查该功能块是否符合相应的接口声明。如果接口中的元素声明与功能块中的元素声明不同，或者如果接口中包含更多元素，而功能块中没有包含这些元素，则编译器会报错。

2. 通过接口变量调用功能块实例的方法和属性

除了通过编译器自动验证接口声明之外，您还可以使用接口来通过接口变量调用功能块实例的接口方法或接口属性。

- 首先，将接口（例如 iSample : I_Sample;）和正确实现接口的功能块实例化（请参见用例 1：通过编译器检查接口声明）。
- 然后，您可以将接口变量分配给正确实现接口的功能块的每个实例。如果接口变量尚未分配，则该变量在在线模式下包含值 0。
- 在最后一步中，您可以通过接口变量调用接口方法或属性。为接口所引用的功能块调用该方法或属性。
- 这种实现可以通过接口变量以一致的方式使用不同但相似的功能块。例如，根据项目状态，您可以为接口变量分配 1 个特定的功能块实例，这样，尽管根据项目状态使用的是不同的功能块实例，但对接口方法和属性的调用是相同的。

在通过接口变量调用方法或属性之前，务必将接口类型变量分配给功能块实例。

注意

- 您不能在接口中声明变量。接口没有实现部分，也没有动作。仅可定义方法和属性的集合，其中包含声明代码，但没有实现代码。
- 接口类型变量是对功能块实例的引用。TwinCAT 总是将定义为接口类型的变量视为引用。

接口引用和在线更改

- 接口引用从 TwinCAT 3.0 build 3100 开始自动重定向，因此，即使是在线更改，也能始终引用正确的接口。这需要额外的代码和时间，根据受影响对象的数量，可能会引发问题。因此，在执行在线更改之前，程序员会看到受影响的变量和接口引用的数量，以便决定是继续执行在线更改还是中止更改。

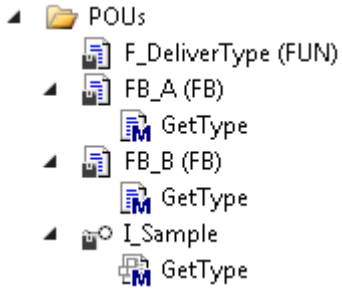
检查接口变量

- 运算符 __ISVALIDREF 仅可用于 REFERENCE TO 类型的操作数。此运算符不能用于检查接口变量。如要检查接口变量是否已分配给 1 个功能块实例，您可以检查接口变量是否等于 0（IF iSample <> 0 THEN …）。

对接口变量进行扩展监控

- 从 TwinCAT 3.1 Build 4024 版本开始提供用于监控/调试接口变量的高级选项。监控区域（声明编辑器、监视列表）中的接口变量可以扩展。然后，在接口变量下会显示当前分配的 FB 实例的符号路径和在线数据。

示例 1



接口声明：

- 您已在您的项目中添加接口 I_Sample。在接口中添加返回类型为 STRING 的方法 GetType。
- I_Sample 和 GetType 不包含实现代码。方法 GetType 只包含所需的（变量）声明（例如，返回类型）。稍后，您可以在实现接口 I_Sample 的功能块中编出方法 GetType。

```
INTERFACE I_Sample
```

方法 I_Sample.GetType：

```
METHOD GetType : STRING
```

接口实现：

- 如果您随后在项目中添加功能块并在对话框 **Add**（添加）的字段 **Implements**（实现）中输入接口 I_Sample，TwinCAT 也会将方法 GetType 自动添加到该功能块中。在此处，您可以在方法中实现功能块特定代码。
- 功能块 FB_A 和 FB_B 都可以实现接口 I_Sample：

```
FUNCTION_BLOCK FB_A IMPLEMENTS I_Sample
```

```
FUNCTION_BLOCK FB_B IMPLEMENTS I_Sample
```

- 因此，这 2 个功能块都必须包含 1 个名称为 GetType 且返回类型为 STRING 的方法。否则，编译器会报错（请参见接口应用 [▶ 172]部分中的用例 1）。

方法 FB_A.GetType：

```
METHOD GetType : STRING
```

```
GetType := 'FB_A';
```

方法 FB_B.GetType：

```
METHOD GetType : STRING
```

```
GetType := 'FB_B';
```

接口应用：

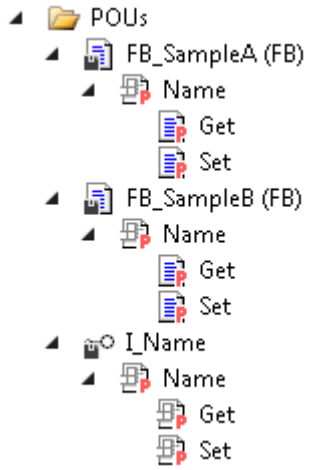
- 函数 F_DeliverType 包含接口 I_Sample 类型的输入变量的声明。在该函数中，通过接口变量 iSample 可以调用接口方法 GetType。在这种情况下，调用 FB_A.GetType 还是 FB_B.GetType 取决于传输的功能块类型（请参见接口应用 [▶ 172]部分中的应用 2）。

```
FUNCTION F_DeliverType : STRING
VAR_INPUT
    iSample : I_Sample;
END_VAR

F_DeliverType := iSample.GetType();
```

- 您可以将实现接口 I_Sample 的功能块实例（例如，FB_A 和 FB_B）分配给函数 F_DeliverType 的输入变量。
- 函数调用示例：
  - 如果将功能块实例 fbA 传输到函数 F_DeliverType，由于接口变量 iSample 指向功能块实例 fbA，因此，在函数内部将会调用方法 fbA.GetType。该方法调用会产生返回值“FB_A”，而该返回值又会由函数 F_DeliverType 返回，并在主程序中被分配给变量 sResultA。
  - 由于在函数 F_DeliverType 内部调用方法 fbB.GetType，因此 sResultB 会接收到值“FB_B”。

```
PROGRAM MAIN
VAR
    fbA       : FB_A;
    fbB       : FB_B;
    sResultA  : STRING;
    sResultB  : STRING;
END_VAR

sResultA := F_DeliverType(iSample := fbA); // call with instance of type FB_A
sResultB := F_DeliverType(iSample := fbB); // call with instance of type FB_B
```

示例 2



接口声明：

- 您已在您的项目中添加接口 I_Name。在接口中添加返回类型为 STRING 的属性 Name。该属性有 Get 和 Set 访问器方法。Get 访问器可用于从实现接口的功能块中读取任何对象的名称。Set 访问器可用于将名称写入该功能块。
- I_Name 和 Name 不包含实现代码。属性 Name 只包含所需的声明（返回类型）。因此，您不能在接口定义中处理 Get 和 Set 方法，但您可以在稍后实现接口 I_Name 的功能块中进行此类处理。

```
INTERFACE I_Name
```

属性 I_Name.Name：

```
PROPERTY Name : STRING
```

接口实现：

- 功能块 FB_SampleA 和 FB_SampleB 可实现接口 I_Name。
- 如果已经指定接口，例如，在对话框 **Add**（添加）中创建功能块时，TwinCAT 会在功能块 FB_SampleA 和 FB_SampleB 下自动添加带有 Get 和 Set 方法的属性 Name。
- 例如，您可以编辑功能块下方的访问器方法，以便读取变量 sVar1，从而获得对象的名称。在实现相同接口 I_Name 的 FB_SampleB 中，您可以实现 Get 方法代码，然后该代码会返回另一个对象的名称。Set 方法可用于将 MAIN 程序提供的名称（“abc”）写入功能块 FB_SampleB 中。
- 功能块 FB_SampleA 和 FB_SampleB 分别可实现接口 I_Name：

```
FUNCTION_BLOCK FB_SampleA IMPLEMENTS I_Name
VAR
    sVar1 : STRING := 'My name is A.';
END_VAR

FUNCTION_BLOCK FB_SampleB IMPLEMENTS I_Name
VAR
    sVar2 : STRING := 'My name is B.';
END_VAR
```

- 因此，这 2 个功能块都必须包含 1 个名称为 Name 且返回类型为 STRING 的属性。属性必须有 Get 和 Set 方法。否则，编译器会报错（请参见接口应用 [▶ 172]部分中的用例 1）。

FB_SampleA.Name.Get：

```
Name := sVar1;
```

FB_SampleA.Name.Set：

```
sVar1 := Name;
```

FB_SampleB.Name.Get：

```
Name := sVar2;
```

FB_SampleB.Name.Set：

```
sVar2 := Name;
```

接口应用：

- 功能块的属性既可以通过相应的功能块实例进行访问，也可以通过类型 I_Name 的接口变量进行访问。通过接口变量进行访问的前提条件是，该变量事先已被分配给 1 个实现接口 I_Name 的特定功能块实例。

```
PROGRAM MAIN
VAR
    iName     : I_Name;

    fbSampleA : FB_SampleA;
    sNameA    : STRING;       // will be 'My name is A.'

    fbSampleB : FB_SampleB;
    sNameB    : STRING;       // will be 'My name is B.' after first cycle
                              // and will be 'New name' afterwards
END_VAR

// assign FB instance fbSample1 to interface variable
iName := fbSampleA;

// access to name property of fbSample1 via interface variable (Get)
sNameA := iName.Name;

// access to name property of fbSample2 via FB instance (Get and Set)
sNameB := fbSampleB.Name;
fbSampleB.Name := 'New name';
```

另请参见：

- 实现接口 [▶ 183]
- 扩展接口 [▶ 182]

7.20.4.1 对象界面方法

符号： 

接口方法是 1 种面向对象的编程的手段。通过命令 **Add > Method...**（添加 > 方法……）可以将 **Interface method**（接口方法）对象添加到接口中

如果已在接口下添加方法，您只能在该方法中添加变量声明（输入、输出和输入/输出变量）并将其实例化。

只有当 1 个功能块“实现”了属于该方法的接口后，程序代码才能被添加到方法中。然后，TwinCAT 会将该方法添加到功能块下。

另请参见：

- 对象接口 [▶ 172]
- 对象方法 [▶ 161]
- 接口的实现 [▶ 183]

7.20.4.2 对象接口属性

符号： 

接口属性是 1 种面向对象的编程的工具。通过命令 **Add > Property...**（添加 > 属性……）可以将对象 **Interface property**（接口属性）添加到接口中，以便使用 Get 和/或 Set 访问器方法扩展对接口的描述。接口属性中的访问器方法不包含实现代码。如果您删除 Set 访问器，则该属性只能进行读取访问，而不能进行写入访问。

Get 访问器可用于对属性进行读取访问。
Set 访问器可用于对属性进行写入访问。

如果属性没有 Get 和/或 Set，则可以使用 **Add**（添加）命令将该访问器添加到接口属性中。

如果您使用包含属性的接口来扩展功能块或程序，则 TwinCAT 会在 POU 下的 PLC 项目树中自动添加该接口以及相应的 Get 和/或 Set 访问器。然后，您可以在 Get 和/或 Set 访问器中实现代码。

另请参见：

- 对象接口 [▶ 172]
- 对象属性 [▶ 166]
- 接口的实现 [▶ 183]

7.20.5 扩展功能块

功能块的扩展基于面向对象的编程中的继承概念。为此，派生功能块“扩展”了基本功能块，因此，除了自身的属性和功能外，派生功能块基本上还获得（“继承”）了基本功能块的属性和功能。

在 TwinCAT 中，“功能块”一词可与“类”互换使用。因此，派生功能块可被称为“子类”，基本功能块可被称为“基类”。

请注意以下信息：

- 继承原则 [▶ 178]
- 继承元素的用例 [▶ 179]

每个基本功能块的扩展数量

每个基本功能块的扩展数量不受限制。因此，使用多个其他功能块可以对 1 个功能块进行扩展和自定义。

- 可能：
```
FUNCTION_BLOCK FB_Sub1 EXTENDS FB_Base
FUNCTION_BLOCK FB_Sub2 EXTENDS FB_Base
FUNCTION_BLOCK FB_Sub3 EXTENDS FB_Base
```

继承级别的数量

继承级别的数量不受限制。因此，使用其他功能块等可以对扩展另一个功能块的功能块进行自定义。

- 可能：
```
FUNCTION_BLOCK FB_Sub EXTENDS FB_Base
FUNCTION_BLOCK FB_SubSub EXTENDS FB_Sub
FUNCTION_BLOCK FB_SubSub EXTENDS FB_SubSub
```

不允许多重继承

功能块不允许多重继承。1 个功能块不能扩展多个其他功能块。

例外：1 个功能块可以实现多个接口，1 个接口可以扩展多个其他接口。

- 不可能：
```
FUNCTION_BLOCK FB_Sub EXTENDS FB_Base1、FB_Base2
```
- 可能：
```
FUNCTION_BLOCK FB_Sample IMPLEMENTS I_Sample1、I_Sample2
```
- 可能：
```
INTERFACE I_Sub EXTENDS I_Base_1、I_Base_2
```

重载

无法重载方法。因此，如果您在子类中覆盖或扩展基类方法（方法名相同），则方法声明必须与基类声明（访问修饰符、返回类型、变量接口）相匹配。

使用新功能块扩展基本功能块

✓ 当前打开的项目有 1 个基本功能块（例如，FB_Sample），使用 1 个新的功能块可以对该功能块进行扩展。

1. 在 PLC 项目树中选择 PLC 项目对象或子文件夹，然后，在上下文中选择命令菜单 **Add** > **POU...**（添加 > POU……）
   ⇨ **Add POU**（添加 POU）对话框会打开。
2. 在名称输入字段中输入新功能块的名称，例如，FB_SampleEx。
3. 选择 **Function Block**（功能块）。
4. 选择 **Extends**（扩展）并点击 ... 。
5. 在输入助手中，从项目下的 **Functionblocks**（功能块）类别中选择 POU(FB) 用作基本功能块（例如，“FB_Sample”），然后点击 **OK**（确定）。
6. 您还可以从组合框中为新功能块选择 **Access specifier**（访问说明符）。
7. 例如，从 **Implementation language**（实现语言）中选择“Structured Text (ST)”（结构化文本（ST））。
8. 点击 **Open**（打开）。

⇨ TwinCAT 会在 PLC 项目树中添加功能块 FB_SampleEx，并打开编辑器。第 1 行写道：
FUNCTION_BLOCK FB_SampleEx EXTENDS FB_Sample
功能块 FB_SampleEx 扩展了基本功能块 FB_Sample。

使用现有功能块扩展基本功能块

✓ 当前打开的项目有 1 个基本功能块（例如，“FB_Sample”），以及另一个尚未从基本功能块派生的功能块（例如，“FB_SampleEx”）。功能块 FB_SampleEx 用于扩展基本功能块，即：FB_SampleEx 用于扩展 FB_Sample。

1. 在 PLC 项目树中，双击功能块 FB_SampleEx。
   ⇨ POU 编辑器会打开。
2. 使用 EXTENDS FB_Sample 扩展顶行中的现有条目 FUNCTION_BLOCK FB_SampleEx。

⇨ 功能块 FB_SampleEx 扩展了基本功能块 FB_Sample。

另请参见：

- 对象功能块 [▶ 159]
- 对象属性 [▶ 166]
- 对象方法 [▶ 161]
- 对象动作 [▶ 78]
- 对象转换 [▶ 79]
- 引用编程：SUPER [▶ 632]
- 引用编程：THIS [▶ 633]

7.20.5.1 继承原则

继承的内容

派生功能块可继承在基本功能块中定义的所有数据、方法、属性、动作和转换。请注意，通过访问修饰符可以定义继承元素的访问选项。

继承元素的访问选项

子类能在多大程度上访问其范围内的继承方法或属性，这取决于在基类中定义方法或属性所使用的访问修饰符。

在基类中使用访问修饰符 PRIVATE 定义的方法和属性不能在子类的范围内调用，也不能被子类覆盖或扩展。如果子类的私有方法和属性在基类的实现中被调用，那么，只有在子类的实例中执行这些方法和属性时，子类才能使用这些方法和属性。

示例：

基类有 1 个 PUBLIC 方法和 1 个 PRIVATE 方法。PUBLIC 方法在实现过程中会调用 PRIVATE 方法。被调用的子类可以调用 PUBLIC 方法，从而同时隐式调用 PRIVATE 方法。但是，子类不能主动调用、覆盖或扩展 PRIVATE 方法。

以下访问修饰符可用于指定方法或属性的访问选项：

公开	对应于没有访问修饰符的规范。从功能块外部可以调用该元素（方法或属性）。因此，子类也可以访问该元素。
私有	对元素的访问仅限于功能块。从功能块外部无法访问。这意味着子类也无法访问该元素。因此，子类既不能调用该元素，也不能覆盖或扩展它。
受保护	对元素的访问仅限于功能块及其导数。子类可以访问该元素，因此，子类可以调用、扩展或覆盖它。从该“继承家族”外部无法访问。
内部	对元素的访问限制在命名空间（库）中。从命名空间外部无法访问。因此，从命名空间外部不能覆盖或扩展该元素。

扩展或覆盖继承元素

- 如果在基类中对元素使用了相应的访问修饰符，则派生功能块可以扩展或覆盖在基本功能块中定义的方法、属性、动作和转换。
- 为了能够扩展或覆盖 1 个元素，务必在子类中以与基类相同的方式声明该元素：
  - 相同的名称
  - 相同的访问修饰符（用于方法和属性）
  - 相同的变量接口（例如，方法输入/输出）
  - 相同的返回类型（用于方法和属性）
- 当 1 个元素被扩展或覆盖时，在子类中只有实现部分会被调整，以便调整元素的行为。

工程提示：为了扩展或覆盖功能块继承的方法、属性、动作和转换，提供以下工程支持：当您向派生功能块添加方法、属性等时，添加对话框的 **Name**（名称）（例如，**Add Method**（添加方法）、**Add Property**（添加属性））会提供 1 个下拉列表，其中包含在基本功能块中使用的方法、属性等的选择。例如，如果您在 **Add Method**（添加方法）对话框中选择了其中 1 个方法，该方法的其他声明设置（返回类型、访问修饰符）将自动应用到基类的方法声明中。在您确认对话框后，该方法将根据这些声明进行创建。您可以自定义方法的实现部分，使其符合所需的子类行为。

有关扩展和覆盖的更多信息，请参见“继承元素的用例 [▶ 179]“部分。

与扩展相关的其他考虑因素

- 在 TwinCAT 期望使用基本功能块类型的功能块的任何情况下均可使用派生功能块的实例。
- 派生功能块不得包含任何与基本功能块声明的名称相同的功能块变量。编译器会将这种情况报告为错误。
  唯一的例外：如果您在基本功能块中将变量声明为 VAR_TEMP，则派生功能块可以定义具有相同名称的变量。这样，派生功能块就不能再访问基本功能块的变量。
- 在派生功能块的范围内，通过使用 SUPER 指针可以直接寻址基本功能块的变量和元素（例如，方法、属性、动作、转换）。

7.20.5.2 继承元素的用例

一般来说，通过 3 种不同的方式可以使用子类对其有适当访问权限的继承元素（请参见“继承元素的访问选项 [▶ 178]”部分）：

- 可以不加改变地使用继承元素。
- 可以覆盖继承元素。
- 可以扩展继承元素。

下面将以“方法”元素为例，解释这 3 种用例。

不加改变地使用

- 要求：子类需要的实现与在基类方法中编程的实现完全相同。
- 实现：在这种情况下，**不会**为子类创建方法。
- 后果：子类使用基类的方法实现。
- 示例：
  - 基类必须控制用于执行流程的轴。

- 同样的要求也适用于子类：子类也必须控制该轴。
- 在这种情况下，**不会**为子类创建 ExecuteProcess 方法。如果为子类（fbSub.ExecuteProcess (...)）的实例调用该方法，则会自动调用该方法的基本实现（FB_Base.ExecuteProcess）。因此，子类可以从在基类中已经实现的实现中获益。

功能块 FB_Base：

```
FUNCTION_BLOCK FB_Base
VAR
    fbAxis  : FB_Axis;
END_VAR
```

方法 FB_Base.ExecuteProcess：

```
METHOD ExecuteProcess : BOOL
VAR_INPUT
    bExecuteProcess  : BOOL;
END_VAR
```

```
// Calling axis module by passing input parameter "bExecuteProcess" of this method to the input
parameter "bExecute" of method "Execute"
fbAxis.Execute(bExecute := bExecuteProcess);

// Setting the return value of this method as inverted error signal of the axis module
ExecuteProcess := NOT fbAxis.Error;
```

功能块 FB_Sub：

```
FUNCTION_BLOCK FB_Sub EXTENDS FB_Base
VAR
END_VAR
```

方法 FB_Sub.ExecuteProcess：

[does not exist]

覆盖

- 要求：与基类相比，子类对方法指令的要求完全不同。
- 实现：在这种情况下，会为子类创建方法，并在实现部分填入其他指令。与基类方法相比，只有实现部分不同，声明部分必须完全相同。
- 后果：子类使用自己的方法实现。子类已覆盖基类方法。
- 示例：
  - 基类必须控制用于执行流程的轴。
  - 相比之下，子类在流程执行过程中不必控制轴，但需控制气缸。
  - 在这种情况下，会为子类创建 ExecuteProcess 方法。该方法的实现部分采用所需的指令进行编程，其效果与基本实现完全不同。

功能块 FB_Base：

```
FUNCTION_BLOCK FB_Base
VAR
    fbAxis  : FB_Axis;
END_VAR
```

方法 FB_Base.ExecuteProcess：

```
METHOD ExecuteProcess : BOOL
VAR_INPUT
    bExecuteProcess  : BOOL;
END_VAR
```

```
// Calling axis module by passing input parameter "bExecuteProcess" of this method to the input
parameter "bExecute" of method "Execute"
fbAxis.Execute(bExecute := bExecuteProcess);

// Setting the return value of this method as inverted error signal of the axis module
ExecuteProcess := NOT fbAxis.Error;
```

功能块 FB_Sub：

```
FUNCTION_BLOCK FB_Sub EXTENDS FB_Base
VAR
    fbCylinder  : FB_Cylinder;
END_VAR
```

方法 FB_Sub.ExecuteProcess：

```
METHOD ExecuteProcess : BOOL
VAR_INPUT
    bExecuteProcess  : BOOL;
END_VAR
```

```
// Calling cylinder module by passing input parameter "bExecuteProcess" of this method to the input
parameter "bExecute" of method "Execute"
fbCylinder.Execute(bExecute := bExecuteProcess);

// Setting the return value of this method as inverted error signal of the cylinder module
ExecuteProcess := NOT fbCylinder.Error;
```

扩展

- 要求：子类既需要在基类中已实现的实现，也需要特定于子类的附加指令。
- 实现：在这种情况下，会为子类创建方法，并在实现部分填入额外需要的指令。在子类方法中的所需位置，通过 SUPER^.SampleMethod (...) 可以调用基类方法。该调用会执行原始基类方法。子类方法中的附加指令也会执行子类特别需要的附加指令。
- 后果：子类使用自己的附加实现以及基类的实现（通过调用 SUPER 指针）。子类已扩展基类方法。
- 示例：
  - 基类必须控制用于执行流程的轴。
  - 子类还须在流程执行期间控制 1 个轴。**此外**，子类必须控制 1 个气缸。
  - 在这种情况下，会为子类创建 ExecuteProcess 方法。该方法的实现部分采用所需的附加指令进行编程。使用 SUPER 指针（SUPER^.ExecuteProcess (...)）在子类方法序列中的适当点调用基类方法（FB_Base.ExecuteProcess）。因此，子类可以从在基类中已经实现的实现中获益。此外，它还可以使用子类所需的指令扩展或自定义实现。

功能块 FB_Base：

```
FUNCTION_BLOCK FB_Base
VAR
    fbAxis  : FB_Axis;
END_VAR
```

方法 FB_Base.ExecuteProcess：

```
METHOD ExecuteProcess : BOOL
VAR_INPUT
    bExecuteProcess  : BOOL;
END_VAR
```

```
// Calling axis module by passing input parameter "bExecuteProcess" of this method to the input
parameter "bExecute" of method "Execute"
fbAxis.Execute(bExecute := bExecuteProcess);

// Setting the return value of this method as inverted error signal of the axis module
ExecuteProcess := NOT fbAxis.Error;
```

功能块 FB_Sub：

```
FUNCTION_BLOCK FB_Sub EXTENDS FB_Base
VAR
    fbCylinder  : FB_Cylinder;
END_VAR
```

方法 FB_Sub.ExecuteProcess：

```
METHOD ExecuteProcess : BOOL
VAR_INPUT
    bExecuteProcess    : BOOL;
END_VAR
```

```
// Extension: Calling cylinder module by passing input parameter "bExecuteProcess" of this method to
the input parameter "bExecute" of method "Execute"
fbCylinder.Execute(bExecute := bExecuteProcess);

// Setting the return value of this method as inverted error signal of the cylinder module PLUS
calling the base method and analyzing its return value
ExecuteProcess := NOT fbCylinder.Error AND SUPER^.ExecuteProcess(bExecuteProcess :=
bExecuteProcess);
```

7.20.6 扩展结构

与功能块一样，结构也可以扩展。然后，除了自身的变量外，该结构还能获得基本结构的变量。

创建 1 个可扩展另一个结构的结构：

1. 在 PLC 项目树中，选择 PLC 项目对象或子文件夹。
2. 在上下文菜单中，选择命令 **Add** > **DUT...**（添加 > DUT……）
   ⇨ 对话框 **Add DUT**（添加 DUT）会打开。
3. 输入名称并选择 **Structure**（结构）作为数据类型。
4. 选择 **Extends**（扩展）选项并点击 [...] 按钮。
   ⇨ **Input Assistant**（输入助手）会打开。
5. 从类别 **Data Unit types**（数据单元类型）中选择新结构要扩展的结构。

⇨ 该结构扩展了基本结构。

不允许多重继承

结构不允许多重继承。1 个结构不可能扩展多个其他结构。

- 不可能：
  TYPE ST_Sub EXTENDS ST_Base1、ST_Base2 :
  STRUCT
  …

另请参见：

- 对象 DUT [▶ 68]

7.20.7 扩展接口

与功能块一样，接口也可以扩展。然后，除了自身的接口方法和属性外，该接口还能获得基本接口的接口方法和属性。

创建 1 个可扩展另一个接口的接口：

1. 在 PLC 项目树中，选择 PLC 项目对象或子文件夹。
2. 在上下文菜单中，选择命令 **Add** > **Interface…**（添加 > 接口……）
   ⇨ **Add Interface**（添加接口）对话框会打开。
3. 输入新界面的名称。
4. 选择 **Advanced**（高级）选项并点击 [...] 按钮。
5. **Input Assistant**（输入助手）会打开。
6. 从类别 **Interfaces**（接口）中选择新接口要扩展的接口。

⇨ 该接口扩展了基本接口。

允许多重继承

接口允许多重继承。1 个接口可能会扩展多个其他接口。

- 可能：
  INTERFACE I_Sub EXTENDS I_Base1、I_Base2

另请参见：

- 对象接口 [▶ 172]

7.20.8 接口的实现

接口的实现基于面向对象的编程的概念。通过通用接口，您能够以相似的方式使用不同但相似的功能块。

实现接口的功能块必须包含在该接口中定义的所有方法和属性（接口方法和接口属性）。这意味着：各个方法或属性的名称、返回类型、输入和输出必须完全相同。如果接口中的元素声明与功能块中的元素声明不同，或者如果接口中包含更多元素，而功能块中没有包含这些元素，则编译器会报错。

有关使用接口的更多信息和示例，请参见“对象接口 [▶ 172]”部分

当您创建 1 个实现接口的新功能块时，TwinCAT 会自动将该接口的所有方法和属性插入树中的新功能块下。

然后，如果您在接口中添加更多的方法或属性，TwinCAT 不会自动将这些元素添加到相关的功能块中。对于更新，您必须明确使用命令 **Implement interfaces**（实现接口）。

在执行该命令时，自动创建的方法或属性将带有编译指示属性，这将引发编译错误或警告。这将有助于您确保自动创建的元素不会在无意中保持为空。有关更多信息，请参见 命令：实现界面 [▶ 987] 的帮助页面。

在新功能块中实现接口

✓ 当前打开的项目至少有 1 个接口对象。

1. 在 PLC 项目树中选择 PLC 项目对象或子文件夹，然后，在上下文中选择命令菜单 **Add** > **POU...**（添加 > POU……）
   ⇨ **Add POU**（添加 POU）对话框会打开。
2. 在 **Name**（名称）输入字段中输入新功能块的名称，例如，“FB_SampleImp”。
3. 选择 Function block（功能块）。
4. 选择 **Implements**（实现）并点击 [...] 按钮。
5. 在输入助手中，从 **Interfaces**（接口）类别中选择接口（例如，“I_Itf1”），然后点击 **OK**（确定）。
6. 如要添加另一个接口，可再次点击 [...] 并选择另一个接口。
7. 您还可以从选择列表中为新功能块选择 **Access specifier**（访问说明符）。
8. 例如，从 **Implementation language**（实现语言）选择列表中选择“Structured Text (ST)”（结构化文本（ST））。
9. 点击 **Open**（打开）。

⇨ TwinCAT 会在 PLC 项目树中添加功能块 FB_SampleImp，并打开编辑器。第 1 行写道：
FUNCTION_BLOCK FB_SampleImp IMPLEMENTS I_Itf1
接口的方法和属性现已插入 PLC 项目树的功能块下，您现在可以在方法和属性的实现部分中输入程序代码。

在现有功能块中实现接口

✓ 当前打开的项目有 1 个功能块（例如，“FB_SampleImp”），以及至少 1 个接口对象（例如，“I_Itf1”）。

1. 在 PLC 项目树中，双击 POU FB_SampleImp。
   ⇨ POU 编辑器会打开。
2. 使用 IMPLEMENTS I_Itf1 扩展顶行中的现有条目 FUNCTION_BLOCK FB_SampleImp。
   ⇨ 功能块 FB_SampleImp 实现了接口 I_Itf1。
3. 在项目树中的功能块的上下文菜单中，通过执行命令 **Implement interfaces**（实现接口），在功能块中可以自动生成在接口中已定义但在功能块中尚未出现的元素（方法或属性）。

⇨ 接口的方法和属性现已插入 PLC 项目树的功能块下，您现在可以在方法和属性的实现部分中输入程序代码。

另请参见：

- 对象功能块 [▶ 159]
- TC3 用户界面文档：命令：实现界面 [▶ 987]

7.20.9 方法调用

如要实现方法调用，可将实际参数（参数）传输至接口变量。或者，也可以省略参数名称。

根据声明的访问修饰符，您可以通过以下方式调用方法：仅在其自身的命名空间内调用（INTERNAL），仅在其自身的编程块及其导数内调用（PROTECTED），或仅在其自身的编程块内调用（PRIVATE）。如果使用 PUBLIC 选项，则您可以在任何地方调用方法。

在实现过程中，1 个方法可以递归调用自身：直接使用 THIS 指针，或者使用分配的功能块的局部变量。

方法调用作为虚拟函数调用

继承可能会导致虚拟函数调用。虚拟函数调用允许程序源代码中的相同调用在运行时调用不同的方法。

在下列情况下，方法调用是动态绑定的：

- 您使用指向功能块的指针（例如，pFB^.SampleMethod）调用方法。
  在这种情况下，指针可以指向功能块类型的实例和所有派生功能块的实例。
- 您调用接口变量的方法（例如，iSample.SampleMethod）。
  接口可以引用实现此接口的功能块的所有实例。
- 1 个方法调用相同功能块的另一个方法。在这种情况下，该方法也可以调用同名扩展功能块的方法。
- 通过引用功能块来调用方法。在这种情况下，引用可以指向功能块类型的实例和所有派生功能块的实例。
- 您将基本功能块类型的 VAR_IN_OUT 变量分配给派生 FB 类型的实例。
  在这种情况下，变量可以指向功能块类型的实例和所有派生功能块的实例。

示例

- 功能块 FB_Sub1 和 FB_Sub2 分别扩展了功能块 FB_Base。
- FB_Base 实现了接口 I_Base，该接口定义了 Method1 方法。
- FB_Base 和 FB_Sub1 提供了方法 Method1。因此，FB_Sub1 覆盖或扩展了基类 FB_Base 的方法。
- FB_Sub2 没有提供该方法。功能块继续使用基类 FB_Base 的方法。
- 在 MAIN 程序中，基类 FB_Base 的实例、子类 FB_Sub1 的实例或子类 FB_Sub2 的实例被分配给 1 个接口变量和 1 个基类引用。分配哪个实例取决于变量 nVar 的值。
- 通过接口变量和引用变量可以调用方法 Method1。该方法调用是动态的，可针对不同的实例（fbBase、fbSub1、fbSub2）执行。各个实例之间的底层方法实现各不相同。
  - 针对实例 fbBase 执行 FB_Base.Method1 的实现。
  - 针对实例 fbSub1 执行 FB_Sub1.Method1 的实现，因为 FB_Sub1 覆盖或扩展了基类方法。
  - 针对实例 fbSub2 执行 FB_Base.Method1 的实现，因为子类 FB_Sub2 既没有覆盖也没有扩展基类方法，而是不加改变地使用该方法。

带有方法 Method1 的接口 I_Base：

```
INTERFACE I_Base
METHOD Method1
```

带有方法 Method1 的功能块 FB_Base：

```
FUNCTION_BLOCK FB_Base IMPLEMENTS I_Base
METHOD Method1
```

带有方法 Method1 的功能块 FB_Sub1：

```
FUNCTION_BLOCK FB_Sub1 EXTENDS FB_Base
METHOD Method1
```

没有自己的方法的功能块 FB_Sub2：

```
FUNCTION_BLOCK FB_Sub2 EXTENDS FB_Base
```

MAIN 程序：

```
PROGRAM MAIN
VAR
    nVar        : INT;
    fbBase      : FB_Base;
    fbSub1      : FB_Sub1;
    fbSub2      : FB_Sub2;
    iBase       : I_Base;
    refBase     : REFERENCE to FB_Base;
END_VAR
```

```
(* Choosing the desired instances via value of nVar:
    0 => fbBase
    1 => fbSub1
    2 => fbSub2 *)

IF nVar = 0 THEN
    iBase     := fbBase;
    refBase REF= fbBase;

ELSIF nVar = 1 THEN
    iBase     := fbSub1;
    refBase REF= fbSub1;

ELSIF nVar = 2 THEN
    iBase     := fbSub2;
    refBase REF= fbSub2;
END_IF

// Regarding each of the following two calls via interface and via reference:
    // If nVar is 0, FB_Base.Method1 will be called for instance fbBase
    // If nVar is 1, FB_Sub1.Method1 will be called for instance fbSub1
    // If nVar is 2, FB_Base.Method1 will be called for instance fbSub2

iBase.Method1();
refBase.Method1();
```

附加输出

根据 IEC 61131-3 标准，方法和正常函数可以声明附加输出。在调用方法时，变量会被分配给附加输出。

有关更多信息，请参见 对象函数 [▶ 73]。

递归调用方法

注意

递归主要用于处理递归数据类型，例如，链表。应谨慎使用递归。意外的深度递归可能会导致堆栈溢出，从而导致设备停机。

在实现过程中，方法可以通过以下任一方式调用自身

- 通过 THIS 指针直接调用，或者
- 使用基本功能块的局部功能块实例间接调用

通常，在进行这种递归调用时，会发出编译器警告。如果方法有编译指示 {attribute 'estimated-stack-usage' := '<estimated stack size in bytes>'}，则可以抑制编译器警告。实现示例请参见 属性 “estimated-stack-usage” [▶ 740] 章节。

另请参见：

- 对象方法 [▶ 161]
- 对象函数 [▶ 73]
- 扩展功能块 [▶ 177]
- 引用编程：SUPER [▶ 632]
- 引用编程：THIS [▶ 633]

7.20.10 ABSTRACT 概念

关键字 ABSTRACT 可用于功能块、方法和属性。它支持具有抽象层级的 PLC 项目的实现。
抽象是面向对象的编程的 1 个关键概念。不同的抽象层级包含一般或特定的实现方面。

TC3.1 Build 4024 及以上可用

抽象的应用

在抽象基类中实现不同类的基本函数或共性是非常有用的。您可以在非抽象子类中实现特定方面。
原则与接口的使用类似。接口对应于只包含抽象方法和属性的纯抽象类。抽象类也可以包含非抽象方法和属性。

关键字 ABSTRACT 的使用规则

- 不能将抽象功能块实例化。
- 抽象功能块可以包含抽象和非抽象方法和属性。
- 抽象方法或属性不包含实现（只有声明）。
- 如果功能块包含抽象方法或属性，则其本身必须是抽象的。
- 为了能够实现抽象方法或属性，务必对抽象功能块进行扩展。
- 因此：派生 FB 必须实现其基本 FB 的方法/属性，否则，它也必须被定义为抽象的。

示例

抽象基类：

```
FUNCTION_BLOCK ABSTRACT FB_System_Base
```

所有系统模块的共性均在该抽象基类中实现。为此，它包含非抽象属性“nSystemID”和抽象方法“Execute”：

```
PROPERTY nSystemID : UINT
```

```
METHOD ABSTRACT Execute
```

虽然“nSystemID”的实现对所有系统都是一样的，但是方法“Execute”的实现却因系统而异。

非抽象子类：

```
FUNCTION_BLOCK FB_StackSystem EXTENDS FB_System_Base
```

从基类派生出来的非抽象类是为特定系统实现的。该子类表示堆栈。由于它不是抽象的，因此，它必须实现定义特定堆栈执行的方法“Execute”：

```
METHOD Execute
```

7.20.11 示例

基本 OOP 示例

描述	本 PLC 示例说明了面向对象的编程（OOP）的一些基本功能。它具有以下元素/功能： • 功能块（FBs） • 方法 • 属性 • FB 继承/扩展 • 接口（ITF）实现和使用
示例项目	https://infosys.beckhoff.com/content/1033/tc3_plc_intro/Resources/7644034443.zip
更多信息	见 PLC 文档：面向对象的编程

扩展 OOP 示例

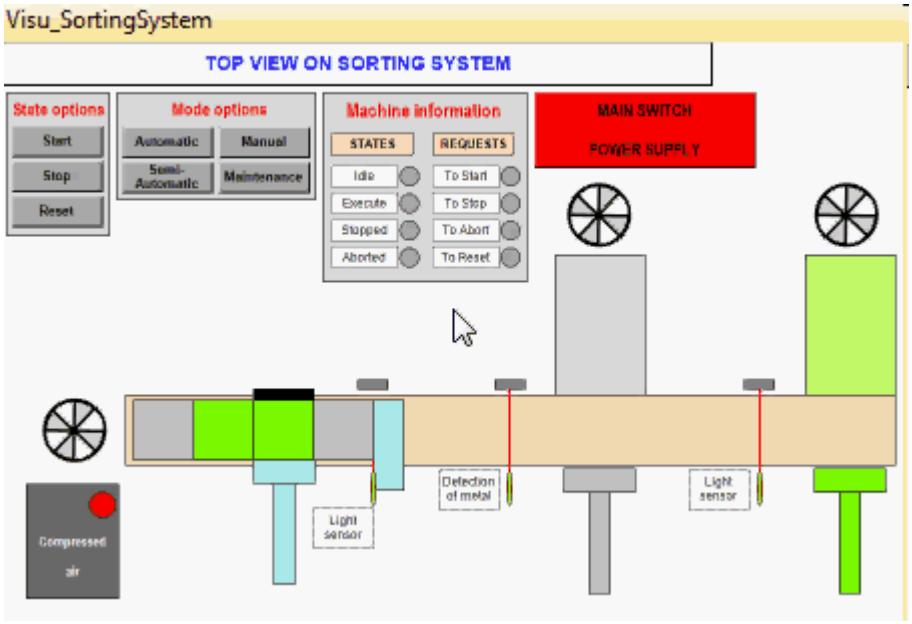
描述	本 PLC 示例包含控制分拣系统的面向对象的程序。该应用程序可通过集成的可视化系统进行控制。
示例项目	https://infosys.beckhoff.com/content/1033/tc3_plc_intro/Resources/7644036107.zip
更多信息	见 PLC 文档：控制分拣设备的面向对象的程序

7.20.11.1 控制分拣设备的面向对象的程序

面向对象的编程提供了多种多样的工具，可用于许多不同的用途。下面的示例说明了面向对象技术的好处，并展示了在 PLC/设备编程中使用面向对象技术的一些思路。在本示例中介绍的面向对象的概念和基本方法并不详尽，也不能推广到其他应用程序中。

用于说明面向对象的编程（OOP）的示例程序用于控制 1 个根据材料类型对金属箱和塑料箱进行分拣的分拣装置。分拣过程包括 1 个分选装置和 2 个卸料装置。通过气缸可以实现这些操作，气缸有不同的型号。不同的气缸型号在复杂程度上有所不同。在本示例中，要求系统能够用不同类型的气缸替换正在使用的气缸。

该系统可通过金属探测传感器区分不同的材料类型，从而将金属箱（灰色）和塑料箱（绿色）分别卸载到不同的辅助传送带上。以下视频通过俯视拍摄的方式展示了系统的分拣过程。

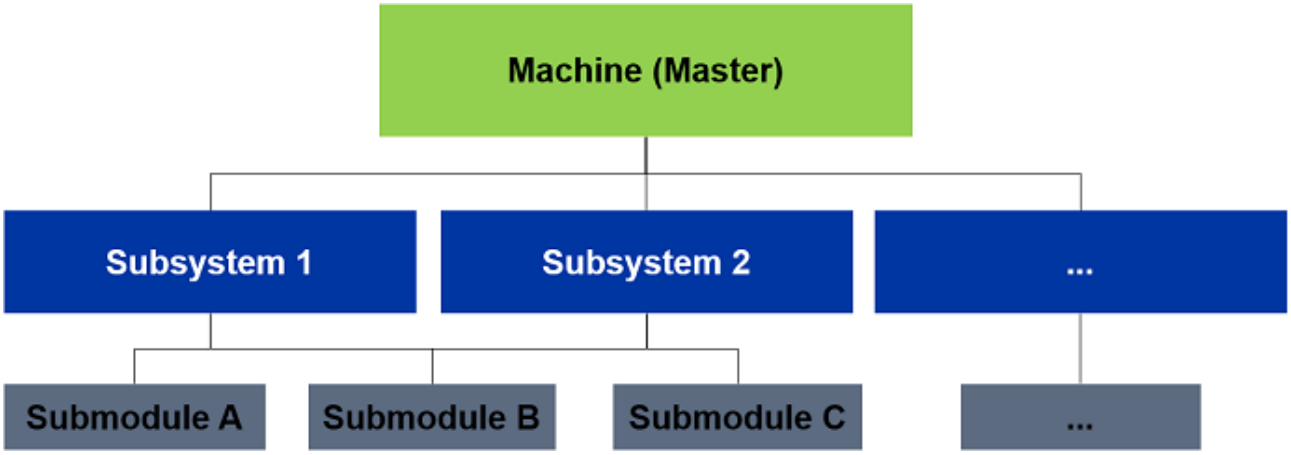


OOP 的一般概念

例如，面向对象的编程的 1 个总体目标就是开发自动化模块。这些对象是与应用程序无关的、现成的功能装置，只需开发 1 次，无需修改即可重复使用。因此，自动化模块并不是为某个特定设备开发的，相反，这些模块一般会提供可用于多个设备的系统功能。例如，1 个用于控制轴的类“FB_Axis”只需开发 1 次，然后，即可在多个系统中进行实例化和使用。

自动化模块可以大大减少系统编程的实现工作量。同时，所使用对象的质量和可靠性也相对较高，因为这些模块也可用于其他系统，因此会不断进行测试，也许还会得到改进。如果不同的模块采用相同的编程风格和实现概念，则可能会达成对象的统一“外观”，从而实现应用标准化。

对自动化模块进行有意义的开发和应用的前提是对系统进行模块化设计和模块化编程。例如，可以将系统细分为多个对象，使系统由不同的子系统组成，而这些子系统又包含不同的子模块。设备、其子系统及其子模块均以自动化模块的形式实现，因此可作为单独且独立的对象。



此外，还可以在子模块基类中概括所有子模块共有的数据和功能。通过开发子系统基类，如果子系统之间存在共性，则这种方法也可应用于子系统。一方面，这将大大减少编程工作，另一方面，在通用实现发生变化时，只需修改基类即可。

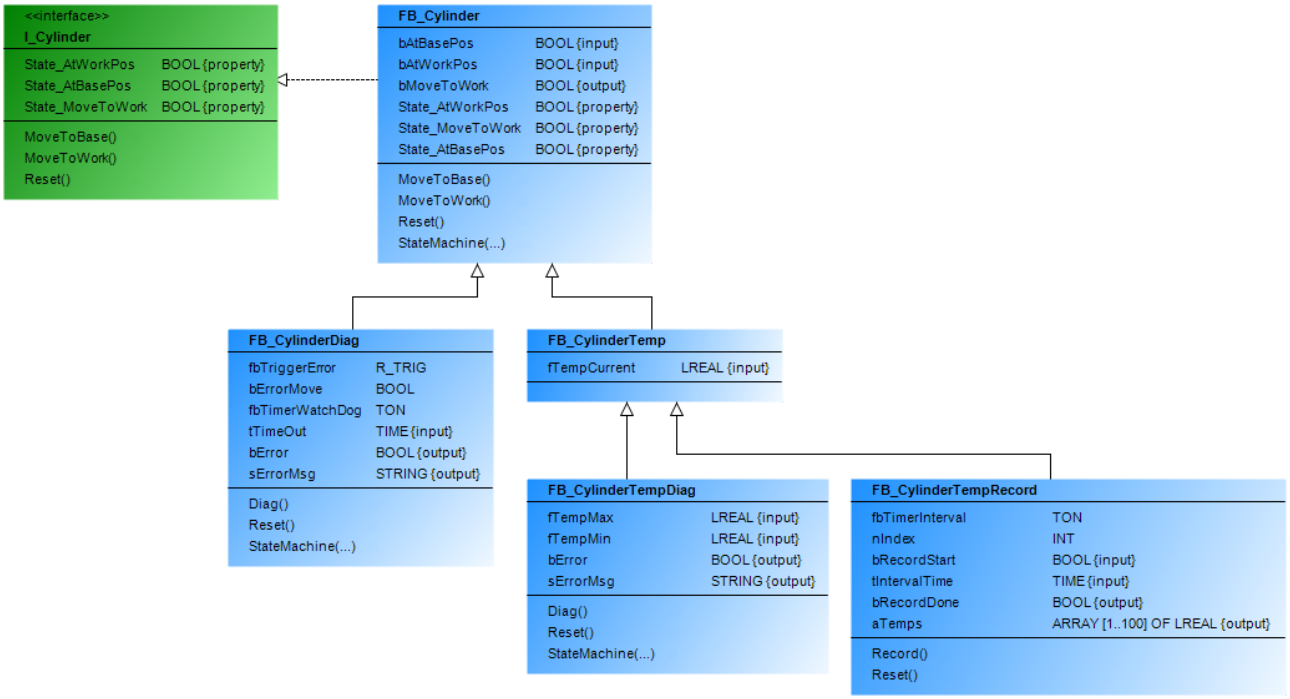
分拣装置 - 方法

下面将以气缸为例，详细解释基于面向对象技术的分拣装置的编程。其他子模块（例如，驱动器和传感器）的实现可从气缸编程的程序中派生。最后简要介绍了将子模块合并为子系统的情况。将气缸作为子模块实现的程序：

- 规划软件结构
- 实现软件概念
- 将程序元素实例化
- 为接口实例分配功能块实例
- 通过接口实例使用功能块实例
- 进一步的软件概念：将子模块合并为子系统

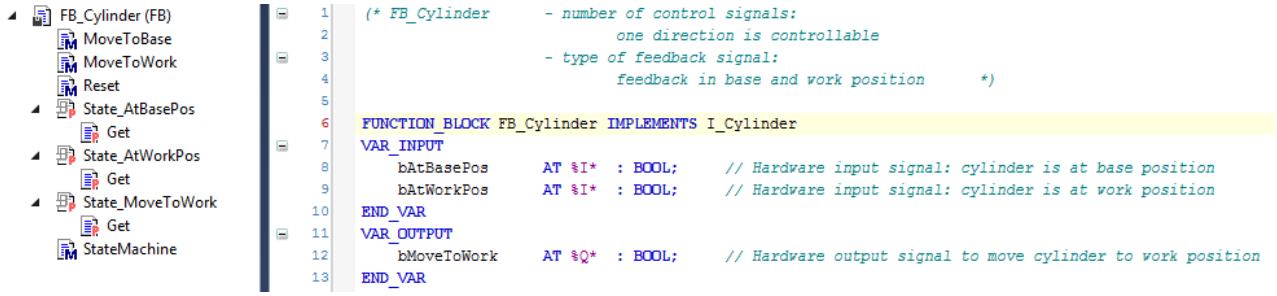
规划软件结构

所需的气缸类型包括只能移动到原点或工作位置的简单气缸，以及具有附加功能的气缸。这些更复杂的气缸可以监测到达其末端位置的情况，并记录或诊断其温度，以检测温度是否超出指定范围。单个设备气缸不应局限于某种气缸类型，但应能够与其他类型的气缸互换。例如，这样就可以用 1 个具有附加功能的气缸替代 1 个简单气缸。
不论是否具有任何辅助功能，所有气缸都应具有缩回和伸出的基本功能。由于该要求适用于应用程序中的每个气缸，因此，与该要求相关的方法和属性均在 1 个接口中进行定义。接口不包含实现，只定义方法和属性。因此，它仅代表了实现接口的功能块的 1 种规范。如果功能块提供了定义的方法和属性，则符合规范。最后，在实现功能块中会对在接口中定义的程序元素的实现进行编程。
定义接口 *I_Cylinder* 及其在气缸功能块中的实现，可确保气缸满足要求并能够访问相关的程序元素。此外，通过定义和实现接口，还可以通过接口实例访问功能块实例。这样可以确保所需的气缸具有互换性。
由于所有气缸都必须提供缩回和伸出的基本功能，因此，除了定义 1 个接口规范之外，对这些功能进行概括并通过 1 个基类来实现这些功能也是合理的。因此，计划使用气缸 *FB_Cylinder*，它具有气缸移动方法，代表了所有所需气缸的基类。这样可以确保所有派生类都能提供缩回和伸出的基本功能，尽管这些功能只在基础气缸中实现 1 次。
功能块 *FB_CylinderDiag* 从超类 *FB_Cylinder* 中派生。由于它还可以监测到达末端位置的情况，因此可通过此功能扩展基础气缸。它代表了 1 种所需气缸的解决方案。
由于气缸需要当前温度值来记录和监控温度，因此带有该组件的功能块从基类 *FB_Cylinder* 中派生。添加温度补充后，类名会变为 *FB_CylinderTemp*。它代表了具有温度记录和监控功能的气缸的超类。名称为 *FB_CylinderTempDiag* 和 *FB_CylinderTempRecord* 的 2 个子类通过这些功能所需的特定行为和变量进行了扩展。



实现软件概念

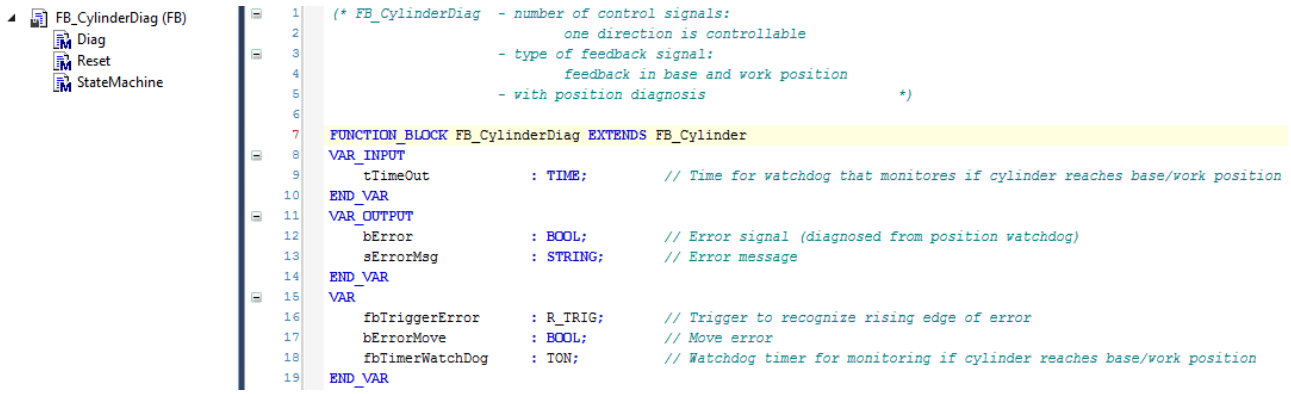
现在可以实现计划的功能块了。为了说明程序，下面将介绍基类 *FB_Cylinder* 和派生类 *FB_CylinderDiag* 的部分内容。
功能块 *FB_Cylinder* 借助关键字 IMPLEMENTS 集成接口 *I_Cylinder*，从而在超类中自动创建接口的方法和属性。这样可以确保功能块及其派生功能块具有接口所需的元素。接口产生的功能块结构和功能块 *FB_Cylinder* 的声明如下图所示。



方法 *Reset* 可重置功能块的气缸输出 *bMoveToWork*，示例如下。

```
// ======================================================
// *** Method Reset of FB_Cylinder ***

    bMoveToWork := FALSE;

// ======================================================
```

借助关键字 EXTENDS，功能块 *FB_CylinderDiag* 可以成为 *FB_Cylinder* 的派生功能块。然后可以使用基类的变量、方法和属性（取决于访问修饰符）。由于子类旨在扩展基类的 *Reset* 和 *StateMachine* 方法，因此这些方法会被插入到子类中，并且可以进行修改。此外，还集成了基类中没有的方法 *Diag*。在 *FB_CylinderDiag* 中可以声明诊断功能所需的附加变量。*FB_CylinderDiag* 的结构和声明如下所示：




```
(* FB_CylinderDiag  - number of control signals:
                          one direction is controllable
                    - type of feedback signal:
                          feedback in base and work position
                    - with position diagnosis                    *)

FUNCTION_BLOCK FB_CylinderDiag EXTENDS FB_Cylinder
VAR_INPUT
    tTimeOut            : TIME;         // Time for watchdog that monitores if cylinder reaches base/work position
END_VAR
VAR_OUTPUT
    bError              : BOOL;         // Error signal (diagnosed from position watchdog)
    sErrorMsg           : STRING;       // Error message
END_VAR
VAR
    fbTriggerError      : R_TRIG;       // Trigger to recognize rising edge of error
    bErrorMove          : BOOL;         // Move error
    fbTimerWatchDog     : TON;          // Watchdog timer for monitoring if cylinder reaches base/work position
END_VAR
```

在派生类的 *Reset* 和 *StateMachine* 方法中，超类的相应方法可以进行扩展或覆盖。如要获得方法扩展，需要使用关键字 SUPER 来调用基类的方法。SUPER 是 1 个功能块指针，它指向基类的功能块实例。基类的行为可以通过子类方法中的进一步指令进行扩展，从而针对气缸 *FB_CylinderDiag* 来调整方法。下面以 *FB_CylinderDiag* 的 *Reset* 方法为例，该方法通过进一步指令扩展了基类 *FB_Cylinder* 的相应方法。

```
// ===========================================================
// *** Method Reset of FB_CylinderDiag ***

    // Calling method Reset of base class FB_Cylinder via 'SUPER^.'
    SUPER^.Reset();

    // Reset error
    bError     := FALSE;
    sErrorMsg  := '';

// ===========================================================
```

将程序元素实例化

现在可以将借助继承和相应关键字创建的功能块实例化，这些功能块代表了具有不同功能的气缸。
将功能块 *FB_Cylinder*、*FB_CylinderDiag*、*FB_CylinderTemp*、*FB_CylinderTempDiag* 和 *FB_CylinderTempRecord* 实例化 1 次，以确保可以将气缸视为变量，并可以用每种气缸类型来表示。接口实例 *iCylinder* 是引用其中 1 个气缸功能块实例的对象，因此也是当前在运行时选择的气缸类型。
用户可以修改变量 *bCylinderDiag*、*bCylinderTemp* 和 *bCylinderRecord*，这些变量指明了气缸是否具有诊断和温度功能。
功能块、接口和 3 个 Boolean 变量的实例化如下所示。

```
// ===== Variables to enable/disable diagnosis and temperature mode ===
   bCylinderDiag         : BOOL;                   // If true the cylinder has diagnosis
functionality
   bCylinderTemp         : BOOL;                   // If true the cylinder has temperature
functionality
   bCylinderRecord       : BOOL;                   // If true the cylinder has recording
functionality

// ===== Function block instances for cylinder ========================
   fbCylinder            : FB_Cylinder;            // Without diagnosis and temperature mode
   fbCylinderDiag        : FB_CylinderDiag;        // With diagnosis of states
   fbCylinderTemp        : FB_CylinderTemp;        // With temperature mode
   fbCylinderTempDiag    : FB_CylinderTempDiag;    // With diagnosis of temperature
   fbCylinderTempRecord  : FB_CylinderTempRecord;  // With record of temperatures

// ===== Interface instance for cylinder ==============================
   iCylinder             : I_Cylinder;             // Interface for flexible access to cylinder FBs
```

为接口实例分配功能块实例

如要更换 1 个气缸，只需调整变量 *bCylinderDiag*、*bCylinderTemp* 和 *bCylinderRecord* 即可，因为相应的功能块实例会根据这 3 个变量的状态被分配给接口实例 *iCylinder*。
例如，如果气缸具有温度监控功能，则 *bCylinderDiag* 和 *bCylinderTemp* 的值为 TRUE，而 *bCylinderRecord* 的值为 FALSE。因此，所需的功能块为 *FB_CylinderTempDiag*，相关实例 *fbCylinderTempDiag* 被分配给接口实例。通过将该类的特定输出 *bError* 和 *sErrorMsg* 分配给局部变量，可以单独拦截它们。由于 FB 实例 *fbCylinderTempDiag* 被分配给接口实例 *iCylinder*，因此，通过接口实例可以访问具有温度监控功能的功能块实例。
反之，如果 Boolean 变量有其他值，接口实例将被相应地分配给不同的功能块实例。下面是 2 个功能块分配示例。

```
// ========================================================
// Selecting cylinder by checking variables to enable / disable diagnosis and temperature mode

   IF bCylinderDiag THEN
       IF bCylinderTemp THEN
           // =========== FB with diagnosis and temperature mode ==================
           bError     := fbCylinderTempDiag.bError;              // Assigning output variable of
chosen FB to local variable
           sErrorMsg  := fbCylinderTempDiag.sErrorMsg;
           iCylinder  := fbCylinderTempDiag;                     // Assigning chosen FB instance to
interface instance

       ELSE
           // =========== FB with diagnosis and without temperature mode ===========
           fbCylinderDiag.tTimeOut  := tTimeOutCylinder;         // Setting special data for
selected FB
           bCylError                := fbCylinderDiag.bError;    // Assigning output variable of
chosen FB to local variable
           sCylErrorMsg             := fbCylinderDiag.sErrorMsg;
           iCylinder                := fbCylinderDiag;           // Assigning chosen FB instance to
interface instance

       …

   END_IF

// ========================================================
```

通过接口实例使用功能块实例

被分配给 1 个 FB 实例的接口实例可以借助点符号调用其方法。通过接口实例进行该方法调用时，接口指向的功能块实例的方法会因接口指针而被调用。
下面的程序部分说明了如何通过接口实例将选定的气缸功能块移动到原点或工作位置。

```
// ========================================================
// Manual cylinder control (Calling methods of FB via interface instance)

   // Cylinder to work position
   IF fbButtonCylToWork.bOut THEN
       iCylinder.MoveToWork();
   // Cylinder to base position
   ELSIF fbButtonCylToBase.bOut THEN
       iCylinder.MoveToBase();
   END_IF

// ========================================================
```

进一步的软件概念：将子模块合并为子系统

此处介绍的借助接口实例的功能块实现及其实例应用代表了小规模的面向对象的编程。为了更好地应用面向对象技术，还需辅以更大规模的 OOP。不同的子模块被合并为子系统。
对于分拣装置来说，将用于材料检测的传感器、用于控制辅助传送带移动的控制器和卸料气缸合并为卸料模块是合理的。这样就可以从该类中将许多不同的对象或分拣模块实例化，只需对 1 个卸料模块进行编程即可。根据这种方法可以配置不同的系统，这些系统具有专门用于其他材料类型（例如，金属、塑料、箔、玻璃等）的不同数量的分拣模块。
分拣装置的另一个子系统负责分选。它由 1 个用于识别箱子的传感器、1 个用于将箱子移到主传送带上的驱动器以及 2 个用于实现实际分选的气缸组成。
由于分选和卸料的 2 个子系统具有共性，因此子系统特定的数据和功能可被合并到 1 个子系统基类中。然后，将分选和卸料声明为该功能块的子类，这意味着它们可继承子系统基类的通用程序元素。

通过开发子模块和子系统，可以在不同层面应用和利用面向对象的编程选项和优势。

TC3.1 来源

在此处可以将示例程序的完整 TC3.1 来源解压缩：https://infosys.beckhoff.com/content/1033/tc3_plc_intro/Resources/7644036107.zip

启动示例程序：

- 激活 TwinCAT 配置并在运行模式下启动 TwinCAT
- 登录 2 个 PLC 并启动它们（**TC3_SortingSystem_PLC** 和 **TC3_SortingSystem_Simu**）
- 通过设备可视化操作应用程序。这可以在以下位置找到：

    - PLC 项目：TC3_SortingSystem_PLC
    - 文件夹：05_Visu
- 例如：按以下按钮，在自动模式下启动系统：
    - “Main Switch Power Supply”（主开关电源）
    - “Automatic”（自动）
    - “Start”（开始）

另请参见：

- PLC 文档：面向对象的编程

8 将 PLC 项目传输到 PLC

为了将 PLC 项目传输到控制器，务必准确无误地编译程序。

8.1 生成程序代码

程序代码是机器代码，当您启动 PLC 程序时，控制器会执行该机器代码。在将 PLC 项目下载到控制器之前，TwinCAT 会根据在开发系统中编写的源代码自动生成程序代码。在生成程序代码之前，系统会检查赋值、数据类型和库的可用性。此外，在生成程序代码时会分配内存地址。

在每次下载时，包含已加载 PLC 项目的代码和标识的编译日志（Compile Info）都会在目标设备上另存为文件。

程序代码生成过程中的消息

由于采用增量编译，仅会为新的和修改后的功能块和变量重新分配内存。因此，内存中可能会出现空白。在线更改也有同样的效果。这种分段储存会减少可用的空闲内存。在这种情况下，您可以使用命令“Clean”（清除）重新分配整个内存，从而增加空闲内存。有关代码生成过程中的消息的更多信息：在“Compile”（编译）消息窗口（错误列表）中会显示语法错误以及 TwinCAT 在代码生成和内存分配过程中检测到的错误。在每次代码生成过程中，还会在那里显示代码大小、数据大小（以字节为单位）、占用内存区的内容以及使用的最高地址（以字节为单位）等信息。这取决于 PLC 将数据和代码存储在哪个内存区。

8.2 加载程序代码、登录和启动 PLC

如要将您的 PLC 程序的源代码下载到控制器上，您必须先登录控制器上的 PLC 项目。如果您的项目中有多个 PLC 项目，您必须首先激活所需的 PLC 项目。

加载 PLC 项目和启动程序

✓ PLC 项目准确无误，且还不在控制器上。

✓ PLC 项目以及与控制器的通信均未加密。

1. 在 **TwinCAT PLC Toolbar Options**（TwinCAT PLC 工具栏选项）的下拉列表 **Active PLC Project**（活动的 PLC 项目）中，选择要加载和启动的 PLC 项目。
   ⇨ 活动的 PLC 项目将作为第 1 个条目出现在下拉列表中。
2. 在 **TwinCAT XAE Base Toolbar Options**（TwinCAT XAE 基础工具栏选项）中，点击 **Activate Configuration**（激活配置） 。
   ⇨ 此时会出现 1 个对话框，询问您是否想要激活配置。
3. 点击 **OK**（确定）。
   ⇨ 此时会出现 1 个对话框，询问是否应该在运行模式下重新启动 TwinCAT。
4. 点击 **OK**（确定）。
   ⇨ 配置已被激活，TwinCAT 已被设置为运行模式。在任务栏中显示当前状态： 。激活还可将 PLC 项目传输至控制器。
5. 在 **PLC** 菜单或 **TwinCAT PLC Toolbar Options**（TwinCAT PLC 工具栏选项） 中，选择命令 **Login**（登录）。
   ⇨ 此时会出现 1 个对话框，询问您是否想要在控制器上创建并加载应用程序。
6. 选择 **YES**（是），确认对话框。
   ⇨ PLC 项目已加载到控制器上。
7. 在 **PLC** 菜单或 **TwinCAT PLC Toolbar Options**（TwinCAT PLC 工具栏选项） 中选择命令 **Start**（开始），或者按 **[F5]**。

⇨ 程序在控制器上运行。

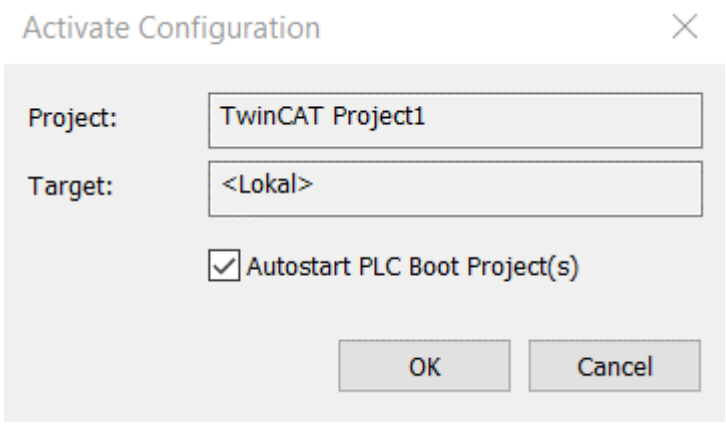
另请参见：

- 更新 PLC 上的 PLC 项目 [▶ 231]

- TC3 用户界面文档：命令登录 [▶ 889]
- TC3 用户界面文档：命令激活配置 [▶ 869]
- 监控值
- 编程语言及其编辑器

8.3 自动加载程序

如果您在您的 PLC 项目 中激活配置，则会打开 1 个对话框，您可以在其中查看或设置是否激活“Autostart PLC Boot Project(s)”（自动运行 PLC 启动项目）。



对话框中的复选框表示当前是否已为目标上的 PLC 激活自动启动功能：

- 已选中：目标上的所有自动启动均已设置。
- 未选中：目标上的所有自动启动均已删除。

如要更改您的 PLC 的自动启动功能，可设置或删除复选框中的复选标记。当您选择 **OK**（确定）确认选择后，您的输入内容将被保存，您将进入 TwinCAT 运行模式。

在解决方案资源管理器中，此选项对应于 PLC 项目的 **Project**（项目）选项卡中的 **Autostart Boot Project**（自动运行启动项目）命令。

9 测试 PLC 项目和故障排除

TwinCAT 3 PLC 为您提供了各种用于测试应用程序和查找错误的选项。即使没有连接硬件，您也可以在仿真模式下启动应用程序。通过断点和逐步执行程序的命令，您可以检查程序的特定部分。通过编写变量值，您可以影响正在运行的程序。

您可以使用命令在不同程度上重置应用程序。从重置非持久变量到将控制器完全重置为出厂状态。

9.1 使用断点

通常，断点可用于查找程序中的错误。您可以在程序中的某些位置设置断点，以强制停止执行并观察变量值。TwinCAT 3 PLC 支持所有 IEC 编辑器中的断点。

断点处的停止可与附加条件相关联。您还可以将断点重新定义为执行点，在这些执行点上，程序不会停止，而会执行某些代码。

Breakpoints（断点）视图（菜单 **PLC** > **Windows**（PLC > 窗口））提供了所有已定义断点的概述。您还可以使用其他命令，以便同时更改多个断点。

在编辑器中，断点和执行点的状态用以下符号标记：

- 断点已激活
- 断点已停用
- 在编辑器中当前打开的功能块的不同实例中设置断点。
- 在断点处停止
- 带有条件的断点已激活
- 带有条件的断点已停用
- 执行点已激活
- 执行点已停用
- 带有条件的执行点已激活
- 带有条件的执行点已停用

另请参见：

- TC3 用户界面文档：PLC [▶ 872]
- TC3 用户界面文档：调试 [▶ 863]
- TC3 用户界面文档：命令：调用栈 [▶ 879]

具有多个任务的 PLC 项目中的断点

如果在执行 PLC 项目时到达断点，则任何任务都不会再执行该 PLC 项目中的代码。在 PLC 项目之外的代码仍会被执行。

如果 PLC 上的程序在断点处停止，则在线更改或下载会停止所有任务。这意味着 PLC 会停止。在这种情况下，TwinCAT 会显示相应的消息，您可以决定是否想要继续登录。

设置单个断点（示例：ST 编辑器）

✓ 项目处于在线模式。

1. 在编辑器中用 ST 语言打开 POU。
2. 将光标置于要设置断点的行中。

3. 在 **Debug**（调试）菜单或上下文菜单中选择命令 **Toggle Breakpoint**（切换断点），或者按 **[F9]** 键。

   ⇨ 该行显示为红色，并标有图标 ● （已启用断点）。如果程序在断点处停止，该行会标有图标 ➡ （在断点处停止）。停止执行程序。

4. 在 **PLC** 菜单或 **TwinCAT PLC Toolbar Options**（TwinCAT PLC 工具栏选项）工具栏中选择命令 **Start**（开始），或者按 **[F5]** 键。

   ⇨ 程序继续运行。

5. 设置更多断点，并检查停止位置上的变量值。
6. 将光标置于要删除断点的行中。
7. 在 **Debug**（调试）菜单或上下文菜单中选择命令 **Toggle Breakpoint**（切换断点），或者按 **[F9]** 键。

⇨ 标记消失。断点被删除。

另请参见：

- TC3 用户界面文档：命令：切换断点 [▶ 867]

定义断点条件（示例：ST 编辑器）

✓ 项目处于在线模式。

1. 在编辑器中用 ST 语言打开 POU。
2. 在菜单 **PLC** > **Windows**（PLC > 窗口）中，选择命令 **Breakpoints**（断点）。

   ⇨ **Breakpoints**（断点）视图会打开。

3. 在工具栏中选择命令 **New**（新建）。

   ⇨ 对话框 **New Breakpoint**（新建断点）会打开。显示选项卡 **Location**（位置）。或者，您也可以在菜单 **Debug**（调试）中使用命令 **New Breakpoint**（新建断点），打开对话框。

4. 选择 POU 和新断点的位置。
5. 选择选项卡 **Condition**（条件）。
6. 在 **Hit Count**（命中次数）部分中，选择选项 **Break when the hit count is a multiple of**（当命中次数达到.....的倍数时断开），并在其右侧字段中输入值 5。
7. 还可定义 1 个 Boolean 条件，以确定何时激活断点。激活选项 **Break if TRUE**（如果为 TRUE，断开）。在其右侧字段中输入 1 个 Boolean 变量。
8. 激活选项 **Enable breakpoint immediately**（立即启用断点）。
9. 关闭对话框。

⇨ 该行显示为红色，并标有图标 ⭘。

现在观察正在运行的程序。只要条件的 Boolean 变量为 FALSE，就不满足断点的条件，程序就会运行。如果您将该变量设置为 TRUE，则会满足条件，程序每运行 5 次就会在该断点处停止。

另请参见：

- TC3 用户界面文档：命令：断点 [▶ 878]
- TC3 用户界面文档：命令：新断点 [▶ 863]

定义执行点（示例：ST 编辑器）

✓ 项目处于在线模式。

1. 在编辑器中用 ST 语言打开 POU。
2. 从菜单 **PLC** > **Window**（PLC > 窗口）中，选择命令 **Breakpoints**（断点）。

   ⇨ **Breakpoints**（断点）视图会打开。

3. 在工具栏中选择命令 **New**（新建）。

   ⇨ 对话框 **New Breakpoint**（新建断点）会打开。显示 **Location**（位置）选项卡。
   或者，您也可以在 **Debug**（调试）菜单中使用 **New Breakpoint**（新建断点）命令，打开对话框。

4. 选择 POU 和执行点的位置。
5. 选择 **Execution point settings**（执行点设置）选项卡。

6. 激活 **Execution point**（执行点）选项。
   在字段 **Execute following code**（执行以下代码）中，输入到达执行点时要执行的所需指令。例如，`nCounter := nCounter + 1;`，如果变量 nCounter 可用的话。
7. 关闭对话框。

⇨ 该行显示为红色，并标有图标 

在到达执行点时，程序不会停止，而会执行定义的代码。

另请参见：

- TC3 用户界面文档：命令：断点 [▶ 878]
- TC3 用户界面文档：命令：新断点 [▶ 863]

9.2 逐步处理程序（步进）

您可以逐步执行 PLC 项目，浏览代码。这对于在运行时确定代码的状态非常有用。您可以检查调用顺序、跟踪变量值或确定错误。为此，在 **Debug**（调试）菜单中提供了步骤命令。当程序处于规定的程序步骤（调试模式）时，这些命令才可用。在调试模式下，当前停止位置以黄色突出显示，并在文本编辑器中用符号  标记。

切换到调试模式

✓ PLC 项目处于在线模式。

1. 在 POU 中您想要检查的代码位置设置断点。
2. 启动程序。
   ⇨ 程序启动，代码处理到第 1 个断点。现在，项目处于调试模式。
   ⇨ 当前停止位置的编辑器已打开。程序停止执行的代码行有 1 个活动断点，该行以黄色突出显示，并用符号  标出（在断点处停止）。指令尚未执行。
   ⇨ 您可以选择不同的步骤命令或显示调用堆栈。

步骤命令的行为

- 命令：单步跳过 [▶ 868]（“单步跳过”） 
  在停止位置执行指令。程序在编程块中的下一条指令之前停止。
  如果语句中有调用（来自程序、功能块实例、函数、方法或动作），则在 1 个步骤中可以完全遍历下级编程块。

- 命令：单步跳入 [▶ 868]（“单步跳入”） 
  在停止位置执行指令。程序在下一条指令之前停止。
  如果指令中有调用（来自程序、功能块实例、函数、方法或动作），程序就会跳转到该下级编程块。在执行第 1 条指令后，程序会在下一条指令之前停止。然后，新的当前停止位置就会出现在被调用的编程块中。

- 命令：单步跳出 [▶ 868]（“单步跳出”） 
  该命令从当前停止位置至块结束位置执行编程块，然后跳回至调用编程块。在调用位置（调用行中），程序停止运行。
  如果当前停止位置在主程序中，则编程块将运行至结束。然后，程序跳回至起点（编程块中第 1 行代码处的程序起点）并停止。

- 命令运行至光标处 [▶ 869]（“运行至光标处”） 
  首先将光标置于任意一行代码上，然后选择命令。程序从当前停止位置开始执行，并在当前光标位置停止，而不执行该行代码。

- 命令设置下一语句 [▶ 869]（“设置下一语句”） 

首先将光标置于任意一行代码上（甚至可以置于当前停止位置之前），然后选择命令。接下来，执行用光标标记的指令。中间的所有指令均会被忽略和跳过。

- 命令：显示下一语句 [▶ 869]（“显示下一语句”）
  如果您没有看当前停止位置，可选择该命令。然后，当前停止位置的窗口会进入活动状态，并显示停止位置。

选择命令“PLC > Window > Call Stack”（PLC > 窗口 > 调用堆栈），可完整显示在程序执行中当前到达的停止位置的上一个调用树。

因此，即使在编译 PLC 项目之前，**Call Tree**（调用树）视图也能随时显示功能块在程序调用结构中的位置。

9.3 强制和写入变量值

在 TwinCAT 中，您可以在在线模式下更改控制器上的变量值。这将覆盖变量最初的输入值。这有 2 种不同的方法：强制和写入先前的准备值。

⚠ 谨慎

由于设备或系统的意外行为而造成的财产和人身损害

在控制器上运行的 PLC 程序中，变量值的异常变化可能会导致受控设备的异常行为。根据不同的受控设备，可能会损坏设备或工件，或者可能会危及人员的健康和生命。

- 在强制变量值前评估可能的风险并采取适当的安全预防措施。

写入是通过命令 **Write values（写入值）** 完成的，将变量 1 次设置为准备值。因此，程序随时可以覆盖该值。

强制是通过命令 **Force values**（强制值） 完成的，永久设置准备值。

在不同的点上可以为强制或写入准备值：

- 声明部分：字段 **Prepared value**（准备值）
- 实现部分：内联监控字段
- 监控窗口：字段 **Prepared value**（准备值）

强制的工作原理

在强制的情况下，TwinCAT 会在每个周期中写入值，从而使变量永久保持在强制值上。用户必须取消强制。与任何其他变量一样，变量的强制值可在 PLC 周期内更改。

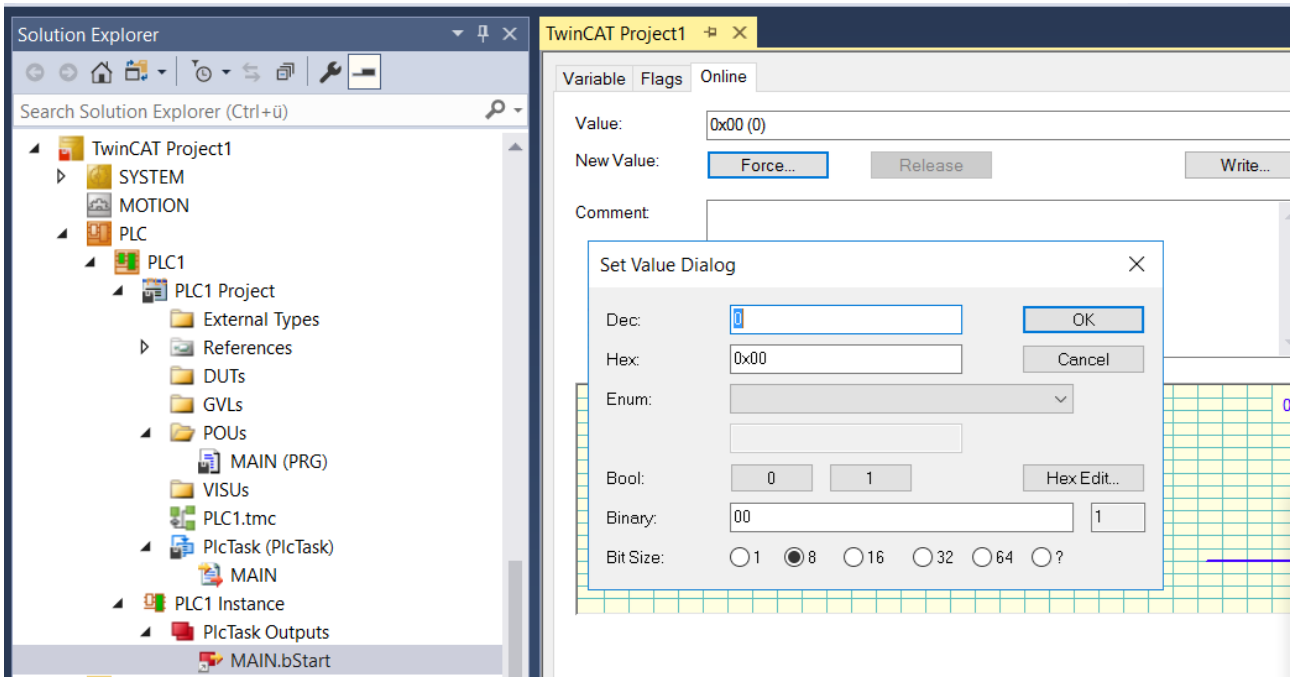
在每个处理周期的开始和结束时，将准备值设置为相应的变量。每个周期的处理顺序：

1. 读取输入
2. 强制值
3. 处理代码
4. 强制值
5. 写入输出。

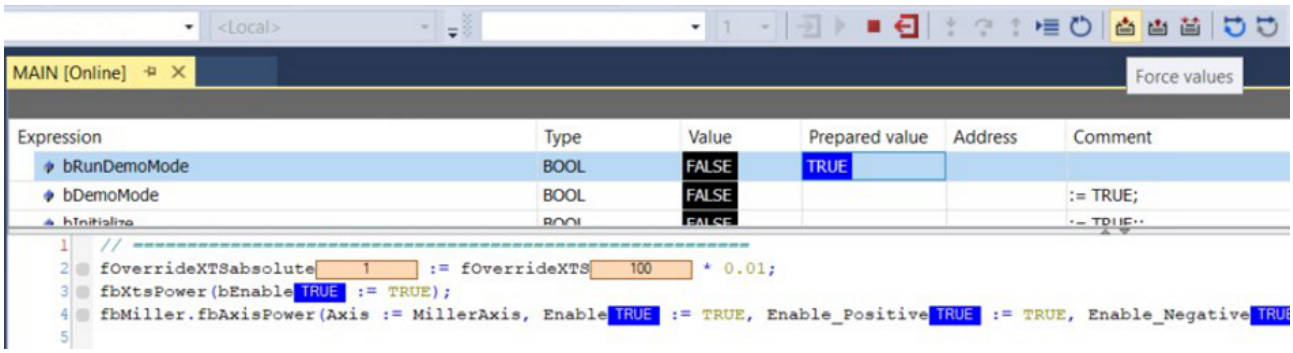
在周期中的代码处理过程中，强制变量有可能因为代码执行赋值而暂时获得不同的值。然后，该变量只有在周期结束时才会再次收到强制值。此外，客户端对应用程序符号的写入访问也会在周期内覆盖变量值。

强制值有 2 种不同的方法。

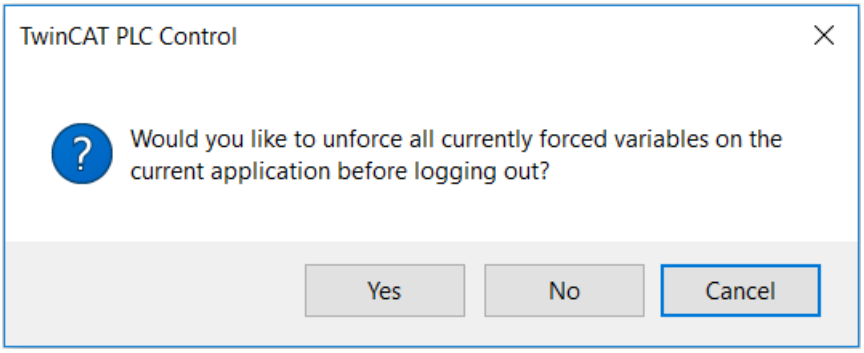
其中 1 种方法是通过解决方案资源管理器打开项目。然后，直接从此处强制值。为此，无需登录运行时系统。这样，即使用户离开运行时系统，变量的值也会保持强制。在这种情况下，既不会出现警告消息，也不会出现查询对话框。



相比之下，只有当用户登录运行时系统 时，才有可能在 PLC 中强制变量。



当退出 运行时系统时，会出现 1 个对话框询问变量是否应该保持强制。



如果您在此对话框中选择 Yes（是），则将取消强制值。如果您在此处选择 No（否），则强制值将存储在运行时系统中，并保持永久不变。这意味着您可以在此期间退出运行时系统，如果您在其他时间重新登录运行时系统，强制值仍然存在。

注意

因永久性强制变量造成的材料损失

变量会被永久保持在强制值上，因此强制值的持续时间可能会比预期时间要长，并且可能会造成材料损坏。这尤其适用于设备在无人看管的情况下运行的情况。

- 为了确保设备不会执行任何意外动作，可在加工过程结束时重置强制值。

请注意，用户必须明确取消强制变量 F。不过，即使在取消强制后，变量的强制值也会持续存在。

- 取消对变量的强制。
- 如要安全取消变量的强制值，可将值改回原始值。
- 退出 TwinCAT 并选择 **Yes**（是），对是否应该取消对所有变量的强制的查询进行确认。

⇨ 现在，变量再次变为原始值。

不能强制临时变量。只有当 PLC 位于定义临时变量的相同 POU 的代码中的断点位置时，才能写入临时变量。在流程控制模式下，写入临时变量也没有影响。

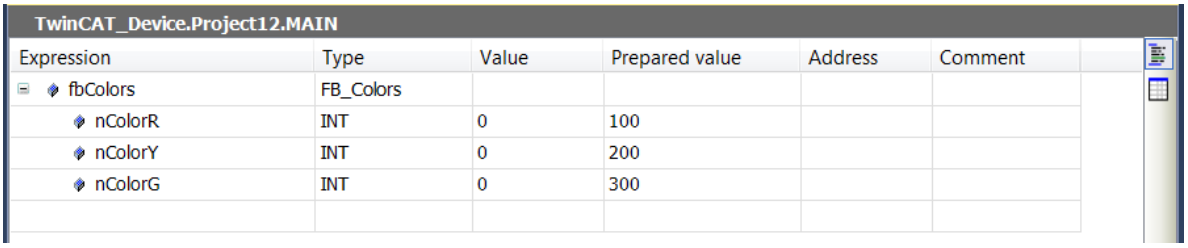
另请参见：

- TC3 用户界面文档：命令：强制值 [▶ 892]
- TC3 用户界面文档：命令：写入值 [▶ 894]
- TC3 用户界面文档：命令：取消强制值 [▶ 893]

声明部分中的强制

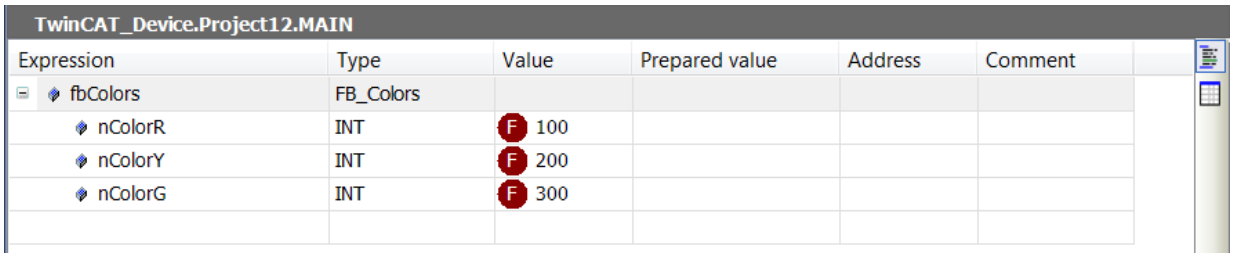
✓ 您的 PLC 项目有 1 个带有声明的 POU。应用程序处于在线模式。

1. 在菜单 **View**（视图）或上下文菜单中，双击对象或点击命令 **Open**（打开），在编辑器中打开 POU。
2. 在编辑器的声明部分，双击变量的 **Prepared value**（准备值）列。
   ⇨ 该字段变为可编辑字段，您可以输入值。



3. 对其他变量执行步骤 2。
4. 在 **PLC** 菜单或 **TwinCAT PLC toolbar options**（TwinCAT PLC 工具栏选项）工具栏中，选择 **Force value**（强制值）命令 。

⇨ 变量的值会被准备值覆盖。该值用符号 F 标出。



您还可以在视图 **PLC** > **Windows** > **Watch** <n>（PLC > 窗口 > 监视 <n>）中强制变量的值。

另请参见：

- TC3 用户界面文档：命令：强制值 [▶ 892]

实现部分中的强制

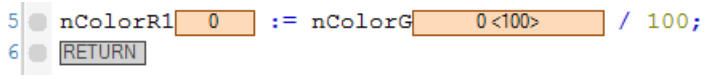
✓ 应用程序处于在线模式。

1. 在菜单 **View**（视图）或上下文菜单中，双击对象或点击命令 **Open**（打开），在编辑器中打开 POU。
2. 双击编辑器的实现部分中的内联监控字段。

⇨ 对话框 **Prepare Value**（准备值）会打开。

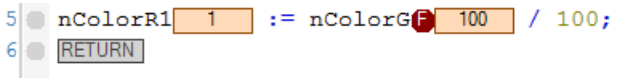
3. 在字段 **Prepare a new value for the next write or force operation**（为下一次写入或强制操作准备新值）中，输入新值。

   ⇨ 在内联监控字段中会显示准备值。



4. 在菜单 **PLC** 或工具栏 **TwinCAT PLC Toolbar Options**（TwinCAT PLC 工具栏选项）中，选择命令 **Force values**（强制值）。

⇨ 变量的值会被准备值覆盖。该值用符号 F 标出。



另请参见：

- TC3 用户界面文档：对话框准备值 [▶ 893]
- TC3 用户界面文档：命令：强制值 [▶ 892]
- TC3 用户界面文档：命令：取消强制值 [▶ 893]

查看和取消列表中的所有强制变量

✓ 应用程序处于在线模式。多个变量处于强制状态。

1. 在菜单 **PLC > Windows**（PLC > 窗口）中，选择命令 **Watch all forces**（监视所有强制执行）。

   ⇨ 视图 **Watch all forces**（监视所有强制执行）会出现。它以监视列表的形式包含 PLC 项目的所有当前强制变量。

2. 选择列表的所有行，然后在左上角的选择列表中选择视图 **Unforce > Unforce and keep all selected values**（取消强制 > 取消强制并保留所有选定值）。

⇨ 对变量取消强制，并赋予它们强制之前的值。

另请参见：

- TC3 用户界面文档：命令：监视所有强制执行 [▶ 875]
- TC3 用户界面文档：命令写入值 [▶ 894]
- TC3 用户界面文档：命令取消强制值 [▶ 893]

9.4 重置 PLC 项目

重置 PLC 项目可停止程序并将变量重置为其初始化值。根据重置的类型，RETAIN 变量和 PERSISTENT 变量也会进行重置。

- 冷重置：除了剩余变量（RETAIN 和 PERSISTENT 变量）之外，活动 PLC 项目的所有变量均被重置为其初始化值。
- 重置原始值：活动 PLC 项目的所有变量，包括剩余变量（RETAIN 和 PERSISTENT 变量），均被重置为其初始化值。控制器上的 PLC 项目已重置。

小型示例程序和下面的说明向您展示了不同重置的行为。

另请参见：

- 引用编程：剩余变量 - PERSISTENT, RETAIN [▶ 630]
- TC3 用户界面文档：命令冷重置 [▶ 891]
- TC3 用户界面文档：命令：初始值复位 [▶ 891]

示例程序

声明：

```
VAR
    nVar : INT := 0;
END_VAR
VAR RETAIN
    nVarRetain : INT :=0;
END_VAR
VAR PERSISTENT
    nVarPersistent : INT:= 0;
END_VAR
```

实现：

```
nVar := 100;
nVarRetain := 200;
nVarPersistent := 300;
```

1. 执行命令 **Build**（构建）。
2. 将 PLC 项目加载到控制器上。
3. 在菜单 **PLC** 或工具栏 **TwinCAT PLC Toolbar Options**（TwinCAT PLC 工具栏选项）中，选择命令“Login to switch to online mode”（登录切换到在线模式）。
4. 启动 PLC 程序。

⇨ 观察变量 nVar、nVarRetain 和 nVarPersistent。

执行冷重置：

1. 在菜单 **PLC** 或 **TwinCAT PLC Toolbar Options**（TwinCAT PLC 工具栏选项）中，选择命令 **Reset cold**（冷重置）。

   ⇨ 此时会出现 1 条查询消息，询问您是否真的想要执行该命令。
2. 选择 **Yes**（是），确认对话框。

⇨ PLC 项目已重置。变量 nVar 被设置为初始化值 0。RETAIN 变量 nVarRetain 和 PERSISTENT 变量 nVarPersistent 保留其值

执行重置原点：

1. 在菜单 **PLC** 或 **TwinCAT PLC Toolbar Options**（TwinCAT PLC 工具栏选项）中，选择命令 **Reset origin**（重置原点）。

   ⇨ 此时会出现 1 条查询消息，询问您是否真的想要执行该命令。
2. 选择 **Yes**（是），确认对话框。

⇨ PLC 项目已退出。所有变量均被重置为其初始化值。

9.5 流程控制

您可以通过流程控制来跟踪程序的执行情况。流程控制适用于语言编辑器 ST、FBD、LD 和 CFC。

在激活流程控制后，TwinCAT 会在相应的执行位置和相应的执行时间显示变量值以及函数调用和操作的结果。准确地说，在当前周期中运行的代码行或网络会用颜色标记。比较：在标准监控下，TwinCAT 只返回变量在 2 个执行周期之间的值。

流程控制在当前打开的编辑器窗口的所有当前可见部分中运行。因此，只要函数处于活动状态且流程控制位置（已运行的代码部分）在编辑器窗口中可见，在状态栏中就会显示 **Flow Control activated**（已激活的流程控制）。

您可以在声明部分和实现部分写入值。不支持强制。

在当前周期结束时完成值的写入。

如果您激活流程控制，则 PLC 项目的运行时会延长。

各种语言编辑器中流程控制的表示

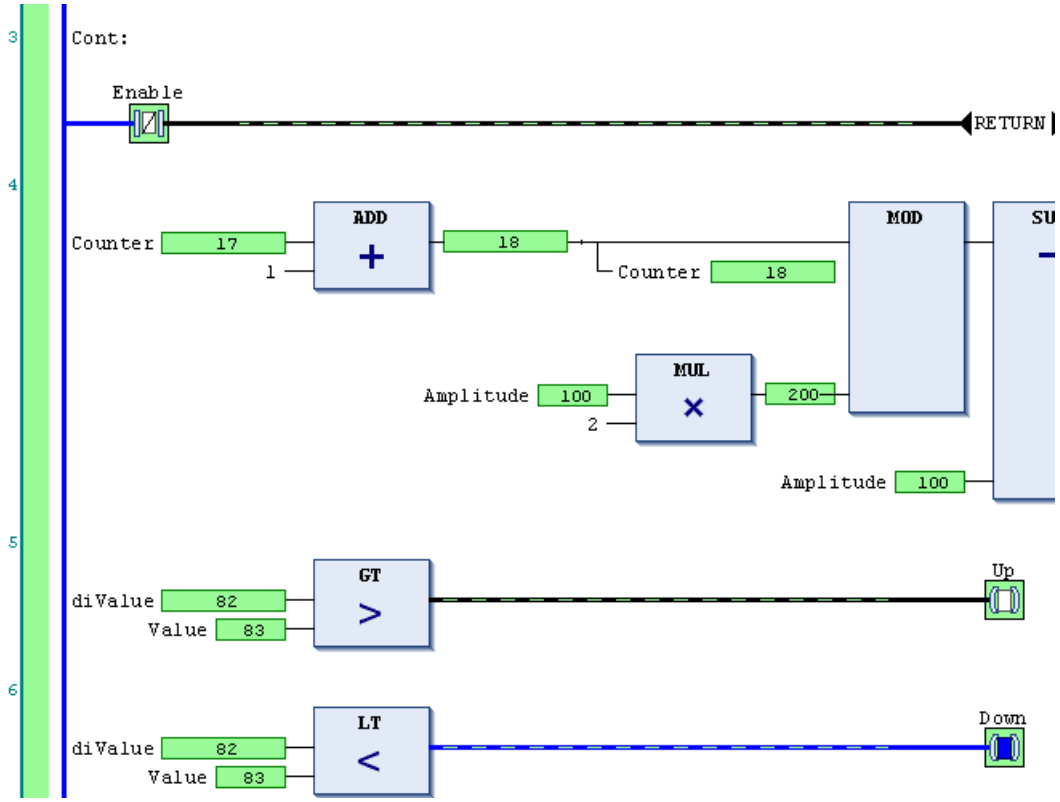
默认情况下，TwinCAT 将已运行的代码部分的流程控制位置表示为绿色区域。尚未运行的代码部分显示为白色。

请注意，尚未运行的代码位置的显示值是“正常”监控值。这是 2 个任务周期之间存在的值。

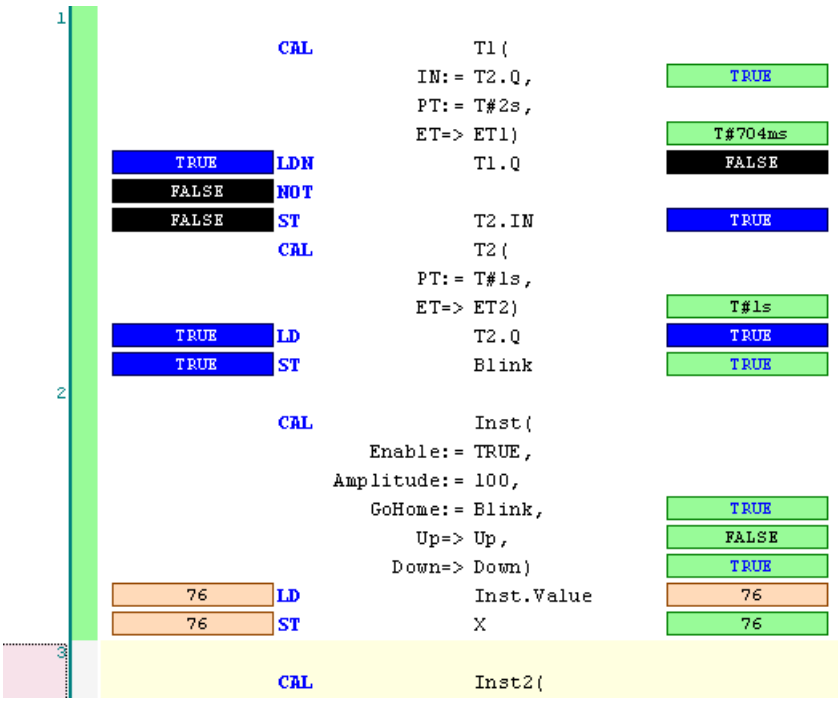
```
1  i 1619  := i 1619  + 1;
2  b 0  := NOT b 0 ;
3  IF str 'abcdefghij  = strl "  THEN
4  fl2 1.5  := fl 1.23 ;
5  ELSE
6  fl2 1.5 :=1.5;
7  D 6.5E+04 :=B255*B255;
8  END_IF;
9  IF D 6.5E+04 <0.0 THEN
```

在网络编辑器中，TwinCAT 会在“流程控制颜色”的左侧边缘用条形图标出已运行的网络。

在 LD 中，TwinCAT 会以绿色显示当前正在运行的连接线，以灰色显示其他连接线。连接的实际值也会显示：蓝色粗线显示 TRUE，黑色粗线显示 FALSE，黑色细线显示未知值或模拟值。如果将各条信息组合起来，可能就会出现虚线。



在 IL 中，TwinCAT 为每个语句提供 2 个区域来显示实际值。1 个在运算符的左边，带有当前的累加器值，另一个在操作数的右边，带有操作数值。



另请参见：

- TC3 用户界面文档：命令：流控制 [▶ 892]

9.6 使用调用堆栈确定当前处理位置

借助调用堆栈，您可以确定程序执行的当前位置。该功能对于逐步执行程序非常有用。

✓ 项目处于在线模式。程序在断点处停止，或者您正在逐步执行程序。

1. 在菜单 **PLC** > **Windows**（PLC > 窗口）中，使用命令 **Call Stack**（调用堆栈）可打开调用堆栈。

⇨ 调用堆栈已打开。列表会显示当前到达的位置以及完整的调用路径。

调用堆栈也可在离线模式和正常在线模式下使用（当前未使用调试功能）。在这种情况下，它包含在逐步执行期间最后显示的位置，字体为“灰色”。

另请参见：

- 逐步处理程序（步进） [▶ 197]
- TC3 用户界面文档：命令：调用栈 [▶ 879]

10 在运行时的 PLC 项目

当应用程序在 PLC 上运行时，TwinCAT 3 开发系统中具有监控和更改变量值以及记录和保存变量开发的功能。

10.1 监控值

在运行时，您可以观察编程对象的变量在项目中各点的当前值。这就是所谓的“监控”：

- 对象的编程编辑器的在线视图：“内联监控”
- 对象的声明编辑器的在线视图
- 与对象无关的可配置监视列表

通过设置编译指示 {attribute 'monitoring'...}，您可以监控函数调用的结果以及 **Property**（属性）类型的对象中的变量的当前值。

另请参见：

- 引用编程：编译指示 > “监控”属性 [▶ 749]

10.1.1 编程对象中的监控

✓ 您已将 1 个项目加载到控制器上并启动它。
您希望在编程模块的编辑器中观察控制器上的变量在 TwinCAT 开发系统中所采用的当前值。

1. 在选项卡 **Monitoring**（监控）上的类别 **TwinCAT > PLC Environment > Text editor**（TwinCAT > PLC 环境 > 文本编辑器）中的菜单 **Tools > Options**（工具 > 选项）中，确保选项 **Enable inline monitoring**（启用内联监控）已被激活。
2. 在菜单 **PLC > Display Mode**（PLC > 显示模式）中选择命令 **Decimal**（十进制），设置值的显示格式。
3. 双击解决方案资源管理器 PLC 项目树中的 POU，打开相关编辑器。

⇨ 您可以在声明部分和实现部分看到变量值的表示方法。此外，您还会看到相关的一般描述，以及取决于 POU 类型和编程编辑器的特殊功能。

声明编辑器中的监控

变量的当前值显示在 **Value**（值）列中。您可以写入并强制在 **Prepared value**（准备值）列中输入的值。强制值前有红色符号  。

MAIN [Online]

TwinCAT_Device.Project3.MAIN

Expression	Type	Value	Prepared value	Address	Comment
nVar2	INT	F 2411		%IB10	
bVar3	BOOL	FALSE	TRUE	%IX0.0	
aVar	ARRAY [0..3] O...				
aVar[0]	INT	0			
aVar[1]	INT	0			
aVar[2]	INT	0			
aVar[3]	INT	0			
stMyStruct	ST_MyStruct				
nTest1	INT	0			
nTest2	INT	0			
fbSample	FB_Sample				instance of function block FB_Sample
nIn	INT	0			
nOut	INT	0			
nVar1	INT	0			
nRes	INT	0			

接口引用的打印输出是可扩展的。如果接口指向 1 个全局实例，则该全局实例会显示为引用下的第 1 个条目。如果随后接口引用发生变化，显示的引用就会折叠。

另请参见：

- 声明编辑器 [▶ 569]
- 强制和写入变量 [▶ 198]

实现过程中的监控（内联监控）

在实现部分显示当前变量值被称为内联监控。根据编程语言的不同，可能出现以下显示内容：

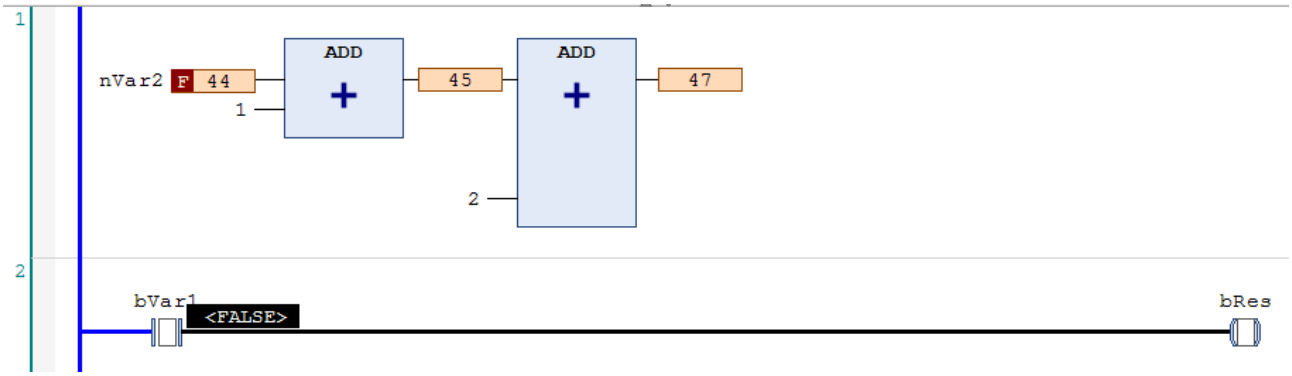
- 变量有 1 个窗口，在其名称后面显示当前值： nResult 17
  如果您为变量准备了强制值或写入值，这些值将在内联监控窗口中显示，在当前值后面用尖括号标出。

  强制后，受影响的值将用符号 F 标出
- 网络编辑器和 CFC：
  连接线（信号线）根据其 Boolean 实际值采用颜色编码：蓝色表示 TRUE，黑色表示 FALSE。
- LD 编辑器：
  此外，触点和线圈元素均有标记。对于触点或线圈，在元素旁边的小窗口中会显示 1 个准备值（TRUE 或 FALSE）。
- SFC 编辑器：
  值为 TRUE 的转换根据其 Boolean 实际值采用颜色编码：蓝色表示 TRUE，黑色表示 FALSE。
  活动步骤被标记为蓝色。
  在实现中，强制转换值被标记为红色。
- IL 编辑器：
  在单独 1 列中显示当前值

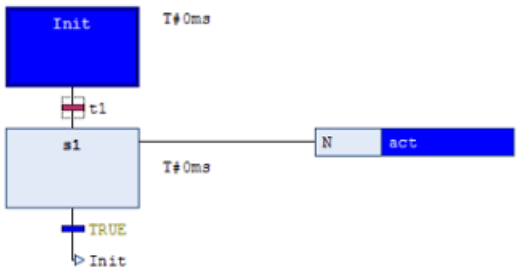
ST 编辑器中的监控：

```
1  nVar2 F 44 := nVar2 F 44 + 1; (*counter*)
2  bVar3 TRUE := TRUE;
3  stMyStruct.nTest1 45 := nVar2 F 44;
4  aVar[2] 45 := nVar2 F 44;
5  nRes 0 := fbSample.nOut 0;
6
```

LD 编辑器中的监控：



SFC 编辑器中的监控：



您可以在此处停用内联监控功能：菜单 **Tools** > **Options**（工具 > 选项），类别 **TwinCAT** > **PLC Environment** > **Text editor**（TwinCAT > PLC 环境 > 文本编辑器），选项卡 **Monitoring**（监控）。

另请参见：

- 引用编程：在线模式下的 ST 编辑器 [▶ 571]
- 引用编程：在线模式下的 FBD/LD/IL 编辑器 [▶ 598]
- 引用编程：在线模式下的 SFC 编辑器 [▶ 581]
- 引用编程：在线模式下的 CFC 编辑器 [▶ 613]

部分监控数组

经过扩展后的数组最多可显示 1000 个元素的实际值。这可能会令人感到困惑。此外，1 个数组可以包含 1000 多个元素。那么，限制显示元素的范围就很有帮助。在在线操作过程中，您可以通过以下方式做到这一点。

✓ 您已将 1 个项目加载到控制器上，其中声明了 1 个包含超过 1000 个元素的多维数组变量。
示例：aSample : ARRAY [1..100, 1..10, 1..20] OF INT;

1. 对于变量 aSample，点击 **Data type**（数据类型）列的字段。
   ⇨ **Monitoring area**（监控区域）对话框会打开。
2. 在 **Start**（开始）处，输入值 [1, 0, 0]。
3. 在 **End**（结束）处，输入值 [1, 10, 20]。

⇨ 显示 200 个数组元素的实际值。范围局限于索引为 [1, <i>, <j>] 的元素。

监控功能块

如果您要在在线模式下打开功能块的编辑器视图，系统将会询问您是想要打开基本实现的视图，还是想要打开功能块实例的视图。只有设置断点后，才能在基本实现中进行监控。如果您在基本实现中设置断点，则会在所有实例中进行设置。在基本实现视图中会首先显示在程序序列中最先执行的实例的值。

如果您在在线操作过程中双击功能块的编辑器视图，则会出现 1 个对话框，您可以在该对话框中在基本实现视图或特定实例视图之间进行选择。

如果您选择基本实现，则在编辑器中会显示代码，但不会显示当前值。现在，在基本实现中设置 1 个断点。如果在那里停止执行，则会显示在程序流程中最先处理的实例的当前值。现在，您可以依次遍历所有实例。

如果您选择其中 1 个实例，则会打开包含功能块实例代码的编辑器。在声明中会显示当前值并不断进行更新，如有必要，在实现中也会如此。

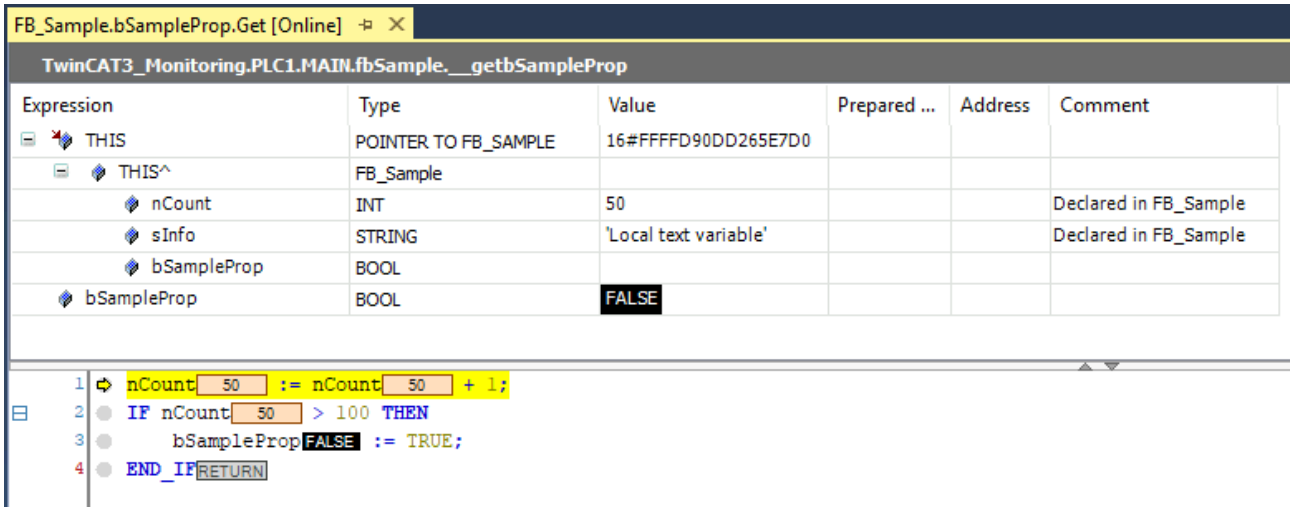
另请参见：

- 对象功能块 [▶ 76]
- 使用断点 [▶ 195]

监控属性

当您在在线操作过程中设置断点时，您可以监控属性对象  中的变量。在那里停止时，将会显示当前值。

除了自身值之外，还会自动显示上一级实例的变量值。在属性的声明部分中，指针 THIS 会出现在第 1 行，指向上一级实例，并带有数据类型规范和当前值。



另请参见：

- 对象属性 [▶ 88]

监控上一级编程对象中的属性访问

除了变量值之外，您还可以监控功能块或程序中的下一级属性  的值。

为此，可在声明中为下一级属性对象插入编译指示 {attribute 'monitoring' = 'variable'} 或编译指示 {attribute 'monitoring' = 'call'}。如果您在运行时打开上一级程序或上一级功能块实例，那么，除了当前的变量值之外，在编辑器中还会显示当前的属性值。

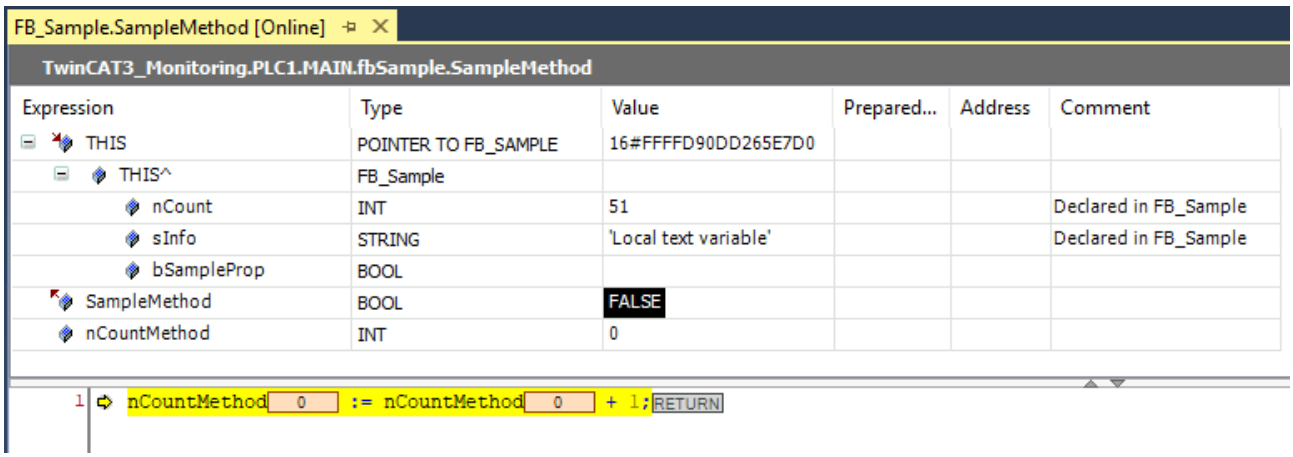
另请参见：

- 引用编程：编译指示 > “监控”属性 [▶ 749]

监控方法

如果您在在线操作过程中在方法中设置断点，则您可以监控方法对象  中的变量。在那里停止时，将会显示当前值。

除了自身值之外，还会自动显示上一级实例的变量值。为此，在方法的声明部分中，指针 THIS 会出现在第 1 行，指向上一级实例，并带有数据类型规范和当前值。



另请参见：

- 对象方法 [▶ 82]

监控函数

如果您在在线操作过程中在函数中设置断点，则您可以监控函数对象中的变量。在那里停止时，将会显示当前值。

- 对象函数 [▶ 73]

监控函数调用的返回值

在 ST 编辑器中，如果满足以下条件，在内联监控中将会显示函数的当前返回值，而不是调用函数的 POU：

- 它是 1 个可被解释为 4 字节数值的值（例如，INT、SINT 或 LINT）。
- 在函数声明中插入编译指示 {attribute 'monitoring' := 'call'}。

另请参见：

- 引用编程：编译指示 > 属性“monitoring” [▶ 749]

10.1.2 使用监视列表

什么是监视列表？

监视列表是用户定义的项目变量列表，在视图中可以收集这些变量，以监控它们的值。您可以在在线模式下在监视列表中写入和强制变量值。

在项目中的菜单 **PLC > Windows**（PLC > 窗口）中，有 4 个监视列表 **Watch <n>**（监视 <n>）可供填写。

创建和编辑监视列表（离线或在线模式）

✓ 在离线或在线模式下打开项目。在项目中声明您想要将其置于 4 个监视列表之一的变量。

1. 在菜单 **PLC > Windows**（PLC > 窗口）中，选择命令 **Watch <n>**（监视 <n>）。
   ⇨ 视图 **Watch <n>**（监视 <n>）会出现。它包含 1 个空表行。
2. 双击 **Expression**（表达式）列中的字段后，手动或使用 **Input Assistant**（输入助手）输入要监控的变量。
   语法：<Device name>.<Project name>.<Object name>.<Variable name>
   示例：“TwinCAT_Device.Project5.MAIN.nVar”
   如果您输入结构化变量的名称，则在在线模式下，在其他行中会自动显示各个组件。
3. 用该列表相继定义所有要监控的变量。您可以通过拖放来更改顺序。

⇨ 根据变量声明，字段 **Execution point**（执行点）、**Type**（类型）、**Address**（地址）、**Comment**（注释）和 **Value**（值）会自动填充。表达式前面的符号可指明该变量是 输入变量、 输出变量，还是 “正常”变量。

在在线模式下，您还可以借助上下文菜单命令 **Add Watch**（添加监视）来创建新的监视列表或编辑现有监视列表。

另请参见：

- TC3 用户界面文档：命令：监视列表 <n> [▶ 875]

借助添加监视命令来添加变量（在线模式）

✓ 在操作期间打开 1 个项目。在该项目中已声明您想要将其置于监视列表中的变量。

1. 在菜单 **PLC** > **Windows**（PLC > 窗口）中，使用命令 **Watch <n>**（监视 <n>）打开所需的监视列表。
2. 将光标置于 POU 的声明或实现部分中的变量上，然后从上下文菜单中选择命令 **Add Watch**（添加监视）。
   ⇨ 所选变量的条目已被添加到列表中。
3. 您可以通过这种方式添加更多变量，或者您可以在字段 **Expression**（表达式）中的列表中输入它们，如上文所述（**创建和编辑监视列表**）。
   ⇨ 监视列表会立即更新。

如果您在对变量执行命令 **Add Watch**（添加监视）时没有打开监视列表，则该变量会被自动添加到“Watch 1”列表中。

您也可以在监视列表中写入和强制变量值。对于这种情况，在在线模式下会出现 **Prepared value**（准备值）列。

另请参见：

- 强制和写入变量值 [▶ 198]
- TC3 用户界面文档：命令：监视列表 <n> [▶ 875]
- TC3 用户界面文档：命令：添加至监视 [▶ 815]

10.2 更改带配方的值

您使用配方可以同时更改或读取控制器上的 1 组特定变量（配方定义）的值。

配方的基本设置（例如，存储位置和格式）均在对象 **Recipe Manager**（配方管理器）中进行定义。在该对象下方，您可以添加 1 个或多个配方定义。1 个 **Recipe Definition**（配方定义）包含 1 个或多个配方，可用于其中的变量。配方由特定的变量值组成。

您可以将配方保存到文件中，或者从文件中直接将其写入控制器。

通过 TwinCAT 编程界面、可视化元素或应用程序可以加载配方。

为了在您的应用程序中使用配方，建议通过低优先级任务调用该程序段。

10.2.1 对象配方管理器

符号： 

配方管理器提供了用于管理用户定义的变量列表的功能：即所谓的配方定义。配方定义可存储在“Recipe files”（配方文件）中。

如果您在 PLC 项目中添加配方管理器，则在库管理器中会自动添加 RecipeManagement 库。

创建对象配方管理器

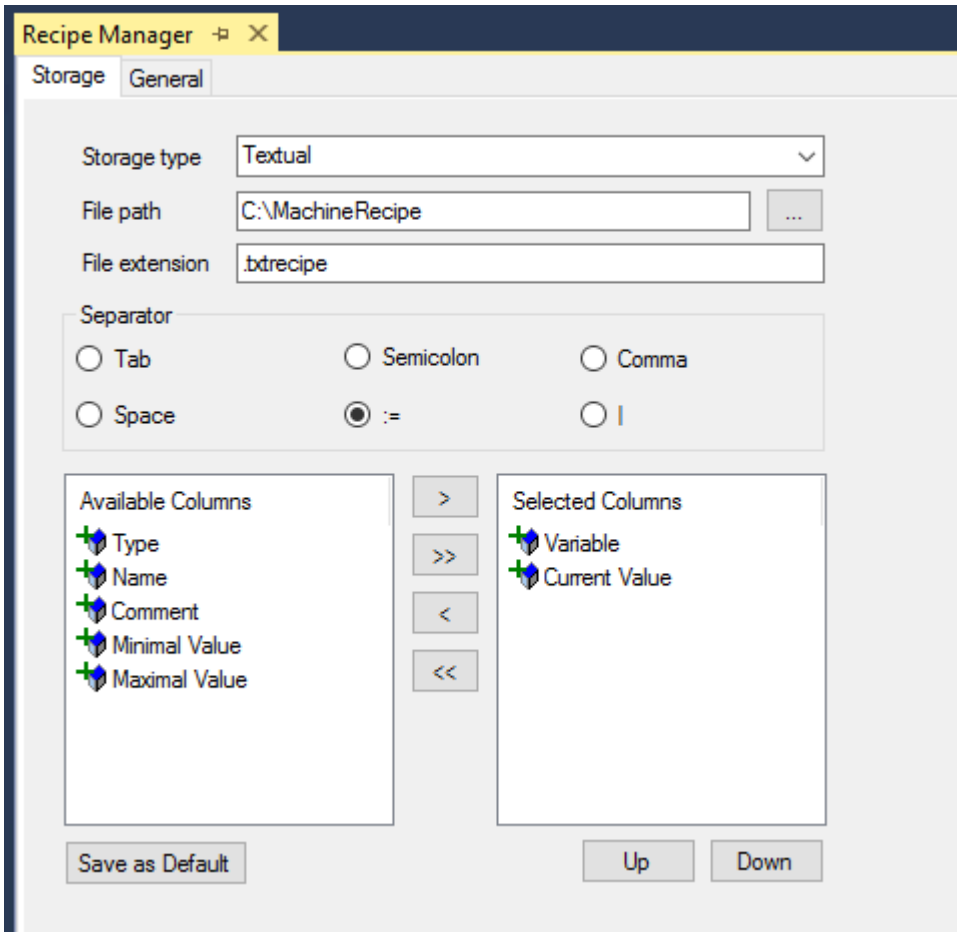
1. 在 Solution Explorer（解决方案资源管理器）中，选择 PLC 项目。
2. 在上下文菜单中，选择命令 Add > Recipe Manager...（添加 > 配方管理器……）。
   ⇨ 对话框 Add Recipe Manager（添加配方管理器）会打开。
3. 输入名称。
4. 点击 Open（打开）。

⇨ 配方管理器被添加到 PLC 项目树中，并在编辑器中打开。随后，您可以将对象 Recipe Definition（配方定义）添加到对象 Recipe Manager（配方管理器）中。

配方管理器的结构

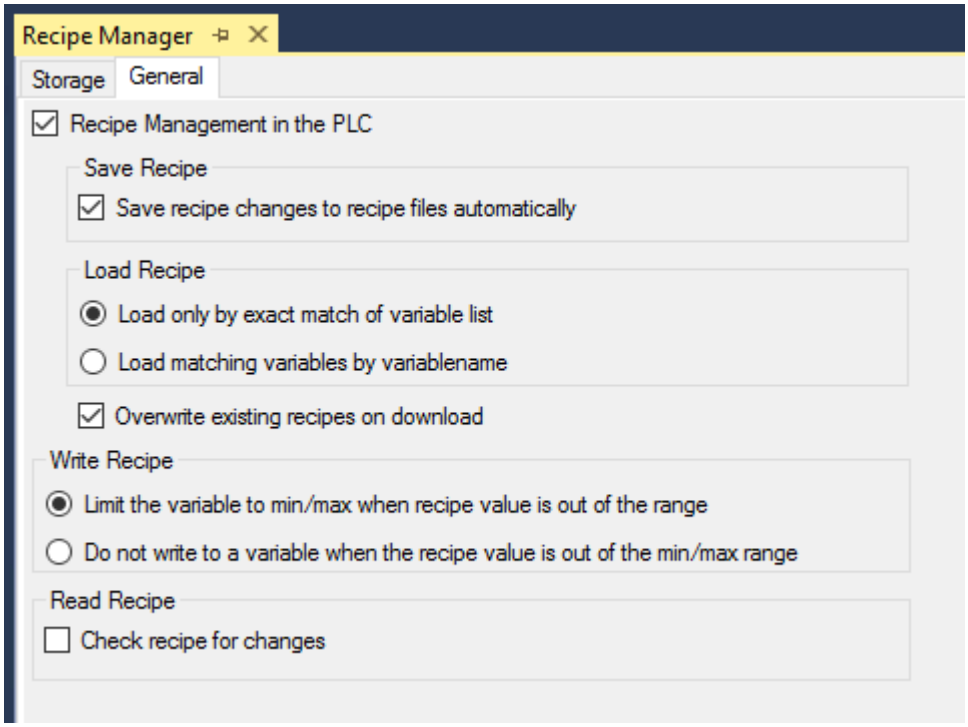
配方管理器的编辑器由 2 个选项卡 Storage（存储）和 General（常规）组成，在这 2 个选项卡中确定了配方管理器的设置。

选项卡存储



存储类型	**文本：** TwinCAT 使用配置的列和分隔符以可读格式保存配方。 **二进制：** TwinCAT 以不可读的二进制格式保存配方。这种格式所需的存储空间较小。 只有在您未更改变量列表的情况下，您才能回读以二进制格式保存的配方。
文件路径	运行时系统上的相对路径。 该路径是在目标系统的运行时文件目录下创建的。在下载到控制器上时，TwinCAT 会为该目录下的每个配方创建 1 个文件。前提条件是已激活选项 **Recipe Management in the PLC**（PLC 中的配方管理）。 每次重新启动 PLC 项目时，这些文件都会被加载到配方管理器中。
文件扩展名	配方文件的文件扩展名。 这样，配方文件的标准名称形式为 <Recipe>.<Recipe definition>.<File extension>。
分隔符	存储文件中的各个值之间的分隔符。
可用列 选定列	说明在配方文件中按照什么顺序保存哪些信息的定义。
另存为默认值	TwinCAT 立即将对话框中的设置用作默认设置。

常规选项卡



PLC 中的配方管理	☑：如果通过可视化元素或用户程序在运行时加载配方，则必须激活。如果配方完全通过 TwinCAT 编程接口传输到控制器，则可以停用该选项。

保存配方

自动将配方更改保存到配方文件中	☑：推荐选项，因为该选项会导致配方管理的“常规”行为。在运行时，配方管理器会在配方发生变化时自动将配方保存到文件中；即在运行时，每当配方发生变化时，内存文件都会自动更新。 只有激活 PLC 中的配方管理选项，该选项才生效。

加载配方

仅通过变量列表的精确匹配进行加载	只有当文件包含 PLC 项目配方定义的变量列表中的所有变量且这些变量的排序相同时，才会加载配方。末尾的附加条目将被忽略。如果没有必要的匹配，则会设置错误状态 ERR_RECIPE_MISMATCH（RecipeManCommands.GetLastError）。
按 variablename 加载匹配的变量	仅为 PLC 项目配方定义中与配方文件中名称相同的变量加载配方值。如果变量列表的组成和排序不同，则不会设置错误状态。 这样，即使已从文件或配方定义中删除变量，也可以加载配方文件。

在下载过程中覆盖现有配方	☑：如果控制器上存在同名的配方文件，则在启动 PLC 项目时，这些文件将被项目中的配置值覆盖。如果要从现有配方文件中加载值，则必须停用该选项。 前提条件：存储类型为**文本**，且选项 **Save recipe changes to recipe files automatically**（自动将配方更改保存到配方文件中）已激活。

写入配方

如果配方值超出范围，则将变量限制为最小值/最大值	如果配方中包含的值超出在定义中输入的值范围，则会将定义的最小值或最大值写入 PLC 变量，而不是该值。
如果配方值超出最小/最大范围，则不写入变量	如果配方中包含的值超出在定义中输入的值范围，则不会将任何值写入 PLC 变量。它会保持当前值。

读取配方

仅通过变量列表的精确匹配进行加载	☑：每次调用方法时，首先将当前 PLC 变量值读入配方。然后，检查这些值是否发生变化。只有当这些值发生变化时，才会保存配方，即用当前配方覆盖配方文件。 只有当 PLC 上的配方值发生变化时，才能使用该选项更新本地文件系统中的配方文件。不过，这会影响性能，因为检查会生成额外的代码。 ☐：每次调用方法时，首先将当前 PLC 变量值读入配方。然后，将配方写入本地文件系统中的配方文件。 由于每次调用都会写入文件系统，因此可以加载控制器。

如果选项 **Save recipe changes to recipe files automatically**（自动将配方更改保存到配方文件中）已激活，则在线模式下的配方：

动作	在项目中定义的配方	在运行时定义的配方
在线暖重置 在线冷重置 下载	为所有配方定义的配方分配当前项目中的值。	动态生成的配方保持不变。
在线重置原点	从 PLC 中删除应用程序。如果在此之后进行新的下载，则会恢复配方，与在线暖重置的情况一样。	
关闭和重新启动 PLC	在重新启动之后，将从自动创建的文件中重新加载配方。这样就会恢复到与关闭前相同的状态。	
在线更改	配方值保持不变。在运行时只能使用功能块 RecipeManCommands 中的命令更改配方。	
停止/启动	在停止/启动 PLC 的情况下，配方保持不变。	

如果选项 **Save recipe changes to recipe files automatically**（自动将配方更改保存到配方文件中）未激活，则在线模式下的配方：

动作	在项目中定义的配方	在运行时定义的配方
在线暖重置 在线冷重置 下载	为所有配方定义的配方分配当前项目中的值。不过，它们只能在内存中进行设置。务必明确使用“Save recipe”（保存配方）命令，才能将配方存储到文件中。	动态生成的配方将会丢失。
在线重置原点	从 PLC 中删除应用程序。如果在此之后进行新的下载，则会恢复配方。	动态生成的配方将会丢失。
关闭和重新启动 PLC	在重新启动之后，将从自动创建的文件中重新加载配方。这样就会恢复到与关闭前相同的状态。	
在线更改	配方值保持不变。在运行时只能使用功能块 RecipeManCommands 中的命令更改配方。	
停止/启动	在停止/启动 PLC 的情况下，配方保持不变。	

10.2.2 对象配方定义

符号： 

在配方定义中，您可以为变量定义各种值集，这些值集被称为配方。

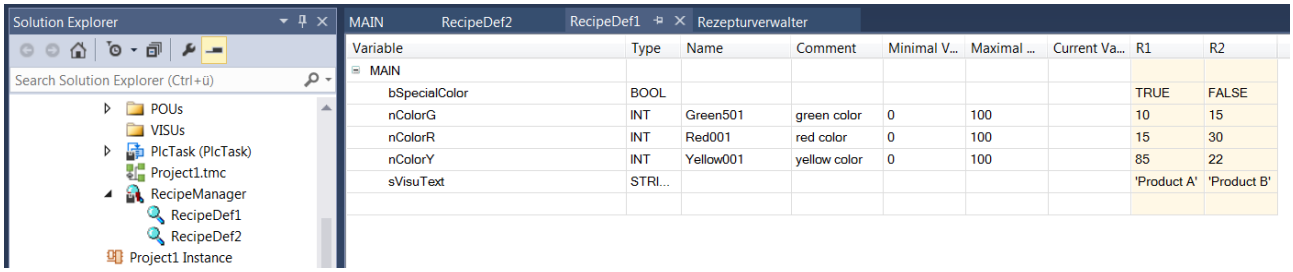
创建对象配方定义

1. 在 PLC 项目树的 **Solution Explorer**（解决方案资源管理器）中，选择对象 **Recipe Manager**（配方管理器）。
2. 在上下文菜单中，选择命令 **Add** > **Recipe Definition...**（添加 > 配方定义……）。
   ⇨ 对话框 **Add Recipe Definition**（添加配方定义）会打开。
3. 输入名称。
4. 点击 **Open**（打开）。

⇨ 配方定义被添加到配方管理器下方，并在编辑器中打开。

配方定义的结构

在配方定义编辑器中，值集以表格形式显示。您可以在简单视图和结构化视图之间切换配方定义的视图。TwinCAT 在结构化视图中显示属于组合在一起的结构的变量。



变量	变量名
类型	自动输入
名称	可选
最小值 最大值	如果变量值小于最小值或大于最大值，则 TwinCAT 会将其分别设置为最小值或最大值。
注释	附加信息，例如，值的单位。
当前值	当前变量值；仅在在线模式下显示。
配方（例如，R1）	配方的变量值

10.2.3 使用配方

在 TwinCAT 用户界面中处理配方

TwinCAT 编程界面为您提供了用于创建配方以及在在线模式下进行读取/写入的命令。

另请参见：

- TC3 用户界面文档：配方 [▶ 975]

在应用程序中使用配方

在运行时，您可以在用户程序中或通过可视化元素使用配方。

在用户程序中，您可以使用 RecipeManagement 库中的功能块 RecipeManCommands 的方法。在可视化系统中，通过可视化元素的输入配置（内部命令）可以使用配方。

在初始化过程中，配方管理功能会读取在配方定义中进行定义的变量值。在应用程序初始化阶段结束时会进行该过程。此时，应用程序变量的所有初始值均已设定。这样做是为了正确初始化配方文件中的缺失值。

另请参见：

- 库配方管理 - RecipeManCommands [▶ 216]

创建配方

1. 在 **Solution Explorer**（解决方案资源管理器）中，选择 PLC 项目。
2. 在上下文菜单中，选择命令 **Add** > **Recipe Manager...**（添加 > 配方管理器……）。
   ⇨ TwinCAT 会将配方管理器添加到 PLC 项目中。
3. 在 PLC 项目树中，选择对象 **Recipe Manager**（配方管理器）。
4. 在上下文菜单中，选择命令 **Add** > **Recipe Definition...**（添加 > 配方定义……）。
   ⇨ TwinCAT 会在配方管理器下方添加配方定义。
5. 双击对象，打开配方定义编辑器。
6. 双击编辑器中的 **Variable**（变量）列中的空字段。输入您想要为其定义配方的变量的名称。输入助手可用于此目的（按钮 ）。
7. 在菜单 **Recipe**（配方）或编辑器的上下文菜单中，选择命令 **Add Recipe**（添加配方），并输入新配方的名称（例如，“MyRec”）。
   ⇨ 在编辑器中会出现配方名称列。
8. 输入此配方的变量值。
9. 如有必要，可添加更多变量。
10. 选择配方的 1 个变量值，然后在菜单 **Recipe**（配方）或上下文菜单中执行命令 **Save Recipe**（保存配方）。选择内存位置和文件名。

⇨ TwinCAT 会按照在配方管理器中定义的格式保存配方。

覆盖隐式配方文件

不得覆盖作为读取和写入配方的剪贴板的隐式配方文件。这意味着该文件名必须与 *<Recipe name>.<Recipe definition name>.txtrecipe* 不同

另请参见：

- TC3 用户界面文档：命令：添加新配方 [▶ 975]
- TC3 用户界面文档：命令：保存配方 [▶ 976]

从文件加载配方

✓ 在 PLC 项目中有配方管理功能。在配方定义中有 1 个配方“MyRec”带有多个变量值。在文件系统中有 1 个配方文件 *MyRec.txt*，其中包含该配方的条目。

1. 在 PLC 项目树中，双击对象 **Recipe Definition**（配方定义），打开用于定义单个配方的表格编辑器。
   ⇨ 您将会看到 **MyRec** 列，其中包含该配方的当前值。
2. 在外部文本编辑器中编辑文件 *MyRec.txt*，并将变量值替换为您想要加载到 TwinCAT 配方定义中的其他值。保存文件。
3. 点击配方定义中的 **MyRec** 列，然后在菜单 **Recipe**（配方）或上下文菜单中选择命令 **Load Recipe**（加载配方）。
   ⇨ 对话框 **Load Recipe**（加载配方）会打开。
4. 从文件资源管理器中选择文件 *MyRec.txt* 进行加载。

⇨ 配方定义中的配方值会根据在文件中读取的值进行更新。如果您通过加载配方更改了配方变量的当前值，那么，在您下次登录时，系统会显示 1 条查询消息，询问您在登录后想要进行在线更改、下载，还是不进行更改。

配方文件的示例：

```
MAIN.nVar1:=0
MAIN.nVar2:=2
MAIN.nVar3:=35232
MAIN.sVar4:='first'
MAIN.wsVar5:='123443245'
```

如果您只想用新值覆盖单个配方变量，请在将配方加载到配方文件中之前删除其他变量的值。不读取没有值规范的条目，这意味着这些变量不受控制器上和项目中更新的影响。

对于 REAL/LREAL 数据类型的值，在某些情况下也会将十六进制值写入配方文件。这样做很有必要，以便在反向转换过程中恢复完全相同的值。在这种情况下，更改十进制值并删除十六进制值。

另请参见：

- TC3 用户界面文档：命令：加载配方 [▶ 975]

读取配方

✓ TwinCAT 处于在线模式。

1. 点击配方定义中的配方列，然后在菜单 **Recipe**（配方）或上下文菜单中选择命令 **Read Recipe**（读取配方）。

⇨ TwinCAT 用从控制器读取的值覆盖所选配方的值。在此过程中，值被隐式存储（在控制器的文件中），并同时显示在配方定义表中。

另请参见：

- TC3 用户界面文档：命令：读取配方 [▶ 976]
- TC3 用户界面文档：命令：读取和保存配方 [▶ 977]

写入配方

✓ TwinCAT 处于在线模式。

1. 点击配方定义中的配方列，然后在菜单 **Recipe**（配方）或上下文菜单中选择命令 **Write Recipe**（写入配方）。

⇨ TwinCAT 用所选配方中的值覆盖控制器中的值。

另请参见：

- TC3 用户界面文档：命令：写入配方 [▶ 976]
- TC3 用户界面文档：命令：加载和写入配方 [▶ 977]

10.2.4 库配方管理 - RecipeManCommands

库“配方管理”中的功能块 RecipeManCommands 的方法可实现以编程方式管理配方。

应用程序会在控制器上自动创建名称为 *<Recipe>.<Recipe definition>.txtrecipe* 的配方文件。它可以作为用于读取和写入配方变量的剪贴板。选项卡 **Recipe Manager > General**（配方管理器 > 常规）中的选项 **Save recipe changes to recipe files automatically**（自动将配方更改保存到配方文件中）会影响对这些文件的访问。

如果选项 **Save recipe changes to recipe files automatically**（自动将配方更改保存到配方文件中）已激活，则在 TwinCAT 中定义的配方和控制器上的隐式配方文件将自动保持一致。另外，更改配方也会导致文件访问。

方法 CreateRecipe

该方法可在指定的配方定义中创建 1 个新配方。随后，它会将当前 PLC 值读入新配方，并以标准名称将其保存为配方文件。标准名称为 <Recipe>.<Recipe definition>.<Recipe extension>。

输入（VAR_INPUT）

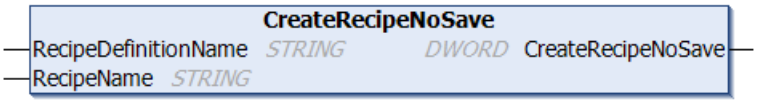
名称	数据类型	描述
RecipeDefinitionName	STRING	配方定义的名称
RecipeName	STRING	配方的名称

返回值

名称	数据类型	描述
CreateRecipe	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_ALREADY_EXIST ERR_RECIPE_NOMEMORY ERR_NO_RECIPE_MANAGER_SET ERR_OK

方法 CreateRecipeNoSave

该方法可在指定的配方定义中创建 1 个新配方。随后，它会将实际值读入新配方。



输入（VAR_INPUT）

名称	数据类型	描述
RecipeDefinitionName	STRING	配方定义的名称
RecipeName	STRING	配方的名称

返回值

名称	数据类型	描述
CreateRecipeNoSave	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_ALREADY_EXIST ERR_RECIPE_NOMEMORY ERR_NO_RECIPE_MANAGER_SET ERR_OK

方法 DeleteRecipe

该方法可从配方定义中删除配方。

输入（VAR_INPUT）

名称	数据类型	描述
RecipeDefinitionName	STRING	配方定义的名称
RecipeName	STRING	配方的名称

返回值

名称	数据类型	描述
DeleteRecipe	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_NOT_FOUND ERR_NO_RECIPE_MANAGER_SET ERR_OK

方法 DeleteRecipeFile

该方法可删除配方的指定配方文件。配方文件必须以标准名称 <Recipe>.<Recipe definition>.<Recipe extension> 存储。

输入（VAR_INPUT）

名称	数据类型	描述
RecipeDefinitionName	STRING	配方定义的名称
RecipeName	STRING	配方的名称

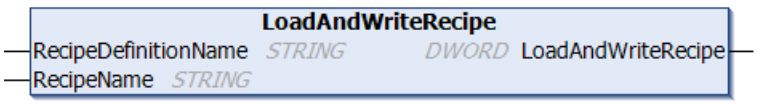
返回值

名称	数据类型	描述
DeleteRecipeFile	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_NOT_FOUND ERR_RECIPE_FILE_NOT_FOUND ERR_NO_RECIPE_MANAGER_SET ERR_OK

方法 LoadAndWriteRecipe

该方法可从指定的配方文件加载配方。配方文件必须以标准名称 <Recipe>.<Recipe definition>.<Recipe extension> 存储。随后，它会将配方写入 PLC 变量。

配方文件中不包含赋值的条目不会被加载和写入。请参见关于菜单命令 **Load and write Recipe**（加载和写入配方）的描述。



输入（VAR_INPUT）

名称	数据类型	描述
RecipeDefinitionName	STRING STRING	配方定义的名称
RecipeName	STRING	配方的名称

返回值

名称	数据类型	描述
LoadAndWriteRecipe	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_NOT_FOUND ERR_RECIPE_FILE_NOT_FOUND ERR_RECIPE_MISMATCH ERR_RECIPE_NOT_ALL_VARIABLES_WERE_LOADED ERR_NO_RECIPE_MANAGER_SET ERR_OK

方法 LoadFromAndWriteRecipe

该方法可将指定的配方文件加载到配方中。随后，它会将配方写入 PLC 变量。

> 配方文件中不包含赋值的条目不会被加载和写入。请参见关于菜单命令 **Load and write Recipe**（加载和写入配方）的描述。

输入（VAR_INPUT）

名称	数据类型	描述
RecipeDefinitionName	STRING	配方定义的名称
RecipeName	STRING	配方的名称
FileName	STRING (255)	文件的名称

返回值

名称	数据类型	描述
LoadFromAndWriteRecipe	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_NOT_FOUND ERR_RECIPE_FILE_NOT_FOUND ERR_RECIPE_MISMATCH ERR_RECIPE_NOT_ALL_VARIABLES_WERE_LOADED ERR_NO_RECIPE_MANAGER_SET ERR_OK

另请参见：

- TC3 用户界面文档：命令：加载和写入配方 [▶ 977]

方法 LoadRecipe

该方法可从配方文件加载配方。配方文件必须以标准名称 <Recipe>.<Recipe definition>.<Recipe extension> 存储。

> 配方文件中不包含赋值的条目不会被加载和写入。请参见关于菜单命令 **Load and write Recipe**（加载和写入配方）的描述。

输入（VAR_INPUT）

名称	数据类型	描述
RecipeDefinitionName	STRING	配方定义的名称
RecipeName	STRING	配方的名称

返回值

名称	数据类型	描述
LoadRecipe	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_NOT_FOUND ERR_RECIPE_FILE_NOT_FOUND ERR_RECIPE_MISMATCH ERR_RECIPE_NOT_ALL_VARIABLES_WERE_LOADED ERR_NO_RECIPE_MANAGER_SET ERR_OK

另请参见：

- TC3 用户界面文档：命令：加载和写入配方 [▶ 977]

方法 ReadAndSaveAs

该方法可从配方定义的变量中读取当前 PLC 值，并将此数据集保存到配方文件中，而无需更改现有的标准配方文件 <recipe.recipedefinition.extension>。

输入（VAR_INPUT）

名称	数据类型	描述
RecipeDefinitionName	STRING	配方定义的名称 读取在配方定义中指定的变量。
FileName	STRING(255)	文件的名称。当前读取的数据集可作为配方保存在文件中。

返回值

名称	数据类型	描述
ReadAndSaveAs	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_SAVE_ERR ERR_NO_RECIPE_MANAGER_SET ERR_OK

方法 ReadAndSaveRecipe

该方法可将当前 PLC 值读入配方。随后，它会将配方保存到 1 个具有标准名称的配方文件中。标准名称为 <Recipe>.<Recipe definition>.<Recipe extension>。任何现有文件的内容都会被覆盖。

输入（VAR_INPUT）

名称	数据类型	描述
RecipeDefinitionName	STRING	配方定义的名称
RecipeName	STRING	配方的名称

返回值

名称	数据类型	描述
ReadAndSaveRecipe	DWORD	返回值，可能值： ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_NOT_FOUND ERR_RECIPE_SAVE_ERR ERR_NO_RECIPE_MANAGER_SET ERR_OK

方法 ReadAndSaveRecipeAs

该方法可将当前 PLC 值读入配方。随后，它会将配方保存到指定的配方文件中。任何现有文件的内容都会被覆盖。

输入（VAR_INPUT）

名称	数据类型	描述
RecipeDefinitionName	STRING	配方定义的名称
RecipeName	STRING	配方的名称
FileName	STRING	文件的名称

返回值

名称	数据类型	描述
ReadAndSaveRecipeAs	DWORD	返回值，可能值： ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_NOT_FOUND ERR_RECIPE_SAVE_ERR ERR_NO_RECIPE_MANAGER_SET ERR_OK

方法 SaveRecipe

该方法可将配方保存到 1 个具有标准名称的配方文件中。标准名称为 <Recipe>.<Recipe definition>.<Recipe extension>。任何现有文件的内容都会被覆盖。

输入（VAR_INPUT）

名称	数据类型	描述
RecipeDefinitionName	STRING	配方定义的名称
RecipeName	STRING	配方的名称

返回值

名称	数据类型	描述
SaveRecipe	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_NOT_FOUND ERR_RECIPE_SAVE_ERR ERR_NO_RECIPE_MANAGER_SET ERR_OK

方法 ReadRecipe

该方法可将当前 PLC 值读入配方。

输入（VAR_INPUT）

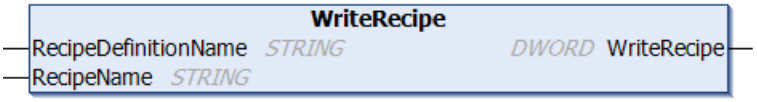
名称	数据类型	描述
RecipeDefinitionName	STRING	配方定义的名称
RecipeName	STRING	配方的名称

返回值

名称	数据类型	描述
ReadRecipe	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_NOT_FOUND ERR_NO_RECIPE_MANAGER_SET ERR_OK 数据服务器错误，从 16#2000 到 16#20FF 数据源驱动程序错误，从 16#2100 到 16#21FF

方法 WriteRecipe

该方法可将配方写入 PLC 变量。



输入（VAR_INPUT）

名称	数据类型	描述
RecipeDefinitionName	STRING	配方定义的名称
RecipeName	STRING	配方的名称

返回值

名称	数据类型	描述
WriteRecipe	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_NOT_FOUND ERR_NO_RECIPE_MANAGER_SET ERR_OK

方法 ReloadRecipes

该方法可从文件系统中读取配方列表。因此，将会考虑在配方管理器中定义的路径中的标准名称为 <Recipe>.<Recipe definition>.<Recipe extension> 的配方。

输入（VAR_INPUT）

名称	数据类型	描述
RecipeDefinitionName	STRING	配方定义的名称

返回值

名称	数据类型	描述
ReloadRecipes	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_NO_RECIPE_MANAGER_SET ERR_OK

方法 GetRecipeCount

该方法可返回配方定义中的配方数量。

输入（VAR_INPUT）

名称	数据类型	描述
RecipeDefinitionName	STRING	配方定义的名称

返回值

名称	数据类型	描述
GetRecipeCount	INT	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_NO_RECIPE_MANAGER_SET ERR_OK

方法 GetRecipeNames

该方法可返回配方定义中的配方名称。

输入（VAR_INPUT）

名称	数据类型	描述
RecipeDefinitionName	STRING	配方定义的名称
pStrings	POINTER TO ARRAY [] OF STRING	指向包含配方名称的数组的指针。
iSize	INT	STRING 数组中的元素数量
iStartIndex	INT	起始索引 示例：1

返回值

名称	数据类型	描述
GetRecipeNames	DWORD	返回值，可能值： ERR_RECIPE_DEFINITION_NOT_FOUND ERR_NO_RECIPE_MANAGER_SET ERR_OK

示例：

例如，有 50 个配方。如果您想要生成 1 个同时显示 10 个配方名称的表格，则您必须定义 1 个 STRING 数组：

```
strArr: ARRAY[0..9] OF STRING;
```

与 iStartIndex 相对应，您就可以获得指定范围内的配方名称。

```
iStartIndex := 0; die Namen 0..9 werden zurückgegeben
iStartIndex := 20; die Namen 20..29 werden zurückgegeben
```

以下内容适用于本示例：

```
iSize := 10;
```

方法 GetRecipeValues

该方法可返回配方的值。

输入（VAR_INPUT）

名称	数据类型	描述
RecipeDefinitionName	STRING	配方定义的名称
RecipeName		配方的名称
pStrings	POINTER TO ARRAY [] OF STRINGS	指向存储配方值的字符串的指针。
iSize	INT	字符串数组的大小。
iStartIndex	INT	起始索引。
iStringLength	INT	数组中的字符串的长度。

返回值

名称	数据类型	描述
GetRecipeValues	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_NOT_FOUND ERR_NO_RECIPE_MANAGER_SET ERR_OK

示例：

例如，有 50 个配方。如果您想要生成 1 个同时显示 10 个配方值的表格，则您必须定义 1 个 STRING 数组：

```
strArr: ARRAY[0..9] OF STRING;
```

与 iStartIndex 相对应，您就可以获得指定范围内的配方值。

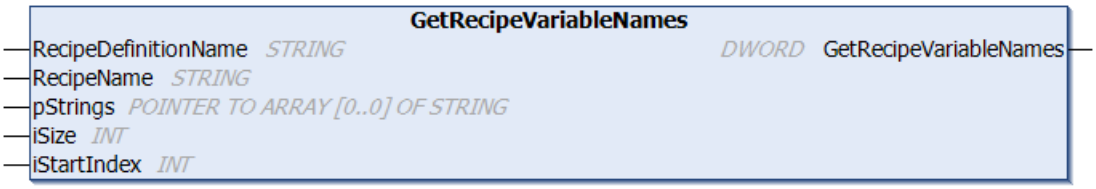
```
iStartIndex := 0; die Werte 0..9 werden zurückgegeben
iStartIndex := 20; die Namen 20..29 werden zurückgegeben
```

以下内容适用于本示例：

```
iStringLength := 80;
iSize := 10;
```

方法 GetRecipeVariableNames

该方法可返回配方的配方变量的名称。



输入（VAR_INPUT）

名称	数据类型	描述
RecipeDefinitionName	STRING	配方定义的名称
RecipeName		配方的名称
pStrings	POINTER TO ARRAY [] OF STRINGS	指向存储配方变量名称的 STRING 数据类型的指针。
iSize	INT	STRING 数组的大小。
iStartIndex	INT	起始索引。

返回值

名称	数据类型	描述
GetRecipeVariableNames	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_NOT_FOUND ERR_NO_RECIPE_MANAGER_SET ERR_OK

示例：

例如，有 50 个配方。如果您想要生成 1 个同时显示配方的 10 个变量名的表格，则您必须定义 1 个 STRING 数组：

```
strArr: ARRAY[0..9] OF STRING;
```

与 iStartIndex 相对应，您就可以获得指定范围内的配方的变量名。

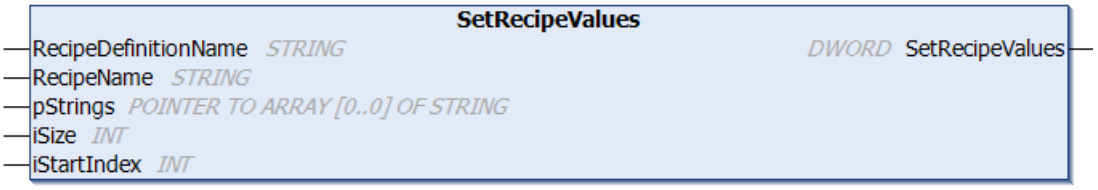
```
iStartIndex := 0; die Namen 0..9 werden zurückgegeben
iStartIndex := 20; die Namen 20..29 werden zurückgegeben
```

以下内容适用于本示例：

```
iSize := 10;
```

方法 SetRecipeValues

该方法可设置配方的配方值。



输入（VAR_INPUT）

名称	数据类型	描述
RecipeDefinitionName	STRING	配方定义的名称
RecipeName		配方的名称
pStrings	POINTER TO ARRAY [] OF STRINGS	指向存储配方值的 STRING 数据类型的指针。
iSize	INT	STRING 数组的大小。
iStartIndex	INT	起始索引。

返回值

名称	数据类型	描述
SetRecipeValues	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_NOT_FOUND ERR_NO_RECIPE_MANAGER_SET ERR_OK

示例：

例如，有 50 个配方。如果您想要生成 1 个同时设置 10 个配方值的表格，则您必须定义 1 个 STRING 数组：

```
strArr: ARRAY[0..9] OF STRING;
```

与 iStartIndex 相对应，您就可以为指定范围设置配方值。

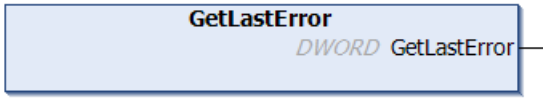
```
iStartIndex := 0; Die Werte 0..9 werden gesetzt
iStartIndex := 20; Die Werte 20..29 werden gesetzt
```

以下内容适用于本示例：

```
iStringLength := 80;
iSize := 10;
```

方法 GetLastError

该方法可返回前一个操作的最后一个错误。



返回值

名称	数据类型	描述
GetLastError	DWORD	返回值，可能值： ERR_NO_RECIPE_MANAGER_SET ERR_OK

方法 ResetLastError

该方法可重置最后一个错误。

返回值

名称	数据类型	描述
ResetLastError	DWORD	ERR_NO_RECIPE_MANAGER_SET ERR_OK

方法 SetStoragePath

该方法可用于设置配方文件的存储路径。它会覆盖配方管理器的对话框“Storage”（存储）中的路径指定。

输入（VAR_INPUT）

名称	数据类型	描述
stPath	STRING	文件路径，示例： • D:/recipefiles/

返回值

名称	数据类型	描述
stPath	BOOL	TRUE（路径已设置） FALSE 可能出现的错误： ERR_OK ERR_NO_RECIPE_MANAGER_SET

返回值

上述函数的返回值包含在 GVL ReturnValues 中。

它们是数据类型为 UDINT 的 InOut 常量。

名称	初始化值	注释
ERR_OK	16#0	成功执行操作。
ERR_FAILED	16#1	操作失败。
ERR_PARAMETER	16#2	参数不正确
ERR_NOTINITIALIZED	16#3	数据服务器对象未初始化。如果将配方管理与 TwinCAT HMI 组合使用，则需要数据服务器。
ERR_NOTIMPLEMENTED	16#C	如果将配方管理与 TwinCAT HMI 组合使用，则数据服务器不会实现所需的接口 IDataServer4。
ERR_NO_OBJECT	16#10	并非配方定义的所有变量都可以通过数据服务器写入。仅可写入有效的变量。
ERR_NOMEMORY	16#11	数据服务器内存不足。
ERR_RECIPE_FILE_NOT_FOUND	16#4000	未找到配方文件。
ERR_RECIPE_MISMATCH	16#4001	配方文件的内容与当前配方不匹配。只有当存储类型为 **textual**（文本）（请参见编辑器 **Recipe Manager**（配方管理器），选项卡 **Storage**（存储）、**Storage Type**（存储类型）），且文件中的变量名与配方定义中的变量名不一致时，才会输出此错误。如果出现此错误，则不会加载配方文件。 可能的原因： 从项目中的配方定义中删除变量。
ERR_RECIPE_SAVE_ERR	16#4002	存储程序失败。 可能的原因 • 由于硬盘已满，无法创建或打开文件。 • 配置的文件路径不存在（请参见编辑器 **Recipe Manager**（配方管理器），选项卡 **Storage**（存储）、**File Path**（文件路径））。 • 运行时系统不允许使用配置的文件扩展名（请参见编辑器 **Recipe Manager**（配方管理器），选项卡 **Storage, File Extension**（存储、文件扩展名））。
ERR_RECIPE_NOT_FOUND	16#4003	配方不存在。
ERR_RECIPE_DEFINITION_NOT_FOUND	16#4004	配方定义不存在。
ERR_RECIPE_ALREADY_EXIST	16#4005	配方定义中已存在该配方。以不同的名称创建新配方。
ERR_NO_RECIPE_MANAGER_SET	16#4006	尚未创建全局配方管理器。 可能的原因： • 在当前 PLC 项目的选项卡 **General**（常规）中，在配方管理器的编辑器中未设置选项 **Recipe management in the PLC**（PLC 中的配方管理）。
ERR_RECIPE_NOT_ALL_VARIABLES_WERE_LOADED	16#4007	配方定义比配方文件包含更多的变量。在这种情况下，无论如何都会写入配方文件的变量值。这一点仅供参考，实际上并非错误。
ERR_RECIPE_NOMEMORY	16#4008	配方定义没有空闲内存来创建新配方。 可能的原因 • 在当前 PLC 项目的选项卡 **General**（常规）中，在配方管理器的编辑器中未设置选项 **Save recipe changes to recipe files automatically**（自动将配方更改保存到配方文件中）。 • 在这种情况下，每个配方定义只能有 50 个配方。如果设置选项 **Save recipe changes to recipe files automatically**（自动将配方更改保存到配方文件中），则不会出现该错误。如果硬盘已满，则会输出错误 ERR_RECIPE_SAVE_ERR。
ERR_RECIPE_MANAGER_LOCKED_DURING_ONLINE_CHANGE	16#4009	在线更改期间，配方管理器已锁定。 可能的原因： 在线更改期间，本应执行一些 RecipeMan 命令，但却没有执行。
ERR_SOURCE_EXHAUSTED	16#40A0	用于 UTF8 助手
ERR_TARGET_EXHAUSTED	16#40A1	用于 UTF8 助手
ERR_SOURCE_ILLEGAL	16#40A2	用于 UTF8 助手

10.3 利用核心转储进行错误分析

核心转储是 PLC 项目数据的转储。

TC3.1 Build 4024.11 及以上可用

核心转储只能与相关的编译信息文件一起使用

如果您存档或保存核心转储文件，请注意，您必须有相关的项目和相关的编译信息文件（*.compileinfo 文件，例如，在创建项目时存储在“_CompileInfo”文件夹中），才能加载核心转储。否则，TwinCAT 以后将无法使用转储。

另请注意此处 设置选项卡 [▶ 859] 上的设置选项。借助 **Core Dump**（核心转储）设置，您可以配置是否将可能位于项目目录下的核心转储文件与可用的编译信息文件一起保存在 TwinCAT 文件存档中。

生成核心转储

1) 自动生成

在出现异常错误时，如果相关的 PLC 项目当前未登录开发环境，则运行时系统会在目标系统上自动存储核心转储。默认情况下，该核心转储会以 *.core 文件的形式存储在目标系统的启动文件夹中。有关（可调节）存储路径的更多信息，请参见下文。如果在 FB_init/FB_reinit/FB_Exit 方法的代码中出现异常错误，也会自动生成核心转储（TC3.1 Build 4024.25 及以上）。

当目标系统上有核心转储时，发出警告消息

如果您登录 PLC 项目，且所选目标系统上有核心转储，则在消息窗口中会显示警告，指明设备上有核心转储。

如果在登录项目时不再显示警告（例如，因为您已完成对该核心转储文件的分析，或者因为您已在其他位置将文件存档），则您可以从目标系统中删除相应的核心转储文件。

2) 手动生成

在在线模式下，如果 PLC 项目当前处于断点或出现异常，您还可以明确生成核心转储。在这种情况下，TwinCAT 只会将核心转储文件存储在项目目录下，而非目标系统中。生成的核心转储文件会直接保存在 PLC 项目目录下（<PLC_project_name>.<PLC_project_GUID>.core）。

加载核心转储

您可以在在线模式下将目标系统中的核心转储加载到项目目录下（使用 命令生成核心转储 [▶ 883]）。此外，您还可以在离线模式下将项目目录下的核心转储加载到项目中（使用 命令加载核心转储 [▶ 884]）。然后，您将会收到 PLC 项目的在线视图，其中包含在发生异常时或生成核心转储文件时的数据和值。

在 TwinCAT 将核心转储加载到项目中时生成的在线视图中，菜单命令显示为可用，但在此状态下无效。如果您选择这样的命令，您将会收到 1 条相应的消息。

此外，您只能使用命令 Close core dump [▶ 885]（关闭核心转储）来关闭核心转储视图。“Logout”（退出）命令在此视图中无效。

存档

如果要在项目存档中包含位于项目目录下的核心转储文件，请在配置文件存档内容期间激活“Core Dump”（核心转储）选项（请参见文档：参考用户界面 > 项目 > PLC 项目设置 > 设置选项卡 [▶ 859]）。

自动创建的存储路径

TwinCAT 3.1 Build 4026 及以上可用

默认值：

默认情况下，自动生成的核心转储会以 *.core 文件的形式存储在目标系统的启动文件夹中。默认情况下，它位于：

- < TC3.1.4026.0: *C:\TwinCAT\3.1\Boot\Plc*
- >=TC3.1.4026.0: *C:\ProgramData\Beckhoff\TwinCAT\3.1\Boot\Plc*

单独调整：

如果需要，您可以调整自动创建核心转储的存储路径。为此，务必在目标系统的 Windows 注册表中输入所需的存储路径。

1. 在目标系统上，打开以下注册表路径：
   - 32 位系统：“*HKEY_LOCAL_MACHINE\SOFTWARE\\Beckhoff\TwinCAT3\Plc*”
   - 64 位系统：“*HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Beckhoff\TwinCAT3\Plc*”
2. 如果“Plc”文件夹尚不存在，请添加该文件夹。
3. 以**字符串**形式添加键“**CoreDumpDir**”，并输入自动创建核心转储所需的文件路径作为值。
4. 在配置中设置 TwinCAT。

⇨ 在下一次 TwinCAT 状态变为“Run”（运行）时，该路径将被激活。

下面介绍 2 个示例工作流程。

从目标系统将核心转储加载到项目中进行分析

要求：

✓ 在开发环境下，您已打开在目标系统上产生异常错误的项目。在开发环境中，PLC 项目处于在线模式。

1. 使用 命令生成核心转储 [▶ 883] 从目标系统加载核心转储。
   ⇒ TwinCAT 会将名称为 <PLC project name>.<PLC project GUID>.core 的核心转储文件复制到本地 PLC 项目目录下。
2. 退出 PLC 项目。
3. 使用 命令加载核心转储 [▶ 884] 将所需的核心转储加载到项目中。
   ⇒ TwinCAT 会显示 PLC 项目的在线视图。您可以看到出错时的变量值和调用堆栈。
4. 在完成核心转储分析之后，选择 命令关闭核心转储 [▶ 885]。

⇨ TwinCAT 会关闭 PLC 项目的核心转储视图。开发环境重新显示正常离线操作的视图。

手动生成已登录 PLC 项目的核心转储

✓ 在开发环境中，PLC 项目处于在线模式。PLC 项目当前处于断点或出现异常错误。

1. 选择 命令生成核心转储 [▶ 883]。

⇨ TwinCAT 会生成新的核心转储，并在本地 PLC 项目目录下创建名称为 <PLC_project_name>.<PLC_project_GUID>.core 的文件。

另请参见：

- 文档 TC3 用户界面：参考用户界面 > PLC > 核心转储 >
  - 命令生成核心转储 [▶ 883]
  - 命令加载核心转储 [▶ 884]
  - 命令关闭核心转储 [▶ 885]

10.4 PLC 操作控制

TC3.1 Build 4026 及以上可用

您有责任确保在安全应用状态下激活运行时系统服务，并且，仅在关键应用状态下停用该服务。

在运行时，设备或项目可能会进入敏感状态，此时的破坏性动作可能会危及整个机器或设备。不过，在这种状态下，您可以抑制某些命令，以防发生危险动作。全局数据类型 PlcAppSystemInfo 可用于此目的。

关于可禁止执行的 TwinCAT 命令的示例：

- 写入值、强制值
- 设置断点
- 下载、在线更改

如果在项目运行时请求运行时系统服务，但该服务当前处于停用状态，则您将会在 TwinCAT 中收到相关消息。

使用 PlcAppSystemInfo 进行操作控制

您可以使用 DWORD 类型的 _AppInfo.Flags 变量激活或停用操作。_AppInfo 是 1 个 PlcAppSystemInfo 类型的实例。

操作	_AppInfo.Flags 的位	示例访问
停用写入值	第 0 位	`_AppInfo.Flags.0 := TRUE;`
停用强制值	第 1 位	`_AppInfo.Flags.1 := TRUE;`
停用设置断点	第 2 位	`_AppInfo.Flags.2 := TRUE;`
停用下载	第 3 位	`_AppInfo.Flags.3 := TRUE;`
停用在线更改	第 4 位	`_AppInfo.Flags.4 := TRUE;`

另请参见：

- 对变量的位访问 [▶ 693]

11 更新 PLC 上的 PLC 项目

情况：您正在向控制器加载 1 个 PLC 项目，该项目与控制器中已存在的项目不同。在这种情况下，会出现 1 个消息窗口，您可以从中选择如何将修改后的 PLC 项目传输到控制器：下载或在线更改。

- 下载会导致对 PLC 程序进行新的编译。除了语法检查之外，还会生成程序代码并将其加载到控制器上。这将会停止正在运行的程序。下载是推荐的数据传输类型，因为在程序停止和重新初始化后，总是会产生 1 个确定的输出状态。
- 在在线更改期间，只有修改后的部分才会被重新加载到控制器上。正在运行的程序不会停止。只有对 PLC 项目进行轻微修改之后，才能使用在线更改选项。在进行更多修改后，则无法可靠地预测程序行为。

修订概述

可能性 1：

上述消息窗口还包含 1 个 **Details**（详细信息）按钮。使用此按钮可打开 **Application information**（应用程序信息）窗口，您可以通过该窗口查看当前 PLC 项目与控制器上的 PLC 项目之间的差异。这包括比较功能块的数量、数据和存储位置。

应用程序信息窗口包含对差异的简要描述，例如：

- MAIN 声明已更改
- 在 MAIN 中已插入变量 fbMyNewInstance
- FB_Sample 的方法/动作的数量已更改

如果已经启用设置 **Download application info**（下载应用程序信息）（PLC 项目属性，编译类别 [▶ 844]），则可选择扩展检查当前 PLC 项目与控制器上的 PLC 项目之间的差异。扩展检查选项的不同之处在于，**Application information**（应用程序信息）窗口包含 1 个附加的 **Online**（在线）比较选项卡，该选项卡会显示树形比较视图。这会告知您哪些 POU 已更改、已删除或已添加。当您执行应用程序信息窗口下部区域中的蓝色下划线命令（“应用程序并非最新。立即生成代码来显示在线比较吗？”）时， 就会显示该附加选项卡。

该树形比较视图只能提供基本概述。有关当前 PLC 项目和控制器上的 PLC 项目的详细比较，请参见选项 2。

可能性 2：

要求：PLC 项目的源代码文件也被传输到目标系统（可通过 PLC 项目设置中的 Settings [▶ 859]（设置）选项卡进行配置）。

TwinCAT 项目比较工具可以实现对当前 TC3 项目和控制器上的 TC3 项目的详细比较（命令将 <TwinCAT project name> 与目标系统进行比较…… [▶ 989]）。这会打开 1 个树形比较视图，可在比较窗口中打开所需的 PLC 编辑器。差异以彩色突出显示。此外，还可以合并目标系统项目的源代码（或部分源代码）。

另请参见：

- 参考用户界面文档：命令：下载 [▶ 887]
- 参考用户界面文档：命令登录 [▶ 889]
- 参考用户界面文档：命令在线更改 [▶ 887]

11.1 执行在线更改

如果您登录的 PLC 项目已经存在于控制器中，但自上次在编程系统中下载后又进行了修改，则 TwinCAT 会自动为您提供在线更改。在此过程中，只有修改后的部分才会被重新加载到控制器上。在线更改期间，控制器上正在运行的程序不会停止。

⚠ 警告

由于设备或系统的意外行为而造成的财产和人身损害

在线更改会修改正在运行的应用程序，并且不会导致重新启动。根据不同的受控设备，可能会损坏设备或工件，或者可能会危及人员的健康和生命。

- 确保新程序代码实现受控系统的所需行为。

项目特定的初始化

在执行在线更改时，由于设备保持其状态，因此不会执行项目特定的初始化（归位等）。为此，新程序代码可能没有所需的效果。

下载代码中的重大更改

如果在线更改导致下载代码发生重大变化（例如，需要切换变量），则会出现 1 个对话框，提供有关效果的信息，并允许您取消在线更改。

快速在线更改

对于微小更改（例如，在实现部分中，不需要切换变量），可执行“快速在线更改”。在这种情况下，仅会编译和重新加载修改后的功能块。特别是，在这种情况下不会生成初始化代码。这也意味着不会生成带“init_on_onlchange”属性的初始化变量代码。通常，这不会产生任何影响，因为该属性一般用于初始化带地址的变量，但变量在快速在线更改时无法更改其地址。

如要确保将 init_on_onlchange 属性应用于整个应用程序代码，可使用 no_fast_online_change 编译器定义停用 PLC 项目的快速在线更改。为此，可在 PLC 项目属性的 Compile [▶ 844]（编译）类别中插入定义。

“init_on_onlchange”属性对单个 FB 变量没有影响

属性“init_on_onlchange” [▶ 746] 仅适用于功能块的全局变量、程序变量和局部静态变量。

如要在在线更改期间重新初始化功能块，务必使用属性来声明功能块实例。对于功能块中的单个变量，不会对属性进行评估。

指针变量

指针变量保留了上一循环的值。如果指针指向已通过在线更改调整大小的变量，则不再正确传递该值。确保在每个循环中重新分配指针变量。

监控函数

在删除隐式监控函数（例如，CheckBound）之后，便无法进行在线更改，只能进行下载。系统会发出相应的消息。

另请参见：

- TC3 用户界面文档：命令在线更改 [▶ 887]

在登录后进行在线更改

✓ 对控制器的连接设置进行正确设置。项目中和控制器上的应用程序完全相同。程序在控制器上运行。PLC 项目已退出。

1. 更改您的 PLC 项目。
2. 在 **PLC** 菜单或 **TwinCAT PLC Toolbar Options**（TwinCAT PLC 工具栏选项）中，选择命令 **Login**（登录）。
   ⇨ 此时会出现 1 个对话框，指明自上次下载后应用程序已更改。
3. 选择选项 **Login with online change**（登录并在线更改），并点击 **OK**（确定）。

⇨ 更改内容已加载到控制器上。在控制器上运行的程序没有停止。PLC 项目已登录。

另请参见：

- TC3 用户界面文档：命令登录 [▶ 889]

登录状态下的在线更改（在线模式）

✓ 对控制器的连接设置进行正确设置。项目中和控制器上的应用程序完全相同。程序在控制器上运行。PLC 项目已登录。

1. 在 PLC 项目树中选择 1 个对象。在这种情况下，最好选择 POU 或 GVL。
2. 在上下文菜单中，选择命令 **Edit Object (Offline)**（编辑对象（离线））。
   ⇨ 在编辑器中打开对象。
3. 例如，通过声明新变量或更改赋值的方式更改对象。
4. 在 **PLC** 菜单中，选择命令 **Online Change**（在线更改）。

⇨ 系统将会询问您是否真的想要执行在线更改。

5. 选择 **Yes**（是），确认对话框。

⇨ 更改内容已加载到控制器上。

另请参见：

- TC3 用户界面文档：命令：编辑对象（脱机）[▶ 822]
- TC3 用户界面文档：命令在线更改 [▶ 887]

什么会阻止在线更改？

在某些 TwinCAT 动作之后，将无法再在控制器上进行在线更改。之后，始终需要下载 [▶ 233]项目。1 个典型的例子是动作 **Clean**（清除）和 **Clean all**（全部清除），它们会删除在上次下载期间存储的编译信息。不过，也有一些“正常”的编辑动作会导致您在下次登录时无法进行在线更改。以下动作可能会阻止在线更改：

检查函数	激活或删除检查函数 [▶ 151]（CheckBounds、CheckRange、CheckDiv 等）。 更改检查函数的接口（包括插入或删除局部变量）。
任务配置	更改配置设置。
项目设置	类别编译 [▶ 844]：更改 **Settings**（设置）部分中的编译器定义（替换常量） 类别：通用 [▶ 841]：最小化 TwinCAT 文件中的 ID 更改。
功能块属性	更改选项 External implementation（外部实现）
功能块	更改功能块的基本块（EXTENDS [▶ 177]FB_Base），或者插入或删除这样的基本块。 更改接口列表（IMPLEMENTS [▶ 183]I_Sample）。例外：在列表末尾添加新接口。
数据类型	将变量的数据类型从 1 种用户定义的数据类型改为另一种用户定义的数据类型（例如，从 TON 改为 TOF）。 将数据类型从用户定义的数据类型改为基本数据类型（例如，从 TON 改为 TIME）。 注意：作为 1 种变通方法，可始终在更改数据类型的同时更改变量的名称。结果，变量被初始化为新变量，旧变量被删除。然后可以进行在线更改。

11.2 执行下载

下载 PLC 项目会导致编译活动项目。除了语法检查之外，还会生成程序代码并将其加载到控制器上。在下载期间，控制器上正在运行的程序会停止。

在下载期间，除了持久变量之外，所有变量都将重新初始化。

另请参见：

- TC3 用户界面文档：命令：下载 [▶ 887]

在登录后进行下载

✓ 对控制器的连接设置进行正确设置。项目中和控制器上的应用程序完全相同。程序在控制器上运行。PLC 项目已退出。

1. 更改您的 PLC 项目。
2. 在 **PLC** 菜单或 **TwinCAT PLC Toolbar Options**（TwinCAT PLC 工具栏选项）中，选择命令 **Login**（登录）。

   ⇨ 此时会出现 1 个对话框，指明自上次下载后应用程序已更改。
3. 选择选项 **Login with download**（登录并下载），并点击 **OK**（确定）。

⇨ 在将更改加载到控制器上时，控制器上正在运行的程序会暂停。PLC 项目已登录。

另请参见：

- TC3 用户界面文档：命令登录 [▶ 889]

登录状态下的下载（在线模式）

✓ 对控制器的连接设置进行正确设置。项目中和控制器上的应用程序完全相同。程序在控制器上运行。PLC 项目已登录。

1. 在 PLC 项目树中选择 1 个对象。在这种情况下，最好选择 POU 或 GVL。
2. 在上下文菜单中，选择命令 **Edit Object (Offline)**（编辑对象（离线））。
   ⇨ 在编辑器中打开对象。
3. 例如，通过声明新变量或更改赋值的方式更改对象。
4. 在 **PLC** 菜单中，选择命令 **Download**（下载）。
   ⇨ 系统将会询问您是否真的想要执行 **Download**（下载）操作。
5. 选择 **Yes**（是），确认对话框。

⇨ 在将更改加载到控制器上时，控制器上正在运行的程序会暂停。

另请参见：

- TC3 用户界面文档：命令：编辑对象 (脱机) [▶ 822]
- TC3 用户界面文档：命令：下载 [▶ 887]

12 使用独立的 PLC 项目

独立的 PLC 项目以单独的项目类型管理传统的 PLC 项目（嵌入在 TwinCAT 项目中的 PLC 项目）。编程与嵌入式 PLC 项目相同。与此相反，在项目中不能直接使用 PLC 实例，只能在集成后在 TwinCAT 项目中对其进行配置。在独立的 PLC 项目中也可使用 TwinCAT 3 类型系统，以管理基于项目的数据类型或配置 TwinCAT 3 事件记录器。

因此，独立的 PLC 项目可以在项目层面上将系统、运动和 I/O 配置与 PLC 开发分离。这样就产生了 2 个独立的项目：独立的 PLC 项目和 TwinCAT 项目。

这 2 个项目可以在同一个解决方案中进行管理，也可以在不同的解决方案中进行管理。为了将 TwinCAT 项目中的 PLC 模块实例与系统、运动和 I/O 层面的相关组件连接起来，需要集成在独立的 PLC 项目中编译的 TMC 文件。

例如，该程序可以实现系统配置和 PLC 编程的严格分离，将 PLC 实例集成到不同的系统配置中，或在特定条件下从 PLC 项目中无一例外地下载 PLC 项目运行时数据。

要求

TwinCAT 版本 3.1.4022.0 及以上可以使用独立的 PLC 项目。

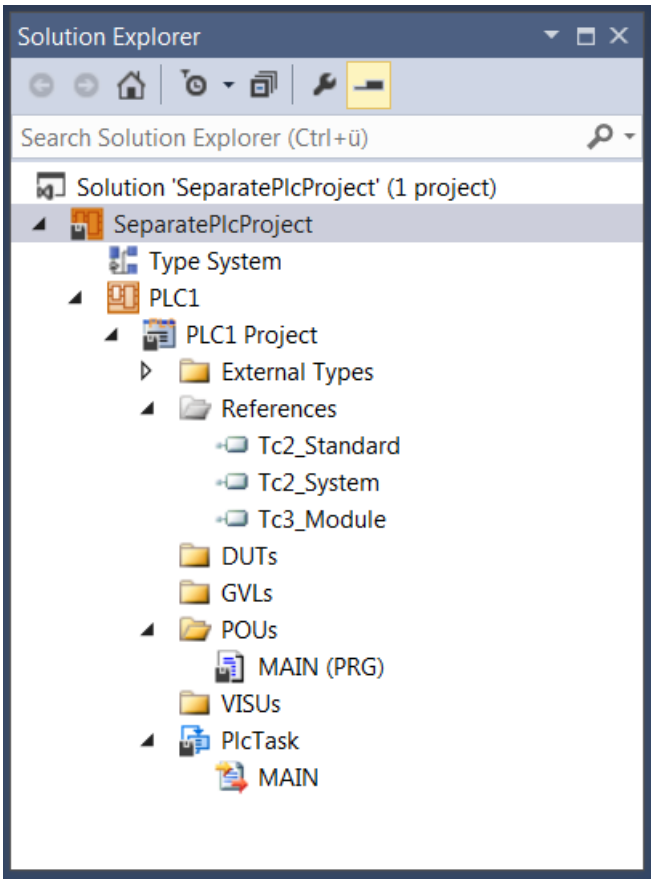
另请参见：

- 创建标准项目 [▶ 48]
- 类型系统

12.1 创建独立的 PLC 项目

1. 在菜单 **File**（文件）中，选择命令 **New** > **Project**（新建 > 项目）。
   ⇨ 对话框 **New Project**（新建项目）会打开。
2. 选择 TwinCAT PLC 模板 **TwinCAT PLC project**（TwinCAT PLC 项目）。
3. 为 PLC 项目命名（例如，“SeparatePlcProject”），并选择存储位置和解决方案。
4. 选择 **OK**（确定），确认对话框。
   ⇨ 在 **Solution Explorer**（解决方案资源管理器）中会打开 1 个新的项目文件夹。在主窗口的标题栏中会显示项目名称“SeparatePlcProject”。
5. 在项目树中标记 PLC 对象，并在菜单 **Project**（项目）或上下文菜单中选择命令 **Add New Item...**（添加新项目……）。
6. 在 **Plc Templates**（Plc 模板）类别中，选择 **Standard PLC Project**（标准 PLC 项目）并输入 1 个名称（例如，“PLC1”）。
7. 选择 **Add**（添加），退出对话框。

⇨ 在视图 **Solution Explorer**（解决方案资源管理器）中已创建以下结构：



⇨ 在 PLC 项目对象（PLC1）下方，会自动显示标准 PLC 项目的基本对象和类型系统。与嵌入到 TwinCAT 项目中的标准 PLC 项目相比，没有创建 PLC 实例。

另请参见：

- 创建标准项目 [▶ 48]

12.2 将独立的 PLC 项目集成到 TwinCAT 项目中

通过 TMC 文件将独立的 PLC 项目集成到 TwinCAT 项目中。在创建独立的 PLC 项目并将其添加到 TwinCAT 项目中时就会产生这种情况。

12.2.1 创建 TMC 文件并将其添加到 TwinCAT 项目中

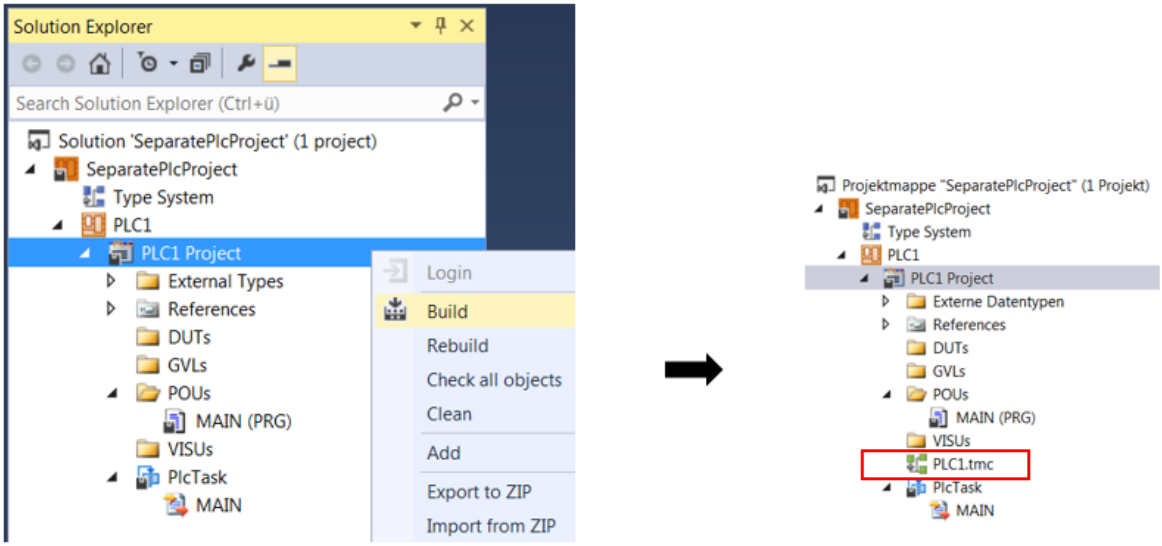
下文将介绍如何创建 TMC 文件并将其添加到 TwinCAT 项目中。不允许在相同的 TwinCAT 项目中多次添加 TMC 文件。

✓ 已创建 1 个独立的 PLC 项目（例如，“SeparatePlcProject”），其中包含 1 个 PLC 项目（例如，“PLC1”）。

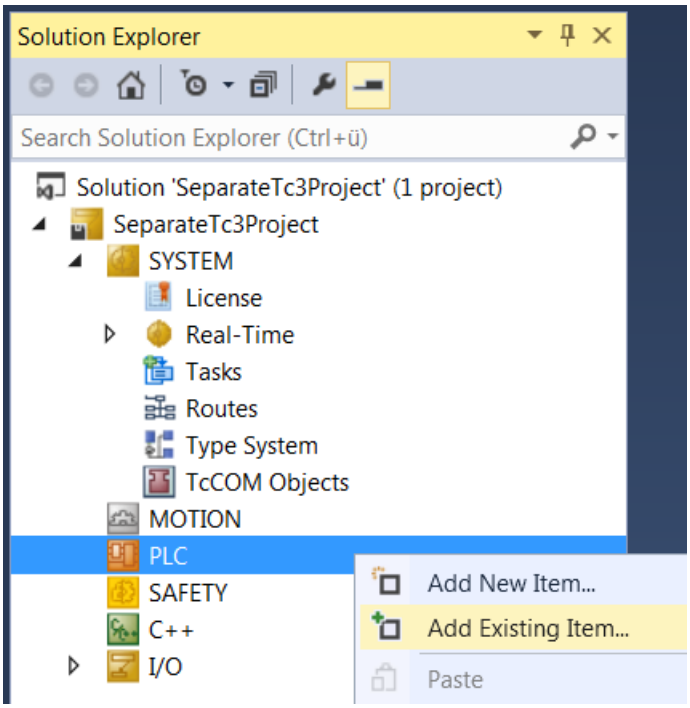
✓ 已创建 1 个 TwinCAT 项目，独立的 PLC 项目将集成到该项目中（例如，“SeparateTc3Project”）。

1. 打开独立的 PLC 项目，并在 PLC 项目树中选择 PLC 项目对象（“PLC1 项目”）。
2. 从上下文菜单或 **Build**（构建）菜单中，选择命令 **Build**（构建）（**Build <PLC Project Name>**）（构建 <PLC Project Name>）。

⇨ 对 PLC 项目进行编译并检查是否存在错误。如果编译成功，则会创建 TMC 文件。

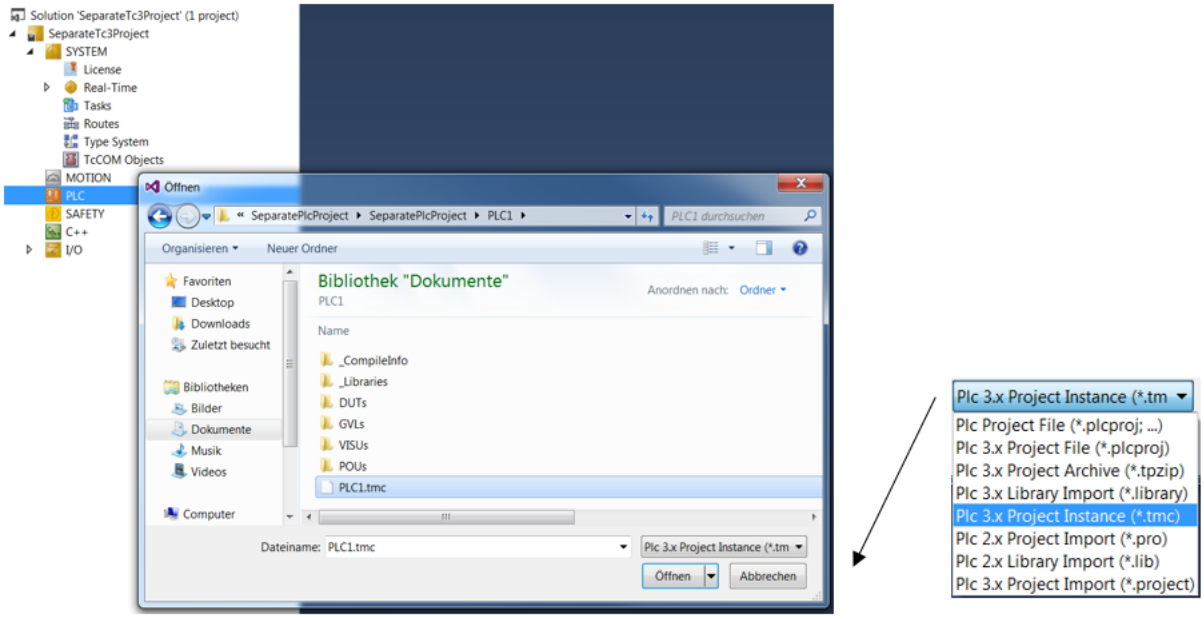


3. 打开 TwinCAT 项目，并在 TwinCAT 项目树中选择 PLC 对象。
4. 从上下文菜单或 **Project**（项目）菜单中，选择 **Add Existing Item**（添加现有项目）。

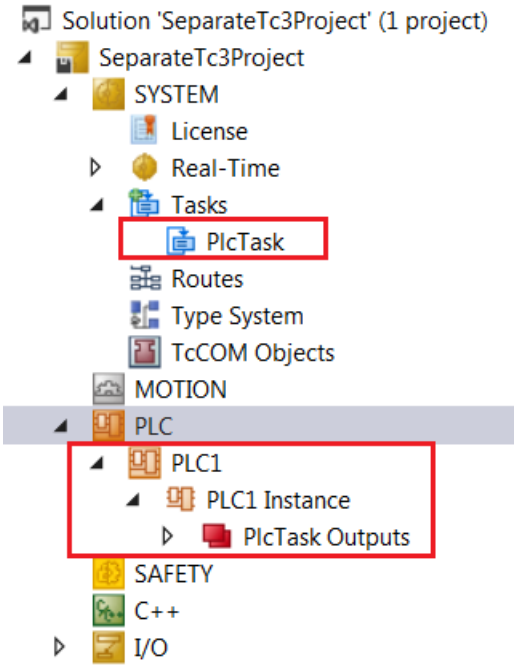


⇨ 用于打开文件的标准对话框会出现。

5. 选择 TMC 文件并点击 **Open**（打开），将其添加到项目中。

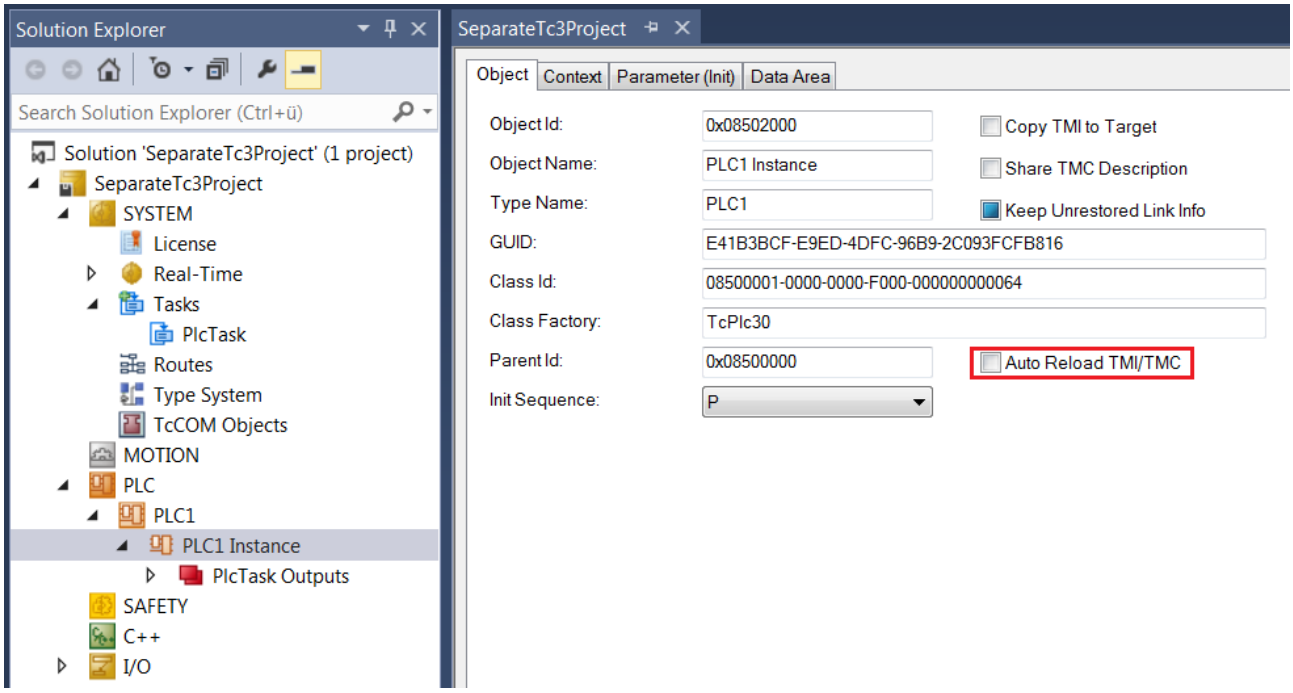


⇨ 独立的 PLC 项目的 PLC 实例和引用的任务已被添加到 TwinCAT 项目中。如果 TwinCAT 项目中已经存在与 PLC 项目的引用任务同名的系统任务，则不会添加新的任务。相反，现有的系统任务会被链接到已添加的 PLC 实例。



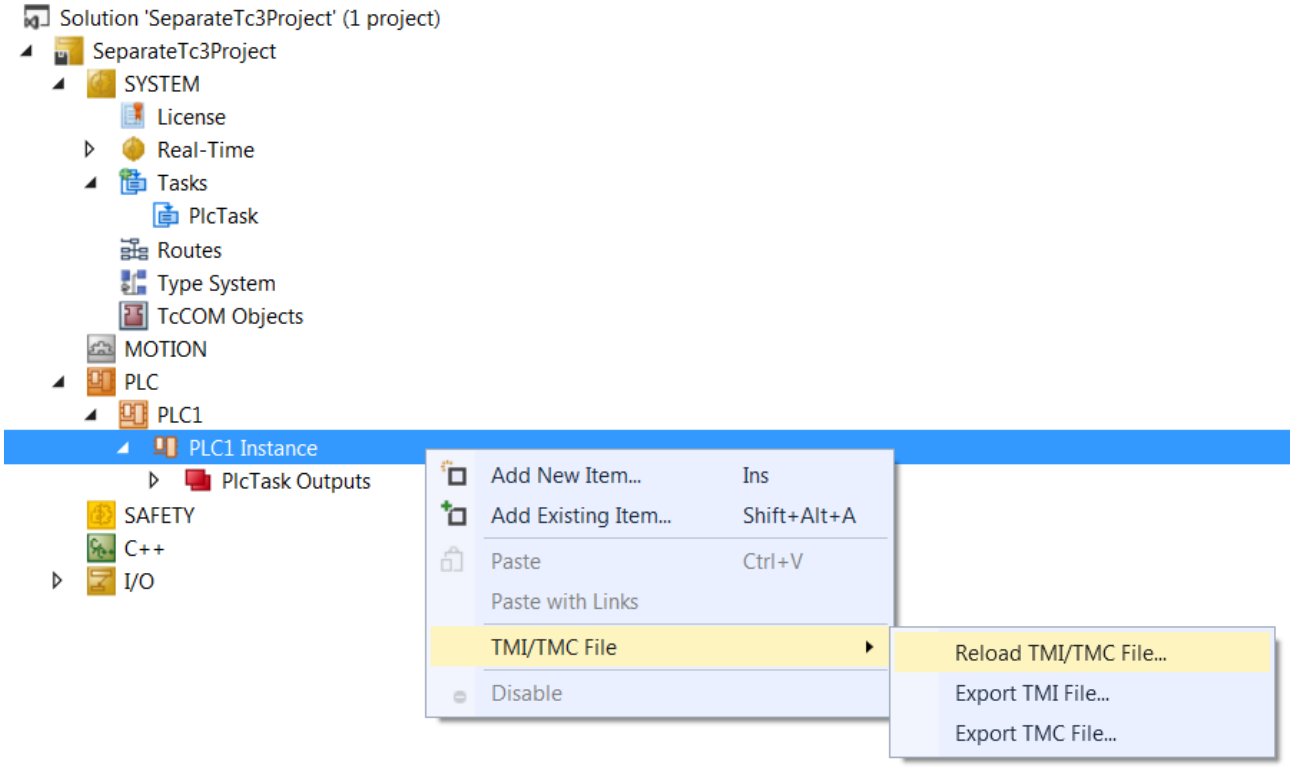
自动重新加载 TMC 文件

在 PLC 实例的设置中，您可以指定在更改独立的 PLC 项目时应该自动重新加载 TMC 文件。双击项目树中独立的 PLC 项目的 PLC 实例，在编辑器中打开 PLC 实例设置。在 **Object**（对象）选项卡上，勾选 **Auto Reload TMI/TMC**（自动重新加载 TMI/TMC）复选框。



手动重新加载 TMC 文件

如要在独立的 PLC 项目发生变化时手动重新加载 TwinCAT 项目中的 TMC 文件，请在项目树中选择独立的 PLC 项目的 PLC 实例，并从上下文菜单中选择 **TMI/TMC File > Reload TM/TMC File**（TMI/TMC 文件 > 重新加载 TM/TMC 文件）。

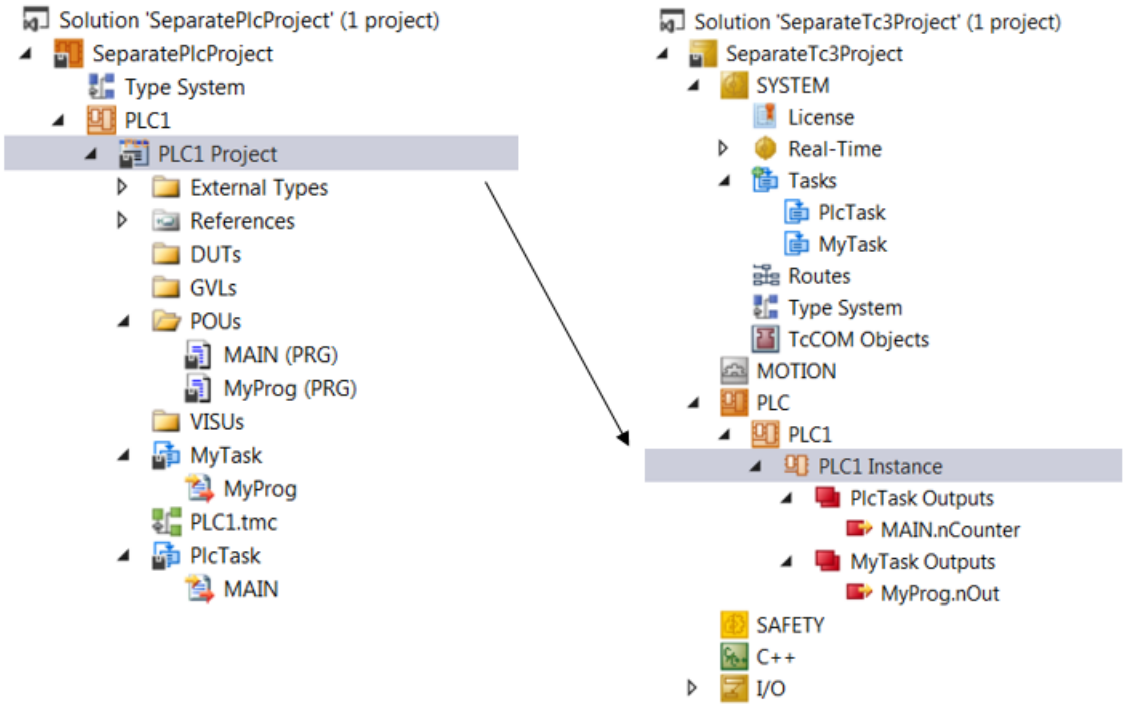


12.2.2 分配任务

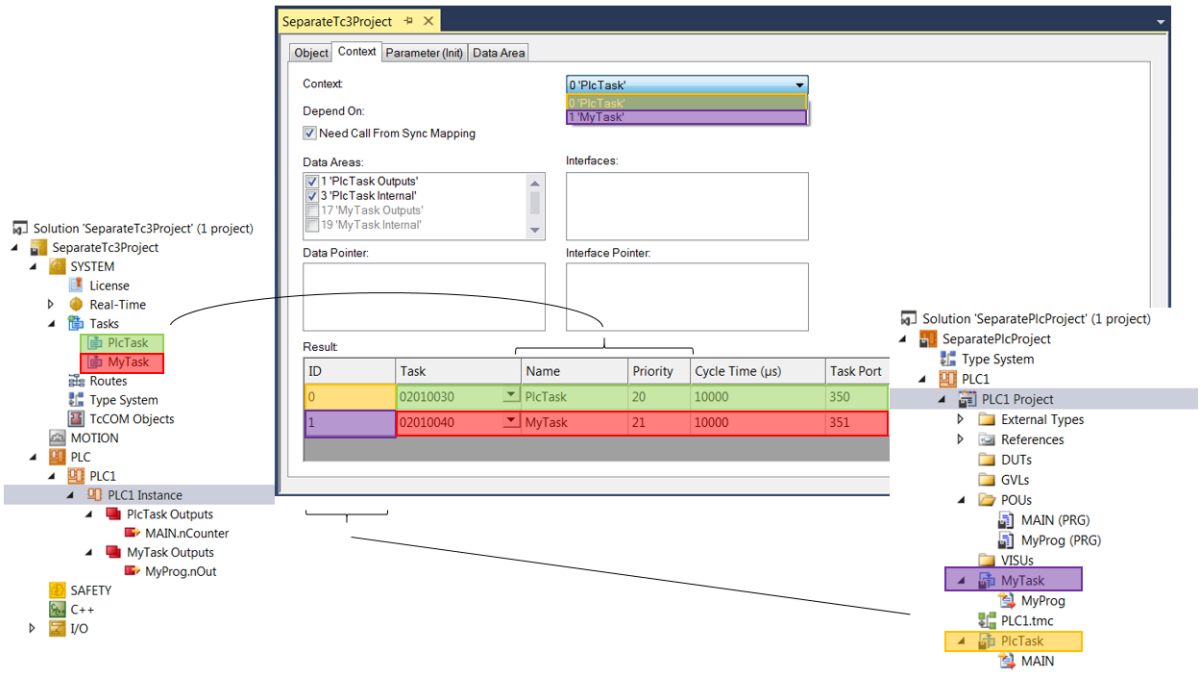
下文将介绍如何将为独立的 PLC 项目创建的任务引用分配给系统任务。

✓ 为独立的 PLC 项目生成各种任务引用（例如，“PlcTask”和“MyTask”），这些任务引用定义了 MAIN 和 MyProg 程序块的处理过程。

✓ 独立的 PLC 项目的 PLC 实例被集成到 TwinCAT 项目中。



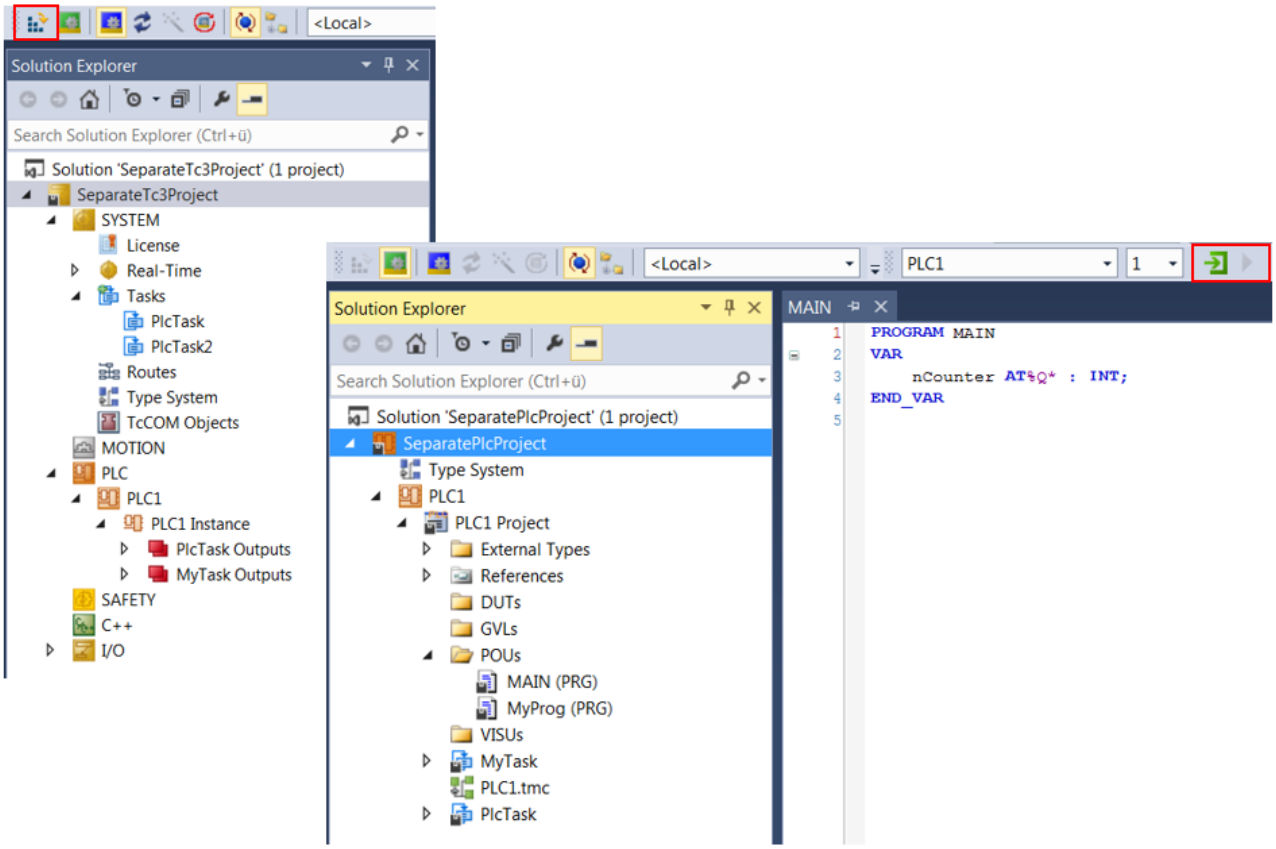
1. 双击 TwinCAT 项目树中的集成 PLC 实例，在编辑器中打开 PLC 实例设置。
2. 选择 **Context**（上下文）选项卡。
3. 将任务引用分配给现有的系统任务。（在本示例中，任务引用被分配给同名的系统任务。）



12.2.3 加载程序代码、登录和启动 PLC

✓ 将独立的 PLC 项目集成到 TwinCAT 项目中。

1. 激活 TwinCAT 项目配置。为此，可从工具栏中选择命令 **Activate Configuration**（激活配置）。
2. 在独立的 PLC 项目中登录 PLC 并启动 PLC。



12.3 最佳实践

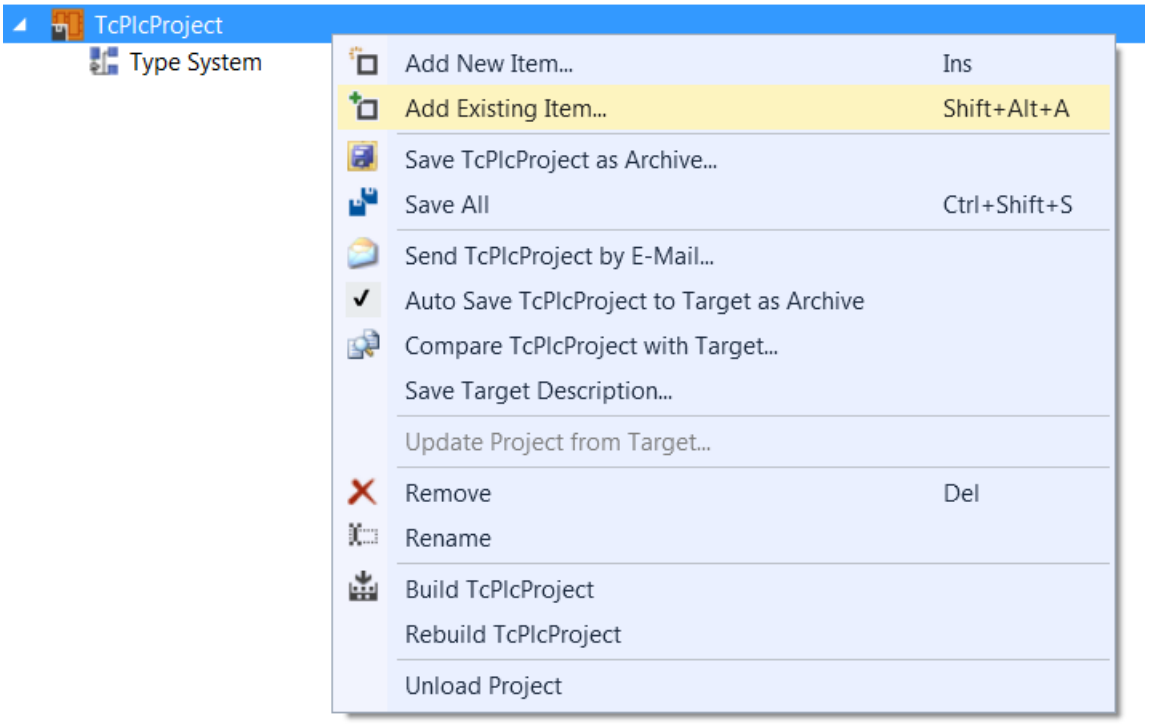
概述

- 将现有 TwinCAT 项目的嵌入式 PLC 项目转换为独立的 PLC 项目 [▶ 241]
- 将独立的 PLC 项目转换为嵌入式 PLC 项目 [▶ 244]

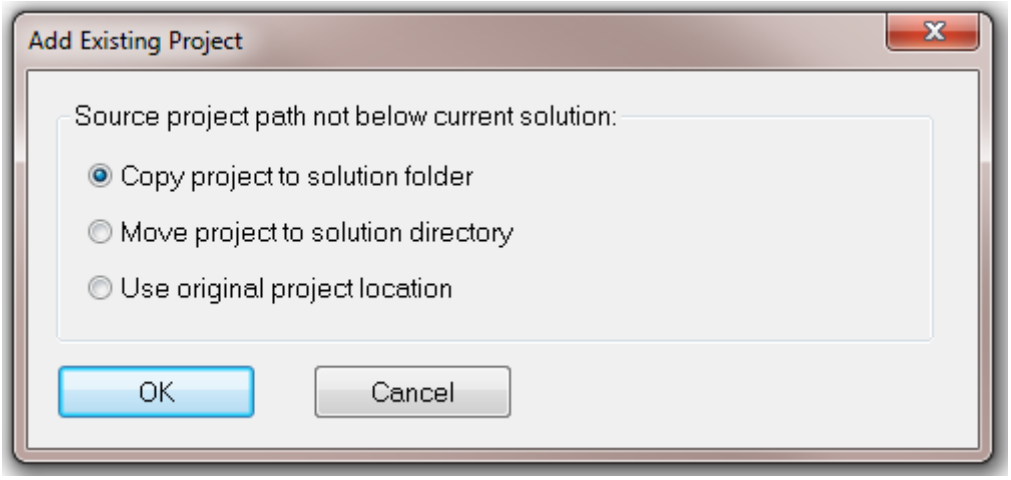
将现有 TwinCAT 项目的嵌入式 PLC 项目转换为独立的 PLC 项目

1. 创建 1 个“独立的 PLC 项目”类型的项目。请勿创建 PLC 项目。

2. 将现有 TwinCAT 项目的嵌入式 PLC 项目添加到独立的 PLC 项目中。为此，可点击 PLC 项目的上下文菜单中的 **Add Existing Item…**（添加现有项目……），并在打开的标准对话框中选择嵌入式 PLC 项目的 .plcproj 文件。确认对话框。



⇨ **Add Existing Project**（添加现有项目）对话框会打开。

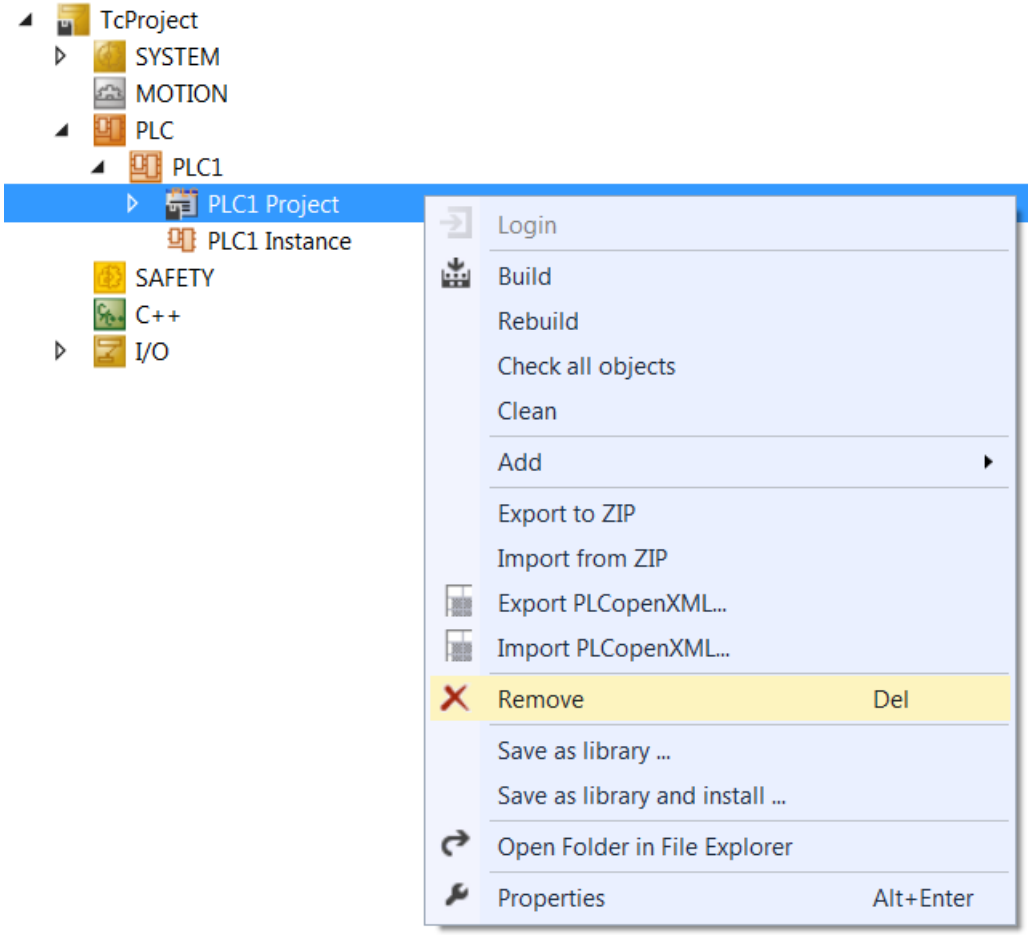


3. 选择选项 **Copy project to solution folder**（将项目复制到解决方案文件夹），可独立于原始 PLC 项目管理 PLC 项目。

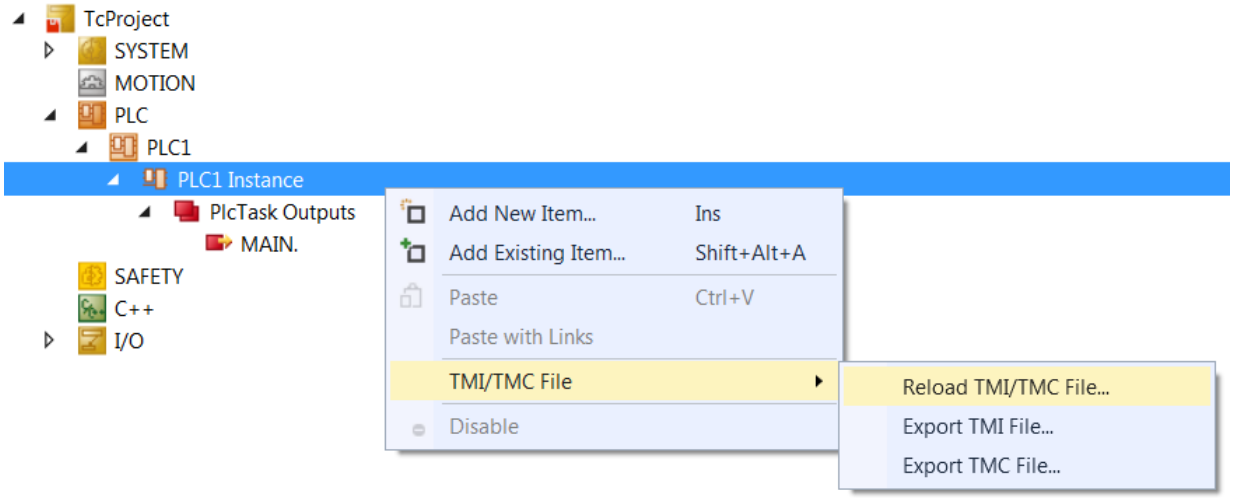
   ⇨ PLC 项目已被添加到独立的 PLC 项目中。

4. 创建 PLC 项目，以创建 TMC 文件。

5. 删除现有 TwinCAT 项目中的嵌入式 PLC 项目，方法为仅删除实际 PLC 项目而不删除相关的实例，这样可以保留现有映射。



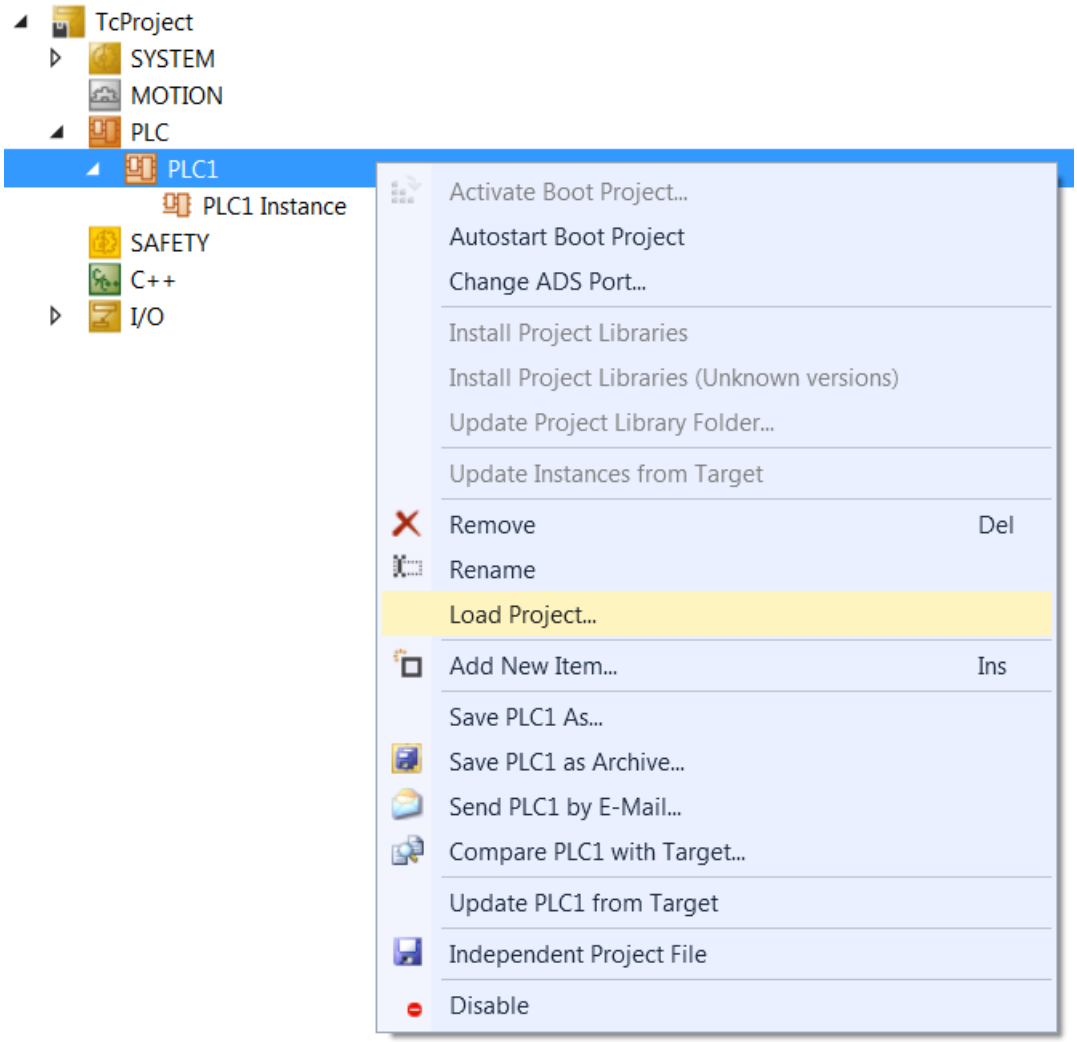
6. 更改 TMC 文件的路径。为此，可点击 PLC 项目实例的上下文菜单中的 **TMI/TMC File > Reload TMI/TMC File…**（TMI/TMC 文件 > 重新加载 TMI/TMC 文件…），并在打开的标准对话框中选择独立的 PLC 项目的 TMC 文件。确认对话框。



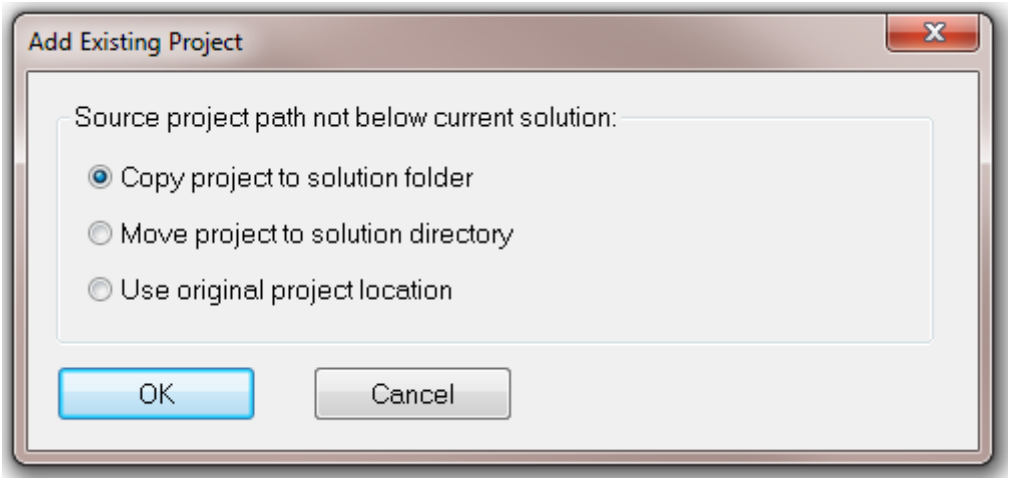
⇨ 现在，先前嵌入 TwinCAT 项目中的 PLC 项目形成了 1 个独立 PLC 项目（独立的 PLC 项目）。

将独立的 PLC 项目转换为嵌入式 PLC 项目

1. 将独立的 PLC 项目的 PLC 项目添加到现有的 PLC 项目中。为此，可点击 PLC 项目的上下文菜单中的 **Load Project…**（加载项目……），并在打开的标准对话框中选择独立的 PLC 项目的 .plcproj 文件。确认对话框。



⇨ **Add Existing Project**（添加现有项目）对话框会打开。



- 如果您想要独立于原始的独立 PLC 项目管理 PLC 项目，并继续使用独立的 PLC 项目，可**将项目复制到解决方案文件夹**。
- 如果您不想再使用独立的 PLC 项目，可**将项目移动到解决方案目录**。
- 如果您想要在同一数据上处理 TwinCAT 项目和独立的 PLC 项目，可**使用原始项目位置**。

2. 选择所需的选项并确认对话框。

⇨ 现在，独立的 PLC 项目已嵌入 TwinCAT 项目中。

12.4 常见问题

概述

为什么无法在 TwinCAT 项目中激活配置？

错误消息： “无法复制目标文件 – 未找到源文件‘...’”

原因： 此错误可能有不同的原因：

1. 独立的 PLC 项目的 ADS 端口和 TwinCAT 项目中的 PLC 实例的 ADS 端口不相同。
   解决方案： 在 2 个级别设置相同的 ADS 端口，对独立的 PLC 项目进行重新编译。如果在 PLC 实例配置器中未激活 **Auto Reload TMI/TMC**（自动重新加载 TMI/TMC）选项，可重新加载 TMC 文件。
2. TwinCAT 项目和独立的 PLC 项目并非针对同一个平台编译。
   解决方案： 针对同一个平台编译这 2 个项目。如果在 PLC 实例配置器中未激活 **Auto Reload TMI/TMC**（自动重新加载 TMI/TMC）选项，可重新加载 TMC 文件。
3. 仅为 TwinCAT 项目提供了 TMC 文件，因此 TMC 文件的路径不会指向独立的 PLC 项目，并且无法访问 PLC 运行时数据。
   解决方案： 在 TwinCAT 项目 PLC 设置的 **Settings**（设置）选项卡中，禁用 **Activate PLC configuration (copy boot files)**（激活 PLC 配置（复制启动文件））选项。
   这样可以无一例外地将 PLC 运行时数据从独立的 PLC 项目传输到目标系统。

为什么无法在独立的 PLC 项目中登录 PLC？(a)

错误消息： “PLC 实例参数（0x0850801a）不匹配。下载将会中止。”

原因： 如果出现此错误，则表示在登录之前尚未更新 PLC 模块实例信息。

解决方案： 选择 **Yes**（是）确认错误消息对话框，并忽略随后出现的错误消息，或者使用 **Update Instances from Target**（从目标更新实例）进行手动更新。

为什么无法在独立的 PLC 项目中登录 PLC？(b)

错误消息： “PLC 实例参数（0x08500005）不匹配。下载将会中止。”

原因： 如果出现此错误，则表示 PLC 项目与已激活的 TwinCAT 项目的 PLC 实例不一致。这意味着在独立的 PLC 项目中进行更改之后，在 TwinCAT 项目中没有更新 TMC 文件和 PLC 实例。

解决方案： 选择 **Yes**（是）或 **No**（否）确认错误消息对话框，并忽略以下错误消息。然后，更新 TwinCAT 项目中的 TMC 文件（**上下文菜单 PLC project instance > TMI/TMC File > Reload TMI/TMC File…**）（PLC 项目实例 > TMI/TMC 文件 > 重新加载 TMI/TMC 文件……）。如果 TMC 文件不是单独提供的，而是通过独立的 PLC 项目路径直接添加的，则应在 PLC 实例配置器中启用 **Auto Reload TMI/TMC**（自动重新加载 TMI/TMC）选项。

13 使用库

库是可重复使用对象的集合，例如：

- 诸如功能块或函数等 POU
- 接口及其方法和属性
- 诸如枚举、结构、别名、联合等数据类型
- 全局变量、常量、参数列表
- 文本列表、图像池、可视化、可视化元素
- 外部文件（例如，文档）

在项目中库集成可实现在项目中以与在项目中直接定义的其他功能块和变量相同的方式使用库模块。

建议和说明

除了对库使用的描述之外，请参见建议和说明 [▶ 248]中的应用说明。

以下步骤与库的应用相关。创建、安装和管理库。

如果已经创建并安装库（例如，系统库就是这种情况），则只需进行库管理或集成步骤。有些库必须事先首先进行创建和安装。

库创建

- 在创建库时，有 2 种库类型可供选择：*.library（源库）和 *.compiled-library（带源代码保护的编译库）。
- 有关前提条件和更多信息，请参见库创建 [▶ 249]部分。

库安装

- 在项目中使用库之前，必须首先在系统中安装库。在本地系统中不同的“存储库”（目录、存储位置）中，可对库进行管理。在项目中集成库之前，必须将其安装到本地系统中具有定义版本号的存储库中。
  - **例外**：作为库引用的项目。
    请参见 使用 PLC 项目作为引用库。
- 如果在存储库中没有安装所使用的库版本，则会在项目树中的引用处进行标记。
- 库安装在 库存储库 [▶ 251] 中。

库管理

- 库管理器 [▶ 254]很好地概述了在项目中使用的 PLC 库引用，可用于在项目中集成库或占位符。集成后，可在项目中使用库引用元素。
- 在库管理器中还会显示在另一个库中作为子库而被引用的库引用。此外，还有“隐藏库”（请参见 命令属性 [▶ 333] 部分）。
- 每当使用 1 个库时，都会引用 1 个唯一的库版本。该版本被指定为有效版本。如果库附带 1 个固定版本（例如，3.3.0.0），则项目将始终使用该库版本，即使有更新的版本可用也是如此。或者，也可以使用设置“Always newest”（始终最新）/“*”来自动确保始终使用最新版本的库。在这种情况下，TwinCAT 总是会使用在库存储库中找到的最新版本的库。有关更多信息和示例，请参见 命令设置为始终最新版本 [▶ 334]。
- 无法将同一个库的相同版本多次添加到库管理器中。在库管理器中，1 个版本的库可以作为库或占位符而被引用。
- 如果库没有经过编译（*.compiled-library），而是以*.library 文件的形式存在，则可通过双击相应条目的方式打开在库管理器中列出的库元素。
- 您可以将库引用以库或占位符的形式添加到库管理器中，并将其包含在应用程序中（请参见 命令添加库 [▶ 329] 部分）。只要可能，您应该使用占位符。有关更多信息，请参见 库占位符 [▶ 257] 部分。
- 在项目中处理库模块时，将按照在库存储库 [▶ 251]中列出的顺序来搜索库和存储库。有关更多信息，请参见对库模块或变量的明确访问 [▶ 247]部分。

库文档

TwinCAT 提供多种库文档选项。有关更多信息，请参见 库文档 [▶ 260] 部分。

有关这些主题的信息，请参见下文：

- 引用库
- 库版本
- 对库模块和变量的明确访问
- TwinCAT 2.x PLC Control 库
- 外部和内部库或库模块，后期绑定

引用库

- 1 个库可以集成其他库（引用库），因此嵌套的深度可以根据需要而定。如果这样 1 个“父级”库本身在 1 个项目中集成，则在该项目中也可以使用其中所引用的库。
- 库引用应始终通过库占位符 [▶ 257]进行定义，以避免因版本相关性或需要使用制造商特定的库而可能产生的问题。
- 在每个引用库的属性 [▶ 333]中，您可以指定当该库通过“父级”库在项目中集成时应该如何表现，例如，它是否应该在库管理器中“隐藏”。

库版本

- 在系统上可同时安装多个版本的库。
- 在项目中可同时集成多个版本的库。然而，这种做法并不可取。在这种情况下，必须为每个库分配 1 个唯一的命名空间，并且必须限定对符号的访问。示例：V1.Send，V2.Send
- 在属性窗口 [▶ 333]中可以配置库或占位符的版本或解析。占位符的解析也可以在占位符对话框中进行调整。
- 如果在 1 个库中引用了其他库，同时为了使项目兼容，强烈建议使用占位符。这样可以避免因版本相关性或需要使用制造商特定的库而产生的问题。

对库模块或变量的明确访问

- 如果在项目和库中有多个同名的模块或变量，则应明确对模块组件的访问。通过在模块名称前添加库的命名空间，可实现与库有关的唯一性。
- 库的**命名空间**的默认设置为库标题。另一方面，也可以为库明确定义不同的命名空间：通常在项目属性 [▶ 841]中创建库 [▶ 249]时针对库定义命名空间，或者在库引用的属性窗口 [▶ 333]中针对项目中的库的本地使用情况定义命名空间。
  库的命名空间必须用作标识符的前缀，这样才能对项目中多次存在的模块进行唯一访问。
- 示例：
  - 库 Lib1 已集成到 1 个应用程序项目中。
  - 在库 Lib1 和应用程序中均已声明函数 F_Sample。
  - 为了实现对这 2 个函数的访问，在调用库函数之前会添加库的命名空间。这样可以明确引用库 Lib1 的函数。
  - 调用应用程序函数 `nResult := F_Sample(nInput := nVar);`
  - 调用库函数 `nResult := Lib1.F_Sample(nInput := nVar);`

TwinCAT 2.x PLC Control 库

- 使用 TwinCAT 2.x PLC Control（*.lib）创建的库会继续得到支持。
- 在 TwinCAT 3 PLC 中可以使用“旧”库项目（*.lib），并将其转换为“新库”（*.library/*.compiled-library）。

- 如果打开 1个引用旧库的现有项目，则用户可以选择是保留这些引用、用其他引用替换这些引用，还是删除这些引用。如果要保留，应将相关的库转换为新格式并在“系统”库存储库中自动安装它们。如果它们不包含所需的项目信息，则可直接添加。在项目选项中可以存储在转换旧项目期间处理特定旧库所依据的程序。如果在转换另一个旧项目的过程中再次出现相同的库，则无需再次定义程序，而应自动执行该程序。
- Add New Item... （添加新项目……） [▶ 832]部分介绍了打开和转换项目和库的程序。

外部和内部库模块，后期绑定

- 外部库模块是固件函数，其实现不包含在 PLC 库中。对于具有固件函数的库，固件必须在目标系统上可用；只有在应用程序运行时才会绑定固件。

13.1 建议和说明

有关库使用的一些建议和说明如下所述。

关于库占位符的建议

通过使用库占位符，对所使用的库版本的调整变得几乎毫不费力，从而使工程项目和库的流程变得非常灵活。因此，我们建议使用占位符，而不是库。

有关更多信息，请参见 库占位符 [▶ 257]。

通过使用“始终最新”版本，可能进行在线更改

示例：您激活并启动 1 个项目，在该项目中引用的库“LibA”是“始终最新”版本。在下载时，在该库存储库中此库的最新版本为版本 1.0.0.0。如果您随后在库存储库中安装更新版本的“LibA”（例如，版本 1.0.1.0），则该版本就会成为库的最新版本。如果您使用应用程序项目登录，而且您没有对其进行主动更改，则 TwinCAT 会检测到应用程序中的更改，因为用作“始终最新”的“LibA”不再以版本 1.0.0.0 被引用，而是以现在的版本 1.0.1.0 被引用。这意味着您在登录时将会获得在线更改或下载的服务，不过，您可能不会察觉到任何项目更改。

由于自动更改库的有效版本（由“始终最新版本”设置引起）而产生的这种烦扰是不必要的，应予以避免。此外，在调试后或项目完成后应对项目源和所使用的组件进行冻结和备份，以便跟踪。

因此，我们强烈建议在项目完成后（即交付前）将所有库引用（即在项目层面上直接使用的库引用和在库内部使用的库引用）设置为固定有效版本。

请注意，该建议不仅适用于用户库，也适用于倍福库，因为在安装新的 XAE 或 RM 设置时，可能会安装新版本的引用倍福库。

不过，在开发阶段，使用“始终最新版本”选项是合理的，因为这样可以始终自动使用最新可用的库版本。

有关如何将库版本设置为固定有效版本的更多信息，请参见 命令设置为有效版本 [▶ 334] 部分。

有关更多信息和关于“始终最新”版本行为的另一个示例，请参见 命令设置为始终最新版本 [▶ 334]。

传输用户库的源代码

如果您已将目标或文件/电子邮件存档设置配置为在其中 1 个存档中包含源库，请注意在传递/交付目标系统时或在传递文件/电子邮件存档时，在项目中使用的源库（*.library）会以可读源代码的形式包含在 ZIP 存档中。在引用库和传递目标系统或存档时，请牢记这一点。

如果您不想在目标或文件/电子邮件存档中共享（用户）库的可读源代码，则您可以将这些库作为编译库（*.compiled-library）进行保存和引用，或者您可以在项目设置中禁用向目标或文件/电子邮件存档中添加源库。

有关目标和文件/电子邮件存档的配置选项的更多信息，请参见 PLC 项目设置 > 设置选项卡 [▶ 859]部分。

有关源代码加密的信息，请参见安全管理文档。

13.2 库创建

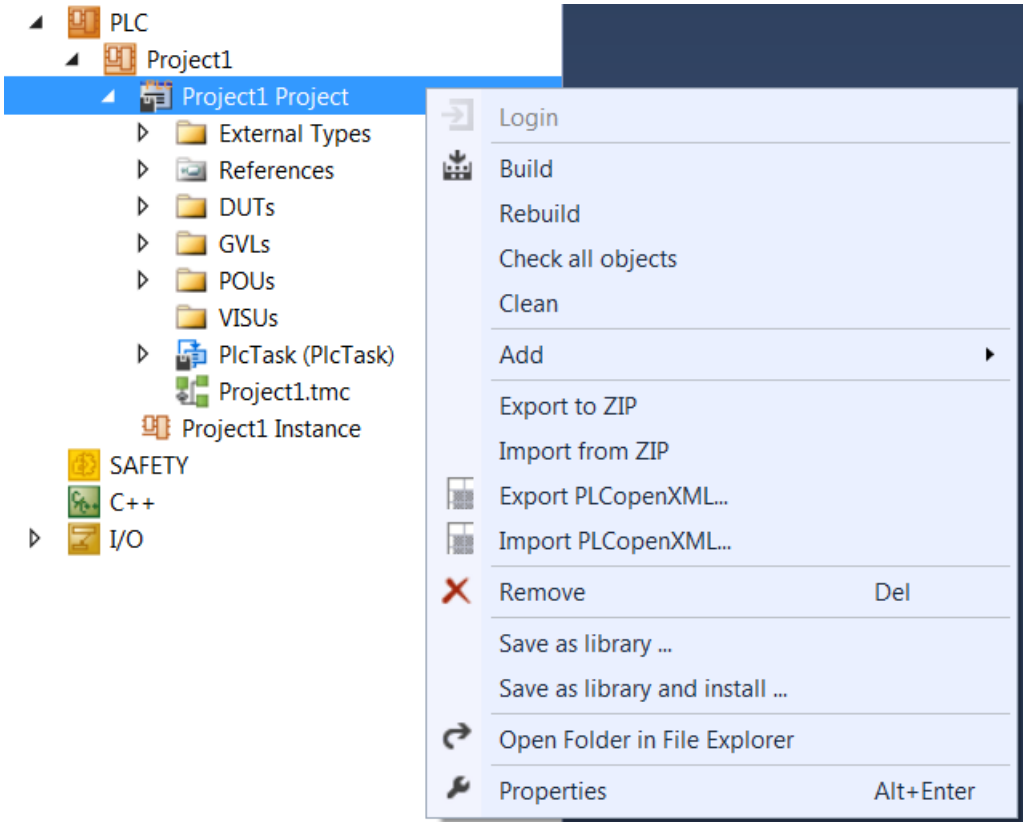
- TwinCAT 3 PLC 项目可以作为库（<Project name>.library/.compiled-library）存储。同时，如果需要，还可以在 库存储库 [▶ 251] 上安装它。如要创建特定的库项目，建议直接在 **New Project**（新建项目）对话框中选择模板 **Empty PLC project**（空 PLC 项目）。
- 如要将项目另存为库，则必须在项目属性 [▶ 841]中输入**标题**、**版本号**和**公司名称**。
- 如果库集成了其他库，则用户应该仔细考虑当“父级”库在项目中集成时，这些引用库应该如何表现。这些考量因素适用于版本处理、命名空间、可见性和访问选项，其中一些可在单个引用库的“属性”对话框中进行配置。如果在项目中集成库时总是引用另一个库，则在定义引用时可以使用占位符。
- 如要使库模块的视图和访问受保护，可将库项目以预编译格式保存（<project name>.compiled-library） - 请参见 命令另存为库 [▶ 249]。
- 库的数据结构可被标记为库内部。这些非公开对象的访问标签为 INTERNAL 或属性为“隐藏” [▶ 744]，因此不会出现在库管理器 [▶ 254]、“列出组件”功能或输入向导中。
- 在功能块参数的声明中可以添加注释，作为 1 种向库用户提供库功能块信息的便捷方式。随后，当库在项目中集成时，在库管理器 [▶ 254]的 **Documentation**（文档）选项卡上就会显示该注释。另请注意 库文档 [▶ 260] 选项。
- 在 PLC 项目的上下文菜单中可使用以下命令来保存库：
  - 命令另存为库 [▶ 249]
  - 命令另存为库并安装 [▶ 250]

13.2.1 命令另存为库

功能：该命令可打开用于将 PLC 项目另存为 PLC 库的标准对话框。

调用：解决方案资源管理器中的 PLC 项目对象（<PLC project name> 项目）的上下文菜单

可将 PLC 项目另存为 PLC 库，以便将源代码作为库并通过定义的接口提供给其他应用程序。在 PLC 项目的上下文菜单中，可使用保存库的命令。



该命令会打开用于在文件系统中保存文件的标准对话框。现有项目名称已自动提供；用户可以根据需要对其进行修改。在将项目另存为库时，可在 2 种库文件格式中做出选择：

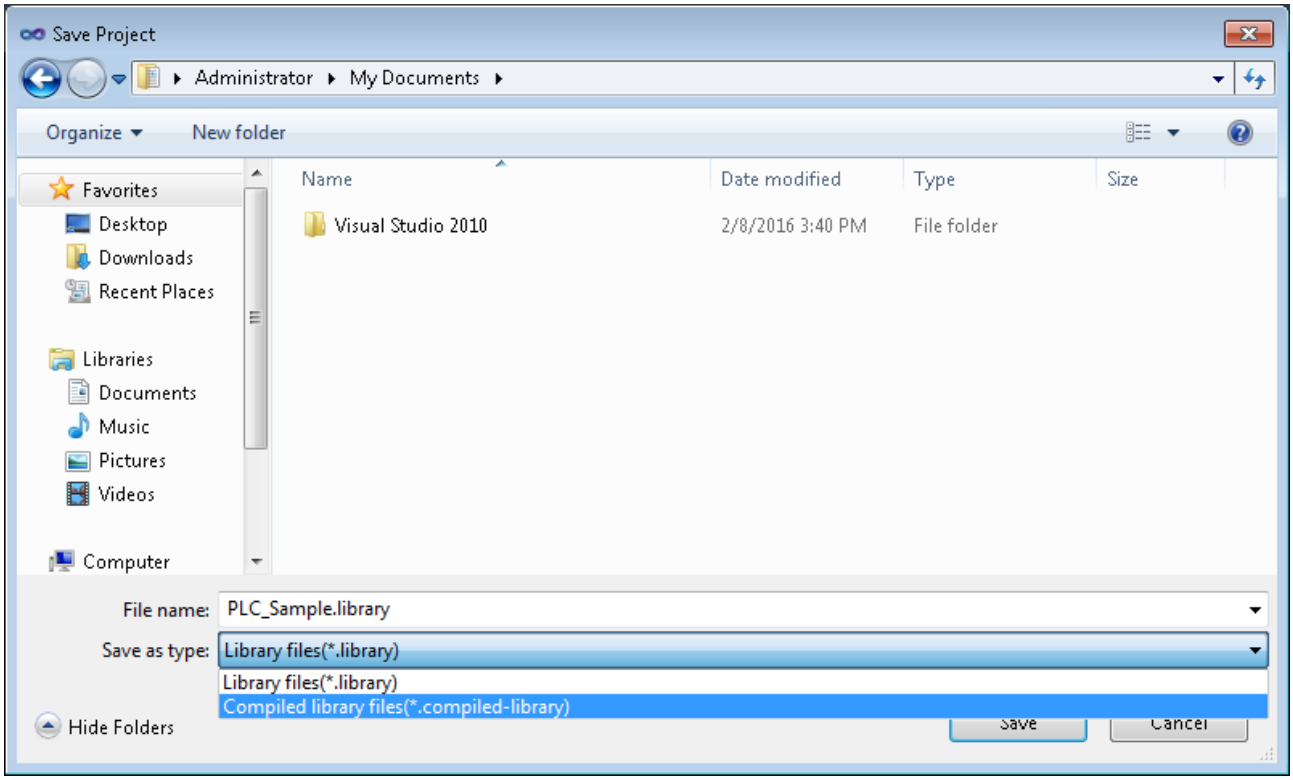
- *.library（源库）

- 在项目树中的 PLC 节点上，您可以使用 **Add Existing Item**（添加现有项目）命令来打开源库（用于查看和/或编辑）。
- 您可以使用常用的调试功能“进入”源库。

- *.compiled-library（编译库）
  - 该文件扩展名可用于以编译格式保存库项目。库的预编译上下文的加密图像被存储起来，这意味着库功能块的实现不再可访问或可见。
  - 无法打开或调试编译库。
  - 除此之外，*.compiled-library 文件的行为与 *.library 文件类似。因此，您可以以同样的方式安装和引用它们。
  - 使用编译库格式可以保护库的源代码。此外，库文件更小，加载时间更短。

代码无法逐步运行

常见的调试功能无法应用于编译库（*.compiled-library）。因此，无法通过调试跳转到 *.compiled-library 的库块。

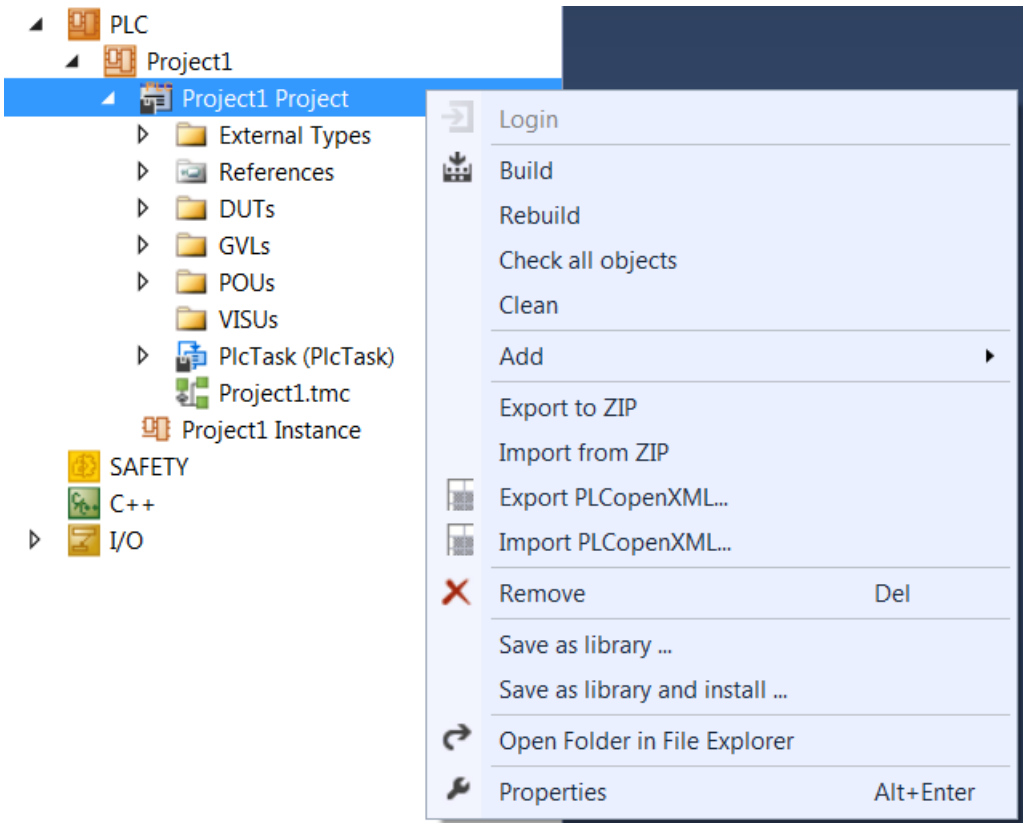


13.2.2 命令另存为库并安装

功能：该命令可打开用于将 PLC 项目另存为 PLC 库的标准对话框。此外，该命令还会在库存储库中安装已存储的库。因此，通过库管理器可以直接将库插入项目中。

调用：解决方案资源管理器中 PLC 项目对象（<PLC project name> 项目）的上下文菜单

该命令可将 PLC 项目另存为 PLC 库，并在 库存储库 [▶ 251] 中安装它。在 PLC 项目的上下文菜单中，可使用保存并安装库的命令。



库的安装与保存是同时进行的，是 命令另存为库 [▶ 249] 的扩展，因为同时会在本地系统上安装库。然后，通过库管理器可以立即将库添加到项目中。

13.3 库安装

- 在项目中使用库之前，必须首先在系统中安装库。在本地系统中不同的“存储库”（目录、存储位置）中，可对库进行管理。在项目中集成库之前，必须将其安装到本地系统中具有定义版本号的存储库中。
  - **例外**：作为库引用的项目。
    请参见 使用 PLC 项目作为引用库。
- 如果在存储库中没有安装所使用的库版本，则会在项目树中的引用处进行标记。
- 库安装在 库存储库 [▶ 251] 中。

13.3.1 库存储库

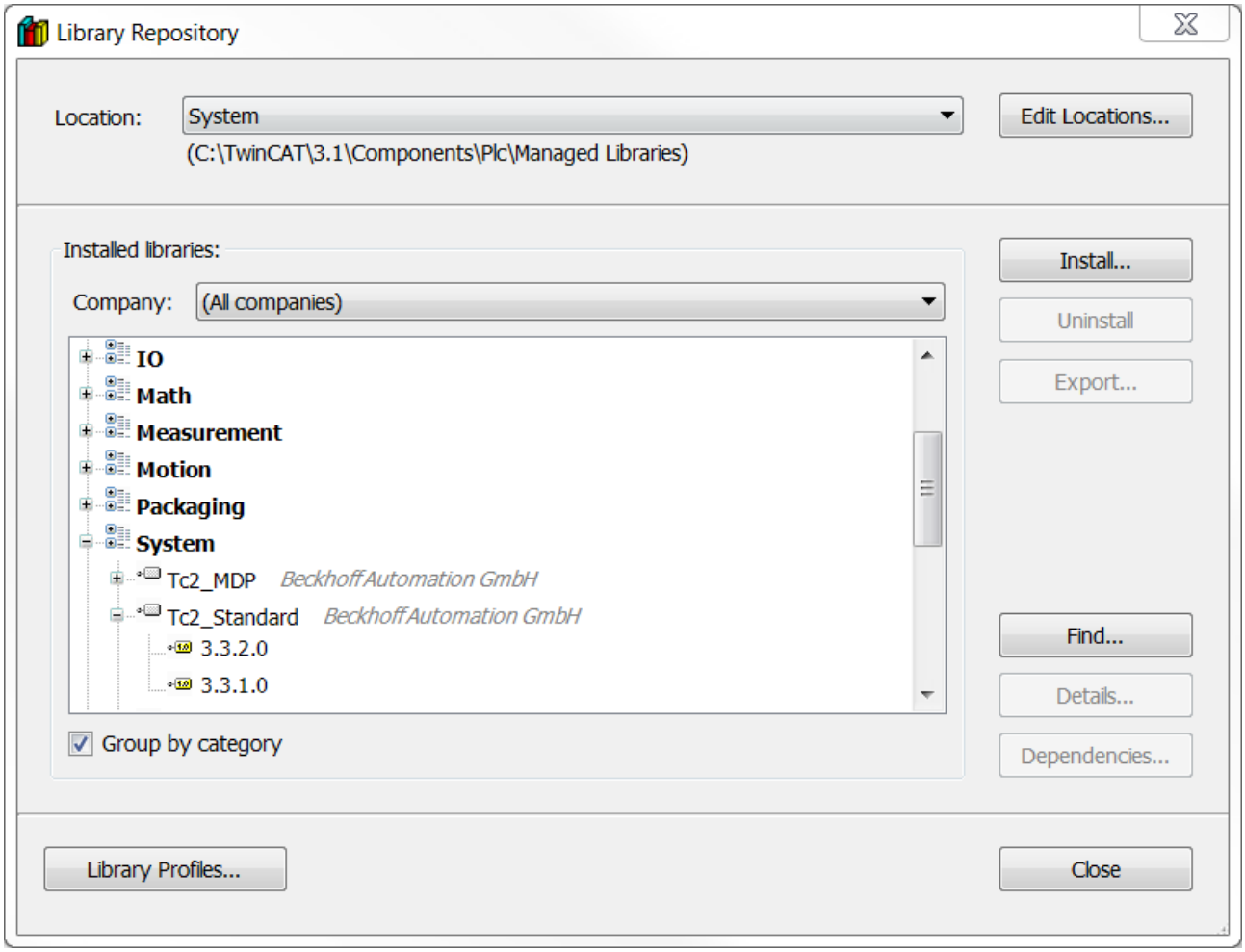
功能：库存储库可用于定义存储位置以及安装或卸载库。

调用：

- 菜单 **PLC**
- PLC 项目树中 **References**（引用）对象的上下文菜单
- 库管理器 [▶ 254] 中的按钮（符号： ）
- 使用命令 添加无占位符解析的库命令 [▶ 330] 之后，通过扩展对话框 **Find library**（查找库）

如要使用 1 个库，务必在存储库中安装它。对于倍福库，这通常会发生在安装 TwinCAT 3 或安装 TwinCAT 3 功能组件期间。库可以源库（*.library）或编译库（*.compiled-library）进行安装。倍福可提供编译库。如果尝试使用存储库中未安装的库版本，则会在项目树中的引用处出现 1 个注释符号。

库存储库中包含所有已安装库。该列表可根据库类别进行排序和显示（已启用选项 **Group by category**（按类别分组）），或根据库标题按字母顺序进行排序和显示（已禁用选项 **Group by category**（按类别分组））。根据类别对库进行排序时，类别会显示为节点。点击节点可以打开相关库或子类别的列表；点击库名称可以打开已安装库版本的列表。



按钮和命令

在 Library Repository（库存储库）中提供以下按钮和命令。

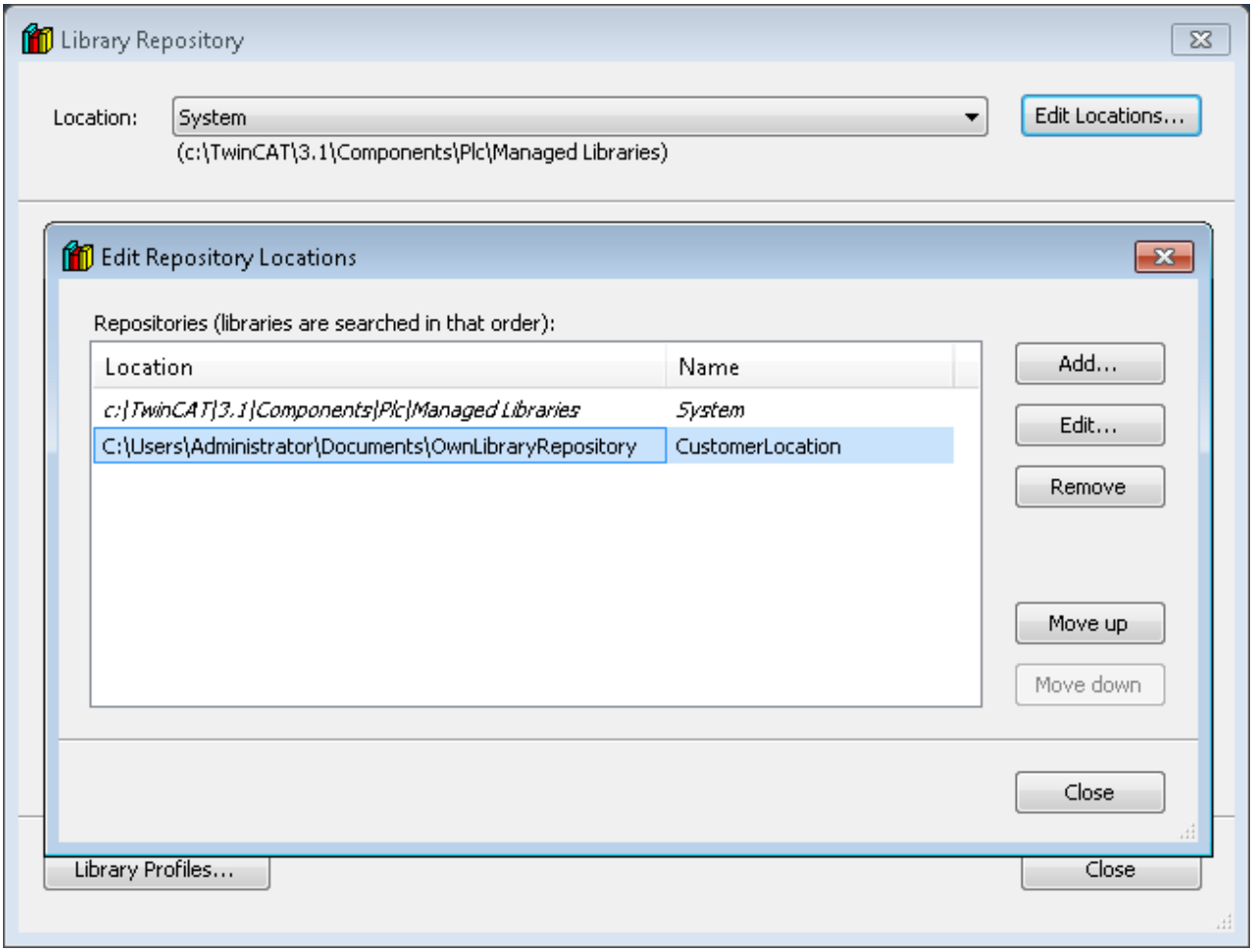
位置

显示本地系统中存储库文件的目录。在 Installed libraries（已安装库）部分会列出该存储位置的库。如果系统上存在多个存储库目录，则此时可以选择 1 个存储库目录进行管理。	
编辑	打开对话框 Edit Repository Locations（编辑存储库位置）。

对于新存储库，您只能使用空目录或有效的现有存储库。

编辑存储库位置

列出存储库的位置和名称。	
添加	通过指定存储库名称和目录来创建新的存储库。 打开对话框 Repository Location（存储库位置）。所选目录（Location（位置）输入字段）必须为空或现有的有效存储库。Name（名称）是符号存储库名称的输入字段。
编辑	打开 Repository Location（存储库位置）对话框（请参见“添加”），可编辑当前所选的库存储库。
删除	系统会询问用户是只删除存储库列表条目，还是应该从文件系统中删除包含库文件的整个目录。如果您想要删除目录，则需要对此进行确认。
向上移动	用于更改存储库顺序的命令，可将当前所选的存储库向上移动 1 个位置。
向下移动	用于更改存储库顺序的命令，可将当前所选的存储库向下移动 1 个位置。



已安装库

树状结构的库列表。每个库均显示类别、名称、公司和版本。	
公司	用于筛选所显示的库的选择列表。
安装	打开 **Select Library**（选择库）对话框，可选择文件系统中的库文件。在本地库存储库中将会安装所选的 *.library 或 *.compiled-library 类型的库文件。然后可以在项目中使用该库。
卸载	卸载所选的库版本。然后再不能在项目中使用它。
导出	该命令让您能够导出在库存储库中安装的库，并将其存储到所需的位置。该命令会打开 **Export Library**（导出库）对话框，用于选择位置。您可以按照安装在存储库中的格式（作为源库 *.library 或作为 *.compiled-library）导出库。
搜索	打开 **Find Library**（查找库）对话框，可查找库名称或库元素。双击搜索结果或选择搜索结果 + [Open]，可关闭搜索对话框，相应的库将在库存储库中被标记。
详细信息	打开详细信息 [▶ 332]对话框，可了解所选的库版本。
相关性	打开相关性 [▶ 332]对话框，可了解所选的库版本。
按类别分组	• ☑：按库类别分组 • ☐：按字母顺序排序 通过外部描述文件“*.libcat.xml”对类别进行定义。

库配置文件

如果在项目中设置特定的编译器版本，则库配置文件会定义 TwinCAT 解析库占位符所使用的库版本。	
导入	导入 *.libraryprofile 文件。 如果导入内容中包含现有的占位符条目，则会出现提示，询问 TwinCAT 是否应该覆盖它们。
导出	导出扩展名为“.libraryprofile”的 xml 文件，其中包含所选占位符条目的赋值。您可以将选择限制为编译器版本中的单个条目。

库存储库中已转换的 TwinCAT 2 库

库存储库包含已转换的 TwinCAT 2 库（Tc2_<LibraryName>）和新的 TwinCAT 3 库（Tc3_<LibraryName>），这些库在 TwinCAT 2 中不可用。

这些库可以将使用 TwinCAT 2 库的 TwinCAT 2 项目转换为使用兼容的 TwinCAT 3 库的 TwinCAT 3 项目，而无需在 PLC 代码中进行重大更改。因此，Tc2_<LibraryName> 库是已转换的 TC2 库。TwinCAT 3 中库的命名与 TwinCAT 2 中库的命名相似。为了简化问题，一些库已被组合在一起。项目转换器会自动将 TwinCAT 2 .pro 文件的 TC2 库分配给 TwinCAT 3 项目中的 TC3 库。

PLC 库分配 TC2 库的列表 → TC3 库存储在 TwinCAT 选项中，可在此处扩展：
Tools\Options\TwinCAT\PLC Environment\Libraries

在转换项目之前，务必替换旧库（即，MC → MC2、COMlib → COMlibv2、PLCSYSTEM → TcSystem、……）。

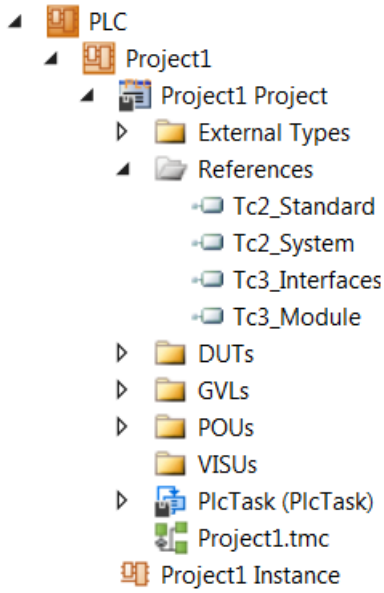
13.4 库管理

- 库管理器 [▶ 254]很好地概述了在项目中使用的 PLC 库引用，可用于在项目中集成库或占位符。集成后，可在项目中使用库引用元素。
- 在库管理器中还会显示在另一个库中作为子库而被引用的库引用。此外，还有“隐藏库”（请参见 命令属性 [▶ 333] 部分）。
- 每当使用 1 个库时，都会引用 1 个唯一的库版本。该版本被指定为有效版本。如果库附带 1 个固定版本（例如，3.3.0.0），则项目将始终使用该库版本，即使有更新的版本可用也是如此。或者，也可以使用设置“Always newest”（始终最新）/“*”来自动确保始终使用最新版本的库。在这种情况下，TwinCAT 总是会使用在库存储库中找到的最新版本的库。有关更多信息和示例，请参见 命令设置为始终最新版本 [▶ 334]。
- 无法将同一个库的相同版本多次添加到库管理器中。在库管理器中，1 个版本的库可以作为库或占位符而被引用。
- 如果库没有经过编译（*.compiled-library），而是以*.library 文件的形式存在，则可通过双击相应条目的方式打开在库管理器中列出的库元素。
- 您可以将库引用以库或占位符的形式添加到库管理器中，并将其包含在应用程序中（请参见 命令添加库 [▶ 329] 部分）。只要可能，您应该使用占位符。有关更多信息，请参见 库占位符 [▶ 257] 部分。
- 在项目中处理库模块时，将按照在库存储库 [▶ 251]中列出的顺序来搜索库和存储库。有关更多信息，请参见对库模块或变量的明确访问 [▶ 247]部分。

13.4.1 库管理器

功能：库管理器可用于在项目中集成和管理库。它很好地概述了在项目中使用的 PLC 库。

调用：在 PLC 项目树中，双击 **References**（引用）对象



在消息窗口中，与库管理器相关的编译错误会以专用类别输出。

库管理器的上部

对话框的上部可显示当前在项目中集成的库。	
名称	标题、版本和公司名称，在库创建 [▶ 249]期间在项目属性 [▶ 841]中进行定义。
命名空间	库的命名空间的默认设置为库标题。或者，也可以明确定义不同的命名空间，通常在库生成 [▶ 249]期间在项目属性 [▶ 841]中针对库定义命名空间，或者在库引用的属性窗口 [▶ 333]中针对项目中的库的本地使用情况定义命名空间。 库的命名空间必须用作标识符的前缀，这样才能对项目中多次使用的模块进行明确访问。
有效版本	引用的有效库版本。如果库被用作“始终最新”/“*”，则此处会显示实际的库版本。有关占位符解析的更多信息，请参见 占位符 [▶ 258] 部分。

- 由插件自动插入项目中的库（例如，可视化或 UML 库）将以灰色字体显示。手动添加的库（**Add library...**（添加库……））将以黑色字体显示。
- 库名称前的符号表示库类型：
  - : TwinCAT 3 PLC 库（包含版本信息）
  - : 未找到的引用文件或引用文件不是有效的库文件（请参见 **Error List**（错误列表）视图中的相应消息）。在这种情况下，请参见：命令尝试重新加载库 [▶ 331]
- 如果 1 个库与其他库有相关性 [▶ 332]，则也会自动集成它们（如果找到的话），并在条目的子树中显示它们，前面带有 1 个符号 。通过加号或减号可以打开或关闭此类子树。例如，下图显示了作为直接库和作为库“Tc2_System”的子库的库“Tc2_Standard”。

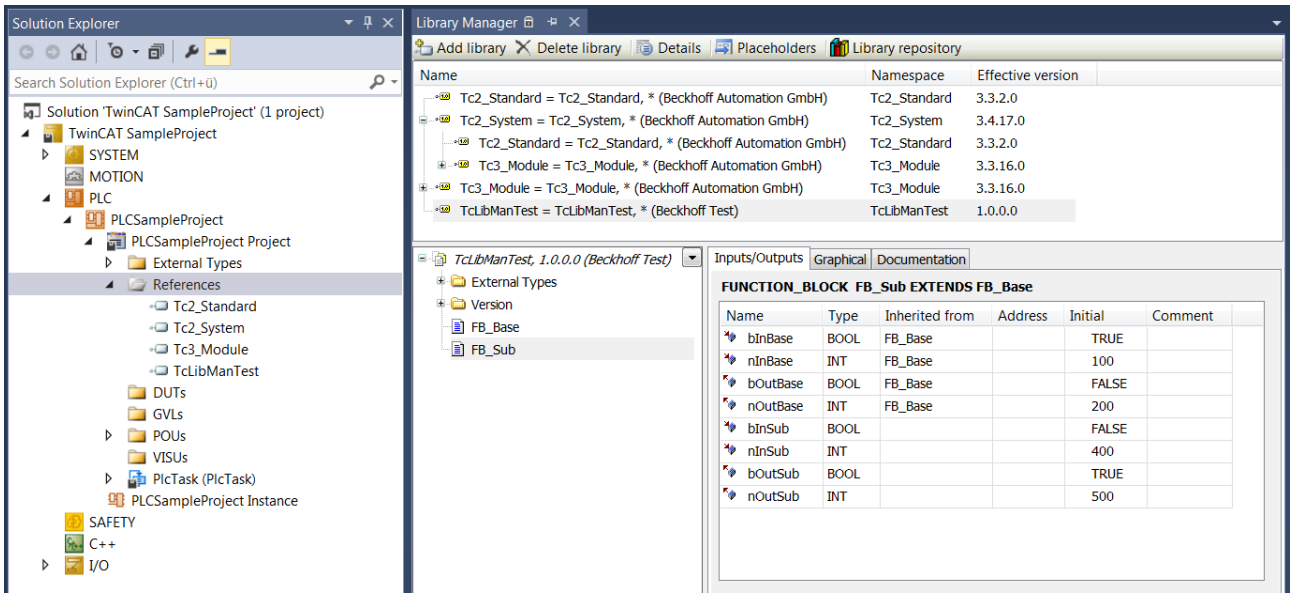
按钮和命令

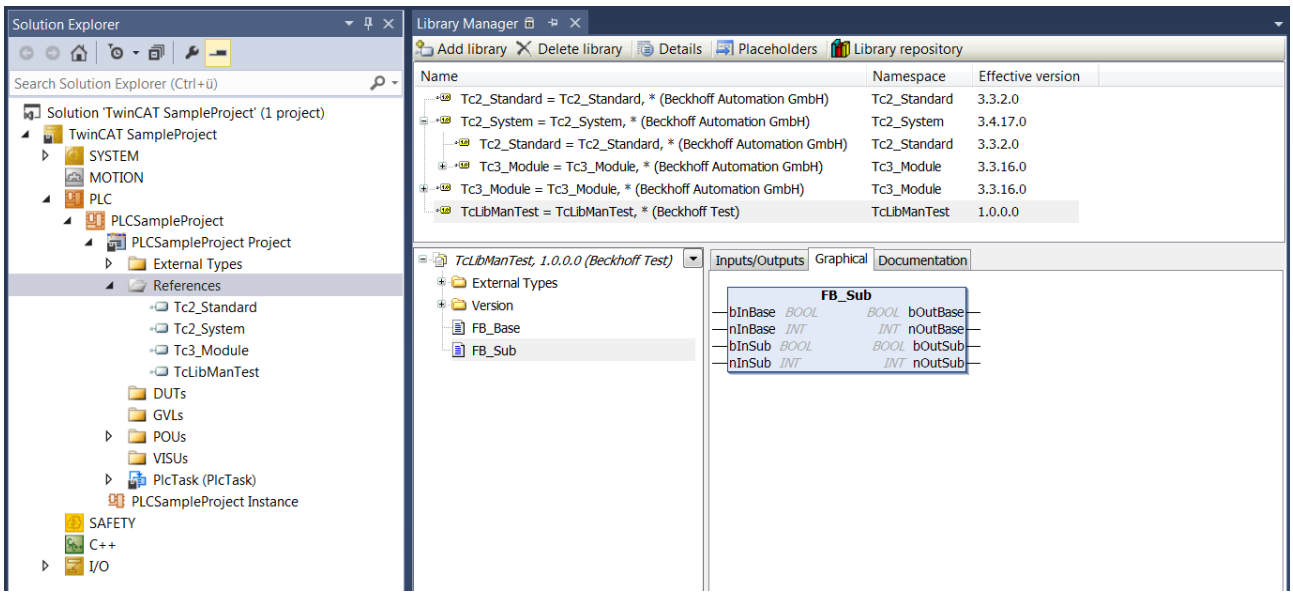
在库管理器的上部提供以下按钮和命令。

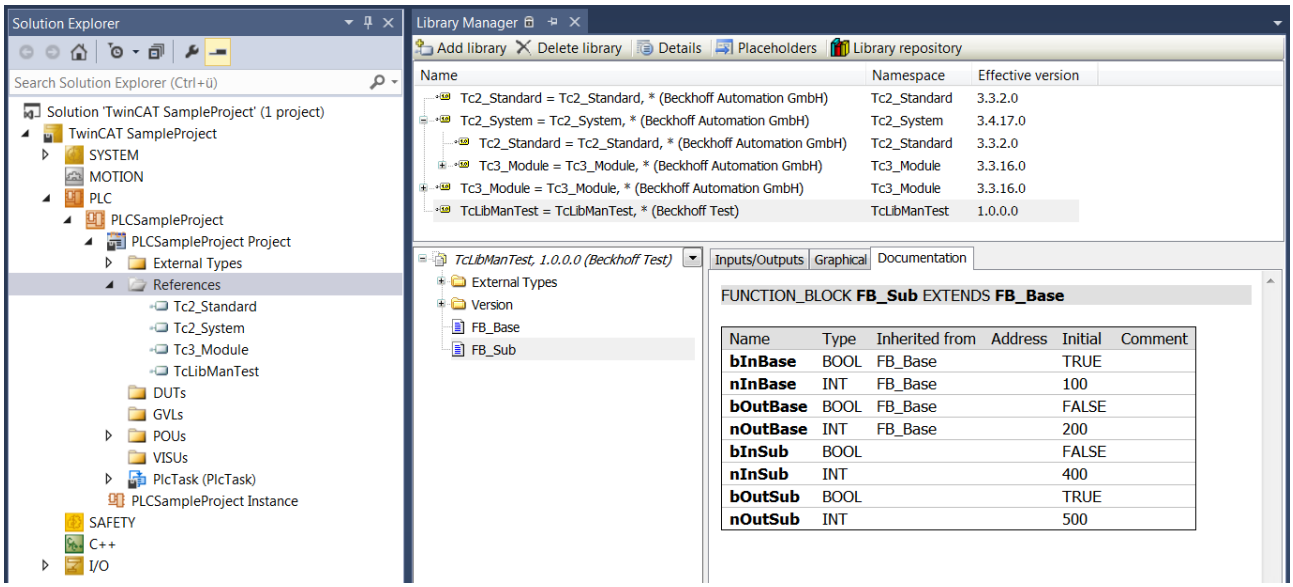
添加库	在项目中集成库或占位符。另请参见 命令添加库 [▶ 329]。
删除库	该命令可从库管理器中删除所选的库。
详细信息	打开详细信息 [▶ 332]对话框，可了解所选的库。
占位符	打开占位符 [▶ 258]对话框。
库存储库	打开 库存储库 [▶ 251] 对话框。
尝试重新加载库	如果在打开项目时未能加载所选的库，则可在库管理器的编辑器窗口中使用该命令。另请参见 命令尝试重新加载库 [▶ 331]

库管理器的下部

编辑器的左下部分会显示在编辑器的上部所选的库的模块。通过在库名称右侧显示的菜单符号，可使用常用的排序和搜索功能。 在右下部分有以下选项卡。	
输入/输出	当前在左侧所选的库对象的组件会以表格形式显示，带有（变量）名称、数据类型、基本功能块（如适用）、地址、基本功能块（如适用）、初始值和注释，如库中所定义。 每个变量前面的符号可指明该变量是输入变量、输出变量，还是输入/输出变量。 • 输入变量：符号带有指向右下方的箭头 • 输出变量：符号带有指向左上方的箭头 • 输入/输出变量：符号带有指向右下方和左上方的符号 方法返回值的符号与输出变量的符号相同。
库参数	只有当前在左侧所选的库对象是参数列表时，该选项卡才可用。 参数列表中的变量都会在 1 个表格中显示，带有名称、数据类型、（可编辑的）值和注释，如库中所定义。 在**（可编辑的）值**列中，将全局常量的值替换为项目特定值。有关更多信息，请参见 对象参数列表 [▶ 67]。
图形	功能块的图形表示
文档	当前在左侧所选的库对象的组件会以表格形式显示，带有（变量）名称、数据类型、基本功能块（如适用）、地址、初始值和注释，它们可能在创建库 [▶ 249] 时已被添加到声明中。因此，添加此类注释是 1 种自动向用户提供功能块文档的便捷方式。有关更多信息，请参见 库文档 [▶ 260] 部分。







13.5 库占位符

库占位符是引用特定库的占位符。占位符既可以被解析为固定版本，也可以被解析为该库的“始终最新”版本。

有关如何创建新的库占位符的说明，请参见添加库命令部分。

项目中的所有占位符的解析均在应用程序层面进行设置。这意味着，在应用程序层面，您可以设置在应用程序层面直接集成的占位符的解析以及在引用库中使用的占位符的解析。
换句话说，您可以从外部指定在另一个库中集成的库占位符应被解析到哪个库。更改内部使用的库占位符的版本时，无需再次保存外部库。有关如何设置占位符解析的更多信息，请参见 更改占位符解析 [▶ 259] 部分。

由于这些选项，对所使用的库版本的调整变得几乎毫不费力，从而使项目和库的工程流程变得非常灵活。

在项目中集成库引用时，建议使用占位符，而不是库。

示例

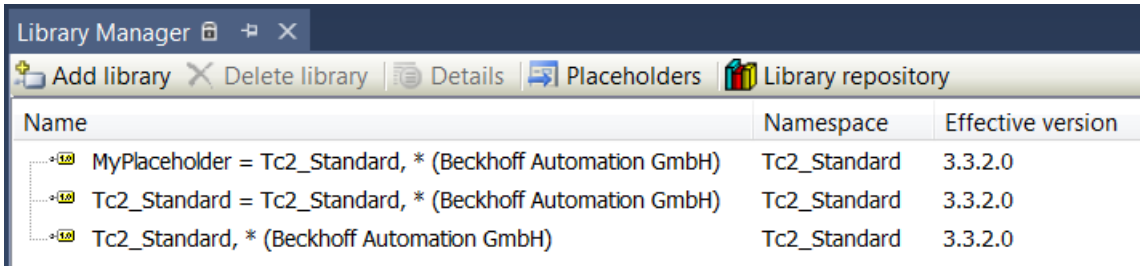
- 应用程序项目需要库 LibA 和 LibB。
- 库 LibB 也需要访问 LibA 的模块。
- 因此，LibA 既可用作应用程序中的直接库，也可用作 LibB 的子库。

- 通过引用 LibA 作为占位符（在应用程序和 LibB 中），在应用程序项目中定义占位符解析，这样用户可以指定在整个项目（应用程序和 LibB）中应该使用哪个版本的 LibA。
- 例如，如果需要在 LibA 中进行更改，但 LibA 的接口要保持不变并确保与先前版本的兼容性，这样可以很容易地在整个项目（应用程序和 LibB）中使用新的库版本。
  为此，修改后的 LibA 将以新的版本号进行创建和安装（例如，从版本 1.0.0.0 更改为 1.0.0.1）。如果占位符 LibA 在应用程序项目中被解析为固定的库版本，即 1.0.0.0，则占位符解析也会相应变为 LibA 版本 1.0.0.1。不过，如果在应用程序项目中将占位符 LibA 定义为“最新版本”，则无需进行调整，即在安装 LibA 版本 1.0.0.1 之后，在整个项目（应用程序和 LibB）中将自动引用并使用该版本。

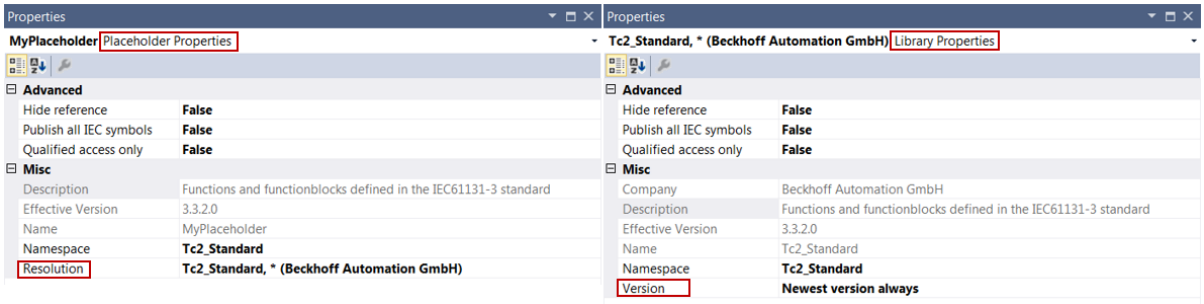
检测库和占位符

根据库管理器 [▶ 254]和属性窗口 [▶ 333]中的差异，可以确定库引用是作为库集成还是作为占位符集成。

- 库管理器：如果使用占位符，则会将库名称分配给占位符名称，以指示占位符所引用的“实际”库（例如，MyPlaceholder = Tc2_Standard, * (Beckhoff Automation GmbH)）。而库却不是这样：此处只会显示库名称（Tc2_Standard, * (Beckhoff Automation GmbH)）。
  在下面的截图中，库管理器中的前 2 个条目是占位符，第 3 个条目表示使用了 1 个库。



- 属性窗口：属性窗口的第 1 行指明元素指的是占位符属性还是库属性。
  相应地，在属性窗口中显示的属性也不同：占位符被**解析**为特定的库（包括特定的库版本），而库是固定的，只能调整**版本**。在这 2 种情况下，可在属性窗口中配置命名空间。



13.5.1 占位符

功能：该命令可打开 Placeholder（占位符）对话框。占位符概览列出了项目中的所有库占位符及其当前的解析定义。此对话框还可用于管理占位符和指定占位符库版本（例如，解析到特定的库版本或“始终最新”/“*”）。对于只在其他库内部使用的占位符，此时也可以进行版本解析。

调用：

- PLC 项目树中 References（引用）对象的上下文菜单
- 库管理器 [▶ 254] 中的按钮（符号： ）

默认情况下，占位符被解析为库的“始终最新”/“*”版本。在占位符对话框中，您可以将在项目中直接或内部使用的每个占位符解析重定向到不同版本的库，或者将占位符解析到不同的库。

对话框的结构

Placeholder（占位符）对话框分为 3 列。

名称	占位符的名称
库	将占位符解析到的库的名称 • 库列的粗体标记： 如果未按照默认解析（通常为“始终最新”/“*”）来解析占位符，则会在 **Placeholder**（占位符）对话框中将库标记为粗体。当库已被改为用户特定的解析时，就会出现这种情况。在这种情况下，**Info**（信息）列会指明“Resolved by placeholder redirection“（按占位符重定向解析）。 • 更改要将占位符解析到的库： 双击 **Library**（库）列中的字段，可打开所选行下方的选择列表，您可以使用该选择列表来更改要将占位符解析到的库。有关更多信息，请参见 更改占位符解析 [▶ 259] 下的内容。
信息	关于占位符解析类型的信息 • 信息“Resolved by library profile”（按库配置文件解析）意味着（倍福）占位符会根据库中的规范进行解析，因此也是根据默认解析进行解析。 • 信息“Resolved by default library”（按默认库解析）意味着占位符会根据其默认解析（“始终最新”/“*”）进行解析。 • 信息“Resolved by placeholder redirection”（按占位符重定向解析）意味着该项目中的占位符解析已被更改。没有使用默认解析，而是使用用户自己设置并保存在 PLC 项目文件中的解析。

13.5.2 更改占位符解析

有 2 种方法可以设置应该将占位符解析到的库：

- 通过 **Placeholder**（占位符）对话框
- 通过 **Properties**（属性）窗口

您可以使用占位符对话框来设置项目中的所有占位符，即在应用程序层面直接集成的占位符和在引用库中使用的占位符。

属性窗口只能用于设置在应用程序层面可用的占位符。

可能性 1：通过占位符对话框

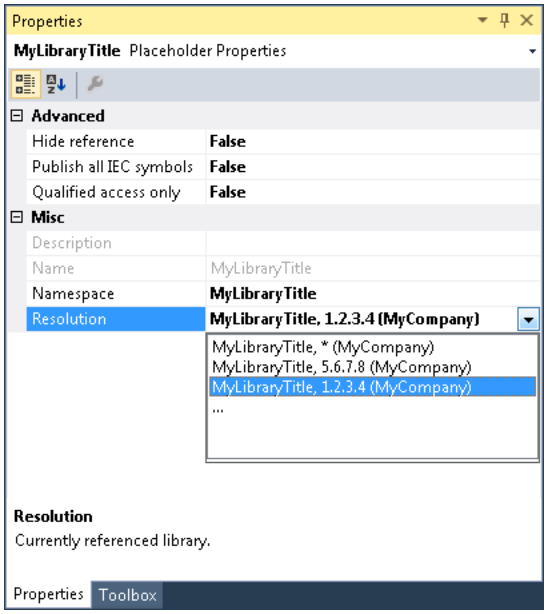
- 打开占位符 [▶ 258]对话框。
- 双击 **Library**（库）列中属于您想要为其更改解析的占位符所在行的字段。
- 在所选行的下方会打开 1 个对话框，通过该对话框您可以选择所需的库。对话框提供以下选项。

重置为默认值	该命令仅适用于当前无法根据其默认解析进行解析的占位符。例如，如果您添加倍福占位符，并将占位符解析更改为另一个库版本，就会出现这种情况。使用命令 **Reset to default**（重置为默认值），您可以重新根据默认解析来解析此类占位符。这样做时，通常会使用“始终最新”/“*”来选择同名的库。
所选库的其他版本	如果该区域显示您想要将占位符解析到的库版本，则您可以点击该版本。然后，占位符会被解析到相同的库，但该库的版本不同。
其他库	如果您想要将占位符解析到另一个库，则您可以通过 **Other library**（其他库）打开对话框，在该对话框中，您可以从所有已安装的库中选择所需的库。如果您在该对话框中激活 **Display all versions**（显示所有版本）选项，则您不仅可以指定所需的库，还可以选择该库的特定版本。

可能性 2：通过“属性”窗口

- 打开属性 [▶ 333]窗口。
- 在“References”（引用）节点下方的项目树中，选择 1 个占位符。
- “Properties”（属性）窗口会显示该占位符的属性。
- **Resolution**（解析）字段提供了 1 个下拉列表，其中包含先前所选库的可用版本和条目“...”.
  - 如果该区域显示您想要将占位符解析到的库版本，则您可以点击该版本。然后，占位符会被解析到相同的库，但该库的版本不同。

- 如果您想要将占位符解析到另一个库，则您可以通过“…”打开对话框，在该对话框中，您可以从所有已安装的库中选择所需的库。如果您在该对话框中激活 **Display all versions**（显示所有版本）选项，则您不仅可以指定所需的库，还可以选择该库的特定版本。



13.6 库文档

您可以使用注释来记录库对象和库对象的各个元素，以解释库所提供的程序元素的功能和使用。

通过注释创建的文档会以文本或表格形式在库管理器（库已集成在其中）的 **Inputs/Outputs**（输入/输出）和 **Library Parameters**（库参数）选项卡（关于库对象的各个元素的描述）以及 **Documentation**（文档）选项卡（关于库对象的一般描述）中显示。（请参见 基础知识 [▶ 260]）

TE1030 | TwinCAT 3 Documentation Generation

如想在文档领域获得更具吸引力的演示效果和更广泛的功能，我们建议使用 TE1030 | TwinCAT 3 Documentation Generation 工具。

TwinCAT 3.1 Build 4024 及以下版本均支持 reStructuredText

在 Build 4024 及以下版本中，文档格式 reStructuredText 均可供选择。

另请参见：

- 库管理器 [▶ 254]

13.6.1 基础知识

TwinCAT 提供借助注释记录库对象的选项。您可以对库对象本身以及库对象的各个元素使用注释：

- 关于库对象的一般描述 [▶ 260]
- 关于库对象的各个元素的描述 [▶ 261]

您可以使用注释来描述对象/元素的目的，以及库用户可以或应该如何使用它。在集成库的库管理器中会显示文档。注释可以是单行（注释运算符“//...”）或多行（注释运算符“(*...*)”）。

关于库对象的一般描述

通过在库对象的声明行上方添加注释，您可以描述库对象。如果声明行上方有属性，则您可以将注释置于属性的上方或下方。

在库管理器 [▶ 254]的 **Documentation**（文档）选项卡中会显示关于库对象的一般描述。

使用一般注释可以描述下列对象：

- POU 对象（程序、功能块、函数）
- 接口
- 方法、属性
- 全局变量列表、参数列表
- DUT 对象（枚举、结构、联合、别名）

放置注释的示例：

关于功能块的一般描述

```
// General FB comment
FUNCTION_BLOCK FB_Lib
VAR
…
```

关于全局变量列表的一般描述

```
// General GVL comment. The comment can be placed over possible attributes.
{attribute 'qualified_only'}
VAR_GLOBAL
    fGlobal1     : REAL;
END_VAR
```

关于参数列表的一般描述

```
{attribute 'qualified_only'}
// General parameter list comment. The comment can be placed below possible attributes.
VAR_GLOBAL CONSTANT
    cParameter1  : LREAL := 12.34;
END_VAR
```

扩展库文档（reStructuredText）

如果您在库的项目属性中选择文档格式 **reStructuredText**，并根据 reStructuredText 的语法为关于库对象的一般描述标注注释文本，则您可以获得更多文档选项。（请参见 扩展 - reStructuredText [▶ 266]）

关于库对象的各个元素的描述

除了库对象本身，您还可以通过为每个元素添加注释的方式来单独描述库对象的各个元素。您可以将单个变量的注释置于变量声明上方的行中，也可以将其置于变量声明之后的同一行中。您可以在方法声明行的返回类型之后插入注释，对方法或函数的返回类型进行注释。

在库管理器 [▶ 254]的 **Input/Output**（输入/输出）和 **Documentation**（文档）选项卡中出现的表格清楚地显示了库对象的各个元素的文档。如果库对象是参数列表，则会在 **Library Parameters**（库参数）和 **Documentation**（文档）选项卡中显示表格。

表格会显示库管理器中的以下元素的注释：

- 各个变量的注释
  - POU 对象的输入/输出（程序、功能块、函数）
  - 方法的输入/输出
  - 全局变量（全局变量列表）
  - 全局常量（全局变量列表、参数列表）
  - DUT 对象的变量（枚举、结构、联合变量）
- 返回类型（方法、函数）的注释

放置注释的示例：

关于各个功能块变量的描述

```
FUNCTION_BLOCK FB_Lib
VAR_INPUT
    // Comment for the first input placed over the declaration
    bInput1   : BOOL;
    bInput2   : BOOL;        // Comment for the second input placed behind the declaration
END_VAR
```

关于方法的返回类型的描述

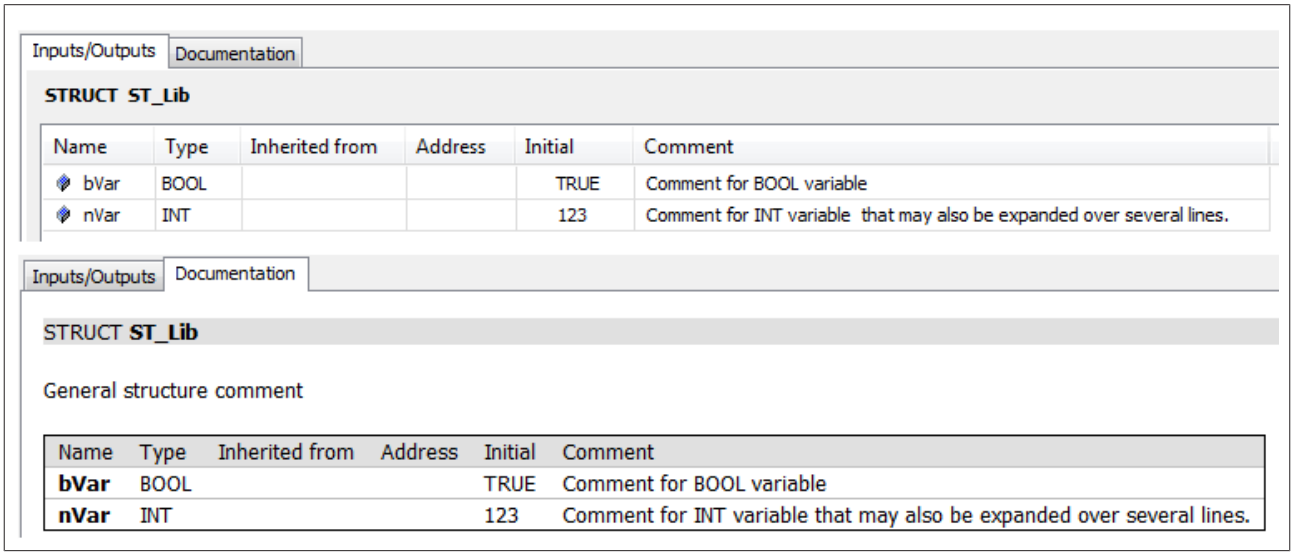
```
// General method comment
METHOD SampleMethod : BOOL  // Comment for the method's return value
VAR_INPUT
    bInput   : BOOL;        // Comment for input variable
END_VAR
```

示例

结构

在下面的示例中，已为结构本身和结构中的各个变量分配了注释。如果焦点在库管理器下部的结构上，则会在库管理器中显示注释。

```
// General structure comment
TYPE ST_Lib :
STRUCT
    bVar    : BOOL := TRUE;  // Comment for BOOL variable
    nVar    : INT  := 123;   (* Comment for INT variable that may also be
                                expanded over several lines. *)
END_STRUCT
END_TYPE
```

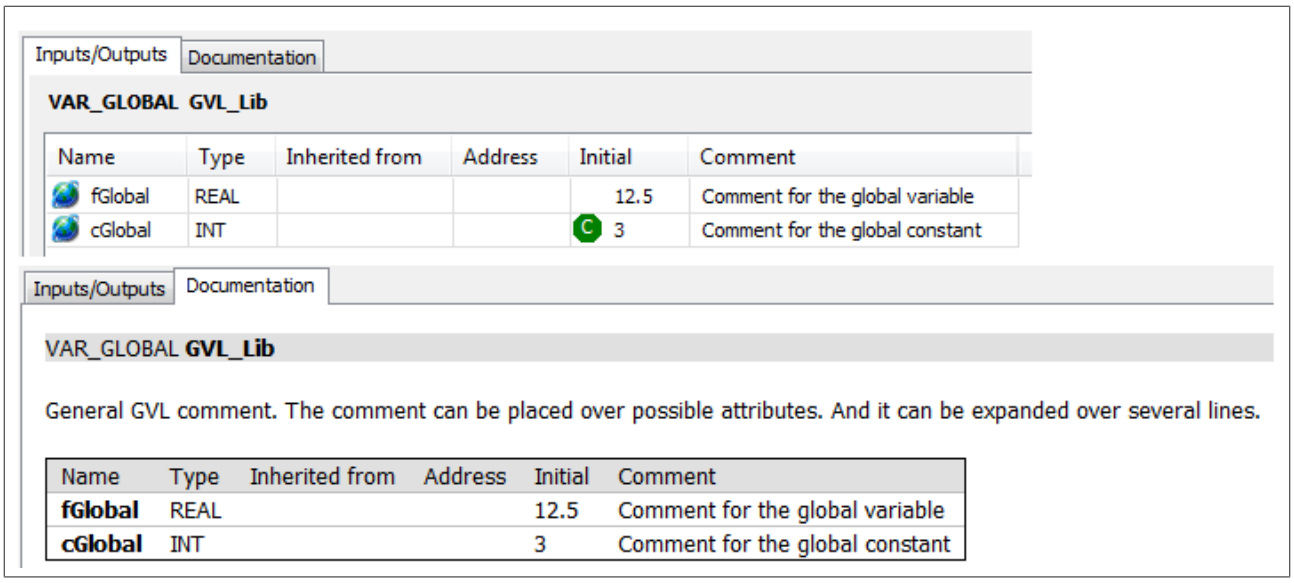


全局变量列表（GVL）

在下面的示例中，已为 GVL 本身和 GVL 中的各个变量分配了注释。如果焦点在库管理器下部的 GVL 上，则会在库管理器中显示注释。

在此示例中，GVL 本身的注释位于所用属性的上方。或者，也可以将注释置于属性的下方。

```
(* General GVL comment.
The comment can be placed over possible attributes.
And it can be expanded over several lines. *)
{attribute 'qualified_only'}
VAR_GLOBAL
    fGlobal    : REAL := 12.5;  // Comment for the global variable
END_VAR
VAR_GLOBAL CONSTANT
    cGlobal    : INT := 3;      // Comment for the global constant
END_VAR
```

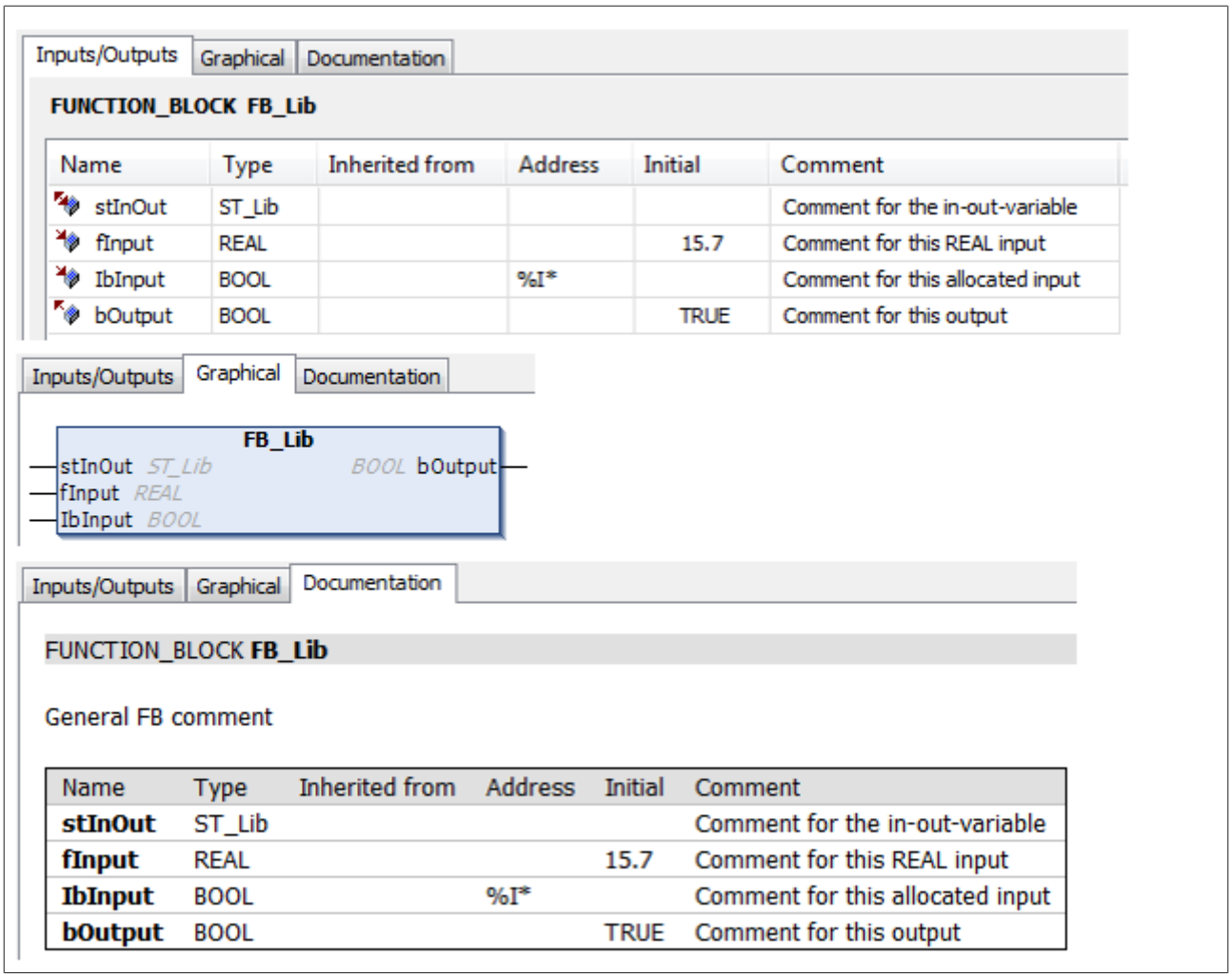


带有方法的功能块

在下面的示例中，1 个功能块有输入/输出变量和 1 个方法。功能块和方法本身、功能块和方法的各个输入/输出变量以及方法的返回类型都有注释。如果焦点在库管理器下部的功能块或方法上，则会在库管理器中显示注释。

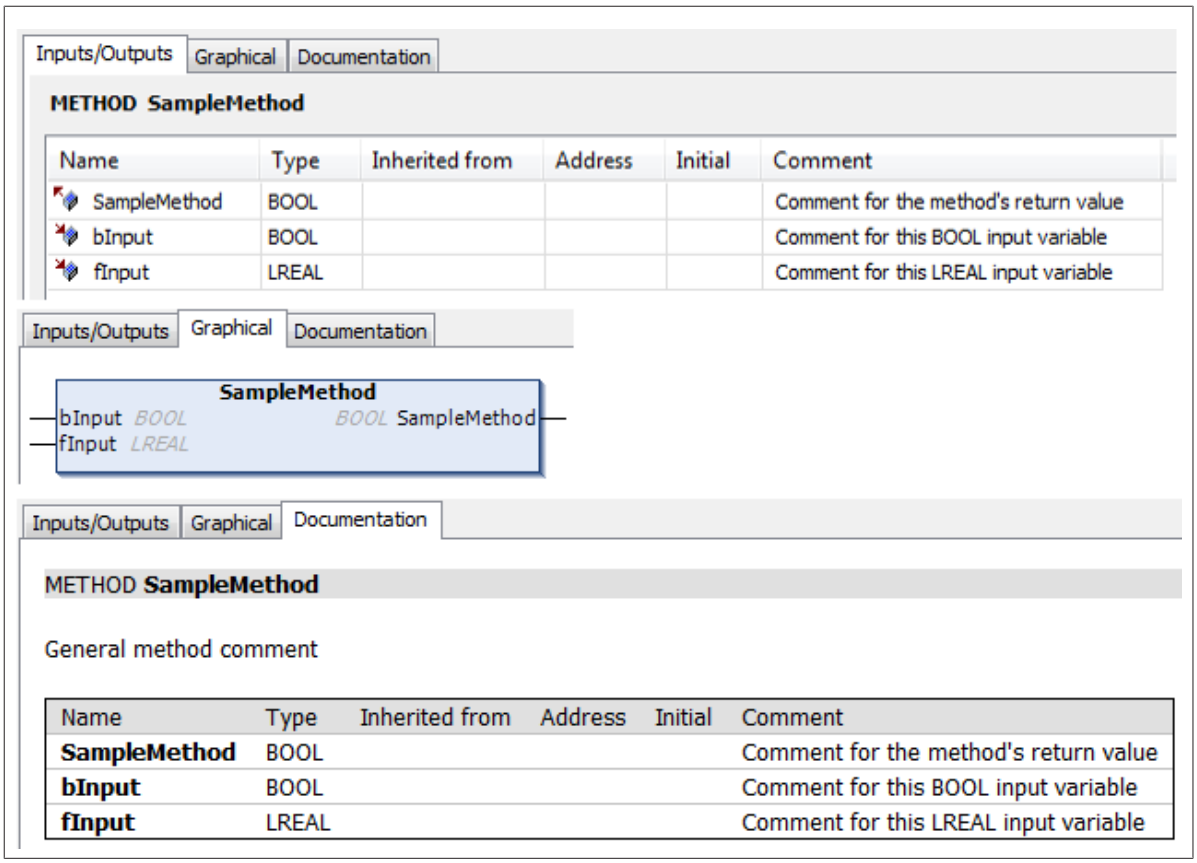
功能块：

```
// General FB comment
FUNCTION_BLOCK FB_Lib
VAR_IN_OUT
    // Comment for the in-out-variable
    stInOut             : ST_Lib;
END_VAR
VAR_INPUT
    // Comment for this REAL input
    fInput              : REAL := 15.7;
    IbInput    AT%I*    : BOOL;           // Comment for this allocated input
END_VAR
VAR_OUTPUT
    bOutput             :  BOOL := TRUE;  // Comment for this output
END_VAR
VAR
END_VAR
```



方法：

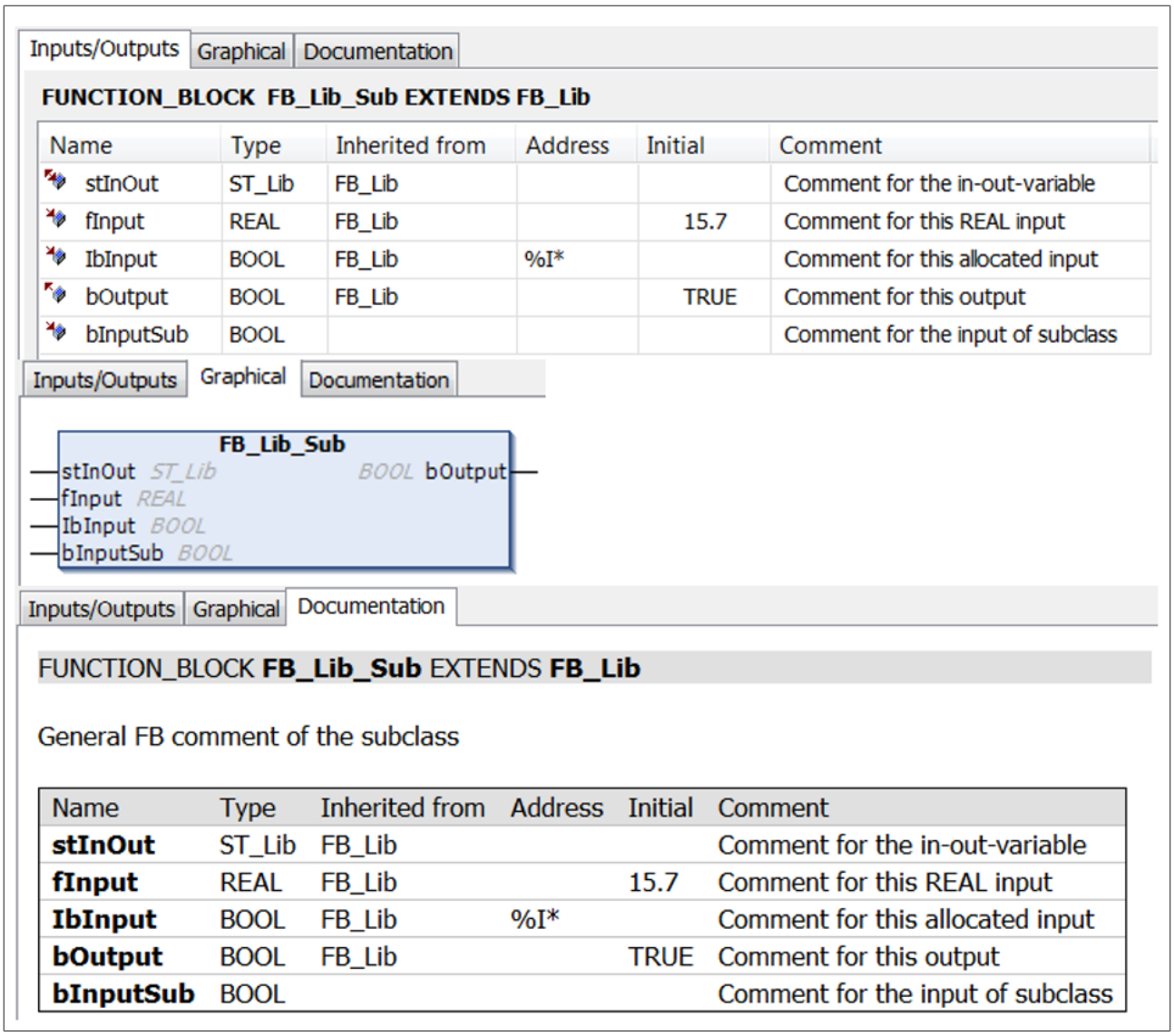
```
// General method comment
METHOD SampleMethod : BOOL   // Comment for the method's return value
VAR_INPUT
    bInput     : BOOL;       // Comment for this BOOL input variable
    fInput     : LREAL;      // Comment for this LREAL input variable
END_VAR
```



作为子类的功能块

在下面的示例中，1 个功能块被声明为上述功能块的子类。子类本身和附加输入变量都有注释。如果焦点在库管理器下部的子类上，则会在库管理器中显示这些注释。子类也会显示超类的变量及其注释。

```
// General FB comment of the subclass
FUNCTION_BLOCK FB_Lib_Sub EXTENDS FB_Lib
VAR_INPUT
    bInputSub  : BOOL;        // Comment for the input of subclass
END_VAR
VAR_OUTPUT
END_VAR
VAR
END_VAR
```



13.6.2 扩展 - reStructuredText

TE1030 | TwinCAT 3 Documentation Generation

如想在文档领域获得更具吸引力的演示效果和更广泛的功能，我们建议使用 TE1030 | TwinCAT 3 Documentation Generation 工具。

TwinCAT 3.1 Build 4024 及以下版本均支持 reStructuredText

在 Build 4024 及以下版本中，文档格式 reStructuredText 均可供选择。

如想在库管理器中以更具吸引力的方式显示库对象文档，您可以在库的项目属性（PLC 项目属性 [▶ 841] > 通用类别 [▶ 841]）中选择文档格式 **reStructuredText**。

reStructuredText 是 1 种易于读取的标记语言，它使用简单的结构来识别文档的结构和特殊文本元素，例如，部分标题、项目符号列表和突出显示。更明确的结构可用于包含图形和说明，或者为文本元素（超链接）分配函数。

您根据 reStructuredText 的语法在库对象声明行上方的注释中标记的文本元素，将在创建库时在库管理器的 **Documentation**（文档）选项卡中进行结构化和格式化，并可选择为其分配函数。

对下列对象声明行上方的注释进行解释：

- POU 对象（程序、功能块、函数）
- 接口

- 方法、属性
- 全局变量列表、参数列表
- DUT 对象（枚举、结构、联合、别名）

另请参见：

- 库创建 [▶ 249]
- 库管理器 [▶ 254]

13.6.2.1 概述

reStructuredText 中的注释由不同的（嵌套的）文本主体元素组成，并可通过标题被划分成不同的部分。

下面将从 1 个介绍性示例开始，概述 reStructuredText 标记的语法。

在描述的第 1 部分，介绍了 reStructuredText 的基本概念和语法。在描述的第 2 部分，即更面向应用的部分，解释了使用不同语法元素的各种注释选项。

正确使用语法是在库管理器中准确无误地显示内容的前提。

根据 Dokutils 文档（reStructuredText 标记规范）的内容提供信息。

示例	• 示例 [▶ 268]
一般说明 （语法元素 [▶ 274]）	• 空行和缩进 [▶ 275] • 简单和较复杂标记 [▶ 277] • 显式标记块 [▶ 278] • 指令 [▶ 279] • 内联标记 [▶ 280] • 转义机制 [▶ 282] • 引用名称 [▶ 282]
注释结构 （注释结构 [▶ 283]）	• 部分 [▶ 283] • 转换 [▶ 286]
文本块 （文本块 [▶ 287]）	• 文本块（段落） [▶ 287] • 缩进文本块（块引用） [▶ 288] • 面向行的文本块（行块） [▶ 289]
列表 （列表 [▶ 290]）	• 无序枚举列表 [▶ 290] • 有序（编号）枚举列表 [▶ 292] • 定义列表 [▶ 293] • 字段列表 [▶ 294]
表格 （表格 [▶ 295]）	• 简单表格 [▶ 295] • 网格表格 [▶ 297] • CSV 表格 [▶ 299] • 列表表格 [▶ 299]
超链接 （超链接 [▶ 300]）	• 内部超链接 [▶ 302] • 外部超链接 [▶ 303] ◦ 独立超链接 [▶ 305] ◦ 嵌入式 URI 和别名 [▶ 305] • 间接超链接 [▶ 306] • 内联超链接 [▶ 308] • 匿名超链接 [▶ 309] • 脚注 [▶ 310] • 引文 [▶ 317]
替换	• 替换 [▶ 318]
说明元素 （说明元素 [▶ 321]）	• 特定说明 [▶ 321] • 通用说明 [▶ 322]
图像	• 图像 [▶ 322]
代码块	• 码块 [▶ 326]
内部注释	• 内部注释 [▶ 328]
字体样式 （字体样式 [▶ 328]）	• 突出显示文本（斜体） [▶ 328] • 突出显示文本（粗体） [▶ 329] • 内联字面量（等宽字体） [▶ 329]

13.6.2.2 示例

13.6.2.2.1 TwinCAT 3 示例项目

此https://infosys.beckhoff.com/content/1033/tc3_plc_intro/Resources/7644044171.zip可用于下载 TwinCAT 3 示例项目，其中包含本文档中作为独立功能块的代码示例。

将 PLC 项目另存为库，并将该库包含在新项目中，以比较库管理器中的文档显示与原始代码。（请参见 库创建 [▶ 249]）

示例项目的“LibPOUs”文件夹被划分以下各项：

TC3 示例项目		
文件夹	功能块的名称	文档文章
A_Samples		
	FB_DocuSample_FunctionBlock	功能块的文档 [▶ 274]
	FB_DocuSample_SyntaxReminder	reStructuredText 语法提醒 [▶ 270]
B_DocuElements		
代码块	FB_Libdoc_CodeBlock	码块 [▶ 326]
注释结构	FB_Libdoc_Sections	部分 [▶ 283]
	FB_Libdoc_Transitions	转换 [▶ 286]
字体样式	FB_Libdoc_FontStyle	字体样式 [▶ 328]
超链接	FB_Libdoc_AnonymousHyperlinks	匿名超链接 [▶ 309]
	FB_Libdoc_Citation	引文 [▶ 317]
	FB_Libdoc_ExternalHyperlinks	外部超链接 [▶ 303] 独立超链接 [▶ 305] 嵌入式 URI 和别名 [▶ 305]
	FB_Libdoc_IndirectHyperlinks	间接超链接 [▶ 306]
	FB_Libdoc_InlineHyperlinks	内联超链接 [▶ 308]
	FB_Libdoc_InternalHyperlinks	内部超链接 [▶ 302]
	FB_Libdoc_LinkToAnotherObject	链接到另一个对象 [▶ 318]
超链接\脚注	FB_Libdoc_AutomaticallyNumberedFootnotes	自动编号的脚注 [▶ 312]
	FB_Libdoc_AutomaticallySymbolFootnotes	自动生成脚注符号 [▶ 315]
	FB_Libdoc_ManuallyAndAutomaticallyNumberedFootnotes	手动和自动编号的脚注 [▶ 316]
	FB_Libdoc_ManuallyNumberedFootnotes	手动编号的脚注 [▶ 310]
	FB_Libdoc_NamedAutomaticallyNumberedFootnotes	已命名的自动编号的脚注 [▶ 313]
图像	FB_Libdoc_Images	图像 [▶ 322]
内部注释	FB_Libdoc_InternalComments	内部注释 [▶ 328]
列表	FB_Libdoc_DefinitionList	定义列表 [▶ 293]
	FB_Libdoc_FieldList	字段列表 [▶ 294]
	FB_Libdoc_OrderedNumberedEnumerationList	有序（编号）枚举列表 [▶ 292]
	FB_Libdoc_UnorderedEnumerationList	无序枚举列表 [▶ 290]
说明元素	FB_Libdoc_GenericNotes	通用说明 [▶ 322]
	FB_Libdoc_SpecificNotes	特定说明 [▶ 321]
替换	FB_Libdoc_Substitution_Images	替换 [▶ 318]
	FB_Libdoc_Substitution_ReplacementText	替换 [▶ 318]
表格	FB_Libdoc_CSVTable	CSV 表格 [▶ 299]
	FB_Libdoc_GridTable	网格表格 [▶ 297]
	FB_Libdoc_ListTable	列表表格 [▶ 299]
	FB_Libdoc_SimpleTables	简单表格 [▶ 295]
文本块	FB_Libdoc_BlockQuote	缩进文本块（块引用） [▶ 288]
	FB_Libdoc_LineBlock	面向行的文本块（行块） [▶ 289]
	FB_Libdoc_Paragraph	文本块（段落） [▶ 287]

有关此的文档

https://infosys.beckhoff.com/content/1033/tc3_plc_intro/Resources/6773482635.zip

13.6.2.2.2 reStructuredText 语法提醒

下面的示例概述了在库对象注释中使用 reStructuredText 语法的情况，以及在库管理器中的相应表示。（在示例项目 [▶ 268]中：A_Samples\FB_DocuSample_SyntaxReminder）

代码

```
(*
================================================
The reStructuredText Cheat Sheet: Syntax Reminder
================================================
:Info: See <https://infosys.beckhoff.de/.
:Author: Beckhoff <support@beckhoff.com>
:Date: 2017-12-14

.. NOTE:: This syntax reminder is based on the docutils documentation

Section Structure
=================

A reStructuredText comment is made up of body elements, and may be
structured into sections. Section titles are underlined or overlined and underlined.
Sections contain body elements and/or subsections.

Paragraphs are flush-left and separated by blank lines.

Body Elements
=============
Grid table:

+--------------------------------+--------------------------------------+
|Block quotes consist of         |Literal block, preceded by "::"::     |
|indented body elements:         |                                      |
|                                |    If (a=TRUE) Then                  |
|    Block quotes are indented.  |      b=3                             |
|                                |                                      |
+--------------------------------+--------------------------------------+
|| Line blocks preserve line breaks and indents.                         |
||    Useful for adornment-free lists;                                   |
||    long lines can be wrapped                                          |
|     with continuation lines.                                          |
+-----------------------------------------------------------------------+

Simple table:

================  ============================================================
List Type         Examples
================  ============================================================
Bullet list       - This is a bullet list
                  - Items begin with "-", "+", or "*"
Enumerated list   1. This is an enumerated list.
                  2. Items use any variation of "1.", "A)", and "(i)"
                  #. Item can also be auto-enumerated
Definition list   Term is flush-left : optional classifier
                      Definition is indented, no blank line between
                      continue
Field list        :Field name: field body
================  ============================================================

Inline Markup
=============
*emphasis*, **strong emphasis**, ``inline literal text``

| Standalone hyperlink: http://beckhoff.de
| Named reference: Beckhoff_
| Anonymous reference: `Anonymous reference`__
| Footnote reference: [1]_
| Citation reference: [CIT2002]_
| Substitution: |substitution|
| Inline internal target: `Inline internal target`_

Explicit Markup
===============

================  ============================================================
Explicit Markup       Examples (visible in the source code)
================  ============================================================
Footnote          .. [1] Manually numbered or [#] auto-numbered
                        (even [#label]) or [*] auto-symbol
```

```
Citation           .. [CIT2002] This is a citation.
Hyperlink Target   .. _Beckhoff: http://beckhoff.de
                   .. _indirect target: Beckhoff_
                   .. _internal target:
Anonymous Target   __ http://beckhoff.de
Directive ("::")   .. image::, .. code::
Substitution Def   .. |substitution| replace:: This is a substitution
Comment            .. This text will not be shown
================   ==========================================================

Directives
===========

================   ==========================================================
Directive Name     Description
================   ==========================================================
attention          Specific admonition ("caution", "danger",
                   "error", "hint", "important", "note", "tip", "warning")
admonition         Generic titled admonition
image              .. image:: C:\Tc3LibDocImages\SampleLib1\logo.gif
                      :width: 40
code               .. code::

                      if(a=true) then b=3;
list-table         Create a table from a uniform two-level bullet list
csv-table          Create a table from CSV data
================   ==========================================================
*)

FUNCTION_BLOCK FB_DocuSample_SyntaxReminder
VAR_INPUT
    nVarInA : INT;        //First input variable
    nVarInB : INT := 5;   //Second input variable
END_VAR
VAR_OUTPUT
    nVarOut : INT;        //Output variable
END_VAR
```

在库管理器中的表示

Inputs/Outputs | Graphical | Documentation

FB_DocuSample_SyntaxReminder (FB)

FUNCTION_BLOCK FB_DocuSample_SyntaxReminder

The reStructuredText Cheat Sheet: Syntax Reminder

Info: See <https://infosys.beckhoff.de/.
Author: Beckhoff <support@beckhoff.com>
Date: 2017-12-14

Note

This syntax reminder is based on the docutils documentation

Section Structure

A reStructuredText comment is made up of body elements, and may be structured into sections. Section titles are underlined or overlined and underlined. Sections contain body elements and/or subsections.

Paragraphs are flush-left and separated by blank lines.

Body Elements

Grid table:

Block quotes consist of indented body elements:
 	
Block quotes are indented.
	Literal block, preceded by "::":
	
	`If (a=TRUE) Then`
	`  b=3`

Line blocks preserve line breaks and indents. Useful for adornment-free lists; long lines can be wrapped with continuation lines.	

Simple table:

List Type	Examples
Bullet list	▪ This is a bullet list ▪ Items begin with "-", "+", or "*"
Enumerated list	1. This is an enumerated list. 2. Items use any variation of "1.", "A)", and "(i)" 3. Item can also be auto-enumerated
Definition list	Term is flush-left : *optional classifier* Definition is indented, no blank line between continue
Field list	**Field name:** field body

Inline Markup

emphasis, **strong emphasis**, `inline literal text`

Standalone hyperlink: http://beckhoff.de
Named reference: Beckhoff
Anonymous reference: Anonymous reference
Footnote reference: [1]
Citation reference: [CIT2002]
Substitution: This is a substitution
Inline internal target: `Inline internal target`_

Explicit Markup

Explicit Markup	Examples (visible in the source code)
Footnote	[1] Manually numbered or [#] auto-numbered (even [#label]) or [*] auto-symbol
Citation	[CIT2002] This is a citation.
Hyperlink Target	

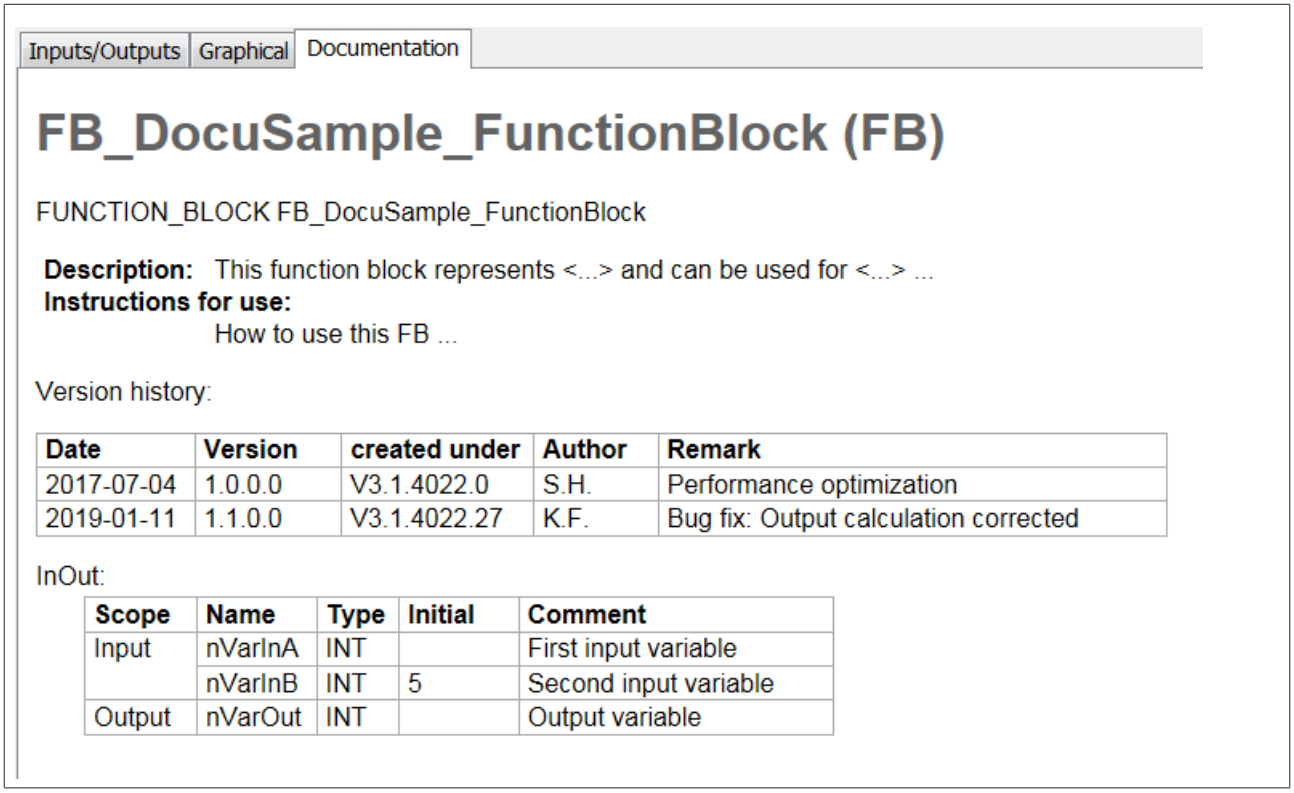
13.6.2.2.3 功能块的文档

下面的示例显示了在功能块注释中使用 reStructuredText 语法的功能块的文档，以及在库管理器中的相应表示。
（在示例项目 [▶ 268]中：A_Samples\FB_DocuSample_FunctionBlock）

代码

```
(*
:Description: This function block represents <...> and can be used for <...> ...
:Instructions for use: How to use this FB ...

Version history:

+-------------+------------+----------------+----------+-----------------------------------------+
|Date         | Version    | created under  | Author   | Remark                                  |
+=============+============+================+==========+=========================================+
|2017-07-04   | 1.0.0.0    | V3.1.4022.0    | S.H.     | Performance optimization                |
+-------------+------------+----------------+----------+-----------------------------------------+
|2019-01-11   | 1.1.0.0    | V3.1.4022.27   | K.F.     | Bug fix: Output calculation corrected   |
+-------------+------------+----------------+----------+-----------------------------------------+
*)

FUNCTION_BLOCK FB_Libdoc_FontStyle
VAR_INPUT
    nVarInA : INT;       //First input variable
    nVarInB : INT :=5    //Second input variable
END_VAR
VAR_OUTPUT
    nVarOut : INT;       //Output variable
END_VAR
```

在库管理器中的表示



13.6.2.3 语法元素

在 reStructuredText 中使用不同的语法元素和结构来标记单词、短语和段落：

- 空行和缩进 [▶ 275]
- 简单和复杂标记 [▶ 277]
- 显式标记块 [▶ 278]

- 指令 [▶ 279]
- 内联标记 [▶ 280]

为了获得用于标记的字符的标准含义，在 reStructuredText 中有 1 个转义机制：

- 转义机制 [▶ 282]

超链接的标识符（引用名称）必须符合某些要求：

- 引用名称 [▶ 282]

13.6.2.3.1 空行和缩进

在 reStructuredText 中，使用以下设计元素可以完成文本主体元素的分离和嵌套：

- 空行 [▶ 275]
- 段落和换行 [▶ 275]
- 缩进 [▶ 275]

空行

- 在 reStructuredText 中，空行可用于分隔段落和其他文本主体元素（请参见示例 [▶ 275]）。
- 多个连续空行等同于 1 个空行。
  （例外：在 码块 [▶ 326] 中会保留所有空行。）
- 如果标记或缩进能明确分隔文本主体元素，则可省略空行。
- 注释中的第 1 行会被视为前面有 1 个空行。注释中的最后一行会被视为后面有 1 个空行。

段落和换行

- 新段落用空行表示（请参见示例 [▶ 275]）。
- 在段落 [▶ 287]内，文本会连续显示，并在达到库管理器的窗口宽度时自动换行。注释中的简单换行不会导致在显示中的换行。为了实现换行，可使用面向行的文本块（行块） [▶ 289]。

缩进

- 缩进可用于识别嵌套内容（请参见示例 [▶ 275]）。
- 缩进小于当前级别的每一行文本都会结束当前级别的缩进。
- 空格或制表符可用于缩进。由于所有缩进都很重要，因此缩进程度必须保持一致性。

示例

文本块（段落）

如果段落或其他结构由多行文本组成，则各行必须左对齐。

```
This is a paragraph. The lines of
this paragraph are aligned at the left.

    This paragraph has problems. The
lines are not left-aligned.
```

This is a paragraph. The lines of this paragraph are aligned at the left.

    This paragraph has problems. The

lines are not left-aligned.

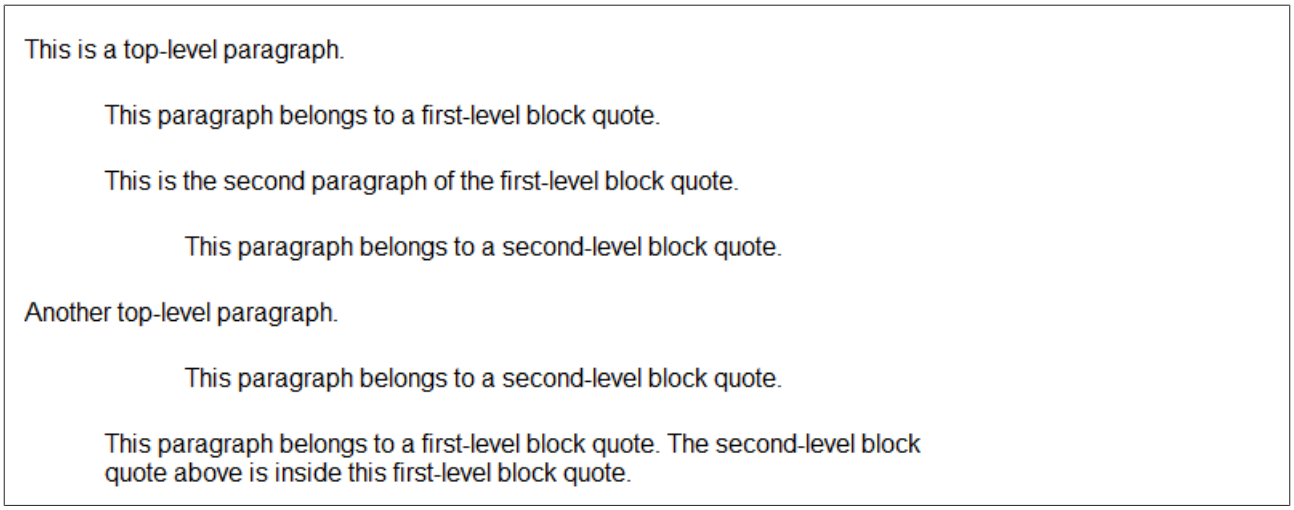
另请参见： 文本块（段落） [▶ 287]

缩进文本块（块引用）

缩进是块引用（缩进文本块）的唯一标记。

- 在 1 个文本块（段落或块引用）和随后的缩进文本块（块引用）之间可以选择插入 1 个空行。下一个文本块的缩进会导致在上一个文本块的末尾处断开。如果在注释中插入 1 个空行，则会发生换行，并且在显示中仍会保留该空行。
- 无论是否在注释中插入 1 个空行，在缩进文本块（块引用）和随后的非缩进文本块（段落或块引用）之间都会自动显示 1 个空行。当前缩进级别结束。
- 在相同缩进级别的 2 个文本块（块引用）之间的注释中必须插入 1 个空行，以分隔这 2 个文本块。

```
This is a top-level paragraph.

    This paragraph belongs to a first-level block quote.

    This is the second paragraph of the first-level block quote.

        This paragraph belongs to a second-level block quote.

Another top-level paragraph.

        This paragraph belongs to a second-level block quote.

    This paragraph belongs to a first-level block quote. The
    second-level block quote above is inside this first-level
    block quote.
```

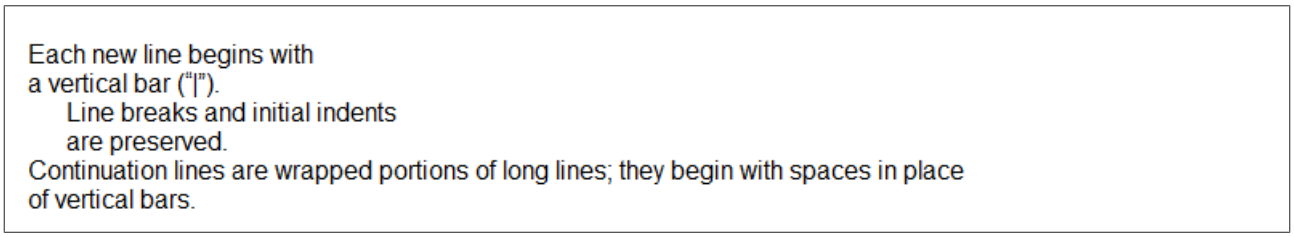


另请参见： 缩进文本块（块引用） [▶ 288]

面向行的文本块（行块）

通过行块可以实现换行。通过缩进法可以实现单行缩进。

```
| Each new line begins with
| a vertical bar ("|").
|     Line breaks and initial indents
|     are preserved.
| Continuation lines are wrapped
  portions of long lines; they begin
  with spaces in place of vertical bars.
```



另请参见： 面向行的文本块（行块） [▶ 289]

简单和复杂标记

一些结构以标记开头。然后，结构的主体必须相对于标记缩进。

对于带有简单标记的结构（无序和有序列表、脚注、引文、超链接目标、指令和注释），主体的缩进程度由与标记在同一行开始的第 1 行文本的位置决定。

```
- This is the first line of a bullet list
  item's paragraph. All lines must align
  relative to the first line. [1]_

    This indented paragraph is interpreted
    as a block quote.

Because it is not sufficiently indented,
this paragraph does not belong to the list
item.

.. [1] Here's a footnote. The second line is aligned
   with the beginning of the footnote label. The ".."
   marker is what determines the indentation.
```

- This is the first line of a bullet list item's paragraph. All lines must align relative to the first line. [1]

  This indented paragraph is interpreted as a block quote.

Because it is not sufficiently indented, this paragraph does not belong to the list item.

[1] Here's a footnote. The second line is aligned with the beginning of the footnote label. The ".." marker is what determines the indentation.

对于带有较复杂标记的结构，标记后第 1 行的缩进通常决定了文本主体的左边缘。如果标记很长，则从下一行的文本主体开始可能会很有用。标记后的行必须缩进至少 1 个空格（最小缩进）。

```
:Field: This field has a short field name, so aligning the field
        body with the first line is feasible.

:This field has a long field name: It would
    be very difficult to align the field body with the left edge
    of the first line.

:This field also has a long field name:
    It would be very difficult to align the field body with the left edge
    of the first line. It may even be preferable not to begin the
    body on the same line as the marker.
```

Field: This field has a short field name, so aligning the field body with the first line is feasible.

This field has a long field name:
It would be very difficult to align the field body with the left edge of the first line.

This field also has a long field name:
It would be very difficult to align the field body with the left edge of the first line. It may even be preferable not to begin the body on the same line as the marker.

另请参见：

- 列表 [▶ 290]
- 显式标记块 [▶ 278]

13.6.2.3.2 简单和较复杂标记

reStructuredText 包括简单和较复杂标记。与简单标记相比，较复杂标记可能包含任何文本。

不带标记：

使用	标记	示例
文本块（段落） [▶ 287]	–	`Paragraph`

简单标记可用于以下结构：

使用	标记	示例
缩进文本块（块引用） [▶ 288]	简单缩进	Paragraph Block quote
面向行的文本块（行块） [▶ 289]	竖条	| Line block
无序枚举列表 [▶ 290]	项目符号	- Item
有序枚举列表 [▶ 292]	枚举符	1. Item
简单表格 [▶ 295]	横条	======== Column 1 ======== Line 1 --------
网格表格 [▶ 297]	横条和竖条	+----------+ | Column 1 | +==========+ | Line 1 | +----------+

较复杂标记可用于以下结构：

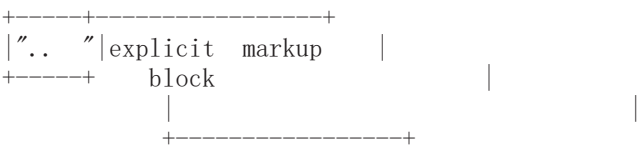
使用	标记	示例
定义列表 [▶ 293]	后面带有冒号和缩进的任何文本	Term 3 : classifier Definition 3
字段列表 [▶ 294]	用冒号括起来的任何文本	:Organization: Beckhoff Automation GmbH & Co. KG

13.6.2.3.3 显式标记块

显式标记块是 reStructuredText 中的文本块，

- 它的第 1 行以 2 个点开头，后面是 1 个空格（“显式标记开始”）
- 它的第 2 行和续行（如果有）相对于第 1 行缩进
- 它在没有缩进的行之前结束

原则



在显式标记块和其他文本主体元素（例如，前一个带有文本的段落）之间需要有 1 个空行。在显式标记块之间可以选择插入 1 个空行。

示例：

```
Paragraph

.. [1] Body elements
.. [2] Body elements
```

使用

显式标记块可用于以下结构：

使用	示例
内部超链接目标 [▶ 302]	`.. _target:`
外部超链接目标 [▶ 304]	`.. _target: http://www.beckhoff.de`
间接超链接目标 [▶ 307]	`.. _target: hyperlink-reference_`
匿名超链接目标 [▶ 309]	`.. __: Anonymous-hyperlink-target`
手动编号的脚注 [▶ 310]	`.. [1] Body elements`
自动编号的脚注 [▶ 312]	`.. [#] Body elements`
自动生成脚注符号 [▶ 315]	`.. [*] Body elements`
引述（目标） [▶ 317]	`.. [CIT] Citation`
替换定义 [▶ 319]	`.. |ref| type: definition`
内部注释 [▶ 328]	`.. comment`

具有附加属性的显式标记块：

- 指令 [▶ 279]

13.6.2.3.4 指令

指令是显式标记块。这些都是影响整个段落的语法元素。

结构

- 指令由 1 个指令标记以及其后的指令块组成。
- 指令标记由 1 个显式标记开始（“.. ”）、指令类型、2 个冒号以及 1 个空格组成。
- 指令块由 3 部分组成，单个指令可以使用这些部分的任意组合：
  - 指令参数
  - 指令选项
  - 指令内容
- 指令参数可以是文件系统路径、标题文本等。
- 使用字段列表可以指定指令选项；可能的字段名称和内容取决于指令类型。
- 指令参数和选项必须组成 1 个以显式标记块的第 1 行或第 2 行开始的相关块。
- 1 个空行表示指令内容块的开始。

原则

在指令和前一个文本主体元素（例如，1 个带有文本的段落）之间需要有 1 个空行。在指令之间可以选择插入 1 个空行。

示例：

```
Paragraph

.. image:: C:\Tc3LibDocImages\SampleLib1\img11.jpg
.. image:: C:\Tc3LibDocImages\SampleLib1\img12.jpg
```

使用

使用指令可以创建以下文本元素：

使用	示例
特定说明 [▶ 321]	`.. note::`
通用说明 [▶ 322]	`.. admonition:: Title`

图像 [▶ 322]	`.. image:: C:\Users\SampleUser\Documents\LibraryDocumentationpicture.png`
	`:height: 100 px`
	`:width: 200 px`

码块 [▶ 326]	`.. code::`
	
	`Code block`

CSV 表格 [▶ 299]	`.. csv-table:: Table`
	`:header: "Column 1", "Column 2"`
	`:widths: 50, 50`
	
	`"Line 1.1","Line 1.2"`

列表表格 [▶ 299]	`.. list-table:: Table`
	`:widths: 50 50`
	`:header-rows: 1`
	
	`* - Column 1`
	`- Column 2`
	`* - Line 1.1`
	`- Line 1.2`

13.6.2.3.5 内联标记

在 reStructuredText 中，内联标记可用于标记文本块中的单词或短语。因此，可以对单词和短语进行格式化或为其提供函数。

内联标记由开始字符和结束字符组成，这 2 个字符会将相关单词或短语围起来。

使用

内联标记可用于以下结构：

- 具有相同的开始字符和结束字符的结构：

使用	示例
突出显示文本（斜体） [▶ 328]	`*emphasized text*`
突出显示文本（粗体） [▶ 329]	`**strong text**`
内联字面量（等宽字体） [▶ 329]	``` ``inline literals`` ```
替换引用 [▶ 318]	`|substitution reference|`

- 具有不同的开始字符和结束字符的结构：

使用	示例

超链接引用（内联、内部、外部、间接）
（超链接 [▶ 300]）
	单词：
	`Target_`
	短语：
	`` `Hyperlink target`_ ``

匿名超链接引用 [▶ 309]	单词：
	`Target__`
	短语：
	`` `Hyperlink target`__ ``

嵌入式 URI 和别名 [▶ 305]	嵌入式 URI： `` `Beckhoff home page <http://www.beckhoff.de>`_ ` 别名： `` `link <Beckhoff home page_>`_ ``
独立超链接 [▶ 305]	`http://www.beckhoff.de` `support@beckhoff.com`

脚注引用
（脚注 [▶ 310]）
	手动编号：
	`[1]_`
	自动编号：
	`[#]_`
	自动符号生成：
	`[*]_`

引述引用 [▶ 317]	`[CIT]_`

内联超链接 [▶ 308]	单词：
	`_Inline-hyperlink-target`
	短语：
	`` _`Inline target` ``

内联标记检测的规则

内联标记不能嵌套。只有满足以下条件，才能识别内联标记的开始字符和结束字符：

1. 开始字符后面不得有空格
2. 结束字符前面不得有空格。
3. 开始字符必须用于开始 1 个文本块，或者必须直接将空格或这些字符之一置于其前：- : / ' " < ([{。
4. 结束字符必须用于结束 1 个文本块，或者必须直接将空格或这些字符之一置于其后：' " . , : ; ! ? -)] } / \ >。
5. 结束字符与开始字符之间必须使用至少 1 个字符隔开。
6. 开始字符和结束字符的前面都不能有反斜线（内联字面量的结束字符除外）。开始字符或结束字符前面的反斜线会禁用标记检测，但内联字面量的结束字符除外。
7. 如果直接将这些字符之一置于开始字符之前：' " ([{ <，则相应字符 ' ")] } > 不能直接跟在后面（不可能："*"text"*"，可能：(*(text)*)）。

字符级内联标记

您可以用反斜线标记单词中的单个字符，这样，任何文本都可以紧跟在内联标记之后。

```
Python ``list``\s use square bracket syntax.
```

Python `lists` use square bracket syntax.

反斜线会从编辑的文档中消失。单词“list”以等宽字体显示，字母“s”以正常文本形式紧随其后，没有空格。

通过使用反斜线和空格，在内联标记前可以添加任何文本。

```
Possible in *re*\ ``Structured``\ *Text*, though not encouraged
```

Possible in *re*`structured`*Text*, though not encouraged.

“re”、“Structured”和“Text”之间的反斜线和空格会从编辑的注释中消失。

不建议在字符级将反斜线用于内联标记。这种用法会导致难以读取未经处理的注释。只有在绝对必要的情况下，才可以慎重地使用该功能。

另请参见： 转义机制 [▶ 282]

13.6.2.3.6 转义机制

一般来说，纯文本文档的字符集是有限的。例如，单个标记字符在书面文本中可能已经具有某种含义，并且它们在文本中出现时并不打算作为标记。为了获得用于标记的字符的标准含义，在 reStructuredText 中有 1 个转义机制。反斜线（“\”）可用作转义字符。

以下规则适用：

- 任何字符（非 URI 环境中的空格除外）后面的反斜线都会使该字符代表其本身，在解释标记时不起任何作用。从输出中会删除反斜线。
- 字面上的反斜线由 2 个连续的反斜线表示。
- 在非 UURI 上下文中，从注释中会删除其后带有空格的反斜线。这样可以在字符级进行内联标记（请参见字符级内联标记 [▶ 281]）。
- 在 2 种上下文中，反斜线没有特殊含义：代码块和内联字面量。在这些上下文中，单个反斜线代表字面反斜线，无需重复（请参见 码块 [▶ 326]）。

13.6.2.3.7 引用名称

reStructuredText 可区分简单的引用名称和短语引用。

简单的引用名称

简单的引用名称是由字母数字字符（字母或数字）以及孤立的（不相邻的）连字符、下划线、点、冒号和加号组成的单个词。不允许使用空格或其他字符。简单的引用名称可用于引用标签和一些超链接：

```
Want to learn about MyFavoriteProgrammingLanguage_?

.. _MyFavoriteProgrammingLanguage: http://www.python.org
```

或

```
Want to learn about My_Favorite_Programming-Language_?

.. _My_Favorite_Programming-Language: http://www.python.org
```

短语引用

“短语引用”是指使用标点符号或其名称为短语（以空格分隔的 2 个或多个单词）的引用名称。短语引用的表达方式是将短语包含在反引号中，并将文本视为引用名称：

```
Want to learn about `my favorite programming language`_?

.. _my favorite programming language: http://www.python.org
```

Want to learn about my favorite programming language?

属性

引用名称没有空格，与字体无关。如果在内部解析引用名称，则以下规则适用：

- 空格标准化（1 个或多个空格和换行符被解释为单个空格）
- 大小写字母标准化（所有字母均转换为小写字母）

例如，以下超链接引用是等效的：

```
- `A HYPERLINK`_
- `a    hyperlink`_
- `A
  Hyperlink`_

.. _A HYPERLINK: https://beckhoff.de/
```

点击其中 1 个超链接引用，就会在库管理器中调用倍福网站。

- A HYPERLINK
- a hyperlink
- A Hyperlink

超链接、脚注和引文的引用名称使用相同的命名空间。引用名称（简单的引用名称）和手工编号的脚注（十进制数）与其他超链接名称一样，都会被输入到相应库对象的同一个数据库中。

这意味着，在 1 个库对象中，通常可以通过脚注引用（[1]_）来引用脚注（定义为 .. [1]），也可以通过简单的超链接引用（1_）来引用。确保与引用名称没有冲突。

另请参见：库对象中的隐式和显式超链接的模糊性 [▶ 301]

13.6.2.4 注释结构

注释可以有不同的结构。

一方面，注释可以被划分为若干部分，而这些部分又可以被进一步分为若干部分和/或包含文本主体元素。各个部分以标题标示。

另一方面，在注释中可以通过转换来分隔段落和文本主体元素。分隔符可用于分隔文本。

13.6.2.4.1 部分

各个部分以标题和编号标示。在注释中，标题文本用“下划线”或者“下划线”和合适的“上划线”标出。

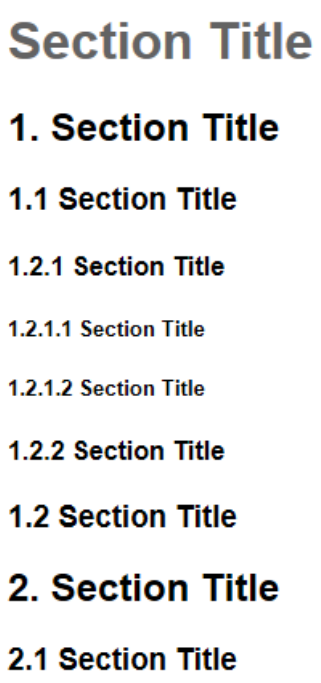
属性

- 下划线/上划线是 1 个单字符，从第 1 列开始，形成 1 行，至少延伸到标题文本的右边缘。如果使用上划线，则长度和字符必须与下划线相匹配。
- 只有下划线或者下划线和上划线的标题文本的格式不相同，即使它们使用相同的字符也是如此。
- 下划线/上划线字符可以是任何非字母数字的 7 位 ASCII 字符。以下字符是部分标题的有效下划线/上划线字符：
  ! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ` { | } ~
  建议使用以下字符：
  “=”、“-”、“*”
- 部分标题级别的数量限制为 5 个。
- 下划线/上划线字符不会决定标题的格式。这是由带有下划线或者下划线和上划线的标题的查找顺序所决定的。遇到的第 1 个样式是最外层的标题，第 2 个样式是副标题，第 3 个是副标题，依此类推。下划线为简单连字符（“-”）的标题，无论位置如何，始终都会显示为灰色加粗的 1 级标题。

```
Section Title
-------------

1. Section Title
****************

1.1 Section Title
=================

-------------------
1.2.1 Section Title
-------------------

=====================
1.2.1.1 Section Title
=====================

=====================
1.2.1.2 Section Title
=====================

-------------------
1.2.2 Section Title
-------------------

1.2 Section Title
=================

2. Section Title
****************

2.1 Section Title
=================
```

```
Section Title
-------------

================
1. Section Title
================

-----------------
1.1 Section Title
-----------------

1.2.1 Section Title
===================

1.2.1.1 Section Title
*********************

1.2.1.2 Section Title
*********************

1.2.2 Section Title
===================

-----------------
1.2 Section Title
-----------------

=================
2. Section Title
=================

-----------------
2.1 Section Title
-----------------
```

这 2 个代码执行都会在库管理器中显示如下内容：



- 不必使用所有的部分标题格式，也不必将特定的样式用于部分标题。不过，在使用部分标题时，注释必须保持一致性：一旦创建标题样式的层次结构，部分就必须使用这种层次结构。
- 每个部分标题都会被自动分配 1 个超链接目标，可以进行引用。超链接的引用名称与部分标题的文本相对应（请参见隐式超链接目标 [▶ 301]）。

示例

下面的示例显示了在 reStructuredText 注释中使用部分和标题的情况。在标题下可以选择插入 1 个空行。相同级别或更高级别的下一个标题之前的所有文本块都包含在 1 个部分（或子部分等）中。
（在示例项目 [▶ 268]中：B_DocuElements\Comment structure\FB_Libdoc_Sections）。

```
(*
Section Title
-------------
This document consists of two main sections and several subsections. All text blocks up to the next
title of the same or higher level
are included in a section (or subsection, etc.).

================
1. Section Title
================
Sections are identified through their titles, which are marked up with adornment:
"underlines" below the title text, or underlines and matching "overlines" above the title.

-----------------
1.1 Section Title
-----------------
A document must be consistent in its use of section titles: once a hierarchy of title styles is esta
blished,
sections must use that hierarchy.
Rather than imposing a fixed number and order of section title adornment styles,
the order enforced will be the order as encountered.
The first style encountered will be an outermost title, the second style will be a subtitle,
the third will be a subsubtitle, and so on.

1.2.1 Section Title
===================
Underline-only adornment styles are distinct
from overline-and-underline styles that use the same character.

1.2.1.1 Section Title
*********************
```

```
An underline/overline is a single repeated punctuation character that begins in column 1
and forms a line extending at least as far as the right edge of the title text.

1.2.1.2 Section Title
*********************

1.2.2 Section Title
===================

-----------------
1.2 Section Title
-----------------

=================
2. Section Title
=================

-----------------
2.1 Section Title
-----------------
*)
```

FB_Libdoc_Sections (FB)

FUNCTION_BLOCK FB_Libdoc_Sections

Section Title

This document consists of two main sections and several subsections. All text blocks up to the next title of the same or higher level are included in a section (or subsection, etc.).

1. Section Title

Sections are identified through their titles, which are marked up with adornment: “underlines” below the title text, or underlines and matching “overlines” above the title.

1.1 Section Title

A document must be consistent in its use of section titles: once a hierarchy of title styles is established, sections must use that hierarchy. Rather than imposing a fixed number and order of section title adornment styles, the order enforced will be the order as encountered. The first style encountered will be an outermost title, the second style will be a subtitle, the third will be a subsubtitle, and so on.

1.2.1 Section Title

Underline-only adornment styles are distinct from overline-and-underline styles that use the same character.

1.2.1.1 Section Title

An underline/overline is a single repeated punctuation character that begins in column 1 and forms a line extending at least as far as the right edge of the title text.

1.2.1.2 Section Title

1.2.2 Section Title

1.2 Section Title

2. Section Title

2.1 Section Title

InOut:

Scope	Name	Type
Input	nVarA	INT
	nVarB	INT
Output	nSum	INT

13.6.2.4.2 转换

段落和文本主体元素之间的分隔符（而非副标题）可用于标记文本分区或表示主题或重点的变化。使用分隔符对段落和文本主体元素进行分隔，这被称为转换。

属性

- 注释或部分不应以转换为开始或结束。此外，2 个转换不应直接在彼此下方。

- 转换标记的语法是 1 条带有 4 个或更多字符的水平线。有效分隔符为所有非字母数字的 7 位 ASCII 字符（例如，“-”、“+”、“*”、“=”）。转换标记的前面和后面必须插入 1 个空行。
- 与部分标题不同，没有创建转换标记的层次结构。转换标记中的差异没有影响。建议使用统一的样式。

示例

（在示例项目 [▶ 268]中：B_DocuElements\Comment structure\FB_Libdoc_Transitions）

```
(*
A transition marker is a horizontal line
of 4 or more repeated punctuation characters.

------------

A transition should not begin or end a section or document,
nor should two transitions be immediately adjacent.
*)
```

A transition marker is a horizontal line of 4 or more repeated punctuation characters.

A transition should not begin or end a section or document, nor should two transitions be immediately adjacent.

13.6.2.5 文本块

在 reStructuredText 注释中，可区分以下类型的文本块：

- 文本块（段落） [▶ 287]
- 缩进文本块（块引用） [▶ 288]
- 面向行的文本块（行块） [▶ 289]

13.6.2.5.1 文本块（段落）

在 reStructuredText 中，文本左对齐且没有显式标记的简单文本块被称为段落。

属性

- 段落之间以及段落与其他文本主体元素之间以空行分隔。
- 段落可包含内联标记。

原则

```
+----------------------+
|paragraph             |
+----------------------+
+----------------------+
|paragraph             |
+----------------------+
```

示例

下面的示例显示了 reStructuredText 注释中的 2 个段落。在段落内，文本会连续显示，并在达到窗口宽度时自动换行。
（在示例项目 [▶ 268]中：B_DocuElements\Text blocks\FB_Libdoc_Paragraph）

```
(*
This is a paragraph.

Paragraphs line up at their left edges, and are normally separated
by blank lines.
*)
```

This is a paragraph.

Paragraphs line up at their left edges, and are normally separated by blank lines.

另请参见：

- 空行和缩进 [▶ 275]
- 缩进文本块（块引用） [▶ 288]
- 面向行的文本块（行块） [▶ 289]

13.6.2.5.2 缩进文本块（块引用）

在 reStructuredText 中，相对于前面文本缩进且没有显式标记的文本块被称为块引用。

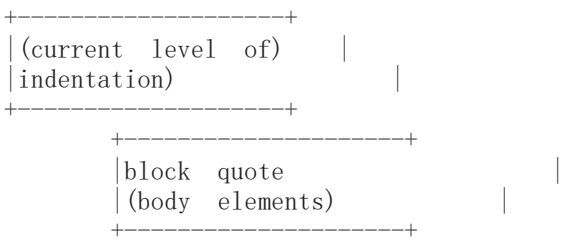
属性

- 最小缩进为 1 个空格。
- 所有标记处理（适用于文本主体元素和内联标记）会在块引用内继续进行。

`Unindented paragraph.`
 	
`  Block quote 1.`
 	
`  Block quote 2.`
 	
`  Block quote 3.`
	Unindented paragraph.
	
	Block quote 1.
	
	Block quote 2.
	
	Block quote 3.

`- List item.`
 	
`  Block quote.`
	■ List item.
	
	Block quote.

原则



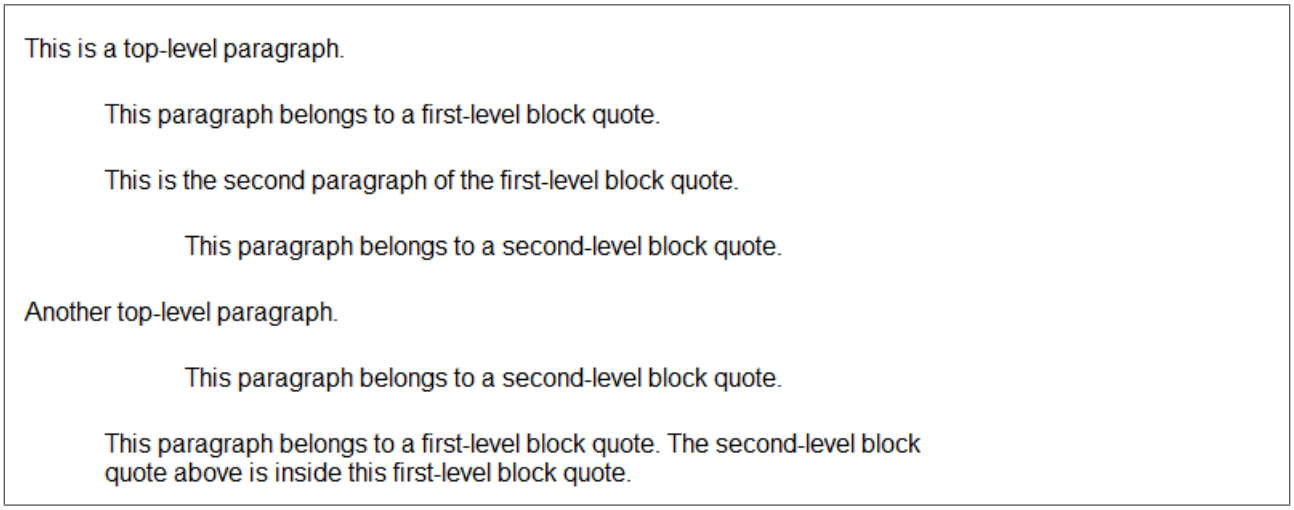
示例

下面的示例显示了嵌套的块引用。
（在示例项目 [▶ 268]中：B_DocuElements\Text blocks\FB_Libdoc_BlockQuote）

- 在 1 个文本块（段落或块引用）和随后的缩进文本块（块引用）之间可以选择插入 1 个空行。下一个文本块的缩进会导致在上一个文本块的末尾处断开。如果在注释中插入 1 个空行，则会发生换行，并且在显示中仍会保留该空行。
- 无论是否在注释中插入 1 个空行，在缩进文本块（块引用）和随后的非缩进文本块（段落或块引用）之间都会自动显示 1 个空行。当前缩进级别结束。
- 在相同缩进级别的 2 个文本块（块引用）之间的注释中必须插入 1 个空行，以分隔这 2 个文本块。

```
(*
This is a top-level paragraph.

    This paragraph belongs to a first-level block quote.

    This is the second paragraph of the first-level block quote.
```

```
        This paragraph belongs to a second-level block quote.

Another top-level paragraph.

        This paragraph belongs to a second-level block quote.

    This paragraph belongs to a first-level block quote. The
    second-level block quote above is inside this first-level
    block quote.
*)
```



另请参见：

- 空行和缩进 [▶ 275]
- 面向行的文本块（行块） [▶ 289]

13.6.2.5.3 面向行的文本块（行块）

在 reStructuredText 中，每行以前缀“|”开头且后跟空格的文本块被称为行块。每个竖条前缀表示 1 个新行。

行块适用于行布局非常重要的结构。此外，行块还可用于在库管理器显示中强制空行或分隔文本主体元素。

属性

- 缩进可能会导致嵌套结构。
- 长行的换行部分是续行。它们从空格（而不是竖条）开始。续行的左边缘必须缩进，但不必与上方文本的左边缘对齐。
- 对内联标记进行处理。
- 行块的前面和后面必须插入 1 个空行。

原则

```
+----+------------------+
|"|  "|line              |
+----|continuation  line    |
          +------------------+
```

示例

下面的示例显示了行块的嵌套和续行的结构。
（在示例项目 [▶ 268]中：B_DocuElements\Text blocks\FB_Libdoc_LineBlock）

```
(*
| Each new line begins with
| a vertical bar ("|").
|     Line breaks and initial indents
|     are preserved.
| Continuation lines are wrapped
```

```
  portions of long lines; they begin
  with spaces in place of vertical bars.
*)
```

Each new line begins with
a vertical bar ("|").
    Line breaks and initial indents
    are preserved.
Continuation lines are wrapped portions of long lines; they begin with spaces in place
of vertical bars.

如果行块的前面有另一个文本主体元素，则整个行块都可以缩进（最小缩进为 1 个空格）。

```
(*
Line blocks are useful where the structure of lines is significant.

| Each new line begins with
| a vertical bar ("|").
|     Line breaks and initial indents
|     are preserved.
| Continuation lines are wrapped
  portions of long lines; they begin
  with spaces in place of vertical bars.

    | Each new line begins with
    | a vertical bar ("|").
    |     Line breaks and initial indents
    |     are preserved.
    | Continuation lines are wrapped
      portions of long lines; they begin
      with spaces in place of vertical bars.
*)
```

Line blocks are useful where the structure of lines is significant.

Each new line begins with
a vertical bar ("|").
    Line breaks and initial indents
    are preserved.
Continuation lines are wrapped portions of long lines; they begin with spaces
in place of vertical bars.

        Each new line begins with
        a vertical bar ("|").
            Line breaks and initial indents
            are preserved.
        Continuation lines are wrapped portions of long lines; they
        begin with spaces in place of vertical bars.

13.6.2.6 列表

在 reStructuredText 中，可区分以下列表类型：

- 无序枚举列表 [▶ 290]
- 有序（编号）枚举列表 [▶ 292]
- 定义列表 [▶ 293]
- 字段列表 [▶ 294]

13.6.2.6.1 无序枚举列表

无序列表由以“-”、“+”或“*”开头且后面有 1 个空格的文本块（项目符号）组成。

属性

- 无论项目符号类型如何，方框都会在库管理器中显示为项目符号。
- 在注释的开头不得使用星号“*”。
- 列表项目左对齐。
- 项目符号的缩进决定了条目是主列表还是子列表。子列表的项目符号字符从主列表中列表项目的文本级别开始。
- 在项目符号和列表项目第 1 行文本之间有 1 个空格。
- 如果列表项目的文本扩展到多行，则续行的文本必须从第 1 行文本的级别开始。
- 在 1 个级别的第 1 个列表项目的前面必须插入 1 个空行。在单个列表项目之间以及在 1 个级别的最后一个列表项目的后面可以选择插入空行。
- 在库管理器中，主列表和子列表会显示为 1 个不带空行的块。

原则

```
+----+-----------------+
|"-  "|list  item               |
|        |(body  elements)    |
+----+-----------------|
```

示例

下面的示例显示了 1 个带有 2 个级别的无序枚举列表。子列表中的列表项目的前面有 2 个空格。（在示例项目 [▶ 268]中：B_DocuElements\Lists\FB_Libdoc_UnorderedEnumerationList）

```
(*
This paragraph is not part of the list.

- This is the first bullet list item.  The blank line above the
  first list item is required; blank lines between list items
  (such as below this paragraph) are optional.

  - This is a sublist. The bullet lines up with the left edge
    of the text blocks above. A sublist is a new list so
    requires a blank line above. A blank line below is optional.

  - This is a sublist. The bullet lines up with the left edge
    of the text blocks above. A sublist is a new list so
    requires a blank line above. A blank line below is optional.

- This is the second item of the main list.

- This is the third item of the main list.

This paragraph is not part of the list.
*)
```

This paragraph is not part of the list.

- This is the first bullet list item. The blank line above the first list item is required; blank lines between list items (such as below this paragraph) are optional.
  - This is a sublist. The bullet lines up with the left edge of the text blocks above. A sublist is a new list so requires a blank line above. A blank line below is optional.
  - This is a sublist. The bullet lines up with the left edge of the text blocks above. A sublist is a new list so requires a blank line above. A blank line below is optional.
- This is the second item of the main list.
- This is the third item of the main list.

This paragraph is not part of the list.

13.6.2.6.2 有序（编号）枚举列表

有序（编号）枚举列表由以数字或字母开头的文本块（项目符号）和后面有 1 个空格的分隔符组成。列表项目按照一定的顺序排列。

属性

- 列表项目左对齐。
- 枚举符缩进决定了项目是主列表还是子列表。子列表的枚举符从主列表中列表项目的文本级别开始。
- 在分隔符和文本之间有 1 个空格。
- 如果列表项目的文本扩展到多行，则续行的文本必须从第 1 行文本的级别开始。
- 在 1 个级别的第 1 个列表项目的前面必须插入 1 个空行。在单个列表项目之间以及在 1 个级别的最后一个列表项目的后面可以选择插入空行。
- 在库管理器中，主列表和子列表会显示为 1 个不带空行的块。
- 可识别以下枚举符：
  - 阿拉伯数字：1、2、3、…… （无上限值）
  - 大写字母：A、B、C、……、Z
  - 小写字母：a、b、c、……、z
- 此外，自动枚举符“#”也可用作为列表自动编号的枚举符。自动编号的列表可从 1 个显式枚举（阿拉伯数字）开始，该枚举决定了序列。全自动编号的列表使用阿拉伯数字并从 1 开始。
- 可识别以下分隔符，但在库管理器中始终会显示 1 个点：
  - 后面带点：“1.”、“A.”、“a.”
  - 用括号括起来：“(1)”、“(A)”、“(a)”
  - 后面带右括号：“1)”、“A)”、“a)”
- 在以下情况下，总是会启动 1 个新列表
  - 发现枚举符的格式和序列类型与当前列表不一致（例如，“1.” “(a)”会创建 2 个独立的列表）；
  - 枚举不在序列中（例如，“1.” 和“3.” 会创建 2 个独立的列表）。
- 对每个枚举的第 2 行进行有效性检查。这是为了防止将以与枚举符相同的文本开头的普通段落误解为列表项目。例如，该文本被解析为 1 个普通段落：

```
A. Einstein was a really
smart dude
```

- 但是，如果段落只有 1 行，则无法避免歧义。该文本被解析为列表中的 1 个枚举：

```
A. Einstein was a really smart dude
```

- 如果单行段落以与枚举符相同的文本开头（“A.”、“1.”）、“(b)”等），则第 1 个字符的前面必须有反斜线，这样该行才能被解释为普通段落：

```
\A. Einstein was a really smart dude
```

原则

```
+-----+-----------------+
|"1.  "|list  item              |
|          |(body  elements)    |
+-----+-----------------|
```

示例

下面的示例显示了 1 个带有 2 个级别的有序（编号）枚举列表。子列表中的列表项目的前面有 3 个空格。（在示例项目 [▶ 268]中：B_DocuElements\Lists\FB_Libdoc_OrderedNumberedEnumerationList）

```
(*
1) This is the first item in the list.

   (a) This is the first subitem (1a).
   (b) This is the second subitem (1b).

2) This is the second item in the list.
*)
```

```
   (a) This is the first subitem (2a).
   (b) This is the second subitem (2b).

#) This item is auto-enumerated.
*)
```

1. This is the first item in the list.
    a. This is the first subitem (1a).
    b. This is the second subitem (1b).
2. This is the second item in the list.
    a. This is the first subitem (2a).
    b. This is the second subitem (2b).
3. This item is auto-enumerated.

13.6.2.6.3 定义列表

定义列表描述了 1 个定义的列表。定义列表中的每个条目都包含 1 个术语、1 个定义和可选的分类符。

通过不同的方式可以使用定义列表：

- 用作字典或术语表。术语是单词本身，分类符可用于识别术语的用途（名词、动词等），后面是定义。
- 用于描述程序变量。术语是变量名。分类符可用于指定变量的类型（字符串、整数等），而定义描述了在程序中如何使用变量。

属性

- 术语是 1 个简单的单行词或短语。
- 定义是 1 个相对于术语缩进（最小缩进为 1 个空格）的块，可以包含多个段落和其他主体元素。
- 可选分类符可以在同一行中跟在术语后面，每个分类符都在“:”之后（空格、冒号、空格）。

原则

```
+--------------------------+
|term[″  :  ″classifier]*        |
+----+---------------------+
          |definition                    |
          |(body elements)            |
          +---------------------+
```

示例

该示例显示了带有术语、分类符和定义的定义列表。
（示例项目 [▶ 268]中：B_DocuElements\Lists\FB_Libdoc_DefinitionList）。

- 在定义列表中还使用了其他文本主体元素（段落、无序列表项目）。
- 在术语行和定义块之间不能有空行（这可用于区别定义列表与块引用）。
- 在定义列表的第 1 个列表项目之前以及最后一个列表项目之后必须有空行，不过，也可以在它们之间插入空行。

```
(*
Term 1
    Definition 1.

Term 2
    Definition 2, paragraph 1.

    Definition 2, paragraph 2.

Term 3 : classifier
    Definition 3.

Term 4 : classifier one : classifier two
    - Item 1
    - Item 2
*)
```

Term 1
Definition 1.

Term 2
Definition 2, paragraph 1.

Definition 2, paragraph 2.

Term 3 : *classifier*
Definition 3.

Term 4 : *classifier one* : *classifier two*
- Item 1
- Item 2

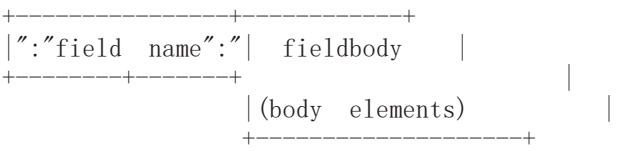
13.6.2.6.4 字段列表

字段列表是将字段名称分配给字段文本，可用于双列表格类型的结构。

属性

- 字段名称可以由任何字符和多个单词组成。如果字段名称中的冒号后面有空格，则必须在其前面加上反斜线。
- 在字段名称中对内联标记进行处理。
- 字段名称与冒号前缀和后缀构成字段标记。字段标记的后面是空格和字段主体。
- 字段主体可以包含多个相对于字段标记缩进的主体元素。字段标记后的第 1 行决定了字段主体的缩进（最小缩进为 1 个空格）。

原则



示例

（在示例项目 [▶ 268]中：B_DocuElements\Lists\FB_Libdoc_FieldList）

```
(*
:Organization: Beckhoff Automation GmbH & Co. KG
:Contact: info@beckhoff.de
:Address: | Hülshorstweg 20
          | 33415 Verl
          | Germany
:Authors: - Me
          - Myself
          - I
:Version: 1.0
:Status: released
:Date: 2017-12-07

:Copyright:
 © Beckhoff Automation GmbH & Co. KG, Germany.
 The reproduction, distribution and utilization of this document as well as
 the communication of its contents to others without express authorization are prohibited.
 Offenders will be held liable for the payment of damages.
 All rights reserved in the event of the grant of a patent, utility model or design.
:Abstract: topic
:Indentation:
 Since the field marker may be quite long, the second
 and subsequent lines of the field body do not have to line up
 with the first line, but they must be indented relative to the
 field name marker, and they must line up with each other.
:LongLongFieldname: Since the field marker is quite long,
                    the field body is shown in the line after the marker.
```

```
:Parameter nIn: Integer
:Parameter nVarB: Integer
:*Parameter nVarC*: Integer
:``Parameter nVarD``: Integer
*)
```

Organization: Beckhoff Automation GmbH & Co. KG
Contact: info@beckhoff.de
Address: Hülshorstweg 20
33415 Verl
Germany

Authors:
- Me
- Myself
- I

Version: 1.0
Status: released
Date: 2017-12-07
Copyright: © Beckhoff Automation GmbH & Co. KG, Germany. The reproduction, distribution and utilization of this document as well as the communication of its contents to others without express authorization are prohibited. Offenders will be held liable for the payment of damages. All rights reserved in the event of the grant of a patent, utility model or design.
Abstract: topic
Indentation: Since the field marker may be quite long, the second and subsequent lines of the field body do not have to line up with the first line, but they must be indented relative to the field name marker, and they must line up with each other.
LongLongFieldname: Since the field marker is quite long, the field body is shown in the line after the marker.
Parameter nIn: Integer
Parameter nVarB: Integer
***Parameter nVarC*:** Integer
``Parameter nVarD``: Integer

13.6.2.7 表格

reStructuredText 提供了 2 种为表格单元格划定界限的语法：

- 简单表格 [▶ 295]
- 网格表格 [▶ 297]

指令为表格生成提供了更多可能性：

- CSV 表格 [▶ 299]
- 列表表格 [▶ 299]

13.6.2.7.1 简单表格

简单表格为简单数据集提供了简洁、易懂但有限的面向行的表格显示。

有关更完整的表格说明，请参阅网格表格 [▶ 297]。

属性

- 尽管在大多数单元格中都可以显示任何文本主体元素，但是单元格内容通常是单独的段落。

- 简单表格允许多线行（除第 1 列外的所有行）和连接列，但不允许连接行。
- 简单表格使用由字符“=”和“-”组成的水平边框进行描述。等号（“=”）可用于表格的上下边框，以分隔可选标题和表格主体。通过为链接列加下划线，连字符（“-”）可用于指明单行中的列间距，也可用于明确和/或直观地分隔行。
- 与其他文本主体元素一样，在表格的前面和后面也需要空行。如要分隔 2 个连续的表格，务必插入 1 个带有文本的段落或 1 个带有竖条的段落。从输出中会删除后者。
- 表格左边框应与前面文本块的左边缘对齐。如果表格有缩进，则将其视为块引用的一部分。最小缩进为 1 个空格。

示例

（在示例项目 [▶ 268]中：B_DocuElements\Tables\FB_Libdoc_SimpleTables）

简单表格从等号的上边界开始，在每个列边界中有 1 个或多个空格（建议使用 2 个或多个空格）。无论表格宽度如何，上边缘必须完全覆盖所有表格列。

表格中必须至少有 2 列（以便区别它们与部分标题）。上边框的下面可以是标题。最后一个可选标题用“=”加下划线，在列边缘处同样也使用空格。标题行的分隔符下方不得有空行。表格的下边界由“=”组成，包括列边界中的空格。

例如，这里有 1 个真值表，它是 1 个 3 列表格，带有 1 行标题和 4 行正文：

```
(*
=====  =====  =======
  A      B    A and B
=====  =====  =======
False  False  False
True   False  False
False  True   False
True   True   True
=====  =====  =======
*)
```

A	B	A and B
False	False	False
True	False	False
False	True	False
True	True	True

连字符“-”行可用于链接相邻列。行必须覆盖所有列，并根据定义的列边界定向。带连字符的文本行不得包含任何其他文本。连字符适用于紧接其上的 1 行。例如，这里有 1 个表格，标题中有连接的列：

```
(*
=====  =====  ======
   Inputs      Output
-----------  ------
  A      B    A or B
=====  =====  ======
False  False  False
True   False  True
False  True   True
True   True   True
=====  =====  ======
*)
```

Inputs		Output
A	B	A or B
False	False	False
True	False	True
False	True	True
True	True	True

除非单元格已跨列合并，否则每行文本在列边界处必须包含空格。

除非第 1 列中有 1 个空单元格，否则每行文本都会开始 1 个新的表格行。在这种情况下，文本行被解释为续行：

```
(*
=====  =====  ======
   Inputs      Output
-----------  ------
  A      B    A or B
=====  =====  ======
False  False  False
       False  True
False  True   True
True   True   True
=====  =====  ======
*)
```

Inputs		Output
A	B	A or B
False	False False	False True
False	True	True
True	True	True

如要在第 1 列中没有文本的简单表格中开始新行，可使用

- 1 个空注释（“..”），它不会在输出中显示，或
- 1 个反斜线（“\”），其后带有空格

简单表格中允许有空行。它们的解释取决于上下文。表格行之间的空行将被忽略。多线行中的空行可以分隔单元格中的段落或其他主体元素。

右列没有限制；文本可以超出表格边框（如表格边框所示）。不过，建议边框要足够长，以便包含整个文本。

下面的示例展示了以下内容：

- 续行（表格行 2 由 2 个文本行组成，表格行 3 由 3 个文本行组成，表格行 4 由 4 个文本行组成）
- 用于分隔段落的空行（第 3 行，第 2 列）
- 超出表格右边缘的文本
- 在第 1 列中不包含任何文本的新行（第 4 行）

```
(*
=====  =====
Col 1  Col 2
=====  =====
1      Second column of row 1.
2      Second column of row 2.
       Second line of paragraph.
3      Second column of row 2.

       Second line of paragraph
4      - Second column of row 3.

       - Second item in bullet
         list (row 4, column 2).
\      Row 5; column 1 will be empty.
=====  =====
*)
```

Col 1	Col 2
1	Second column of row 1.
2	Second column of row 2. Second line of paragraph.
3	Second column of row 3. Second line of paragraph
4	■ Second column of row 4. ■ Second item in bullet list (row 4, column 2).
	Row 5; column 1 will be empty.

13.6.2.7.2 网格表格

网格表格可提供完整的表格显示。它们允许任何单元格内容（文本主体元素）以及行和列的链接。不过，创建网格表格可能比较困难，对于简单的数据记录更是如此。

有关较简单（但有限）的表格表示法，请参见“简单表格 [▶ 295]”部分。

属性

- 网格表格使用由字符“-”、“=”、“|”和“+”组成的可视化网格进行描述。连字符（“-”）可用于横线（行分隔符）。竖条（“|”）可用于竖线（列分隔符）。加号（“+”）可用于横线和竖线的交叉。等号（“=”）可用于分隔标题和表格主体。
- 与其他文本主体元素一样，在表格的前面和后面也需要空行。如要分隔 2 个连续的表格，务必插入 1 个带有文本的段落或 1 个带有竖条的段落。从输出中会删除后者。
- 表格左边框应与前面文本块的左边缘对齐。如果表格有缩进，则将其视为块引用的一部分。最小缩进为 1 个空格。

示例

下面的示例展示了以下内容：

- 合并的列和行
- 单元格中的不同文本主体元素

（在示例项目 [▶ 268]中：B_DocuElements\Tables\FB_Libdoc_GridTable）

```
(*
+------------------------+------------+----------+----------+
| Header row, column 1   | Header 2   | Header 3 | Header 4 |
| (header rows optional) |            |          |          |
+========================+============+==========+==========+
| body row 1, column 1   | column 2   | column 3 | column 4 |
+------------------------+------------+----------+----------+
| body row 2             | Cells may span columns.          |
+------------------------+------------+---------------------+
| body row 3             | Cells may  | - Table cells       |
+------------------------+ span rows. | - contain           |
| body row 4             |            | - body elements.    |
+------------------------+------------+---------------------+
*)
```

Header row, column 1 (header rows optional)	Header 2	Header 3	Header 4
body row 1, column 1	column 2	column 3	column 4
body row 2	Cells may span columns.		
body row 3	Cells may span rows.	■ Table cells ■ contain ■ body elements.	
body row 4			

网格字符与单元格文本之间不必要的交互示例

下面的表格在第 2 行中包含 1 个单元格，其范围涵盖第 2 列至第 4 列：

```
+--------------+----------+-----------+-----------+
| row 1, col 1 | column 2 | column 3  | column 4  |
+--------------+----------+-----------+-----------+
| row 2        |                                  |
+--------------+----------+-----------+-----------+
| row 3        |          |           |           |
+--------------+----------+-----------+-----------+
```

在该单元格的文本中使用竖条时，如果不小心将其与列边界对齐，则可能会产生非预期效应：

```
+--------------+----------+-----------+-----------+
| row 1, col 1 | column 2 | column 3  | column 4  |
+--------------+----------+-----------+-----------+
| row 2        | Use the command ``ls | more``.   |
+--------------+----------+-----------+-----------+
| row 3        |          |           |           |
+--------------+----------+-----------+-----------+
```

一些解决方案可能会打破网格结构的连续性。

1 种方法是在单元格文本前额外添加 1 个空格来移动文本。在输出中会删除空格。

```
+--------------+----------+-----------+-----------+
| row 1, col 1 | column 2 | column 3  | column 4  |
+--------------+----------+-----------+-----------+
| row 2        |  Use the command ``ls | more``.  |
+--------------+----------+-----------+-----------+
| row 3        |          |           |           |
+--------------+----------+-----------+-----------+
```

另一种选项是在第 2 行中额外添加 1 行。从输出中会删除空行。

```
+--------------+----------+-----------+-----------+
| row 1, col 1 | column 2 | column 3  | column 4  |
+--------------+----------+-----------+-----------+
| row 2        | Use the command ``ls | more``.   |
|              |                                  |
+--------------+----------+-----------+-----------+
| row 3        |          |           |           |
+--------------+----------+-----------+-----------+
```

13.6.2.7.3 CSV 表格

csv table 指令可用于根据逗号分隔值（CSV）创建表格。

描述	指令选择由 1 个显式标记开始（“.. ”），以及后面的指令类型（csv table）和 2 个冒号组成。 （另请参见：指令 [▶ 279]）
原则	.. csv-table::
属性	• 在指令和前一个文本主体元素（例如，1 个带有文本的段落）之间需要有 1 个空行。

选项

行块可以选择性地包含表格选项的平面列表。可识别以下选项：

宽度：整数 [integer...]	列宽的加权 用逗号或空格分隔的相对列宽列表。默认情况下，各列具有相同的宽度（100%/#columns）。
标题行：整数	标题表格 用作表格标题的行数。默认值为 0。
存根列：整数	标题列表格 用作表格标题的表格列数。默认值为 0。
标题：CSV 数据	表标题的附加数据，与每个标题行无关且添加在每个标题行之前。

示例

下面的示例显示了 1 个包含 4 列和 2 行的标题表格。
（在示例项目 [▶ 268]中：B_DocuElements\Tables\FB_Libdoc_CSVTable）

```
(*
.. csv-table:: Property list
   :header: "Items", "Property 1", "Property 2", "Property 3"
   :widths: 10, 15, 15, 15

   "Item 1", 1.67, angular, red
   "Item 2", "not specified", round, blue
*)
```

或

```
(*
.. csv-table:: Property list
   :header-rows: 1
   :widths: 10, 15, 15, 15

   "Items", "Property 1", "Property 2", "Property 3"
   "Item 1", 1.67, angular, red
   "Item 2", "not specified", round, blue
*)
```

Property list

Items	Property 1	Property 2	Property 3
Item 1	1.67	angular	red
Item 2	not specified	round	blue

13.6.2.7.4 列表表格

list table 指令允许您以 2 级枚举列表的形式创建数据表格。

描述	指令选择由 1 个显式标记开始（“.. ”），以及后面的指令类型（list-table）和 2 个冒号组成。 （另请参见：指令 [▶ 279]）
原则	.. list-table::
属性	• 枚举列表的子列表必须包含相同数量的列表项目。 • 在指令和前一个文本主体元素（例如，1 个带有文本的段落）之间需要有 1 个空行。

选项

列表表格行块可以选择性地包含表格选项的平面列表。可识别以下选项：

宽度：整数 [integer...]	列宽的加权 用逗号或空格分隔的相对列宽列表。默认情况下，各列具有相同的宽度（100%/#columns）。
标题行：整数	标题表格 用作表格标题的行数。默认值为 0。
存根列：整数	标题列表格 用作表格标题的列数。默认值为 0。

示例

（在示例项目 [▶ 268]中：B_DocuElements\Tables\FB_Libdoc_ListTable）

```
(*
.. list-table:: Property list
    :widths: 50 50 50
    :header-rows: 1

    * - Items
      - Property 1
      - Property 2
    * - Item 1
      - angular
      - red
    * - Item 2
      - round
      - blue
*)
```

Property list

Items	Property 1	Property 2
Item 1	angular	red
Item 2	round	blue

13.6.2.8 超链接

超链接指向注释内部或外部的另一个位置。它们通常由 2 部分组成：超链接引用（来源）和超链接目标。如果文本主体中包含源链接，则注释中的其他地方也必须有目标链接（例外：分离式超链接 [▶ 305]）。

reStructuredText 可区分显式、隐式和内联式超链接目标。

此外，还可以设置指向该库中包含的另一个库对象的文档的链接（请参见 链接到另一个对象 [▶ 318]）。

显式超链接目标

显式超链接目标指的是功能块文档中的 1 个部分或 1 个外部页面，这些显式超链接目标可以链接在一起。它们可以具名或匿名。与具名超链接不同，匿名超链接不使用引用名称来匹配引用及其目标（请参见匿名超链接 [▶ 309]）。

- 内部超链接 [▶ 302]（链接到注释或功能块文档中的某个位置）
- 外部超链接 [▶ 303]（链接到网站或电子邮件程序）
  - 嵌入式 URI 和别名 [▶ 305]

    - 独立超链接 [▶ 305]
- 间接超链接 [▶ 306]（链接显式超链接目标）

内联超链接目标

内联超链接目标指的是注释或功能块文档的当前文本。

- 内联超链接 [▶ 308]

隐式超链接目标

隐式超链接目标由部分标题、脚注和引文生成。与显式超链接目标不同，部分标题、脚注和引文会自动生成 1 个指向自身的超链接目标；它们的定义中不包含链接块。引用名称与部分标题或脚注或引用标签相对应。否则，隐式超链接的行为与显式超链接相同。

- 脚注 [▶ 310]
- 引文 [▶ 317]
- 部分 [▶ 283]

库对象中的隐式和显式超链接的模糊性

- 具有相同引用名称的显式和隐式超链接目标：超链接不起作用。

错误消息：重复的目标名称不能用作唯一引用：“1”（“倍福”）。

示例：

```
This in an explicit internal hyperlink reference: 1_

For more information see [1]_

--------

.. _1:

This is an explicit internal hyperlink target.

.. [1] Footnote
```

或

```
See Beckhoff_.

This is an explicit hyperlink to Beckhoff_.

--------

.. _Beckhoff: http:\\www.beckhoff.de

Beckhoff
========
```

- 重复的隐式超链接目标：超链接不起作用。

错误消息：重复的目标名称不能用作唯一引用：“chapter a”。

示例：

```
Chapter 1
=========

Chapter a
*********

Chapter 2
=========

Chapter a
*********

--------

See `Chapter a`_
```

- 重复的显式超链接目标：超链接不起作用。例外：重复的外部超链接目标（相同的引用名称和引用 URI）没有冲突，不会被删除。

错误消息：重复的目标名称不能用作唯一引用：“1”。

示例：

```
This in an explicit internal hyperlink reference: 1_

This in another explicit internal hyperlink reference: 1_

--------

.. _1:

This is an explicit internal hyperlink target.

.. _1:

This is another explicit internal hyperlink target.
```

另请参见： 引用名称 [▶ 282]

13.6.2.8.1 内部超链接

内部超链接可以将注释中的 1 个位置与另一个位置相连接。超链接目标始终指向随后的文本主体元素。

超链接引用

描述	超链接引用由引用名称以及后面的下划线组成： reference-name_ 在反引号中必须指定短语引用： `reference name`_ （另请参见：引用名称 [▶ 282]）
开始字符和结束字符	• 无开始字符，结束字符 = “_” • 开始字符 = “`”，结束字符 = “`_”（短语引用） （另请参见：内联标记 [▶ 280]）

超链接目标

描述	超链接目标由 1 个显式标记开始（“..”）、下划线、引用名称和冒号组成： .. _reference-name: 超链接目标中的短语引用可以选择性地包含在反引号中： .. _`reference name`: .. _reference name: （另请参见：显式标记块 [▶ 278]，引用名称 [▶ 282]）
原则	+-----+----------------+ |".. "|"_"name":" | +-----+ | | | +----------------+
属性	• 内部超链接目标有 1 个空链接块。 • 在显式标记块和随后的文本主体元素之间需要有 1 个空行。在显式标记块之间可以选择插入空行。

示例

（在示例项目 [▶ 268]中：B_DocuElements\Hyperlinks\FB_Libdoc_InternalHyperlinks）

简单的内部超链接目标

点击超链接引用 target_，可显示 .. _target: 下的文本主体元素。

```
(*
Clicking on this internal hyperlink will take us to the target_
below.

.. _target:

The hyperlink targets above point to this paragraph.
*)
```

Clicking on this internal hyperlink will take us to the target below.

The hyperlink targets above point to this paragraph.

内部超链接目标的嵌套

如果内部超链接目标“嵌套”在缩进文本块中，则超链接也可以正常工作。例如，可以将超链接目标设置为单个列表元素（第 1 个元素除外，因为之前的内部超链接目标适用于整个列表）：

```
(*
Clicking on this internal hyperlink will take us to the `third item`_ of the bullet list.

* First list item
* Second list item

  .. _third item:

* Third list item, with hyperlink target.
*)
```

Clicking on this internal hyperlink will take us to the third item of the bullet list.

- First list item
- Second list item
- Third list item, with hyperlink target.

内部超链接目标的串联

内部超链接目标可以“串联”。然后，几个相邻的内部超链接目标会指向同一个元素：

```
(*
Clicking on this internal hyperlink will take us to target1_
and clicking on this internal hyperlink will take us to target2_.
Both targets point the the same paragraph.

.. _target1:
.. _target2:

The targets "target1" and "target2" are synonyms; they both
point to this paragraph.
*)
```

Clicking on this internal hyperlink will take us to target1 and clicking on this internal hyperlink will take us to target2. Both targets point to the same paragraph.

The targets “target1” and “target2” are synonyms; they both point to this paragraph.

在当前文本中也可以插入内部超链接目标（请参见：内联超链接 [▶ 308]）。

13.6.2.8.2 外部超链接

外部超链接可以从注释或功能块文档链接到外部网站或打开电子邮件程序。

超链接引用

描述	超链接引用由引用名称以及后面的下划线组成： reference-name_ 在反引号中必须指定短语引用： `reference name`_ （另请参见：引用名称 [▶ 282]）
开始字符和结束字符	• 无开始字符，结束字符 = “_” • 开始字符 = “`”，结束字符 = “`_”（短语引用） （另请参见：内联标记 [▶ 280]）

超链接目标

描述	超链接目标由 1 个显式标记开始（“..”）、下划线、引用名称、冒号、空格和链接块组成： .. _reference-name: link-block 超链接目标中的短语引用可以选择性地包含在反引号中： .. _`reference name`: link-block .. _reference name: 链接块 （另请参见：显式标记块 [▶ 278]，引用名称 [▶ 282]）
原则	+-----+-----------------+ |".. "|"_"name":" link | +-----+block | | | +-----------------+

属性	• 外部超链接目标的链接块中有绝对 URI 或电子邮件地址。
	• 外部超链接的 URI 可以与显式标记在同一行中开始，也可以紧接着在缩进文本块中开始，中间不留空行。
	• 如果链接块中有多行，则将它们串联起来。链接块中的换行符会被删除。因此，下列外部引用目标是等效的：
	`.. _one-liner: https://infosys.beckhoff.de/`
	`.. _starts-on-this-line: http://infosys.beckhoff.de/`
	`.. _entirely-below: https://infosys.beckhoff.de/`
	如果外部超链接目标的 URI 包含下划线作为最后一个字符，则必须转义，以免与间接引用目标混淆（请参见 转义机制 [▶ 282]）。

示例

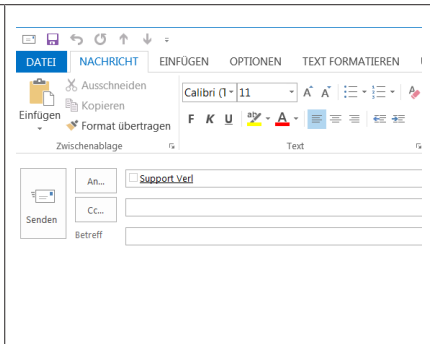
通过点击超链接引用，可在库管理器中直接调用网站或打开相应的电子邮件程序。
（在示例项目 [▶ 268]中：B_DocuElements\Hyperlinks\FB_Libdoc_ExternalHyperlinks）

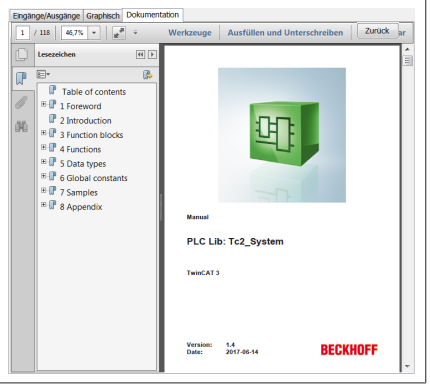
```
(*
See the Beckhoff_ home page for more information.

Please contact the `Beckhoff Support`_ for technical assistance.

.. _Beckhoff: http://www.beckhoff.de
.. _Beckhoff Support: support@beckhoff.com

For more information see the `PLC Lib Tc2_System`_ and `PLC Lib Tc2_Standard`_.

.. _PLC Lib Tc2_System: https://download.beckhoff.com/download/document/automation/twincat3/
TwinCAT_3_PLC_Lib_Tc2_System_EN.pdf
.. _PLC Lib Tc2_Standard: https://download.beckhoff.com/download/document/automation/twincat3/
TwinCAT_3_PLC_Lib__Tc2_Standard_EN.pdf
*)
```

See the Beckhoff home page for more information.

Please contact the Beckhoff Support for technical assistance.

For more information see PLC Lib Tc2_System and PLC Lib Tc2_Standard.





此外，还可以在引用中直接包含 URI。

```
(*
External hyperlinks, like `Beckhoff <http://www.beckhoff.de/>`_.
*)
```

External hyperlinks, like Beckhoff.

另请参见： 嵌入式 URI 和别名 [▶ 305]

13.6.2.8.2.1 独立超链接

文本块中的 URI（统一资源标识符）可被视为一般的外部超链接，URI 本身会显示为链接文本。

开始字符和结束字符	没有开始字符或结束字符。
属性	• 可识别绝对 URI 和独立电子邮件地址。

示例

下面的示例显示了 2 个单个超链接。URI 本身与链接文本相对应。
（在示例项目 [▶ 268]中：B_DocuElements\Hyperlinks\FB_Libdoc_ExternalHyperlinks）

```
(*
See http://www.beckhoff.de for info.

Contact support@beckhoff.com
*)
```

See http://www.beckhoff.de for info.

Contact support@beckhoff.com

13.6.2.8.2.2 嵌入式 URI 和别名

属性

- 带括号的 URI 前面必须有空格。
- 带括号的 URI 必须是字符串结束前的最后一个文本。
- 在末尾加上 1 个下划线，可为超链接引用“命名”，并可再次引用相同的目标 URI。

- 如果末尾有 2 个下划线，则超链接引用和超链接目标都是匿名的，无法再次引用目标。这些都是“独特的”超链接。

示例：

```
See the `Beckhoff Online Information System <http://infosys.beckhoff.de/
>`__ or the `Beckhoff home page <http://beckhoff.de/>`__.
```

这对应于：

```
See the `Beckhoff Online Information System`__ or the `Beckhoff home page`__.

__ http://infosys.beckhoff.de/
__ http://beckhoff.de/
```

See the Beckhoff Online Information System or the Beckhoff home page.

引用文本也可以省略，在这种情况下，复制 URI 可用作引用文本：

```
See the `<http://infosys.beckhoff.de/>`__ or the `<http://beckhoff.de/>`__.
```

See the http://infosys.beckhoff.de/ or the http://beckhoff.de/.

嵌入式 URI 结构便于创建和维护超链接，但却会损害一般识读率。内联 URI（特别是长 URI）不可避免地会干扰自然文本流。

示例

（在示例项目中：B_DocuElements\Hyperlinks\FB_Libdoc_ExternalHyperlinks）

超链接引用可直接将目标 URI 嵌入尖括号（“<...>”）中。

```
See the `Beckhoff home page <http://www.beckhoff.de>`_ for info.

This `link <Beckhoff home page_>`_ is an alias to the link above.
```

这对应于：

```
See the `Beckhoff home page`_ for info.

This link_ is an alias to the link above.

.. _Beckhoff home page: http://www.beckhoff.de
.. _link: `Beckhoff home page`_
```

See the Beckhoff home page for info.

This link is an alias to the link above.

13.6.2.8.3 间接超链接

对于间接超链接，超链接目标本身包含 1 个超链接引用。因此，间接超链接可以链接显式超链接目标。

超链接引用

描述	超链接引用由引用名称以及后面的下划线组成： reference-name_ 在反引号中必须指定短语引用： `reference name`_ （另请参见：引用名称 [▶ 282]）
开始字符和结束字符	• 无开始字符，结束字符 = “_” • 开始字符 = “`”，结束字符 = “`_”（短语引用） （另请参见：内联标记 [▶ 280]）

超链接目标

描述	超链接目标由 1 个显式标记开始（“..”）、下划线、引用名称、冒号、空格和链接块组成： .. _reference-name: link-block 超链接目标中的短语引用可以选择性地包含在反引号中： .. _`reference name`: link-block .. _reference name：链接块 （另请参见：显式标记块 [▶ 278]，引用名称 [▶ 282]）
原则	+-----+-----------------+ |".. "|"_"name":" link | +-----+block | | | +-----------------+
属性	• 间接超链接目标在其链接块中有 1 个超链接引用。 • 与外部超链接目标一样，间接超链接目标的链接块可以与显式标记块在同一行中开始，也可以在下一行中开始。 例如，以下间接超链接目标是等效的： `.. _one-liner: `A HYPERLINK`_` `.. _entirely-below:` `  `a hyperlink`_` `.. _split: `A` ` Hyperlink`_` 如果引用名称包含冒号： • 在超链接目标的链接块中，短语必须用反引号括起来 `Beckhoff Support:`_ * worldwide support * design, programming and commissioning of complex automation systems * training program for Beckhoff system components .. _`Beckhoff Support:`: support@beckhoff.com • 或者冒号必须带有反斜线： `Beckhoff Support:`_ * worldwide support * design, programming and commissioning of complex automation systems * training program for Beckhoff system components .. _Beckhoff Support\:: support@beckhoff.com

示例

（在示例项目 [▶ 268]中：B_DocuElements\Hyperlinks\FB_Libdoc_IndirectHyperlinks）

对内部引用目标的间接引用

在下面的示例中，超链接目标 .. _one 间接指的是目标 .. _two，目标 .. _two 间接指的是目标 .. _three，即内部引用目标。实际上，这 3 个目标指的是同一个目标（同一个段落）：

```
(*
This hyperlink points to target one_ and indirect to target three.

.. _one: two_
.. _two: three_
.. _three:

The hyperlink targets above point to this paragraph.
*)
```

This hyperlink points to target one and indirect to target three.

The hyperlink targets above point to this paragraph.

对内部引用目标的间接引用

在下面的示例中，目标 .. _Beckhoff 间接指的是目标 .. _Beckhoff Information System，即外部引用目标。

```
(*
The `Beckhoff Information System`_ is a reference source for Beckhoff_ products

.. _Beckhoff: `Beckhoff Information System`_

.. _Beckhoff Information System: https://infosys.beckhoff.de/
*)
```

The Beckhoff Information System is a reference source for Beckhoff products

此外，还可以在超链接目标中直接插入别名（请参见嵌入式 URI 和别名 [▶ 306]）。

13.6.2.8.4 内联超链接

内联超链接与内部超链接相对应，但超链接目标在文本中内联。

超链接引用

描述	超链接引用由引用名称以及后面的下划线组成： reference-name_ 在反引号中必须指定短语引用： `reference name`_ （另请参见：引用名称 [▶ 282]）
开始字符和结束字符	• 无开始字符，结束字符 = “_” • 开始字符 = “`”，结束字符 = “`_”（短语引用） （另请参见：内联标记 [▶ 280]）

超链接目标

描述	超链接目标由下划线以及后面的引用名称组成。 _reference-name 在反引号中必须指定短语引用。 _`reference name` 内部内联目标不能是匿名的。
开始字符和结束字符	• 开始字符 = “_”，无结束字符 • 开始字符 =“ _`”，结束字符 = “`” （另请参见：内联标记 [▶ 280]）

示例

例如，下面的段落包含 1 个超链接引用和 1 个名为“FB_Sample”的内联超链接目标。通过点击超链接引用，可显示带有超链接目标的句子。
（在示例项目 [▶ 268]中：B_DocuElements\Hyperlinks\FB_Libdoc_InlineHyperlinks）

```
(*
The function block _`FB_Sample` is used to...

For more information see the description of the `FB_Sample`_.
*)
```

The function block FB_Sample is used to...

For more information see the description of the FB_Sample.

13.6.2.8.5 匿名超链接

一般来说，超链接的名称应该尽可能详细且有意义。不过，在超链接目标中复制较长的引用名称可能既费时，又容易出错。

对于匿名超链接，超链接目标不包含引用名称，也就是说，不会使用超链接引用的名称来匹配引用及其目标。相反，注释中的匿名超链接目标的顺序很重要：第 1 个匿名超链接引用链接到第 1 个匿名超链接目标，以此类推。注释中匿名超链接引用的数量必须与匿名超链接目标的数量相匹配。

匿名超链接可能会导致误导性和难以辨认的注释。出于识读率的考虑，建议将超链接目标放在尽可能靠近超链接引用的位置。

在编辑带有匿名超链接的注释时，请注意相应超链接目标的顺序也必须进行调整，在添加、删除和重新排列超链接引用时更是如此。

超链接引用

描述	超链接引用由引用名称以及后面的 2 个下划线组成： anonymous-hyperlink-reference-name__. 在反引号中必须指定短语引用。 `anonymous hyperlink reference name`__.
开始字符和结束字符	• 无开始字符，结束字符 = “__” • 开始字符 = “`”，结束字符 = “`__”（短语引用） （另请参见：内联标记 [▶ 280]）

超链接目标

描述	超链接目标由 1 个显式标记开始（“.. ”）、2 个下划线、冒号、空格和链接块组成。没有引用名称。 .. __: anonymous-hyperlink-target-link-block 或者，匿名超链接也可以由 2 个下划线、空格和链接块组成： __ anonymous-hyperlink-target-link-block （另请参见：显式标记块 [▶ 278]）
原则	+-----+------------------+ |".. "|"__"name":" link | +-----+block | | | +------------------+

示例

在下面的示例中，第 1 个匿名超链接引用指的是倍福在线信息系统，第 2 个匿名超链接引用指的是倍福网站。如果匿名超链接目标的顺序改变，则引用的赋值也会随之改变。
（在示例项目 [▶ 268]中：B_DocuElements\Hyperlinks\FB_Libdoc_AnonymousHyperlinks）

```
(*
See the `Beckhoff Online Information System`__ or the `Beckhoff home page`__.

.. __: http://infosys.beckhoff.de/

.. __: http://beckhoff.de/
*)
```

See the Beckhoff Online Information System or the Beckhoff home page.

13.6.2.8.6 脚注

脚注可以删除文本中的注解、备注或对文本段落的引用，从而使文本更加清晰易读。

与超链接一样，脚注也由 2 部分组成：脚注引用（来源）和脚注（目标）。数字名称可用作引用名称。

脚注的类型

脚注可以手动、自动或手动/自动混合编号：

- 手动编号的脚注 [▶ 310]
- 自动编号的脚注 [▶ 312]
- 已命名的自动编号的脚注 [▶ 313]
- 手动和自动编号的脚注 [▶ 316]

此外，还可以自动生成脚注符号：

- 自动生成符号 [▶ 315]

13.6.2.8.6.1 手动编号的脚注

对于手动编号的脚注，为每个脚注都分配了 1 个名称。

脚注引用

描述	脚注引用由方括号内的脚注标签和末尾的下划线组成。 脚注标签是由 1 个或多个数字组成的任意整数。
开始字符和结束字符	开始字符 = “[”，结束字符 = “]_” （另请参见：内联标记 [▶ 280]）

脚注

描述	脚注（脚注标签）由 1 个显式标记开始（“.. ”）、用方括号括起来的脚注标签、空格以及后面的缩进主体元素（脚注内容）组成。 （另请参见：显式标记块 [▶ 278]）
原则	+-----+----------------------+ |".. "|"[label]" footnote | +-----+ | |(body elements) | +----------------------+

属性	• 在脚注标签和脚注内容之间可能有空行，也可能没有空行。
	• 脚注内容必须缩进（至少 1 个空格）。
	• 您可以将脚注放在注释中的任何位置，而不仅仅是末尾处。
	示例：
	`Footnote references, like [1]_.`
	`.. [1] Body elements go here.`
	或
	`Footnote references, like [1]_`
	`.. [1]`
	` Body elements go here.`
	Footnote references, like [1].
	[1] Body elements go here.
	或
	`Footnote references, like [1]_.`
	`.. [1]`
	` - Body elements go here.`
	` - Second body element.`
	Footnote references, like [1].
	[1] ■ Body elements go here.
	■ Second body element.

示例

点击脚注引用 [5]_ 可显示脚注内容。
（在示例项目 [▶ 268]中：
B_DocuElements\Hyperlinks\Footnotes\FB_Libdoc_ManuallyNumberedFootnotes）

```
(*
**Numerical footnote**

Footnote references, like [5]_. Note that footnotes may get rearranged, e.g., to the bottom of the "
page".

--------

.. [5] A numerical footnote. Note there's no colon after the ``]``.
*)
```

Numerical footnote

Footnote references, like [5]. Note that footnotes may get rearranged, e.g., to the bottom of the “page”.

[5] A numerical footnote. Note there’s no colon after the].

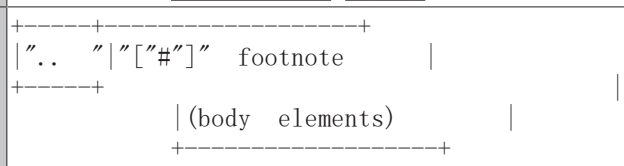
13.6.2.8.6.2 自动编号的脚注

对于自动编号的脚注，为每个脚注都自动分配了 1 个名称。

脚注引用

描述	脚注引用由方括号内的脚注标签和末尾的下划线组成。 脚注标签为单个“#”。
开始字符和结束字符	开始字符 = “[”，结束字符 = “]_” （另请参见：内联标记 [▶ 280]）

脚注

描述	每个脚注（脚注标签）由 1 个显式标记开始（“.. ”）、用方括号括起来的脚注标签、空格以及后面的缩进主体元素（脚注内容）组成。 （另请参见：显式标记块 [▶ 278]）
原则	

属性	• 为第 1 个要求自动编号的脚注分配标签“1”，为第 2 个脚注分配标签“2”，等等。（如果没有手动编号的脚注）。自动获得标签“1”的脚注会创建 1 个名称为“1”的隐式超链接目标，就像明确指定了标签一样。
	• 编号由脚注的顺序决定，而不是由脚注引用的顺序决定。对于自动编号的脚注（[#]_），脚注和脚注引用的相对顺序必须相同，但不必链接。
	示例：
	`[#]_ is a reference to footnote 1, and [#]_ is a reference to footnote 2.`
	
	`.. [#] This is footnote 1.`
	`.. [#] This is footnote 2.`
	`.. [#] This is footnote 3.`
	
	`[#]_ is a reference to footnote 3.`
	
	[1] is a reference to footnote 1, and [2] is a reference to footnote 2.
	
	[1] This is footnote 1.
	[2] This is footnote 2.
	[3] This is footnote 3.
	
	[3] is a reference to footnote 3.
	
	如果脚注本身包含自动编号的脚注引用，或者如果紧邻脚注的地方有多个引用，则需要特别注意。脚注和引用按其在注释中出现的顺序进行标注，但不一定与个人阅读的顺序一致。
	（另请参见：已命名的自动编号的脚注 [▶ 313]）

示例

下面的示例自动显示了带有和不带附加脚注标签的编号脚注。
（在示例项目 [▶ 268]中：
B_DocuElements\Hyperlinks\Footnotes\FB_Libdoc_AutomaticallyNumberedFootnotes）

```
(*
**Autonumbered footnotes**

Autonumbered footnotes are possible, like using [#]_ and [#]_.

--------

.. [#] This is the first one.

.. [#] This is the second one.
*)
```

Autonumbered footnotes

Autonumbered footnotes are possible, like using [1] and [2].

---

[1] This is the first one.
[2] This is the second one.

13.6.2.8.6.3 已命名的自动编号的脚注

对于已命名的自动编号脚注，为每个脚注都自动分配了 1 个名称，同时明确指定 1 个标签。

脚注引用

描述	脚注引用由方括号内的脚注标签和末尾的下划线组成。 脚注标签为单个“#”，后面带有任何标签。
开始字符和结束字符	开始字符 = “[”，结束字符 = “]_” （另请参见：内联标记 [▶ 280]）

脚注

描述	每个脚注（脚注标签）由 1 个显式标记开始（“.. ”）、用方括号括起来的脚注标签、空格以及后面的缩进主体元素（脚注内容）组成。 （另请参见：显式标记块 [▶ 278]）
原则	+-----+----------------------+ |".. "|"["#label"]" footnote | +-----+ | |(body elements) | +----------------------+

属性	• 在脚注本身上会创建 1 个名称与指定标签（不带“#”）相对应的超链接目标。
	• 显式标签允许对自动编号的脚注进行唯一分配，并将其多次指定为脚注引用或超链接引用。
	示例显示了 1 个自动编号的脚注，该脚注同时引用了脚注引用（[#note]_ 或 [#label]_）和超链接引用（note_ 或 label_）。
	`If [#note]_ is the first footnote, it will show up as "[1]".`
	`We can refer to it again as [#note]_. We can also refer to it as note_ (an ordin`
	`ary internal hyperlink reference).`
	
	`If [#label]_ is the second footnote, it will show up as "[2]".`
	`We can refer to it again as [#label]_. We can also refer to it as label_ (an ord`
	`inary internal hyperlink reference).`
	
	`.. [#note] This is the first footnote labeled "note".`
	`.. [#label] This is the second footnote labeled "label".`
	
	If [1] is the first footnote, it will show up as “[1]”. We can refer to it again as [1]. We can also refer to it as note (an ordinary internal hyperlink reference).
	
	If [2] is the second footnote, it will show up as “[2]”. We can refer to it again as [2]. We can also refer to it as label (an ordinary internal hyperlink reference).
	
	[1] (1, 2) This is the first footnote labeled “note”.
	[2] (1, 2) This is the second footnote labeled “label”.
	
	（另请参见：自动编号的脚注 [▶ 312]）

示例

下面的示例自动显示了带有显式脚注标签的编号脚注。

（在示例项目 [▶ 268]中：
B_DocuElements\Hyperlinks\Footnotes\FB_Libdoc_NamedAutomaticallyNumberedFootnotes）

```
(*
**Autonumbered labeled footnotes**

They may be assigned 'autonumber labels' - for instance, [#first]_ and [#second]_.

--------

.. [#first] a.k.a. first_

.. [#second] a.k.a. second_
*)
```

Autonumbered labeled footnotes

They may be assigned ‘autonumber labels’ - for instance, [1] and [2].

---

[1] a.k.a. first
[2] a.k.a. second

13.6.2.8.6.4 自动生成脚注符号

在自动生成脚注符号时，会为每个脚注分配 1 个符号。

脚注引用

描述	脚注引用由方括号内的脚注标签和末尾的下划线组成。 脚注标签为单个星号“*”。
开始字符和结束字符	开始字符 = “[”，结束字符 = “]_” （另请参见：内联标记 [▶ 280]）

脚注

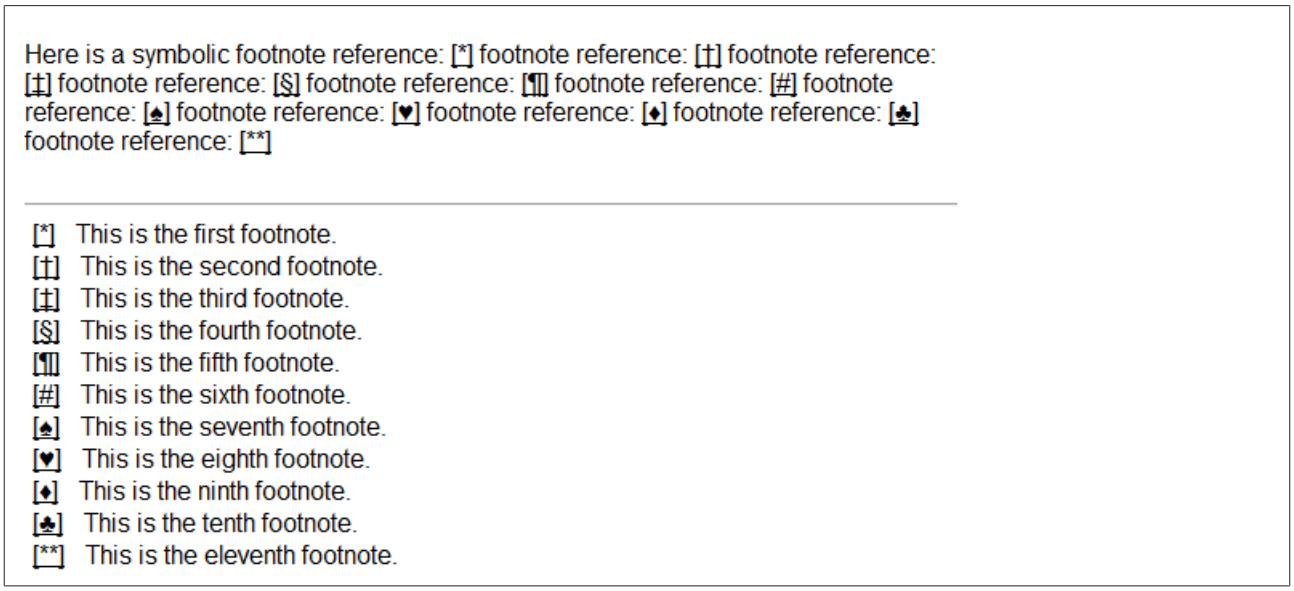
描述	每个脚注（脚注标签）由 1 个显式标记开始（“.. ”）、用方括号括起来的脚注标签、空格以及后面的缩进主体元素（脚注内容）组成。 （另请参见：显式标记块 [▶ 278]）
原则	+-----+--------------------+ |".. "|"["*"]" footnote | +-----+ | |(body elements) | +--------------------+
属性	• 脚注引用的数量必须与脚注的数量相对应。 • 不得多次引用符号脚注。 • 转换会在相应的脚注和脚注引用中插入符号作为标签。以下符号可用于脚注： ◦ 星号/星形（“*”） ◦ 匕首（“†”） ◦ 双匕首（“‡”） ◦ 段落符号（“§”） ◦ 段落标记（“¶”） ◦ 数字符号（“#”） ◦ 黑桃（“♠”） ◦ 红桃（“♥”） ◦ 方块（“♦”） ◦ 梅花（“♣”） • 如果需要 10 个以上的符号，则应重复使用相同的顺序，双倍、然后 3 倍，以此类推。（“**”等）。

示例

下面的示例显示了脚注符号的自动生成及其在库管理器中的表示。
（在示例项目 [▶ 268]中：
B_DocuElements\Hyperlinks\Footnotes\FB_Libdoc_AutomaticallySymbolFootnotes）

```
(*
Here is a symbolic
footnote reference: [*]_ footnote reference: [*]_
footnote reference: [*]_ footnote reference: [*]_
footnote reference: [*]_ footnote reference: [*]_
footnote reference: [*]_ footnote reference: [*]_
footnote reference: [*]_ footnote reference: [*]_
footnote reference: [*]_

.. [*] This is the first footnote.
.. [*] This is the second footnote.
.. [*] This is the third footnote.
.. [*] This is the fourth footnote.
.. [*] This is the fifth footnote.
.. [*] This is the sixth footnote.
.. [*] This is the seventh footnote.
.. [*] This is the eighth footnote.
```

```
.. [*] This is the ninth footnote.
.. [*] This is the tenth footnote.
.. [*] This is the eleventh footnote.
*)
```



13.6.2.8.6.5 手动和自动编号的脚注

在注释中既可以使用手动编号的脚注，也可以使用自动编号的脚注；手动生成的脚注优先编号。只有未使用的脚注标签才会被自动分配给编号的脚注。

示例

（在示例项目 [▶ 268]中：
B_DocuElements\Hyperlinks\Footnotes\FB_Libdoc_ManuallyAndAutomaticallyNumberedFootnotes）

```
(*
[2]_ will be "2" (manually numbered footnote),

[3]_ will be "3" (another manually numbered footnote),

[#]_ will be "4" (anonymous auto-numbered footnote), and

[#label]_ will be "1" (labeled auto-numbered).

--------

.. [2] This footnote is labeled manually, so its number is fixed.

.. [3] This footnote is also labeled manually, so its number is also fixed.

.. [#label] This autonumber-labeled footnote will be labeled "1".
   It is the first auto-numbered footnote and no other footnote
   with label "1" exists.  The order of the footnotes is used to
   determine numbering, not the order of the footnote references!

.. [#] This footnote will be labeled "4".  It is the second
   auto-numbered footnote, but footnote labels "1", "2", "3" are already used.
*)
```

[2] will be “2” (manually numbered footnote),

[3] will be “3” (another manually numbered footnote),

[4] will be “4” (anonymous auto-numbered footnote), and

[1] will be “1” (labeled auto-numbered).

---

[2] This footnote is labeled manually, so its number is fixed.
[3] This footnote is also labeled manually, so its number is also fixed.
[1] This autonumber-labeled footnote will be labeled “1”. It is the first auto-numbered footnote and no other footnote with label “1” exists. The order of the footnotes is used to determine numbering, not the order of the footnote references!
[4] This footnote will be labeled “4”. It is the second auto-numbered footnote, but footnote labels “1”, “2”, “3” are already used.

另请参见：

- 手动编号的脚注 [▶ 310]
- 自动编号的脚注 [▶ 312]

13.6.2.8.7 引文

引文与脚注相同，只是它们使用字母数字名称，例如，[note] 或 [GVR2001]，并在库管理器中以表格形式显示。

与超链接一样，引文也由 2 部分组成：引文引用 [▶ 317]（来源）和引文 [▶ 317]（目标）。

引述引用

描述	每个引文引用由方括号内的引文标签和后面的下划线组成。 引文标签是简单的引用名称（单个词，由字母数字字符组成，无空格）。
开始字符和结束字符	开始字符 = “[”，结束字符 = “]_” （另请参见：内联标记 [▶ 280]）

引述（目标）

描述	每个引文（引文标签）由 1 个显式标记开始（“.. ”）、用方括号括起来的引文标签、空格以及后面的缩进主体元素（引文内容）组成。 （另请参见：显式标记块 [▶ 278]）
原则	.. [CIT]
属性	• 引文与脚注不同，它们在库管理器的表格中显示。 • 在引文标签和引文内容之间可能有空行，也可能没有空行。 • 引用内容必须缩进（至少 1 个空格）。 • 您可以将引文放在注释中的任何位置，而不仅仅是末尾处。

示例

（在示例项目 [▶ 268]中：B_DocuElements\Hyperlinks\FB_Libdoc_Citation）

```
(*
Citation references, like [CIT2002]_. Note that citations may get rearranged, e.g., to the bottom of
 the "page".

.. [CIT2002] This is the citation.  It's just like a footnote,
   except the label is textual.

Given a citation like [this]_, one can also refer to it like this_.

.. [this] here.
*)
```

Citation references, like [CIT2002]. Note that citations may get rearranged, e.g., to the bottom of the "page".

[CIT2002]	This is the citation. It's just like a footnote, except the label is textual.

Given a citation like [this], one can also refer to it like this.

[this]	here.

13.6.2.8.8 链接到另一个对象

您可以设置指向该库中包含的另一个库对象的文档的链接。点击该链接可显示链接对象或功能块的文档。

描述	引用由 :ref: 以及后面的引用名称组成，引用名称用反引号指定： :ref:`Objectname`

示例

（在示例项目 [▶ 268]中：B_DocuElements\Hyperlinks\FB_Libdoc_LinkToAnotherObject）

以下示例代码包含 2 个链接，分别指向不同的功能块。点击该链接可在库管理器中显示链接功能块的文档。

```
(*
| It is also possible to create a reference to the documentation of another object which is also
part of this library.
| For example, by clicking on this link :ref:`FB_Libdoc_InlineHyperlinks`, the documentation of this
FB will be shown.
| Or you can also create a link to :ref:`FB_Libdoc_CodeBlock` which is part of another folder.

*)
```

13.6.2.9 替换

替换让您可以在文本中加入任意数量的复杂内联结构，同时还能将细节保留在文本流之外。

它们由 2 部分组成：替换引用 [▶ 318]和替换定义 [▶ 319]。处理系统会用相应替换定义的内容替换替换引用。

替换引用

描述	替换引用由用竖条括起来的引用文本组成。 通过附加“_”（具名）或“__”（匿名），替换引用也可以是超链接引用。
开始字符和结束字符	开始字符 = “|”，结束字符 = “|”（可选后面带有“_”或“__”）。 （另请参见：内联标记 [▶ 280]）
属性	• 使用替换定义的内容对替换引用进行内联替换。 • 引用文本不得以空格开始或结束

替换定义

描述	替换定义由 1 个显式标记开始（“.. ”）、后面用竖条括起来的引用文本、空格和定义块组成。 定义块包含嵌入式内联兼容指令，例如，“图像”或“替换”。 （另请参见：显式标记块 [▶ 278]）
原则	+-----+----------+ |".. "|"|"reference text"|" directive type"::" data | +----- +directive block |
属性	• 替换定义的内容会替换内联的替换引用。 • 引用文本不得以空格开始或结束。

应用

下文介绍了替换机制的一些用例：

- 替换文本 [▶ 319]
- 图像 [▶ 320]

替换文本

替换机制可用于简单的文本替换。如果替换文本在注释中多次重复出现，尤其是在以后需要修改时，这将非常有用。

指令类型：replace

示例

（在示例项目 [▶ 268]中：B_DocuElements\Substitution\FB_Libdoc_Substitution_ReplacementText）

```
(*
|RST|_ is an easy-to-read, what-you-see-is-what-you-get plaintext markup syntax.
Among others it is useful for library documentation. If the |RST| option is activated,
comments written in |RST|
 will be shown in the Documentation tab of the library manager in an attractive format.
|RST| is a little annoying to type over and over and spelling out the
bicapitalized word |RST| every time isn't really necessary for |RST| source readability.

.. |RST| replace:: reStructuredText
.. _RST: https://infosys.beckhoff.de/
*)
```

reStructuredText is an easy-to-read, what-you-see-is-what-you-get plaintext markup syntax. Among others it is useful for library documentation. If the reStructuredText option is activated, comments written in reStructuredText will be shown in the Documentation tab of the library manager in an attractive format. reStructuredText is a little annoying to type over and over and spelling out the bicapitalized word reStructuredText every time isn't really necessary for reStructuredText source readability.

当您第 1 次使用替换引用时，请注意最后的下划线。这表示对相应超链接目标的引用。在替换定义中不会处理内联标记。

```
(*
This is a simple |substitution reference|. It will be replaced by
the processing system.

This is a |substitution and hyperlink reference|_. In
addition to being replaced, the replacement text or element will
refer to the "substitution and hyperlink reference" target.
```

```
.. |substitution reference| replace:: example of substitution
.. |substitution and hyperlink reference| replace:: combination of substitution and hyperlink
reference
.. _substitution and hyperlink reference: https://beckhoff.de/
*)
```

This is a simple example of substitution. It will be replaced by the processing system.

This is a combination of substitution and hyperlink reference. In addition to being replaced, the replacement text or element will refer to the "substitution and hyperlink reference" target.

图像

（在示例项目 [▶ 268]中：B_DocuElements\Substitution\FB_Libdoc_Substitution_Images）

替换机制还可用于在注释文本中加入图像。

指令类型：image

（另请参见：图像 [▶ 322]）

示例

在段落中替换

徽标对注释文本 |Beckhoff|__ 加以补充。替换引用后面的下划线表示对超链接目标的引用。点击图像，可在库管理器中打开倍福网站。

```
(*
Images are a common use for substitution references: |Beckhoff|_.

.. |Beckhoff| image:: C:\Tc3LibDocImages\SampleLib1\logo.gif
 :height: 16
 :width: 64
.. _Beckhoff: http://www.beckhoff.de/
*)
```

Images are a common use for substitution references: BECKHOFF.

```
(*
The |safety| symbol indicates a hazardous situation which,
if not avoided, could result in minor or moderate injury.

.. |safety| image:: C:\Tc3LibDocImageslogo\SampleLib1\safetysymbol.png
*)
```

The ⚠ symbol indicates a hazardous situation which, if not avoided, could result in minor or moderate injury.

在列表或表格中替换

```
(*
* |Run mode| Run mode
* |Stop mode| Stop mode
* |Config mode| Config mode

============= ======
Symbol        Status
============= ======
|Run mode|    Run mode
|Stop mode|   Stop mode
|Config mode| Config mode
============= ======
```

```
.. |Run mode| image:: C:\Tc3LibDocImages\SampleLib1\tc3rtrunmode.png
.. |Stop mode| image:: C:\Tc3LibDocImages\SampleLib1\tc3rtstopmode.png
.. |Config mode| image:: C:\Tc3LibDocImages\SampleLib1\tc3rtconfigmode.png
*)
```

- Run mode
- Stop mode
- Config mode

Symbol	Status
	Run mode
	Stop mode
	Config mode

13.6.2.10 说明元素

reStructuredText 可区分特定说明和一般说明。在库管理器中，说明可显示为带有标题（信号词）阴影的偏移块。

13.6.2.10.1 特定说明

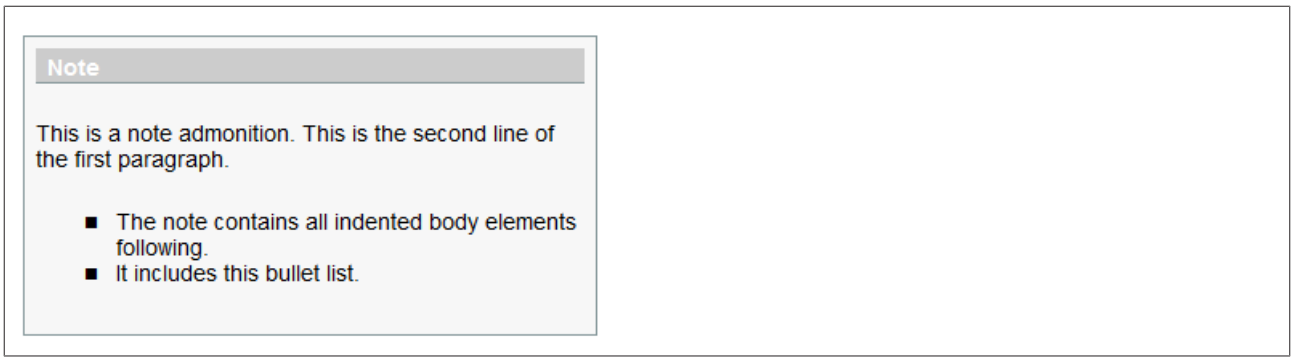
使用指令 attention、caution、danger、error、hint、important、note、tip 和、warning 可以生成特定说明（例如，安全和警告通知）。说明块的标题由指令指定，并与说明的类型相对应。

描述	指令标记由 1 个显式标记开始（“.. ”），以及后面的指令类型（例如，note）和 2 个冒号组成。 （另请参见：指令 [▶ 279]）
原则	.. note::
属性	• 在注释中，特定说明可用作普通主体元素，并包含任何主体元素。 • 以下说明类型已实现： ◦ 注意 ◦ 小心 ◦ 危险 ◦ 错误 ◦ 暗示 ◦ 重要信息 ◦ 说明 ◦ 提示 ◦ 警告 • 在同一行中紧跟指令标记和/或在随后行中缩进的任何文本都会被解释为指令块（指令内容）。 • 在指令和前一个文本主体元素（例如，1 个带有文本的段落）之间需要有 1 个空行。

示例

（在示例项目 [▶ 268]中：B_DocuElements\Note elements\FB_Libdoc_SpecificNotes）

```
(*
.. note:: This is a note admonition.
   This is the second line of the first paragraph.

   - The note contains all indented body elements
     following.
   - It includes this bullet list.
*)
```



13.6.2.10.2 通用说明

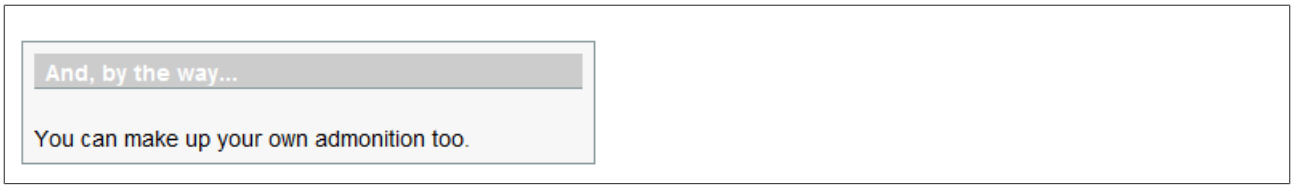
使用 admonition 指令可以创建通用说明。对于通用说明，您可以选择标题，从而选择说明类型。

描述	指令标记由 1 个显式标记开始（“.. ”），以及后面的指令类型（admonition）和 2 个冒号组成。 （另请参见：指令 [▶ 279]）
原则	.. admonition::
属性	• 指令参数与引用标题相对应 • 在指令选择和指令块（指令内容）之间必须插入 1 个空行。 • 在指令和前一个文本主体元素（例如，1 个带有文本的段落）之间需要有 1 个空行。

示例

（在示例项目 [▶ 268]中：B_DocuElements\Note elements\FB_Libdoc_GenericNotes）

```
(*
.. admonition:: And, by the way...

   You can make up your own admonition too.
*)
```



13.6.2.11 图像

使用 image 指令可以在注释中加入图像。

描述	指令标记由 1 个显式标记开始（“.. ”），以及后面的指令类型（image）和 2 个冒号组成。 （另请参见：指令 [▶ 279]）
原则	.. image::
属性	• 在指令的参数中可以指定图像源文件的文件路径。与超链接目标一样，文件路径可以紧接着在显式标记开始的同一行中或缩进文本块（中间不留空行）中开始。文件路径可以绝对指定，也可以相对指定，且不得包含任何空格。（请参见：作为指令参数的绝对或相对文件路径 [▶ 323]） • 根据库管理器的窗口宽度，可尽可能宽地显示图像（=100%）。如果更改库管理器的窗口宽度，则图像的比例将保持不变。 • 在指令和前一个文本主体元素（例如，1 个带有文本的段落）之间需要有 1 个空行。

选项

指令块可以选择性地包含 1 个平面字段列表，其中包含图像选项。可识别以下选项：

高度：长度	图像的高度。 用于垂直缩放图像。尽可能按比例调整图像的宽度。 • 以 px 为单位的规格（带空格或不带空格）： ◦ 无论库管理器窗口的高度和宽度如何，显示图像的高度始终对应于以 px 为单位的指定图像高度。如果库管理器窗口的高度小于指定的图像高度，则您可以通过在库管理器内滚动的方式来显示图像的剩余部分。 ◦ 图像宽度最多与库管理器的窗口宽度一样大：如果库管理器窗口足够宽，则会按比例显示图像。否则显示内容会几何畸变。 ◦ 因此，图像的比例取决于库管理器的窗口宽度。
宽度：当前行宽的长度或百分比	图像的宽度。 用于水平缩放图像。尽可能按比例调整图像的高度。 • 以 px 为单位的规格（带空格或不带空格）： ◦ 显示图像的宽度尽可能对应于以 px 为单位的指定图像宽度，但最多与库管理器的窗口宽度一样大。这意味着，如果库管理器窗口的宽度小于指定的图像宽度，则图像的宽度仅为库管理器窗口的宽度。 ◦ 根据指定的图像宽度按比例调整图像的高度。 ◦ 因此，图像的比例取决于库管理器的窗口宽度。 • 以 % 为单位的规格（带空格或不带空格）： ◦ 显示图像的宽度占库管理器的窗口宽度的指定百分比。 ◦ 根据图像宽度按比例调整图像的高度。 ◦ 这样，图像的比例将保持不变。
对齐：“左”或“右”	图像的对齐方式。 “左”和“右”的值可以控制图像的水平对齐方式，以便图像可以浮动，且文本可以围绕图像流动。
目标：文本（URI 或引用名称）	将图像转化为引用（“可点击”）。选项的参数可以是 URI（相对或绝对），也可以是带有下划线后缀的引用名称（例如，name_）。
比例：整数百分比	图像的均匀缩放因子。 用于按比例缩放图像。 • 以 % 为单位的规格（带空格或不带空格）： ◦ 显示图像的宽度和高度始终对应于原始图像大小（即在文件系统中引用的图形大小）的指定百分比，与库管理器窗口的宽度和高度无关。如果库管理器窗口的宽度或高度小于指定的图像宽度或高度，则您可以通过在库管理器内滚动的方式来显示图像的剩余部分。 ◦ 这样，图像的比例将保持不变。

如果包含的图像没有高度或宽度选项，则显示图像的宽度会对应于库管理器的窗口宽度。根据图像宽度按比例调整图像的高度。可尽可能大地显示图像。图像的比例将保持不变。

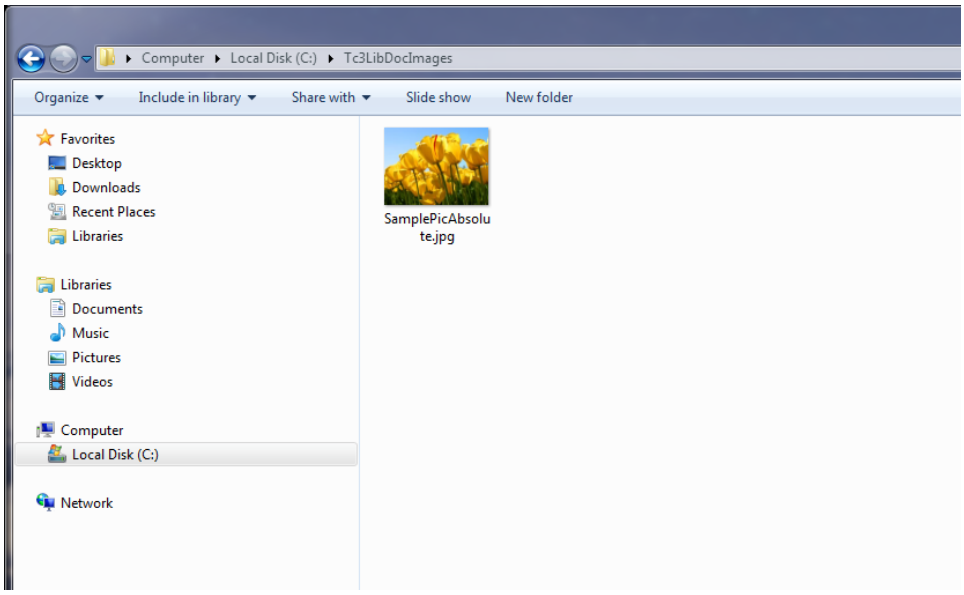
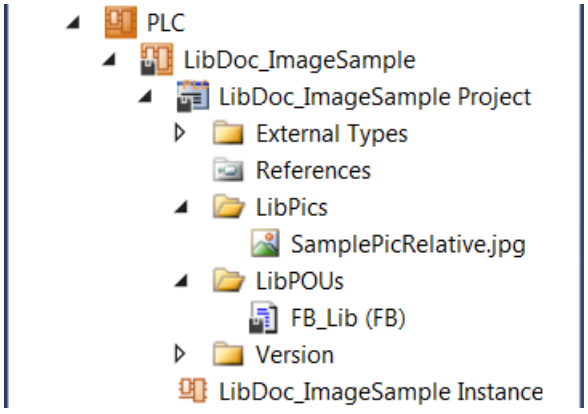
作为指令参数的绝对或相对文件路径

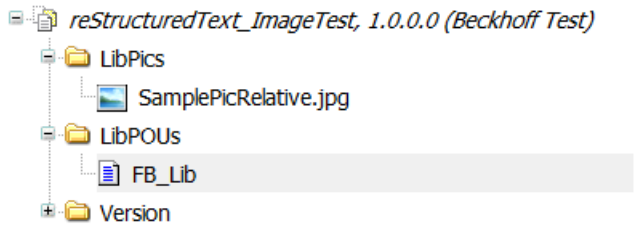
您可以在图像指令的参数中指定绝对和相对文件路径，以便引用图像。

如果您将图像保存在文件系统中的任意文件夹中，则可指定绝对文件路径。如果图像集成在库项目中，则可指定相对路径。

转发库文件

如果您在库项目中集成图像并使用相对文件路径，然后将库项目作为文件进行转发，则所需的图像信息将作为库项目的一部分直接随库项目一起发送。如果您使用绝对文件路径，则必须单独转发图像，并且图像必须存在于文件路径中为使用库的人员指定的文件系统中的位置。后者也是 1 种静态方法，在这种方法中，文件系统中图像信息的组织和维护需要额外工作。

<table>
<tr><td>绝对文件路径</td><td>例如，C:\Tc3LibDocImages\SamplePicAbsolute.jpg
 
</td></tr>
<tr><td>相对文件路径
要求：TwinCAT 3.1.4022.22</td><td>例如，../../LibPics/SamplePicRelative.jpg
在 PLC 项目中集成图像
1. 在 PLC 项目树中创建 1 个图像文件夹，以便确保统一的存储位置和明确的项目结构（例如，LibPics）。
2. 在文件夹的上下文菜单中，选择命令 Add ＞ Existing element（添加 ＞ 现有元素）。
3. 在文件系统中，选择图像（例如，SamplePicRelative）并确认对话框。
⇨ 图像会被添加到 PLC 项目中，并且可在库对象文档的 reStructuredText 注释中引用。（请参见：示例 [▶ 325]）
⇨ 在将 PLC 项目保存为库时，图像将与项目一起保存在库文件（*.library/*.compiled-library）中。如果在 PLC 项目中引用该库，则在库管理器中将会显示图像与该库。双击图库管理器中的图像，即可打开图像。
解决方案资源管理器：
 

库管理器：
 
</td></tr>
</table>

示例

（在示例项目 [▶ 268]中：B_DocuElements\Images\FB_Libdoc_Images）

作为指令参数的绝对或相对文件路径

```
(*
Absolute path

.. image:: C:\Tc3LibDocImages\SamplePicAbsolute.jpg
   :width: 70px

Relative path

.. image:: ../../LibPics/SamplePicRelative.jpg
   :width: 70px
*)
```

Absolute path

Relative path

左对齐的图像

```
(*
.. image:: C:\Tc3LibDocImages\SampleLib1\img11.jpg
   :width: 20%

Description of the function block. Description of the function block. Description of the function bl
ock.
Description of the function block. Description of the function block. Description of the function bl
ock.
Description of the function block. Description of the function block. Description of the function bl
ock.
Description of the function block. Description of the function block. Description of the function bl
ock.
*)
```

Description of the function block. Description of the function block.
Description of the function block. Description of the function block.
Description of the function block. Description of the function block.
Description of the function block. Description of the function block.
Description of the function block. Description of the function block.
Description of the function block. Description of the function block.

浮动图像，左对齐

```
(*
.. image:: C:\Tc3LibDocImages\SampleLib1\img11.jpg
   :width: 20%
   :align: left

Description of the function block. Description of the function block. Description of the function bl
ock.
Description of the function block. Description of the function block. Description of the function bl
ock.
Description of the function block. Description of the function block. Description of the function bl
```

```
ock.
Description of the function block. Description of the function block. Description of the function bl
ock.
*)
```

Description of the function block. Description of the function block. Description of the function block. Description of the function block. Description of the function block. Description of the function block. Description of the function block. Description of the function block. Description of the function block. Description of the function block.

浮动图像，右对齐

```
(*
.. image:: C:\Tc3LibDocImages\SampleLib1\img11.jpg
   :width: 20%
   :align: right

Description of the function block. Description of the function block. Description of the function bl
ock.
Description of the function block. Description of the function block. Description of the function bl
ock.
Description of the function block. Description of the function block. Description of the function bl
ock.
Description of the function block. Description of the function block. Description of the function bl
ock.
*)
```

Description of the function block. Description of the function block. Description of the function block. Description of the function block. Description of the function block. Description of the function block. Description of the function block. Description of the function block. Description of the function block. Description of the function block.

使用替换定义中的 image 指令可以定义内联图像（请参见 替换 [▶ 318]）。

13.6.2.12 码块

code 指令允许在注释中显示代码区域。它们在库管理器中显示为灰色阴影，以等宽字体显示，采用颜色编码语法。

描述	指令标记由 1 个显式标记开始（“.. ”），以及后面的指令类型（code）和 2 个冒号组成。 （另请参见：指令 [▶ 279]）
原则	.. code::
属性	• 在指令和前一个文本主体元素（例如，1 个带有文本的段落）之间需要有 1 个空行。

选项

指令块可以选择性地包含 1 个代码选项的平面列表。可识别以下选项：

编号行：[start line number]	每行前面都有 1 个行号。可选参数是第 1 行的编号。默认值为 1。 在这种情况下，语法不会以彩色突出显示。

示例

（在示例项目 [▶ 268]中：B_DocuElements\Code Block\FB_Libdoc_CodeBlock）

采用颜色编码语法的代码块

```
(*
.. code::

    // Attempts to return the value of a boolean property.

    FUNCTION GetBooleanProperty : BOOL
    VAR_INPUT
        sKey: STRING;
    END_VAR

    // This structure defines a special profile.

    TYPE ST_Profile :
    STRUCT
        nId     : INT := -1;
        sBuffer : STRING(255) := 'Hello';
    END_STRUCT
    END_TYPE
*)
```

```
// Attempts to return the value of a boolean property.

FUNCTION GetBooleanProperty : BOOL
VAR_INPUT
    sKey: STRING;
END_VAR

// This structure defines a special profile.

TYPE ST_Profile :
STRUCT
    nId     : INT := -1;
    sBuffer : STRING(255) := 'Hello';
END_STRUCT
END_TYPE
```

带有行号的代码块

```
(*
.. code::
   :number-lines: 1

    // Attempts to return the value of a boolean property.

    FUNCTION GetBooleanProperty : BOOL
    VAR_INPUT
        sKey: STRING;
    END_VAR

    // This structure defines a special profile.

    TYPE ST_Profile :
    STRUCT
        nId     : INT := -1;
        sBuffer : STRING(255) := 'Hello';
    END_STRUCT
    END_TYPE
*)
```

```
// Attempts to return the value of a boolean property.

FUNCTION GetBooleanProperty : BOOL
VAR_INPUT
    sKey: STRING;
END_VAR

// This structure defines a special profile.

TYPE ST_Profile :
STRUCT
    nId     : INT := -1;
    sBuffer : STRING(255) := 'Hello';
END_STRUCT
END_TYPE
```

13.6.2.13 内部注释

在输出时会删除内部注释，在库管理器中不会显示它们。

描述	内部注释由 1 个显式标记开始（“.. ”）以及后面的任何缩进文本组成。 （另请参见：显式标记块 [▶ 278]）
原则	+-----+----------------+ |".. "|comment | +-----+block | | | +----------------+

属性	• 注释文本可以与显式标记开始处于同一行，也可以在下一行中缩进。
	`.. This is a comment`
	`..`
	`  This is a comment`

示例

（在示例项目 [▶ 268]中：B_DocuElements\Comments\FB_Libdoc_InternalComments）

```
(*
This is the first paragraph.

.. Comments begin with two dots and one space. Anything can
   follow, except for the syntax of footnotes/citations,
   hyperlink targets, directives, or substitution definitions.

Since comments are not shown, this paragraph follows the previous paragraph directly.
*)
```

This is the first paragraph.

Since comments are not shown, this paragraph follows the previous paragraph directly.

13.6.2.14 字体样式

内联标记允许在库管理器中以粗体、斜体或等宽字体显示单词或短语。

（在示例项目 [▶ 268]中：B_DocuElements\Font style\FB_Libdoc_FontStyle）

突出显示文本（斜体）

用单个星号字符括起来的文本以斜体突出显示。

开始字符和结束字符	开始字符 = 结束字符 = “*”
示例	`This is *emphasized text*.` This is *emphasized text*.

突出显示文本（粗体）

用双星号括起来的文本以粗体突出显示。

开始字符和结束字符	开始字符 = 结束字符 = “**”。
示例	`This is **strong text**.` This is **strong text**.

内联字面量（等宽字体）

由双引号括起来的文本以等宽字体显示（打字稿）。

内联字面量可用于短代码范围或内联代码。

开始字符和结束字符	开始字符 = 结束字符 = “``””
示例	`This text is an example of ``inline literals``.` This text is an example of `inline literals`.

13.7 其他命令和对话框

13.7.1 命令添加库

功能：该命令可打开 Add library（添加库）对话框。在此对话框中，您可以将库引用以占位符形式添加到库管理器中，从而将其集成到应用程序中。

调用：PLC 项目树中 References（引用）对象的上下文菜单

前提条件：库已安装在库存储库 [▶ 251]的本地系统中。

占位符与 库

通过使用库占位符，对所使用的库版本的调整变得几乎毫不费力，从而使工程项目和库的流程变得非常灵活。在项目或库项目中集成库引用时，建议使用占位符，而不是库。这意味着在 PLC 项目中没有集成特定的库，而是集成占位符。然后，将 1 个“真正的”库分配给占位符。

有关更多信息，请参见：库占位符 [▶ 257]。

由于占位符的优势，建议使用 命令添加库 [▶ 329]，而不是 添加无占位符解析的库命令 [▶ 330]。

添加库对话框

Add library（添加库）命令会打开 1 个对话框，通过该对话框可以将库引用作为占位符集成到项目中，默认情况下会带有相应的占位符名称。前提条件是在创建库时 [▶ 249]已发布显式或隐式占位符名称（如果在创建库时未输入显式占位符名称，则在默认情况下会将输入的库标题作为占位符名称）。如果要集成的库不存在默认占位符名称（例如，对于没有发布显式占位符名称的旧库，可能会出现这种情况），则会将库引用作为库进行集成。

通过此对话框添加的库引用将作为占位符集成到项目中；默认情况下，占位符将作为“始终最新版本”（“ * ”）进行解析，即没有固定版本。通过占位符对话框 [▶ 258]或属性窗口 [▶ 333]可以调整版本解析。

在库列表上方的行中，您可以通过键入相应的字符串的方式搜索库名称或库功能块。双击搜索结果或聚焦搜索结果 + [OK]，即可将与搜索结果相关的占位符包含在项目或库管理器 [▶ 254]中。	
库	在库存储库中已安装的所有库的列表。 您可以使用对话框右上角的按钮来更改列表的类型： • 按库类别（“类别视图”）排列：在按库类别排序时，类别会显示为节点。点击节点可以打开相应库或子类别的列表。 • 按库名称（“列表视图”）字母顺序排列 通过双击或聚焦 + [OK] 的方式，您可以将所需的占位符集成到项目或库管理器 [▶ 254]中。
公司	库提供者

另请参见：

- 库占位符 [▶ 257]
- 库存储库 [▶ 251]
- TwinCAT 3 用户界面\参考用户界面\上下文菜单 PLC 项目
  - 。
  - 。
  - 。

13.7.2 添加无占位符解析的库命令

功能：该命令可打开 **Find library**（查找库）对话框。在此对话框中，您可以将库引用以库的形式添加到库管理器中，从而将其集成到应用程序中。

调用：PLC 项目树中 **References**（引用）对象的上下文菜单

前提条件：库已安装在库存储库 [▶ 251]的本地系统中。

占位符与 库

通过使用库占位符，对所使用的库版本的调整变得几乎毫不费力，从而使工程项目和库的流程变得非常灵活。在项目或库项目中集成库引用时，建议使用占位符，而不是库。这意味着在 PLC 项目中没有集成特定的库，而是集成占位符。然后，将 1 个“真正的”库分配给占位符。

有关更多信息，请参见：库占位符 [▶ 257]。

由于占位符的优势，建议使用 命令添加库 [▶ 329]，而不是 添加无占位符解析的库命令 [▶ 330]。

另请参见：

- 命令添加库 [▶ 329]

查找库对话框

Add library without placeholder resolution（添加无占位符解析的库）命令会打开 1 个对话框，通过该对话框可以将库引用作为库（而不是占位符）包含在项目中。

在库列表上方的行中，您可以通过键入相应的字符串的方式搜索库名称或库功能块。双击搜索结果或聚焦搜索结果 + [OK]，即可将与搜索结果相关的库包含在项目或库管理器 [▶ 254]中。	
公司	按制造商筛选列表。
按类别分组	☑：以树状结构显示按类别分组的库。在按库类别排序时，类别会显示为节点。点击节点可以打开相应库或子类别的列表。 ☐：以扁平结构按字母顺序显示库。
显示所有版本	☑：显示所有库版本。版本规格“*”表示存储库中的最新版本可用。 ☐：只显示最新的库版本。在这种显示中，您可以实现多个库选择：在选择条目时，按住 [Shift] 键。
详细信息	打开详细信息 [▶ 332]对话框。
库存储库	打开 库存储库 [▶ 251] 对话框。如要添加在本地系统中尚未安装的库，您可以使用此按钮打开库存储库进行安装。

另请参见：

- 库占位符 [▶ 257]
- 库存储库 [▶ 251]
- TwinCAT 3 用户界面\用户界面参考\PLC 项目上下文菜单
  - 。
  - 。
  - 。

13.7.3 命令尝试重新加载库

功能：该命令可尝试重新加载所选库。

调用：

- PLC 项目树中库的上下文菜单
- 库管理器 [▶ 254] 中的按钮

要求：在打开项目时加载库失败。在 PLC 项目树或库管理器的编辑器中，选择加载失败的库。

如果由于某种原因而在打开项目时无法在定义的存储库位置看到可用的库，则会发出相应的错误消息。当您已校正故障且库再次可用时，您可以使用该命令重新加载此库，而无需退出项目。

13.7.4 命令删除库

功能：该命令可从库管理器中删除所选的库。

调用：

- PLC 项目树中库的上下文菜单
- 库管理器 [▶ 254] 中的按钮（符号： ）

要求：库在库管理器中可用。

另请参见：

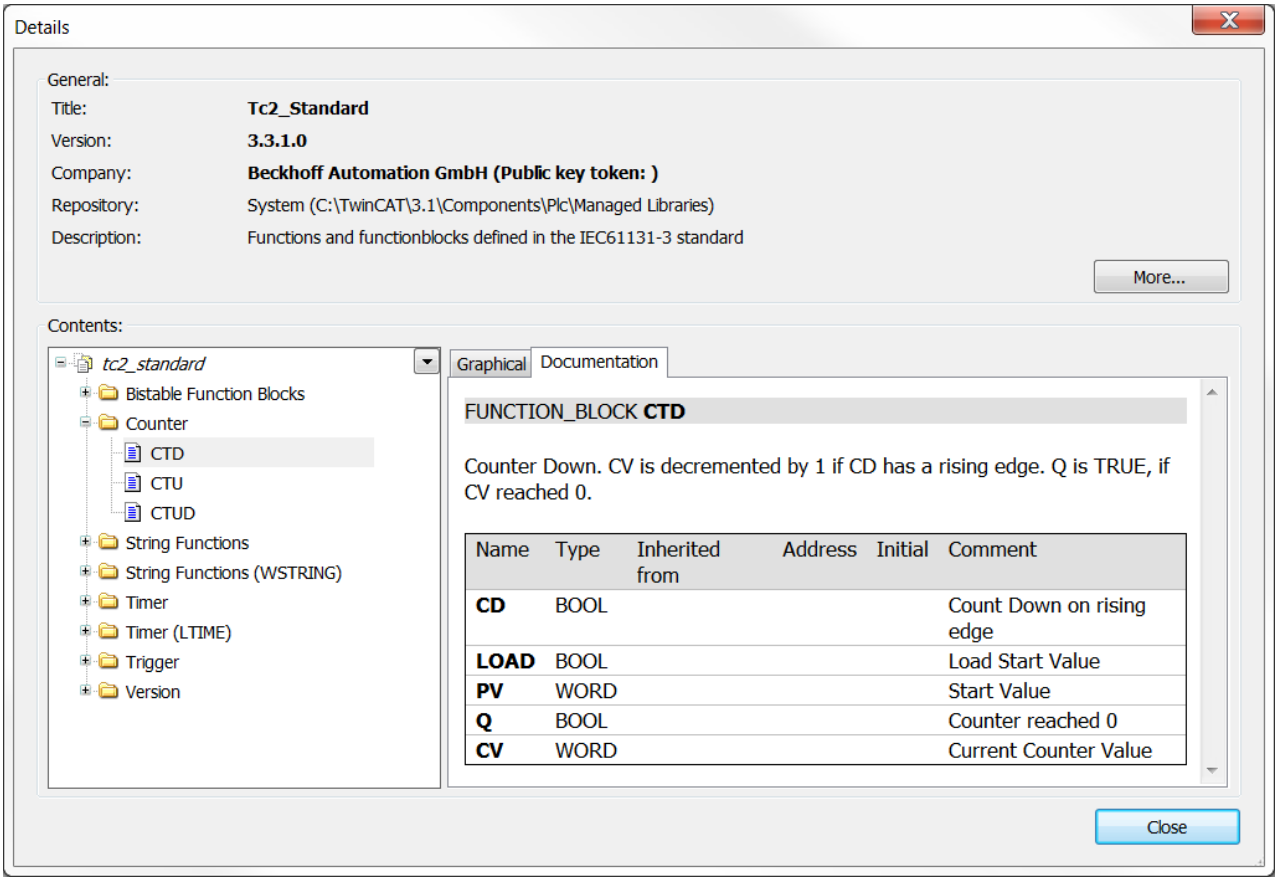
- TwinCAT 3 用户界面\参考用户界面\上下文菜单 PLC 项目
  - 。
  - 。
  - 。

13.7.5 命令详细信息

功能：该命令可打开 Details（详细信息）对话框，其中包含所选库版本的库信息。

调用：

- PLC 项目树中库的上下文菜单
- 库存储库 [▶ 251]中的按钮
- 库管理器 [▶ 254] 中的按钮（符号： ）



使用 More…（更多……）按钮，可获得以下信息：

- 大小：以字节为单位的规格
- 已创建：创建日期
- 已修改：最后修改的日期
- 最后访问：日期
- 特性
- 属性

13.7.6 命令相关性

功能：该命令可为所选库版本打开 Dependencies（相关性）对话框，以显示所选库的相关性。

调用：

- PLC 项目树中库的上下文菜单
- 库存储库 [▶ 251] 中的按钮

如果所选库中已包含或已引用其他库，则会在对话框中列出它们。对于每个库引用，都显示了标题、版本和公司。使用占位符的引用以语法 #<PlaceholderName> 表示。

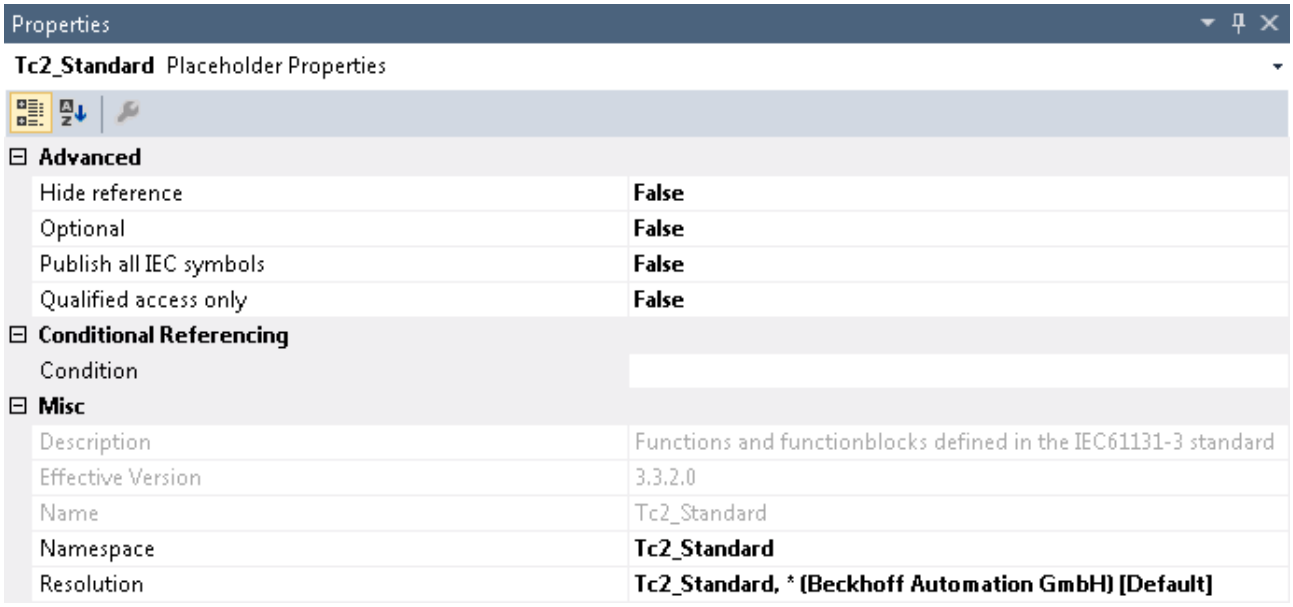
13.7.7 命令属性

功能：该命令可激活所选库的 Properties（属性）视图，以配置设置。

调用：

- PLC 项目树中库的上下文菜单
- 如果 Properties（属性）窗口已经打开：在 PLC 项目树中选择库

属性视图



高级

一旦将库集成（引用）到另一个库中，下列设置就会受到关注。默认情况下，它们已禁用：	
隐藏引用	TRUE：如果当前项目在另一个项目中作为库引用，则该库引用不会在相关性树中显示，因此也不会在库管理器中显示。因此，您可以添加“隐藏”库。对于因库错误而导致的编译错误，可能很难找到具体原因。 FALSE：如果当前项目在另一个项目中作为库引用，则该库引用会在依赖树中显示，因此也不会在库管理器中显示。
可选	TRUE：所选库被视为可选库。在加载引用库的项目时，即使该库不存在于库存储库中，也不会输出任何错误。
发布所有 IEC 符号	TRUE：如果当前项目后来作为库包含在另一个项目中，则会在此项目中发布所选库引用的 IEC 符号，就像所选库引用直接包含在该项目中一样。如果您使用相应的命名空间路径，则您可以明确地寻址库功能块。这由“父级”库的命名空间及其自身的命名空间组成，位于功能块名称之前。 只有当您想要创建所谓的“容器库”时，才应启用该选项。容器库是 1 种不定义自身功能块的库，它只引用其他库来创建 1 种库束。如果您需要在 1 个项目中同时包含多个库，这将非常有用。不过，在这种情况下，最好将顶级包的各个库放在项目的库管理器中，以缩短库功能块的访问路径。这正是您通过提前启用该选项所能达到的效果。 FALSE：如果当前项目在另一个项目中作为库引用，则通过使用相应的命名空间可以明确寻址在属性中显示的库引用的功能块。命名空间由当前库项目的命名空间和所选库引用的命名空间组成。
仅限定访问	TRUE：只有使用命名空间前缀，您才能在项目中访问库的功能块和变量。

条件引用

条件	您可以在此处添加与在 PLC 项目中设置的编译器定义进行比较的条目。如果至少有 1 个条目对应于其中 1 个定义，则会主动引用库。如果没有与任何定义对应的条目，则库将被停用并显示为灰色。 如果您想要输入多个条目，则可以用逗号将它们分开。 如果不存在任何条目，则此设置无效。 示例：def1,def2,def3

其它

如果不是库占位符，则您可以在此处（重新）定义在项目中使用的库版本，或者您可以编辑占位符解析。 尽可能避免使用固定库。相反，应使用占位符来引用库。	
描述	库描述
有效版本	占位符所指向的库的有效版本。如果在 **Resolution**（解析）字段中已配置设置“始终最新”/“*”，则会显示该字段。尽管解析设置为通用的“始终最新”，但这让用户能够看到实际使用的库版本。
名称	库名称
命名空间	显示当前的命名空间。默认情况下，这与库名称相同，除非您在创建库时在项目信息中明确定义了不同的默认命名空间。在“Properties”（属性）视图中，您可以在此处更改本地项目的命名空间。
解析	如果您在库管理器中选择库占位符，则该字段包含要替换占位符的库的名称和版本。此时，占位符会被解析为“真正的”库，并被解析为库的特定版本（例如，“1.0.0.0”）或“始终最新”/“*”。 如果您想要特定版本的库，可从列表中选择它，TwinCAT 将完全使用该版本。 如果您想要“始终最新”/“*”版本的库，可从列表中选择该选项，TwinCAT 总是会使用在库存储库中找到的最新版本的库。当有更新版本的库可用时，这可能会导致实际使用的库模块发生变化。

另请参见：

- 使用库 [▶ 246]
- 占位符 [▶ 258]

13.7.8 命令设置为有效版本

功能：根据是在 PLC 项目树中还是在单个库中选择 **References**（引用）对象的情况，该命令会将项目中可用的所有占位符和库或者所选的库或占位符设置为有效版本。

调用：

- PLC 项目树中 **References**（引用）对象的上下文菜单，用于将项目中可用的所有占位符和库设置为有效版本
- PLC 项目树中库的上下文菜单，可将所选的库或占位符设置为有效版本

如果您将该命令应用于项目中可用的所有占位符和库，则未在引用中列出的内部使用的库也将被设置为有效版本。

示例：例如，如果 1 个库先前作为“始终最新”/“*”使用且使用的是库版本 3.3.10.0，则通过命令 **Set to Effective Version**（设置为有效版本），该库不再作为“始终最新”引用，而是作为固定版本 3.3.10.0 引用。

13.7.9 命令设置为始终最新版本

功能：根据是在 PLC 项目树中还是在单个库中选择 **References**（引用）对象的情况，该命令会将项目中可用的所有占位符和库或者所选的库或占位符设置为“始终最新”版本（“*”）。这意味着 TwinCAT 总是会使用在库存储库中找到的最新版本的相应库。

调用：

- PLC 项目树中 **References**（引用）对象的上下文菜单，用于将项目中可用的所有占位符和库设置为“始终最新”版本
- PLC 项目树中库的上下文菜单，可将所选的库或占位符设置为“始终最新”版本

如果您将该命令应用于项目中可用的所有占位符和库，则未在引用中列出的内部使用的库也将被设置为最新版本（*）。

示例：在 1 个项目中，库“Lib1”作为“始终最新”（“*”）使用。在使用开发主机 A 的库存储库中，库版本 3.0.0.0 和 3.1.2.0 已安装。开发计算机 B 的库存储库已安装库版本 3.0.1.0。
如果在开发计算机 A 上打开所述项目，则会引用“Lib1”的版本 3.1.2.0。如果在开发计算机 B 上使用同一项目，则会使用库版本 3.0.1.0。

14 PLC 中的多任务数据访问同步

当多个任务访问 1 组数据时，这些任务可能会同时访问相同的数据，具体取决于任务/实时配置。如果数据由至少 1 个任务写入，那么在更改期间或之后，数据可能会出现不一致的状态。为了防止出现这种情况，务必同步所有并发访问，以便每次只能有 1 个任务访问共享数据。

例如，需要同步的来自多个任务的并发访问包括以下情况：

- 直接访问全局变量或其他非临时变量，例如，使用运算符
- 间接访问全局变量或其他非临时变量，例如，在函数、方法或其他 POU 调用中（尤其是在功能块实例被全局实例化的情况下）

简而言之：如果多个任务访问 1 组数据，且其中至少有 1 个访问会写入数据，则所有读取和写入访问必须同步。无论任务是在单核还是多核 CPU 上运行，这一点都适用。

⚠ 警告

由于意外的轴运动而导致受伤的风险

如果不同步并发访问，则会存在数据记录不一致或无效的风险。根据在程序的后续阶段中使用数据的方式，这可能会导致错误的程序行为、意外的轴运动，甚至是程序突然进入停滞状态。根据不同的受控系统，可能会损坏设备和工件，或者危及人员健康和生命。

- 同步并发访问。
- 您将会在 MUTEX 程序的示例程序中找到功能测试及相应的说明，从而了解同步访问的必要性。

同步选项

下列选项可用于同步访问：

- Mutex 程序（TestAndSet、FB_IecCriticalSection）用于确保临界区的安全 [▶ 337]
  - 务必始终保持尽可能少的临界区数量。
  - 临界区必须保持简短。
  - 对于相对较短的临界区，通常建议使用 FB_IecCriticalSection [▶ 341]。
  - 对于相对较长的临界区，通常建议使用 TestAndSet [▶ 340]。
- 通过 PLC 过程映像进行数据交换 [▶ 342]
  - 只有在仅有 1 个任务可以对相同数据进行写入访问的情况下，才有可能并建议使用这种变体。
  - 由于需要执行内部复制操作，可能的数据量会受到限制。有关详细信息，请参见“通过 PLC 过程映像进行数据交换”下的说明。
- 通过同步缓冲区进行数据交换 [▶ 343]
  - 只有在仅有 1 个任务可以对相同数据进行写入访问的情况下，才有可能使用这种变体。
  - 这是 1 种用户特定的实现方式，提供有多种选择。
  - 务必确保对单个缓冲区的访问安全，例如，使用 TestAndSet() 的情况。
  - 由于执行复制操作，可能的数据量会受到限制。有关详细信息，请参见“通过同步缓冲区进行数据交换”下的说明。

在原子访问的情况下也可以同步

即使对变量的单次访问（例如，写入 1 个整数）可以被描述为原子访问（即不间断访问），通常也有必要进行同步。

由于原子访问的属性取决于所使用的处理器架构等因素，因此，为了简单和安全起见，每次访问都应被视为非原子访问。

同样应该注意到的是，如果更仔细地考虑，即使是所谓的安全访问，也几乎总是不安全的。下面将通过 2 个示例场景对此进行说明：

- 通常情况下，所需的函数表达式（例如，读取、更改、写入）需要多次原子访问。如果这种多部分函数表达式存在于多个任务中，在多个任务中同时执行表达式可能会导致与顺序执行表达式不同的结果。

  - 示例：1 个全局计数器变量（初始化为 0）要递增 1（从 2 个任务开始）（nGlobal := nGlobal + 1;）。如果同时执行递增，则 2 次都会计算 0 + 1 = 1 且全局变量的结果值为 1，但该值本应为 2。
- 在执行任务期间，在不同时间进行的多次读取访问可能会产生不同的变量值。
  - 示例：从 1 个任务中写入 1 个全局变量。该值先前为 50，现在被写为 0（nGlobal := 0;）。在不同的任务上下文中，对该全局变量的值进行查询（IF nGlobal>10 AND nGlobal<20 THEN）。查询由 2 次读取访问组成。如果在这 2 次读取访问之间，从其他任务中进行上述写入访问，那么，即使该全局变量的值在任何时候都不在 10 到 20 之间，也已满足条件。

补充说明

- 诸如 ADD、SIZEOF 或 __NEW 等系统运算符可以同时被多个任务调用。该语句仅指所使用的运算符。如果在调用期间访问数据，而这些数据又被多个任务使用，则必须相应地同步这些数据访问。
- 在内部使用 ADS 的功能块实例不能用于不同的任务。不过，如果从其他任务中调用该实例，则会出现错误 ADSERR_DEVICE_INVALIDCONTEXT=0x709=1801。
- 由于在堆栈上执行，因而在 TwinCAT 3 中在不同任务中使用 STRING 函数（Tc2_Standard 库）并不重要（在 TwinCAT 2 中，STRING 函数在默认情况下对任务更改而言并不安全）。
- 还要注意编译器扩展（文档 TE1200 TC3 PLC 静态分析）的可能性，它提供了有关“并发/竞争访问”主题的规则。这应被视为确定潜在同步要求的工具。

14.1 Mutex 程序（TestAndSet、FB_IecCriticalSection）用于确保临界区的安全

在使用 Mutex 程序或实现互斥时，发生竞争访问的部分被定义为临界区。使用函数 TestAndSet() [▶ 340] 或功能块 FB_IecCriticalSection [▶ 341]（两者均来自 PLC 库 Tc2_System）可以对这些部分进行同步，从而将这些部分置于互斥条件下，每次仅有 1 个任务可以访问共享数据。

进入临界区可能取决于 1 个或多个条件。此外，不同的临界区可能取决于不同的条件。

示例

- 如果部分 1a 和部分 1b 分别对“Data1”进行读取和写入，则部分 1a 和部分 1b 必须相互同步。如果部分 2a、2b 和 2c 单独对“Data2”数据进行读取和写入访问，则部分 2a、2b 和 2c 必须相互同步。范围 1x 和 2x 均仅取决于 1 个条件（“Data1”或“Data2”未锁定）。此外，在这种情况下，它们并不相互依存，因为每个部分访问的数据不同。这意味着，即使部分 1b 锁定“Data1”数据，部分 2a 也可以同时锁定并访问“Data2”数据。
  因此，在以下临界区之间必须对数据资源进行同步：
  - 在部分 1a 和 1b 之间使用资源“Data1”
  - 在部分 2a、2b 和 2c 之间使用资源“Data2”
- 如果部分 1x 和 2x 额外访问“DataA”数据（至少进行 1 次写入访问），情况就会改变。
  然后，所有部分 1x 和 2x 均取决于 2 个条件：如要进入部分 1x，则必须启用数据“Data1”和“DataA”；如要进入部分 2x，则必须启用数据“Data2”和“DataA”。
  此外，由于共享使用“DataA”数据，部分 1x 和 2x 现在相互依存，不能再同时执行。
  因此，在以下临界区之间必须对数据资源进行同步：
  - 在部分 1a 和 1b 之间使用资源“Data1”
  - 在部分 2a、2b 和 2c 之间使用资源“Data2”
  - 在所有部分（1a、1b、2a、2b、2c）之间使用“DataA”资源

使用 TestAndSet() 函数时，标志（Boolean 变量）代表进入临界区所依赖的条件（例如，bLockData1）。使用 FB_IecCriticalSection 时，1 个功能块实例可用作锁定条件。

阻塞

- TestAndSet()：该函数可用于选择和检查临界区的内容。但是，该功能不具有阻塞效果，可能会出现在 1 个周期内无法处理该部分的情况。
- FB_IecCriticalSection：如果其他任务尝试通过调用 Enter() 方法来访问已占用的临界区，则会被 TwinCAT 调度程序阻塞。**在再次启用该部分之前，任务阻塞会一直存在。**

⚠ 谨慎

由于任务停止而导致周期超时

任务阻塞的持续时间可能会导致任务超出循环时间，具体取决于循环时间（的利用情况）。

- 确保临界区保持简短，以避免等待任务的周期超限。如果多个任务正在等待进入临界区，则应根据其优先级授予访问权限。

因此，与功能块 FB_IecCriticalSection 相比，TestAndSet() 函数的优势在于，在任何情况下都不会发生任务阻塞。其缺点为，如果函数 TestAndSet() 不授予访问权限，则必须提供其他实现方法。例如，通过状态机可以实现这一点，以便在下一个周期中再次尝试访问。

死锁

请注意，如果功能块 FB_IecCriticalSection 使用不当，则任务可能会被锁定，从而发生死锁。死锁总是涉及至少 2 个任务。当任务互相等待对方释放另一个任务已经锁定的资源时，就会出现这种情况。

⚠ 谨慎

由于死锁而导致永久性任务停滞

一旦出现上述死锁现象，就无法再通过编程进行消除。相关任务将进入永久性停滞状态。

示例：

- 任务 1 和任务 2 都需要访问“DataA”和“DataB”数据。
- 任务 1 最初锁定资源“DataA”，任务 2 同时锁定资源“DataB”。
- 然后，任务 1 想要锁定“DataB”数据。由于该资源已被任务 2 锁定，因此无法实现这一点，而任务 1 已被锁定。
- 而除了“DataB”之外，任务 2 还要尝试锁定“DataA”。由于该数据已被任务 1 锁定，因此也无法实现此项锁定，而任务 2 也被锁定。
- 因此，这 2 个任务均已被锁定。然后，任务等待对方释放已被其锁定的数据，但由于其自身锁定而无法释放数据。因此，这 2 个任务会无限期地相互等待。

即使同步不正确，死锁情况也不是不可避免的。只有当进程以不利的顺序随机出现时，才会发生死锁。

避免死锁

一般来说，如果每个任务 1 次只想锁定 1 个资源，您就可以避免死锁。

您还可以通过仅按照一定的顺序请求和锁定资源来避免死锁。

示例：

- 上述示例中的问题在于，任务 1 和任务 2 按照不同的顺序请求并锁定“DataA”和“DataB”资源。
- 不过，如果 2 个任务总是按照相同的顺序锁定资源，就可以避免死锁问题。这意味着：如果每个需要访问临界区中的“DataA”和“DataB”的任务总是首先请求“DataA”并锁定它（如果可能），并且仅在成功锁定“DataA”之后才请求“DataB”资源并锁定它（如果可能），那么就不会出现上述示例中的任务按照不同的顺序“抢走”所需资源的死锁现象。

示例程序：使用 TestAndSet() 实现访问同步

本示例展示了如何从不同的任务上下文中安全地访问共享数据。数据合并到 1 个结构实例中。其中还包含 1 个 Boolean 变量 bLocked 作为测试标志。

在您读取或写入该全局结构实例的数据之前，您必须通过调用带有相应测试标志的函数 TestAndSet() [▶ 340] 来请求对其进行访问。如果未被授予访问权限，则您无法访问此周期中的数据。在这种情况下，必须提供替代处理方法。如有必要，可在下一个周期中再次请求访问。如果已被授予访问权限且已成功调用 TestAndSet()，则您可以读取或修改数据。在您完成编辑后，请将测试标记设为 FALSE，以再次释放访问权限。

功能测试

启动示例程序。在 MAIN1 和 MAIN2 中有 1 个计数器变量（nLocalBlockedCounter），如果 TestAndSet() 调用失败，该计数器变量就会递增。如果计数器为 0，则表示每次尝试执行对全局变量的数据访问均已成功。

如要了解访问同步的必要性，您可以从不同的 CPU 内核运行这 2 个任务。如果您现在启动示例程序，就会看到计数器变量如何不规律地递增，这表明数据访问偶尔会受阻。

访问同步不仅在多核使用的情况下十分必要，而且，当 2 个任务在同一个 CPU 上运行时也是如此。任务相互干扰也可能会导致在未保存的数据访问中出现严重的不一致。

下载：https://infosys.beckhoff.com/content/1033/tc3_plc_intro/Resources/7644032779.zip。

示例程序：使用 FB_IecCriticalSection 实现访问同步

Windows CE

对于 TwinCAT v3.1.4022.29 及以上版本，在 Windows CE 操作系统下支持 FB_IecCriticalSection 的功能。

该示例显示了在使用现金账户转账时，在 PLC 中使用临界区的情况。账户由功能块 FB_Account 表示。涉及 4 个账户。所有 4 个功能块实例均在 1 个全局变量列表中声明，以便从不同的任务上下文中进行访问（此处：转账）。

每个账户的初始余额为 1000。使用 Get() 和 Set() 方法可以读取和重置每个账户的余额。在 4 种不同的任务上下文中可以实施以下转账。

任务 1：A->B 500

任务 2：B->C 250，B->D 250

任务 3：C->A 250

任务 4：E->A 250

在完成上述转账之后，每个账户的余额应为 1000。

务必确保绝对不会发生同时从 2 个任务上下文中访问 1 个账户的情况。

在 FB_Account 中使用功能块 FB_IecCriticalSection [▶ 341]。方法 FB_Account.Lock() 在临界区执行 Enter()，方法 FB_Account.Unlock() 在临界区执行 Leave()。在访问账户余额之前，必须要成功执行 Lock() 方法。然后才能阻止其他功能块访问该账户。

如要避免死锁，可定义锁定顺序。其定义如下：

锁定顺序：A 先于 B 先于 C 先于 D

这将导致以下解锁顺序：D 先于 C 先于 B 先于 A

在任务 3 中实施转账：

```
(* Task 3: C->A 250*)
IF GVL.fbDepotA.Lock() THEN
    IF GVL.fbDepotC.Lock () THEN
        GVL.fbDepotC.Set (GVL.fbDepotC.Get() - 250);
        GVL.fbDepotA.Set (GVL.fbDepotA.Get() + 250);
        GVL.fbDepotC.Unlock();
    END_IF
    GVL.fbDepotA.Unlock();
END_IF
```

功能测试

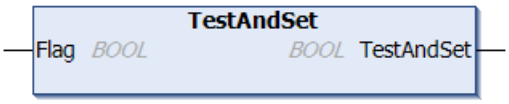
启动示例程序。在 Main1 中，您可以看到所有账户余额的总和，在 PLC 在线视图中，该余额总和必须始终为 4000。

如要了解在本例中使用临界区的必要性，您可以从不同的 CPU 中运行这 4 个任务，并将全局变量 bIgnoreLock 设置为 TRUE。如果您现在启动示例程序，您将看到账户余额如何产生不正确的值，而且，所有账户余额的总和也会表明转账出现故障。

访问同步不仅在多核使用的情况下十分必要，而且，当 4 个任务在同一个 CPU 内核上运行时也是如此。任务相互干扰也可能会导致在未保存的数据访问中出现严重的不一致。

下载：https://infosys.beckhoff.com/content/1033/tc3_plc_intro/Resources/7643978251.zip

14.1.1 TestAndSet



您可以使用该函数来检查和设置标志。没有中断进程的选项。这样可以同步数据访问。通过 TestAndSet 可以实现信号的运行模式。

如果函数调用成功，则函数会返回 TRUE，并且可以访问所需的数据。如果函数调用不成功，则函数会返回 FALSE，并且无法访问所需的数据。在这种情况下，必须提供替代处理方法。

输入/输出

```
VAR_IN_OUT
    Flag : BOOL; (* Flag to check if TRUE or FALSE *)
END_VAR
```

名称	类型	描述
标志	BOOL	要检查的 Boolean 标志 • 如果它为 FALSE，则标志尚未被分配并会被设置（因此，从现在起被阻塞），而且函数会返回 TRUE • 如果它为 TRUE，则标志已被分配（因此被阻塞），而且函数会返回 FALSE

示例

```
VAR_GLOBAL
    bGlobalTestFlag : BOOL;
END_VAR
```

```
VAR
    nLocalBlockedCounter : DINT;
END_VAR
```

```
IF TestAndSet(GVL.bGlobalTestFlag) THEN
    (* bGlobalTestFlag was FALSE, nobody was blocking, NOW
    bGlobalTestFlag is set to TRUE and blocking others *)

    (* ... *)

    (* remove blocking by resetting the flag *)
    GVL.bGlobalTestFlag := FALSE;
ELSE
    (* bGlobalTestFlag was TRUE, somebody is blocking *)
    nLocalBlockedCounter := nLocalBlockedCounter + 1;

    (* ... *)
END_IF
```

NEGATIVE 示例

在进一步封装（例如，在功能块中）时应谨慎，因为这可能会破坏所需的原子操作。这样就无法再进行数据访问的安全同步。下面包含的 NEGATIVE 示例说明了不得如何使用该功能。如果在此项实现中 2 个上下文同时请求访问数据，那么这 2 个上下文都会认为可以进行访问，从而同时对数据进行不安全的访问。

```
FUNCTION_BLOCK FB_MyGlobalLock
VAR_INPUT
    bLock    : BOOL; // set TRUE to lock & set FALSE to unlock
END_VAR
VAR_OUTPUT
    bLocked : BOOL;
END_VAR

IF bLock THEN
    TestAndSet(bLocked);
ELSE // unlock
    bLocked := FALSE;
END_IF
```

```
IF NOT GVL.fbGlobalLock.bLocked THEN
    GVL.fbGlobalLock(bLock := TRUE);
```

```
    (* ... *)

    GVL.fbGlobalLock(bLock := FALSE);
END_IF
```

功能块 FB_IecCriticalSection [▶ 341] 可提供临界区的应用，作为 1 种替代的 Mutex 方法。

要求

开发环境	目标系统类型	要包括的 PLC 库（类别组）
TwinCAT v3.1.0	PC 或 CX（x86、x64、ARM）	Tc2_System（System）

14.1.2 FB_IecCriticalSection

该功能块可用于使临界区相互排斥。临界区的特点是修改会影响 1 个或多个变量（通常情况下），而这些变量在修改期间具有不一致的状态。因此，每次必须仅由 1 个任务执行此类修改。为此，功能块可提供方法 Enter() 和 Leave()。成功调用 Enter() 可访问临界区，然后该区被视为已被占用。修改完成后，必须通过 Leave() 退出临界区。

由于任务停止而导致周期超时

如果其他任务尝试通过 Enter() 调用来访问已占用的临界区，则会被 TwinCAT 调度程序阻塞。在再次启用该部分之前，任务阻塞会一直存在！一旦启用，将会继续处理程序代码，并进入临界区。

- 确保临界区保持简短，以避免等待任务的周期超限。如果多个任务正在等待进入临界区，则应根据其优先级授予访问权限。

如果任务因尝试进入已占用的临界区而被 TwinCAT 调度程序阻塞，则可在没有“忙等待”的情况下实现这一点。因此，低优先级任务可以在这段时间内使用 CPU 容量。

Windows CE

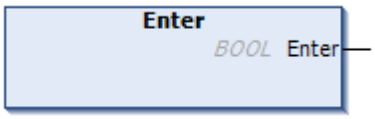
对于 TwinCAT v3.1.4022.29 及以上版本，在 Windows CE 操作系统下支持该功能。（在旧版 TwinCAT 中，这些方法会返回 FALSE。）

替代方案

通过函数 TestAndSet() [▶ 340] 也可以实现临界区。该函数可用于选择和检查临界区的内容。但是，该功能不具有阻塞效果，可能会出现在 1 个周期内无法处理该部分的情况。

通常情况下，应该保持尽可能少的临界区数量和长度。

Enter() 方法



该方法标志着临界区的开始。

可能的返回值：

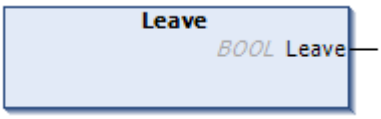
TRUE：

- 可以进入临界区。

FALSE：

- 不可以进入临界区。
- 运行时尚不支持该功能块。
- 临界区被另一个 PLC 任务占用。该任务在断点处停止。返回值 FALSE 可以避免任务的永久性阻塞，并确保 I/O 的更新。

Leave() 方法



该方法标志着临界区的结束。务必始终在临界区完成时调用它。

可能的返回值：

TRUE：

- 已成功退出该部分。

FALSE：

- 运行时不支持该功能块。
- 临界区没有被 Enter 占用。

功能块的应用示例：

功能块 FB_IecCriticalSection 可实现对要保护的共享文件的访问。在全局范围内创建功能块实例和要保护的数据。

```
VAR_GLOBAL
    fbCrititcalSection : FB_IecCriticalSection;
END_VAR

IF fbCrititcalSection.Enter() THEN
    (* start of critical section *)

    (* end of critical section *)
    fbCrititcalSection.Leave();
END_IF
```

要求

开发环境	目标平台	要集成的 PLC 库（类别组）
TwinCAT v3.1.4020	PC 或 CX（x86、x64） WES、WES7、Win7、Win10	Tc2_System（系统）
TwinCAT v3.1.4022.29	PC 或 CX（x86、ARM） WinCE	Tc2_System（系统）

14.2 通过 PLC 过程映像进行数据交换

通常从 PLC 过程映像与 IO 过程映像进行数据交换。同样，也可以在 PLC 内部的 2 个任务上下文之间交换数据。

该方法适用的 1 个基本条件是，只有 1 个任务应写入相同的数据。然后，对数据进行声明，使其成为该任务输出过程映像的一部分。

示例

- 从任务上下文 1 和任务上下文 2 可以分别读取以及写入“DataA”。
- 此外，从任务上下文 2 和任务上下文 1 可以分别读取和写入以及访问“DataB”。
- 实现：
  - 您可以分别为输出过程映像中的任务上下文 1 以及输入过程映像中的任务上下文 2 定义“DataA”。
  - 因此，您可以分别为输出过程映像中的任务上下文 2 以及输入过程映像中的任务上下文 1 定义“DataB”。
  - 您可以直接在使用“DataA”和“DataB”的地方声明它们，而不是在全局范围内声明。
  - 在 TwinCAT XAE 的 PLC 实例下方的过程映像显示中，您可以将“DataA”相互关联并将“DataB”相互关联。

只有在其他任务上下文中需要的数据才能被添加到输出过程映像中，以避免过程映像变得过于庞大。异步映射在每个周期内执行复制操作，以便在 PLC 过程映像和附加的临时缓冲区之间交换数据。由于使用大数据块的复制操作需要大量的计算时间（调用 MEMCPY 函数时也是如此），这通常会将可能的数据量限制在明显小于 1 MB 的范围内。映射由相关的任务本身执行，因此相应的复杂映射会减少实际程序处理的任务运行时。

该方法适用的第 2 个条件是，数据不得是功能块实例或任何类型的指针（POINTER TO、接口指针/接口变量、引用.....）。因此，无法在 2 个不同的任务上下文中调用同一个功能块实例的方法。

如有必要，可将所需数据合并为 1 个结构。

示例程序：通过 PLC 过程映像交换数据实现同步

该示例显示了 PLC 实例中慢任务和快任务之间的数据交换。

以任意数据块（无功能块实例或指针）为例，1 个整数变量被传输到另一个任务。它还展示了您可以如何使用 Boolean 变量在其他任务上下文中触发函数。

下载：https://infosys.beckhoff.com/content/1033/tc3_plc_intro/Resources/7643979915.zip

14.3 通过同步缓冲区进行数据交换

该同步选项是 1 种用户特定的实现方式。因此，不会使用库的特殊功能块。程序代码的复杂程度更高，也取决于所选择的实现方式。

该方法适用的 1 个基本条件是，只有 1 个任务应写入相同的数据。

通过附加的临时数据缓冲器可以同步对数据的写入和读取访问。如要同步这些数据缓冲区，需要使用 MUTEX 程序 [▶ 337]。不过，与独家直接使用 TestAndSet() 相比，该方法的优势在于，始终允许访问其中 1 个附加的数据缓冲区，并且无需进行任何替代性处理。

该方法适用的第 2 个条件是，数据不得是功能块实例或任何类型的指针（POINTER TO、接口指针/接口变量、引用.....）。因此，无法在 2 个不同的任务上下文中调用同一个功能块实例的方法。

您只能定义在其他任务上下文中需要的数据交换的数据，以避免数据量变得过于庞大。实现会执行复制操作，以便在缓冲区之间交换数据。由于使用大数据块的复制操作需要大量的计算时间（调用 MEMCPY 函数），这通常会将可能的数据量限制在明显小于 1 MB 的范围内。

该同步选项所需的方法由用户实现。以不同的方法可以执行该特定实现。一些术语（例如，3 路缓冲区或 3 缓冲原理）也可用于表示实现类型。

示例程序：通过同步缓冲区交换数据实现同步

该示例显示了 PLC 实例中慢任务和快任务之间的数据交换。

以任意数据块（无功能块实例或指针）为例，2 个结构 ST_DataA 和 ST_DataB 被传输到另一个任务。从任务上下文 1 和任务上下文 2 可以分别读取以及写入“DataA”。此外，从任务上下文 2 和任务上下文 1 可以分别读取和写入以及访问“DataB”。“DataA”and“DataB”被定义为任务上下文 1 和任务上下文 2 的结构实例。

示例中包含功能块 FB_DataSync，以及用于同步这些数据缓冲区的相关扩展 FB_MyDataASync 和 FB_MyDataBSync。这些功能块的实例可以在全局变量列表中进行声明，并在程序序列中使用。此外，只有 1 个任务上下文可以向这些功能块实例写入（方法 Write()），而另一个任务上下文可以读取（方法 Read()）。出于安全考虑，如果忽略这些方法，则会导致失败（返回值为 FALSE）。

正如开头所提到的，实现比其他 PLC 示例更为复杂。在本示例中，数据交换的实际转换是在功能块 FB_DataSync 中进行的，该功能块可以管理对多个缓冲区的访问。如果您想要使用自己的新数据结构来测试功能，则您可以保持 FB_DataSync 不变。与 FB_MyDataASync 一样，定义 1 个从 FB_DataSync 派生的新功能块。此外，您还可以 2 次实例化自己的数据结构，并附加方法 Read()/Write()，这 2 个方法都需要为您的新数据结构分配 1 个引用，并在内部调用基类的方法 ReadData()/WriteData()。

如要转换本示例中的 FB_DataSync，则可以使用 2 个内部缓冲区进行临时数据存储。写入任务上下文会将其数据复制到这些缓冲区中，而读取任务上下文则会从这些缓冲区中复制其数据。函数 TestAndSet() 可确保对 2 个内部缓冲区中的 1 个缓冲区进行正确访问。这意味着不能同时访问相同的数据，但可以在应用程序代码的外部同时写入和读取数据。

下载：https://infosys.beckhoff.com/content/1033/tc3_plc_intro/Resources/7643976587.zip

15 创建可视化

与 TwinCAT 2 相比，可视化客户端只是作为绘图命令的解释器。无论其类型如何（PLC HMI [▶ 551]、PLC HMI Web [▶ 555]），每个客户端都会接收到相同的指令，因此每种类型的客户端所产生的可视化都是一样的。这样做的后果是，务必始终将相关的可视化应用程序下载到 PLC 中。这对小型控制器来说可能是个问题。

在 PLC 项目中，3 种不同的可视化变体是有区别的：

- 集成的可视化 [▶ 550] - 可视化在编程系统上的 TwinCAT 开发环境中运行
- PLC HMI [▶ 551] - 可视化在控制计算机或第 3 台计算机上运行，无需开发环境
- PLC HMI Web [▶ 555] - 可视化在浏览器中运行

尽管有 3 种不同的变体，但只需要进行 1 次工程设计，因此所有可视化的外观都是相同的。只需添加 TargetVisualization [▶ 554] 和 WebVisualization [▶ 556] 对象，即可启用 PLC HMI 和 PLC HMI Web。在“可用性 [▶ 558]”部分可以查看各个 可视化元素 [▶ 369] 的可用性以及不同变体的功能

可视化可以由多个可视化页面组成。在可视化对象 [▶ 367]中可以对其中的每个页面进行管理。例如，在可视化对象的设置中，可以更改大小和预期用途 [▶ 367]（“可视化”、“数字键盘/小键盘”、“对话框”）。在添加第 1 个可视化页面时，会自动添加可视化库 [▶ 366]和可视化管理器 [▶ 354]，该管理器可用于管理 PLC 项目的所有可视化页面的常规设置。

在 可视化编辑器 [▶ 344] 中可以开发可视化页面。该编辑器由 工具箱 [▶ 352] 和 属性窗口 [▶ 353] 补充，前者能够提供可用的可视化元素，后者可以用于配置添加的元素。可视化库可以根据当前选择的配置文件 [▶ 367]提供可视化元素。根据需要可以对可视化元素进行方便地排列和分组 [▶ 345]。

与 TwinCAT 2 相比，所有可视化元素和可视化对象本身都是以 IEC61131-3 功能块的形式实现的。它们可以保存在库中，从而在其他项目中使用。此外，在可视化对象中管理的可视化页面可以在其他可视化页面中实例化，即引用 [▶ 559]。该概念由输入和输出形式的占位符 [▶ 559]进行补充，通过这些占位符可以配置所引用的页面。在接口编辑器中可以对占位符进行编辑。这样可以一次性开发出默认页面或对话框，以便随后以不同的参数设置无限制地重复使用。

通过项目变量可以对可视化元素进行动画处理，在相应的设置中可以直接或通过表达式（即变量与运算符和常量的组合）输入这些变量。例如，这样可以对变量值进行缩放，使其适合在可视化中应用。例如，可以针对绝对移动对矩形类型的元素进行编程。

此外，还可以根据文本列表实现用户管理 [▶ 360]和语言选择 [▶ 563]。ANSI 和 Unicode [▶ 355] 可以作为可视化中的字符编码变体。在可视化中还可以使用任意表达式，甚至是函数调用。

TwinCAT 2 可视化可被导入。

15.1 可视化编辑器

通过可视化编辑器可以创建和配置可视化。双击 PLC 项目中的 可视化对象 [▶ 367]，可打开编辑器。它还有 3 个编辑器：

- 接口编辑器 [▶ 345]，用于定义占位符
- 热键配置编辑器 [▶ 350]，用于在动作和按键或热键之间建立关联
- 元素列表 [▶ 351]，包含在可视化编辑器中打开的可视化页面的所有元素列表

通过菜单项“Visualization”（可视化）或可视化编辑器顶部的箭头可以显示或隐藏这些元素。

通过工具箱 [▶ 352]和属性窗口 [▶ 353]可以在功能上对可视化编辑器进行扩展，工具箱可以将现有的可视化元素 [▶ 369]插入到可视化编辑器中，而在属性窗口中则可以对当前在可视化编辑器中突出显示的元素属性进行显示和编辑。通过菜单项“View”（视图）可以打开它们。

选择可视化元素

在可视化编辑器中选中的 可视化元素 [▶ 369] 在其框内会显示黑色小方块，点击鼠标可以选择并移动它们。这样可以修改元素的位置和大小。

如要选择 1 个元素，可将光标置于该元素上。当光标符号变成手形时， 按鼠标左键。此时可进行多项选择。为此，可按住 Shift 键并选择各个元素，或者在按住鼠标左键的同时在元素周围画 1 个矩形。您也可以在元素列表 [▶ 351]中选择元素。然后，您还可以选择 1 组中的单个子元素。

键盘操作

如果已经选中 1 个元素，则可以通过制表键将选择移动到添加序列中的下一个元素，或者通过 Shift + Tab 键将选择移至上一个元素。

使用空格键可以打开当前所选元素中的文本输入字段。输入字符串并按 Enter 键保存。

在 Properties（属性）窗口中会立即显示所选元素的属性，并可在此进行编辑。如果已经选中多个元素，则对属性的更改会应用到所有元素。

位置、大小、对齐方式、顺序

在插入可视化元素 [▶ 369]之后，在编辑器窗口中可以直接更改以下属性；请注意，在 属性窗口 [▶ 353]中也可以对大小和位置进行编辑。

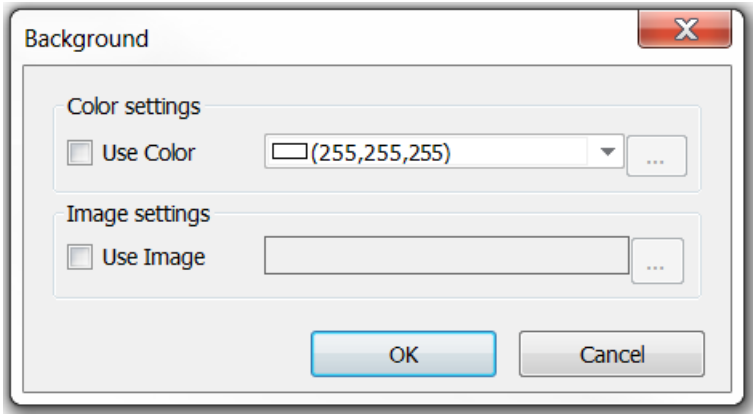
位置	通过鼠标点击选中元素，然后在不松开鼠按钮的情况下直接将其拖到所需的位置，或者再次点击已经选中的元素，然后按住鼠标按钮或通过箭头键来移动它。另一个选项是在元素属性 [▶ 353]中编辑位置值。
大小	选择元素，然后点击其边框中的 1 个小方块。光标图标（交叉双箭头或者垂直或水平双箭头）表示可以从哪个方向修改方块的大小。
对齐方式	如要对齐 2 个或多个元素，可首先选择您想要对齐的元素。然后，通过菜单项“Visualization”（可视化）或者在可视化编辑器中点击右键的方式，打开子菜单“Alignment”（对齐方式），以更改对齐方式。
顺序	如要将 1 个或多个可视化元素置于可视化的背景、前景或两者之间，可首先选择您想要放置的元素。然后，通过菜单项“Visualization”（可视化）或者在可视化编辑器中点击右键的方式，打开子菜单“Order”（顺序），以更改顺序。

可视化元素的分组

如要将多个可视化元素合并为 1 组，可首先选择您想要分组的元素。然后，通过菜单项“Visualization”（可视化）或者在可视化编辑器中点击右键的方式，选择“Group”（组），对元素进行分组。在形成组之后，仅可通过元素列表 [▶ 351]来选择该组的子元素。

背景图像

可视化页面提供了 1 个指定背景颜色和/或背景图像的选项。在可视化编辑器中点击右键，可打开上下文菜单，在该菜单中可以选择条目“Background”（背景）。该条目可打开以下对话框：



背景颜色和背景图像都有复选框。如要选择 1 个图像作为背景图像，首先必须将其添加到 PLC 项目的图像池中 [▶ 134]。

15.1.1 接口编辑器

接口编辑器可用于定义可视化中的接口。这些接口用于在另一个可视化页面的边框中引用。该编辑器是可视化编辑器的一部分，在该编辑器的上部显示为 1 个单独的选项卡，紧邻热键配置 [▶ 350]和元素列表 [▶ 351]的编辑器。

接口声明

由于可视化可被视为 1 个功能块，因此接口编辑器会在可视化编辑器的上部显示为 1 个普通的声明编辑器。在此处可以看到输入变量（例如，边框参数）。

参数声明

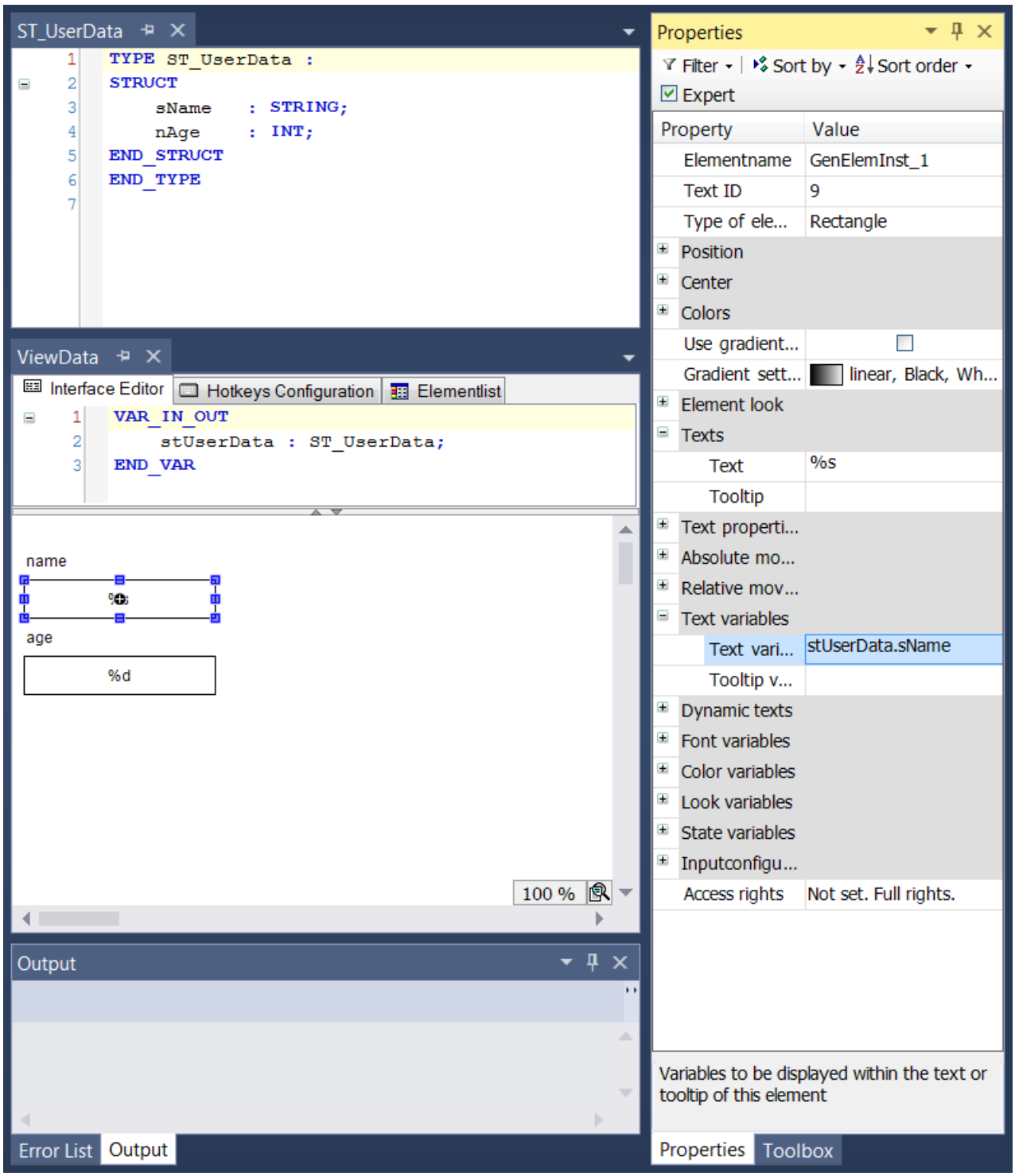
- VAR_INPUT
- 带有“parameterstringof”属性的 VAR_INPUT：带有该属性的变量的值是变量的名称，在边框元素中调用时，该名称被设置为待定义的输入变量。

```
VAR_INPUT
{attribute 'parameterstringof' := 'bInput'}
sVarName : STRING;
bInput : BOOL;
END_VAR
```

- VAR_IN_OUT：如果传输数据结构，则必须将其定义为 VAR_IN_OUT。在不使用编译指示的情况下，可在打开可视化时复制参数值。条目会被写入复制的数据结构中。在关闭可视化时，值会被写回参数中。
- 带有 VAR_IN_OUT_AS_POINTER 属性的 VAR_IN_OUT：对象可以作为对可视化的引用进行传输。当需要将信息传输到可视化而不实际复制信息时，或者在打开对话框的情况下从外部更新信息时，这种方法可能非常有用。只有被设置为对话框并作为对话框使用的可视化才可以使用该属性。

示例

ViewData 可视化使用 ST_UserData 类型的接口 stUserData。ViewData 可视化的可视化元素 GenElemInst_1 是基于用户的，可显示 stUserData.sName 的变量值，就像在 GenElemInst_1 的视图属性中进行编程一样。



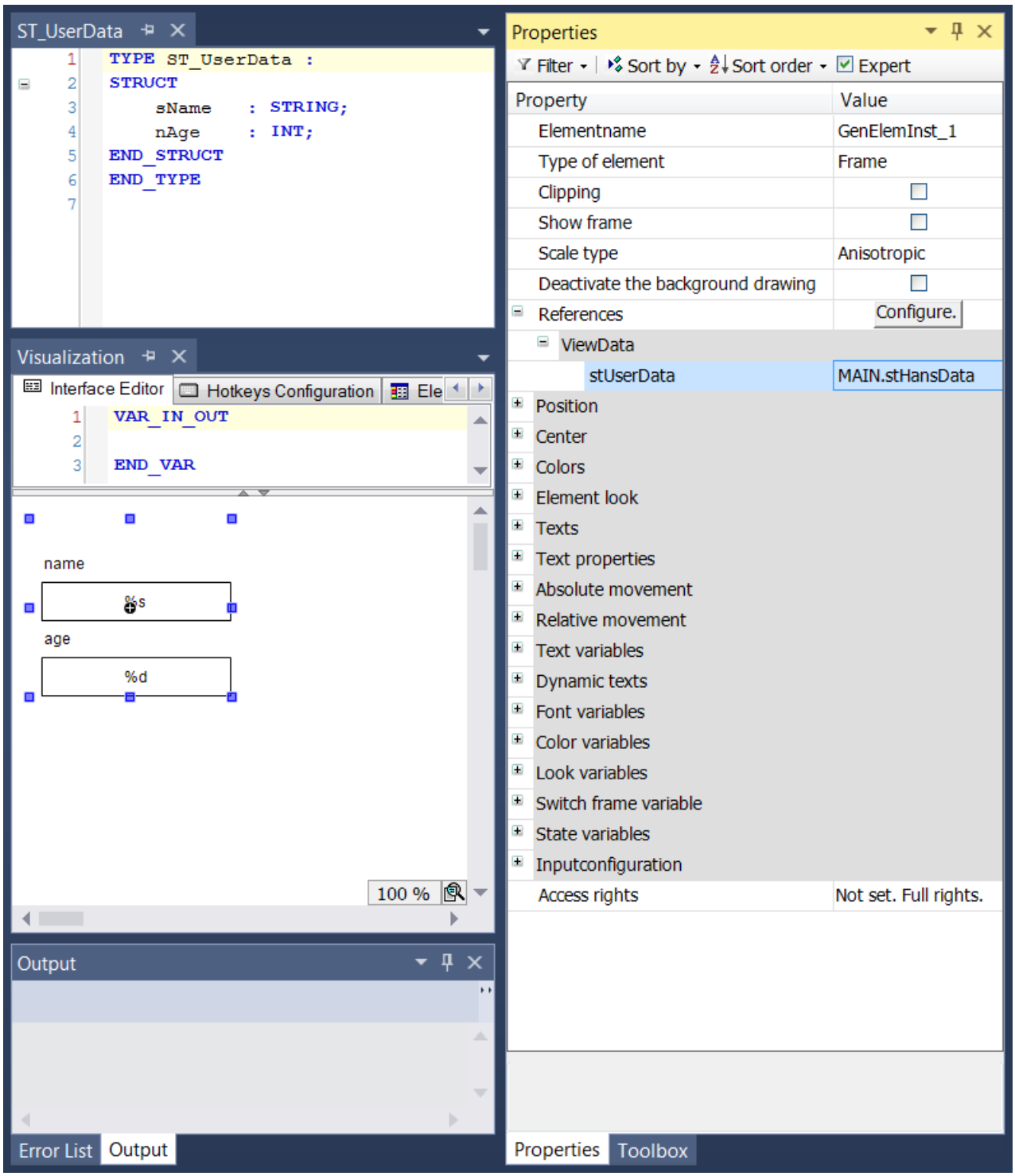
边框编程

如要对边框元素进行编程，可按照以下步骤操作：

1. 在您的主可视化中，添加 1 个边框元素。
2. 将引用设置为具有接口的可视化，该接口可以存储在库中。
3. 为每个边框参数分配变量。

示例

边框可视化元素会引用 ViewData 可视化。在 Properties（属性）视图中会列出赋值与所有边框参数。点击进入此类变量的值域，可分配 1 个相同类型的变量。在此处也可以分配 MAIN.stHansData。



更新边框参数

如果对添加到边框中的可视化接口进行修改，则在保存或编译项目时或者在打开受影响的对象时，在示例中显示的对话框就会自动打开。然后，您可以在此对话框中调整边框参数。如果对源于库的可视化接口进行修改，则还须更新您的项目中的参数。

参数	此列以树形视图显示修改后的参数。首先列出可视化，然后列出相应的边框以及对修改后的可视化的引用。在其下列出当前有效的参数，然后是先前的参数。
类型	此列显示参数类型。
值	值字段包含分配给参数的变量。请注意字段的颜色： • 米色：从先前的配置中自动获取该参数。 • 灰色：尚未给该新参数赋值。 • 绿色：已给该新参数赋值。 • 白色：此处没有需要配置的内容。

编辑值

如要编辑值，可通过点击鼠标来选择值字段。点击进入字段或按空格键，可打开行编辑器。或者，在您选中字段后，直接开始键入即可。如要分配不同的变量，可键入变量名或使用输入向导选择变量。

您可以将现有值条目复制到另一个字段。选择要复制的条目，使用“Copy”（复制）并选择您想要插入该条目的字段，然后使用“Paste”（粘贴）。

选择“OK”（确定），确认对话框中的设置。对话框会关闭，在 Properties（属性）视图下的 References（引用）类别中以及相应的边框元素中会显示新的参数赋值。

在可视化中使用实例名称

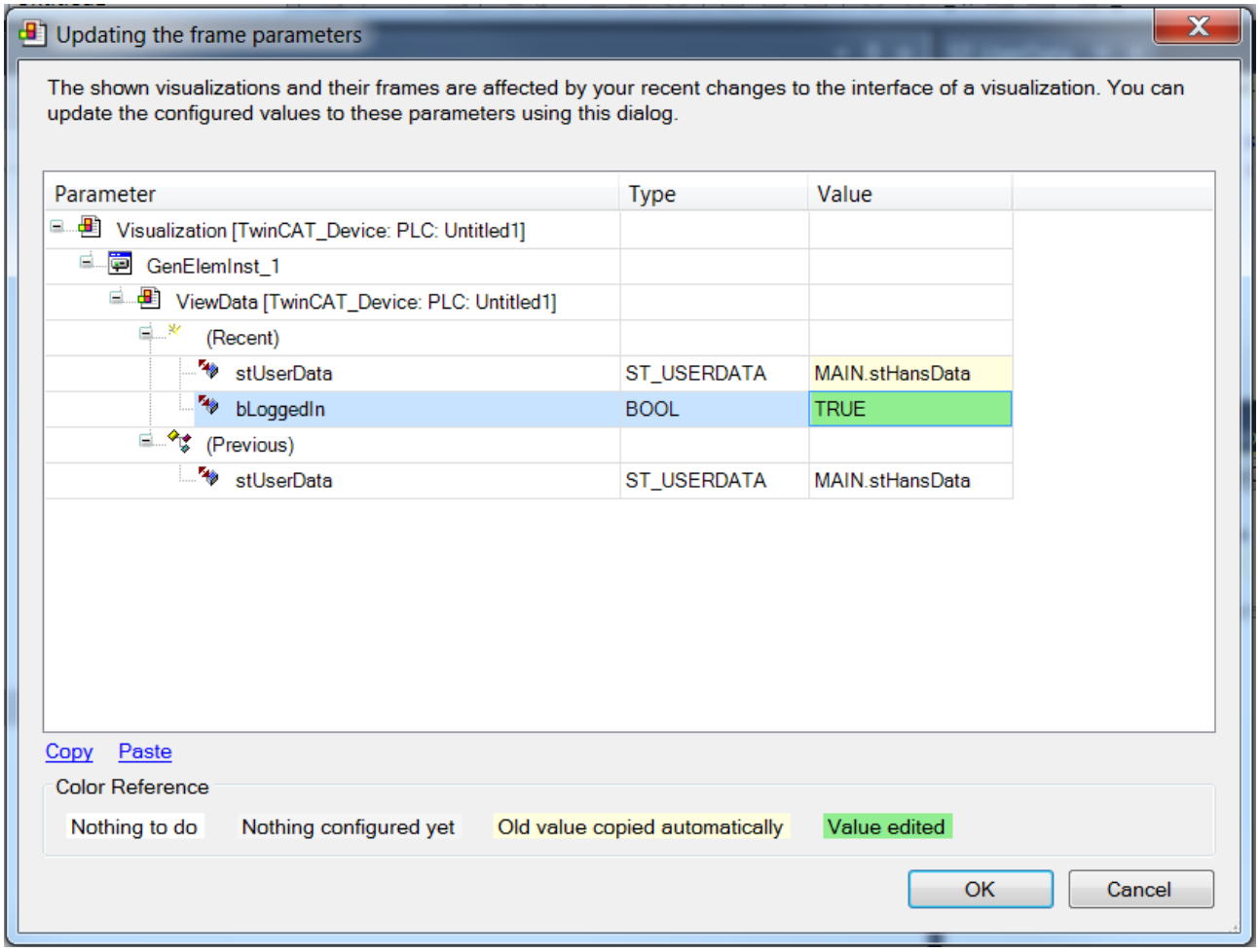
如要在引用的可视化功能块中使用已传输变量的实例名称，可使用提供相应字符串的编译指示属性“parameterstringof [▶ 758]”。

示例

使用变量 bLoggedIn 已扩展 ViewData 可视化接口：

```
VAR_IN_OUT
stUserData : ST_UserData;
bLoggedIn : BOOL;
END_VAR
```

在编译或保存项目时，会出现以下用于配置新参数的对话框：



它包含当前参数和先前参数之间的比较。bLoggedIn 被赋予新值 TRUE。在选择 OK（确定）关闭对话框之后，也可以在边框元素的属性中复制新参数。

编译指示

在接口编辑器中可以使用以下 2 个参数。

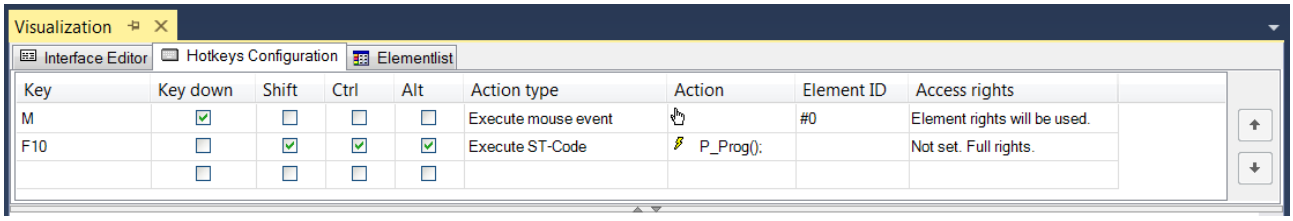
- 属性“VAR_IN_OUT_AS_POINTER”
- 属性“parameterstringof”

15.1.2 热键配置

除了默认热键 [▶ 567]之外，在可视化编辑器的该部分中还可以定义特殊热键，用于在在线模式下运行可视化页面。也就是说，可以将 1 个动作分配给特定的按键或热键。按键配置仅适用于当前的可视化页面。在 默认热键 [▶ 358] 中应该对可用于 PLC 项目的所有可视化页面的按键配置进行定义。

在这种情况下，请注意可视化元素的“Hoykey”（热键）属性，在输入配置 [▶ 400]中可以设置该属性。在热键配置编辑器中也可以对这种元素特定的按键配置进行管理。在 2 个地方可以对它进行编辑，而且，在 2 个编辑器中可以应用更新。

在可视化编辑器的上部，热键配置编辑器可显示为 1 个单独的选项卡，紧邻接口编辑器 [▶ 345]和元素列表 [▶ 351]。

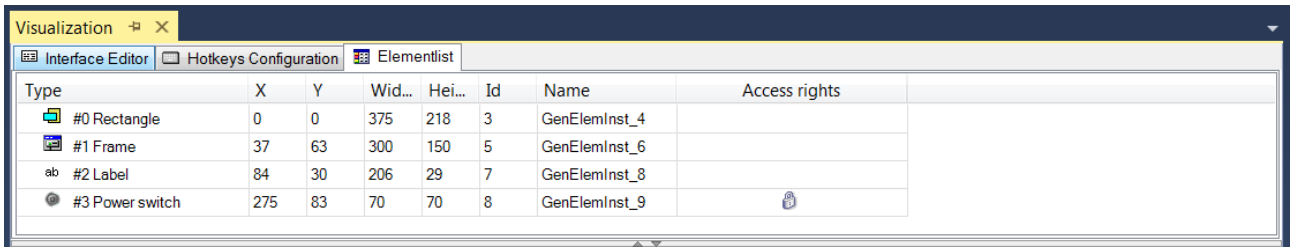


在表格编辑器的每一行中，都可以将 1 个按键或热键与 1 个动作建立关联。请参见以下列：

按键	按键的名称。您可以手动输入按键名称，也可以通过选择列表进行输入，双击单元格，即可打开选择列表。该列表包含在设备描述中定义的所有按键。
按下按键	如果启用该选项，则会在按下按键时执行动作。否则，会在松开按键时执行动作。如果要在按下按键（KeyDown）和松开按键（KeyUp）时执行动作，则表格必须包含 2 个相应的按键定义。
Shift、Ctrl、Alt	如果启用该选项，则必须同时按下 Shift 或 Ctrl 或 Alt 键才能执行动作。
动作类型	通过选择列表可以定义动作类型，双击单元格，即可打开选择列表。动作类型与可视化元素的输入配置 [▶ 371]中可用的类型相对应。
动作	对要执行的动作进行更精确地配置。这取决于动作类型，并与在可视化元素的输入配置 [▶ 371]中可以使用的鼠标动作相对应。
元素 ID	可视化元素的 ID，通过“热键 [▶ 400]”属性可以将按键分配给该元素。当前可视化中的唯一 ID。 如果 1 个按键与多个动作建立关联，则会按照在配置表中列出的顺序（从上到下）执行这些动作。通过选择按键定义（点击进入表格行）并使用表格右侧的箭头键上下移动可以更改此配置。 请注意处理按键动作的以下调用顺序： 1. 应用程序的事件处理程序（如果启用）（可选），例如，用于检测事件和条目 2. 可视化管理器中的热键配置 [▶ 358]，适用于应用程序中的所有可视化 3. 在线模式下的标准键盘操作 [▶ 567]的定义 4. 在此处介绍的各个可视化的特殊热键配置，即在边框中引用可视化之前查看主可视化。
访问权限	该设置可用于定义允许使用相关热键的用户组的选择。点击打开用于选择现有组的对话框。此时有以下 2 种状态消息： • 未设置。全部权限。 如果热键可用于所有组，则会设置该默认消息。 • 已设置权限：有限的权限。 如果至少有 1 个组的热键被阻塞，则会设置该消息。 • 已使用元素权限 如果通过“热键 [▶ 400]”属性将热键分配给可视化元素，则会自动应用相应的元素权限。

15.1.3 元素列表编辑器

可视化编辑器的该部分显示了当前在编辑器中打开的可视化页面的所有元素列表。



这些元素根据其在可视化 Z 轴上的位置从上到下列出，即第 1 行指的是背景中最远的元素。此处可以显示以下值，但不能进行编辑：

类型	在工具箱 [▶ 352]中使用的元素类型和图标，以及最初根据添加元素的顺序而产生的元素编号。
X，Y	元素左上角的位置（0，0 = 可视化区域的左上角）
宽度，高度	元素尺寸
ID	内部分配的元素 ID
名称	元素名称，在元素属性 [▶ 353]中进行定义
访问权限	如果某个元素的行为被限制在一些用户组内，则会使用挂锁符号来表示这种情况。

通过选择相应的表格行可以选择元素，即选择过程始终与可视化编辑器的主窗口同步。但请注意，仅可在元素列表中选择组的子元素。

仅可在可视化编辑器 [▶ 345]中更改所选元素在 Z 轴（即背景/前景轴）上的位置。

在元素列表中也可以使用在元素列表中撤销编辑操作（[Ctrl] + [z]）或删除元素（Del）的命令。

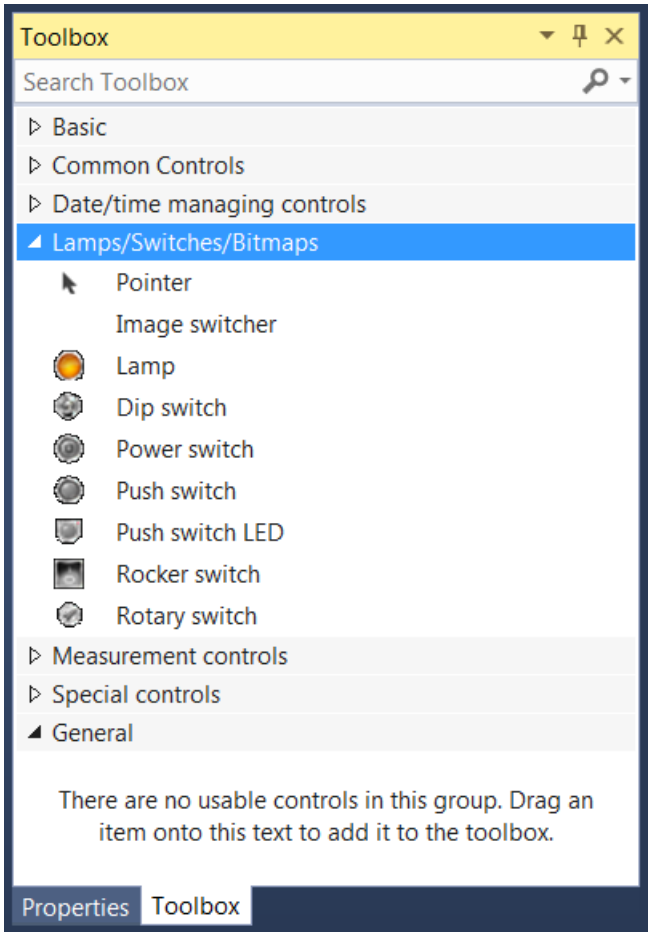
15.1.4 工具箱

工具箱会提供所有可视化元素，在可视化编辑器 [▶ 344]中可以添加这些元素。通过库 [▶ 366]可以在项目中使用这些元素。工具箱窗口中的选择取决于当前激活的可视化配置文件 [▶ 367]。

如果工具箱窗口尚未显示，则可在“View”（视图）菜单中通过“Toolbox”（工具箱）将其打开。工具箱包含以下类别：

- 通用控制 [▶ 385]
- 基础 [▶ 433]
- 日期/时间管理控制
- 灯/开关/图像 [▶ 479]
- 测量设备 [▶ 488]
- 特殊控制 [▶ 528]

在这些类别下，列出了相应的可视化元素 [▶ 369]及其符号和名称。

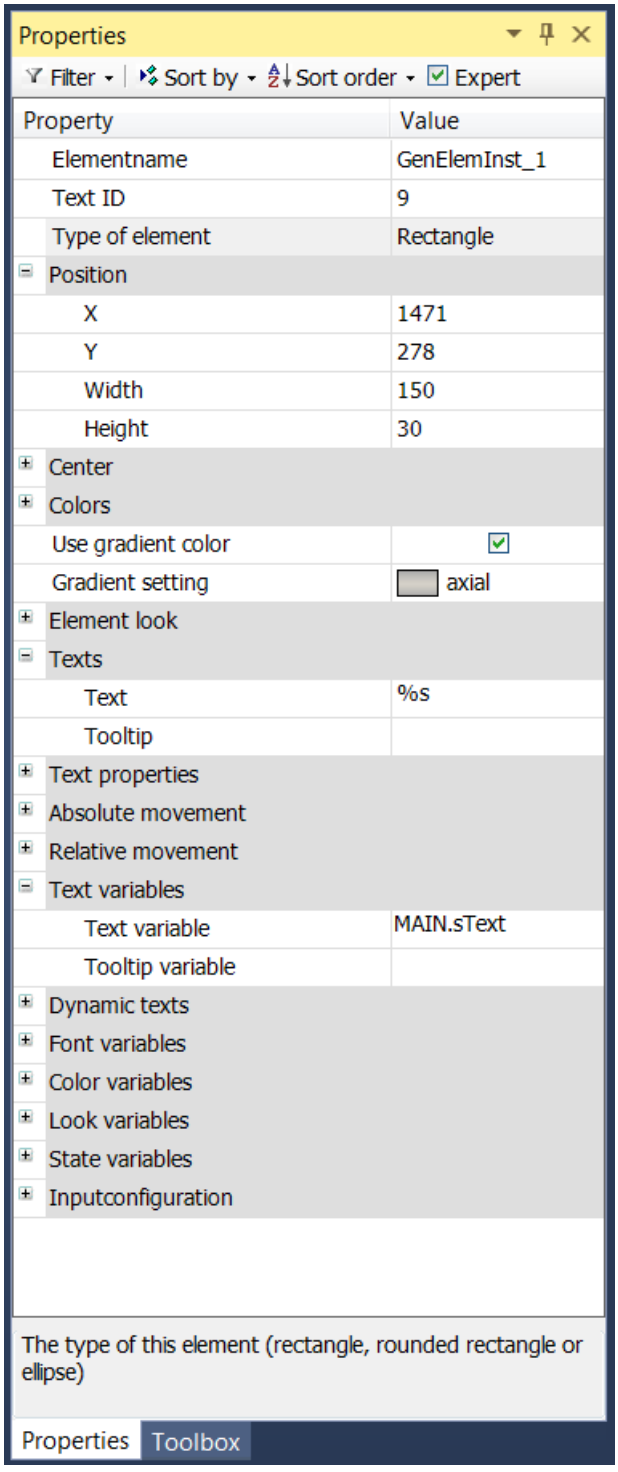


通过拖放的方式可以在可视化编辑器中插入可视化元素。在拖动元素时，光标符号上的加号表示可以将该元素拖放到编辑器窗口中。松开鼠标按钮后，元素就会出现在编辑器中。

也可以在“Visualization”（可视化）菜单的“Add visualization element”（添加可视化元素）下也已添加元素。此处提供的元素范围与工具箱中的相同。如有必要，根据需要通过自定义对话框（命令类别“Visualization commands”（可视化命令））可以对可视化菜单进行自定义。

15.1.5 属性窗口

当前在可视化编辑器 [▶ 344]中打开的可视化页面中的任何可视化元素 [▶ 369]均可供选择 [▶ 344]。然后，在编辑器中可以直接修改其位置、大小、配置和对齐方式 [▶ 345]（鼠标动作或可视化命令）。在属性窗口中可以配置更多参数。



属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，

- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用 ... 按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

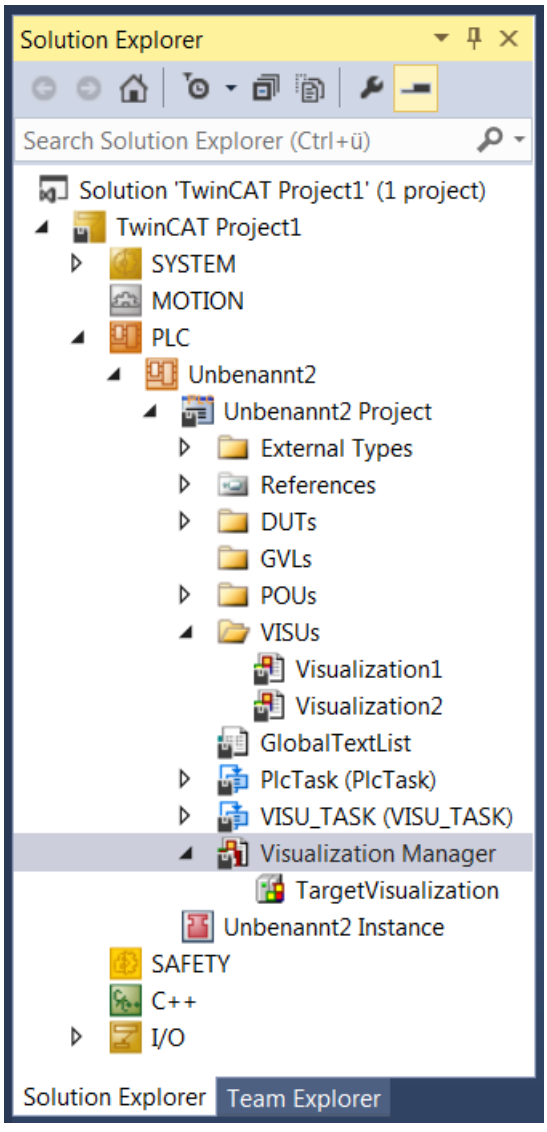
借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

15.2 可视化管理器

可视化管理器对 PLC 项目的所有可视化对象 [▶ 367]的常规设置进行管理。在 PLC 项目中创建第 1 个可视化对象之后，就会自动添加 可视化管理器对象。有关 PLC 项目中的可视化概述，请参见“创建可视化 [▶ 344]”部分。

如果设备支持该功能，则可在管理器下添加 PLC HMI [▶ 551] 客户端和/或 PLC HMI Web [▶ 555] 客户端的客户端对象。它们会管理各自客户端类型的特殊设置。如果在可视化管理器下没有添加客户端对象，则会自动使用集成的可视化 [▶ 550]。

在下图所示的示例中，可视化管理器对可视化对象“Visualization1”和“Visualization2”负责。此外，还可以添加 TargetVisualization 对象，从而启用 PLC HMI。

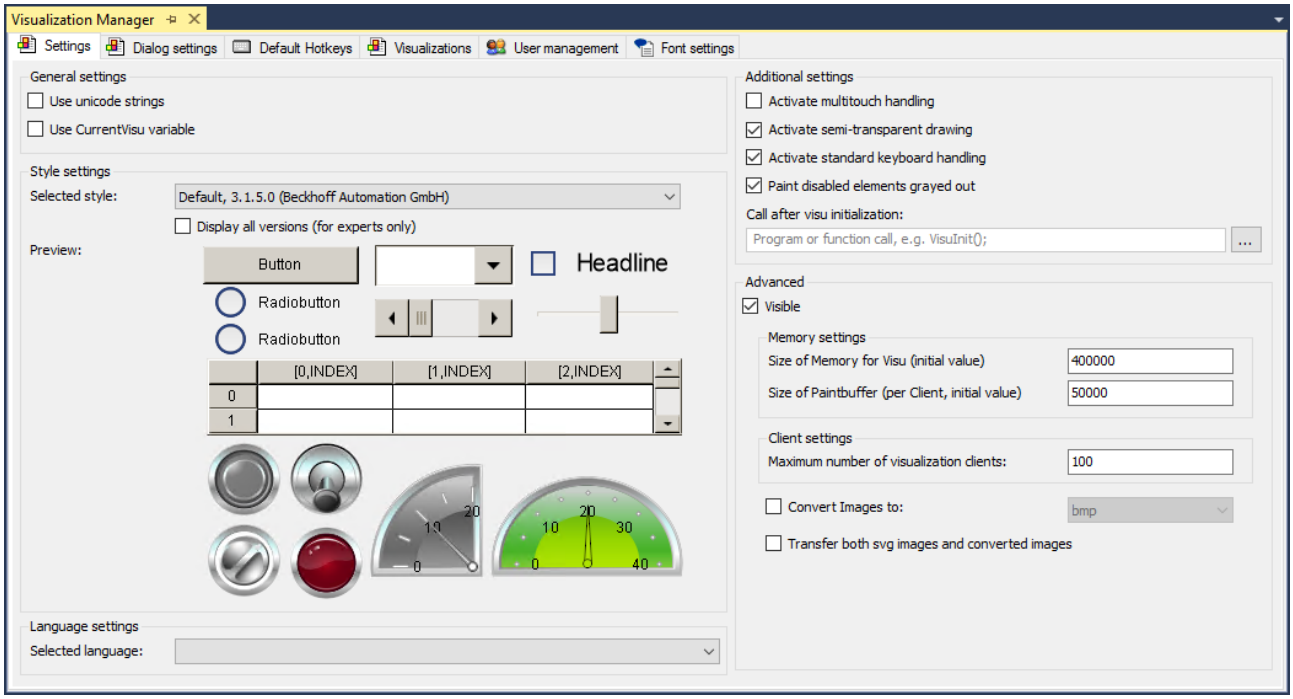


如要打开可视化管理器编辑器，可双击条目。编辑器会打开 1 个窗口，其中有以下选项卡：

- 设置 [▶ 355]
- 对话框设置 [▶ 357]
- 默认热键 [▶ 358]
- 可视化 [▶ 359]
- 用户管理 [▶ 360]
- 字体 [▶ 365]

15.2.1 设置

该选项卡包含适用于 PLC 项目的所有可视化的设置。



常规设置

使用 Unicode 字符串	如果启用该选项，则在可视化中使用的所有字符串 [▶ 561]都将以 Unicode 格式进行处理。为此，必须在PLC 项目设置 [▶ 367]中的 Compile > Settings > Compiler defines（编译 > 设置 > 编译器定义）下额外输入“VISU_USEWSTRING”。
使用 CurrentVisu 变量	PLC 项目知道并使用 STRING 类型的全局变量 VisuElems.CurrentVisu。它包含当前在运行时激活的可视化的名称。 在读取模式下，可以访问该变量，以获取当前激活的可视化的名称；在写入模式下，可以访问该变量，以更改可视化。在所有显示设备上进行并行切换，以便在所有连接的客户端上都会显示相同的可视化页面。 要求：应用程序包含可调用其他可视化的可视化。 示例： • 分配变量：VisuElems.CurrentVisu := sVisuName; • 分配文本：VisuElems.CurrentVisu := ‘Visualization1 ‘;

样式设置

选定样式	每种可视化都会以这种风格呈现元素。
显示所有版本（仅限专家）	如果启用此设置，则在系统上安装的所有样式版本都可以通过选择菜单进行选择。
预览	在预览中显示的元素代表所选样式。

语言设置

选定语言	在启动可视化时，将使用所选语言。

在可视化开始时的默认语言只能与 PLC HMI [▶ 551] 和/或 PLC HMI Web [▶ 555] 一起设置。通过集成的可视化 [▶ 550]，在启动时会自动使用文本版“标准”。在集成的可视化中，还可以通过可视化按钮在运行时进行语言更改 [▶ 563]。

另请参见：

- 文本和语言 [▶ 561]

对于 build 4024.0 及以上版本，在单独的 对话框设置 [▶ 357] 选项卡上将会列出用户管理对话框的设置，并在那里进行相应的记录。

附加设置

激活多点触控处理	在运行时，可视化系统希望用户通过手势和触控事件进行输入。受影响的元素： • 具有输入配置的元素 • 边框类型的元素 • 选项卡控制类型的元素
激活半透明绘图	可视化系统会以半透明的颜色绘制元素。 在定义颜色时，您可以额外指定透明度的分度值。在 Transparency（透明度）属性中可以定义透明度。 预设值：已激活。 要求：您创建 1 个新的可视化，显示变体可以半透明方式绘制。
激活默认键盘操作	• Tab • Shift + Tab • 输入 • 向上箭头 • 向下箭头 • 右箭头 • 左箭头
将停用的元素绘制成灰色	可视化系统会将停用的元素绘制成灰色。

扩展设置

可见	内存设置、文件传输模式和客户端设置均可见。（标准应用程序不需要）

如果使用集成的可视化系统，则不可用的设置将显示为灰色或者根本不会显示。

内存设置

Visu 的内存大小	可视化在运行时分配的内存大小（以字节为单位）。 预设值：400000
Paintbuffer 的大小（每个客户端）	可视化为每个显示选项分配且可用于绘图动作的内存大小（以字节为单位）。 预设值：50000

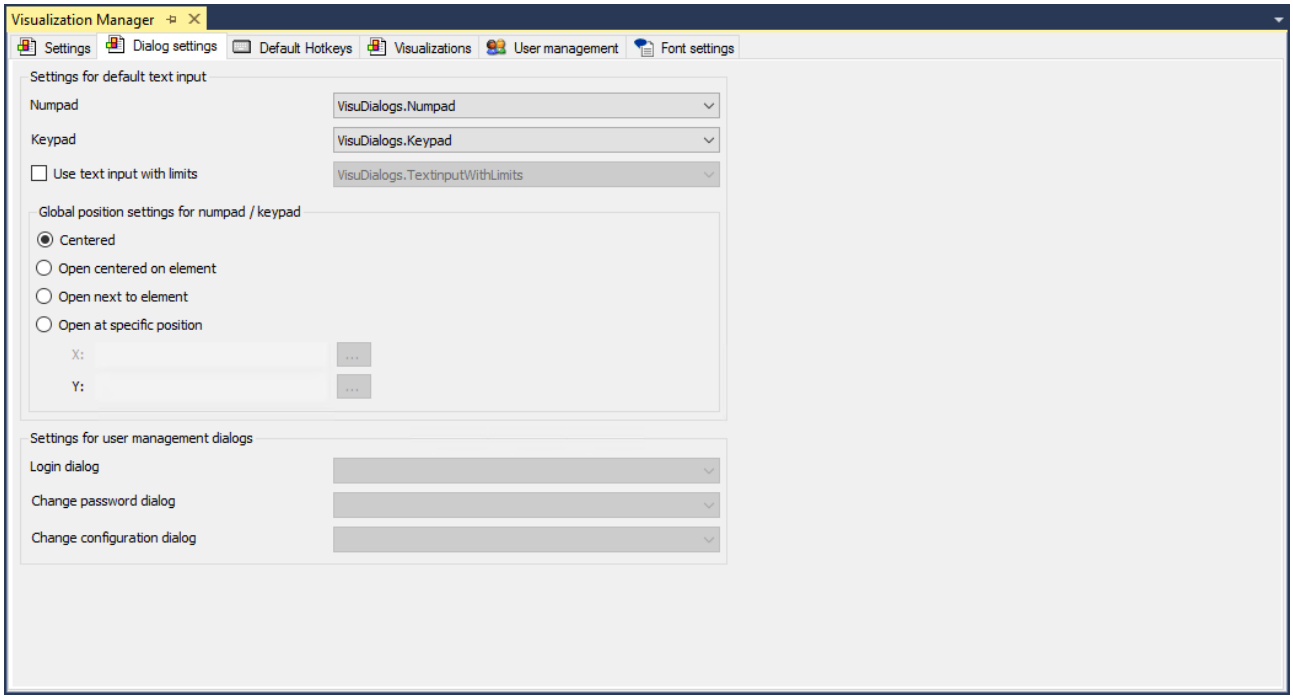
客户端设置

可视化客户端的最大数量	限制可同时运行的显示变体的数量。 如果您配置的元素会随显示变体的不同而变化，则您需要限制显示变体的数量。在运行时会为可视化分配 1 个 ID，用于识别显示变体。然后，对数据进行相应地处理。通过系统变量 CURRENTCLIENTID，TwinCAT 可以查询 ID，以获得有关受影响的当前变体的信息。 示例：arr[CURRENTCLIENTID].dwColor 要求：库 VisuGlobalClientManager 已集成在项目中。

将图像转换为	如果启用此设置，则您可以从下拉菜单中选择是以 bmp 格式还是 png 格式编译标准元素 [▶ 369]的图像。如果在目标系统上安装 Windows Embedded Compact 操作系统，则有必要执行此操作，因为在 PLC HMI [▶ 551] 客户端中只能显示 bmp、png 和 jpg 图像。
传输 svg 图像和转换图像	如果启用此设置，则先前已转换为 bmp 或 png 的图像文件也能够以原始的 svg 格式加载到目标系统上。PLC HMI [▶ 551] 客户端在倍福 CE 设备本地使用 bmp 或 png 格式的图像文件，PLC HMI Web [▶ 555] 客户端使用 svg 格式的图像文件。 如果同时激活 PLC HMI [▶ 551] 和 PLC HMI Web [▶ 555]，则可以使用该设置。

15.2.2 对话框设置

该选项卡包含在运行时可视化中用于文本输入和用户管理的对话框的默认设置。在相关可视化元素的输入配置中可以定义要使用的对话框。



可视化管理器中的全局设置仅对在 PLC HMI [▶ 551] 或 PLC HMI Web [▶ 555] 中使用的情况有效。

标准文本输入的设置

对于具有标准文本输入的元素，在运行时会出现 1 个支持输入的对话框。您可以决定让哪个对话框出现。

数字键盘	数字键盘形式的对话框，如果用户激活数字的输入字段，则可视化在运行时会打开该对话框。 预设值：VisuDialogs.Numpad
小键盘	键盘形式的对话框，如果用户激活文本的输入字段，则可视化在运行时会打开该对话框。 预设值：VisuDialogs.Keypad
使用有限制的文本输入	要求：将 PLC HMI [▶ 551] 或 PLC HMI Web [▶ 555] 配置为显示选项。默认文本输入模式为小键盘。在这种情况下，可视化支持在运行时通过键盘输入文本。 对于值的范围有限的输入，可以调用 1 个对话框（而非输入字段）来显示值的范围。 预设值：VisuDialogs.TextinputWithLimits 此对话框显示了值的范围，不接受超出这些限值的值。
数字键盘/小键盘的全局位置设置	• 居中：在屏幕的中央打开对话框。 • 在元素的中央打开：在元素上打开对话框并覆盖它。 • 在元素的旁边打开：将经过动态优化的对话框放在元素的旁边。 • 在定义的位置打开：在可视化窗口中，在此处定义的位置打开对话框。X，Y：变量或直接数字（像素），用于在可视化窗口的坐标系中定义对话框的左上角。 注意：可视化坐标系的原点在左上角。正向水平 X 轴朝右。正向垂直 Y 轴朝下。

用户管理对话框的设置

您可以通过用户管理来配置您的可视化。为此，您可以在元素上配置 1 个输入，使用户管理对话框出现。如要使用其他可视化作为用户管理对话框，您必须在此处更改预设值。

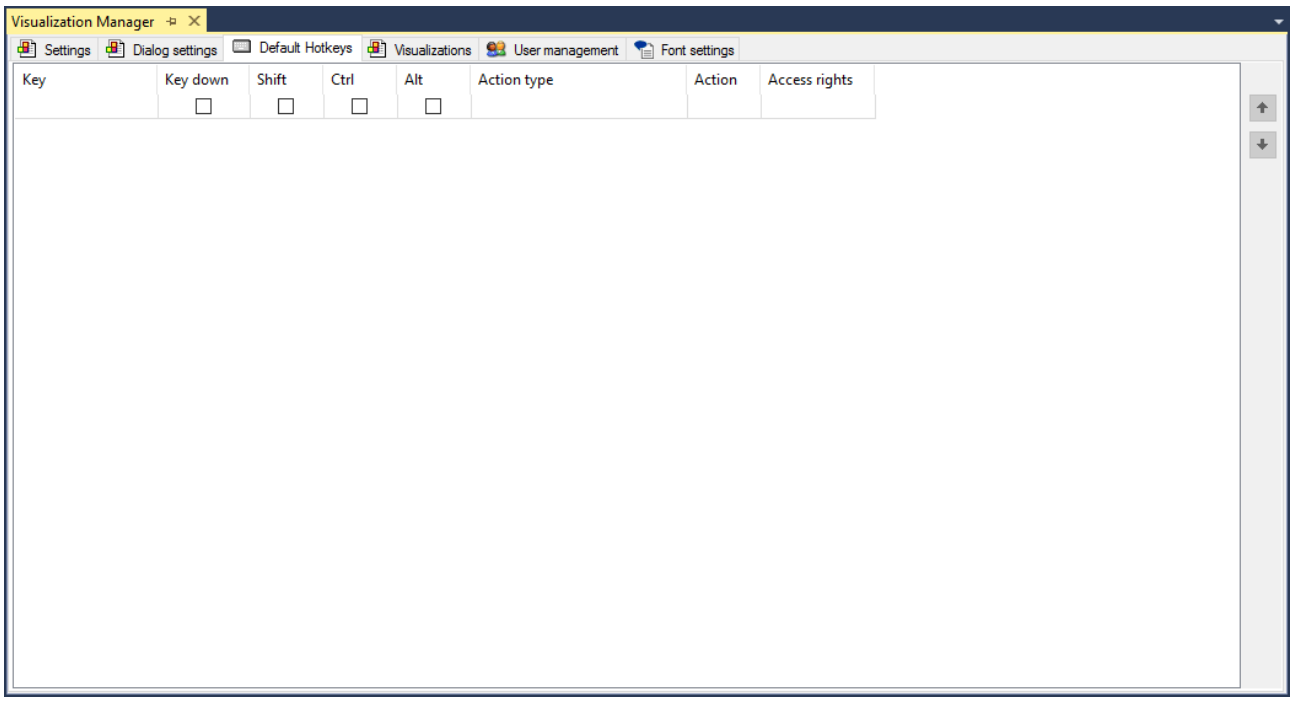
登录对话框	可实现登录的用户管理对话框，通常需要输入用户名和密码。该对话框会在 1 个元素上的输入事件中出现，作为后续动作，该元素可以执行用户管理的登录动作。 预设值：VisuUserManagement.VUM_Login
更改密码对话框	可实现密码更改的用户管理对话框，通常需要输入当前密码和新密码。该对话框会在 1 个元素上的输入事件中出现，作为后续动作，该元素可以执行用户管理的更改用户密码动作。 预设值：VisuUserManagement.ChangePassword
更改配置对话框	可实现用户管理的配置更改的用户管理对话框，即通常会显示当前用户配置以及更改配置的选项。该对话框会在输入事件中出现，作为后续动作，该对话框可以执行用户管理的打开用户管理动作。 预设值：VisuUserManagement.VUM_UserManagement

另请参见：

- 用户管理 [▶ 360]

15.2.3 默认热键

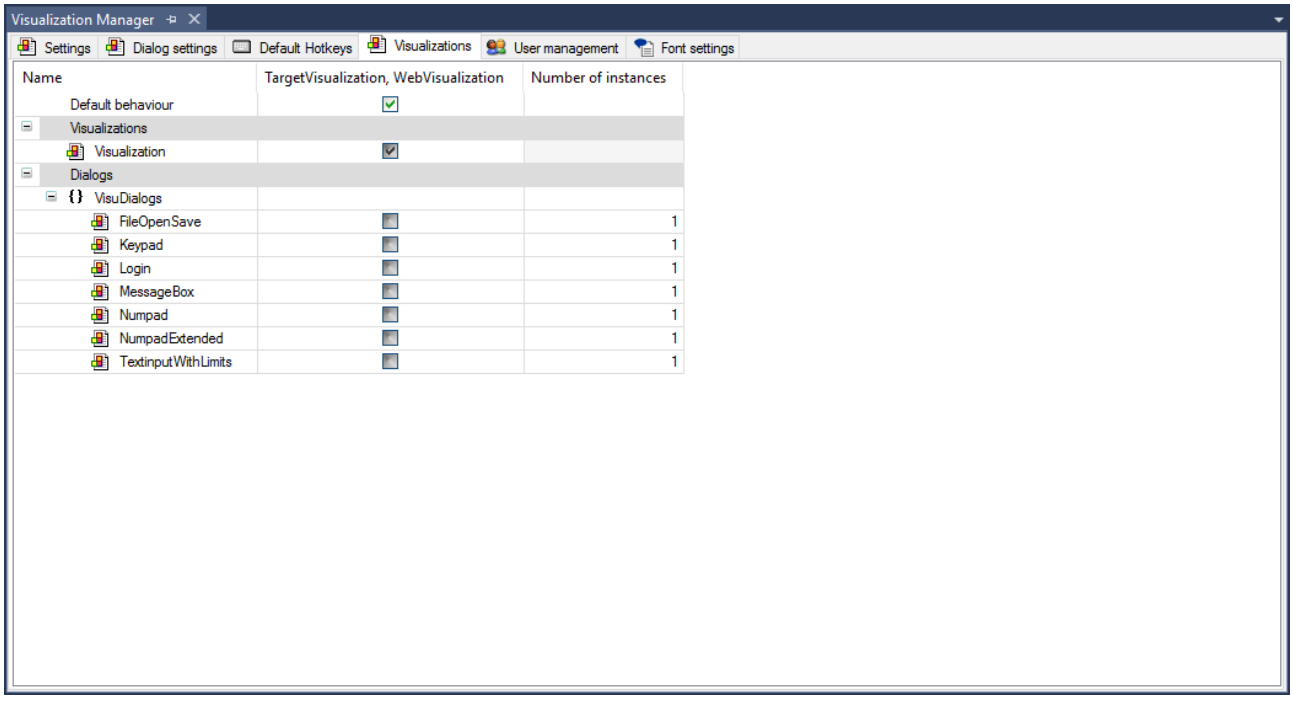
该选项卡包含适用于相关 PLC 项目中所有可视化页面的键盘配置。换句话说，如果相关设备支持的话，在此处定义的按键/按键组合可用于在线模式下可视化中的用户输入。



配置器的操作方法与可视化编辑器中用于特殊可视化的配置器相同。请参见热键配置编辑器 [▶ 350]的说明。此外，某些与设备无关的默认热键 [▶ 567]始终可用于可视化中的导航。

15.2.4 可视化

该选项卡列出了所有可用的可视化，可根据显示变体为加载行为分配可视化。



默认行为	如果激活该选项，则 PLC 项目中的可视化对象将自动加载到相关的目标系统中。已激活的复选框会指明这些对象。 如果未激活该选项，则您可以明确定义每个可视化的加载行为。
可视化	这里列出了在项目中创建的所有可视化。
对话框	这里列出了项目中以及库中所有对话框类型的可视化。 注意：实例数列表示相关对话框的实例化频率。

该功能取代了先前可能使用的“可视化引用”对象。虽然已添加的对象仍将继续运行，但此类对象已无法再添加。

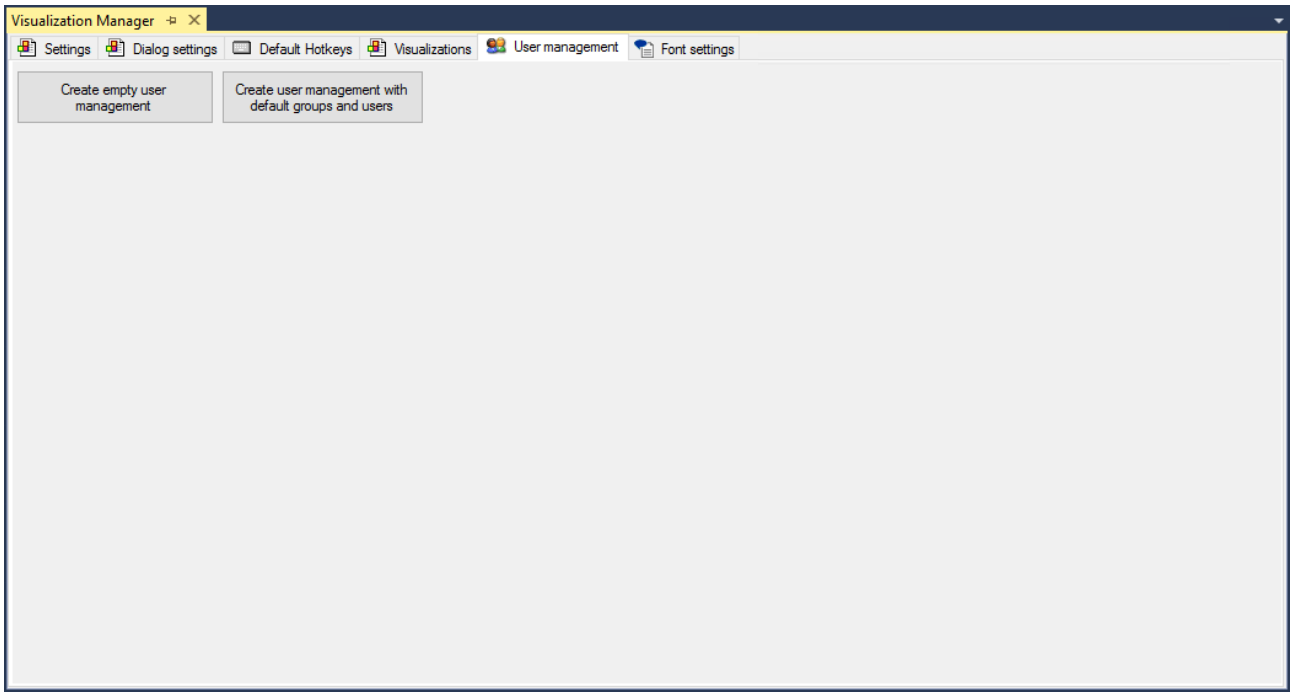
15.2.5 用户管理

用户管理可用于根据用户的具体情况来管理可视化元素的可见性和可操作性。可视化的切换也可以根据用户的具体情况进行配置。用户以组为单位进行组织。

用户管理只能与 PLC HMI [▶ 551] 或 PLC HMI Web [▶ 555] 组合使用。

第 1 步

首先，您需要创建唯一的组和用户。在可视化管理器编辑器中，打开“User management”（用户管理）选项卡。如果尚未配置用户管理，则会出现以下对话框：

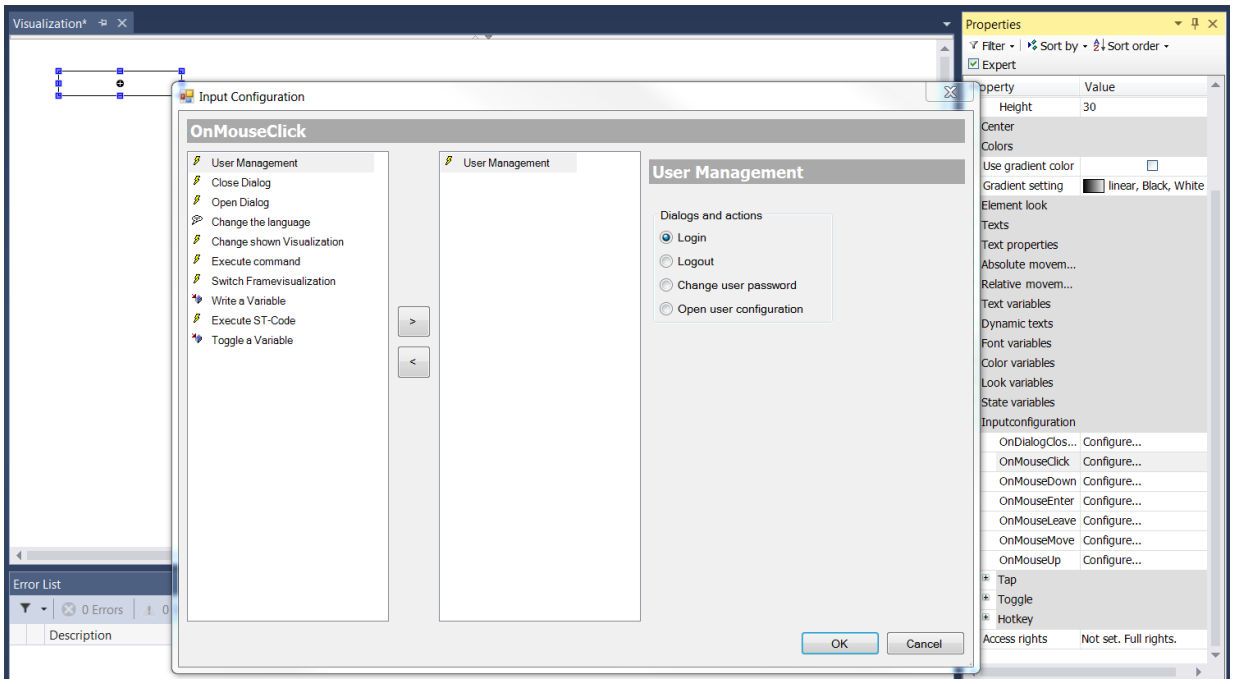


创建空用户管理	用户管理会打开。组 None 已创建。
创建带有默认组和用户的用户管理	用户管理会打开。以下组和用户已创建： • 组 Admin 和用户 Admin • 组 Service 和用户 Service • 组 Operator 和用户 Operator • 组 None

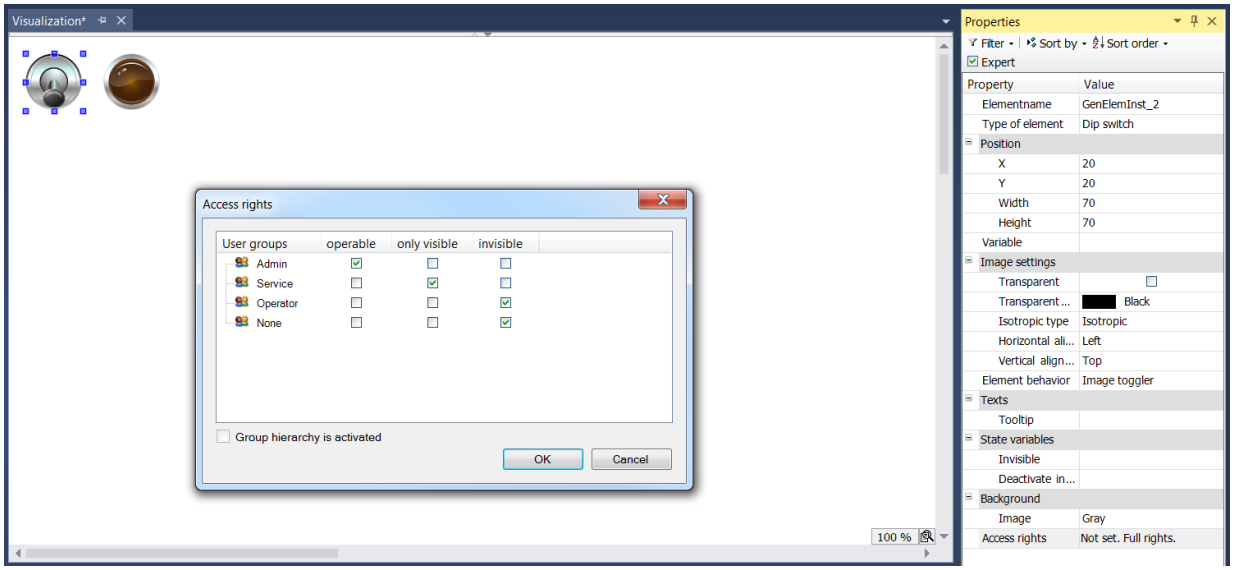
对可视化进行编程

1. 通过定义组 [▶ 362]和用户 [▶ 363]来配置您的用户管理。

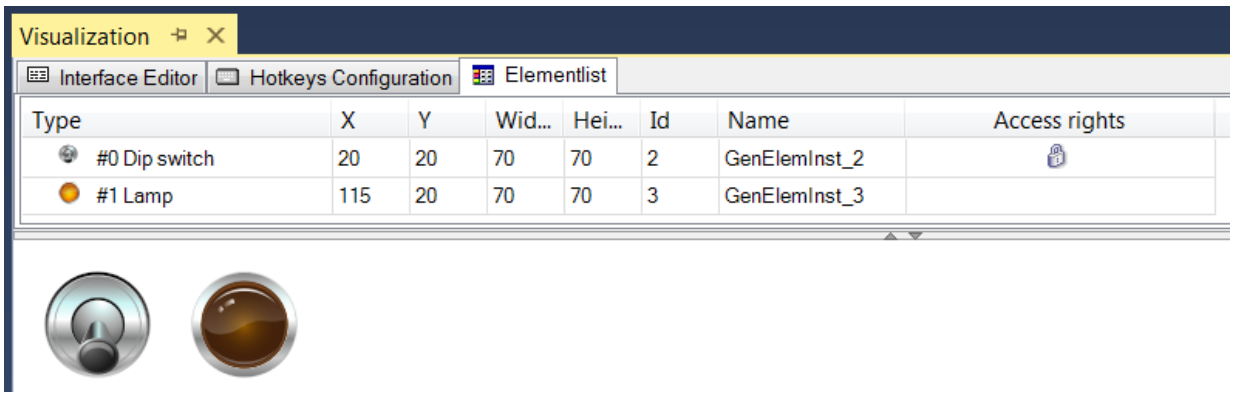
2. 通过使用输入配置 [▶ 399]对元素进行编程，将库“VisuUserManagement”提供的对话框集成到您的可视化中。定义的事件会触发 1 个带有预编程输入配置的对话框。您可以对自己的对话框进行编程。



3. 通过在“Property”（属性）视图中设置属性“Access rights”（访问权限），对可视化元素进行编程。因此，行为与组相关。



在元素列表中会突出显示具有有限权限的元素。



包含用户管理数据的 CSV 文件

作为 CSV 文件，用户管理数据以下列格式存储：

- 用户组：

```
ID;group name;automatic logoff TRUE/FALSE;logoff time;unit logoff time;permission to change user data TRUE/FALSE
```

- 用户：

```
login name;full name;password encrypt TRUE/FALSE;password;group ID;user deactivated TRUE/FALSE
```

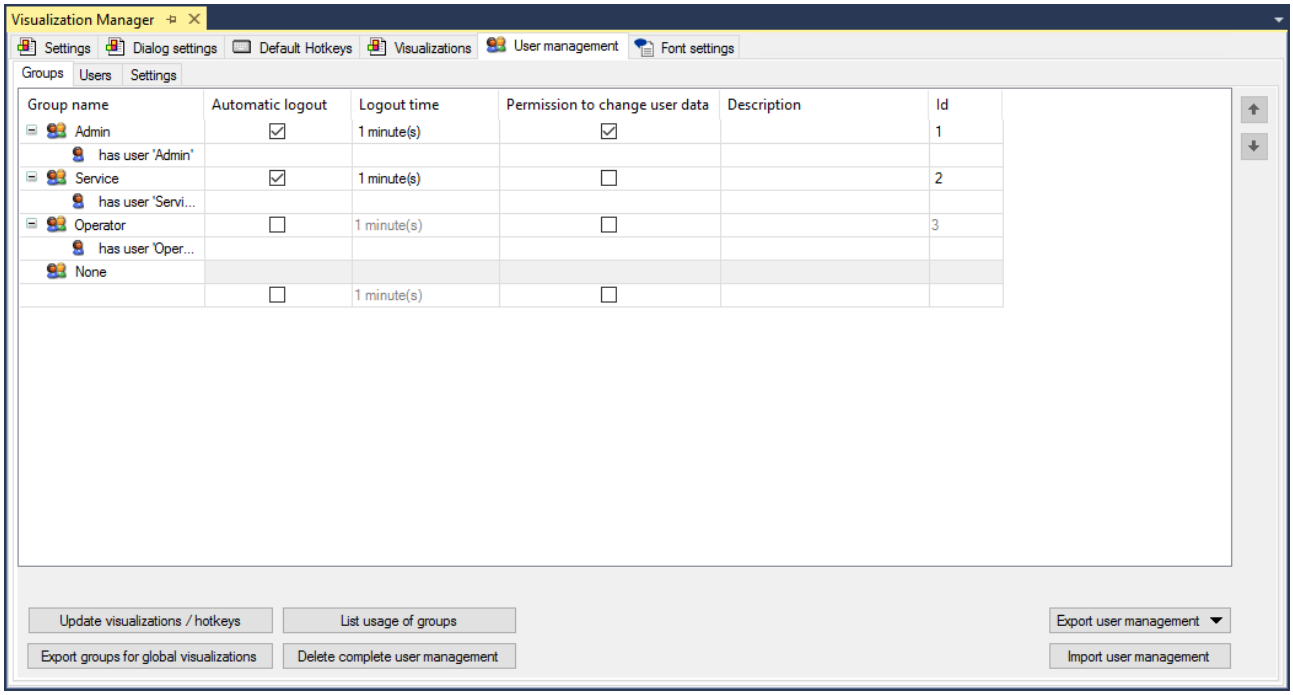
使用此格式，可使用您选择的任何工具来编辑用户管理数据。如果“密码加密”被设置为 FALSE，则可以输入未加密的密码，如示例中用户“Hugo”所示。在导入之后，密码会立即加密。

示例：

```
V1.0.0.1
Usergroups:
1;Admin;TRUE;1;Minute;TRUE
2;Service;FALSE;5;Minute;FALSE
3;Operator;FALSE;1;Minute;FALSE
0;None;FALSE;1;Minute;FALSE
User:
HansM;Hans Mayer;TRUE;F9307D9940B6F7D78320E7E008377593;1;FALSE;administrator
PeterS;Peter Schmidt;TRUE;C5972629BF18E0E82D06FFF29B5BADFF;2|3;FALSE;team leader 1
UllaM;Ulla Müller;TRUE;569D35AC3272623AECDDCA021916C2AB;2|3;FALSE;team leader 2
ElkeF;Elke Fischer;TRUE;C634F54AF9343142159FE0435D93929D;3;FALSE;operator team 1
PaulK;Koch;TRUE;01E2CBD4AE5442D9EACE33669549A3CC;3;FALSE;operator team 2
```

15.2.5.1 组

在“Groups”（组）视图中，所有组均按层级顺序列出。



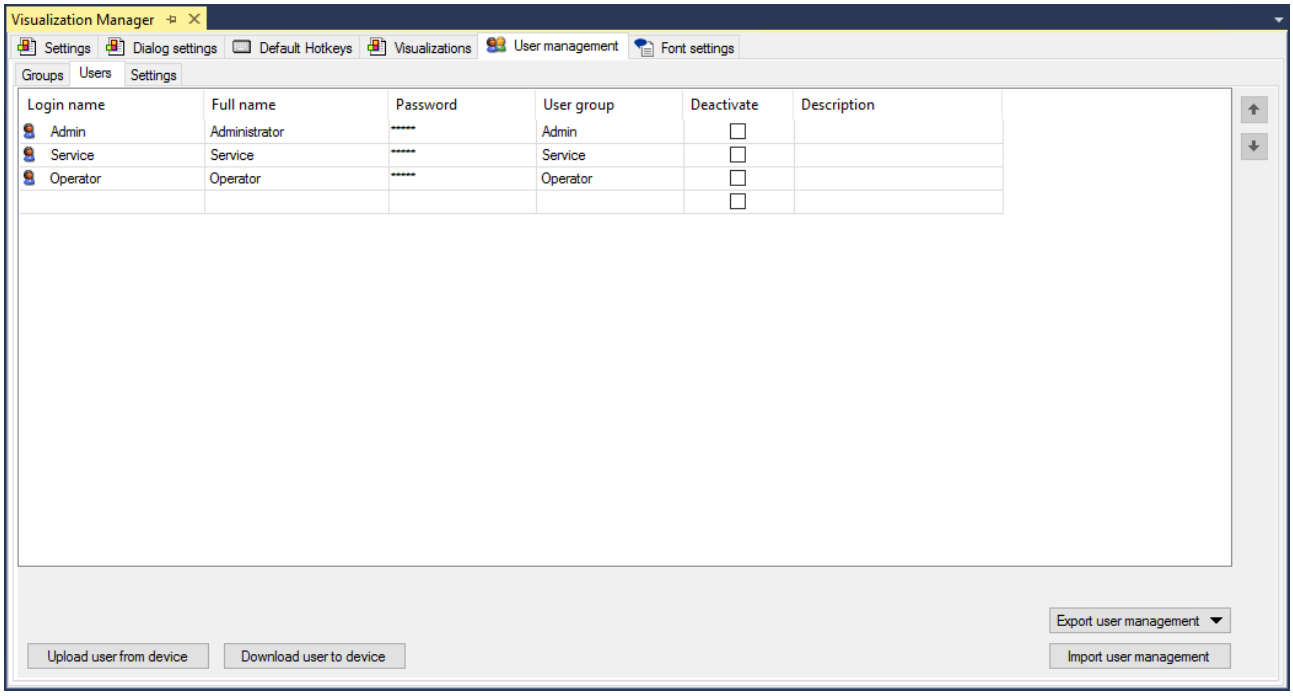
组名称	如果展开组节点，则会显示分配给该组的所有用户。
自动退出	勾选该复选框，以定义固定的退出时间。
退出时间	输入用户退出的时间。使用行编辑器输入数字，使用选择列表设置单位。
更改用户数据的权限	勾选该复选框，允许该组在可视化在线时编辑用户数据。
描述	您可以在此处输入有关组的注释或说明。该文本只在编程系统中提供，不会在运行时加载。
ID	每个组的唯一 ID。由系统自动发布。
添加组	如要创建新组，可点击进入 Group（组）列中表格末尾的行。
删除组	通过选择相应表格行中的字段来选择组。按 [Del] 可删除该行以及与其相关的组。组 None 无法删除。

按钮

更新可视化/热键	打开对话框“Update visualizations and hotkeys”（更新可视化和热键）。 如果在可视化或热键已具有受限访问可能性的某个时间点更改组，则应进行更新。
列出组的使用情况	具有受限访问权限的可视化和热键的列表。 在 Messages（消息）视图中会显示该列表。
为全局可视化导出组	点击此按钮可将上面定义的组传输到 Tools > Options > TwinCAT > PLC Environment > Visualization（工具 > 选项 > TwinCAT > PLC 环境 > 可视化）用户管理中，在“Use the following user group list”（使用以下用户组列表）下会列出它们。
删除全部用户管理	用户管理会被删除，但会保留元素的访问权限。
导出用户管理	1 个标准对话框会打开，从该对话框中可将 CSV 文件保存在任何名称的任意目录下。
导入用户管理	1 个标准对话框会打开，从该对话框中可从任何名称的任意目录中加载 CSV 文件
	如要按一定顺序按层级组织组，可点击向上箭头图标将组向上移动 1 行，或者点击向下箭头图标将组向下移动 1 行。表格中的顺序反映了能够以这种形式加载到运行时且在在线模式下可用的层级顺序。 层次结构较高的组对元素的访问权限不能少于层次结构较低的组。

15.2.5.2 用户

在“Users”（用户）视图中列出了所有用户。



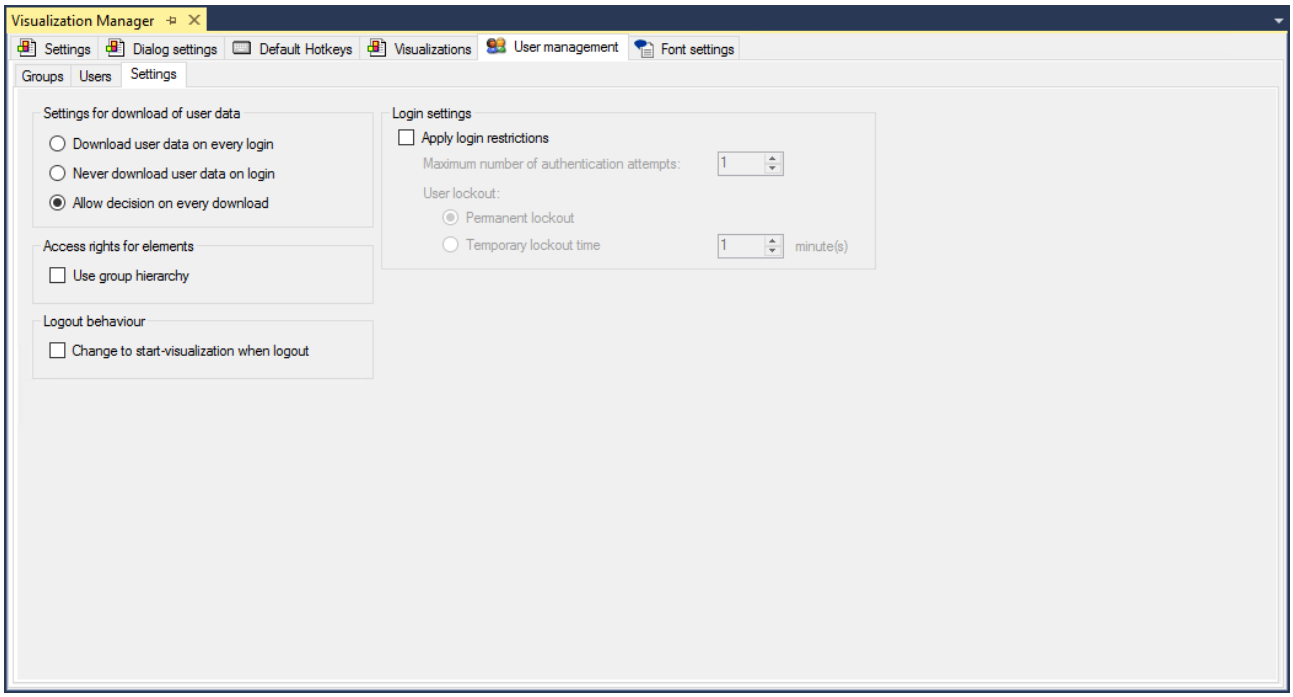
登录名	输入 1 个唯一且可编译的名称，用户可以使用该名称在运行时登录可视化。
全名	输入用户的全名。在用户管理中，这种情况可能存在不止 1 次。
密码	输入密码，密码将被加密。为每个用户分配其登录名作为默认密码。
用户组	输入用户所属的 1 个或多个组。属于多个用户组的用户可以在可视化中操作或查看某个元素，前提是其中 1 个用户组具有相关权限。有关更多详情，请参见访问权限 [▶ 386]部分。
停用	勾选该复选框，可停用用户。
描述	在这里，您可以输入有关用户的注释或备注。该文本只在编程系统中提供，不会在运行时加载。

按钮

从设备加载用户	从 PLC 上传用户管理数据。任何现有的用户数据都会被覆盖。
将用户下载至设备	将用户管理数据下载至 PLC。现有的用户管理会被覆盖。
导出用户管理	1 个标准对话框会打开，从该对话框中可将 CSV 文件保存在任何名称的任意目录下。CSV 文件包含组和用户的数据。
导入用户管理	1 个标准对话框会打开，从该对话框中可从任何名称的任意目录中加载 CSV 文件并在这里显示，前提是格式兼容。

15.2.5.3 设置

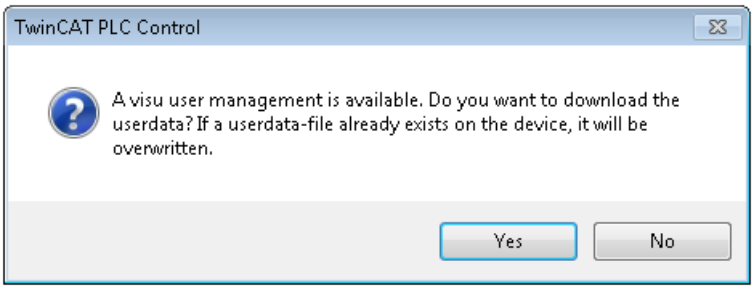
该屏幕显示了下载用户数据的设置。



选择登录程序：

下载用户数据的设置	
在每次登录后进行下载	在每次登录后，都会将存储在编程系统中的用户管理数据下载到 PLC 中。已存在的数据将被覆盖。
在登录时不下载	即使用户管理数据已改变，也永不下载。
每次下载后再做决定	启用由对话框控制的下载。

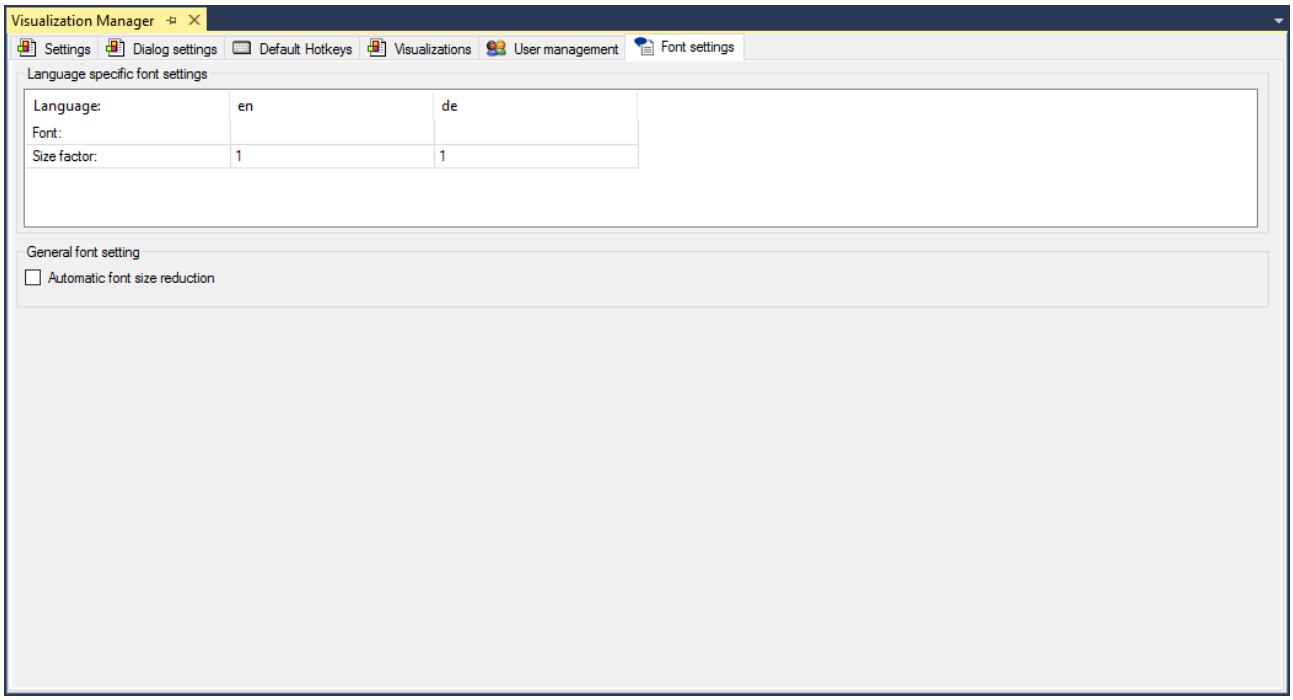
如果选择“Allow decision on every download”（允许在每次下载时做出决定）设置，则在每次下载之前都会显示以下对话框。



元素的访问权限	
使用组层次结构	根据组配置，按层级排列各组。因此，仅可按层级分配访问权限。
退出行为	
在退出时，切换到启动可视化	如果要在用户退出时自动打开启动可视化，可勾选该复选框。
登录设置	
应用登录限制	勾选该复选框，以便在 PLC 可视化中使用设置“Maximum number of authentication attempts”（身份验证尝试的最大次数）和“User blocking”（用户阻塞）。
身份验证尝试的最大次数	定义身份验证尝试的最大可能次数。 可能的值：[1 … 10] 如果达到身份验证尝试的最大次数达，则会暂时或永久阻塞用户，具体取决于“User blocking”（用户阻塞）设置。
用户阻塞	从 2 种阻塞模式中选择 1 种：“permanent”（永久）或“temporary”（临时）： • 永久阻塞：永久阻塞用户。不能永久阻塞具有更改用户数据权限的用户。 • 临时阻塞，时间：以分钟为单位指定时间，可能的值 [1 ... 2880]。总是会临时阻塞具有更改用户数据权限的用户长达 10 分钟，即使在这里输入不同的值也是如此。

15.2.6 字体

该选项卡提供了根据语言调整文本的字体和字号的设置。它们适用于项目中的所有可视化。



语言特定字体设置

语言	1 个可用的预设语言列表。在项目中使用的语言扩展了此项选择。对项目的所有文本列表进行扫描。
字体	可视化使用的字体，取决于语言。
大小因子	该因子会影响可视化中所有文本的字体大小。如果该因子小于 1，则字体就会变小。如果该因子为 1，则显示的所有文本都保持不变，如属性中所定义。 预设值：1
以红色突出显示单元格	由于在项目或库的文本列表中不再提供相应的语言，因此无法在运行时使用该设置。

所选表格行的上下文菜单	
删除	删除相应列。如果列中的设置以红色突出显示，则特别建议这样做。
复制	所有列设置都会被复制到剪贴板。
粘贴	所有列设置都会被剪贴板中的值覆盖。

常规字体设置

自动缩小字体大小	如果无法按照设置的格式将要显示的文本放入文本框内，则字体大小会自动缩小，直到完全可以将文本放入文本框内。 提示：在切换到需要更多空间的语言时，这样可以防止无法完全显示文本。前提是有足够小的字体。

另请参见：

- 可视化中的文本和语言 [▶ 561]

15.3 可视化库

通过库可以提供可作为功能块进行编程的可视化元素 [▶ 369]。如果为项目添加可视化对象 [▶ 367]，则某些可视化库会集成到 PLC 项目中。在当前使用的可视化配置文件 [▶ 367]中会定义这些库的名称和版本。配置文件还精确地指定了在可视化编辑器 [▶ 344]的工具箱 [▶ 352]中可以使用这些库中的哪些元素。

可视化库总是被配置为 1 种特殊类型的“占位符库”。因此，只要没有集成到项目中，就不会指定要使用的库的精确版本。只有这样，当前的可视化配置文件才能确定实际需要的版本。请注意，这种类型的库与设备特定的占位符库不同，占位符库中的占位符会根据设备描述进行解析。

有关基本库的说明，请参见下文，默认情况下，在将可视化对象添加到标准项目中后就会集成这些基本库。它们还会引用这里没有提到的其他库。默认情况下，您现在必须显式添加可视化库或在应用程序中使用它们：

- System_VisuElem3DPath（仅适用于 TwinCAT 3.1 build 4020.0 以下版本）
- System_VisuElemMeter
- System_VisuElems
- System_VisuElemsDateTime（仅适用于 TwinCAT 3.1 Build 4020.0 以下版本）
- System_VisuElemsSpecialControls
- System_VisuElemsWinControls
- System_VisuElemTextEditor
- System_visuinputs
- System_VisuNativeControl

15.4 先决条件

由于 TwinCAT 3 可视化根据 IEC61131-3 标准实现，因此必须在项目中集成特定的库 [▶ 366]，这些库可以提供所需的功能和可视化元素。可视化配置文件 [▶ 367]可用于定义可视化库的选择，并从中选择可在可视化编辑器对象中使用的元素。在添加第 1 个可视化页面时，会将配置文件指定的库自动添加到 PLC 项目中。

15.5 PLC 项目属性

在 PLC 项目属性中可以修改默认设置。例如，在“Visualization”（可视化）类别中可以修改文本列表文件和图像文件的默认目录。右键点击 PLC 项目，即可在上下文菜单的“Properties”（属性）下打开该对话框。

15.6 可视化配置文件

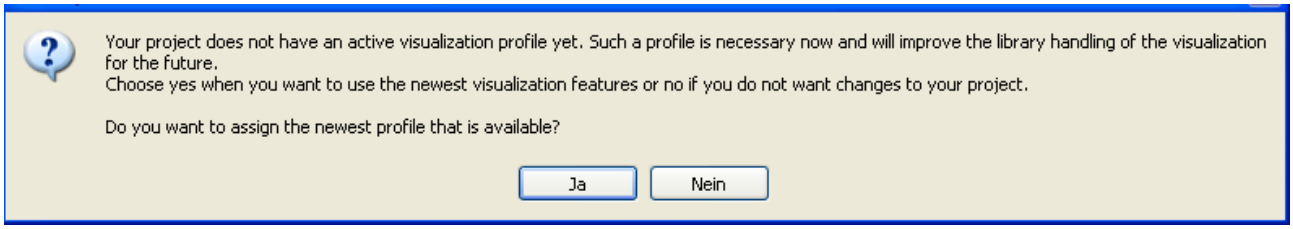
每个可视化项目（即至少包含 1 个可视化对象 [▶ 367]的项目）都必须以 1 个可视化配置文件为依据。该配置文件可定义以下内容：

- 可视化库 [▶ 366]的名称和版本，这些库会在创建 [▶ 367]可视化对象时自动集成到项目中。
- 从可视化编辑器的工具箱 [▶ 352]中提供的集成库中选择可视化元素。

在“Visualization Profile”（可视化配置文件）下的项目属性 [▶ 367]中定义了项目的默认配置文件。随时可以选择不同的配置文件。请注意，所选配置文件适用于项目的所有可视化对象。

如果选择不同的配置文件，则会出现 1 条消息，指明该配置文件可能会阻止登录，而无法进行在线更改或下载。更改配置文件会触发库管理器中有关所需库的自动更新。换句话说，无需手动调整库。

如果打开的旧项目并非基于可视化配置文件创建，则系统会询问您是否希望切换到新的配置文件机制。



如为“是”，则使用最新的配置文件。如为“否”，则在项目设置中输入最旧的可用配置文件（被称为“兼容性配置文件”），但不会在项目中做进一步更改。

15.7 可视化对象

PLC 项目中的可视化 [▶ 344]可能由不同的可视化页面组成。1 个可视化页面可以在另一个可视化页面中使用 [▶ 559]，而且，在运行时可以在不同的页面之间切换 [▶ 560]。每个可视化页面都由 1 个单独的可视化对象表示。

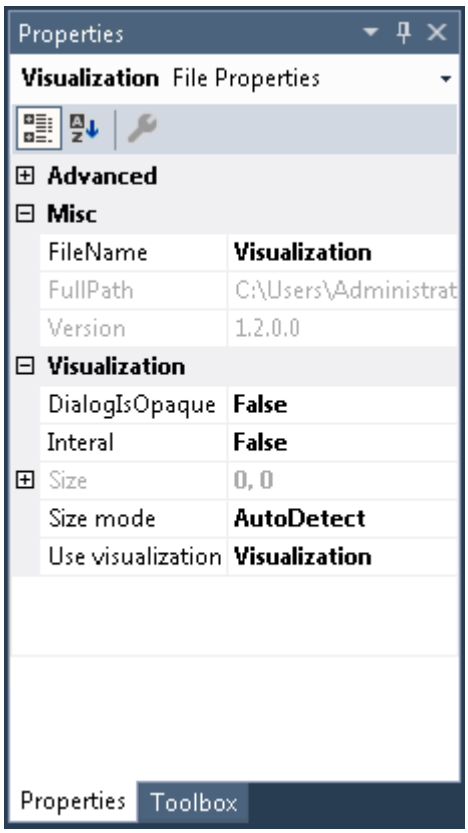
创建可视化对象

如要在 PLC 项目中创建可视化对象，可右键点击 PLC 项目或其中的文件夹（例如，“Visus”），在 PLC 项目中打开上下文菜单。然后，使用菜单项“Add”（添加），选择 1 个可视化对象。

在添加第 1 个可视化对象时，会将可视化库 [▶ 366]和可视化管理器 [▶ 354]自动添加到项目中。在可视化编辑器 [▶ 344]中会打开新创建的可视化对象。

可视化对象的设置

如果选择可视化对象，在“Properties”（属性）窗口中会显示它的设置，如下所示：



杂项

FileName	可视化对象的名称
FullPath	对象的内存位置。此时不能更改内存位置。因此显示为灰色。

可视化

对话框不透明	如果激活此设置，则不会刷新此对话框隐藏的屏幕区域。这对绘图和输入性能有积极的影响。 注意：只有当您绘制的对话框是矩形且完全不透明（即不包含透明部分）时，才能使用此选项。
内部	在 PLC 项目中，被标记为内部的可视化与往常一样可见且可用。如果该可视化保存在库中，则在使用该库的 PLC 项目中，该可视化不再可见或可用。
大小	可视化页面的大小 - 如果在“Size mode”（大小模式）下已选择条目”自动检测“，则该设置为灰色。 • 宽度：以像素为单位的宽度 • 高度：以像素为单位的高度
大小模式	您可以在此处指定大小是否适应以及如何适应可视化页面。 • AutoDetect：确定可视化页面的大小，在该尺寸下，当前包含的所有可视化元素均可见。 • AutoDetectWithBgImage：确定可视化页面的大小，在该尺寸下，当前包含的所有可视化元素和背景图像 [▶ 345]均可见。 • 已指定：在“Size”（大小）下，已指定页面大小。用户应该考虑所有可视化元素（和背景图像，如适用）是否都适合该部分，从而确保完全可见。 如果这些设置产生的纵横比与屏幕大小不匹配，则可视化页面将会显示相应的白边。这样可以防止可视化元素发生几何畸变。
使用可视化作为	在此设置中，您可以为对象指定以下可视化类型之一： • 可视化：在此默认设置下，可视化对象被声明为 1 个单独的可视化页面。 • 对话框：可视化对象表示对话框。如果启用此设置，则该可视化对象可用作其他可视化对象中的对话框 [▶ 374]。 • NumKeybad：可视化对象表示数字键盘/小键盘。如果启用此设置，则该可视化对象可用作数字键盘或小键盘，例如，用于在可视化中描述变量 [▶ 381]。这种数字键盘/小键盘的接口必须与库“VisuDialogs”提供的默认数字键盘/默认小键盘的接口完全相同。

编辑可视化对象

用于创建可视化效果的可视化编辑器 [▶ 344]可与工具箱 [▶ 352]和属性编辑器 [▶ 353]结合使用，前者可提供来自集成元素库的可视化元素，后者则可用于配置可视化元素。如要打开可视化对象，可双击项目树中的对象。

定义启动可视化

在 TargetVisualization [▶ 554] 或 WebVisualization [▶ 556] 对象中必须输入“start visualization”（启动可视化），即在 PLC HMI [▶ 551] 或 PLC HMI Web [▶ 555] 客户端启动时首先显示的可视化对象。

15.8 可视化元素

在打开可视化对象 [▶ 367]后，在工具箱 [▶ 352]中可以使用不同的可视化元素。它们分为以下几类：

- 通用控制 [▶ 385]
- 基础 [▶ 433]
- 灯/开关/图像 [▶ 479]
- 测量设备 [▶ 488]
- 特殊控制 [▶ 528]

15.8.1 常规配置

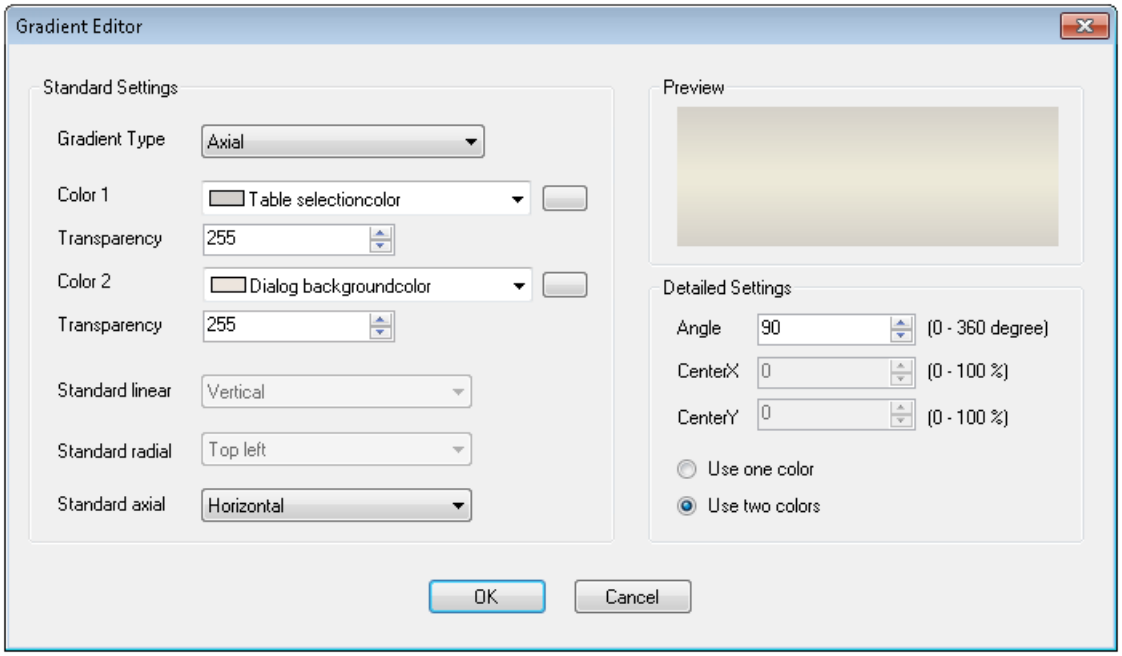
以下 3 个通用对话框可用于不同的可视化元素：

- 颜色渐变对话框 [▶ 370]
- 输入配置 [▶ 371]

- 访问权限对话框 [▶ 384]

15.8.1.1 渐变编辑器

点击“Properties”（属性）窗口中颜色渐变的值字段，打开渐变编辑器。



默认设置

渐变类型	以下 3 种渐变类型可供选择： • 线性 • 径向 • 轴向
颜色 1	第 1 种渐变色 - 可从组合框或颜色对话框中做出此项选择，通过按钮 ... 可以打开颜色对话框。
透明度	0 到 255 之间的值可用于指定第 1 种颜色的“强度”。
颜色 2	第 2 种渐变色 - 可从组合框或颜色对话框中做出此项选择，通过按钮 ... 可以打开颜色对话框。
透明度	0 到 255 之间的值可用于指定第 2 种颜色的“强度”。
标准线性	线性颜色渐变类型的默认设置（渐变角度）： • 水平 • 垂直 • 从左上角开始 • 从右上角开始 • 水平颜色切换 • 垂直颜色切换 • 从左下角开始 • 从右下角开始
标准径向	径向颜色渐变类型的预定义设置（中心位置）： • 中心 • 左上角 • 右上角 • 左下角 • 右下角
标准轴向	轴向颜色渐变类型的预定义设置（渐变角度）： • 水平 • 垂直 • 从右上角开始 • 从左上角开始

详细设置

角度（度）	仅适用于线性和轴向渐变类型
中心 X（%）	中心的 X 位置（0-100%） - 仅适用于径向颜色渐变类型。
中心 Y（%）	中心的 Y 位置（0-100%） - 仅适用于径向颜色渐变类型。
使用 1 种颜色	颜色 1 与亮度不同的相同颜色之间的颜色渐变类型。在 0（黑色）和 100（白色）之间可以设置亮度。
使用 2 种颜色	在 2 种选定颜色“颜色 1”和“颜色 2”之间的颜色渐变类型。

15.8.1.2 输入配置

在输入配置对话框中可以为 1 个特定的事件 [▶ 399]（例如，OnMouseClick）定义下文所述的 1 个或多个后续动作。点击“Properties”（属性）窗口中该事件的值字段，打开对话框。这些选择的后续动作的事件的名称会在灰色行中显示为标题。

如要添加后续动作，必须首先在对话框的左侧选择相应的动作。然后，按带有右箭头图标的按钮，即可添加动作。如要删除某个动作，可在右侧选择该动作，然后按带有左箭头图标的按钮。

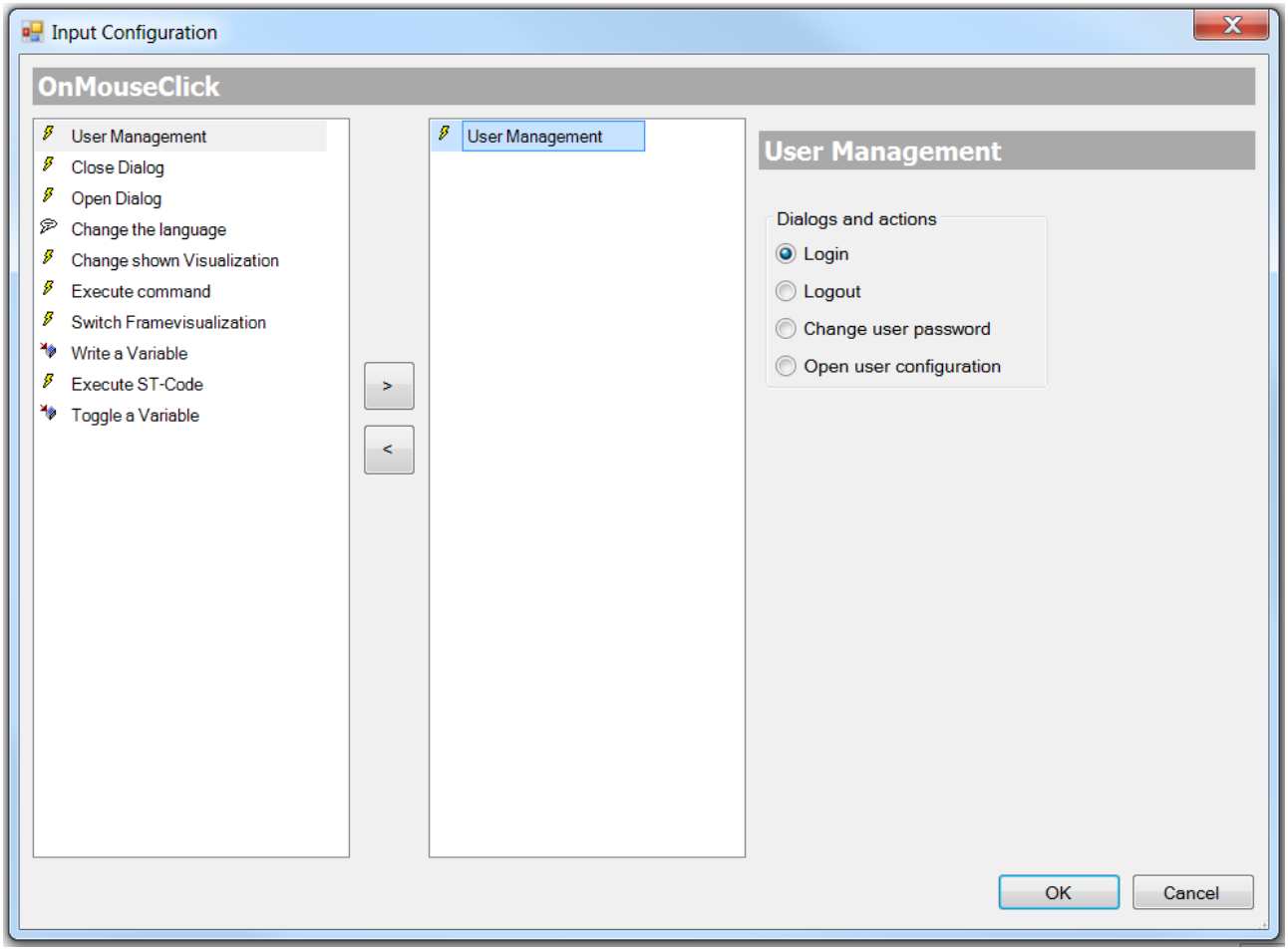
可提供以下动作：

- 用户管理 [▶ 372]

- 关闭对话框 [▶ 373]
- 打开对话框 [▶ 374]
- 语言切换 [▶ 375]
- 更改显示可视化 [▶ 376]
- 执行命令 [▶ 377]
- 切换边框可视化 [▶ 379]
- 编写 1 个变量 [▶ 381]
- 执行 ST 代码 [▶ 383]
- 切换变量 [▶ 383]

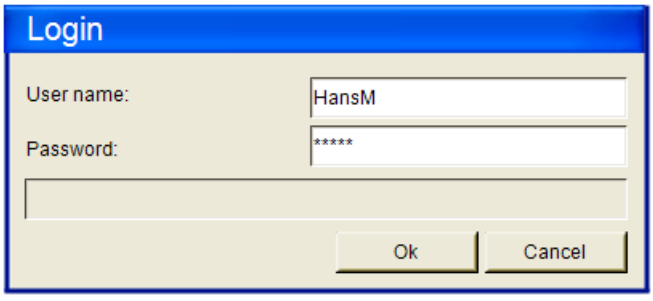
用户管理

只有首先创建 用户管理 [▶ 360] 或手动添加“VisuUserManagement”库，才能选择用户管理作为后续动作。



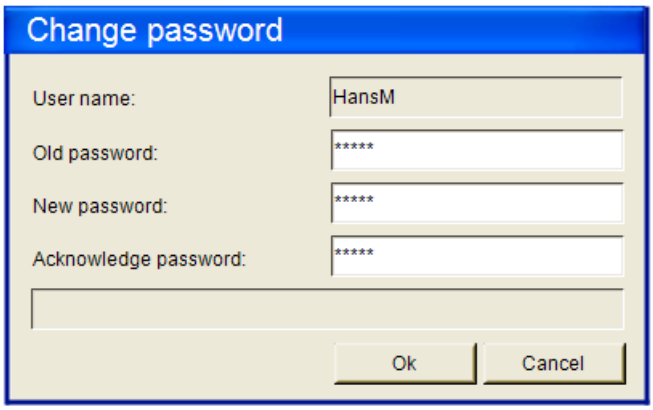
通过用户管理，可以添加以下 4 个标准对话框和动作。它使可视化用户能够在在线模式下执行以下动作：

- 登录

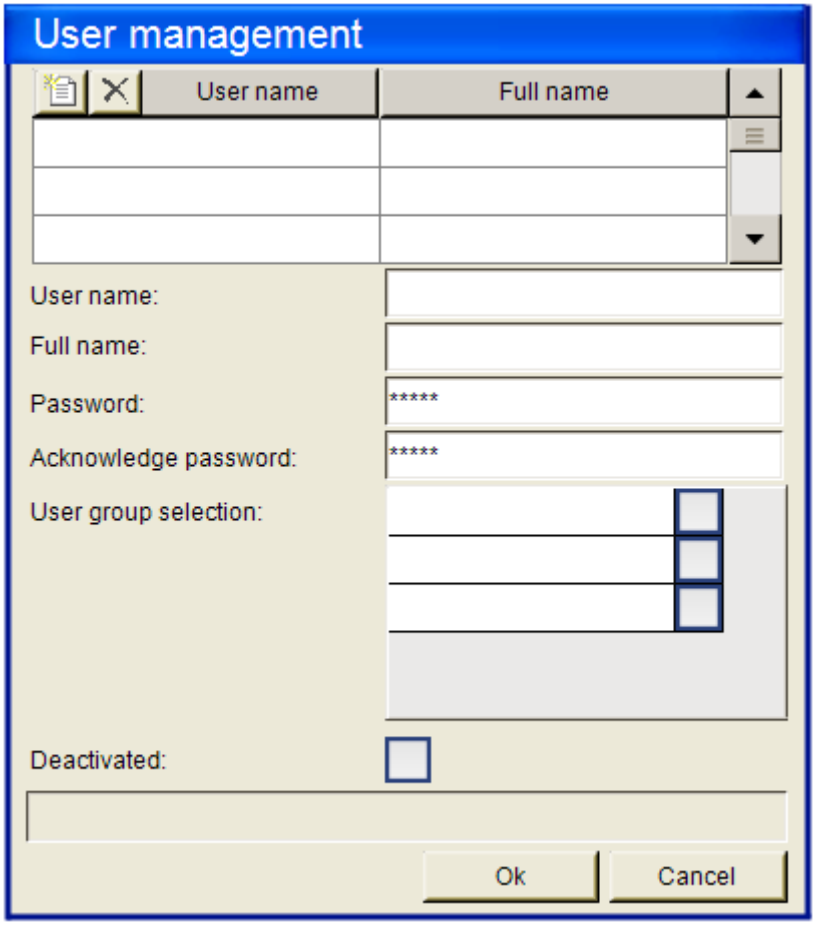


- 退出

- 更改用户密码

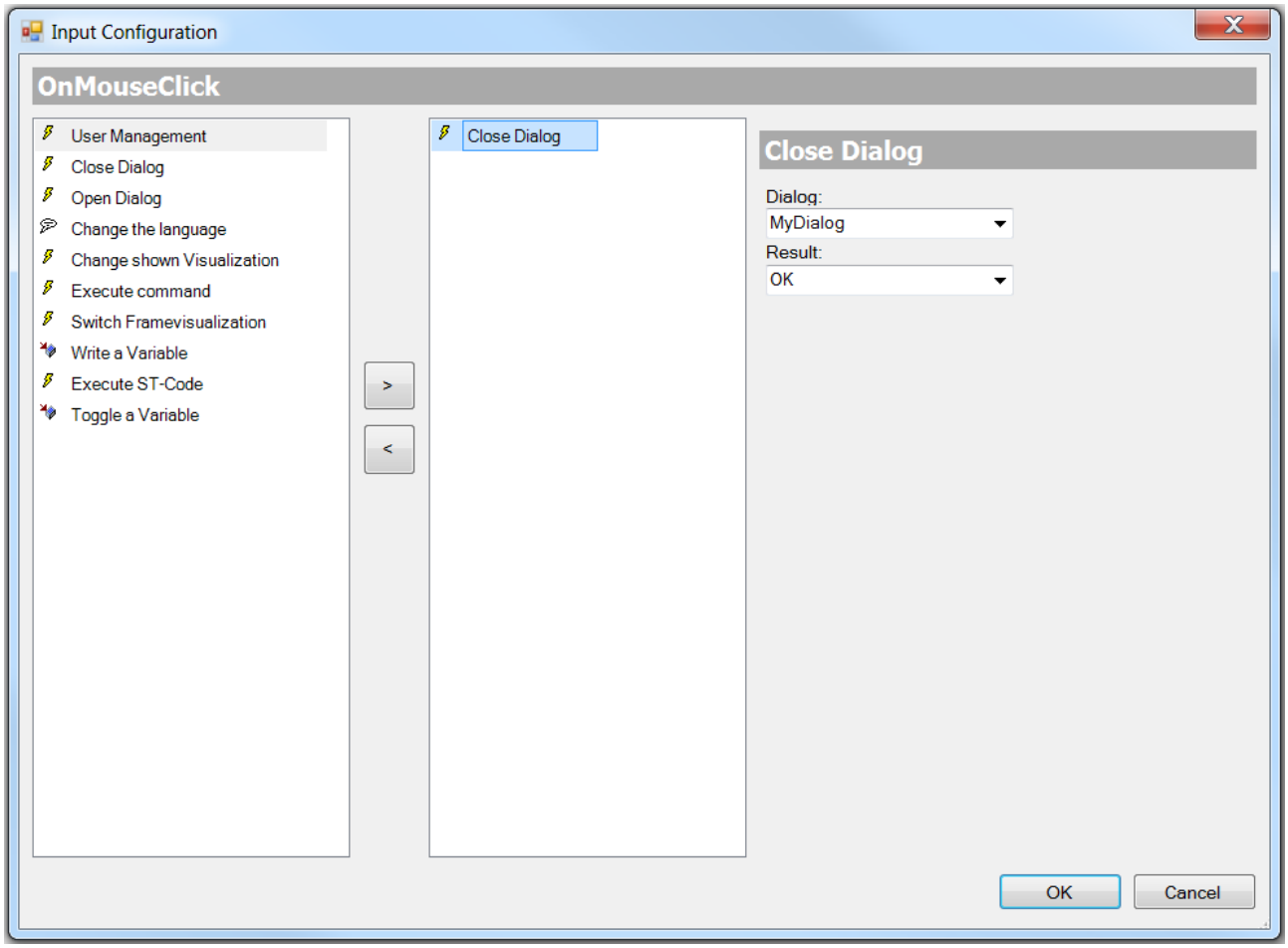


- 打开用户管理



关闭对话框

该选项可用于指定在事件发生后，应该使用指定的结果关闭指定的对话框。从提供所有当前可用的输入对话框的选择列表中，选择所需的对话框。



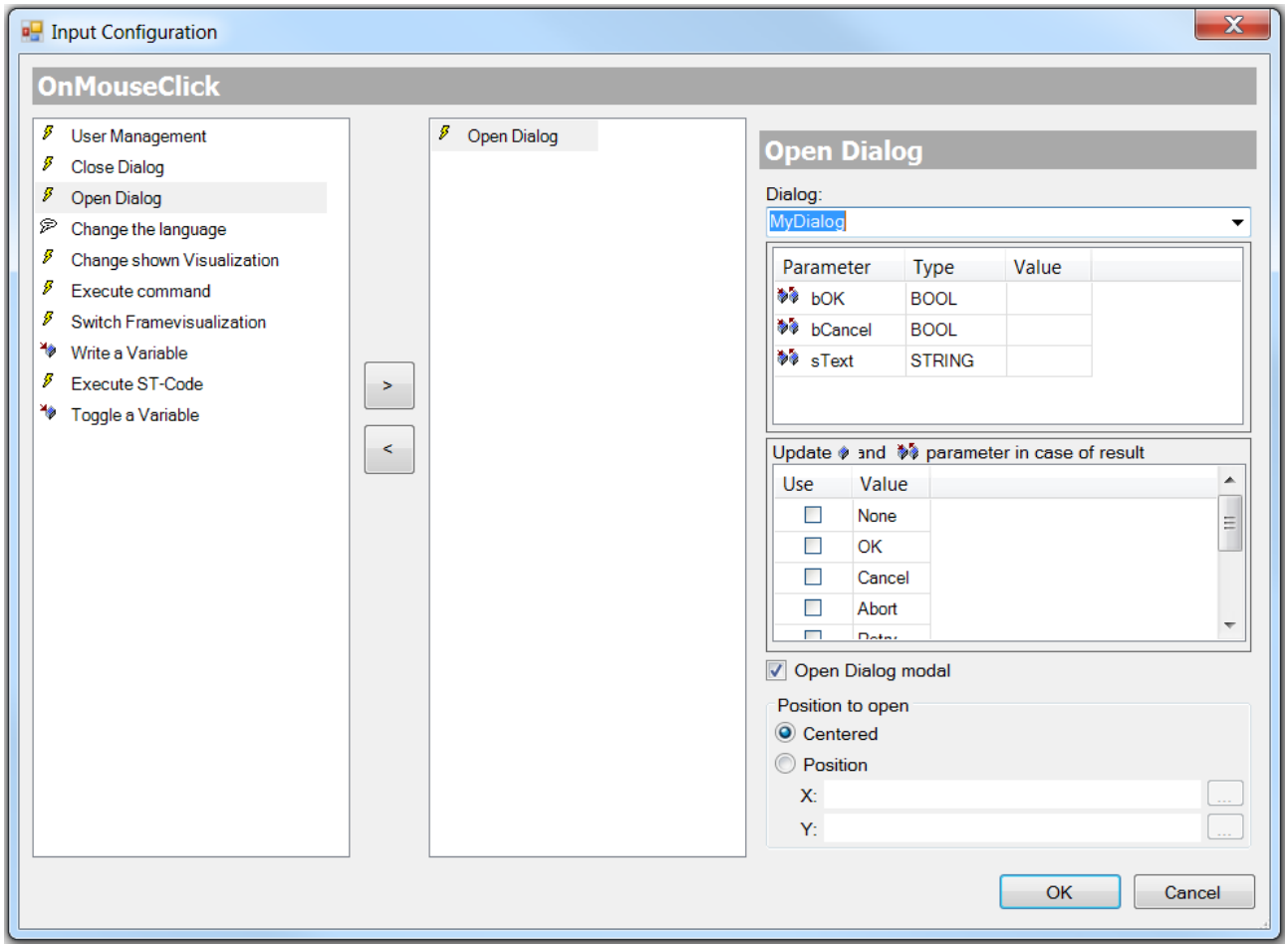
对话框	从选择列表中选择对话框
结果	该列表可提供在对话框中使用的标准选项。这些标准选项需要用户响应。 • 确定 • 取消 • 中止 • 重试 • 忽略 • 是 • 否

请注意，您可以检索最后关闭的对话框的结果，然后在相同可视化的任何元素中配置相应的响应。为此，可使用配置选项“OnDialogClosed”。

打开对话框

您可以在此处定义在 1 个鼠标动作后打开的对话框，该对话框由另一个（标准或用户创建的）可视化表示。选择列表可提供在对象属性 [▶ 367]中输入“Dialog”（对话框）目的的所有可视化。它使用户创建的对话框能够在可视化中用作用户输入掩膜。

如果在项目中集成 VisuDialogs.library，则至少可以使用标准对象“VisuDialogs.Login”和“VisuDialogs.FileOpenSave”。此外，还可以使用用户创建的输入对话框。

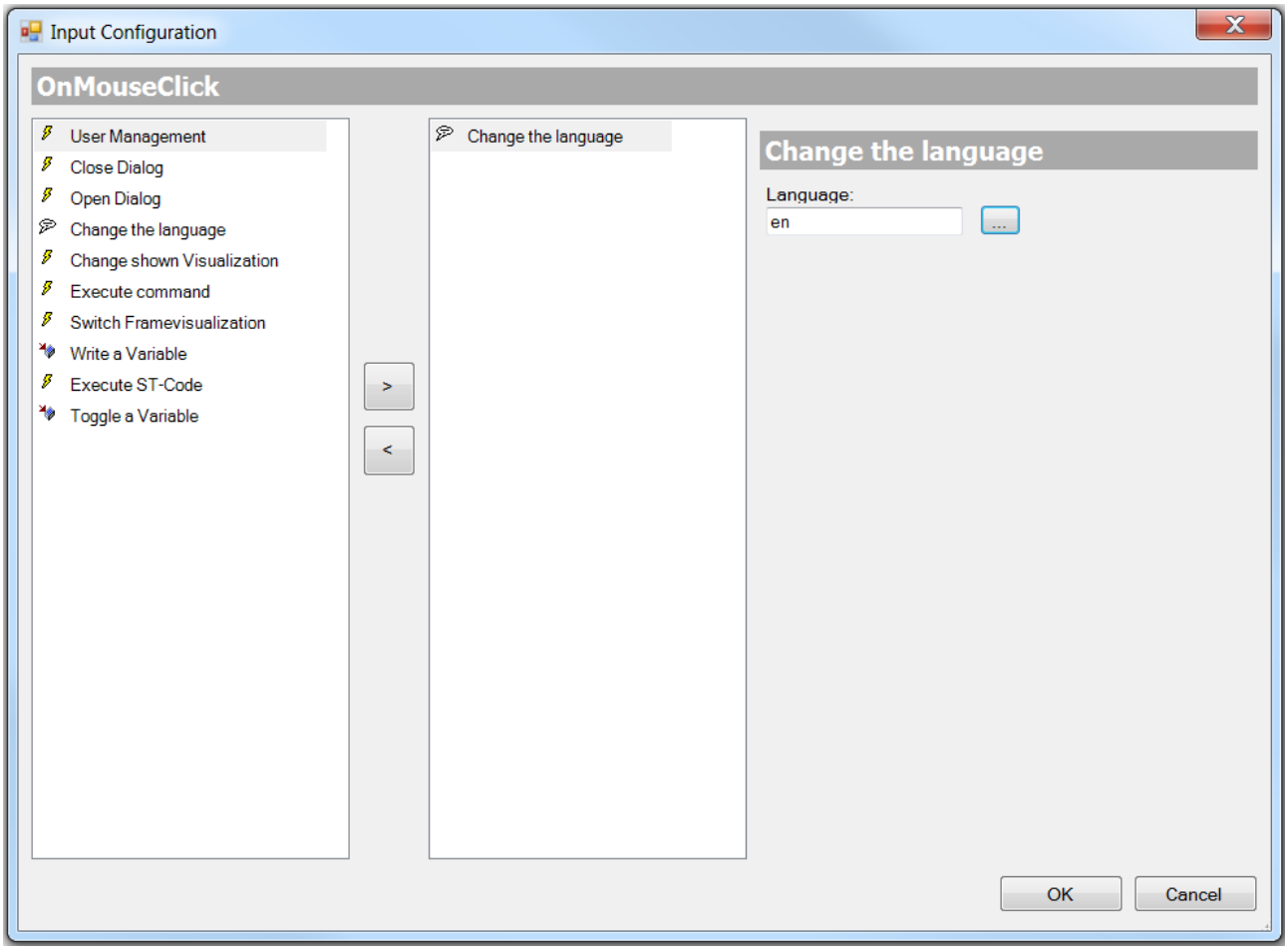


对话框	从选择列表中选择对话框
参数	对于所选的对话框可视化，将会显示在接口编辑器 [▶ 345]中定义的输入（VAR_INPUT）、输出（VAR_OUTPUT）和输入/输出参数（VAR_IN_OUT）。（参数、类型、值）在最后一列（即“值”）中，参数可与 PLC 中的变量相关联。在打开可视化对话框时，变量值将被写入参数（VAR_INPUT、VAR_IN_OUT）中。（VAR_OUTPUT、VAR_IN_OUT）。
事件	用户必须指定使用哪种对话框可视化结果作为编写其输出和输入/输出参数的基础：勾选 Use（使用）列，可激活所需的结果值。可能的值： • 无 • 确定 • 取消 • 中止 • 重试 • 忽略 • 是 • 否

请注意，在对话框关闭之前，不能编写对话框可视化的输出和输入/输出参数！在此之前，这些值只存储在堆栈中，即它们不会被视为引用，而是副本。

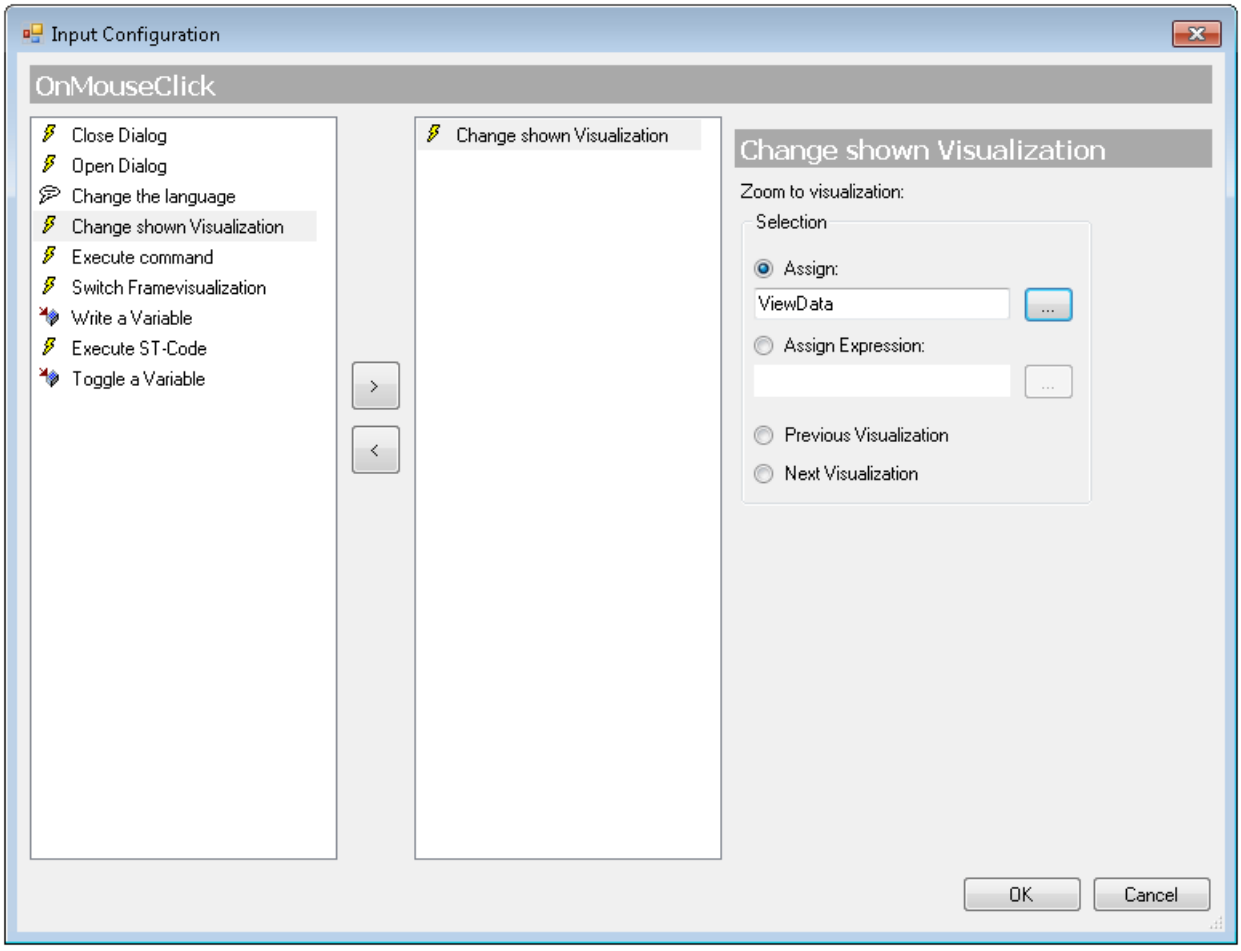
语言切换

您可以在此处指定用于显示可视化文本的界面语言。使用在相应的文本列表 [▶ 127]中使用的语言代码。或者，点击 ... 按钮，在输入向导中选择所需的语言。（另请参见语言和文本 [▶ 561]）



更改显示可视化

您可以在此处选择更改当前显示的可视化页面作为后续动作。（另请参见“在可视化页面之间切换”部分）

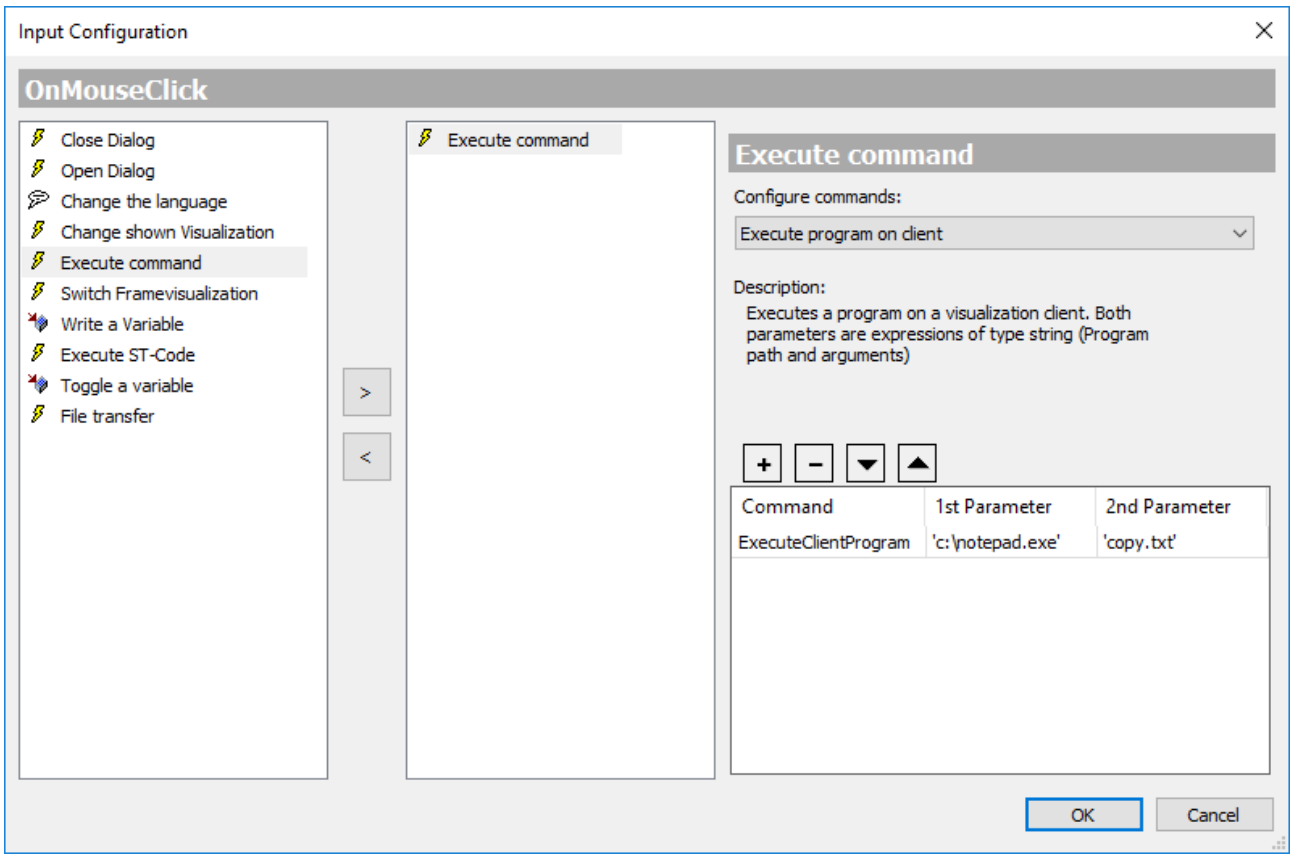


从以下选项中选择 1 个，定义在在线模式下执行鼠标动作时显示的可视化页面：

分配	可直接进入可视化页面。最好使用输入向导，通过按钮 [...] 可以打开它。
分配表达式	您可以在此处指定 1 个字符串类型的变量，该变量可由 PLC 项目使用并可提供可视化页面的名称。 示例：sVisualizationName : STRING := ‘MyVisualization ‘;
	在内部可以存储可视化页面通过用户输入依次显示的顺序。该信息可用于以下 2 个选项。
上一个可视化	再次显示先前已显示的可视化页面。如果之前没有调用可视化，则会继续显示当前的可视化。
下一个可视化	显示记录的可视化更改序列中的下一个可视化页面。因此，只有在通过“Previous Visualization”（上一个可视化）进行上一个可视化更改的情况下，才有可能这样做。

执行命令

您可以在此处定义 1 个或多个要在鼠标动作后执行的命令。



从“Configure commands”（配置命令）列表中选择 1 个命令，然后按  按钮添加到对话框下部的表格中。该表格包含输入配置的所有选定命令。

对话框的中间部分会显示列表中所选命令的简要说明。在表格的“1st Parameter”（第 1 个参数）和“2nd Parameter”（第 2 个参数）列中，添加命令参数。“Command”（命令）列会显示内部命令名称。另请参见下表，了解各个命令的说明。

使用  按钮，从列表中删除当前所选条目。随后，当用户输入可视化元素时，系统会根据配置命令在表格中的排列从上到下逐一执行它们。如要更改顺序，可使用  和  按钮移动条目。

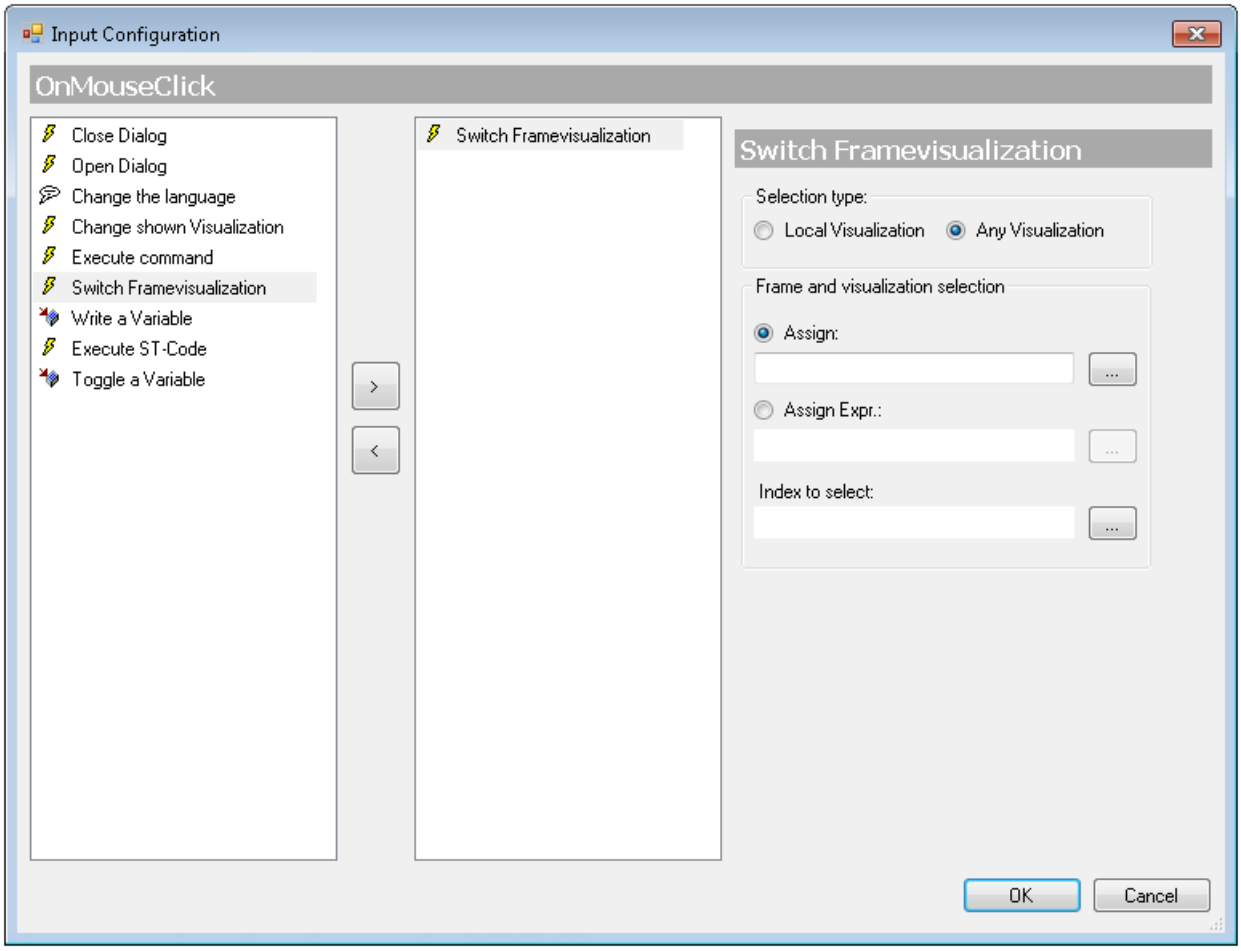
可提供以下命令：

在控制器上执行程序 在客户端上执行程序	在控制器或可视化客户端上执行指定程序（*.exe）。 第 1 个参数：字符串，程序文件的路径（示例：“c:\programs\notepad.exe”） 第 2 个参数：字符串，要执行的程序的参数，例如，程序要打开的文件的名称（示例：copyfile.txt）
打印	打开标准对话框“Print”（打印），在该对话框中可以设置打印机区域和打印机参数等。此外，还可以将当前可视化打印出来。该命令仅支持 Windows 下的 PLC HMI。
导航至 URL（WebVisu）	要求：可视化作为 PLC HMI Web 执行。 当输入事件发生时，可视化会导航至指定 URL 的网页 第 1 个参数：网址 URL — 作为字符串类型的变量，以编程方式设置起始页面。（示例：MAIN.sUrl） — 作为字面量，用单引号标出。（示例：http://www.beckhoff.com） 第 2 个参数：如果未指定参数，则会在新窗口或新选项卡中显示网页。如果指定“替换”，则 PLC HMI Web 将被网页替换。

切换边框可视化

要求：项目中的可视化至少有 1 个边框元素，该元素通过边框选择 [▶ 478]分配给各种可视化。在该边框内，可视化的索引为 0、1、2 等。默认情况下，已分配的可视化中的第 1 个（索引 0）会在线显示在边框中。

在当前对话框中，您现在可以将特定边框元素中的当前可视化元素上的鼠标动作与显示特定相关可视化的边框元素建立关联。因此，当前元素可用于对边框中的可视化切换进行编程。（另请参见“在边框内的可视化页面之间切换 [▶ 559]”部分）



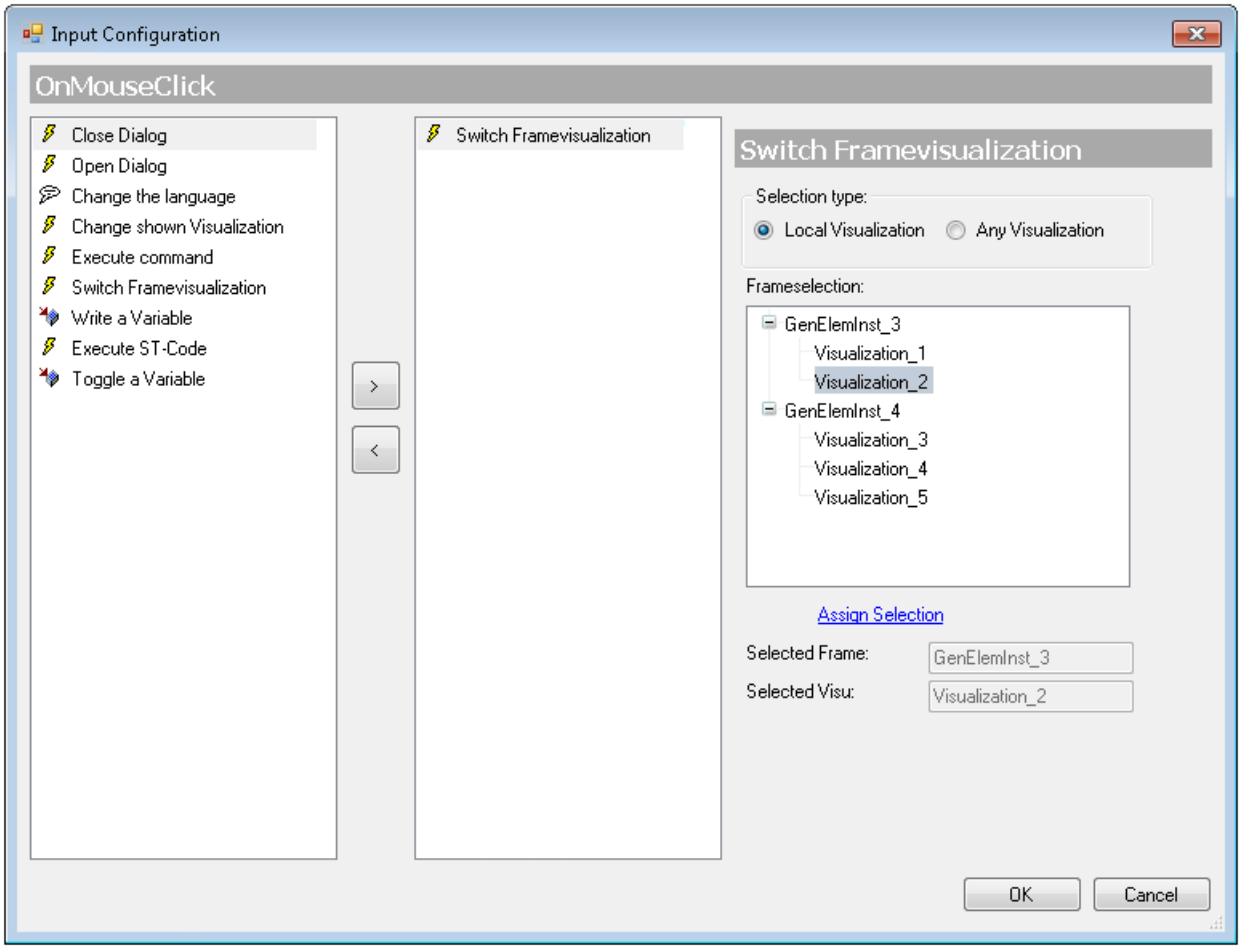
受影响的边框元素的选择类型

- 可以限制当前的可视化，从而简化配置对话框，但没有动态配置选项。（→ 本地可视化）
- 可以扩展至项目中的所有可视化，包括边框和已显示的可视化的动态定义（→ 任何可视化）

本地可视化

只能处理当前可视化的边框，因此，该边框在“Frame selection”（边框选择）字段中可用（见下文）。这是 1 个简单的对话框，可用于在本地可视化中快速、直接地进行配置。但是，无法通过项目变量来指定所需的可视化。如果需要这样操作，可选择“Any Visualization”（任何可视化）。

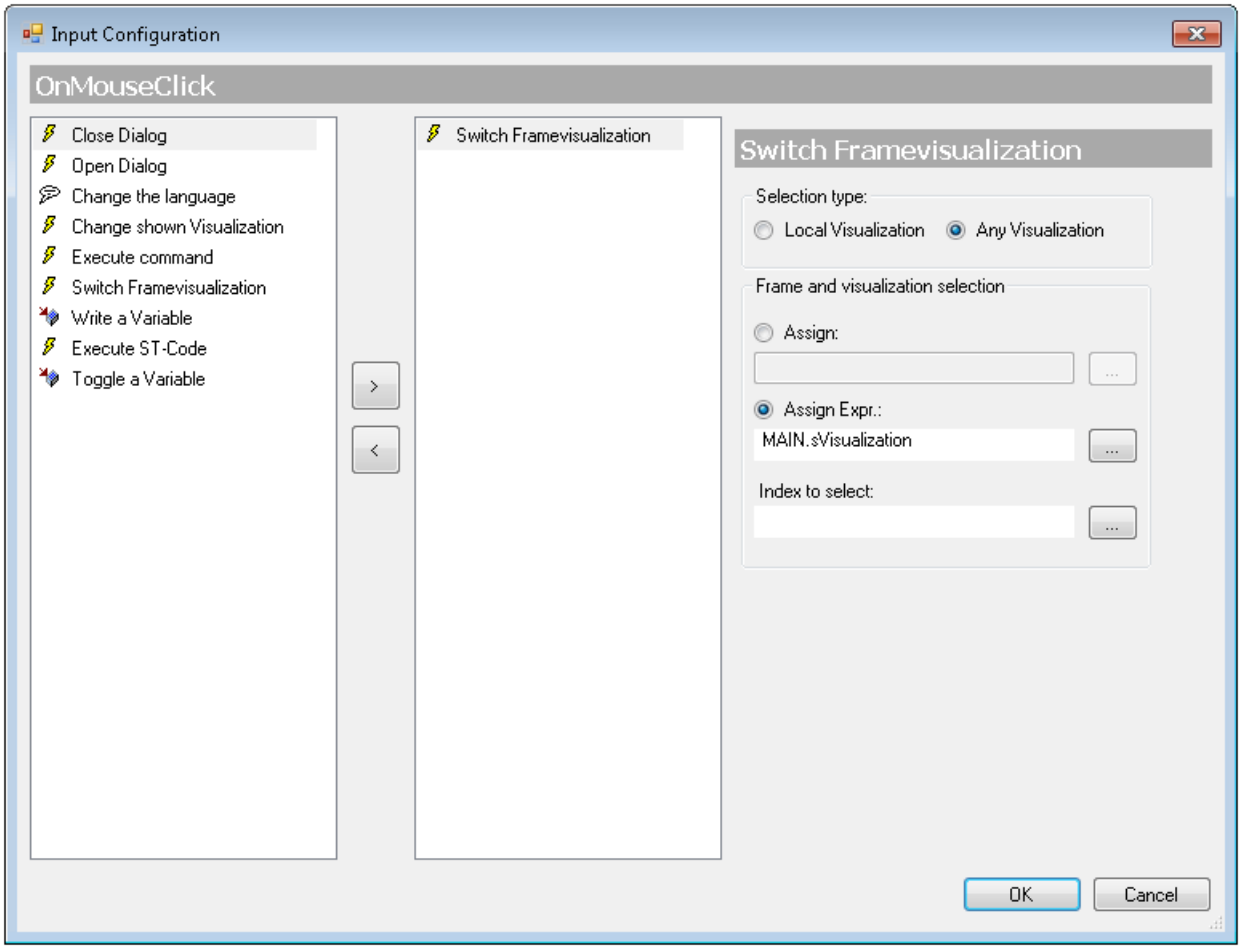
边框选择会显示本地边框元素，并在相关可视化下方缩进。



从受影响的边框中选择所需的可视化，该可视化将在鼠标动作后显示。点击“Assign Selection”（分配选择），可保存设置。然后，在“Selected Frame ”（已选边框）和“Selected Visu”（已选 Visu）字段中会显示当前选择。

任何可视化

可提供项目的所有边框及其分配的可视化。在这种情况下，也可以通过项目变量（即动态地）指定边框和可视化。

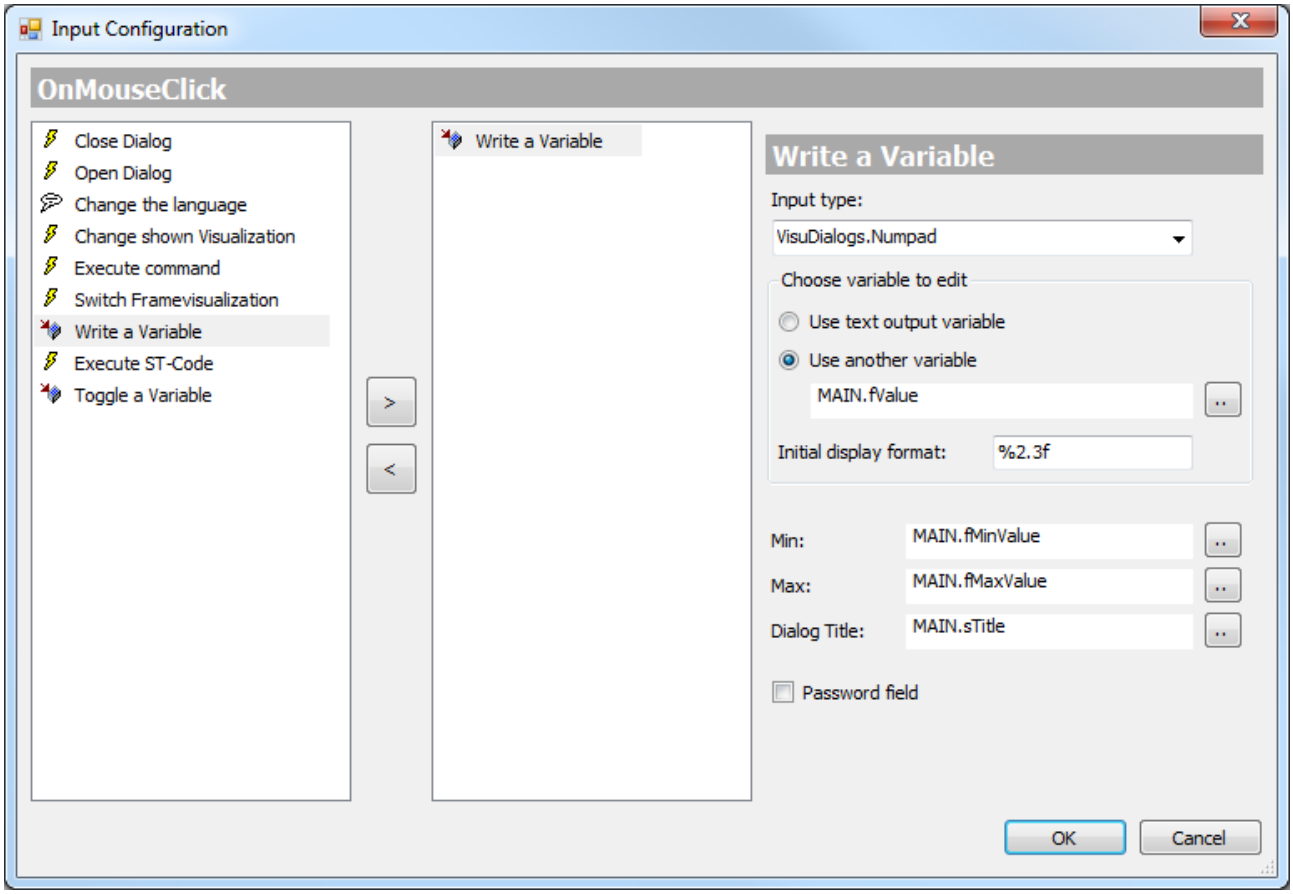


然后，通过以下字段和选项进行“Frame and visualization selection”（边框和可视化选择）：

直接分配	如果启用该选项，则可直接输入受影响的边框元素的名称和相应的可视化。按 … 可激活输入助手。 示例：Visualization_1.MyFrame
通过表达式赋值	启用该选项可使用字符串类型的项目变量来指定边框元素。按 … 可激活输入助手。然后，该变量必须返回边框元素的名称和相应的可视化名称。 示例：sVisualization : STRING := ‘Visualization_1.MyFrame’;
	通过索引可以指定在鼠标动作后要在所选边框中显示的可视化。该可视化索引是 1 个从 0 开始的递增整数，在边框选择 [▶ 478]中被分配给 1 个边框。因此，默认情况下，第 1 个可视化总是首先显示在此列表中。
要选择的索引	直接或通过应用程序中的项目变量输入所需可视化的索引。按 … 可打开输入助手，从而选择变量。

编写 1 个变量

如果某个可视化元素存在“编写 1 个变量”类型的输入配置，那么，在执行相应的鼠标动作之后，该元素会提供 1 个输入值的选项。然后可以通过数字键盘或小键盘将该值作为字符串输入，再将其写入在此处指定的输入配置对话框中的项目变量。根据项目变量的数据类型，该值可被解释为文本或数字。



从对话框右侧的选择列表中，选择以下输入类型之一：

标准	使用默认输入类型。在可视化管理器的设置中可以指定默认值。
文本输入	打开值/文本输入字段。需要使用键盘来输入值。
有限制的文本输入	打开值/文本输入字段。此外，在顶部会显示输入限值。如果它们超出限值，则以红色字体显示。需要使用键盘来输入值。
VisuDialogs.Keypad	鼠标动作会打开 1 个仿真键盘。点击相应按键，您可以输入字符串。
VisuDialogs.Numpad	鼠标动作会打开 1 个仿真数字键盘。点击相应按键，您可以输入数字序列。
	除了默认输入类型之外，还可提供所有可视化，这些可视化在其属性中被定义为“数字键盘”、“键盘”或“输入配置对话框”。（请参见对象属性 [▶ 367]）

如要使用小键盘或数字键盘，必须在库管理器中添加“VisuDialogs”库。

选择要编辑的变量：

使用文本输出变量	该值会被写入在可视化元素中被指定为文本输出变量的变量中。
使用另一个变量	指定项目变量。按 [...] 可打开输入助手。
初始显示格式	指定带有格式化数据的占位符 示例：%2.3f
最小	要输入的变量的下限值
最大	要输入的变量的上限值
对话框标题	要在对话框标题处显示的对话框标题，可被指定为文本或通过文本变量进行指定
密码字段	如要显示“*********”而不是文本（例如，在密码的情况下），可勾选 Password（密码）字段的复选框。

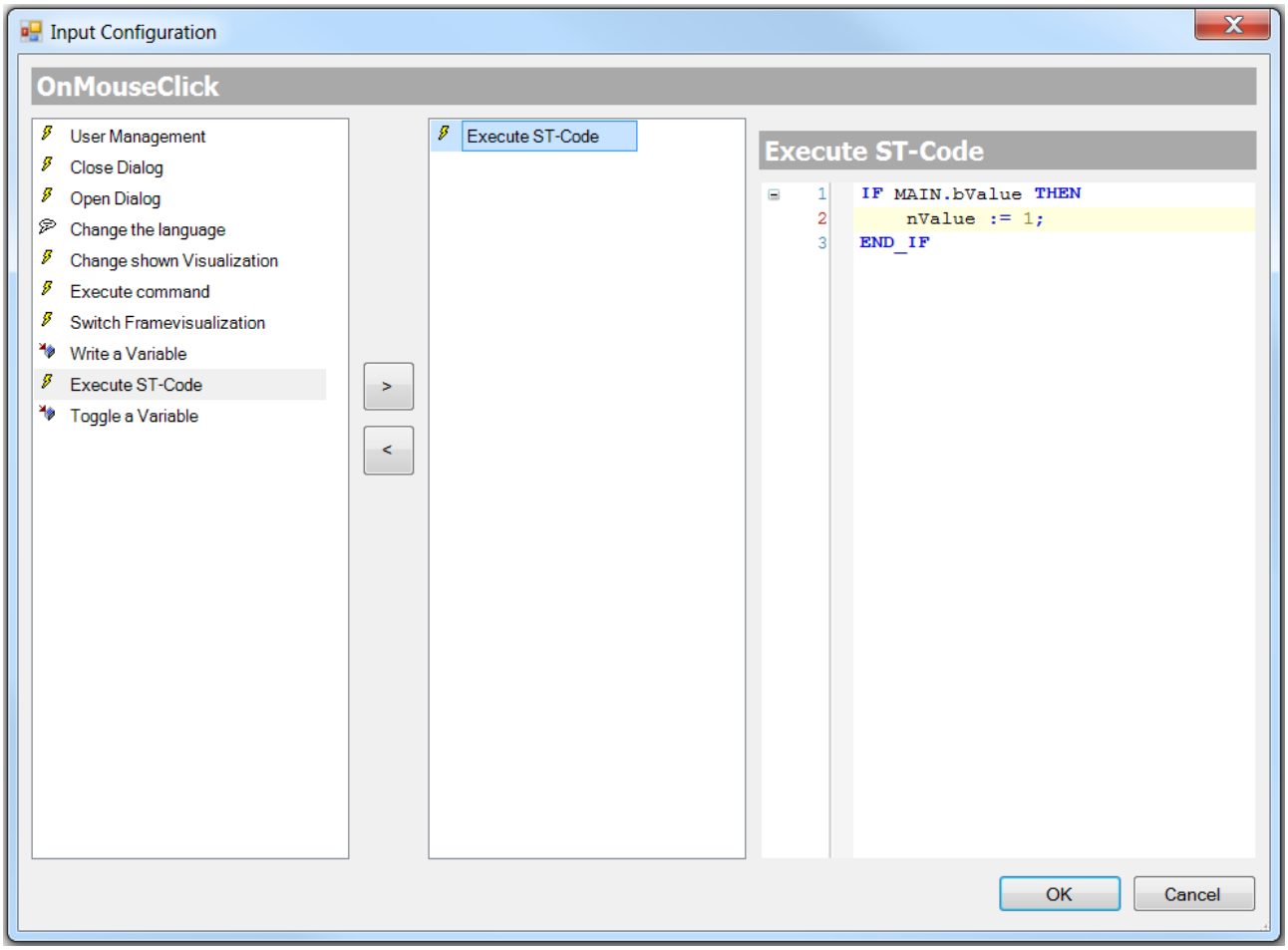
示例

我们假设上面显示的对话框是 1 个矩形元素的配置对话框。如果在在线模式下按下该矩形上的鼠标按钮，则会出现 1 个数字小键盘，并显示由变量“MAIN.sTitle”返回的标题。

例如，点击 1、2 和 3，可输入值“123”。在对话框的上部会显示该值，同时也会显示条目的最小值和最大值，通过“MAIN.fMinValue”和“MAIN.fMaxValue”可以进行指定。使用鼠标点击 OK（确定）确认输入后，“123”将被写入变量“MAIN.fValue”。如果将“fValue”定义为 STRING 变量，则为其赋值“123”。如果“fValue”是 1 个数值变量，则为其赋值 123。

执行 ST 代码

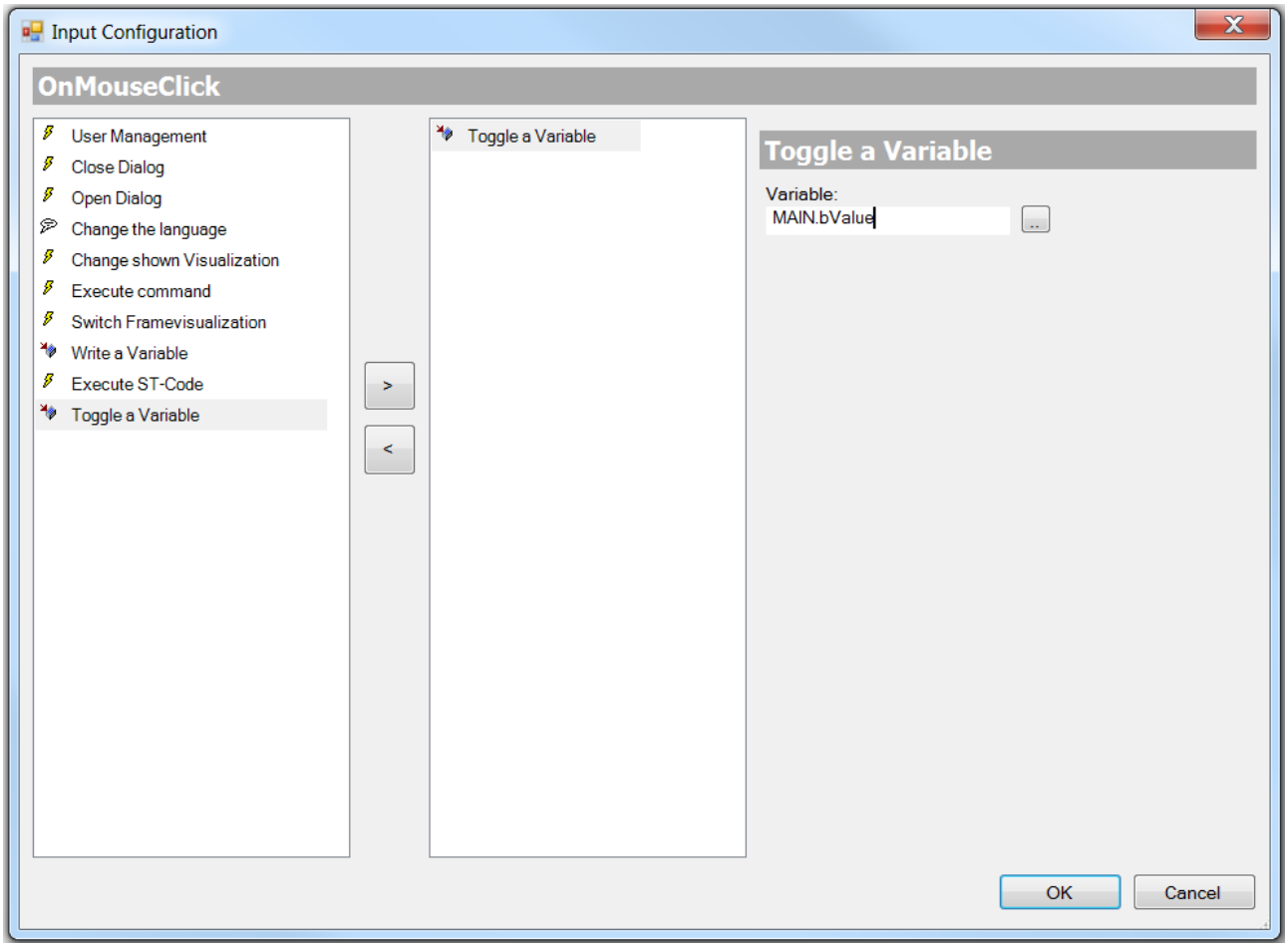
在该后续动作的输入字段中，可以在结构化文本中输入代码，该代码将在鼠标动作后执行。



只有启用 PLC HMI [▶ 551] 和/或 PLC HMI Web [▶ 555]，才能实现复杂的 ST 程序。否则，只能进行简单的分配。

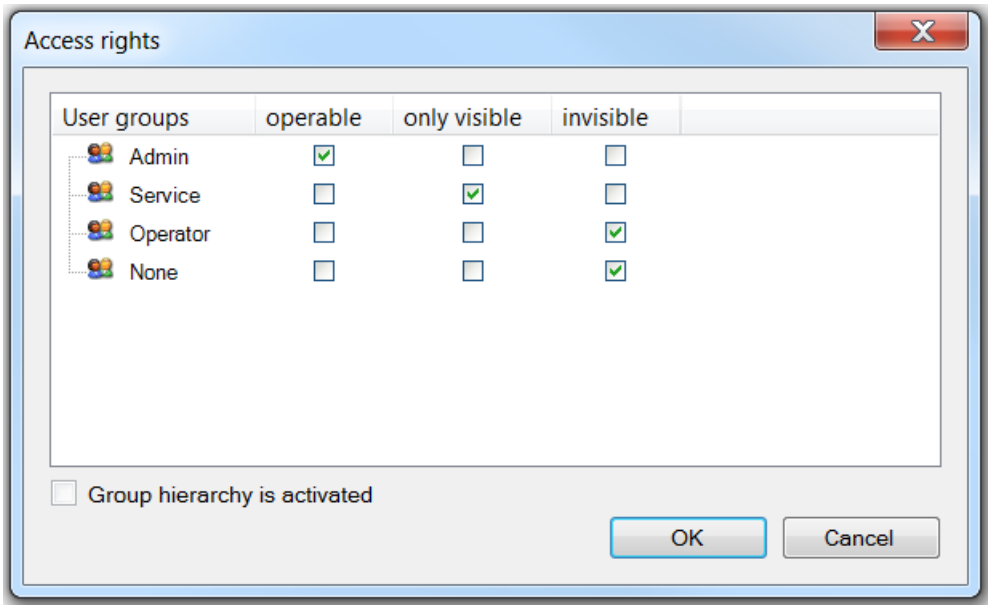
切换变量

您可以在此处输入 1 个 Boolean 变量，该变量将在重复鼠标动作时在 TRUE 和 FALSE 之间切换。



15.8.1.3 访问权限对话框

在元素的属性 [▶ 353]下的访问权限的值字段中点击鼠标，可打开以下对话框：



用户组	这里列出了在用户管理 → 组下的可视化管理器中配置的组。勾选要分配访问权限的组所在行的复选框。
可操作	如果勾选该复选框，则该元素可提供全部功能。
仅可见	该元素不提供任何功能，但它是可见的
不可见	各用户组不显示该元素。
组层次结构已激活	如果在用户管理 → 设置下的可视化管理器中将组层次结构设置为“Use”（使用），则这会在此处显示，并勾选已停用的复选框。

标准组“None”

“None”是默认组，无法删除。如果没有用户登录，则可视化元素将按照“None”的配置发挥作用。

如果某个元素的访问权限有限，建议只将最低权限分配给“None”。

15.8.2 通用控制

15.8.2.1 标签

标签元素可用于为可视化添加标签、标题和任何其他类型的文本。

属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用  按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

文本

这些属性可用于元素标签的静态定义。

文本	输入文本。它可用于标注元素。 也可以直接输入文本。如果在可视化编辑器中选择元素，则可以通过按空格键打开输入字段。

文本属性

这些属性可用于字体的静态定义。在“Font variables”（字体变量）类别中可以对字体进行动态定义。

水平对齐方式	通过选择以下各项来定义文本的水平对齐方式： • 居左 • 居中 • 居右
垂直对齐方式	通过选择以下各项来定义文本的垂直对齐方式： • 顶部 • 居中 • 底部
文本格式	定义因文本太长而无法在元素中完整显示时的显示方式： • 默认 - 文本延伸到元素之外。 • 换行 - 文本自动换行。 • 省略 - 尽可能地显示文本，然后用“...”截断.
字体：	通过从预定义字体中进行选择来定义字体： • 标准 • 页首标题 • 大标题 • 标题 • 说明 按 … 打开用户定义的字体属性的对话框。
字体颜色	定义元素的字体颜色。可以从选择列表中进行选择，也可以通过点击 … 时打开的对话框进行选择。

状态变量

这些是对元素在在线模式下的可用性的动态定义。

不可见	Boolean 变量。如果返回 TRUE，则元素在在线模式下不可见。

访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

15.8.2.2 整数组合框

整数组合框可用于从下拉菜单中选择 1 个值。所选值的索引会被写入变量。条目可以从文本列表中获取，也可以作为图像池中的图像列表获取。

属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在 View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用  按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

文本

工具提示	您可以在此处设置要用作元素的工具提示的文本。它只会在运行时出现在可视化中。

文本属性

使用	以下 2 个设置可供选择： • 默认样式值 • 用户定义的设置 ◦ 水平对齐方式 ◦ 垂直对齐方式 ◦ 字体 ◦ 字体颜色

用户定义的文本属性

水平对齐方式	通过选择以下各项来定义文本的水平对齐方式： • 居左 • 居中 • 居右
垂直对齐方式	通过选择以下各项来定义文本的垂直对齐方式： • 顶部 • 居中 • 底部
字体：	通过从预定义字体中进行选择来定义字体： • 标准 • 页首标题 • 大标题 • 标题 • 说明 按 … 打开用户定义的字体属性的对话框。
字体颜色	定义元素的字体颜色。可以从选择列表中进行选择，也可以通过点击 … 时打开的对话框进行选择。

子部分

使用子范围	如果启用该选项，则只使用已分配列表条目的子部分。
索引开始	整数类型的变量，用于为首先使用的条目建立索引
索引结束	整数类型的变量，用于为最后使用的条目建立索引
筛选缺少文本的条目	如果启用该选项，则只显示文本列表中实际包含文本的条目。

访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

附加设置

变量	整数类型的变量，其值为所选列表元素的索引。
文本列表	文本列表的名称（字符串），该列表包含下拉菜单的所有条目并为它们建立索引。
图像池	图像池的名称（字符串），该图像池包含下拉菜单的所有图像并为它们建立索引。
最大值	下拉菜单中的最大条目数

状态变量

这些是对元素在在线模式下的可用性的动态定义。

不可见	指定 1 个 Boolean 变量。如果返回 TRUE，则元素在在线模式下不可见。
输入已禁用	指定 1 个 Boolean 变量。如果返回 TRUE，则元素的输入无效。此外，元素本身在可视化中显示为灰色，表示用户无法输入。 如果可视化使用用户管理 [▶ 360]，则访问权限为“only visible”（仅可见）的用户组的元素会显示为灰色。

15.8.2.3 组合框数组

组合框数组可用于从下拉菜单中选择 1 行值。所选行的索引会被写入变量。

属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在 View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用  按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

列

元素组合框数组在表格视图中将数组或结构变量可视化。数组元素或结构组件的索引在 1 列或 1 行中显示。二维数组或结构数组在多列中显示。要显示的变量在属性“Data array”（数据数组）中进行指定。如果在那里分配 1 个变量，则可以指定表格列的显示方式，即在表格列中显示相应的数组元素。每一个被分配索引 [<n>] 的列都可以单独配置。

列 • [<n>]	在“Data array”（数据数组）下定义的变量结构可用于确定列数并分配索引 <n>。 示例： `aStringTable : ARRAY [0..2, 0..4] OF STRING := [,BMW‘, ,Audi‘, ,Mercedes‘, ,VW‘, ,Fiat‘, ,150’, ,150‘, ,150‘, ,150‘, ,100‘, ,blau‘, ,grau‘, ,silber‘, ,blau‘, ,rot‘` → 形成 3 列 [0]、[1] 和 [2]。
最大数组索引	数值变量，用于指定最大数组索引（可选）
行高	行高（单位：像素）
可见行数	该参数可用于指定在下拉菜单中显示多少行。如果该行数小于数组中的行数，则会显示滚动条。
滚动条的大小	垂直滚动条的宽度（单位：像素）

列 [<n>]

宽度	列宽（单位：像素）
图像列	如果启用该选项，则此列可用于显示全局图像池或用户定义的图像池中的图像。表格单元格中的值会指定图像池中的图像的 ID。
图像配置 • 填充模式 • 透明度 • 透明度颜色	在这里可以实现以下配置： • 填充模式 填充单元格：无论宽高比如何，图像都会根据单元格大小进行调整。 居中：图像居中，同时在单元格内保持比例。即使单独调整高度或宽度，也能保持宽高比。 • 透明度 如果启用该选项，则在透明度颜色下设置的颜色会变成透明。 • 透明度颜色 您可以在此处从选择列表中或通过颜色选择对话框来选择颜色，通过  按钮可以打开颜色选择对话框。如果启用相应的透明度选项，则颜色将显示为透明。
列中的文本对齐方式	您可以在此处修改列的对齐方式： • 居左 • 居中 • 居右

文本

工具提示	您可以在此处设置要用作元素的工具提示的文本。它只会在运行时出现在可视化中。

文本属性

使用	以下 2 个设置可供选择： • 默认样式值 • 用户定义的设置 ◦ 水平对齐方式 ◦ 垂直对齐方式 ◦ 字体 ◦ 字体颜色

用户定义的文本属性

水平对齐方式	通过选择以下各项来定义文本的水平对齐方式： • 居左 • 居中 • 居右
垂直对齐方式	通过选择以下各项来定义文本的垂直对齐方式： • 顶部 • 居中 • 底部
字体：	通过从预定义字体中进行选择来定义字体： • 标准 • 页首标题 • 大标题 • 标题 • 说明 按 … 打开用户定义的字体属性的对话框。
字体颜色	定义元素的字体颜色。可以从选择列表中进行选择，也可以通过点击 … 时打开的对话框进行选择。

访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

附加设置

变量	整数类型的变量，其值为所选行的索引。
数据数组	要显示的数组类型的变量。变量的结构可决定表格中的列数和行数。

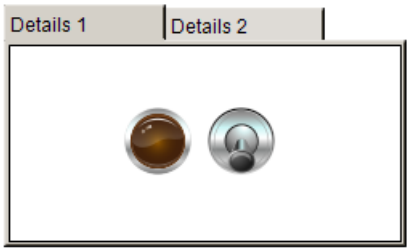
状态变量

这些是对元素在在线模式下的可用性的动态定义。

不可见	指定 1 个 Boolean 变量。如果返回 TRUE，则元素在在线模式下不可见。
输入已禁用	指定 1 个 Boolean 变量。如果返回 TRUE，则元素的输入无效。此外，元素本身在可视化中显示为灰色，表示用户无法输入。 如果可视化使用用户管理 [▶ 360]，则访问权限为“only visible”（仅可见）的用户组的元素会显示为灰色。

15.8.2.4 选项卡控制

选项卡控制可以在 1 个窗口中显示多个可视化页面。选项卡可用于在可视化页面之间切换。“切换变量 [▶ 393]”可用于从程序代码中指定要显示的页面。



如果由于元素宽度较小而无法显示所有选项卡，则会在元素右上角显示 2 个导航箭头。

属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在 View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用  按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

引用可视化

您可以在此处打开对话框“边框选择 [▶ 478]”，该对话框可用于选择要引用的可视化页面。在选择 1 个或多个可视化页面之后，它们会与占位符 [▶ 559]（如果有的话）一起列在下面。如果占位符更改，则会为所有实例自动打开对话框“更新边框参数 [▶ 478]”。

切换变量

该属性可用于在边框的可视化之间切换。

变量	整数变量，其值包含要显示的可视化的 ID。可视化的 ID 由边框选择 [▶ 563]中分配的可视化列表中的顺序决定。例如，该列表中第 1 个条目的 ID 是 0，第 2 个条目的 ID 是 1。 使用变量可以切换分配给边框的可视化。可视化的值（ID）由该元素在对话框“边框可视化的配置”中分配的可视化列表中的顺序决定。对于整数值，该列表中第 1 个条目的值是 0，第 2 个条目的值是 1，以此类推。

访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

附加设置

制表符宽度	制表符宽度（单位：像素）
缩放类型	该参数可用于指定窗口如何响应大小变化： • 各向同性 引用元素保持其比例。这意味着，即使独立改变高度和宽度，可视化的高度和宽度比例也不会改变。 • 各向异性 选项卡控制元素遵循大小变化，因此可以独立更改引用可视化的高度和宽度。 • 未缩放 无论选项卡控制的大小如何，都会保持可视化的原始大小。 • 未缩放且可滚动 如果使用该选项，则会显示没有缩放的引用可视化。如果显示区域大于边框的窗口区域，则边框会提供滚动条，以便移动可视化的显示区域。如要使用变量设置滚动条的位置，可使用属性“Variable scroll position horizontal”（水平可变滚动位置）和“Variable scroll position vertical”（垂直可变滚动位置）。
停用背景图	如要优化可视化的性能，则可将边框元素中的非动画元素作为背景位图绘制。这可能会导致元素显示的顺序与预期不符。为了避免这种行为，可以禁用该功能。

滚动条设置

只有输入缩放类型“Unscaled and scrollable”（未缩放且可滚动）时，才会显示滚动条设置。强烈建议根据客户端的具体情况使用变量。在这种情况下，如果变量更改或者如果用鼠标移动滚动条，则这种变化只会影响相应客户端的边框。否则，所有客户端都会更新。

水平可变滚动位置	变量，包含水平滚动条的位置。
垂直可变滚动位置	变量，包含垂直滚动条的位置。

状态变量

这些是对元素在在线模式下的可用性的动态定义。

不可见	指定 1 个 Boolean 变量。如果返回 TRUE，则元素在在线模式下不可见。
输入已禁用	指定 1 个 Boolean 变量。如果返回 TRUE，则元素的输入无效。此外，元素本身在可视化中显示为灰色，表示用户无法输入。 如果可视化使用用户管理 [▶ 360]，则访问权限为“only visible”（仅可见）的用户组的元素会显示为灰色。

15.8.2.5 按钮

通过按钮可以触发已分配给控制元素的功能。通过分配图像 [▶ 395]或配置浮雕视图 [▶ 395]可定制显示内容。

属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在 View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用  按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

中心

在编辑数值时，相应的元素 ⊕ 也会同时在可视化编辑器 [▶ 344]中移动。

X	元素枢轴的水平位置（单位：像素）
Y	元素枢轴的垂直位置（单位：像素）

颜色

颜色根据由红/绿/蓝（RGB）组件组成的十六进制数进行定义。这 3 种颜色各有 256（0-255）个值，您可以在此处静态输入。从选择列表中或通过颜色选择对话框可以选择颜色，通过按钮 ... 可以打开颜色选择对话框。此外，还可以为每种颜色设置透明度级别（0：完全透明，255：完全不透明）。

颜色	为默认状态选择边框和填充颜色。如果颜色切换变量被定义为 FALSE，则元素处于默认状态。
报警颜色	为处于报警状态的元素选择边框和填充颜色。如果颜色切换变量被定义为 TRUE，则会触发此操作。

使用颜色渐变	默认情况下，未勾选该复选框。如果勾选它，则会以颜色渐变的方式绘制相应的元素。
颜色渐变选择	渐变编辑器 [▶ 370]打开。

按钮高度	浮雕视图中的按钮的高度（单位：像素）

Windows CE 不支持颜色渐变和透明度。

图像

静态 ID	按钮的图像文件的静态 ID。在图像池中对 ID（字符串）进行定义。图像池的名称应加前缀，以便确保条目明确。对于在 GlobalImagePool 中管理的图像文件，无需指定图像池，因为总是首先搜索该图像池。点击 ... 按钮，打开输入助手，会显示所有可用图像池及其图像的列表。
缩放类型	您可以在此处指定图像文件如何响应按钮大小的变化： • 各向同性：图像会保持其比例；即，即使按钮的高度和宽度分别改变，高宽比也会保持不变。 • 各向异性：图像会保持其比例；即，即使按钮的高度和宽度分别改变，高宽比也会保持不变。 • 未缩放：即使按钮的大小改变，图像也会保持原来的大小。
透明度	如果启用该选项，则位图将以“Transparency color”（透明度颜色）下设置的颜色变成透明。
透明度颜色	您可以在此处设置在图像中完全透明显示的颜色。 要求：透明度已启用
水平对齐方式	您可以在此处定义位图的水平对齐方式： • 居左 • 居中 • 居右
垂直对齐方式	您可以在此处定义位图的垂直对齐方式： • 顶部 • 居中 • 底部

文本

这些属性可用于元素标签的静态定义。每个属性都可以包含 1 个格式化序列 [▶ 563]，例如 %s。在在线模式下，在“Text variables”（文本变量）中定义的变量内容可以替换序列。

文本	输入文本。它可用于标注元素。文本可以包含 1 个格式化序列，例如 %s。在“Text variables”（文本变量）中可以定义相应变量。 注意：也可以直接输入文本。如果在可视化编辑器中选择元素，则可以通过按空格键打开输入字段。
工具提示	输入文本。它可以用作元素的工具提示，只有在在线模式下将光标置于元素上时才会出现在可视化中。文本可以包含 1 个格式化序列，例如 %s。在“Text variables”（文本变量）中可以定义相应变量。

文本属性

字体	通过从预定义字体中进行选择来定义字体： • 标准 • 页首标题 • 大标题 • 标题 • 说明 按 … 打开用户定义的字体属性的对话框。
字体颜色	定义元素的字体颜色。可以从选择列表中进行选择，也可以通过点击 … 时打开的对话框进行选择。

绝对移动

通过 1 个整数变量改变元素左上角的 x 和 y 位置（像素），从而可以移动元素。这里使用的是绝对坐标值。

运动 • X • Y	X：此处输入的整数变量可以定义元素左上角当前的 x 位置（单位：像素）。它可用于沿 x 方向移动元素。（正值会将元素从左至右移动）。 Y：此处输入的整数变量可以定义元素左上角当前的 y 位置（单位：像素）。它可用于沿 y 方向移动元素。（正值会将元素从上至下移动）。
旋转	此处输入的整数变量可以定义围绕旋转点旋转元素的角度（角度数）。 正值 = 顺时针方向 注意：与“internal rotation”（内部旋转）行为（见下文）相比，元素本身不会旋转。 点击元素，可显示旋转点 ⊕。按住鼠标按钮，即可移动它。
缩放	此处输入的整数变量可以定义当前的缩放因子（百分比）。元素大小可根据该值进行线性调整。该值隐式除以 1000，因此无需使用 REAL 变量来缩小元素。缩放总是指旋转点（中心）。点击元素，可显示旋转点 ⊕。按住鼠标按钮，即可移动它。
内部旋转	此处输入的整数变量可以定义元素绕其旋转点旋转的角度（角度数）；正值=数学上的正数=顺时针方向。与“Rotation”（旋转）（见上文）相比，元素本身会旋转。点击元素，可显示旋转点（中心）⊕。按住鼠标按钮，即可移动它。

相对移动

元素可以相对于其固定位置移动。按照整数变量定义的值（像素），元素的左上边缘和右下边缘会沿 x 或 y 方向移动。与绝对移动相比，可以定义相对位置，即与原始位置的距离。此功能可用于更改元素的形状。正值会使水平边缘向下移动和/或使垂直边缘向右移动。

左上移动 • X • Y	• X：整数变量，其值表示左上角沿 x 方向移动的像素数。 • Y：整数变量，其值表示左上角沿 y 方向移动的像素数。
右下移动 • X • Y	• X：整数变量，其值表示右下角沿 x 方向移动的像素数。 • Y：整数变量，其值表示右下角沿 y 方向移动的像素数。

文本变量

您可以显示存储在变量中的文本。为此，首先要在“Texts”（文本）属性下定义的文本中添加 1 个格式化序列。然后，分配 1 个变量。在在线模式下，变量的内容可以替换格式化序列 [▶ 563]。

示例 %f：

在“Texts”（文本）属性中，输入“Result: %2.5f”。在“Text variables”（文本变量）属性中，输入“fValue”。该变量必须是已定义的 IEC 变量。如果 fValue = 12.1234567，则该元素将在在线模式下标注为“Result: 12.12345”。

文本变量	（默认数据类型的）变量，包含要显示的信息。类型必须与“Texts”（文本）设置中的格式化序列一致。
Tooltipvariable	字符串类型的变量，包含要显示的工具提示文本。“Texts”（文本）属性中的条目必须包含格式化序列。

动态文本

这些参数可用于定义源自文本列表 [▶ 127]的动态文本。例如，这可以实现语言更改 [▶ 563]。

动态定义文本的另一种可能性是通过字符串变量提供文本。（请参见“文本变量”类别）。

文本列表	在项目树中，作为字符串使用的文本列表的名称 示例：“TL_ErrorList”
Textindex	在文本列表中，作为字符串定义的文本的索引（ID）。它可以以静态方式直接指定，也可以作为字符串变量进行指定。
Tooltipindex	在文本列表中，作为字符串定义的工具提示文本的索引（ID）。它可以以静态方式直接指定，也可以作为字符串变量进行指定。

字体变量

这些变量可用于通过项目变量对元素文本进行动态字体定义。在“Text properties”（文本属性）下可以配置静态定义。

字体名称	指定 1 个字符串类型的变量，该变量包含用于标注元素的字体名称。（指定的名称，如在标准字体对话框中所示） 示例：MAIN.sFont (sFont := ‘Arial’ ;)
大小	INT 类型的变量，包含元素文本的大小（单位：像素），如在默认对话框“Font”（字体）中所示。 示例：MAIN.nHeight (nHeight := 16;)
标志	DWORD 类型的变量，用于通过下列标志值之一定义字体显示。通过将相关的标志值相加并指定总和可以实现组合定义。 • 1：斜体 • 2：粗体 • 4：下划线 • 8：删除 示例： MAIN.nFlag (nFlag := 6;) 文字显示为粗体并加下划线。
字符集	通过默认字体编号可以定义字体所使用的字符集。通过 DWORD 变量可以指定该数字（另请参见默认字体对话框中的定义）。
颜色	DWORD 类型的变量，用于定义元素文本的颜色。
文本对齐方式的标志	DWORD 类型的变量，用于通过下列标志值之一定义文本对齐方式。通过将相关的标志值相加并指定总和可以实现组合定义。 • 0：左上 • 1：水平居中 • 2：居右 • 4：垂直居中 • 8：居下 **示例** MAIN.nFlag (nFlag := 5;) 以水平和垂直居中的方式显示文本。

颜色变量

用于通过 DWORD 类型的项目变量动态定义元素颜色的颜色变量。颜色根据由红、绿、蓝（RGB）组件组成的十六进制数进行定义。此外，变量还可用于指定颜色的透明度（FF：完全不透明 - 00：完全透明）。DWORD 的结构如下：16#TTRRGGBB

十六进制数的结构与 TwinCAT 2 不同。在 TwinCAT 3 中，除了 RGB 比例之外，还可以用十六进制数来定义颜色的透明度。透明度由“16#”后的前 2 位数字表示。定义以“16#00”开头的颜色不可见，因为它们是完全透明的。

示例：

nFillColor := 16#FF8FE03F;

- FF：透明度（完全不透明）
- 8F：红色
- E0：绿色
- 3F：蓝色

颜色更改	Boolean 变量，用于控制元素颜色在“正常状态”（变量 = FALSE）和“报警状态”（变量 = TRUE）之间的切换。
正常状态	DWORD 类型的变量，用于定义元素颜色。它会覆盖当前在“Colors”（颜色）中定义的值。如果在“Color change”（颜色更改）中定义的变量为 FALSE，则会使用项目变量中的值。
报警状态	DWORD 类型的变量，用于定义报警状态下的元素颜色。它会覆盖当前在“Colors”（颜色）中定义的值。如果在“Color change”（颜色更改）中定义的变量为 TRUE，则会使用项目变量中的值。

Windows CE 不支持透明度。

状态变量

这些是对元素在在线模式下的可用性的动态定义。

不可见	指定 1 个 Boolean 变量。如果返回 TRUE，则元素在在线模式下不可见。
输入已禁用	指定 1 个 Boolean 变量。如果返回 TRUE，则元素的输入无效。此外，元素本身在可视化中显示为灰色，表示用户无法输入。 如果可视化使用用户管理 [▶ 360]，则访问权限为“only visible”（仅可见）的用户组的元素会显示为灰色。

按钮状态变量

Boolean 值	Boolean 变量，如果变量为 TRUE，则它控制的按钮显示为“已按下”；如果变量为 FALSE，则它控制的按钮显示为“未按下”。

图像 ID 变量

图像 ID	字符串类型的项目变量，包含要显示元素的图像 ID。在图像池 [▶ 134]中对 ID 进行定义。图像池的名称应加前缀，以便确保条目明确。对于在“GlobalImagePool”中管理的图像文件，无需指定图像池，因为总是首先搜索该图像池。

输入配置

您可以在此处定义当用户在在线模式下对元素进行输入时应执行的相应动作。只要没有定义后续动作，“Configure…”（配置...）就会出现在 Properties（属性）字段中。点击“Configure…”（配置...），打开输入配置 [▶ 371]，您可以在此处分配后续动作。您可以将许多不同的后续动作分配给每个输入动作。

元素可以使用以下输入事件：

- OnDialogClosed
- OnMouseClick
- OnMouseDown
- OnMouseEnter
- OnMouseLeave
- OnMouseMove
- OnMouseUp

OnDialogClosed	如果在可视化中关闭 1 个为用户输入而打开的对话框，则会触发该事件。 注意：此属性不仅限于为其配置的元素，而是适用于整个可视化。因此，它不会对每个对话框的关闭动作做出响应。目前还没有办法为整个可视化定义这样的属性。因此，必须将它分配给其中 1 个元素。
OnMouseClick	当光标指向 1 个元素并在该项目上执行 1 次完整的鼠标点击（按下并松开鼠标按钮）时，就会触发该鼠标事件。
OnMouseDown	在光标指向 1 个元素的同时按下鼠标按钮时，就会触发该鼠标事件。这与您在可视化上再次松开鼠标按钮的位置无关。
OnMouseEnter	在将光标拖到元素上时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseLeave	在光标离开元素时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseMove	当光标在元素内移动时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseUp	在元素上松开鼠标按钮时，就会发生该鼠标事件。在此之前，已在元素之外按下鼠标按钮。

输入配置 - 按键

“按键”可用于指定根据事件“按键”的鼠标行为设置 Boolean 项目变量的值。

变量	Boolean 变量，如果在光标指向元素的同时按下鼠标按钮，则该变量的值为 TRUE。当松开鼠标按钮或光标离开元素时，该值又会变为 FALSE。
FALSE 按键	如果启用该选项，则相应变量的上述按键行为将会反转。也就是说，当按下鼠标按钮时，变量值会被设置为 FALSE。当松开鼠标按钮时，变量值会被设置为 TRUE。
如果鼠标受限，则启动输入	如果启用该选项，只要光标在元素区域内，变量值就会如“Variables”（变量）中所述发挥作用。如果在按下鼠标按钮的同时光标离开元素区域，则该变量会被设置为 FALSE。不过，如果在光标返回元素区域时没有松开鼠标按钮，则会将它再次自动设置为 TRUE。换句话说，只要按下鼠标按钮，即使光标离开元素区域，系统也会考虑到鼠标“受限”的情况。

输入配置 - 切换

“切换”可用于指定根据鼠标行为针对该事件设置 Boolean 项目变量的值。

变量	Boolean 变量，每当鼠标点击元素时，该变量值会在 TRUE 和 FALSE 之间切换。如果在按下鼠标按钮的同时光标远离元素，则不会发生切换。可用于取消切换动作。
如果鼠标受限，则启动松开	如果启用该选项，只要光标在元素区域内，变量值就会如“Variables”（变量）中所述发挥作用。如果在按下鼠标按钮的同时光标离开元素区域，然后松开鼠标按钮，则仍会切换变量。换句话说，只要按下鼠标按钮，即使光标离开元素区域，系统也会考虑到鼠标“受限”的情况。

输入配置 - 热键

热键可用于定义 1 个按键或按键组合，并将其与后续动作（例如，MouseDown、MouseUp）建立关联，当按键事件（KeyDown、KeyUp）发生时，就会执行后续动作。默认情况下，在 KeyDown（按下按键）时会执行 MouseDown 动作，在 KeyUp（松开按键）时会执行 MouseUp 动作。如果要同时通过鼠标动作和键盘输入来操作可视化，这将非常有用，因为输入动作只需配置 1 次。在可视化的热键配置 [▶ 358]中也可以对元素的按键配置进行管理。任何更改始终都会在此编辑器和元素属性编辑器之间同步进行。

按键	分配按键。选择列表包含当前支持的所有按键，例如 M。
事件	如果使用按键或按键和修饰符，则定义要执行的事件。选择列表中可用的可能值： • 无动作 • 在按下按键时，MouseDown 动作 • 在松开按键时，MouseUp 动作 • 在按下松开按键时，MouseDown/MouseUp 动作
移位	如果启用该选项，则按键必须与 Shift 键组合使用。
控制	如果启用该选项，则按键必须与 Ctrl 键组合使用。
Alt	如果启用该选项，则按键必须与 Alt 键组合使用。

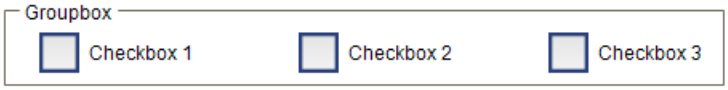
访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

15.8.2.6 Groupbox

根据拖放原则，可将可视化元素拖入分组框，以形成 1 个视觉单元。它们可以多次嵌套。



属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在 View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用  按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

文本

这些属性可用于元素标签的静态定义。

文本	输入文本。它可用于标注元素。 注意：也可以直接输入文本。如果在可视化编辑器中选择元素，则可以通过按空格键打开输入字段。
工具提示	输入文本。它可以用作元素的工具提示，只有在在线模式下将光标置于元素上时才会出现在可视化中。

文本属性

字体	通过从预定义字体中进行选择来定义字体： • 标准 • 页首标题 • 大标题 • 标题 • 说明 按 … 打开用户定义的字体属性的对话框。
字体颜色	定义元素的字体颜色。可以从选择列表中进行选择，也可以通过点击 … 时打开的对话框进行选择。

访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

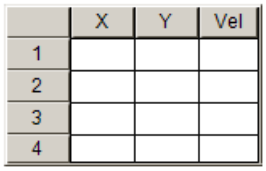
状态变量

这些是对元素在在线模式下的可用性的动态定义。

不可见	指定 1 个 Boolean 变量。如果返回 TRUE，则元素在在线模式下不可见。
输入已禁用	指定 1 个 Boolean 变量。如果返回 TRUE，则元素的输入无效。此外，元素本身在可视化中显示为灰色，表示用户无法输入。 如果可视化使用用户管理 [▶ 360]，则访问权限为“only visible”（仅可见）的用户组的元素会显示为灰色。

15.8.2.7 表格

在可视化中可以添加表格，以便显示 POU 的一维或二维数组、结构或局部变量。有关表格配置示例，请参见“表格配置 [▶ 407]”部分。



属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在 View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用  按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

附加设置

数据数组	该参数可用于指定要可视化的变量，包括完整路径。变量的结构可决定表格中的列数和行数。如果变量的数组元素数不同，则可使用“Max. Array-Index”（最大数组索引）。如要在数组大小或结构变量已更改的情况下更新表格，则可将光标置于数据数组值字段并按 Enter 键。 示例： 以下数组构成 1 个 3 列 4 行的表格。 `aData : ARRAY [1..3,1..4] OF INT;`
最大阵列索引	数值变量，动态指定最大数组索引，以避免超过数组限值

如果数据数组值已更改，即分配了 1 个新的变量，则其他表格属性将被重置为默认值。

列

表格元素在表格视图中将变量可视化。数组/结构组件的索引在 1 列或 1 行中显示。对于二维数组或结构数组，可视化会使用多个列。在这些元素设置中定义了表格列的外观，其中显示了各个数组元素/结构组件/变量的值。与特定索引相关的每一列都可以单独配置。

列
• [<n>]
	在“Data array”（数据数组）下定义的变量结构可用于确定列数并分配索引 <n>。
	示例：
	`aDataArray1 : ARRAY [2..8] OF INT;`
	→ 形成 1 列 [0]。
	`aDataArray2 : ARRAY [1..3, 1..10] OF INT;`
	→ 形成 3 列 [0]、[1] 和 [2]。

显示行标题	如果启用该选项，则行标签将以分配的行索引的形式显示。
显示列标题	如果启用该选项，则列标签将以分配的行索引的形式显示。
行高	行高（单位：像素）
行标题的宽度	包含行标题的列的宽度（单位：像素）
滚动条的大小	垂直滚动条的宽度或水平滚动条的高度（单位：像素）

列 [<n>]

列标题	您可以在此处输入新标题，从而更改列标题。默认情况下，带有与列相关的索引或结构组件的数组或结构的名称可被用作标题。
宽度	列宽（单位：像素）
图像列	如果启用该选项，则此列可用于显示全局图像池或用户定义的图像池中的图像。表格单元格中的值会指定图像池中的图像的 ID。
图像配置 • 填充模式 • 透明度 • 透明度颜色	在这里可以实现以下配置： • 填充模式 填充单元格：无论宽高比如何，图像都会根据单元格大小进行调整。 居中：图像居中，同时在单元格内保持比例。即使单独调整高度或宽度，也能保持宽高比。 • 透明度 如果启用该选项，则在透明度颜色下设置的颜色会变成透明。 • 透明度颜色 您可以在此处从选择列表中或通过颜色选择对话框来选择颜色，通过 [...] 按钮可以打开颜色选择对话框。如果启用相应的透明度选项，则颜色将显示为透明。
标题的文本对齐方式	您可以在此处修改列标题的对齐方式： • 居左 • 居中 • 居右
使用模板	如果启用该选项，则会在该表格列的每个单元格中再添加 1 个（类型为矩形、圆角矩形或椭圆形 [▶ 433]的）可视化元素。属性列表会根据模板中该元素的属性自动扩展。
模板中的标题的文本对齐方式	如果启用该选项，则插入模板中的字体（大小）和对齐方式的设置也会应用于列标题。 只有在同时启用“Use template”（使用模板）时，启用该选项才会生效。
模板	在 Template（模板）下，可依次列出分配给该列的所有元素的属性，并可按照矩形、圆角矩形或椭圆形 [▶ 433]下的说明进行编辑。 只有在启用“Use template”（使用模板）时，才会出现此条目。

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

文本属性

这些属性可用于字体的静态定义。在“Font variables”（字体变量）类别中可以对字体进行动态定义。

水平对齐方式	通过选择以下各项来定义文本的水平对齐方式： • 居左 • 居中 • 居右
垂直对齐方式	通过选择以下各项来定义文本的垂直对齐方式： • 顶部 • 居中 • 底部
文本格式	定义因文本太长而无法在元素中完整显示时的显示方式： • 默认 – 文本延伸到元素之外。 • 换行 – 文本自动换行。 • 省略 – 尽可能地显示文本，然后用“...”截断.
字体：	通过从预定义字体中进行选择来定义字体： • 标准 • 页首标题 • 大标题 • 标题 • 说明 按 [...] 打开用户定义的字体属性的对话框。
字体颜色	定义元素的字体颜色。可以从选择列表中进行选择，也可以通过点击 [...] 时打开的对话框进行选择。

动态文本

这些参数可用于定义源自文本列表 [▶ 127]的动态文本。例如，这可以实现语言更改 [▶ 563]。

动态定义文本的另一种可能性是通过字符串变量提供文本。（请参见“文本变量”类别）。

文本列表	在项目树中，作为字符串使用的文本列表的名称 示例：“TL_ErrorList”
Textindex	在文本列表中，作为字符串定义的文本的索引（ID）。它可以以静态方式直接指定，也可以作为字符串变量进行指定。
Tooltipindex	在文本列表中，作为字符串定义的工具提示文本的索引（ID）。它可以以静态方式直接指定，也可以作为字符串变量进行指定。

字体变量

这些变量可用于通过项目变量对元素文本进行动态字体定义。在“Text properties”（文本属性）下可以配置静态定义。

字体名称	指定 1 个字符串类型的变量，该变量包含用于标注元素的字体名称。（指定的名称，如在标准字体对话框中所示） 示例：MAIN.sFont (sFont := ‘Arial’;)
大小	INT 类型的变量，包含元素文本的大小（单位：像素），如在默认对话框“Font”（字体）中所示。 示例：MAIN.nHeight (nHeight := 16;)
标志	DWORD 类型的变量，用于通过下列标志值之一定义字体显示。通过将相关的标志值相加并指定总和可以实现组合定义。 • 1：斜体 • 2：粗体 • 4：下划线 • 8：删除 示例： MAIN.nFlag (nFlag := 6;) 文字显示为粗体并加下划线。
字符集	通过默认字体编号可以定义字体所使用的字符集。通过 DWORD 变量可以指定该数字（另请参见默认字体对话框中的定义）。
颜色	DWORD 类型的变量，用于定义元素文本的颜色。
文本对齐方式的标志	DWORD 类型的变量，用于通过下列标志值之一定义文本对齐方式。通过将相关的标志值相加并指定总和可以实现组合定义。 • 0：左上 • 1：水平居中 • 2：居右 • 4：垂直居中 • 8：居下 **示例** MAIN.nFlag (nFlag := 5;) 以水平和垂直居中的方式显示文本。

状态变量

这些是对元素在在线模式下的可用性的动态定义。

不可见	指定 1 个 Boolean 变量。如果返回 TRUE，则元素在在线模式下不可见。
输入已禁用	指定 1 个 Boolean 变量。如果返回 TRUE，则元素的输入无效。此外，元素本身在可视化中显示为灰色，表示用户无法输入。 如果可视化使用用户管理 [▶ 360]，则访问权限为“only visible”（仅可见）的用户组的元素会显示为灰色。

选择

选择颜色	所选表格单元格的填充颜色
选择类型	该参数可用于定义在点击表格行时如何进行选择： • 无选择 • 单元格选择：只选择被点击的单元格。 • 行选择：选择包含被点击单元格的行。 • 列选择：选择包含被点击单元格的列。 • 行和列选择：选择包含被点击单元格的行和列。
所选单元格周围的边框	如果选择该选项，则会在所选单元格周围绘制 1 个边框。
列选择的变量	整数类型的变量，用于存储所选单元格列的索引。如果数据数组指向 1 个结构，则从 0 开始为结构组件建立索引。请注意，如果没有列被排除在表格之外，则该索引仅代表数组中的正确位置。
行选择的变量	整数类型的变量，用于存储所选单元格的行索引
列选择有效性的变量	Boolean 变量，如果列选择的变量包含有效值，则该变量为 TRUE
行选择有效性的变量	Boolean 变量，如果列选择的变量包含有效值，则该变量为 TRUE

访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

15.8.2.7.1 配置表格

首先，必须指定变量中要可视化的变量。为此，可在“data array”（数据数组）中输入带有完整路径的变量。变量的结构可决定表格中的行数和列数 对于二维数组，第 1 个索引可决定列数，第 2 个索引可决定行数。对于结构，单个组件代表列的结构：

- 示例 1：

```
aDim1 : ARRAY [2..5] OF INT;
```

	MAIN.aDim1[INDEX]
2	
3	
4	
5	

- 示例 2：

```
aDim2 : ARRAY [0..2, 0..3] OF INT;
```

	MAIN.aDim2[0,INDEX]	MAIN.aDim2[1,INDEX]	MAIN.aDim2[2,INDEX]
0			
1			
2			
3			

- 示例 3：

```
TYPE ST_ProductInformation :
STRUCT
    nOrderInPieces : INT;
    bStocked : BOOL;
    nArcticelNumber : INT;
END_STRUCT
END_TYPE

stProductInformation : ST_ProductInformation;
```

	MAIN.stProductInformation.nOrderInPieces	MAIN.stProductInformation.bStocked	MAIN.stProductInformation.nArticelNumber
0			

- 示例 4：

```
aProductInformation : ARRAY [0..3] OF ST_ProductInformation;
```

	MAIN.aProductInformation[INDEX].nOrderInPieces	MAIN.aProductInformation[INDEX].bStocked	MAIN.aProductInformation[INDEX].nArticelNumber
0			
1			
2			
3			

- 示例 5：

```
fbTimer : TON;
```

	MAIN.fbTimer.IN	MAIN.fbTimer.PT	MAIN.fbTimer.Q	MAIN.fbTimer.ET	MAIN.fbTimer.M	MAIN.fbTimer.StartTime
0						

- 示例 6：

```
aTimers : ARRAY [0..3] OF TON;
```

	MAIN.aTimers[INDEX].IN	MAIN.aTimers[INDEX].PT	MAIN.aTimers[INDEX].Q	MAIN.aTimers[INDEX].ET	MAIN.aTimers[INDEX].M	MAIN.aTimers[INDEX].StartTime
0						
1						
2						
3						

如果更改“data array”（数据数组）字段中的条目，则元素属性的所有其他设置将被重置为默认值！

下面使用示例 4 进行说明。

在“data array”（数据数组）中输入变量后，可以更改每一列的属性。首先，更改列标题，使其更有意义。然后，更改列宽和标题对齐方式。

2 个选项可用于更改列宽：

1. 在元素属性中，在相应列下可以更改属性“Width”（宽度）中的值。
2. 在表格元素中，可以将鼠标直接移动到 2 列之间的直线上。此时会出现 1 个光标图标  ，这表示您现在可以在按下鼠标按钮的同时更改列的大小。

	Order in pieces	Stocked?	Articel number
0			
1			
2			
3			

在属性“Row height”（行高）中，可将行高设置为 25 像素。下图显示了较宽的滚动条。在“Scrollbar size”（滚动条大小）下，可设置大小。当表格缩小到小于行高或列宽的和时，就会将滚动条添加到表格中。

	Order in pieces	Stocked?	Articel n
0			
1			
2			

通过调整表格大小，可再次删除滚动条。通过将行高与行数相乘可以计算得出高度（如果启用列标题，则行数+1）。宽度是各个列的宽度与标签列的宽度（如果启用）相加的总和。

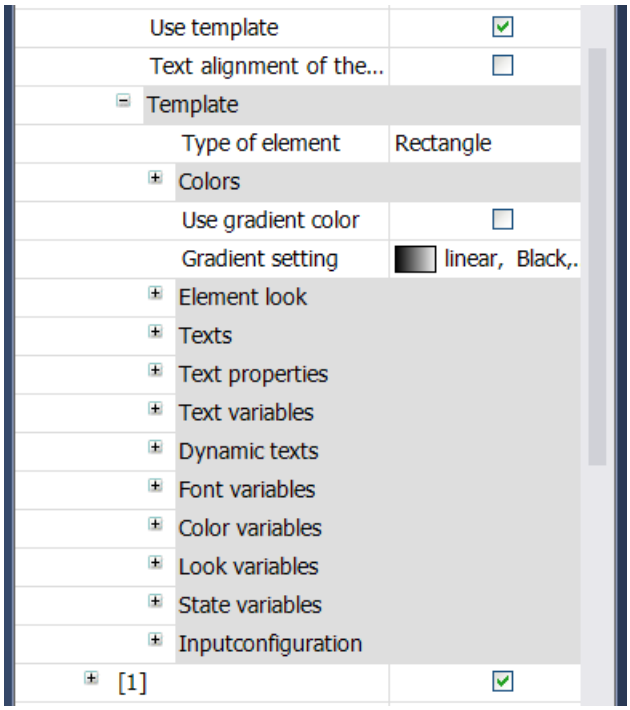
点击“Font”（字体）中的  按钮，可在打开的对话框中选择文本字体和大小。在本示例中，将“Font color”（字体颜色）下的字体颜色更改为“Blue”（蓝色）。

在运行时，表格如下所示：

	Order in pieces	Stocked?	Articel number
0	250	TRUE	32136832
1	480	FALSE	45983920
2	1500	TRUE	46001235
3	680	TRUE	55129547

请注意，在“Text alignment of the heading”（标题的文本对齐方式）下的相关列的设置中可以指定单元格的水平对齐方式。如下所述，按列使用模板的情况除外。

您可以逐列编辑表格字段。为此，可为所选列勾选复选框“Use template”（使用模板），这样就会将“Template”（模板）作为另一个项目添加到列属性中并展开。



3 种不同的模板类型可供选择。默认输入“Rectangle”（矩形）。不过，通过点击属性“Element type”（元素类型）的值字段内部，可以将其更改为“rounded rectangle”（圆角矩形）或“ellipse”（椭圆形）。相应的配置选项会显示所选模板的元素属性。与往常一样，您可以选择正常状态和报警状态的填充和边框颜色，也可以通过变量进行切换。现在，列单元格（标题单元格除外）中条目的对齐方式由模板中的设置决定。

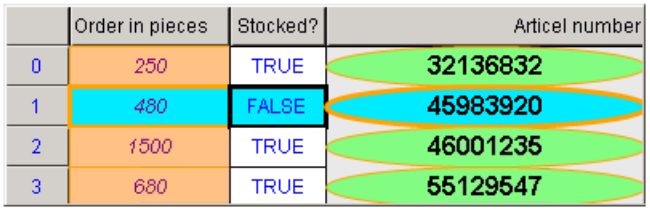
您可以在“Text variable”（文本变量）下输入任何其他项目变量，而不是默认输入的数组组件。特别是，这可以实现在表格中以不同的顺序显示数组元素。

在表格内部还可以交换列。为此，可点击标题中某一列的中心位置，然后按下鼠标按钮，将该列拖到新位置。

在本示例中，第 1 列和第 3 列使用了模板。

	Order in pieces	Stocked?	Articel number
0	250	TRUE	32136832
1	480	FALSE	45983920
2	1500	TRUE	46001235
3	680	TRUE	55129547

如要显示单元格的选择，可在“Selection”（选择）下调整设置。最初，颜色被设置为“light blue”（浅蓝色），选择类型被设置为“row selection”（行选择）。如果在运行时选择某个单元格，则当前行中的所有其他单元格也会突出显示。



	Order in pieces	Stocked?	Articel number
0	250	TRUE	32136832
1	480	FALSE	45983920
2	1500	TRUE	46001235
3	680	TRUE	55129547

15.8.2.8 文本字段

该元素允许显示文本，在 Properties [▶ 412]（属性）中可以直接分配文本，或者在“Text variables [▶ 413]”（文本变量）中可以变化地分配文本。与矩形相比，边框可以显示阴影。

属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在 View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用  按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

颜色

颜色根据由红/绿/蓝（RGB）组件组成的十六进制数进行定义。这 3 种颜色各有 256（0-255）个值，您可以在此处静态输入。从选择列表中或通过颜色选择对话框可以选择颜色，通过按钮  可以打开颜色选择对话框。此外，还可以为每种颜色设置透明度级别（0：完全透明，255：完全不透明）。

正常状态 • 边框颜色 • 填充颜色	为默认状态选择边框和填充颜色。如果颜色切换变量被定义为 FALSE，则元素处于默认状态。
报警状态 • 边框颜色 • 填充颜色	为处于报警状态的元素选择边框和填充颜色。如果颜色切换变量被定义为 TRUE，则会触发此操作。

Windows CE 不支持透明度。

颜色变量

用于通过 DWORD 类型的项目变量动态定义元素颜色的颜色变量。颜色根据由红、绿、蓝（RGB）组件组成的十六进制数进行定义。此外，变量还可用于指定颜色的透明度（FF：完全不透明 - 00：完全透明）。DWORD 的结构如下：16#TTRRGGBB

与 TwinCAT 2 相比，十六进制数的结构有所不同。在 TwinCAT 3 中，除了 RGB 组件之外，还可以用十六进制数来定义颜色透明度。透明度由“16#”后的前 2 位数字表示。定义以“16#00”开头的颜色不可见，因为它们是完全透明的。

示例：

nFillColor := 16#FF8FE03F;

- FF：透明度（完全不透明）
- 8F：红色
- E0：绿色
- 3F：蓝色

颜色更改	Boolean 变量，用于控制元素颜色在“正常状态”（变量 = FALSE）和“报警状态”（变量 = TRUE）之间的切换。
正常状态 • 边框颜色 • 填充颜色	DWORD 类型的变量，用于定义元素的边框和填充颜色。它们会覆盖当前在“Colors”（颜色）中定义的值。如果在“Color change”（颜色更改）中定义的变量为 FALSE，则会使用项目变量中的值。
报警状态 • 边框颜色 • 填充颜色	DWORD 类型的变量，用于定义报警状态下的元素的边框和填充颜色。它们会覆盖当前在“Colors”（颜色）中定义的值。如果在“Color change”（颜色更改）中定义的变量为 TRUE，则会使用项目变量中的值。

Windows CE 不支持透明度。

外观

Appearance（外观）下的设置是对边框和元素填充的静态定义。

线宽	以像素为单位定义边框线宽。0 的代码与 1 相同，并将线宽设置为 1 像素。如果不需要边框，则将线条类型设置为不可见。
填充类型	定义填充颜色的填充类型： • 填充：填充颜色可见 • 不可见：填充颜色不可见
线条类型	为轮廓定义以下 1 种线条类型： • 实线 • 虚线 • 点线 • 点划线 • 双点划线 • 不可见：轮廓不可见。

辅助设置

阴影样式	您可以在此处指定轮廓的阴影样式： • 凹下 • 无阴影 • 凸起 • 来自样式 [▶ 355]：标准设置

文本

这些属性可用于元素标签的静态定义。每个属性都可以包含 1 个格式化序列 [▶ 563]，例如 %s。在在线模式下，在“Text variables”（文本变量）中定义的变量内容可以替换序列。

文本	输入文本。它可用于标注元素。文本可以包含 1 个格式化序列，例如 %s。在“Text variables”（文本变量）中可以定义相应变量。 注意：也可以直接输入文本。如果在可视化编辑器中选择元素，则可以通过按空格键打开输入字段。
工具提示	输入文本。它可以用作元素的工具提示，只有在在线模式下将光标置于元素上时才会出现在可视化中。文本可以包含 1 个格式化序列，例如 %s。在“Text variables”（文本变量）中可以定义相应变量。

文本属性

这些属性可用于字体的静态定义。在“Font variables”（字体变量）类别中可以对字体进行动态定义。

水平对齐方式	通过选择以下各项来定义文本的水平对齐方式： • 居左 • 居中 • 居右
垂直对齐方式	通过选择以下各项来定义文本的垂直对齐方式： • 顶部 • 居中 • 底部
文本格式	定义因文本太长而无法在元素中完整显示时的显示方式： • 默认 – 文本延伸到元素之外。 • 换行 – 文本自动换行。 • 省略 – 尽可能地显示文本，然后用“...”截断.
字体：	通过从预定义字体中进行选择来定义字体： • 标准 • 页首标题 • 大标题 • 标题 • 说明 按  打开用户定义的字体属性的对话框。
字体颜色	定义元素的字体颜色。可以从选择列表中进行选择，也可以通过点击  时打开的对话框进行选择。

文本变量

您可以显示存储在变量中的文本。为此，首先要在“Texts”（文本）属性下定义的文本中添加 1 个格式化序列。然后，分配 1 个变量。在在线模式下，变量的内容可以替换格式化序列 [▶ 563]。

示例 %f：

在“Texts”（文本）属性中，输入“Result: %2.5f”。在“Text variables”（文本变量）属性中，输入“fValue”。该变量必须是已定义的 IEC 变量。如果 fValue = 12.1234567，则该元素将在在线模式下标注为“Result: 12.12345”。

文本变量	（默认数据类型的）变量，包含要显示的信息。类型必须与“Texts”（文本）设置中的格式化序列一致。
Tooltipvariable	字符串类型的变量，包含要显示的工具提示文本。“Texts”（文本）属性中的条目必须包含格式化序列。

状态变量

这些是对元素在在线模式下的可用性的动态定义。

不可见	指定 1 个 Boolean 变量。如果返回 TRUE，则元素在在线模式下不可见。
输入已禁用	指定 1 个 Boolean 变量。如果返回 TRUE，则元素的输入无效。此外，元素本身在可视化中显示为灰色，表示用户无法输入。 如果可视化使用用户管理 [▶ 360]，则访问权限为“only visible”（仅可见）的用户组的元素会显示为灰色。

输入配置

您可以在此处定义当用户在在线模式下对元素进行输入时应执行的相应动作。只要没有定义后续动作，“Configure…”（配置...）就会出现在 Properties（属性）字段中。点击“Configure…”（配置...），打开输入配置 [▶ 371]，您可以在此处分配后续动作。您可以将许多不同的后续动作分配给每个输入动作。

元素可以使用以下输入事件：

- OnDialogClosed
- OnMouseClick
- OnMouseDown
- OnMouseEnter
- OnMouseLeave
- OnMouseMove
- OnMouseUp

OnDialogClosed	如果在可视化中关闭 1 个为用户输入而打开的对话框，则会触发该事件。 注意：此属性不仅限于为其配置的元素，而是适用于整个可视化。因此，它不会对每个对话框的关闭动作做出响应。目前还没有办法为整个可视化定义这样的属性。因此，必须将它分配给其中 1 个元素。
OnMouseClick	当光标指向 1 个元素并在该项目上执行 1 次完整的鼠标点击（按下并松开鼠标按钮）时，就会触发该鼠标事件。
OnMouseDown	在光标指向 1 个元素的同时按下鼠标按钮时，就会触发该鼠标事件。这与您在可视化上再次松开鼠标按钮的位置无关。
OnMouseEnter	在将光标拖到元素上时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseLeave	在光标离开元素时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseMove	当光标在元素内移动时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseUp	在元素上松开鼠标按钮时，就会发生该鼠标事件。在此之前，已在元素之外按下鼠标按钮。

输入配置 - 热键

热键可用于定义 1 个按键或按键组合，并将其与后续动作（例如，MouseDown、MouseUp）建立关联，当按键事件（KeyDown、KeyUp）发生时，就会执行后续动作。默认情况下，在 KeyDown（按下按键）时会执行 MouseDown 动作，在 KeyUp（松开按键）时会执行 MouseUp 动作。如果要同时通过鼠标动作和键盘输入来操作可视化，这将非常有用，因为输入动作只需配置 1 次。在可视化的热键配置 [▶ 358]中也可以对元素的按键配置进行管理。任何更改始终都会在此编辑器和元素属性编辑器之间同步进行。

按键	分配按键。选择列表包含当前支持的所有按键，例如 M。
事件	如果使用按键或按键和修饰符，则定义要执行的事件。选择列表中可用的可能值： • 无动作 • 在按下按键时，MouseDown 动作 • 在松开按键时，MouseUp 动作 • 在按下松开按键时，MouseDown/MouseUp 动作
移位	如果启用该选项，则按键必须与 Shift 键组合使用。
控制	如果启用该选项，则按键必须与 Ctrl 键组合使用。
Alt	如果启用该选项，则按键必须与 Alt 键组合使用。

输入配置 - 按键

“按键”可用于指定根据事件“按键”的鼠标行为设置 Boolean 项目变量的值。

变量	Boolean 变量，如果在光标指向元素的同时按下鼠标按钮，则该变量的值为 TRUE。当松开鼠标按钮或光标离开元素时，该值又会变为 FALSE。
FALSE 按键	如果启用该选项，则相应变量的上述按键行为将会反转。也就是说，当按下鼠标按钮时，变量值会被设置为 FALSE。当松开鼠标按钮时，变量值会被设置为 TRUE。
如果鼠标受限，则启动输入	如果启用该选项，只要光标在元素区域内，变量值就会如“Variables”（变量）中所述发挥作用。如果在按下鼠标按钮的同时光标离开元素区域，则该变量会被设置为 FALSE。不过，如果在光标返回元素区域时没有松开鼠标按钮，则会将它再次自动设置为 TRUE。换句话说，只要按下鼠标按钮，即使光标离开元素区域，系统也会考虑到鼠标“受限”的情况。

输入配置 - 切换

“切换”可用于指定根据鼠标行为针对该事件设置 Boolean 项目变量的值。

变量	Boolean 变量，每当鼠标点击元素时，该变量值会在 TRUE 和 FALSE 之间切换。如果在按下鼠标按钮的同时光标远离元素，则不会发生切换。可用于取消切换动作。
如果鼠标受限，则启动松开	如果启用该选项，只要光标在元素区域内，变量值就会如“Variables”（变量）中所述发挥作用。如果在按下鼠标按钮的同时光标离开元素区域，然后松开鼠标按钮，则仍会切换变量。换句话说，只要按下鼠标按钮，即使光标离开元素区域，系统也会考虑到鼠标“受限”的情况。

访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

15.8.2.9 滚动条

该元素会生成 1 个滚动条。如果用户在可视化运行时将滚动框拖到不同位置，则可相应设置属性“Value [▶ 416]”（值）。滚动条的对齐方式可从水平改为 [▶ 416]垂直。

属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在 View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用  按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

附加设置

值	整数类型的变量，包含滚动条中滚动框的位置。
最小值	最小可能的滚动条值
最大值	最大可能的滚动条值
页面大小	如果用户在运行时点击滚动条，则滚动框的位置会沿着鼠标位置的方向按页面大小移动。 您可以将页面大小指定为以像素为单位的固定值或者整数类型的变量。

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

条

对齐方式	您可以在此处指定条的对齐方式： • 水平 • 垂直 通过更改滚动条的宽度和高度，也可以直接在可视化中配置对齐方式。
行进方向	如果对齐方式为水平，则您可以按如下方式选择滚动方向： • 从左至右 • 从右至左 如果对齐方式为垂直，则您可以按如下方式选择滚动方向： • 从下至上 • 从上至下

颜色

颜色根据由红/绿/蓝（RGB）组件组成的十六进制数进行定义。这 3 种颜色各有 256（0-255）个值，您可以在此处静态输入。从选择列表中或通过颜色选择对话框可以选择颜色，通过按钮  可以打开颜色选择对话框。此外，还可以为每种颜色设置透明度级别（0：完全透明，255：完全不透明）。

正常状态 • 边框颜色 • 填充颜色	为默认状态选择边框和填充颜色。如果颜色切换变量被定义为 FALSE，则元素处于默认状态。
报警状态 • 边框颜色 • 填充颜色	为处于报警状态的元素选择边框和填充颜色。如果颜色切换变量被定义为 TRUE，则会触发此操作。

Windows CE 不支持透明度。

颜色变量

用于通过 DWORD 类型的项目变量动态定义元素颜色的颜色变量。颜色根据由红、绿、蓝（RGB）组件组成的十六进制数进行定义。此外，变量还可用于指定颜色的透明度（FF：完全不透明 - 00：完全透明）。DWORD 的结构如下：16#TTRRGGBB

与 TwinCAT 2 相比，十六进制数的结构有所不同。在 TwinCAT 3 中，除了 RGB 组件之外，还可以用十六进制数来定义颜色透明度。透明度由“16#”后的前 2 位数字表示。定义以“16#00”开头的颜色不可见，因为它们是完全透明的。

示例：

```
nFillColor := 16#FF8FE03F;
```

- FF：透明度（完全不透明）
- 8F：红色
- E0：绿色
- 3F：蓝色

颜色更改	Boolean 变量，用于控制元素颜色在“正常状态”（变量 = FALSE）和“报警状态”（变量 = TRUE）之间的切换。
正常状态 • 边框颜色 • 填充颜色	DWORD 类型的变量，用于定义元素的边框和填充颜色。它们会覆盖当前在“Colors”（颜色）中定义的值。如果在“Color change”（颜色更改）中定义的变量为 FALSE，则会使用项目变量中的值。
报警状态 • 边框颜色 • 填充颜色	DWORD 类型的变量，用于定义报警状态下的元素的边框和填充颜色。它们会覆盖当前在“Colors”（颜色）中定义的值。如果在“Color change”（颜色更改）中定义的变量为 TRUE，则会使用项目变量中的值。

Windows CE 不支持透明度。

文本

这些属性可用于元素标签的静态定义。每个属性都可以包含 1 个格式化序列 [▶ 563]，例如 %s。在在线模式下，在“Text variables”（文本变量）中定义的变量内容可以替换序列。

文本	输入文本。它可用于标注元素。文本可以包含 1 个格式化序列，例如 %s。在“Text variables”（文本变量）中可以定义相应变量。 注意：也可以直接输入文本。如果在可视化编辑器中选择元素，则可以通过按空格键打开输入字段。
工具提示	输入文本。它可以用作元素的工具提示，只有在在线模式下将光标置于元素上时才会出现在可视化中。文本可以包含 1 个格式化序列，例如 %s。在“Text variables”（文本变量）中可以定义相应变量。

文本属性

这些属性可用于字体的静态定义。在“Font variables”（字体变量）类别中可以对字体进行动态定义。

水平对齐方式	通过选择以下各项来定义文本的水平对齐方式： • 居左 • 居中 • 居右
垂直对齐方式	通过选择以下各项来定义文本的垂直对齐方式： • 顶部 • 居中 • 底部
文本格式	定义因文本太长而无法在元素中完整显示时的显示方式： • 默认 – 文本延伸到元素之外。 • 换行 – 文本自动换行。 • 省略 – 尽可能地显示文本，然后用“...”截断.
字体：	通过从预定义字体中进行选择来定义字体： • 标准 • 页首标题 • 大标题 • 标题 • 说明 按 [...] 打开用户定义的字体属性的对话框。
字体颜色	定义元素的字体颜色。可以从选择列表中进行选择，也可以通过点击 [...] 时打开的对话框进行选择。

文本变量

您可以显示存储在变量中的文本。为此，首先要在“Texts”（文本）属性下定义的文本中添加 1 个格式化序列。然后，分配 1 个变量。在在线模式下，变量的内容可以替换格式化序列 [▶ 563]。

示例 %f：

在“Texts”（文本）属性中，输入“Result: %2.5f”。在“Text variables”（文本变量）属性中，输入“fValue”。该变量必须是已定义的 IEC 变量。如果 fValue = 12.1234567，则该元素将在在线模式下标注为“Result: 12.12345”。

文本变量	（默认数据类型的）变量，包含要显示的信息。类型必须与“Texts”（文本）设置中的格式化序列一致。
Tooltipvariable	字符串类型的变量，包含要显示的工具提示文本。“Texts”（文本）属性中的条目必须包含格式化序列。

动态文本

这些参数可用于定义源自文本列表 [▶ 127]的动态文本。例如，这可以实现语言更改 [▶ 563]。

动态定义文本的另一种可能性是通过字符串变量提供文本。（请参见“文本变量”类别）。

文本列表	在项目树中，作为字符串使用的文本列表的名称 示例：“TL_ErrorList”
Textindex	在文本列表中，作为字符串定义的文本的索引（ID）。它可以以静态方式直接指定，也可以作为字符串变量进行指定。
Tooltipindex	在文本列表中，作为字符串定义的工具提示文本的索引（ID）。它可以以静态方式直接指定，也可以作为字符串变量进行指定。

字体变量

这些变量可用于通过项目变量对元素文本进行动态字体定义。在“Text properties”（文本属性）下可以配置静态定义。

字体名称	指定 1 个字符串类型的变量，该变量包含用于标注元素的字体名称。（指定的名称，如在标准字体对话框中所示） 示例：MAIN.sFont (sFont := ‘Arial’;)
大小	INT 类型的变量，包含元素文本的大小（单位：像素），如在默认对话框“Font”（字体）中所示。 示例：MAIN.nHeight (nHeight := 16;)
标志	DWORD 类型的变量，用于通过下列标志值之一定义字体显示。通过将相关的标志值相加并指定总和可以实现组合定义。 • 1：斜体 • 2：粗体 • 4：下划线 • 8：删除 示例： MAIN.nFlag (nFlag := 6;) 文字显示为粗体并加下划线。
字符集	通过默认字体编号可以定义字体所使用的字符集。通过 DWORD 变量可以指定该数字（另请参见默认字体对话框中的定义）。
颜色	DWORD 类型的变量，用于定义元素文本的颜色。
文本对齐方式的标志	DWORD 类型的变量，用于通过下列标志值之一定义文本对齐方式。通过将相关的标志值相加并指定总和可以实现组合定义。 • 0：左上 • 1：水平居中 • 2：居右 • 4：垂直居中 • 8：居下 **示例** MAIN.nFlag (nFlag := 5;) 以水平和垂直居中的方式显示文本。

访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

状态变量

这些是对元素在在线模式下的可用性的动态定义。

不可见	指定 1 个 Boolean 变量。如果返回 TRUE，则元素在在线模式下不可见。
输入已禁用	指定 1 个 Boolean 变量。如果返回 TRUE，则元素的输入无效。此外，元素本身在可视化中显示为灰色，表示用户无法输入。 如果可视化使用用户管理 [▶ 360]，则访问权限为“only visible”（仅可见）的用户组的元素会显示为灰色。

15.8.2.10 滑块

使用鼠标可以移动滑动条，因此，分配给滑块的变量会在定义的限制范围内改变其值。

属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在 View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用  按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

辅助设置

变量	数值变量，包含滑块的位置。

标度

显示标度	如果启用该选项，则会显示标度。
标度开始	标度下限值
标度结束	标度上限值
主标度	粗标度的 2 条线之间的距离
子标度	细标度的 2 条线之间的距离。如果不需要对粗标度进行进一步细分，则可将该值设置为 0。
标度格式（C 语言语法）	如果配置标度格式（例如，“%d s”），则标度会显示标签。
标度比	标度的大小，占总大小的百分比

条形

图表类型	下拉列表提供了用于放置条形和标度的各种选项： • 条形旁的标度 • 条形中的标度 • 标度中的条形 • 无标度
对齐方式	条形可以水平或垂直对齐。对齐方式由宽度和高度的比例决定，在此处无法编辑。在可视化编辑器 [▶ 344]中，通过使用鼠标“抓取”元素的 1 个角点并以水平或垂直的方式拖动它，可以对它进行更改。
移动的方向	如果对齐方式为水平，则有以下选择： • 从左至右 • 从右至左 如果对齐方式为垂直，则有以下选择： • 从下至上 • 从上至下

状态变量

这些是对元素在在线模式下的可用性的动态定义。

不可见	指定 1 个 Boolean 变量。如果返回 TRUE，则元素在在线模式下不可见。
输入已禁用	指定 1 个 Boolean 变量。如果返回 TRUE，则元素的输入无效。此外，元素本身在可视化中显示为灰色，表示用户无法输入。 如果可视化使用用户管理 [▶ 360]，则访问权限为“only visible”（仅可见）的用户组的元素会显示为灰色。

访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

15.8.2.11 SpinControl

通过点击元素“SpinControl”右侧的小箭头键，该元素可使变量的值递增或递减。

属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用  按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

附加设置

变量	数值变量，其值在用户输入之后会增加或减少。
数字格式	格式化序列，用于显示变量值。 示例：%i 表示整数变量
间隔时间	在用户输入之后，变量值递增或递减的间隔时间（增量）。

值范围

最小值	输入最大值。
最大值	输入最小值。

文本属性

使用	以下 2 个设置可供选择： • 默认样式值 • 用户定义的设置 ◦ 水平对齐方式 ◦ 垂直对齐方式 ◦ 字体 ◦ 字体颜色

用户定义的文本属性

水平对齐方式	通过选择以下各项来定义文本的水平对齐方式： • 居左 • 居中 • 居右
垂直对齐方式	通过选择以下各项来定义文本的垂直对齐方式： • 顶部 • 居中 • 底部
字体：	通过从预定义字体中进行选择来定义字体： • 标准 • 页首标题 • 大标题 • 标题 • 说明 按 … 打开用户定义的字体属性的对话框。
字体颜色	定义元素的字体颜色。可以从选择列表中进行选择，也可以通过点击 … 时打开的对话框进行选择。

颜色变量

用于通过 DWORD 类型的项目变量动态定义元素颜色的颜色变量。颜色根据由红、绿、蓝（RGB）组件组成的十六进制数进行定义。此外，变量还可用于指定颜色的透明度（FF：完全不透明 - 00：完全透明）。DWORD 的结构如下：16#TTRRGGBB

与 TwinCAT 2 相比，十六进制数的结构有所不同。在 TwinCAT 3 中，除了 RGB 组件之外，还可以用十六进制数来定义颜色透明度。透明度由“16#”后的前 2 位数字表示。定义以“16#00”开头的颜色不可见，因为它们是完全透明的。

示例：

```
nFillColor := 16#FF8FE03F;
```

- FF：透明度（完全不透明）
- 8F：红色
- E0：绿色
- 3F：蓝色

颜色更改	Boolean 变量，用于控制元素颜色在“正常状态”（变量 = FALSE）和“报警状态”（变量 = TRUE）之间的切换。

Windows CE 不支持透明度。

状态变量

这些是对元素在在线模式下的可用性的动态定义。

不可见	指定 1 个 Boolean 变量。如果返回 TRUE，则元素在在线模式下不可见。
输入已禁用	指定 1 个 Boolean 变量。如果返回 TRUE，则元素的输入无效。此外，元素本身在可视化中显示为灰色，表示用户无法输入。 如果可视化使用用户管理 [▶ 360]，则访问权限为“only visible”（仅可见）的用户组的元素会显示为灰色。

输入配置

您可以在此处定义当用户在在线模式下对元素进行输入时应执行的相应动作。只要没有定义后续动作，“Configure…”（配置...）就会出现在 Properties（属性）字段中。点击“Configure…”（配置...），打开输入配置 [▶ 371]，您可以在此处分配后续动作。您可以将许多不同的后续动作分配给每个输入动作。

元素可以使用以下输入事件：

- OnDialogClosed
- OnMouseClick
- OnMouseDown
- OnMouseEnter
- OnMouseLeave
- OnMouseMove
- OnMouseUp

OnDialogClosed	如果在可视化中关闭 1 个为用户输入而打开的对话框，则会触发该事件。 注意：此属性不仅限于为其配置的元素，而是适用于整个可视化。因此，它不会对每个对话框的关闭动作做出响应。目前还没有办法为整个可视化定义这样的属性。因此，必须将它分配给其中 1 个元素。
OnMouseClick	当光标指向 1 个元素并在该项目上执行 1 次完整的鼠标点击（按下并松开鼠标按钮）时，就会触发该鼠标事件。
OnMouseDown	在光标指向 1 个元素的同时按下鼠标按钮时，就会触发该鼠标事件。这与您在可视化上再次松开鼠标按钮的位置无关。
OnMouseEnter	在将光标拖到元素上时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseLeave	在光标离开元素时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseMove	当光标在元素内移动时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseUp	在元素上松开鼠标按钮时，就会发生该鼠标事件。在此之前，已在元素之外按下鼠标按钮。

输入配置 - 热键

热键可用于定义 1 个按键或按键组合，并将其与后续动作（例如，MouseDown、MouseUp）建立关联，当按键事件（KeyDown、KeyUp）发生时，就会执行后续动作。默认情况下，在 KeyDown（按下按键）时会执行 MouseDown 动作，在 KeyUp（松开按键）时会执行 MouseUp 动作。如果要同时通过鼠标动作和键盘输入来操作可视化，这将非常有用，因为输入动作只需配置 1 次。在可视化的热键配置 [▶ 358]中也可以对元素的按键配置进行管理。任何更改始终都会在此编辑器和元素属性编辑器之间同步进行。

按键	分配按键。选择列表包含当前支持的所有按键，例如 M。
事件	如果使用按键或按键和修饰符，则定义要执行的事件。选择列表中可用的可能值： • 无动作 • 在按下按键时，MouseDown 动作 • 在松开按键时，MouseUp 动作 • 在按下松开按键时，MouseDown/MouseUp 动作
移位	如果启用该选项，则按键必须与 Shift 键组合使用。
控制	如果启用该选项，则按键必须与 Ctrl 键组合使用。
Alt	如果启用该选项，则按键必须与 Alt 键组合使用。

访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

15.8.2.12 不可见的输入

该元素在编辑器中显示为虚线，但在在线模式下不可见。在输入配置中必须指定元素的行为。

属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在 View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用  按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

状态变量

这些是对元素在在线模式下的可用性的动态定义。

不可见	指定 1 个 Boolean 变量。如果返回 TRUE，则元素在在线模式下不可见。
输入已禁用	指定 1 个 Boolean 变量。如果返回 TRUE，则元素的输入无效。此外，元素本身在可视化中显示为灰色，表示用户无法输入。 如果可视化使用用户管理 [▶ 360]，则访问权限为“only visible”（仅可见）的用户组的元素会显示为灰色。

输入配置

您可以在此处定义当用户在在线模式下对元素进行输入时应执行的相应动作。只要没有定义后续动作，“Configure…”（配置...）就会出现在 Properties（属性）字段中。点击“Configure…”（配置...），打开输入配置 [▶ 371]，您可以在此处分配后续动作。您可以将许多不同的后续动作分配给每个输入动作。

元素可以使用以下输入事件：

- OnDialogClosed
- OnMouseClick
- OnMouseDown
- OnMouseEnter
- OnMouseLeave
- OnMouseMove
- OnMouseUp

OnDialogClosed	如果在可视化中关闭 1 个为用户输入而打开的对话框，则会触发该事件。 注意：此属性不仅限于为其配置的元素，而是适用于整个可视化。因此，它不会对每个对话框的关闭动作做出响应。目前还没有办法为整个可视化定义这样的属性。因此，必须将它分配给其中 1 个元素。
OnMouseClick	当光标指向 1 个元素并在该项目上执行 1 次完整的鼠标点击（按下并松开鼠标按钮）时，就会触发该鼠标事件。
OnMouseDown	在光标指向 1 个元素的同时按下鼠标按钮时，就会触发该鼠标事件。这与您在可视化上再次松开鼠标按钮的位置无关。
OnMouseEnter	在将光标拖到元素上时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseLeave	在光标离开元素时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseMove	当光标在元素内移动时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseUp	在元素上松开鼠标按钮时，就会发生该鼠标事件。在此之前，已在元素之外按下鼠标按钮。

输入配置 - 热键

热键可用于定义 1 个按键或按键组合，并将其与后续动作（例如，MouseDown、MouseUp）建立关联，当按键事件（KeyDown、KeyUp）发生时，就会执行后续动作。默认情况下，在 KeyDown（按下按键）时会执行 MouseDown 动作，在 KeyUp（松开按键）时会执行 MouseUp 动作。如果要同时通过鼠标动作和键盘输入来操作可视化，这将非常有用，因为输入动作只需配置 1 次。在可视化的热键配置 [▶ 358]中也可以对元素的按键配置进行管理。任何更改始终都会在此编辑器和元素属性编辑器之间同步进行。

按键	分配按键。选择列表包含当前支持的所有按键，例如 M。
事件	如果使用按键或按键和修饰符，则定义要执行的事件。选择列表中可用的可能值： • 无动作 • 在按下按键时，MouseDown 动作 • 在松开按键时，MouseUp 动作 • 在按下松开按键时，MouseDown/MouseUp 动作
移位	如果启用该选项，则按键必须与 Shift 键组合使用。
控制	如果启用该选项，则按键必须与 Ctrl 键组合使用。
Alt	如果启用该选项，则按键必须与 Alt 键组合使用。

状态变量

这些是对元素在在线模式下的可用性的动态定义。

输入已禁用	Boolean 变量。如果返回 TRUE，则元素的输入无效。此外，元素本身在可视化中显示为灰色，表示用户无法输入。 如果可视化使用用户管理 [▶ 360]，则访问权限为“only visible”（仅可见）的用户组的元素会显示为灰色。

访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

15.8.2.13 进度条

进度条可显示操作的进度。这需要 1 个变量，其值显示为条形。您可以设置变量的限值 [▶ 428]和显示样式 [▶ 428]。

属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在 View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用  按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

附加设置

变量	数值变量，其值显示为进度条。
最小值	最小值
最大值	最大值
样式	选择以下样式之一： • 块 • 条

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

文本

文本	您可以在此处指定要输出到按钮栏左侧和上方的文本。如果元素水平对齐，则将文本置于左侧。如果元素垂直对齐，则将文本置于顶部。

状态变量

这些是对元素在在线模式下的可用性的动态定义。

不可见	Boolean 变量。如果返回 TRUE，则元素在在线模式下不可见。

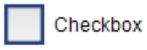
访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

15.8.2.14 复选框

复选框可用于设置和重置 Boolean 变量。如果勾选该方框，则变量将被设置为 TRUE。



属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在 View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用  按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

附加设置

变量	Boolean 变量，用于存储复选框的状态。
边框大小	复选框的边框大小 标准设置：来自样式 [▶ 355]

文本

这些属性可用于元素标签的静态定义。

文本	输入文本。它可用于标注元素。 注意：也可以直接输入文本。如果在可视化编辑器中选择元素，则可以通过按空格键打开输入字段。
工具提示	输入文本。它可以用作元素的工具提示，只有在在线模式下将光标置于元素上时才会出现在可视化中。

文本属性

使用	以下 2 个设置可供选择： • 默认样式值 • 用户定义的设置 ◦ 水平对齐方式 ◦ 垂直对齐方式 ◦ 字体 ◦ 字体颜色

用户定义的文本属性

水平对齐方式	通过选择以下各项来定义文本的水平对齐方式： • 居左 • 居中 • 居右
垂直对齐方式	通过选择以下各项来定义文本的垂直对齐方式： • 顶部 • 居中 • 底部
字体：	通过从预定义字体中进行选择来定义字体： • 标准 • 页首标题 • 大标题 • 标题 • 说明 按 … 打开用户定义的字体属性的对话框。
字体颜色	定义元素的字体颜色。可以从选择列表中进行选择，也可以通过点击 … 时打开的对话框进行选择。

状态变量

这些是对元素在在线模式下的可用性的动态定义。

不可见	指定 1 个 Boolean 变量。如果返回 TRUE，则元素在在线模式下不可见。
输入已禁用	指定 1 个 Boolean 变量。如果返回 TRUE，则元素的输入无效。此外，元素本身在可视化中显示为灰色，表示用户无法输入。 如果可视化使用用户管理 [▶ 360]，则访问权限为“only visible”（仅可见）的用户组的元素会显示为灰色。

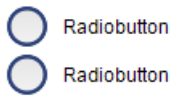
访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

15.8.2.15 单选按钮

单选按钮可以通过 Create new 按钮配置许多不同的选项，这些选项可按 1 列或多列排列。



属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用 ... 按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

附加设置

变量	整数类型的变量，用于存储所选单选按钮的索引。
列数	用于排列单选按钮的列数
单选按钮的排列	选择如何排列单选按钮： • 从左到右：逐行排列单选按钮，直到确定所有按钮的位置。 • 从上到下：逐列排列单选按钮，直到确定所有按钮的位置。
边框大小	单选按钮的大小 标准设置：来自样式 [▶ 355]
行高	行高的定义 标准设置：来自样式 [▶ 355]

文本属性

使用	以下 2 个设置可供选择： • 默认样式值 • 用户定义的设置 ◦ 水平对齐方式 ◦ 垂直对齐方式 ◦ 字体 ◦ 字体颜色

用户定义的文本属性

水平对齐方式	通过选择以下各项来定义文本的水平对齐方式： • 居左 • 居中 • 居右
垂直对齐方式	通过选择以下各项来定义文本的垂直对齐方式： • 顶部 • 居中 • 底部
字体：	通过从预定义字体中进行选择来定义字体： • 标准 • 页首标题 • 大标题 • 标题 • 说明 按 … 打开用户定义的字体属性的对话框。
字体颜色	定义元素的字体颜色。可以从选择列表中进行选择，也可以通过点击 … 时打开的对话框进行选择。

状态变量

这些是对元素在在线模式下的可用性的动态定义。

不可见	指定 1 个 Boolean 变量。如果返回 TRUE，则元素在在线模式下不可见。
输入已禁用	指定 1 个 Boolean 变量。如果返回 TRUE，则元素的输入无效。此外，元素本身在可视化中显示为灰色，表示用户无法输入。 如果可视化使用用户管理 [▶ 360]，则访问权限为“only visible”（仅可见）的用户组的元素会显示为灰色。

单选按钮设置

单选按钮 • 区域 ◦ [<n>]	点击 Create new 按钮，在编辑器中创建 1 个新的单选按钮。然后，在属性编辑器中还会列出更多部分。为每个单选按钮创建 1 个区域，其中涵盖相应的设置。 [<n>]：数字表示该区域的索引。点击 Delete（删除），可删除相应的单选按钮及其设置。

部分 [<n>]

文本	按钮名称 默认值：单选按钮
工具提示	作为工具提示显示的文本
行距（单位：像素）	与顶部按钮的距离（单位：像素）

访问权限

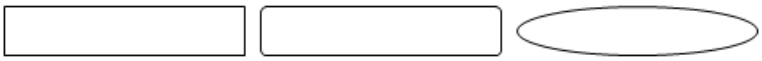
该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

15.8.3 基础

15.8.3.1 矩形、圆角矩形和椭圆形

可视化元素矩形、圆角矩形和椭圆形属于相同的元素类型。只有通过更改元素类型属性，才能将它们互相转换。



属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在 View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用  按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

中心

在编辑数值时，相应的元素 ⊕ 也会同时在可视化编辑器 [▶ 344]中移动。

X	元素枢轴的水平位置（单位：像素）
Y	元素枢轴的垂直位置（单位：像素）

颜色

颜色根据由红/绿/蓝（RGB）组件组成的十六进制数进行定义。这 3 种颜色各有 256（0-255）个值，您可以在此处静态输入。从选择列表中或通过颜色选择对话框可以选择颜色，通过按钮 ... 可以打开颜色选择对话框。此外，还可以为每种颜色设置透明度级别（0：完全透明，255：完全不透明）。

正常状态 • 边框颜色 • 填充颜色	为默认状态选择边框和填充颜色。如果颜色切换变量被定义为 FALSE，则元素处于默认状态。
报警状态 • 边框颜色 • 填充颜色	为处于报警状态的元素选择边框和填充颜色。如果颜色切换变量被定义为 TRUE，则会触发此操作。

使用颜色渐变	默认情况下，未勾选该复选框。如果勾选它，则会以颜色渐变的方式绘制相应的元素。
颜色渐变选择	渐变编辑器 [▶ 370]打开。

Windows CE 不支持颜色渐变和透明度。

外观

Appearance（外观）下的设置是对边框和元素填充的静态定义。

线宽	以像素为单位定义边框线宽。0 的代码与 1 相同，并将线宽设置为 1 像素。如果不需要边框，则将线条类型设置为不可见。
填充类型	定义填充颜色的填充类型： • 填充：填充颜色可见 • 不可见：填充颜色不可见
线条类型	为轮廓定义以下 1 种线条类型： • 实线 • 虚线 • 点线 • 点划线 • 双点划线 • 不可见：轮廓不可见。

文本

这些属性可用于元素标签的静态定义。每个属性都可以包含 1 个格式化序列 [▶ 563]，例如 %s。在在线模式下，在“Text variables”（文本变量）中定义的变量内容可以替换序列。

文本	输入文本。它可用于标注元素。文本可以包含 1 个格式化序列，例如 %s。在“Text variables”（文本变量）中可以定义相应变量。 注意：也可以直接输入文本。如果在可视化编辑器中选择元素，则可以通过按空格键打开输入字段。
工具提示	输入文本。它可以用作元素的工具提示，只有在在线模式下将光标置于元素上时才会出现在可视化中。文本可以包含 1 个格式化序列，例如 %s。在“Text variables”（文本变量）中可以定义相应变量。

文本属性

这些属性可用于字体的静态定义。在“Font variables”（字体变量）类别中可以对字体进行动态定义。

水平对齐方式	通过选择以下各项来定义文本的水平对齐方式： • 居左 • 居中 • 居右
垂直对齐方式	通过选择以下各项来定义文本的垂直对齐方式： • 顶部 • 居中 • 底部
文本格式	定义因文本太长而无法在元素中完整显示时的显示方式： • 默认 - 文本延伸到元素之外。 • 换行 - 文本自动换行。 • 省略 - 尽可能地显示文本，然后用“...”截断.
字体：	通过从预定义字体中进行选择来定义字体： • 标准 • 页首标题 • 大标题 • 标题 • 说明 按 打开用户定义的字体属性的对话框。
字体颜色	定义元素的字体颜色。可以从选择列表中进行选择，也可以通过点击 时打开的对话框进行选择。

绝对移动

通过 1 个整数变量改变元素左上角的 x 和 y 位置（像素），从而可以移动元素。这里使用的是绝对坐标值。

运动 • X • Y	X：此处输入的整数变量可以定义元素左上角当前的 x 位置（单位：像素）。它可用于沿 x 方向移动元素。（正值会将元素从左至右移动）。 Y：此处输入的整数变量可以定义元素左上角当前的 y 位置（单位：像素）。它可用于沿 y 方向移动元素。（正值会将元素从上至下移动）。
旋转	此处输入的整数变量可以定义围绕旋转点旋转元素的角度（角度数）。 正值 = 顺时针方向 注意：与“internal rotation”（内部旋转）行为（见下文）相比，元素本身不会旋转。 点击元素，可显示旋转点 ⊕ 。按住鼠标按钮，即可移动它。
缩放	此处输入的整数变量可以定义当前的缩放因子（百分比）。元素大小可根据该值进行线性调整。该值隐式除以 1000，因此无需使用 REAL 变量来缩小元素。缩放总是指旋转点（中心）。点击元素，可显示旋转点 ⊕ 。按住鼠标按钮，即可移动它。
内部旋转	此处输入的整数变量可以定义元素绕其旋转点旋转的角度（角度数）；正值=数学上的正数=顺时针方向。与“Rotation”（旋转）（见上文）相比，元素本身会旋转。点击元素，可显示旋转点（中心） ⊕ 。按住鼠标按钮，即可移动它。

相对移动

元素可以相对于其固定位置移动。按照整数变量定义的值（像素），元素的左上边缘和右下边缘会沿 x 或 y 方向移动。与绝对移动相比，可以定义相对位置，即与原始位置的距离。此功能可用于更改元素的形状。正值会使水平边缘向下移动和/或使垂直边缘向右移动。

左上移动 • X • Y	• X：整数变量，其值表示左上角沿 x 方向移动的像素数。 • Y：整数变量，其值表示左上角沿 y 方向移动的像素数。
右下移动 • X • Y	• X：整数变量，其值表示右下角沿 x 方向移动的像素数。 • Y：整数变量，其值表示右下角沿 y 方向移动的像素数。

文本变量

您可以显示存储在变量中的文本。为此，首先要在“Texts”（文本）属性下定义的文本中添加 1 个格式化序列。然后，分配 1 个变量。在在线模式下，变量的内容可以替换格式化序列 [▶ 563]。

示例 %f：

在“Texts”（文本）属性中，输入“Result: %2.5f”。在“Text variables”（文本变量）属性中，输入“fValue”。该变量必须是已定义的 IEC 变量。如果 fValue = 12.1234567，则该元素将在在线模式下标注为“Result: 12.12345”。

文本变量	（默认数据类型的）变量，包含要显示的信息。类型必须与“Texts”（文本）设置中的格式化序列一致。
Tooltipvariable	字符串类型的变量，包含要显示的工具提示文本。“Texts”（文本）属性中的条目必须包含格式化序列。

动态文本

这些参数可用于定义源自文本列表 [▶ 127]的动态文本。例如，这可以实现语言更改 [▶ 563]。

动态定义文本的另一种可能性是通过字符串变量提供文本。（请参见“文本变量”类别）。

文本列表	在项目树中，作为字符串使用的文本列表的名称 示例：“TL_ErrorList”
Textindex	在文本列表中，作为字符串定义的文本的索引（ID）。它可以以静态方式直接指定，也可以作为字符串变量进行指定。
Tooltipindex	在文本列表中，作为字符串定义的工具提示文本的索引（ID）。它可以以静态方式直接指定，也可以作为字符串变量进行指定。

字体变量

这些变量可用于通过项目变量对元素文本进行动态字体定义。在“Text properties”（文本属性）下可以配置静态定义。

字体名称	指定 1 个字符串类型的变量，该变量包含用于标注元素的字体名称。（指定的名称，如在标准字体对话框中所示） 示例：MAIN.sFont (sFont := ‘Arial’;)
大小	INT 类型的变量，包含元素文本的大小（单位：像素），如在默认对话框“Font”（字体）中所示。 示例：MAIN.nHeight (nHeight := 16;)
标志	DWORD 类型的变量，用于通过下列标志值之一定义字体显示。通过将相关的标志值相加并指定总和可以实现组合定义。 • 1：斜体 • 2：粗体 • 4：下划线 • 8：删除 示例： MAIN.nFlag (nFlag := 6;) 文字显示为粗体并加下划线。
字符集	通过默认字体编号可以定义字体所使用的字符集。通过 DWORD 变量可以指定该数字（另请参见默认字体对话框中的定义）。
颜色	DWORD 类型的变量，用于定义元素文本的颜色。
文本对齐方式的标志	DWORD 类型的变量，用于通过下列标志值之一定义文本对齐方式。通过将相关的标志值相加并指定总和可以实现组合定义。 • 0：左上 • 1：水平居中 • 2：居右 • 4：垂直居中 • 8：居下 **示例** MAIN.nFlag (nFlag := 5;) 以水平和垂直居中的方式显示文本。

颜色变量

用于通过 DWORD 类型的项目变量动态定义元素颜色的颜色变量。颜色根据由红、绿、蓝（RGB）组件组成的十六进制数进行定义。此外，变量还可用于指定颜色的透明度（FF：完全不透明 - 00：完全透明）。DWORD 的结构如下：16#TTRRGGBB

与 TwinCAT 2 相比，十六进制数的结构有所不同。在 TwinCAT 3 中，除了 RGB 组件之外，还可以用十六进制数来定义颜色透明度。透明度由“16#”后的前 2 位数字表示。定义以“16#00”开头的颜色不可见，因为它们是完全透明的。

示例：

```
nFillColor := 16#FF8FE03F;
```

- FF：透明度（完全不透明）
- 8F：红色
- E0：绿色
- 3F：蓝色

颜色更改	Boolean 变量，用于控制元素颜色在“正常状态”（变量 = FALSE）和“报警状态”（变量 = TRUE）之间的切换。
正常状态 • 边框颜色 • 填充颜色	DWORD 类型的变量，用于定义元素的边框和填充颜色。它们会覆盖当前在“Colors”（颜色）中定义的值。如果在“Color change”（颜色更改）中定义的变量为 FALSE，则会使用项目变量中的值。
报警状态 • 边框颜色 • 填充颜色	DWORD 类型的变量，用于定义报警状态下的元素的边框和填充颜色。它们会覆盖当前在“Colors”（颜色）中定义的值。如果在“Color change”（颜色更改）中定义的变量为 TRUE，则会使用项目变量中的值。

Windows CE 不支持透明度。

外观变量

用于轮廓外观和元素填充的动态定义。在“Appearance”（外观）下可以指定静态定义。

线宽	整数类型的变量，用于定义元素的线宽（单位：像素）。它会覆盖在“Appearance”（外观）下指定的固定值。
填充类型	DWORD 类型的变量，用于定义元素填充。可以显示或忽略在颜色变量下指定的颜色： • 变量值 = 0：填充 • 变量值 > 0：不可见，即不显示填充
线条类型	DWORD 类型的变量，用于定义轮廓。以下值与以下线条类型相对应： • 0：实线 • 1：虚线 • 2：点线 • 3：点划线 • 4：双点划线 • 8：不可见。也就是说，显示的元素没有轮廓。

状态变量

这些是对元素在在线模式下的可用性的动态定义。

不可见	指定 1 个 Boolean 变量。如果返回 TRUE，则元素在在线模式下不可见。
输入已禁用	指定 1 个 Boolean 变量。如果返回 TRUE，则元素的输入无效。此外，元素本身在可视化中显示为灰色，表示用户无法输入。 如果可视化使用用户管理 [▶ 360]，则访问权限为“only visible”（仅可见）的用户组的元素会显示为灰色。

输入配置

您可以在此处定义当用户在在线模式下对元素进行输入时应执行的相应动作。只要没有定义后续动作，“Configure…”（配置...）就会出现在 Properties（属性）字段中。点击“Configure…”（配置...），打开输入配置 [▶ 371]，您可以在此处分配后续动作。您可以将许多不同的后续动作分配给每个输入动作。

元素可以使用以下输入事件：

- OnDialogClosed
- OnMouseClick
- OnMouseDown
- OnMouseEnter
- OnMouseLeave
- OnMouseMove
- OnMouseUp

OnDialogClosed	如果在可视化中关闭 1 个为用户输入而打开的对话框，则会触发该事件。 注意：此属性不仅限于为其配置的元素，而是适用于整个可视化。因此，它不会对每个对话框的关闭动作做出响应。目前还没有办法为整个可视化定义这样的属性。因此，必须将它分配给其中 1 个元素。
OnMouseClick	当光标指向 1 个元素并在该项目上执行 1 次完整的鼠标点击（按下并松开鼠标按钮）时，就会触发该鼠标事件。
OnMouseDown	在光标指向 1 个元素的同时按下鼠标按钮时，就会触发该鼠标事件。这与您在可视化上再次松开鼠标按钮的位置无关。
OnMouseEnter	在将光标拖到元素上时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseLeave	在光标离开元素时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseMove	当光标在元素内移动时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseUp	在元素上松开鼠标按钮时，就会发生该鼠标事件。在此之前，已在元素之外按下鼠标按钮。

输入配置 - 热键

热键可用于定义 1 个按键或按键组合，并将其与后续动作（例如，MouseDown、MouseUp）建立关联，当按键事件（KeyDown、KeyUp）发生时，就会执行后续动作。默认情况下，在 KeyDown（按下按键）时会执行 MouseDown 动作，在 KeyUp（松开按键）时会执行 MouseUp 动作。如果要同时通过鼠标动作和键盘输入来操作可视化，这将非常有用，因为输入动作只需配置 1 次。在可视化的热键配置 [▶ 358]中也可以对元素的按键配置进行管理。任何更改始终都会在此编辑器和元素属性编辑器之间同步进行。

按键	分配按键。选择列表包含当前支持的所有按键，例如 M。
事件	如果使用按键或按键和修饰符，则定义要执行的事件。选择列表中可用的可能值： • 无动作 • 在按下按键时，MouseDown 动作 • 在松开按键时，MouseUp 动作 • 在按下松开按键时，MouseDown/MouseUp 动作
移位	如果启用该选项，则按键必须与 Shift 键组合使用。
控制	如果启用该选项，则按键必须与 Ctrl 键组合使用。
Alt	如果启用该选项，则按键必须与 Alt 键组合使用。

输入配置 - 按键

“按键”可用于指定根据事件“按键”的鼠标行为设置 Boolean 项目变量的值。

变量	Boolean 变量，如果在光标指向元素的同时按下鼠标按钮，则该变量的值为 TRUE。当松开鼠标按钮或光标离开元素时，该值又会变为 FALSE。
FALSE 按键	如果启用该选项，则相应变量的上述按键行为将会反转。也就是说，当按下鼠标按钮时，变量值会被设置为 FALSE。当松开鼠标按钮时，变量值会被设置为 TRUE。
如果鼠标受限，则启动输入	如果启用该选项，只要光标在元素区域内，变量值就会如“Variables”（变量）中所述发挥作用。如果在按下鼠标按钮的同时光标离开元素区域，则该变量会被设置为 FALSE。不过，如果在光标返回元素区域时没有松开鼠标按钮，则会将它再次自动设置为 TRUE。换句话说，只要按下鼠标按钮，即使光标离开元素区域，系统也会考虑到鼠标“受限”的情况。

输入配置 - 切换

“切换”可用于指定根据鼠标行为针对该事件设置 Boolean 项目变量的值。

变量	Boolean 变量，每当鼠标点击元素时，该变量值会在 TRUE 和 FALSE 之间切换。如果在按下鼠标按钮的同时光标远离元素，则不会发生切换。可用于取消切换动作。
如果鼠标受限，则启动松开	如果启用该选项，只要光标在元素区域内，变量值就会如“Variables”（变量）中所述发挥作用。如果在按下鼠标按钮的同时光标离开元素区域，然后松开鼠标按钮，则仍会切换变量。换句话说，只要按下鼠标按钮，即使光标离开元素区域，系统也会考虑到鼠标“受限”的情况。

访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

15.8.3.2 线条

元素“Line”（线条）是 1 条简单的线，其特征由 2 个定义的点进行描述。

属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在 View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用  按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

位置

可视化元素的特征位置（X/Y 坐标）必须以像素为单位进行定义。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改相应的元素。

定义元素的点数可以增加或减少。在可视化编辑器中选择 1 个点，同时按下 Ctrl 键，就会复制该点。这会为属性中的另一点自动添加 1 个条目。如要删除 1 个点，可选择该点，同时按下 Ctrl 和 Shift 键。然后就会从属性中删除相应的条目。

[0] X / Y ..... [n] X / Y	各个元素点的 X/Y 位置（单位：像素） [0] 是起点的位置。以下各点按顺序编号。
X	水平位置（单位：像素）。 X=0 为窗口的左边缘。
Y	垂直位置（单位：像素）。 Y=0 为窗口的上边缘。

中心

在编辑数值时，相应的元素 ⊕ 也会同时在可视化编辑器 [▶ 344]中移动。

X	元素枢轴的水平位置（单位：像素）
Y	元素枢轴的垂直位置（单位：像素）

颜色

颜色根据由红/绿/蓝（RGB）组件组成的十六进制数进行定义。这 3 种颜色各有 256（0-255）个值，您可以在此处静态输入。从选择列表中或通过颜色选择对话框可以选择颜色，通过按钮 [...] 可以打开颜色选择对话框。此外，还可以为每种颜色设置透明度级别（0：完全透明，255：完全不透明）。

颜色	为默认状态选择边框和填充颜色。如果颜色切换变量被定义为 FALSE，则元素处于默认状态。
报警颜色	为处于报警状态的元素选择边框和填充颜色。如果颜色切换变量被定义为 TRUE，则会触发此操作。

Windows CE 不支持透明度。

外观

线宽	以像素为单位定义边框线宽。0 的代码与 1 相同，并将线宽设置为 1 像素。如果不需要边框，则将线条类型设置为不可见。
线条类型	为轮廓定义以下 1 种线条类型： • 实线 • 虚线 • 点线 • 点划线 • 双点划线 • 不可见：轮廓不可见。

文本

这些属性可用于元素标签的静态定义。每个属性都可以包含 1 个格式化序列 [▶ 563]，例如 %s。在在线模式下，在“Text variables”（文本变量）中定义的变量内容可以替换序列。

文本	输入文本。它可用于标注元素。文本可以包含 1 个格式化序列，例如 %s。在“Text variables”（文本变量）中可以定义相应变量。 注意：也可以直接输入文本。如果在可视化编辑器中选择元素，则可以通过按空格键打开输入字段。
工具提示	输入文本。它可以用作元素的工具提示，只有在在线模式下将光标置于元素上时才会出现在可视化中。文本可以包含 1 个格式化序列，例如 %s。在“Text variables”（文本变量）中可以定义相应变量。

文本属性

这些属性可用于字体的静态定义。在“Font variables”（字体变量）类别中可以对字体进行动态定义。

水平对齐方式	通过选择以下各项来定义文本的水平对齐方式： • 居左 • 居中 • 居右
垂直对齐方式	通过选择以下各项来定义文本的垂直对齐方式： • 顶部 • 居中 • 底部
文本格式	定义因文本太长而无法在元素中完整显示时的显示方式： • 默认 - 文本延伸到元素之外。 • 换行 - 文本自动换行。 • 省略 - 尽可能地显示文本，然后用“...”截断.
字体：	通过从预定义字体中进行选择来定义字体： • 标准 • 页首标题 • 大标题 • 标题 • 说明 按  打开用户定义的字体属性的对话框。
字体颜色	定义元素的字体颜色。可以从选择列表中进行选择，也可以通过点击  时打开的对话框进行选择。

绝对移动

通过 1 个整数变量改变元素左上角的 x 和 y 位置（像素），从而可以移动元素。这里使用的是绝对坐标值。

运动 • X • Y	X：此处输入的整数变量可以定义元素左上角当前的 x 位置（单位：像素）。它可用于沿 x 方向移动元素。（正值会将元素从左至右移动）。 Y：此处输入的整数变量可以定义元素左上角当前的 y 位置（单位：像素）。它可用于沿 y 方向移动元素。（正值会将元素从上至下移动）。
旋转	此处输入的整数变量可以定义围绕旋转点旋转元素的角度（角度数）。 正值 = 顺时针方向 注意：与“internal rotation”（内部旋转）行为（见下文）相比，元素本身不会旋转。 点击元素，可显示旋转点 ⊕ 。按住鼠标按钮，即可移动它。
缩放	此处输入的整数变量可以定义当前的缩放因子（百分比）。元素大小可根据该值进行线性调整。该值隐式除以 1000，因此无需使用 REAL 变量来缩小元素。缩放总是指旋转点（中心）。点击元素，可显示旋转点 ⊕ 。按住鼠标按钮，即可移动它。
内部旋转	此处输入的整数变量可以定义元素绕其旋转点旋转的角度（角度数）；正值=数学上的正数=顺时针方向。与“Rotation”（旋转）（见上文）相比，元素本身会旋转。点击元素，可显示旋转点（中心） ⊕ 。按住鼠标按钮，即可移动它。

相对移动

元素可以相对于其固定位置移动。按照整数变量定义的值（像素），元素的左上边缘和右下边缘会沿 x 或 y 方向移动。与绝对移动相比，可以定义相对位置，即与原始位置的距离。此功能可用于更改元素的形状。正值会使水平边缘向下移动和/或使垂直边缘向右移动。

移动点 [n] • X • Y	• X：整数变量，其值表示 point[0]/point[1] 沿 x 方向移动的像素数。 • Y：整数变量，其值表示 point[0]/point[1] 沿 y 方向移动的像素数。

文本变量

您可以显示存储在变量中的文本。为此，首先要在“Texts”（文本）属性下定义的文本中添加 1 个格式化序列。然后，分配 1 个变量。在在线模式下，变量的内容可以替换格式化序列 [▶ 563]。

示例 %f：

在“Texts”（文本）属性中，输入“Result: %2.5f”。在“Text variables”（文本变量）属性中，输入“fValue”。该变量必须是已定义的 IEC 变量。如果 fValue = 12.1234567，则该元素将在在线模式下标注为“Result: 12.12345”。

文本变量	（默认数据类型的）变量，包含要显示的信息。类型必须与“Texts”（文本）设置中的格式化序列一致。
Tooltipvariable	字符串类型的变量，包含要显示的工具提示文本。“Texts”（文本）属性中的条目必须包含格式化序列。

动态文本

这些参数可用于定义源自文本列表 [▶ 127]的动态文本。例如，这可以实现语言更改 [▶ 563]。

动态定义文本的另一种可能性是通过字符串变量提供文本。（请参见“文本变量”类别）。

文本列表	在项目树中，作为字符串使用的文本列表的名称 示例：“TL_ErrorList”
Textindex	在文本列表中，作为字符串定义的文本的索引（ID）。它可以以静态方式直接指定，也可以作为字符串变量进行指定。
Tooltipindex	在文本列表中，作为字符串定义的工具提示文本的索引（ID）。它可以以静态方式直接指定，也可以作为字符串变量进行指定。

字体变量

这些变量可用于通过项目变量对元素文本进行动态字体定义。在“Text properties”（文本属性）下可以配置静态定义。

字体名称	指定 1 个字符串类型的变量，该变量包含用于标注元素的字体名称。（指定的名称，如在标准字体对话框中所示） 示例：MAIN.sFont (sFont := ‘Arial’ ;)
大小	INT 类型的变量，包含元素文本的大小（单位：像素），如在默认对话框“Font”（字体）中所示。 示例：MAIN.nHeight (nHeight := 16;)
标志	DWORD 类型的变量，用于通过下列标志值之一定义字体显示。通过将相关的标志值相加并指定总和可以实现组合定义。 • 1：斜体 • 2：粗体 • 4：下划线 • 8：删除 示例： MAIN.nFlag (nFlag := 6;) 文字显示为粗体并加下划线。
字符集	通过默认字体编号可以定义字体所使用的字符集。通过 DWORD 变量可以指定该数字（另请参见默认字体对话框中的定义）。
颜色	DWORD 类型的变量，用于定义元素文本的颜色。
文本对齐方式的标志	DWORD 类型的变量，用于通过下列标志值之一定义文本对齐方式。通过将相关的标志值相加并指定总和可以实现组合定义。 • 0：左上 • 1：水平居中 • 2：居右 • 4：垂直居中 • 8：居下 **示例** MAIN.nFlag (nFlag := 5;) 以水平和垂直居中的方式显示文本。

颜色变量

用于通过 DWORD 类型的项目变量动态定义元素颜色的颜色变量。颜色根据由红、绿、蓝（RGB）组件组成的十六进制数进行定义。此外，变量还可用于指定颜色的透明度（FF：完全不透明 - 00：完全透明）。DWORD 的结构如下：16#TTRRGGBB

十六进制数的结构与 TwinCAT 2 不同。在 TwinCAT 3 中，除了 RGB 比例之外，还可以用十六进制数来定义颜色的透明度。透明度由“16#”后的前 2 位数字表示。定义以“16#00”开头的颜色不可见，因为它们是完全透明的。

示例：

```
nFillColor := 16#FF8FE03F;
```

- FF：透明度（完全不透明）
- 8F：红色
- E0：绿色
- 3F：蓝色

颜色更改	Boolean 变量，用于控制元素颜色在“正常状态”（变量 = FALSE）和“报警状态”（变量 = TRUE）之间的切换。
正常状态	DWORD 类型的变量，用于定义元素颜色。它会覆盖当前在“Colors”（颜色）中定义的值。如果在“Color change”（颜色更改）中定义的变量为 FALSE，则会使用项目变量中的值。
报警状态	DWORD 类型的变量，用于定义报警状态下的元素颜色。它会覆盖当前在“Colors”（颜色）中定义的值。如果在“Color change”（颜色更改）中定义的变量为 TRUE，则会使用项目变量中的值。

Windows CE 不支持透明度。

状态变量

这些是对元素在在线模式下的可用性的动态定义。

不可见	指定 1 个 Boolean 变量。如果返回 TRUE，则元素在在线模式下不可见。
输入已禁用	指定 1 个 Boolean 变量。如果返回 TRUE，则元素的输入无效。此外，元素本身在可视化中显示为灰色，表示用户无法输入。 如果可视化使用用户管理 [▶ 360]，则访问权限为“only visible”（仅可见）的用户组的元素会显示为灰色。

线条类型变量

通过变量动态定义线条类型。

整数值	整数变量，用于指定数据类型。该值定义了线条类型。支持下列变量值： • 0 = 实线 • 1 = 虚线 • 2 = 点线 • 3 = 点划线 • 4 = 双点划线 • 8 = 不可见：线条不可见。 这会覆盖在“Appearance”（外观）类别中指定的固定值。

线宽变量

通过变量动态定义线条元素的宽度。

整数值	整数类型的变量，用于定义元素的线宽（单位：像素）。这会覆盖在“Appearance”（外观）类别中指定的固定值。请注意，0 的代码效果与 1 相同，并将线宽设置为 1 像素。

输入配置

您可以在此处定义当用户在在线模式下对元素进行输入时应执行的相应动作。只要没有定义后续动作，“Configure…”（配置...）就会出现在 Properties（属性）字段中。点击“Configure…”（配置...），打开输入配置 [▶ 371]，您可以在此处分配后续动作。您可以将许多不同的后续动作分配给每个输入动作。

元素可以使用以下输入事件：

- OnDialogClosed
- OnMouseClick
- OnMouseDown
- OnMouseEnter
- OnMouseLeave
- OnMouseMove

- OnMouseUp

OnDialogClosed	如果在可视化中关闭 1 个为用户输入而打开的对话框，则会触发该事件。 注意：此属性不仅限于为其配置的元素，而是适用于整个可视化。因此，它不会对每个对话框的关闭动作做出响应。目前还没有办法为整个可视化定义这样的属性。因此，必须将它分配给其中 1 个元素。
OnMouseClick	当光标指向 1 个元素并在该项目上执行 1 次完整的鼠标点击（按下并松开鼠标按钮）时，就会触发该鼠标事件。
OnMouseDown	在光标指向 1 个元素的同时按下鼠标按钮时，就会触发该鼠标事件。这与您在可视化上再次松开鼠标按钮的位置无关。
OnMouseEnter	在将光标拖到元素上时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseLeave	在光标离开元素时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseMove	当光标在元素内移动时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseUp	在元素上松开鼠标按钮时，就会发生该鼠标事件。在此之前，已在元素之外按下鼠标按钮。

输入配置 - 热键

热键可用于定义 1 个按键或按键组合，并将其与后续动作（例如，MouseDown、MouseUp）建立关联，当按键事件（KeyDown、KeyUp）发生时，就会执行后续动作。默认情况下，在 KeyDown（按下按键）时会执行 MouseDown 动作，在 KeyUp（松开按键）时会执行 MouseUp 动作。如果要同时通过鼠标动作和键盘输入来操作可视化，这将非常有用，因为输入动作只需配置 1 次。在可视化的热键配置 [▶ 358]中也可以对元素的按键配置进行管理。任何更改始终都会在此编辑器和元素属性编辑器之间同步进行。

按键	分配按键。选择列表包含当前支持的所有按键，例如 M。
事件	如果使用按键或按键和修饰符，则定义要执行的事件。选择列表中可用的可能值： • 无动作 • 在按下按键时，MouseDown 动作 • 在松开按键时，MouseUp 动作 • 在按下松开按键时，MouseDown/MouseUp 动作
移位	如果启用该选项，则按键必须与 Shift 键组合使用。
控制	如果启用该选项，则按键必须与 Ctrl 键组合使用。
Alt	如果启用该选项，则按键必须与 Alt 键组合使用。

输入配置 - 按键

“按键”可用于指定根据事件“按键”的鼠标行为设置 Boolean 项目变量的值。

变量	Boolean 变量，如果在光标指向元素的同时按下鼠标按钮，则该变量的值为 TRUE。当松开鼠标按钮或光标离开元素时，该值又会变为 FALSE。
FALSE 按键	如果启用该选项，则相应变量的上述按键行为将会反转。也就是说，当按下鼠标按钮时，变量值会被设置为 FALSE。当松开鼠标按钮时，变量值会被设置为 TRUE。
如果鼠标受限，则启动输入	如果启用该选项，只要光标在元素区域内，变量值就会如“Variables”（变量）中所述发挥作用。如果在按下鼠标按钮的同时光标离开元素区域，则该变量会被设置为 FALSE。不过，如果在光标返回元素区域时没有松开鼠标按钮，则会将它再次自动设置为 TRUE。换句话说，只要按下鼠标按钮，即使光标离开元素区域，系统也会考虑到鼠标“受限”的情况。

输入配置 - 切换

“切换”可用于指定根据鼠标行为针对该事件设置 Boolean 项目变量的值。

变量	Boolean 变量，每当鼠标点击元素时，该变量值会在 TRUE 和 FALSE 之间切换。如果在按下鼠标按钮的同时光标远离元素，则不会发生切换。可用于取消切换动作。
如果鼠标受限，则启动松开	如果启用该选项，只要光标在元素区域内，变量值就会如“Variables”（变量）中所述发挥作用。如果在按下鼠标按钮的同时光标离开元素区域，然后松开鼠标按钮，则仍会切换变量。换句话说，只要按下鼠标按钮，即使光标离开元素区域，系统也会考虑到鼠标“受限”的情况。

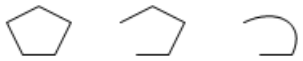
访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

15.8.3.3 多边形、折线或贝塞尔曲线

多边形、折线和贝塞尔曲线属于相同的元素类型。只有通过更改元素类型属性，才能将它们互相转换。



属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在 View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用  按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

位置

可视化元素的特征位置（X/Y 坐标）必须以像素为单位进行定义。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改相应的元素。

定义元素的点数可以增加或减少。在可视化编辑器中选择 1 个点，同时按下 Ctrl 键，就会复制该点。这会为属性中的另一点自动添加 1 个条目。如要删除 1 个点，可选择该点，同时按下 Ctrl 和 Shift 键。然后就会从属性中删除相应的条目。

[0] X / Y ..... [n] X / Y	各个元素点的 X/Y 位置（单位：像素） [0] 是起点的位置。以下各点按顺序编号。
X	水平位置（单位：像素）。 X=0 为窗口的左边缘。
Y	垂直位置（单位：像素）。 Y=0 为窗口的上边缘。

中心

在编辑数值时，相应的元素 ⊕ 也会同时在可视化编辑器 [▶ 344]中移动。

X	元素枢轴的水平位置（单位：像素）
Y	元素枢轴的垂直位置（单位：像素）

颜色

颜色根据由红/绿/蓝（RGB）组件组成的十六进制数进行定义。这 3 种颜色各有 256（0-255）个值，您可以在此处静态输入。从选择列表中或通过颜色选择对话框可以选择颜色，通过按钮 [...] 可以打开颜色选择对话框。此外，还可以为每种颜色设置透明度级别（0：完全透明，255：完全不透明）。

正常状态 • 边框颜色 • 填充颜色	为默认状态选择边框和填充颜色。如果颜色切换变量被定义为 FALSE，则元素处于默认状态。
报警状态 • 边框颜色 • 填充颜色	为处于报警状态的元素选择边框和填充颜色。如果颜色切换变量被定义为 TRUE，则会触发此操作。

使用颜色渐变	默认情况下，未勾选该复选框。如果勾选它，则会以颜色渐变的方式绘制相应的元素。
颜色渐变选择	渐变编辑器 [▶ 370]打开。

Windows CE 不支持颜色渐变和透明度。

外观

Appearance（外观）下的设置是对边框和元素填充的静态定义。

线宽	以像素为单位定义边框线宽。0 的代码与 1 相同，并将线宽设置为 1 像素。如果不需要边框，则将线条类型设置为不可见。
填充类型	定义填充颜色的填充类型： • 填充：填充颜色可见 • 不可见：填充颜色不可见
线条类型	为轮廓定义以下 1 种线条类型： • 实线 • 虚线 • 点线 • 点划线 • 双点划线 • 不可见：轮廓不可见。

文本

这些属性可用于元素标签的静态定义。每个属性都可以包含 1 个格式化序列 [▶ 563]，例如 %s。在在线模式下，在“Text variables”（文本变量）中定义的变量内容可以替换序列。

文本	输入文本。它可用于标注元素。文本可以包含 1 个格式化序列，例如 %s。在“Text variables”（文本变量）中可以定义相应变量。 注意：也可以直接输入文本。如果在可视化编辑器中选择元素，则可以通过按空格键打开输入字段。
工具提示	输入文本。它可以用作元素的工具提示，只有在在线模式下将光标置于元素上时才会出现在可视化中。文本可以包含 1 个格式化序列，例如 %s。在“Text variables”（文本变量）中可以定义相应变量。

文本属性

这些属性可用于字体的静态定义。在“Font variables”（字体变量）类别中可以对字体进行动态定义。

水平对齐方式	通过选择以下各项来定义文本的水平对齐方式： • 居左 • 居中 • 居右
垂直对齐方式	通过选择以下各项来定义文本的垂直对齐方式： • 顶部 • 居中 • 底部
文本格式	定义因文本太长而无法在元素中完整显示时的显示方式： • 默认 - 文本延伸到元素之外。 • 换行 - 文本自动换行。 • 省略 - 尽可能地显示文本，然后用“...”截断.
字体：	通过从预定义字体中进行选择来定义字体： • 标准 • 页首标题 • 大标题 • 标题 • 说明 按 … 打开用户定义的字体属性的对话框。
字体颜色	定义元素的字体颜色。可以从选择列表中进行选择，也可以通过点击 … 时打开的对话框进行选择。

绝对移动

通过 1 个整数变量改变元素左上角的 x 和 y 位置（像素），从而可以移动元素。这里使用的是绝对坐标值。

运动 • X • Y	X：此处输入的整数变量可以定义元素左上角当前的 x 位置（单位：像素）。它可用于沿 x 方向移动元素。（正值会将元素从左至右移动）。 Y：此处输入的整数变量可以定义元素左上角当前的 y 位置（单位：像素）。它可用于沿 y 方向移动元素。（正值会将元素从上至下移动）。
旋转	此处输入的整数变量可以定义围绕旋转点旋转元素的角度（角度数）。 正值 = 顺时针方向 注意：与“internal rotation”（内部旋转）行为（见下文）相比，元素本身不会旋转。 点击元素，可显示旋转点 ⊕ 。按住鼠标按钮，即可移动它。
缩放	此处输入的整数变量可以定义当前的缩放因子（百分比）。元素大小可根据该值进行线性调整。该值隐式除以 1000，因此无需使用 REAL 变量来缩小元素。缩放总是指旋转点（中心）。点击元素，可显示旋转点 ⊕ 。按住鼠标按钮，即可移动它。
内部旋转	此处输入的整数变量可以定义元素绕其旋转点旋转的角度（角度数）；正值=数学上的正数=顺时针方向。与“Rotation”（旋转）（见上文）相比，元素本身会旋转。点击元素，可显示旋转点（中心） ⊕ 。按住鼠标按钮，即可移动它。

动态点

对定义元素的点进行动态定义

点的数组	变量，指向具有 VisuElems.VisuStructPoint 结构的数组。VisuElems.VisuStructPoint 组件 iX 和 iY 包含 1 个点的 X/Y 坐标。VisuElems.VisuStructPoint 组件 iX 和 iY 包含 1 个点的 X/Y 坐标。 在下面的“Number of points”（点的数量）设置中必须明确声明元素点的数量。 示例：pPoints : POINTER TO ARRAY[1..100] OF VisuElems.VisuStructPoint;
点的数量	整数类型的变量，用于定义元素的点的数量。 示例：nCount : INT := 24; 通过 24 个单独的点可以定义该元素。该规范十分必要的，因为通过指针可以定义各个点，而无法控制数量。

如要启用点的动态定义，则必须启用 PLC HMI [▶ 551]、PLC HMI Web [▶ 555]，或者同时启用它们。

文本变量

您可以显示存储在变量中的文本。为此，首先要在“Texts”（文本）属性下定义的文本中添加 1 个格式化序列。然后，分配 1 个变量。在在线模式下，变量的内容可以替换格式化序列 [▶ 563]。

示例 %f：

在“Texts”（文本）属性中，输入“Result: %2.5f”。在“Text variables”（文本变量）属性中，输入“fValue”。该变量必须是已定义的 IEC 变量。如果 fValue = 12.1234567，则该元素将在在线模式下标注为“Result: 12.12345”。

文本变量	（默认数据类型的）变量，包含要显示的信息。类型必须与“Texts”（文本）设置中的格式化序列一致。
Tooltipvariable	字符串类型的变量，包含要显示的工具提示文本。“Texts”（文本）属性中的条目必须包含格式化序列。

动态文本

这些参数可用于定义源自文本列表 [▶ 127]的动态文本。例如，这可以实现语言更改 [▶ 563]。

动态定义文本的另一种可能性是通过字符串变量提供文本。（请参见“文本变量”类别）。

文本列表	在项目树中，作为字符串使用的文本列表的名称 示例：“TL_ErrorList”
Textindex	在文本列表中，作为字符串定义的文本的索引（ID）。它可以以静态方式直接指定，也可以作为字符串变量进行指定。
Tooltipindex	在文本列表中，作为字符串定义的工具提示文本的索引（ID）。它可以以静态方式直接指定，也可以作为字符串变量进行指定。

字体变量

这些变量可用于通过项目变量对元素文本进行动态字体定义。在“Text properties”（文本属性）下可以配置静态定义。

字体名称	指定 1 个字符串类型的变量，该变量包含用于标注元素的字体名称。（指定的名称，如在标准字体对话框中所示） 示例：MAIN.sFont (sFont := ‘Arial’;)
大小	INT 类型的变量，包含元素文本的大小（单位：像素），如在默认对话框“Font”（字体）中所示。 示例：MAIN.nHeight (nHeight := 16;)
标志	DWORD 类型的变量，用于通过下列标志值之一定义字体显示。通过将相关的标志值相加并指定总和可以实现组合定义。 • 1：斜体 • 2：粗体 • 4：下划线 • 8：删除 示例： MAIN.nFlag (nFlag := 6;) 文字显示为粗体并加下划线。
字符集	通过默认字体编号可以定义字体所使用的字符集。通过 DWORD 变量可以指定该数字（另请参见默认字体对话框中的定义）。
颜色	DWORD 类型的变量，用于定义元素文本的颜色。
文本对齐方式的标志	DWORD 类型的变量，用于通过下列标志值之一定义文本对齐方式。通过将相关的标志值相加并指定总和可以实现组合定义。 • 0：左上 • 1：水平居中 • 2：居右 • 4：垂直居中 • 8：居下 **示例** MAIN.nFlag (nFlag := 5;) 以水平和垂直居中的方式显示文本。

外观变量

用于轮廓外观和元素填充的动态定义。在“Appearance”（外观）下可以指定静态定义。

线宽	整数类型的变量，用于定义元素的线宽（单位：像素）。它会覆盖在“Appearance”（外观）下指定的固定值。
填充类型	DWORD 类型的变量，用于定义元素填充。可以显示或忽略在颜色变量下指定的颜色： • 变量值 = 0：填充 • 变量值 > 0：不可见，即不显示填充
线条类型	DWORD 类型的变量，用于定义轮廓。以下值与以下线条类型相对应： • 0：实线 • 1：虚线 • 2：点线 • 3：点划线 • 4：双点划线 • 8：不可见。也就是说，显示的元素没有轮廓。

状态变量

这些是对元素在在线模式下的可用性的动态定义。

不可见	指定 1 个 Boolean 变量。如果返回 TRUE，则元素在在线模式下不可见。
输入已禁用	指定 1 个 Boolean 变量。如果返回 TRUE，则元素的输入无效。此外，元素本身在可视化中显示为灰色，表示用户无法输入。 如果可视化使用用户管理 [▶ 360]，则访问权限为“only visible”（仅可见）的用户组的元素会显示为灰色。

输入配置

您可以在此处定义当用户在在线模式下对元素进行输入时应执行的相应动作。只要没有定义后续动作，“Configure…”（配置...）就会出现在 Properties（属性）字段中。点击“Configure…”（配置...），打开输入配置 [▶ 371]，您可以在此处分配后续动作。您可以将许多不同的后续动作分配给每个输入动作。

元素可以使用以下输入事件：

- OnDialogClosed
- OnMouseClick
- OnMouseDown
- OnMouseEnter
- OnMouseLeave
- OnMouseMove
- OnMouseUp

OnDialogClosed	如果在可视化中关闭 1 个为用户输入而打开的对话框，则会触发该事件。 注意：此属性不仅限于为其配置的元素，而是适用于整个可视化。因此，它不会对每个对话框的关闭动作做出响应。目前还没有办法为整个可视化定义这样的属性。因此，必须将它分配给其中 1 个元素。
OnMouseClick	当光标指向 1 个元素并在该项目上执行 1 次完整的鼠标点击（按下并松开鼠标按钮）时，就会触发该鼠标事件。
OnMouseDown	在光标指向 1 个元素的同时按下鼠标按钮时，就会触发该鼠标事件。这与您在可视化上再次松开鼠标按钮的位置无关。
OnMouseEnter	在将光标拖到元素上时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseLeave	在光标离开元素时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseMove	当光标在元素内移动时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseUp	在元素上松开鼠标按钮时，就会发生该鼠标事件。在此之前，已在元素之外按下鼠标按钮。

输入配置 - 热键

热键可用于定义 1 个按键或按键组合，并将其与后续动作（例如，MouseDown、MouseUp）建立关联，当按键事件（KeyDown、KeyUp）发生时，就会执行后续动作。默认情况下，在 KeyDown（按下按键）时会执行 MouseDown 动作，在 KeyUp（松开按键）时会执行 MouseUp 动作。如果要同时通过鼠标动作和键盘输入来操作可视化，这将非常有用，因为输入动作只需配置 1 次。在可视化的热键配置 [▶ 358]中也可以对元素的按键配置进行管理。任何更改始终都会在此编辑器和元素属性编辑器之间同步进行。

按键	分配按键。选择列表包含当前支持的所有按键，例如 M。
事件	如果使用按键或按键和修饰符，则定义要执行的事件。选择列表中可用的可能值： • 无动作 • 在按下按键时，MouseDown 动作 • 在松开按键时，MouseUp 动作 • 在按下松开按键时，MouseDown/MouseUp 动作
移位	如果启用该选项，则按键必须与 Shift 键组合使用。
控制	如果启用该选项，则按键必须与 Ctrl 键组合使用。
Alt	如果启用该选项，则按键必须与 Alt 键组合使用。

输入配置 - 按键

“按键”可用于指定根据事件“按键”的鼠标行为设置 Boolean 项目变量的值。

变量	Boolean 变量，如果在光标指向元素的同时按下鼠标按钮，则该变量的值为 TRUE。当松开鼠标按钮或光标离开元素时，该值又会变为 FALSE。
FALSE 按键	如果启用该选项，则相应变量的上述按键行为将会反转。也就是说，当按下鼠标按钮时，变量值会被设置为 FALSE。当松开鼠标按钮时，变量值会被设置为 TRUE。
如果鼠标受限，则启动输入	如果启用该选项，只要光标在元素区域内，变量值就会如“Variables”（变量）中所述发挥作用。如果在按下鼠标按钮的同时光标离开元素区域，则该变量会被设置为 FALSE。不过，如果在光标返回元素区域时没有松开鼠标按钮，则会将它再次自动设置为 TRUE。换句话说，只要按下鼠标按钮，即使光标离开元素区域，系统也会考虑到鼠标“受限”的情况。

输入配置 - 切换

“切换”可用于指定根据鼠标行为针对该事件设置 Boolean 项目变量的值。

变量	Boolean 变量，每当鼠标点击元素时，该变量值会在 TRUE 和 FALSE 之间切换。如果在按下鼠标按钮的同时光标远离元素，则不会发生切换。可用于取消切换动作。
如果鼠标受限，则启动松开	如果启用该选项，只要光标在元素区域内，变量值就会如“Variables”（变量）中所述发挥作用。如果在按下鼠标按钮的同时光标离开元素区域，然后松开鼠标按钮，则仍会切换变量。换句话说，只要按下鼠标按钮，即使光标离开元素区域，系统也会考虑到鼠标“受限”的情况。

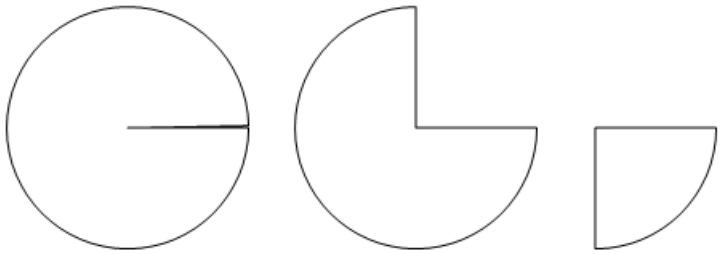
访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

15.8.3.4 扇形

元素“Circular sector”（扇形）基本上表示 1 个完整的圆，其中心位置在将该元素置于可视化中的点上。在设置 [▶ 454]中可以调整要显示的段。



属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用 [...] 按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

附加设置

开始	扇形线的起点
结束	扇形线的终点
开始的变量	变量，用于定义扇形线的起点。启动 [...] 按钮，寻求输入助手的帮助。
结束的变量	变量，用于定义扇形线的终点。启动 [...] 按钮，寻求输入助手的帮助。
只显示圆弧	如果启用该选项，则显示的扇形没有起点和终点的半径线。

中心

在编辑数值时，相应的元素 ⊕ 也会同时在可视化编辑器 [▶ 344]中移动。

X	元素枢轴的水平位置（单位：像素）
Y	元素枢轴的垂直位置（单位：像素）

颜色

颜色根据由红/绿/蓝（RGB）组件组成的十六进制数进行定义。这 3 种颜色各有 256（0-255）个值，您可以在此处静态输入。从选择列表中或通过颜色选择对话框可以选择颜色，通过按钮 ... 可以打开颜色选择对话框。此外，还可以为每种颜色设置透明度级别（0：完全透明，255：完全不透明）。

正常状态 • 边框颜色 • 填充颜色	为默认状态选择边框和填充颜色。如果颜色切换变量被定义为 FALSE，则元素处于默认状态。
报警状态 • 边框颜色 • 填充颜色	为处于报警状态的元素选择边框和填充颜色。如果颜色切换变量被定义为 TRUE，则会触发此操作。

使用颜色渐变	默认情况下，未勾选该复选框。如果勾选它，则会以颜色渐变的方式绘制相应的元素。
颜色渐变选择	渐变编辑器 [▶ 370]打开。

Windows CE 不支持颜色渐变和透明度。

外观

Appearance（外观）下的设置是对边框和元素填充的静态定义。

线宽	以像素为单位定义边框线宽。0 的代码与 1 相同，并将线宽设置为 1 像素。如果不需要边框，则将线条类型设置为不可见。
填充类型	定义填充颜色的填充类型： • 填充：填充颜色可见 • 不可见：填充颜色不可见
线条类型	为轮廓定义以下 1 种线条类型： • 实线 • 虚线 • 点线 • 点划线 • 双点划线 • 不可见：轮廓不可见。

文本

这些属性可用于元素标签的静态定义。每个属性都可以包含 1 个格式化序列 [▶ 563]，例如 %s。在在线模式下，在“Text variables”（文本变量）中定义的变量内容可以替换序列。

文本	输入文本。它可用于标注元素。文本可以包含 1 个格式化序列，例如 %s。在“Text variables”（文本变量）中可以定义相应变量。 注意：也可以直接输入文本。如果在可视化编辑器中选择元素，则可以通过按空格键打开输入字段。
工具提示	输入文本。它可以用作元素的工具提示，只有在在线模式下将光标置于元素上时才会出现在可视化中。文本可以包含 1 个格式化序列，例如 %s。在“Text variables”（文本变量）中可以定义相应变量。

文本属性

这些属性可用于字体的静态定义。在“Font variables”（字体变量）类别中可以对字体进行动态定义。

水平对齐方式	通过选择以下各项来定义文本的水平对齐方式： • 居左 • 居中 • 居右
垂直对齐方式	通过选择以下各项来定义文本的垂直对齐方式： • 顶部 • 居中 • 底部
文本格式	定义因文本太长而无法在元素中完整显示时的显示方式： • 默认 – 文本延伸到元素之外。 • 换行 – 文本自动换行。 • 省略 – 尽可能地显示文本，然后用“...”截断.
字体：	通过从预定义字体中进行选择来定义字体： • 标准 • 页首标题 • 大标题 • 标题 • 说明 按 [...] 打开用户定义的字体属性的对话框。
字体颜色	定义元素的字体颜色。可以从选择列表中进行选择，也可以通过点击 [...] 时打开的对话框进行选择。

绝对移动

通过 1 个整数变量改变元素左上角的 x 和 y 位置（像素），从而可以移动元素。这里使用的是绝对坐标值。

运动 • X • Y	X：此处输入的整数变量可以定义元素左上角当前的 x 位置（单位：像素）。它可用于沿 x 方向移动元素。（正值会将元素从左至右移动）。 Y：此处输入的整数变量可以定义元素左上角当前的 y 位置（单位：像素）。它可用于沿 y 方向移动元素。（正值会将元素从上至下移动）。
旋转	此处输入的整数变量可以定义围绕旋转点旋转元素的角度（角度数）。 正值 = 顺时针方向 注意：与“internal rotation”（内部旋转）行为（见下文）相比，元素本身不会旋转。 点击元素，可显示旋转点 ⊕ 。按住鼠标按钮，即可移动它。
缩放	此处输入的整数变量可以定义当前的缩放因子（百分比）。元素大小可根据该值进行线性调整。该值隐式除以 1000，因此无需使用 REAL 变量来缩小元素。缩放总是指旋转点（中心）。点击元素，可显示旋转点 ⊕ 。按住鼠标按钮，即可移动它。
内部旋转	此处输入的整数变量可以定义元素绕其旋转点旋转的角度（角度数）；正值=数学上的正数=顺时针方向。与“Rotation”（旋转）（见上文）相比，元素本身会旋转。点击元素，可显示旋转点（中心） ⊕ 。按住鼠标按钮，即可移动它。

动态文本

这些参数可用于定义源自文本列表 [▶ 127]的动态文本。例如，这可以实现语言更改 [▶ 563]。

动态定义文本的另一种可能性是通过字符串变量提供文本。（请参见“文本变量”类别）。

文本列表	在项目树中，作为字符串使用的文本列表的名称 示例：“TL_ErrorList”
Textindex	在文本列表中，作为字符串定义的文本的索引（ID）。它可以以静态方式直接指定，也可以作为字符串变量进行指定。
Tooltipindex	在文本列表中，作为字符串定义的工具提示文本的索引（ID）。它可以以静态方式直接指定，也可以作为字符串变量进行指定。

文本变量

您可以显示存储在变量中的文本。为此，首先要在“Texts”（文本）属性下定义的文本中添加 1 个格式化序列。然后，分配 1 个变量。在在线模式下，变量的内容可以替换格式化序列 [▶ 563]。

示例 %f：

在“Texts”（文本）属性中，输入“Result: %2.5f”。在“Text variables”（文本变量）属性中，输入“fValue”。该变量必须是已定义的 IEC 变量。如果 fValue = 12.1234567，则该元素将在在线模式下标注为“Result: 12.12345”。

文本变量	（默认数据类型的）变量，包含要显示的信息。类型必须与“Texts”（文本）设置中的格式化序列一致。
Tooltipvariable	字符串类型的变量，包含要显示的工具提示文本。“Texts”（文本）属性中的条目必须包含格式化序列。

字体变量

这些变量可用于通过项目变量对元素文本进行动态字体定义。在“Text properties”（文本属性）下可以配置静态定义。

字体名称	指定 1 个字符串类型的变量，该变量包含用于标注元素的字体名称。（指定的名称，如在标准字体对话框中所示） 示例：MAIN.sFont (sFont := ‘Arial’ ;)
大小	INT 类型的变量，包含元素文本的大小（单位：像素），如在默认对话框“Font”（字体）中所示。 示例：MAIN.nHeight (nHeight := 16;)
标志	DWORD 类型的变量，用于通过下列标志值之一定义字体显示。通过将相关的标志值相加并指定总和可以实现组合定义。 • 1：斜体 • 2：粗体 • 4：下划线 • 8：删除 示例： MAIN.nFlag (nFlag := 6;) 文字显示为粗体并加下划线。
字符集	通过默认字体编号可以定义字体所使用的字符集。通过 DWORD 变量可以指定该数字（另请参见默认字体对话框中的定义）。
颜色	DWORD 类型的变量，用于定义元素文本的颜色。
文本对齐方式的标志	DWORD 类型的变量，用于通过下列标志值之一定义文本对齐方式。通过将相关的标志值相加并指定总和可以实现组合定义。 • 0：左上 • 1：水平居中 • 2：居右 • 4：垂直居中 • 8：居下 **示例** MAIN.nFlag (nFlag := 5;) 以水平和垂直居中的方式显示文本。

颜色变量

用于通过 DWORD 类型的项目变量动态定义元素颜色的颜色变量。颜色根据由红、绿、蓝（RGB）组件组成的十六进制数进行定义。此外，变量还可用于指定颜色的透明度（FF：完全不透明 - 00：完全透明）。DWORD 的结构如下：16#TTRRGGBB

与 TwinCAT 2 相比，十六进制数的结构有所不同。在 TwinCAT 3 中，除了 RGB 组件之外，还可以用十六进制数来定义颜色透明度。透明度由“16#”后的前 2 位数字表示。定义以“16#00”开头的颜色不可见，因为它们是完全透明的。

示例：

```
nFillColor := 16#FF8FE03F;
```

- FF：透明度（完全不透明）
- 8F：红色
- E0：绿色
- 3F：蓝色

颜色更改	Boolean 变量，用于控制元素颜色在“正常状态”（变量 = FALSE）和“报警状态”（变量 = TRUE）之间的切换。
正常状态 • 边框颜色 • 填充颜色	DWORD 类型的变量，用于定义元素的边框和填充颜色。它们会覆盖当前在“Colors”（颜色）中定义的值。如果在“Color change”（颜色更改）中定义的变量为 FALSE，则会使用项目变量中的值。
报警状态 • 边框颜色 • 填充颜色	DWORD 类型的变量，用于定义报警状态下的元素的边框和填充颜色。它们会覆盖当前在“Colors”（颜色）中定义的值。如果在“Color change”（颜色更改）中定义的变量为 TRUE，则会使用项目变量中的值。

Windows CE 不支持透明度。

状态变量

这些是对元素在在线模式下的可用性的动态定义。

不可见	指定 1 个 Boolean 变量。如果返回 TRUE，则元素在在线模式下不可见。
输入已禁用	指定 1 个 Boolean 变量。如果返回 TRUE，则元素的输入无效。此外，元素本身在可视化中显示为灰色，表示用户无法输入。 如果可视化使用用户管理 [▶ 360]，则访问权限为“only visible”（仅可见）的用户组的元素会显示为灰色。

外观变量

用于轮廓外观和元素填充的动态定义。在“Appearance”（外观）下可以指定静态定义。

线宽	整数类型的变量，用于定义元素的线宽（单位：像素）。它会覆盖在“Appearance”（外观）下指定的固定值。
填充类型	DWORD 类型的变量，用于定义元素填充。可以显示或忽略在颜色变量下指定的颜色： • 变量值 = 0：填充 • 变量值 > 0：不可见，即不显示填充
线条类型	DWORD 类型的变量，用于定义轮廓。以下值与以下线条类型相对应： • 0：实线 • 1：虚线 • 2：点线 • 3：点划线 • 4：双点划线 • 8：不可见。也就是说，显示的元素没有轮廓。

输入配置

您可以在此处定义当用户在在线模式下对元素进行输入时应执行的相应动作。只要没有定义后续动作，“Configure…”（配置...）就会出现在 Properties（属性）字段中。点击“Configure…”（配置...），打开输入配置 [▶ 371]，您可以在此处分配后续动作。您可以将许多不同的后续动作分配给每个输入动作。

元素可以使用以下输入事件：

- OnDialogClosed
- OnMouseClick
- OnMouseDown
- OnMouseEnter
- OnMouseLeave
- OnMouseMove
- OnMouseUp

OnDialogClosed	如果在可视化中关闭 1 个为用户输入而打开的对话框，则会触发该事件。 注意：此属性不仅限于为其配置的元素，而是适用于整个可视化。因此，它不会对每个对话框的关闭动作做出响应。目前还没有办法为整个可视化定义这样的属性。因此，必须将它分配给其中 1 个元素。
OnMouseClick	当光标指向 1 个元素并在该项目上执行 1 次完整的鼠标点击（按下并松开鼠标按钮）时，就会触发该鼠标事件。
OnMouseDown	在光标指向 1 个元素的同时按下鼠标按钮时，就会触发该鼠标事件。这与您在可视化上再次松开鼠标按钮的位置无关。
OnMouseEnter	在将光标拖到元素上时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseLeave	在光标离开元素时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseMove	当光标在元素内移动时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseUp	在元素上松开鼠标按钮时，就会发生该鼠标事件。在此之前，已在元素之外按下鼠标按钮。

输入配置 - 热键

热键可用于定义 1 个按键或按键组合，并将其与后续动作（例如，MouseDown、MouseUp）建立关联，当按键事件（KeyDown、KeyUp）发生时，就会执行后续动作。默认情况下，在 KeyDown（按下按键）时会执行 MouseDown 动作，在 KeyUp（松开按键）时会执行 MouseUp 动作。如果要同时通过鼠标动作和键盘输入来操作可视化，这将非常有用，因为输入动作只需配置 1 次。在可视化的热键配置 [▶ 358]中也可以对元素的按键配置进行管理。任何更改始终都会在此编辑器和元素属性编辑器之间同步进行。

按键	分配按键。选择列表包含当前支持的所有按键，例如 M。
事件	如果使用按键或按键和修饰符，则定义要执行的事件。选择列表中可用的可能值： • 无动作 • 在按下按键时，MouseDown 动作 • 在松开按键时，MouseUp 动作 • 在按下松开按键时，MouseDown/MouseUp 动作
移位	如果启用该选项，则按键必须与 Shift 键组合使用。
控制	如果启用该选项，则按键必须与 Ctrl 键组合使用。
Alt	如果启用该选项，则按键必须与 Alt 键组合使用。

输入配置 - 按键

“按键”可用于指定根据事件“按键”的鼠标行为设置 Boolean 项目变量的值。

变量	Boolean 变量，如果在光标指向元素的同时按下鼠标按钮，则该变量的值为 TRUE。当松开鼠标按钮或光标离开元素时，该值又会变为 FALSE。
FALSE 按键	如果启用该选项，则相应变量的上述按键行为将会反转。也就是说，当按下鼠标按钮时，变量值会被设置为 FALSE。当松开鼠标按钮时，变量值会被设置为 TRUE。
如果鼠标受限，则启动输入	如果启用该选项，只要光标在元素区域内，变量值就会如“Variables”（变量）中所述发挥作用。如果在按下鼠标按钮的同时光标离开元素区域，则该变量会被设置为 FALSE。不过，如果在光标返回元素区域时没有松开鼠标按钮，则会将它再次自动设置为 TRUE。换句话说，只要按下鼠标按钮，即使光标离开元素区域，系统也会考虑到鼠标“受限”的情况。

输入配置 - 切换

“切换”可用于指定根据鼠标行为针对该事件设置 Boolean 项目变量的值。

变量	Boolean 变量，每当鼠标点击元素时，该变量值会在 TRUE 和 FALSE 之间切换。如果在按下鼠标按钮的同时光标远离元素，则不会发生切换。可用于取消切换动作。
如果鼠标受限，则启动松开	如果启用该选项，只要光标在元素区域内，变量值就会如“Variables”（变量）中所述发挥作用。如果在按下鼠标按钮的同时光标离开元素区域，然后松开鼠标按钮，则仍会切换变量。换句话说，只要按下鼠标按钮，即使光标离开元素区域，系统也会考虑到鼠标“受限”的情况。

访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

15.8.3.5 图像

该元素可用于为可视化添加图像。通过指定相应的静态 ID 和图像池 [▶ 134]名称，可将图像分配给图像元素。“图像 ID 变量 [▶ 464]”设置可用于定义要动态显示的图像。

通过可视化编辑器中的背景对话框 [▶ 345]可以设置可视化页面的背景图像。

属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在 View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用  按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

附加设置

静态 ID	静态定义的图像文件的 ID。 输入在相应图像池（字符串）中定义的图像文件的 ID。图像池的名称应加前缀，以便确保条目明确。（如果在全局图像池中管理图像文件，则无需这样做，因为总是首先扫描该图像池。） 示例：IP_ImagePool.ButtonImage 点击 [...] 按钮，打开“输入助手 [▶ 468]”对话框，会显示所有可用图像池以及相关图像的列表。
绘制边框	如果启用该选项，则图像文件会显示 1 个边框。
截断	如果启用该选项与缩放类型“Unscaled”（未缩放），则仅会显示适合元素的那部分图像。
透明度	如果启用该选项，则在属性“Transparency color”（透明度颜色）中指定的颜色将显示为透明。
透明度颜色	如果启用“Transparency”（透明度）选项，则按钮 [...] 会打开颜色选择对话框，用于选择要显示为透明的颜色。
缩放类型	您可以在此处定义图像文件如何响应元素边框大小的变化： • 各向同性 图像保持其比例。也就是说，即使元素边框的高度和宽度分别改变，高宽比也会保持不变。 • 各向异性 图像会自动适应元素边框的大小，即，可以互相独立更改高度和宽度。 • 未缩放 即使元素边框的大小改变，图像也会保持原来的大小。另一个需要考虑的因素是，是否启用“Truncate”（截断）选项。 通过这项设置，每当分配 1 个新的图像 ID 时，元素大小就会自动调整为图像大小。
水平对齐方式	只有启用属性编辑器 [▶ 353]中的“Expert”（专家）选项且图像缩放类型为“Isotropic”（各向同性）时，该设置才可用。它可用于定义相对于元素边框的水平对齐方式。 • 居左 • 居中 • 居右
垂直对齐方式	只有启用属性编辑器 [▶ 353]中的“Expert”（专家）选项且图像缩放类型为“Isotropic”（各向同性）时，该设置才可用。它可用于定义相对于元素边框的垂直对齐方式。 • 顶部 • 居中 • 底部

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

中心

在编辑数值时，相应的元素 ⊕ 也会同时在可视化编辑器 [▶ 344]中移动。

X	元素枢轴的水平位置（单位：像素）
Y	元素枢轴的垂直位置（单位：像素）

颜色

颜色根据由红/绿/蓝（RGB）组件组成的十六进制数进行定义。这 3 种颜色各有 256（0-255）个值，您可以在此处静态输入。从选择列表中或通过颜色选择对话框可以选择颜色，通过按钮 ... 可以打开颜色选择对话框。此外，还可以为每种颜色设置透明度级别（0：完全透明，255：完全不透明）。

颜色	为默认状态选择边框和填充颜色。如果颜色切换变量被定义为 FALSE，则元素处于默认状态。
报警颜色	为处于报警状态的元素选择边框和填充颜色。如果颜色切换变量被定义为 TRUE，则会触发此操作。

Windows CE 不支持透明度。

外观

线宽	以像素为单位定义边框线宽。0 的代码与 1 相同，并将线宽设置为 1 像素。如果不需要边框，则将线条类型设置为不可见。
线条类型	为轮廓定义以下 1 种线条类型： • 实线 • 虚线 • 点线 • 点划线 • 双点划线 • 不可见：轮廓不可见。

文本

这些属性可用于元素标签的静态定义。每个属性都可以包含 1 个格式化序列 [▶ 563]，例如 %s。在在线模式下，在“Text variables”（文本变量）中定义的变量内容可以替换序列。

文本	输入文本。它可用于标注元素。文本可以包含 1 个格式化序列，例如 %s。在“Text variables”（文本变量）中可以定义相应变量。 注意：也可以直接输入文本。如果在可视化编辑器中选择元素，则可以通过按空格键打开输入字段。
工具提示	输入文本。它可以用作元素的工具提示，只有在在线模式下将光标置于元素上时才会出现在可视化中。文本可以包含 1 个格式化序列，例如 %s。在“Text variables”（文本变量）中可以定义相应变量。

文本属性

这些属性可用于字体的静态定义。在“Font variables”（字体变量）类别中可以对字体进行动态定义。

水平对齐方式	通过选择以下各项来定义文本的水平对齐方式： • 居左 • 居中 • 居右
垂直对齐方式	通过选择以下各项来定义文本的垂直对齐方式： • 顶部 • 居中 • 底部
文本格式	定义因文本太长而无法在元素中完整显示时的显示方式： • 默认 – 文本延伸到元素之外。 • 换行 – 文本自动换行。 • 省略 – 尽可能地显示文本，然后用“...”截断.
字体：	通过从预定义字体中进行选择来定义字体： • 标准 • 页首标题 • 大标题 • 标题 • 说明 按 … 打开用户定义的字体属性的对话框。
字体颜色	定义元素的字体颜色。可以从选择列表中进行选择，也可以通过点击 … 时打开的对话框进行选择。

图像 ID 变量

图像 ID	字符串类型的项目变量，包含要显示元素的图像 ID。在图像池 [▶ 134]中对 ID 进行定义。图像池的名称应加前缀，以便确保条目明确。对于在“GlobalImagePool”中管理的图像文件，无需指定图像池，因为总是首先搜索该图像池。

动态图像

通过该属性，在运行时可以重新加载图像文件。例如，这样可以刷新相机图像。

重新加载的图像文件必须

- 在已启动 PLC HMI 客户端的设备的客户端目录中交换（*C:\TwinCAT\3.1\Components\Plc\Tc3PlcHmi\[Port_X]\Visu*），适用于 4022.28 以下版本的 PLC HMI。
- 在执行控制代码的设备的启动目录中交换（*C:\TwinCAT\3.1\Boot\Plc\[Port_X]\Visu*），适用于 4022.28 及以上版本的 PLC HMI 以及 PLC HMI Web。
- 在执行控制代码的设备的启动目录中交换（*:\ProgramData\Beckhoff\TwinCAT\3.1\Boot\Plc\[Port_X]\Visu*），适用于 PLC HMI 以及 TC3.1.4026.0 及以上版本的 PLC HMI Web。

位图版本	包含图像版本的整数数据类型的变量。如果变量值更改，则可视化会重新加载在“Image ID”（图像 ID）属性中引用的图像并显示它。

绝对移动

通过 1 个整数变量改变元素左上角的 x 和 y 位置（像素），从而可以移动元素。这里使用的是绝对坐标值。

运动 • X • Y	X：此处输入的整数变量可以定义元素左上角当前的 x 位置（单位：像素）。它可用于沿 x 方向移动元素。（正值会将元素从左至右移动）。 Y：此处输入的整数变量可以定义元素左上角当前的 y 位置（单位：像素）。它可用于沿 y 方向移动元素。（正值会将元素从上至下移动）。
旋转	此处输入的整数变量可以定义围绕旋转点旋转元素的角度（角度数）。 正值 = 顺时针方向 注意：与“internal rotation”（内部旋转）行为（见下文）相比，元素本身不会旋转。 点击元素，可显示旋转点 ⊕ 。按住鼠标按钮，即可移动它。
缩放	此处输入的整数变量可以定义当前的缩放因子（百分比）。元素大小可根据该值进行线性调整。该值隐式除以 1000，因此无需使用 REAL 变量来缩小元素。缩放总是指旋转点（中心）。点击元素，可显示旋转点 ⊕ 。按住鼠标按钮，即可移动它。
内部旋转	此处输入的整数变量可以定义元素绕其旋转点旋转的角度（角度数）；正值=数学上的正数=顺时针方向。与“Rotation”（旋转）（见上文）相比，元素本身会旋转。点击元素，可显示旋转点（中心） ⊕ 。按住鼠标按钮，即可移动它。

相对移动

元素可以相对于其固定位置移动。按照整数变量定义的值（像素），元素的左上边缘和右下边缘会沿 x 或 y 方向移动。与绝对移动相比，可以定义相对位置，即与原始位置的距离。此功能可用于更改元素的形状。正值会使水平边缘向下移动和/或使垂直边缘向右移动。

左上移动 • X • Y	• X：整数变量，其值表示左上角沿 x 方向移动的像素数。 • Y：整数变量，其值表示左上角沿 y 方向移动的像素数。
右下移动 • X • Y	• X：整数变量，其值表示右下角沿 x 方向移动的像素数。 • Y：整数变量，其值表示右下角沿 y 方向移动的像素数。

文本变量

您可以显示存储在变量中的文本。为此，首先要在“Texts”（文本）属性下定义的文本中添加 1 个格式化序列。然后，分配 1 个变量。在在线模式下，变量的内容可以替换格式化序列 [▶ 563]。

示例 %f：

在“Texts”（文本）属性中，输入“Result: %2.5f”。在“Text variables”（文本变量）属性中，输入“fValue”。该变量必须是已定义的 IEC 变量。如果 fValue = 12.1234567，则该元素将在在线模式下标注为“Result: 12.12345”。

文本变量	（默认数据类型的）变量，包含要显示的信息。类型必须与“Texts”（文本）设置中的格式化序列一致。
Tooltipvariable	字符串类型的变量，包含要显示的工具提示文本。“Texts”（文本）属性中的条目必须包含格式化序列。

动态文本

这些参数可用于定义源自文本列表 [▶ 127]的动态文本。例如，这可以实现语言更改 [▶ 563]。

动态定义文本的另一种可能性是通过字符串变量提供文本。（请参见“文本变量”类别）。

文本列表	在项目树中，作为字符串使用的文本列表的名称 示例：“TL_ErrorList”
Textindex	在文本列表中，作为字符串定义的文本的索引（ID）。它可以以静态方式直接指定，也可以作为字符串变量进行指定。
Tooltipindex	在文本列表中，作为字符串定义的工具提示文本的索引（ID）。它可以以静态方式直接指定，也可以作为字符串变量进行指定。

字体变量

这些变量可用于通过项目变量对元素文本进行动态字体定义。在“Text properties”（文本属性）下可以配置静态定义。

字体名称	指定 1 个字符串类型的变量，该变量包含用于标注元素的字体名称。（指定的名称，如在标准字体对话框中所示） 示例：MAIN.sFont (sFont := ‘Arial’;)
大小	INT 类型的变量，包含元素文本的大小（单位：像素），如在默认对话框“Font”（字体）中所示。 示例：MAIN.nHeight (nHeight := 16;)
标志	DWORD 类型的变量，用于通过下列标志值之一定义字体显示。通过将相关的标志值相加并指定总和可以实现组合定义。 • 1：斜体 • 2：粗体 • 4：下划线 • 8：删除 示例： MAIN.nFlag (nFlag := 6;) 文字显示为粗体并加下划线。
字符集	通过默认字体编号可以定义字体所使用的字符集。通过 DWORD 变量可以指定该数字（另请参见默认字体对话框中的定义）。
颜色	DWORD 类型的变量，用于定义元素文本的颜色。
文本对齐方式的标志	DWORD 类型的变量，用于通过下列标志值之一定义文本对齐方式。通过将相关的标志值相加并指定总和可以实现组合定义。 • 0：左上 • 1：水平居中 • 2：居右 • 4：垂直居中 • 8：居下 **示例** MAIN.nFlag (nFlag := 5;) 以水平和垂直居中的方式显示文本。

状态变量

这些是对元素在在线模式下的可用性的动态定义。

不可见	指定 1 个 Boolean 变量。如果返回 TRUE，则元素在在线模式下不可见。
输入已禁用	指定 1 个 Boolean 变量。如果返回 TRUE，则元素的输入无效。此外，元素本身在可视化中显示为灰色，表示用户无法输入。 如果可视化使用用户管理 [▶ 360]，则访问权限为“only visible”（仅可见）的用户组的元素会显示为灰色。

输入配置

您可以在此处定义当用户在在线模式下对元素进行输入时应执行的相应动作。只要没有定义后续动作，“Configure…”（配置...）就会出现在 Properties（属性）字段中。点击“Configure…”（配置...），打开输入配置 [▶ 371]，您可以在此处分配后续动作。您可以将许多不同的后续动作分配给每个输入动作。

元素可以使用以下输入事件：

- OnDialogClosed
- OnMouseClick
- OnMouseDown

- OnMouseEnter
- OnMouseLeave
- OnMouseMove
- OnMouseUp

OnDialogClosed	如果在可视化中关闭 1 个为用户输入而打开的对话框，则会触发该事件。 注意：此属性不仅限于为其配置的元素，而是适用于整个可视化。因此，它不会对每个对话框的关闭动作做出响应。目前还没有办法为整个可视化定义这样的属性。因此，必须将它分配给其中 1 个元素。
OnMouseClick	当光标指向 1 个元素并在该项目上执行 1 次完整的鼠标点击（按下并松开鼠标按钮）时，就会触发该鼠标事件。
OnMouseDown	在光标指向 1 个元素的同时按下鼠标按钮时，就会触发该鼠标事件。这与您在可视化上再次松开鼠标按钮的位置无关。
OnMouseEnter	在将光标拖到元素上时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseLeave	在光标离开元素时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseMove	当光标在元素内移动时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseUp	在元素上松开鼠标按钮时，就会发生该鼠标事件。在此之前，已在元素之外按下鼠标按钮。

输入配置 - 热键

热键可用于定义 1 个按键或按键组合，并将其与后续动作（例如，MouseDown、MouseUp）建立关联，当按键事件（KeyDown、KeyUp）发生时，就会执行后续动作。默认情况下，在 KeyDown（按下按键）时会执行 MouseDown 动作，在 KeyUp（松开按键）时会执行 MouseUp 动作。如果要同时通过鼠标动作和键盘输入来操作可视化，这将非常有用，因为输入动作只需配置 1 次。在可视化的热键配置 [▶ 358]中也可以对元素的按键配置进行管理。任何更改始终都会在此编辑器和元素属性编辑器之间同步进行。

按键	分配按键。选择列表包含当前支持的所有按键，例如 M。
事件	如果使用按键或按键和修饰符，则定义要执行的事件。选择列表中可用的可能值： • 无动作 • 在按下按键时，MouseDown 动作 • 在松开按键时，MouseUp 动作 • 在按下松开按键时，MouseDown/MouseUp 动作
移位	如果启用该选项，则按键必须与 Shift 键组合使用。
控制	如果启用该选项，则按键必须与 Ctrl 键组合使用。
Alt	如果启用该选项，则按键必须与 Alt 键组合使用。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

颜色变量

用于通过 DWORD 类型的项目变量动态定义元素颜色的颜色变量。颜色根据由红、绿、蓝（RGB）组件组成的十六进制数进行定义。此外，变量还可用于指定颜色的透明度（FF：完全不透明 - 00：完全透明）。DWORD 的结构如下：16#TTRRGGBB

十六进制数的结构与 TwinCAT 2 不同。在 TwinCAT 3 中，除了 RGB 比例之外，还可以用十六进制数来定义颜色的透明度。透明度由“16#”后的前 2 位数字表示。定义以“16#00”开头的颜色不可见，因为它们是完全透明的。

示例：

nFillColor := 16#FF8FE03F;

- FF：透明度（完全不透明）
- 8F：红色
- E0：绿色
- 3F：蓝色

颜色更改	Boolean 变量，用于控制元素颜色在“正常状态”（变量 = FALSE）和“报警状态”（变量 = TRUE）之间的切换。
正常状态	DWORD 类型的变量，用于定义元素颜色。它会覆盖当前在“Colors”（颜色）中定义的值。如果在“Color change”（颜色更改）中定义的变量为 FALSE，则会使用项目变量中的值。
报警状态	DWORD 类型的变量，用于定义报警状态下的元素颜色。它会覆盖当前在“Colors”（颜色）中定义的值。如果在“Color change”（颜色更改）中定义的变量为 TRUE，则会使用项目变量中的值。

Windows CE 不支持透明度。

访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

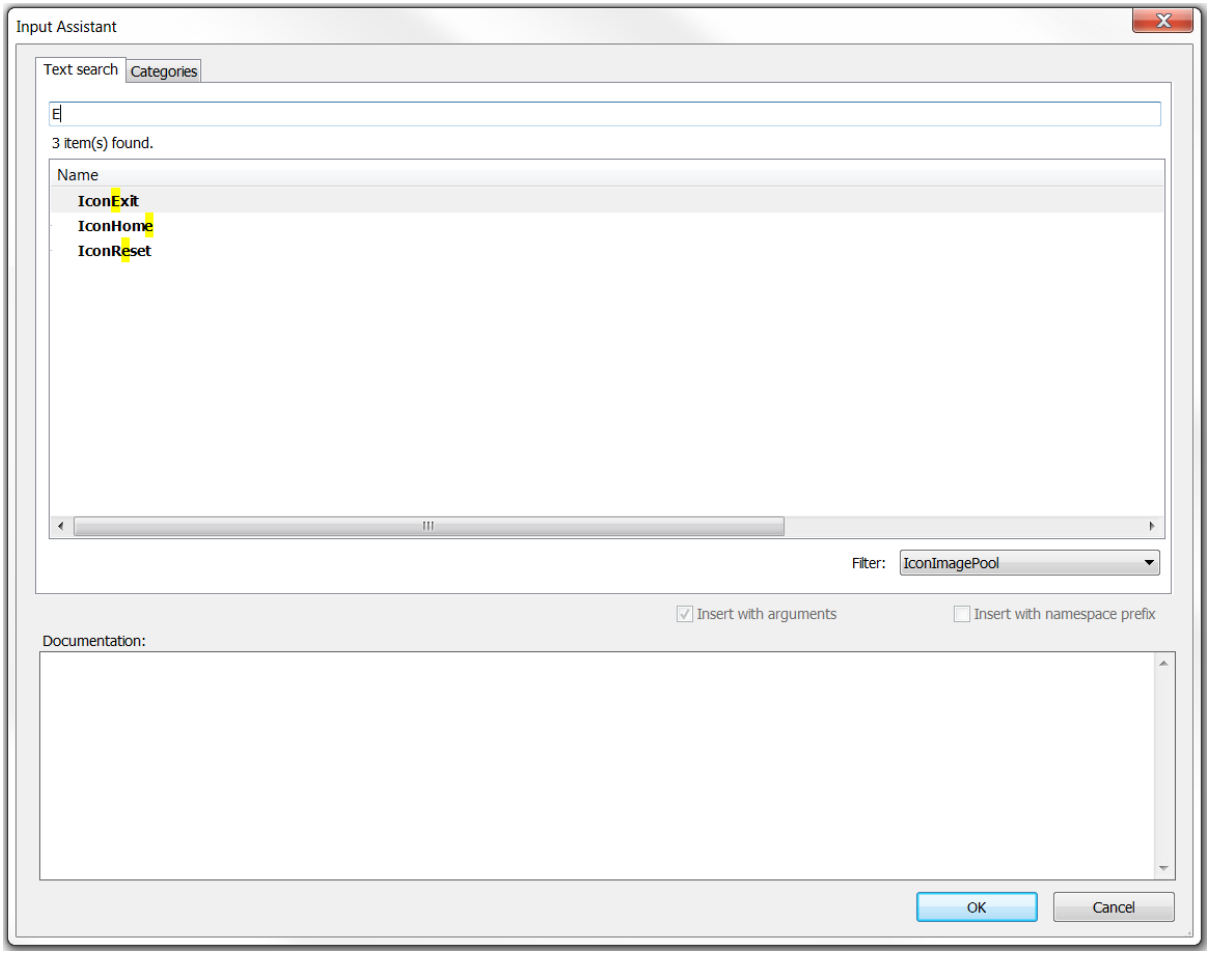
未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

15.8.3.5.1 输入助手

当首次将图像 [▶ 461]类型的可视化元素添加到可视化页面时，图像池 [▶ 134]中的图像的输入助手会自动打开。在图像元素的属性中选择值字段“静态 ID [▶ 462]”，然后点击 ... 按钮，也可以进入该对话框。

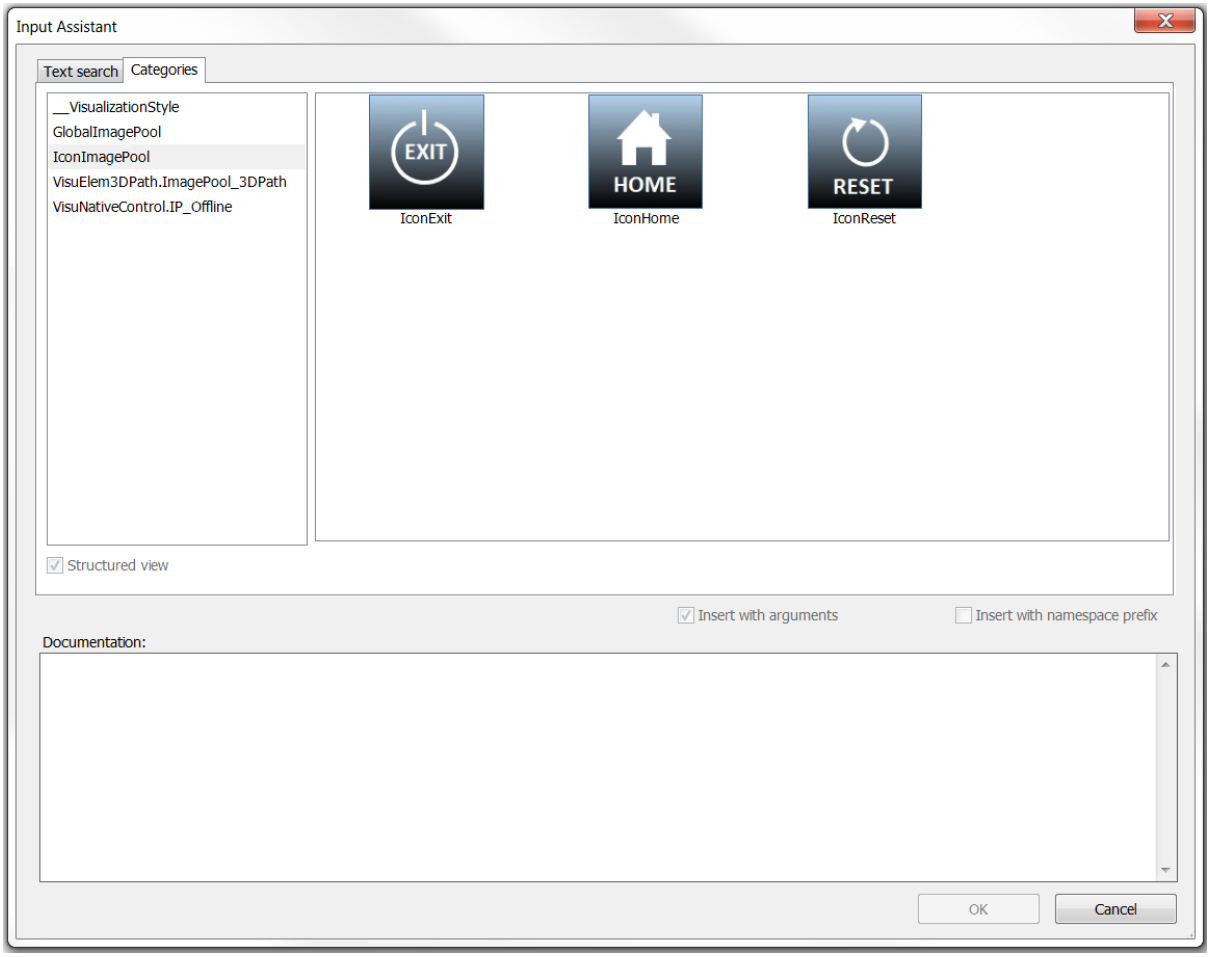
此对话框提供 2 个选项，用于从 PLC 项目中的图像池中选择图像：

- 文本搜索



在顶部的行编辑器中输入所需图像文件的名称或部分名称。所有与文本相匹配的图像文件名都会显示。此外，您还可以设置筛选器，将搜索范围缩小到特定的图像池。

- 类别



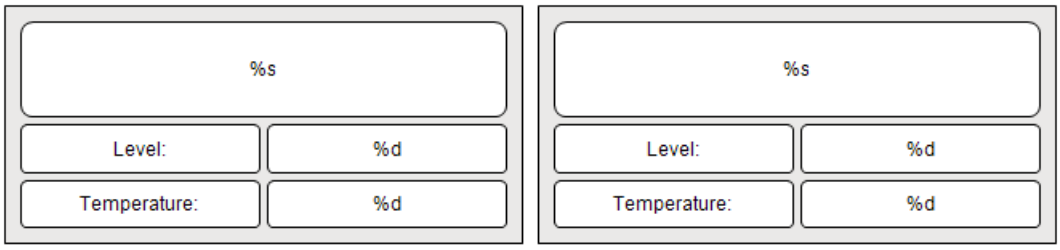
在项目中可用的所有图像池都会在左侧列中列出。在选择图像池后，在右侧会显示图像的预览。

使用 2 个选项之一选择图像，然后选择 OK（确定），关闭对话框。或者，您也可以双击所需的图像。在关闭对话框之后，图像将在“静态 ID [▶ 462]”属性中以 <name of image pool>.<name of image> 完全引用。

15.8.3.6 边框

使用边框元素

- 作为另一种可视化的引用 [▶ 559]
- 作为占位符的接口 [▶ 559]
- 用于在边框内的不同可视化页面之间切换 [▶ 559]



属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在 View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用 ... 按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

附加设置

截断	如果启用该选项与缩放类型“Unscaled”（未缩放），则仅会显示适合边框的可视化部分。
绘制边框	如果启用该选项，则引用的可视化会显示 1 个边框。
缩放类型	您可以在此处指定边框如何响应可视化大小的变化： • 各向同性 边框保持其比例。可独立更改可视化的高度和宽度比例。 • 各向异性 边框基于可视化的大小，因此可以独立更改引用可视化的高度和宽度。 • 未缩放 无论可视化大小如何，都会保持边框的原始大小。如果也启用选项“Truncate”（截断），则只显示适合的部分。 • 未缩放且可滚动 如果使用该选项，则会显示没有缩放的引用可视化。如果显示区域大于边框的窗口区域，则边框会提供滚动条，以便移动可视化的显示区域。如要使用变量设置滚动条的位置，可使用属性“Variable scroll position horizontal”（水平可变滚动位置）或“Variable scroll position vertical”（垂直可变滚动位置）。
停用背景图	如要优化可视化的性能，则可将边框元素中的非动画元素作为背景位图绘制。这可能会导致元素显示的顺序与预期不符。为了避免这种行为，可以禁用该功能。

滚动条设置

只有输入缩放类型“Unscaled and scrollable”（未缩放且可滚动）时，才会显示滚动条设置。强烈建议根据客户端的具体情况使用变量。在这种情况下，如果变量更改或者如果用鼠标移动滚动条，则这种变化只会影响相应客户端的边框。否则，所有客户端都会更新。

水平可变滚动位置	变量，包含水平滚动条的位置。
垂直可变滚动位置	变量，包含垂直滚动条的位置。

滚动位置的客户端特定变量：

```
PROGRAM MAIN
VAR
    aScrollPositionsHorizontal : ARRAY[0..20] OF INT;
    aScrollPositionsVertical : ARRAY[0..20] OF INT;
END_VAR
```

边框元素的指定属性：

水平可变滚动位置	MAIN.aScrollPositionsHorizontal[CURRENTCLIENTID]
垂直可变滚动位置	MAIN.aScrollPositionsVertical[CURRENTCLIENTID]

引用可视化

您可以在此处打开对话框“边框选择 [▶ 478]”，该对话框可用于选择要引用的可视化页面。在选择 1 个或多个可视化页面之后，它们会与占位符 [▶ 559]（如果有的话）一起列在下面。如果占位符更改，则会为所有实例自动打开对话框“更新边框参数 [▶ 478]”。

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

中心

在编辑数值时，相应的元素 ⊕ 也会同时在可视化编辑器 [▶ 344]中移动。

X	元素枢轴的水平位置（单位：像素）
Y	元素枢轴的垂直位置（单位：像素）

颜色

颜色根据由红/绿/蓝（RGB）组件组成的十六进制数进行定义。这 3 种颜色各有 256（0-255）个值，您可以在此处静态输入。从选择列表中或通过颜色选择对话框可以选择颜色，通过按钮 ... 可以打开颜色选择对话框。此外，还可以为每种颜色设置透明度级别（0：完全透明，255：完全不透明）。

颜色	为默认状态选择边框和填充颜色。如果颜色切换变量被定义为 FALSE，则元素处于默认状态。
报警颜色	为处于报警状态的元素选择边框和填充颜色。如果颜色切换变量被定义为 TRUE，则会触发此操作。

Windows CE 不支持透明度。

外观

线宽	以像素为单位定义边框线宽。0 的代码与 1 相同，并将线宽设置为 1 像素。如果不需要边框，则将线条类型设置为不可见。
线条类型	为轮廓定义以下 1 种线条类型： • 实线 • 虚线 • 点线 • 点划线 • 双点划线 • 不可见：轮廓不可见。

文本

这些属性可用于元素标签的静态定义。每个属性都可以包含 1 个格式化序列 [▶ 563]，例如 %s。在在线模式下，在“Text variables”（文本变量）中定义的变量内容可以替换序列。

文本	输入文本。它可用于标注元素。文本可以包含 1 个格式化序列，例如 %s。在“Text variables”（文本变量）中可以定义相应变量。 注意：也可以直接输入文本。如果在可视化编辑器中选择元素，则可以通过按空格键打开输入字段。
工具提示	输入文本。它可以用作元素的工具提示，只有在在线模式下将光标置于元素上时才会出现在可视化中。文本可以包含 1 个格式化序列，例如 %s。在“Text variables”（文本变量）中可以定义相应变量。

文本属性

这些属性可用于字体的静态定义。在“Font variables”（字体变量）类别中可以对字体进行动态定义。

水平对齐方式	通过选择以下各项来定义文本的水平对齐方式： • 居左 • 居中 • 居右
垂直对齐方式	通过选择以下各项来定义文本的垂直对齐方式： • 顶部 • 居中 • 底部
文本格式	定义因文本太长而无法在元素中完整显示时的显示方式： • 默认 - 文本延伸到元素之外。 • 换行 - 文本自动换行。 • 省略 - 尽可能地显示文本，然后用“...”截断.
字体：	通过从预定义字体中进行选择来定义字体： • 标准 • 页首标题 • 大标题 • 标题 • 说明 按 … 打开用户定义的字体属性的对话框。
字体颜色	定义元素的字体颜色。可以从选择列表中进行选择，也可以通过点击 … 时打开的对话框进行选择。

绝对移动

通过 1 个整数变量改变元素左上角的 x 和 y 位置（像素），从而可以移动元素。这里使用的是绝对坐标值。

运动 • X • Y	X：此处输入的整数变量可以定义元素左上角当前的 x 位置（单位：像素）。它可用于沿 x 方向移动元素。（正值会将元素从左至右移动）。 Y：此处输入的整数变量可以定义元素左上角当前的 y 位置（单位：像素）。它可用于沿 y 方向移动元素。（正值会将元素从上至下移动）。
旋转	此处输入的整数变量可以定义围绕旋转点旋转元素的角度（角度数）。 正值 = 顺时针方向 注意：与“internal rotation”（内部旋转）行为（见下文）相比，元素本身不会旋转。 点击元素，可显示旋转点 ⊕ 。按住鼠标按钮，即可移动它。
缩放	此处输入的整数变量可以定义当前的缩放因子（百分比）。元素大小可根据该值进行线性调整。该值隐式除以 1000，因此无需使用 REAL 变量来缩小元素。缩放总是指旋转点（中心）。点击元素，可显示旋转点 ⊕ 。按住鼠标按钮，即可移动它。

相对移动

元素可以相对于其固定位置移动。按照整数变量定义的值（像素），元素的左上边缘和右下边缘会沿 x 或 y 方向移动。与绝对移动相比，可以定义相对位置，即与原始位置的距离。此功能可用于更改元素的形状。正值会使水平边缘向下移动和/或使垂直边缘向右移动。

左上移动 • X • Y	• X：整数变量，其值表示左上角沿 x 方向移动的像素数。 • Y：整数变量，其值表示左上角沿 y 方向移动的像素数。
右下移动 • X • Y	• X：整数变量，其值表示右下角沿 x 方向移动的像素数。 • Y：整数变量，其值表示右下角沿 y 方向移动的像素数。

文本变量

您可以显示存储在变量中的文本。为此，首先要在“Texts”（文本）属性下定义的文本中添加 1 个格式化序列。然后，分配 1 个变量。在在线模式下，变量的内容可以替换格式化序列 [▶ 563]。

示例 %f：

在“Texts”（文本）属性中，输入“Result: %2.5f”。在“Text variables”（文本变量）属性中，输入“fValue”。该变量必须是已定义的 IEC 变量。如果 fValue = 12.1234567，则该元素将在在线模式下标注为“Result: 12.12345”。

文本变量	（默认数据类型的）变量，包含要显示的信息。类型必须与“Texts”（文本）设置中的格式化序列一致。
Tooltipvariable	字符串类型的变量，包含要显示的工具提示文本。“Texts”（文本）属性中的条目必须包含格式化序列。

动态文本

这些参数可用于定义源自文本列表 [▶ 127]的动态文本。例如，这可以实现语言更改 [▶ 563]。

动态定义文本的另一种可能性是通过字符串变量提供文本。（请参见“文本变量”类别）。

文本列表	在项目树中，作为字符串使用的文本列表的名称 示例：“TL_ErrorList”
Textindex	在文本列表中，作为字符串定义的文本的索引（ID）。它可以以静态方式直接指定，也可以作为字符串变量进行指定。
Tooltipindex	在文本列表中，作为字符串定义的工具提示文本的索引（ID）。它可以以静态方式直接指定，也可以作为字符串变量进行指定。

字体变量

这些变量可用于通过项目变量对元素文本进行动态字体定义。在“Text properties”（文本属性）下可以配置静态定义。

字体名称	指定 1 个字符串类型的变量，该变量包含用于标注元素的字体名称。（指定的名称，如在标准字体对话框中所示） 示例：MAIN.sFont (sFont := ‘Arial’ ;)
大小	INT 类型的变量，包含元素文本的大小（单位：像素），如在默认对话框“Font”（字体）中所示。 示例：MAIN.nHeight (nHeight := 16;)
标志	DWORD 类型的变量，用于通过下列标志值之一定义字体显示。通过将相关的标志值相加并指定总和可以实现组合定义。 • 1：斜体 • 2：粗体 • 4：下划线 • 8：删除 示例： MAIN.nFlag (nFlag := 6;) 文字显示为粗体并加下划线。
字符集	通过默认字体编号可以定义字体所使用的字符集。通过 DWORD 变量可以指定该数字（另请参见默认字体对话框中的定义）。
颜色	DWORD 类型的变量，用于定义元素文本的颜色。
文本对齐方式的标志	DWORD 类型的变量，用于通过下列标志值之一定义文本对齐方式。通过将相关的标志值相加并指定总和可以实现组合定义。 • 0：左上 • 1：水平居中 • 2：居右 • 4：垂直居中 • 8：居下 **示例** MAIN.nFlag (nFlag := 5;) 以水平和垂直居中的方式显示文本。

颜色变量

用于通过 DWORD 类型的项目变量动态定义元素颜色的颜色变量。颜色根据由红、绿、蓝（RGB）组件组成的十六进制数进行定义。此外，变量还可用于指定颜色的透明度（FF：完全不透明 - 00：完全透明）。DWORD 的结构如下：16#TTRRGGBB

十六进制数的结构与 TwinCAT 2 不同。在 TwinCAT 3 中，除了 RGB 比例之外，还可以用十六进制数来定义颜色的透明度。透明度由“16#”后的前 2 位数字表示。定义以“16#00”开头的颜色不可见，因为它们是完全透明的。

示例：

```
nFillColor := 16#FF8FE03F;
```

- FF：透明度（完全不透明）
- 8F：红色
- E0：绿色
- 3F：蓝色

颜色更改	Boolean 变量，用于控制元素颜色在“正常状态”（变量 = FALSE）和“报警状态”（变量 = TRUE）之间的切换。
正常状态	DWORD 类型的变量，用于定义元素颜色。它会覆盖当前在“Colors”（颜色）中定义的值。如果在“Color change”（颜色更改）中定义的变量为 FALSE，则会使用项目变量中的值。
报警状态	DWORD 类型的变量，用于定义报警状态下的元素颜色。它会覆盖当前在“Colors”（颜色）中定义的值。如果在“Color change”（颜色更改）中定义的变量为 TRUE，则会使用项目变量中的值。

Windows CE 不支持透明度。

外观变量

用于轮廓外观和元素填充的动态定义。在“Appearance”（外观）下可以指定静态定义。

线宽	整数类型的变量，用于定义元素的线宽（单位：像素）。它会覆盖在“Appearance”（外观）下指定的固定值。
填充类型	DWORD 类型的变量，用于定义元素填充。可以显示或忽略在颜色变量下指定的颜色： • 变量值 = 0：填充 • 变量值 > 0：不可见，即不显示填充
线条类型	DWORD 类型的变量，用于定义轮廓。以下值与以下线条类型相对应： • 0：实线 • 1：虚线 • 2：点线 • 3：点划线 • 4：双点划线 • 8：不可见。也就是说，显示的元素没有轮廓。

切换变量

该属性可用于在边框的可视化之间切换。

变量	整数变量，其值包含要显示的可视化的 ID。可视化的 ID 由边框选择 [▶ 563]中分配的可视化列表中的顺序决定。例如，该列表中第 1 个条目的 ID 是 0，第 2 个条目的 ID 是 1。 使用变量可以切换分配给边框的可视化。可视化的值（ID）由该元素在对话框“边框可视化的配置”中分配的可视化列表中的顺序决定。对于整数值，该列表中第 1 个条目的值是 0，第 2 个条目的值是 1，以此类推。

状态变量

这些是对元素在在线模式下的可用性的动态定义。

不可见	指定 1 个 Boolean 变量。如果返回 TRUE，则元素在在线模式下不可见。
输入已禁用	指定 1 个 Boolean 变量。如果返回 TRUE，则元素的输入无效。此外，元素本身在可视化中显示为灰色，表示用户无法输入。 如果可视化使用用户管理 [▶ 360]，则访问权限为“only visible”（仅可见）的用户组的元素会显示为灰色。

输入配置

您可以在此处定义当用户在在线模式下对元素进行输入时应执行的相应动作。只要没有定义后续动作，“Configure…”（配置...）就会出现在 Properties（属性）字段中。点击“Configure…”（配置...），打开输入配置 [▶ 371]，您可以在此处分配后续动作。您可以将许多不同的后续动作分配给每个输入动作。

元素可以使用以下输入事件：

- OnDialogClosed
- OnMouseClick
- OnMouseDown
- OnMouseEnter
- OnMouseLeave
- OnMouseMove
- OnMouseUp

OnDialogClosed	如果在可视化中关闭 1 个为用户输入而打开的对话框，则会触发该事件。 注意：此属性不仅限于为其配置的元素，而是适用于整个可视化。因此，它不会对每个对话框的关闭动作做出响应。目前还没有办法为整个可视化定义这样的属性。因此，必须将它分配给其中 1 个元素。
OnMouseClick	当光标指向 1 个元素并在该项目上执行 1 次完整的鼠标点击（按下并松开鼠标按钮）时，就会触发该鼠标事件。
OnMouseDown	在光标指向 1 个元素的同时按下鼠标按钮时，就会触发该鼠标事件。这与您在可视化上再次松开鼠标按钮的位置无关。
OnMouseEnter	在将光标拖到元素上时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseLeave	在光标离开元素时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseMove	当光标在元素内移动时，就会触发该鼠标事件。这与按下或松开鼠标按钮无关。
OnMouseUp	在元素上松开鼠标按钮时，就会发生该鼠标事件。在此之前，已在元素之外按下鼠标按钮。

输入配置 - 热键

热键可用于定义 1 个按键或按键组合，并将其与后续动作（例如，MouseDown、MouseUp）建立关联，当按键事件（KeyDown、KeyUp）发生时，就会执行后续动作。默认情况下，在 KeyDown（按下按键）时会执行 MouseDown 动作，在 KeyUp（松开按键）时会执行 MouseUp 动作。如果要同时通过鼠标动作和键盘输入来操作可视化，这将非常有用，因为输入动作只需配置 1 次。在可视化的热键配置 [▶ 358]中也可以对元素的按键配置进行管理。任何更改始终都会在此编辑器和元素属性编辑器之间同步进行。

按键	分配按键。选择列表包含当前支持的所有按键，例如 M。
事件	如果使用按键或按键和修饰符，则定义要执行的事件。选择列表中可用的可能值： • 无动作 • 在按下按键时，MouseDown 动作 • 在松开按键时，MouseUp 动作 • 在按下松开按键时，MouseDown/MouseUp 动作
移位	如果启用该选项，则按键必须与 Shift 键组合使用。
控制	如果启用该选项，则按键必须与 Ctrl 键组合使用。
Alt	如果启用该选项，则按键必须与 Alt 键组合使用。

输入配置 - 按键

“按键”可用于指定根据事件“按键”的鼠标行为设置 Boolean 项目变量的值。

变量	Boolean 变量，如果在光标指向元素的同时按下鼠标按钮，则该变量的值为 TRUE。当松开鼠标按钮或光标离开元素时，该值又会变为 FALSE。
FALSE 按键	如果启用该选项，则相应变量的上述按键行为将会反转。也就是说，当按下鼠标按钮时，变量值会被设置为 FALSE。当松开鼠标按钮时，变量值会被设置为 TRUE。
如果鼠标受限，则启动输入	如果启用该选项，只要光标在元素区域内，变量值就会如“Variables”（变量）中所述发挥作用。如果在按下鼠标按钮的同时光标离开元素区域，则该变量会被设置为 FALSE。不过，如果在光标返回元素区域时没有松开鼠标按钮，则会将它再次自动设置为 TRUE。换句话说，只要按下鼠标按钮，即使光标离开元素区域，系统也会考虑到鼠标“受限”的情况。

输入配置 - 切换

“切换”可用于指定根据鼠标行为针对该事件设置 Boolean 项目变量的值。

变量	Boolean 变量，每当鼠标点击元素时，该变量值会在 TRUE 和 FALSE 之间切换。如果在按下鼠标按钮的同时光标远离元素，则不会发生切换。可用于取消切换动作。
如果鼠标受限，则启动松开	如果启用该选项，只要光标在元素区域内，变量值就会如“Variables”（变量）中所述发挥作用。如果在按下鼠标按钮的同时光标离开元素区域，然后松开鼠标按钮，则仍会切换变量。换句话说，只要按下鼠标按钮，即使光标离开元素区域，系统也会考虑到鼠标“受限”的情况。

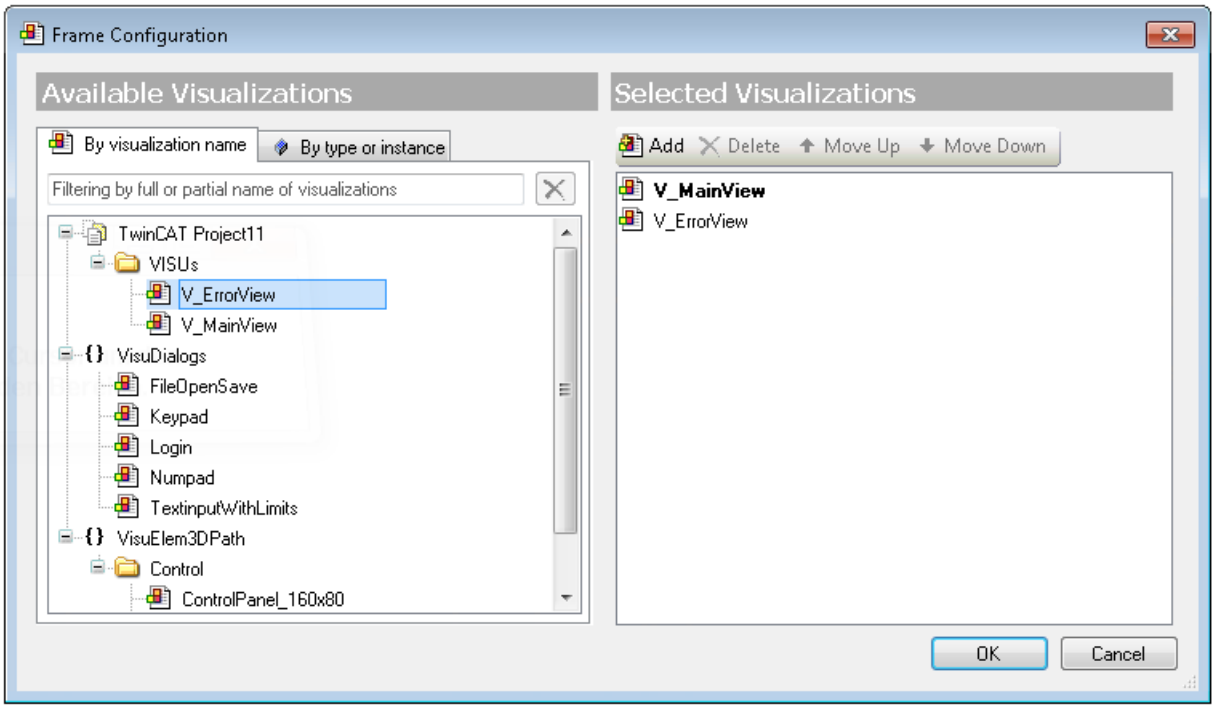
访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

15.8.3.6.1 边框选择

此对话框可用于配置边框 [▶ 470]元素。它可用于选择通过边框元素引用的 1 个或多个可视化页面 [▶ 367]。

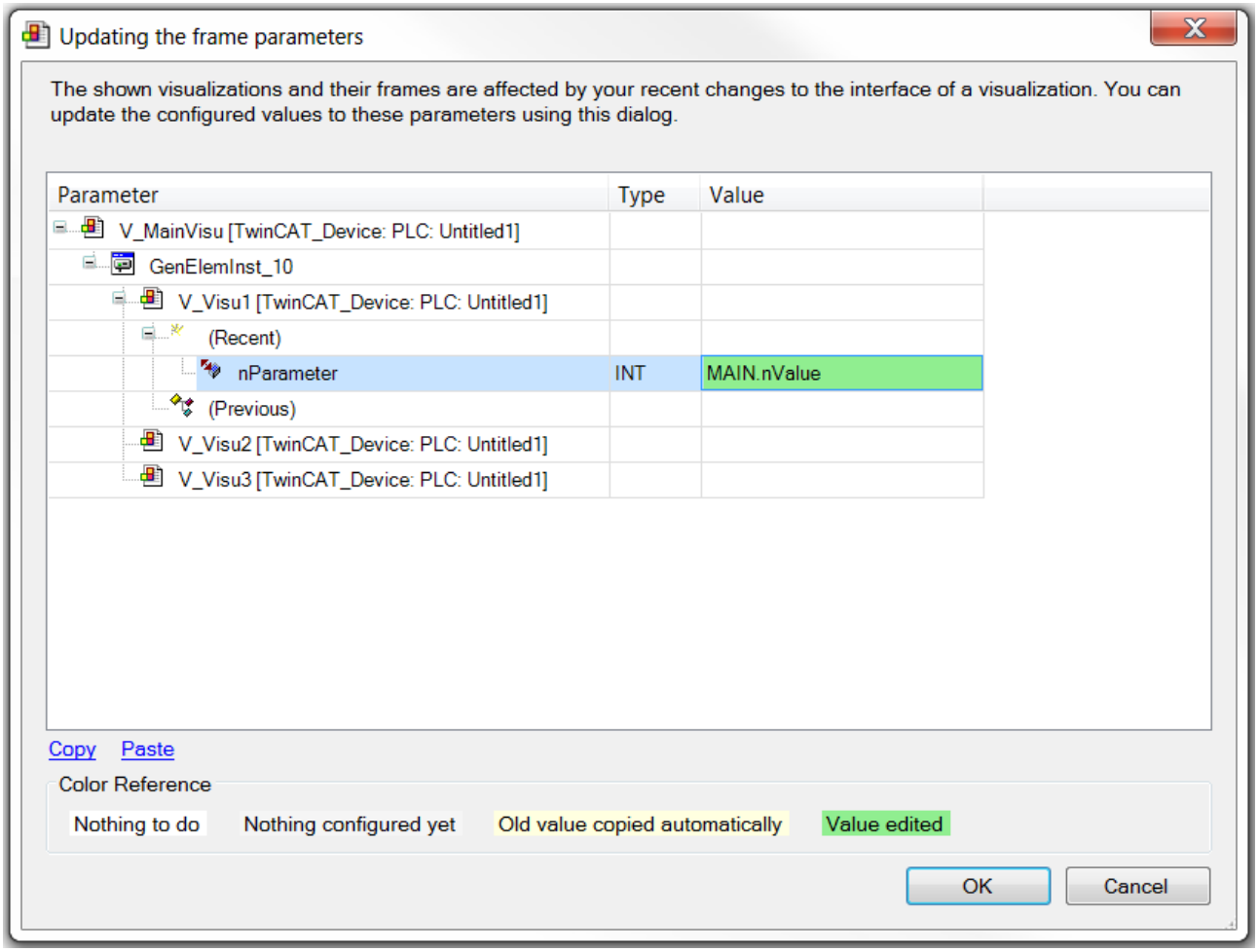


在对话框的左侧会显示在项目中可用的可视化页面或对象。选择要在边框中引用的内容。双击或使用“Add”（添加）按钮可以将它们添加到对话框右侧的所选可视化页面的列表中。双击或通过“Delete”（删除）按钮可以从列表中删除所选的可视化。可实现可视化页面的多项选择，从而进行添加或删除。使用“Move Up”（向上移动）和“Move Down”（向下移动）按钮可以更改列表内的顺序。

所选可视化对象从上到下的顺序由自动生成的隐式可视化索引号决定。为顶部的对象分配“0”，然后为下面的对象分配“1”、“2”等。在为另一个元素配置切换功能 [▶ 379]时，需要使用索引号。首先显示的是索引为“0”的可视化页面。

15.8.3.6.2 更新边框参数

如果在接口编辑器 [▶ 345]中更改引用的可视化页面 [▶ 367]的参数，则会出现该对话框。编辑器可用于添加或更新边框参数 [▶ 348]。



在接口编辑器 [▶ 345]下，介绍了使用该对话框的方法。

15.8.4 灯/开关/位图

15.8.4.1 图像切换器

元素图像切换器可显示由 3 个引用图像 [▶ 480]组成的图像。如果在可视化运行期间发生鼠标动作，则显示的图像会改变。通过元素行为可以配置鼠标动作的后果。有关在可视化中集成图像的选项的更多信息，请参见“应用程序提示“中的”图像 [▶ 567]“部分。

属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用 ... 按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

辅助设置

变量	Boolean 变量的赋值，其状态随用户输入而变化。根据元素行为的不同，只要按下鼠标按钮（图像按钮），该变量就是 TRUE；或者每次点击鼠标时（图像切换器），该值都会改变。

图像设置

如要使图像名称的条目保持统一，可指定图像池的名称作为前缀。如果通过 GlobalImagePool 管理图像，则无需这样做，因为在任何情况下都会首先搜索它。

图像开启	图像池中的位图的 ID。图像显示为开启状态。
图像关闭	图像池中的位图的 ID。图像显示为关闭状态。
图像按下	图像池中的位图的 ID。在运行时，只要元素接收到“按下鼠标右键”的鼠标动作，可视化就会显示引用的图像。 要求：元素行为是图像切换器。
透明度	如果启用该选项，则会为透明度颜色定义的部分绘制完全透明的图像。
透明度颜色	您可以在此处设置在图像中完全透明显示的颜色。 示例：如果图像背景是白色且透明度颜色也是白色，则会绘制完全透明的背景。 要求：透明度已启用。
缩放类型	您可以在此处指定元素如何响应边框大小的变化： • 各向同性：图像的高度和宽度随框架变化。不过，它们会保留原始比例。 • 各向异性：图像填满整个边框，与比例无关。 • 未缩放：如果边框大小更改，则指定图像大小并保持不变。
水平对齐方式	只有当属性编辑器的“Expert”（专家）选项被激活，且图像缩放类型为“isotropic”（各向同性）时才可用。如果在缩放边框内使用可视化，则它可用于明确保持与另一个元素的水平对齐（如基本可视化中所定义）。见上文：“各向同性”设置 • 居左 • 居中 • 居右
垂直对齐方式	只有当属性编辑器的“Expert”（专家）选项被激活，且图像缩放类型为“isotropic”（各向同性）时才可用。如果在缩放边框内使用可视化，则它可用于明确保持与另一个元素的垂直对齐（如基本可视化中所定义）。见上文：“各向同性”设置 • 顶部 • 居中 • 底部

元素行为	您可以在此处选择元素行为： • 图像切换器：每次点击鼠标时，元素的状态以及图像和变量都会随之切换。 • 图像按钮：只要按下鼠标按钮，就会显示“Image on”（图像开启），该变量就是 TRUE。在这种状态下，不会使用在“Image pressed”（图像按下）中引用的图像。
FALSE 按键	如果启用该选项，则在按下鼠标按钮后，该变量将被设置为 FALSE。否则，变量将被设置为 TRUE。 要求：在元素行为中设置图像切换器。

文本

工具提示	您可以在此处设置要用作元素的工具提示的文本。它只会在运行时出现在可视化中。

状态变量

这些是对元素在在线模式下的可用性的动态定义。

不可见	指定 1 个 Boolean 变量。如果返回 TRUE，则元素在在线模式下不可见。
输入已禁用	指定 1 个 Boolean 变量。如果返回 TRUE，则元素的输入无效。此外，元素本身在可视化中显示为灰色，表示用户无法输入。 如果可视化使用用户管理 [▶ 360]，则访问权限为“only visible”（仅可见）的用户组的元素会显示为灰色。

访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

15.8.4.2 灯

如果设置已分配的变量，则灯亮。在灯设置 [▶ 483]中可以更改灯的颜色。

属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在 View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用 ... 按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

辅助设置

变量	Boolean 变量，其值可切换灯的状态。该值为 TRUE 时，灯会打开并闪亮。

图像设置

缩放类型	您可以在此处指定元素如何响应边框大小的变化： • 各向同性：图像的高度和宽度随框架变化。不过，它们会保留原始比例。 • 各向异性：图像填满整个边框，与比例无关。 • 未缩放：如果边框大小更改，则指定图像大小并保持不变。
水平对齐方式	只有当属性编辑器的“Expert”（专家）选项被激活，且图像缩放类型为“isotropic”（各向同性）时才可用。如果在缩放边框内使用可视化，则它可用于明确保持与另一个元素的水平对齐（如基本可视化中所定义）。见上文：“各向同性”设置 • 居左 • 居中 • 居右
垂直对齐方式	只有当属性编辑器的“Expert”（专家）选项被激活，且图像缩放类型为“isotropic”（各向同性）时才可用。如果在缩放边框内使用可视化，则它可用于明确保持与另一个元素的垂直对齐（如基本可视化中所定义）。见上文：“各向同性”设置 • 顶部 • 居中 • 底部

文本

工具提示	您可以在此处设置要用作元素的工具提示的文本。它只会在运行时出现在可视化中。

背景

图像	您可以在此处选择灯的颜色： • 黄色 • 红色 • 绿色 • 蓝色 • 灰色

状态变量

这些是对元素在在线模式下的可用性的动态定义。

不可见	指定 1 个 Boolean 变量。如果返回 TRUE，则元素在在线模式下不可见。
输入已禁用	指定 1 个 Boolean 变量。如果返回 TRUE，则元素的输入无效。此外，元素本身在可视化中显示为灰色，表示用户无法输入。 如果可视化使用用户管理 [▶ 360]，则访问权限为“only visible”（仅可见）的用户组的元素会显示为灰色。

访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

15.8.4.3 拨码开关、电源开关、按压式开关、LED 指示灯按压式开关、摇杆开关

开关可用于设置 Boolean 变量的值。

属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在 View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用  按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

辅助设置

变量	Boolean 变量，其值随用户输入而变化。根据元素行为的不同，只要按下鼠标按钮（瞬时），该变量就是 TRUE；或者每次点击鼠标时（切换），该值都会改变。

图像设置

缩放类型	您可以在此处指定元素如何响应边框大小的变化： • 各向同性：图像的高度和宽度随框架变化。不过，它们会保留原始比例。 • 各向异性：图像填满整个边框，与比例无关。 • 未缩放：如果边框大小更改，则指定图像大小并保持不变。
水平对齐方式	只有当属性编辑器的“Expert”（专家）选项被激活，且图像缩放类型为“isotropic”（各向同性）时才可用。如果在缩放边框内使用可视化，则它可用于明确保持与另一个元素的水平对齐（如基本可视化中所定义）。见上文：“各向同性”设置 • 居左 • 居中 • 居右
垂直对齐方式	只有当属性编辑器的“Expert”（专家）选项被激活，且图像缩放类型为“isotropic”（各向同性）时才可用。如果在缩放边框内使用可视化，则它可用于明确保持与另一个元素的垂直对齐（如基本可视化中所定义）。见上文：“各向同性”设置 • 顶部 • 居中 • 底部

状态变量

这些是对元素在在线模式下的可用性的动态定义。

不可见	指定 1 个 Boolean 变量。如果返回 TRUE，则元素在在线模式下不可见。
输入已禁用	指定 1 个 Boolean 变量。如果返回 TRUE，则元素的输入无效。此外，元素本身在可视化中显示为灰色，表示用户无法输入。 如果可视化使用用户管理 [▶ 360]，则访问权限为“only visible”（仅可见）的用户组的元素会显示为灰色。

文本

工具提示	您可以在此处设置要用作元素的工具提示的文本。它只会在运行时出现在可视化中。

背景

图像	您可以在此处选择灯的颜色： • 黄色 • 红色 • 绿色 • 蓝色 • 灰色

访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

15.8.4.4 旋转开关

旋转开关可用于设置 Boolean 变量的值。

属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用  按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

辅助设置

变量	Boolean 变量，其值随用户输入而变化。根据元素行为的不同，只要按下鼠标按钮（瞬时），该变量就是 TRUE；或者每次点击鼠标时（切换），该值都会改变。

图像设置

缩放类型	您可以在此处指定元素如何响应边框大小的变化： • 各向同性：图像的高度和宽度随框架变化。不过，它们会保留原始比例。 • 各向异性：图像填满整个边框，与比例无关。 • 未缩放：如果边框大小更改，则指定图像大小并保持不变。
水平对齐方式	只有当属性编辑器的“Expert”（专家）选项被激活，且图像缩放类型为“isotropic”（各向同性）时才可用。如果在缩放边框内使用可视化，则它可用于明确保持与另一个元素的水平对齐（如基本可视化中所定义）。见上文：“各向同性”设置 • 居左 • 居中 • 居右
垂直对齐方式	只有当属性编辑器的“Expert”（专家）选项被激活，且图像缩放类型为“isotropic”（各向同性）时才可用。如果在缩放边框内使用可视化，则它可用于明确保持与另一个元素的垂直对齐（如基本可视化中所定义）。见上文：“各向同性”设置 • 顶部 • 居中 • 底部

元素行为	您可以在此处选择元素行为： • 图像切换器：每次点击鼠标时，元素的状态以及图像和变量都会随之切换。 • 图像按钮：只要按下鼠标按钮，就会显示“Image on”（图像开启），该变量就是 TRUE。在这种状态下，不会使用在“Image pressed”（图像按下）中引用的图像。
FALSE 按键	如果启用该选项，则在按下鼠标按钮后，该变量将被设置为 FALSE。否则，变量将被设置为 TRUE。 要求：在元素行为中设置图像切换器。
对齐方式	• 顶部：开关指向上方。 • 侧面：开关指向侧面。
颜色更改	如果未启用该选项，则开关在打开时会亮起。 如果未启用该选项，即使开关打开，它也不会亮起。

文本

工具提示	您可以在此处设置要用作元素的工具提示的文本。它只会在运行时出现在可视化中。

状态变量

这些是对元素在在线模式下的可用性的动态定义。

不可见	指定 1 个 Boolean 变量。如果返回 TRUE，则元素在在线模式下不可见。
输入已禁用	指定 1 个 Boolean 变量。如果返回 TRUE，则元素的输入无效。此外，元素本身在可视化中显示为灰色，表示用户无法输入。 如果可视化使用用户管理 [▶ 360]，则访问权限为“only visible”（仅可见）的用户组的元素会显示为灰色。

背景

图像	您可以在此处选择灯的颜色： • 黄色 • 红色 • 绿色 • 蓝色 • 灰色

访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

15.8.5 测量控制

15.8.5.1 条形显示

该元素可用于为可视化添加条形显示。条形显示采用预定义设计，背景颜色 [▶ 489]可以进行更改。可选的是，用户指定的背景图像可以替换该设计。显示方向可从水平改为 [▶ 489]垂直，而且，可将条形细分为多个颜色区域 [▶ 490]。

属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在 View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用  按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

辅助设置

变量	数值变量，其值显示为条形长度。

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

背景

如果没有使用用户指定的背景图像，则可使用以下属性：

背景颜色	为条形图选择 1 种颜色： • 黄色 • 红色 • 绿色 • 蓝色 • 灰色

如果使用用户指定的背景图像，则可使用以下属性：

图像	您可以在此处通过指定图像文件的名称或其 ID，从图像池中分配图像。
透明度颜色	对于具有透明背景的图像，您可以选择 1 种要显示为透明的颜色。 ... 按钮可以打开颜色选择对话框。

条形

图表类型	下拉列表提供了用于放置条形和标度的各种选项： • 条形旁的标度 • 条形中的标度 • 标度中的条形 • 无标度
对齐方式	条形可以水平或垂直对齐。对齐方式由宽度和高度的比例决定，在此处无法编辑。在可视化编辑器 [▶ 344]中，通过使用鼠标“抓取”元素的 1 个角点并以水平或垂直的方式拖动它，可以对它进行更改。
移动的方向	如果对齐方式为水平，则有以下选择： • 从左至右 • 从右至左 如果对齐方式为垂直，则有以下选择： • 从下至上 • 从上至下

标度

标度开始	标度下限值
标度结束	标度上限值
主标度	粗标度的 2 条线之间的距离
子标度	细标度的 2 条线之间的距离。如果不需要对粗标度进行进一步细分，则可将该值设置为 0。
标度线宽	标度线的宽度（单位：像素）
标度颜色	通过选择列表或标准颜色选择对话框中的 ... 按钮可以设置颜色。
3D 标度	如果启用该选项，则会以三维方式显示标度。
元素边框	如果启用该选项，则会在条形显示周围绘制 1 个边框。

标签

单位	输入的文本可用作元素标签。例如，它在标度中心下方显示，可用于指定标度的单位。
字体	您可以在此处为单位和标度设置字体： • 标准 • 页首标题 • 大型 • 标题 • 说明 点击 … 按钮，打开用户定义的字体属性设置的对话框。
标度格式（C 语言语法）	使用 C 语言语法指定标度标签的格式。例如，在该字段中输入字符串“%3.2f s”，标度标签就会显示 3 位数字，其中 2 位是小数位，后面跟字母“s”。
标签的最大文本宽度	指定标度标签的最大宽度的值。该值通常会自动进行正确设置。只有在自动调整无法达到要求的结果时，才会使用该设置选项。
标签的文本高度	指定标度标签的高度的值。该值通常会自动进行正确设置。只有在自动调整无法达到要求的结果时，才会使用该设置选项。
字体颜色	通过选择列表或标准颜色选择对话框中的 … 按钮可以设置标签的颜色。

定位

水平移位	标度或条形与水平元素边框之间的距离（单位：像素）
垂直移位	标度或条形与垂直元素方框之间的距离（单位：像素）
水平缩放	在水平方向上增加（负值）或减少（正值）标度或条形的因子
垂直缩放	在垂直方向上增加（负值）或减少（正值）标度或条形的因子

颜色

颜色根据由红/绿/蓝（RGB）组件组成的十六进制数进行定义。这 3 种颜色各有 256（0-255）个值，您可以在此处静态输入。从选择列表中或通过颜色选择对话框可以选择颜色，通过按钮 … 可以打开颜色选择对话框。此外，还可以为每种颜色设置透明度级别（0：完全透明，255：完全不透明）。

图形颜色	条形颜色
条形背景	如果启用该选项，条形显示的背景颜色将更改为黑色。否则，该颜色为白色。
边框颜色	如果勾选元素边框复选框，则为条形显示周围边框的颜色
使用颜色区域	如果启用该选项，则会显示在颜色区域下定义的颜色区域。
切换整体颜色	如果启用该选项，那么，当变量值低于标度的起始值或高于标度的终止值时，整个条形就会切换颜色。
为条形使用颜色渐变	如果启用该选项，则条形会显示颜色渐变。
颜色区域标记	您可以在此处选择颜色区域指向哪个方向。可提供下列设置： • 无标记 • 向前标记 • 向后标识
颜色区域 • 区域 ◦ [<n>]	点击 Create new 按钮，生成新的颜色区域。为每个颜色区域创建 1 个区域，其中涵盖相应的设置。 [<n>]：数字表示该区域的索引。点击“Delete”（删除），可删除相应的颜色区域及其设置。

区域 [<n>]

区域开始	颜色范围的开始。它必须在规定的标度 [▶ 489]范围内。
区域结束	颜色范围的结束。它必须在规定的标度 [▶ 489]范围内。
颜色	条形区域的颜色

Windows CE 不支持颜色渐变和透明度。

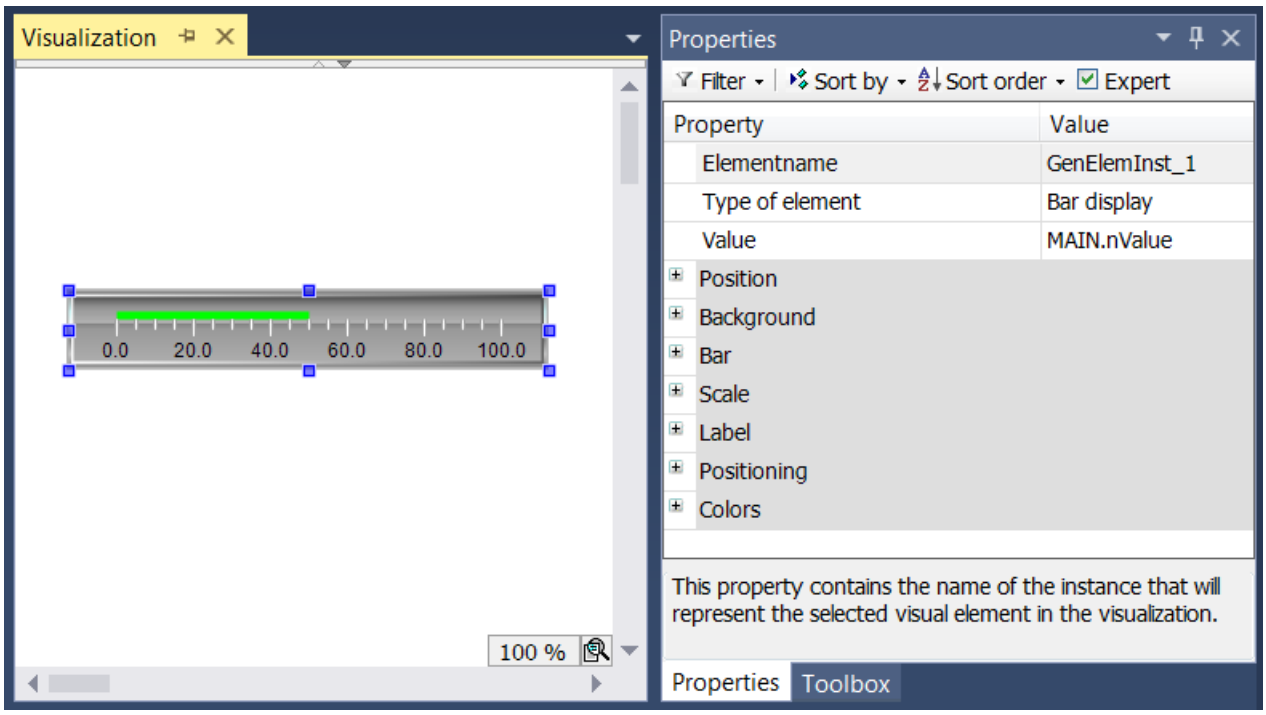
访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

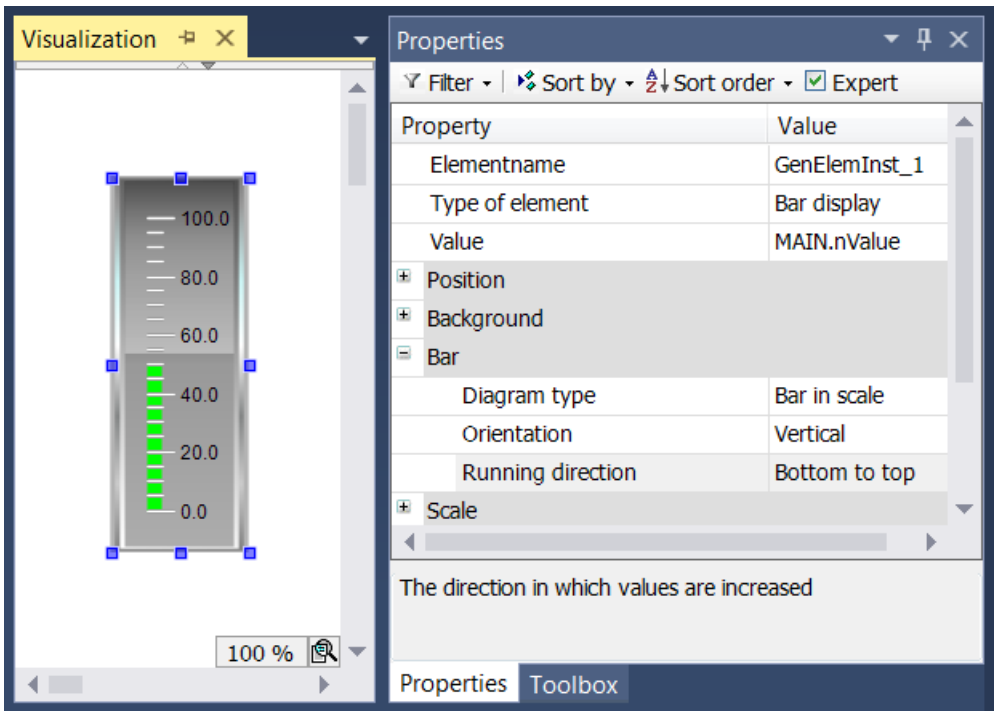
15.8.5.1.1 条形显示的配置

下一部分将根据 1 个示例解释指针式仪表的配置。



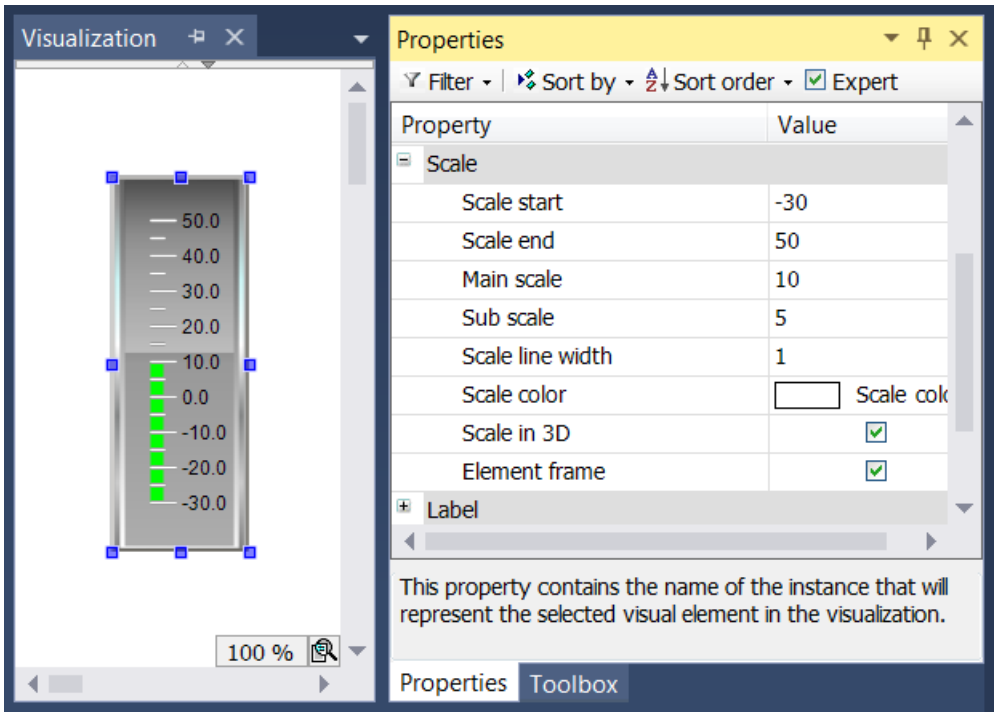
在“Bar display”（条形显示）的元素属性中必须指定要关联的输入变量（在本示例中为名称为“nValue”的变量）。在点击进入“Variable”（变量）属性的输入字段之后，即可使用 [...] 按钮，该按钮可用于浏览项目内部的变量。

在“Bar”（条形）部分中可以设置条形相对于标度的方向和位置。

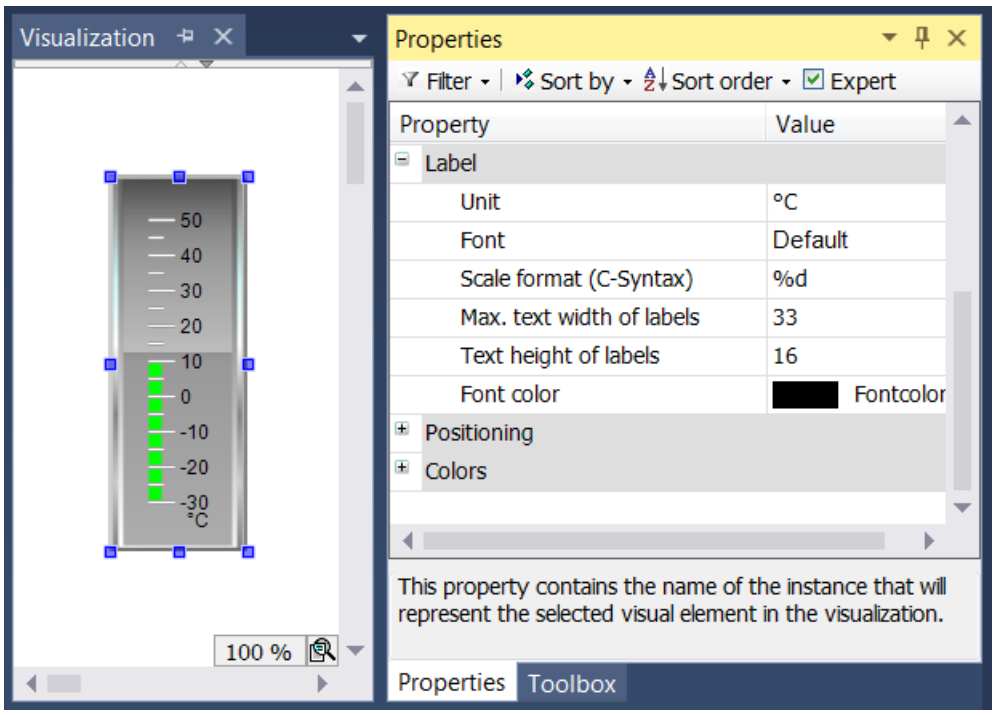


在本示例中，通过将长宽比从宽度改为高度，可将条形的方向从“Horizontal”（水平）改为“Vertical”（垂直）。该设置还可更改有关“Running direction”（运行方向）的可能信息。现在，您可以选择“bottom to top”（从下至上）或“top to bottom”（从上至下），而不是“left to right”从左至右“或”right to left“（从右至左）。通过设置”Diagram type“（图表类型），可以确定条形相对于标度的位置。

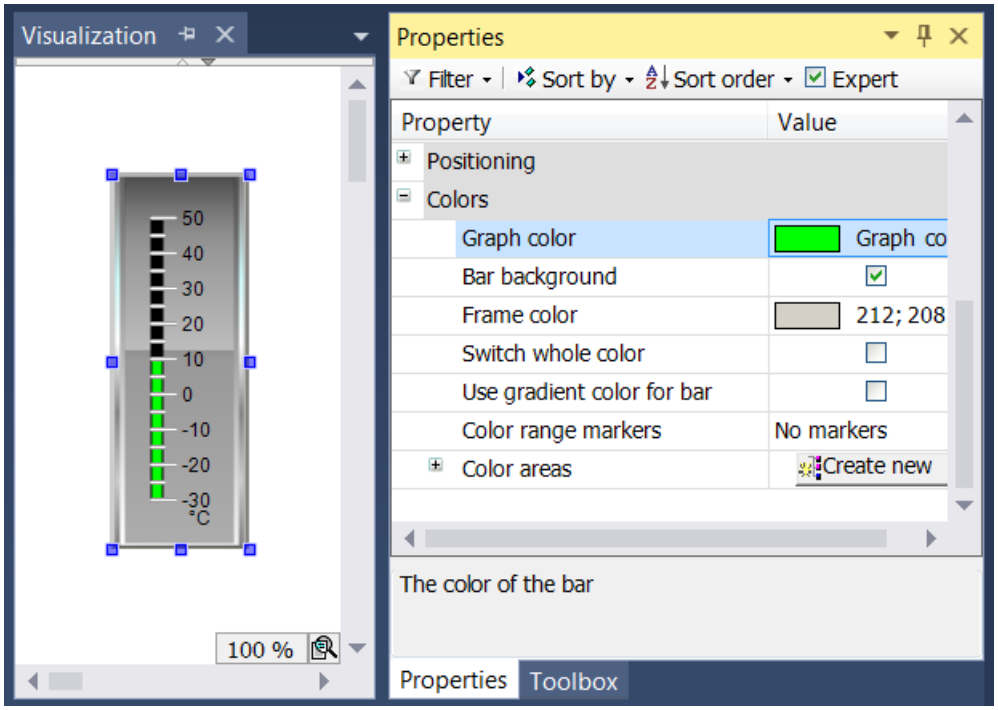
”Scale“标度部分用于设置标度范围、主标度和子标度。



标度范围受”Scale start“（标度开始）和”Scale end“（标度结束）中指定值的限制。”Scale start“（标度开始）的值必须小于”Scale end“（标度结束）的值。“Main scale”（主标度）和“Sub scale”（子标度）的值可被设置为 0，以禁用标度线的显示。如果主标度的值被设置为 0，则无论将子标度的值设置为多少，都不会绘制标度线。如果子标度的值被设置为 0，则只会绘制主标度的标度线。此时已勾选“Element frame”（元素边框）复选框，因为在本示例中，我们希望在元素周围绘制 1 个边框。



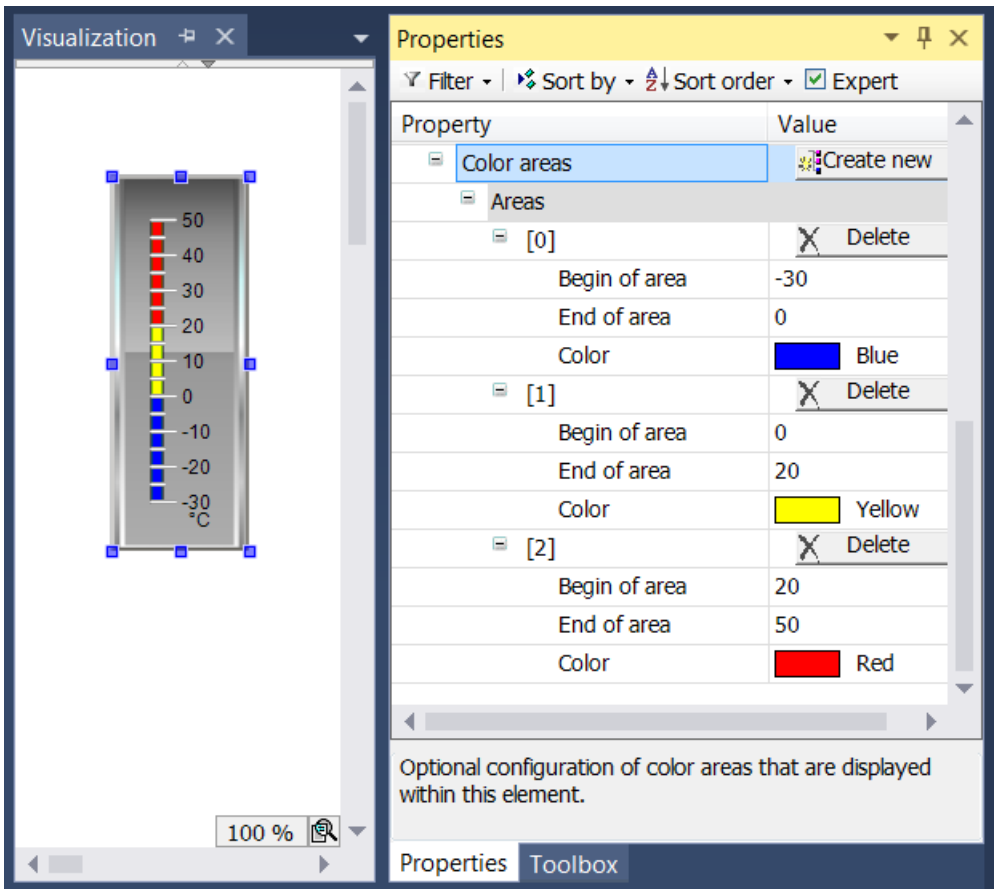
“Label”（标签）部分可用于指定条形标签属性。“Unit”（单位）条目可用于指定单位。它在条形下方居中显示。在选择合适的字体和字体颜色之后，您可以调整标签格式。务必按照 C 语言语法指定数值。“%d”可用于整数值，“%.Xf”可用于浮点数，其中“X”应被替换为所需的小数位数。



最后，在“Colors”（颜色）部分中可以指定元素颜色。首先，在“Graph color”（图形颜色）下可以指定条形本身的颜色。默认情况下，不会绘制条形背景。这是当前条形未填充的条形线部分。如果勾选“Bar background”（条形背景）复选框，则条形的未填充部分将显示为黑色。

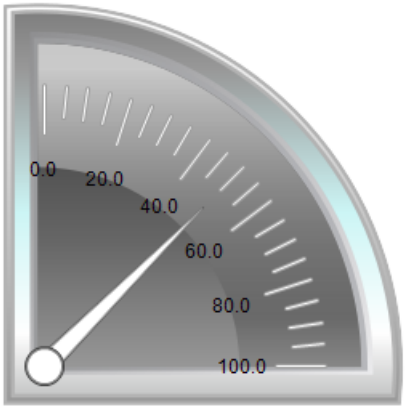
在“Color areas”（颜色区域）子部分中，可将标度细分为多个子区域。使用 Create new 按钮创建颜色区域，可为每个子区域分配 1 个特定的颜色。颜色区域按升序编号。在元素属性中，可为每个颜色区域分配其自己的输入字段。

在“Begin of area”（区域开始）和“End of area”（区域结束）下可以指定子区域的界限。通过下拉菜单可以选择颜色。在创建颜色区域之后，点击相应的 Delete 按钮，即可将其删除。



15.8.5.2 指针式仪表 90°

例如，指针式仪表可用于在可视化中添加转数计。箭头根据分配变量的值进行定位。元素采用预先配置的设计，背景颜色 [▶ 495]可以进行设置。可选的是，用户指定的背景图像 [▶ 495]可以替换该设计。可将标度细分为多个颜色区域 [▶ 497]。有关配置示例，请参见“指针式仪表的配置 [▶ 498]”部分。



属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用 ... 按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

辅助设置

值	数值变量，其值显示为箭头的偏移量。

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

背景

如果没有使用用户指定的背景图像，则可使用以下属性：

背景颜色	为条形图选择 1 种颜色： • 黄色 • 红色 • 绿色 • 蓝色 • 灰色

如果使用用户指定的背景图像，则可使用以下属性：

图像	您可以在此处通过指定图像文件的名称或其 ID，从图像池中分配图像。
透明度颜色	对于具有透明背景的图像，您可以选择 1 种要显示为透明的颜色。 ... 按钮可以打开颜色选择对话框。

箭头

箭头类型	许多不同的箭头类型可供选择： • 正常箭头 • 细箭头 • 宽箭头 • 细指针 • 3D 细箭头 • 3D 细指针
颜色	显示箭头的颜色
角度范围	您可以在此处选择 1 个 90° 部分： • 右上角 • 左上角 • 左下角 • 右下角
附加箭头	如果启用该选项，则会在标度上指针的对面显示 1 个附加箭头。

标度

子标度位置	在标度环的外半径或内半径处可以显示子标度： • 外部 • 内部
标度类型	标度可以显示为： • 线 • 点 • 正方形
标度开始	标度下限值
标度结束	标度上限值
主标度	粗标度的 2 条线之间的距离
子标度	细标度的 2 条线之间的距离。如果不需要对粗标度进行进一步细分，则可将该值设置为 0。
标度线宽	标度线的宽度（单位：像素）
标度颜色	通过选择列表或标准颜色选择对话框中的  按钮可以设置颜色。
3D 标度	如果启用该选项，则会以三维方式显示标度。
显示标度	如果启用该选项，则会显示标度。
边框内部	如果启用该选项，则会使用内部边框绘制标度环。默认情况下，已禁用该选项。
边框外部	如果启用该选项，则会使用外部边框绘制标度环。默认情况下，已禁用该选项。

标签

标签	您可以在此处指定标度值是位于标度的外部还是内部。
单位	输入的文本可用作元素标签。例如，它在标度中心下方显示，可用于指定标度的单位。
字体	您可以在此处为单位和标度设置字体： • 标准 • 页首标题 • 大型 • 标题 • 说明 点击 [...] 按钮，打开用户定义的字体属性设置的对话框。
标度格式（C 语言语法）	使用 C 语言语法指定标度标签的格式。例如，在该字段中输入字符串“%3.2f s”，标度标签就会显示 3 位数字，其中 2 位是小数位，后面跟字母“s”。
标签的最大文本宽度	指定标度标签的最大宽度的值。该值通常会自动进行正确设置。只有在自动调整无法达到要求的结果时，才会使用该设置选项。
标签的文本高度	指定标度标签的高度的值。该值通常会自动进行正确设置。只有在自动调整无法达到要求的结果时，才会使用该设置选项。
字体颜色	通过选择列表或标准颜色选择对话框中的 [...] 按钮可以设置标签的颜色。

定位

箭头距离	箭头长度（单位：像素）
标签偏移	标签定位的距离（单位：像素）
单位偏移	文本定位的垂直距离（单位：像素）（在 Label（标签） → Unit（单位）下输入）

颜色

颜色根据由红/绿/蓝（RGB）组件组成的十六进制数进行定义。这 3 种颜色各有 256（0-255）个值，您可以在此处静态输入。从选择列表中或通过颜色选择对话框可以选择颜色，通过按钮 [...] 可以打开颜色选择对话框。此外，还可以为每种颜色设置透明度级别（0：完全透明，255：完全不透明）。

颜色区域

持久颜色区域	如果启用该选项，则颜色区域将永久可见。该选项的效果仅在在线模式下可见。
[<n>]	点击 [Create new] 按钮，生成新的颜色区域。为每个颜色区域创建 1 个区域，其中涵盖相应的设置。 [<n>]：数字表示该区域的索引。点击“Delete”（删除），可删除相应的颜色区域及其设置。

区域 [<n>]

区域开始	颜色范围的开始。它必须在规定的标度 [▶ 489]范围内。
区域结束	颜色范围的结束。它必须在规定的标度 [▶ 489]范围内。
颜色	条形区域的颜色

Windows CE 不支持透明度。

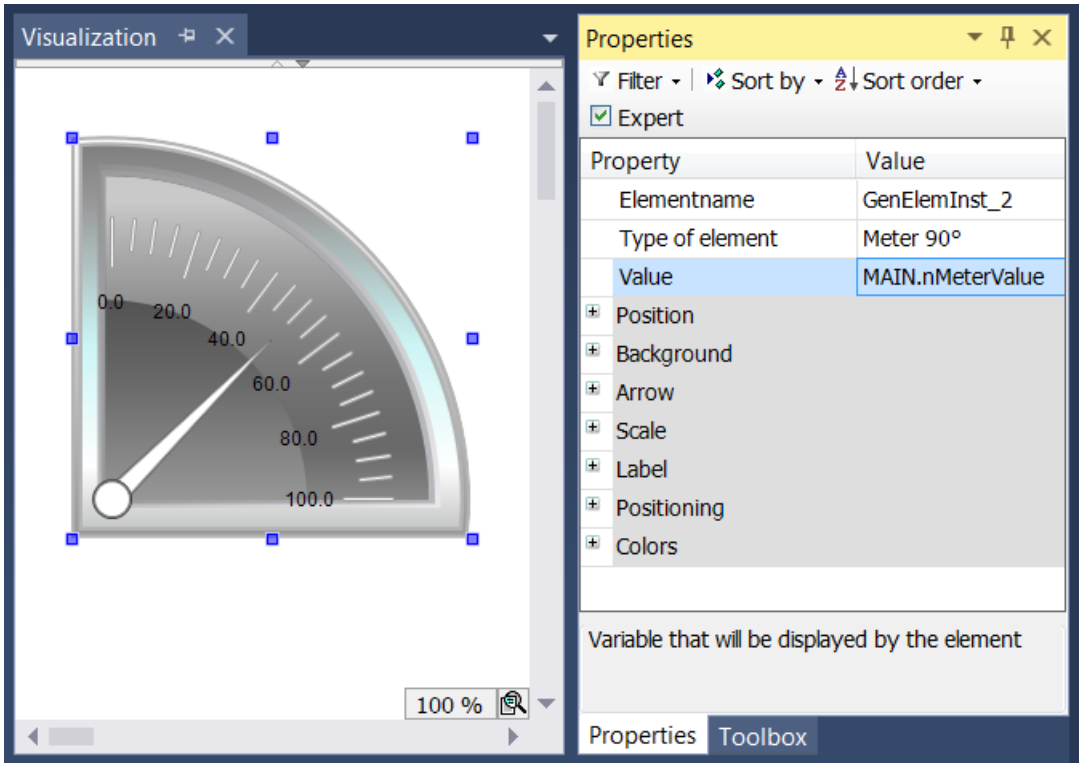
访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

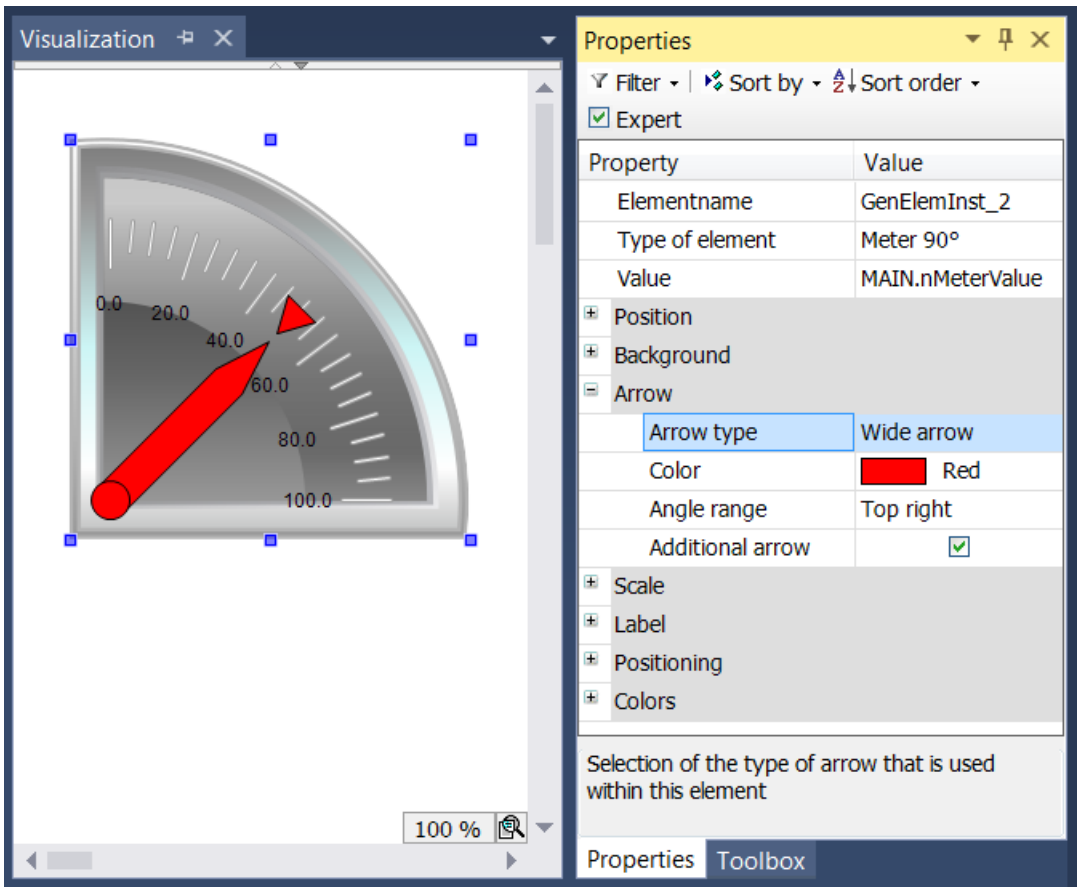
15.8.5.2.1 指针式仪表的配置

下一部分将根据 1 个示例解释指针式仪表的配置。本示例适用于所有可用的指针式仪表。



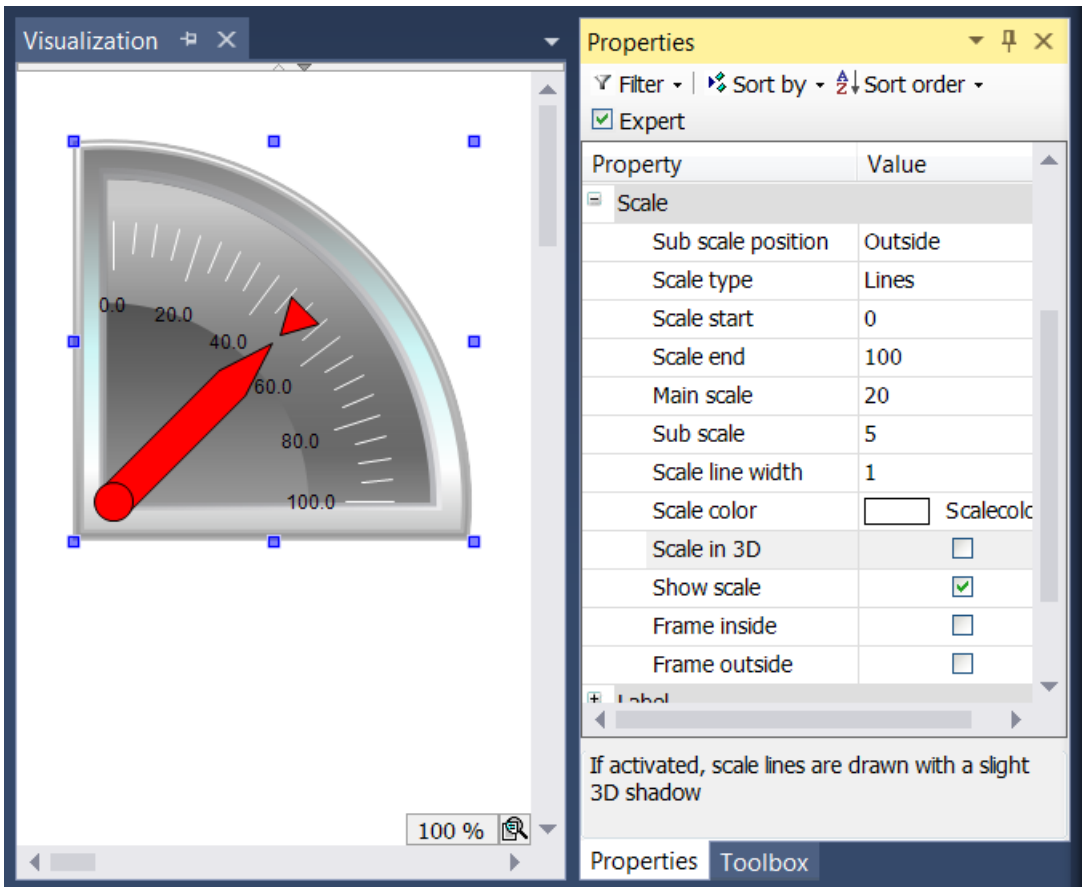
在指针式仪表元素的元素属性中的“Value”（值）下必须指定要关联的输入变量（在本示例中为名称为“nMeterValue”的变量）。在点击进入输入字段之后，即可使用  按钮。它可用于在项目中搜索输入变量。务必在项目树中指定输入变量及其完整路径。

在元素属性的“Arrow”（箭头）部分中可以指定标度显示的方向和大小以及箭头的颜色和外观。



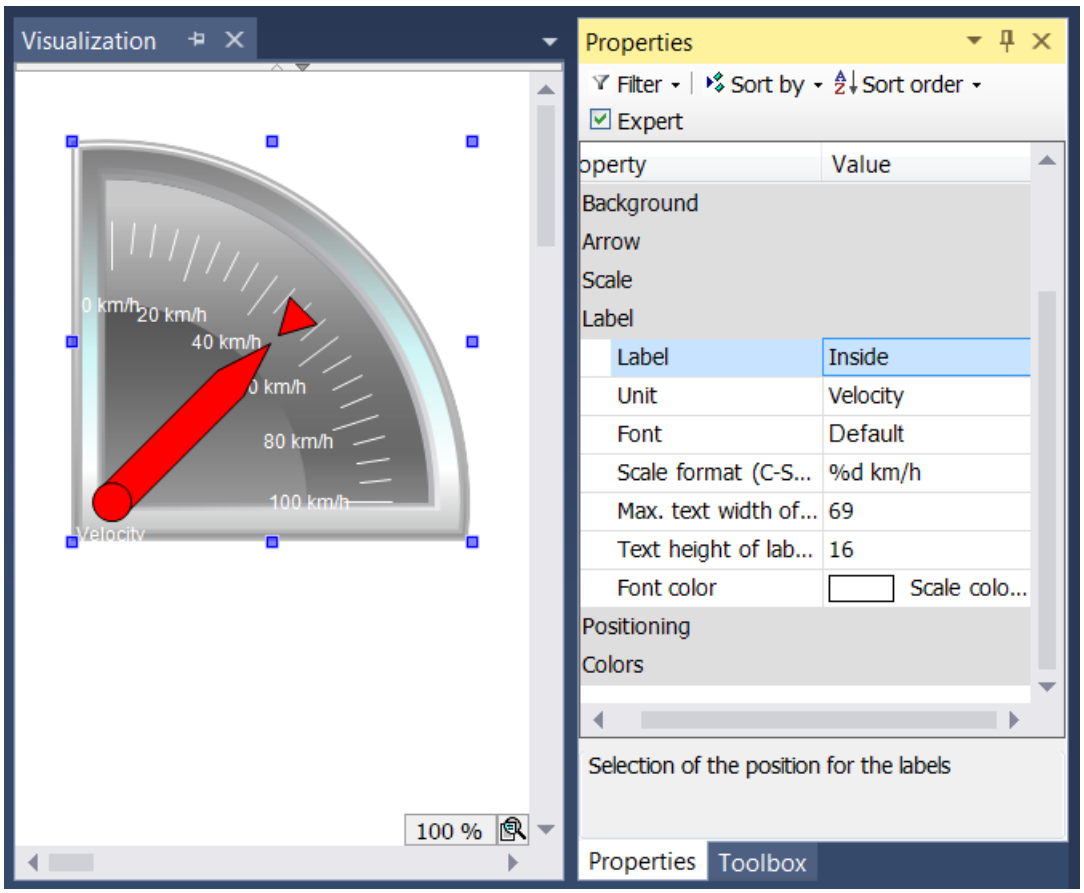
对于名称为指针式仪表的元素，可以设置“Pointer start”（指针开始）和“Pointer end”（指针结束）。它们表示标度左边缘或右边缘与水平线之间的角度（单位：度）。该角度在数学上定向（即逆时针方向），因此，“Pointer start”（指针开始）字段中的值必须始终大于“Pointer end”（指针结束）字段中的值。由起始值和终止值组成的对是 360 度周期性的。

在 Scale（标度）部分中可以指定标度（主标度和子标度）的值范围。



在标度的左边缘会显示标度的起始值。因此，它必须小于标度的终止值，在标度的右边缘会显示终止值。与主标度相比，通过将距离值设置为 0 可省略细标度（子标度）。在这种情况下，不会显示子标度标记。由于在本示例中未勾选“Frame inside”（边框内部）和“Frame outside”（边框外部）复选框，因此，标度的内弧和外弧会被隐藏。

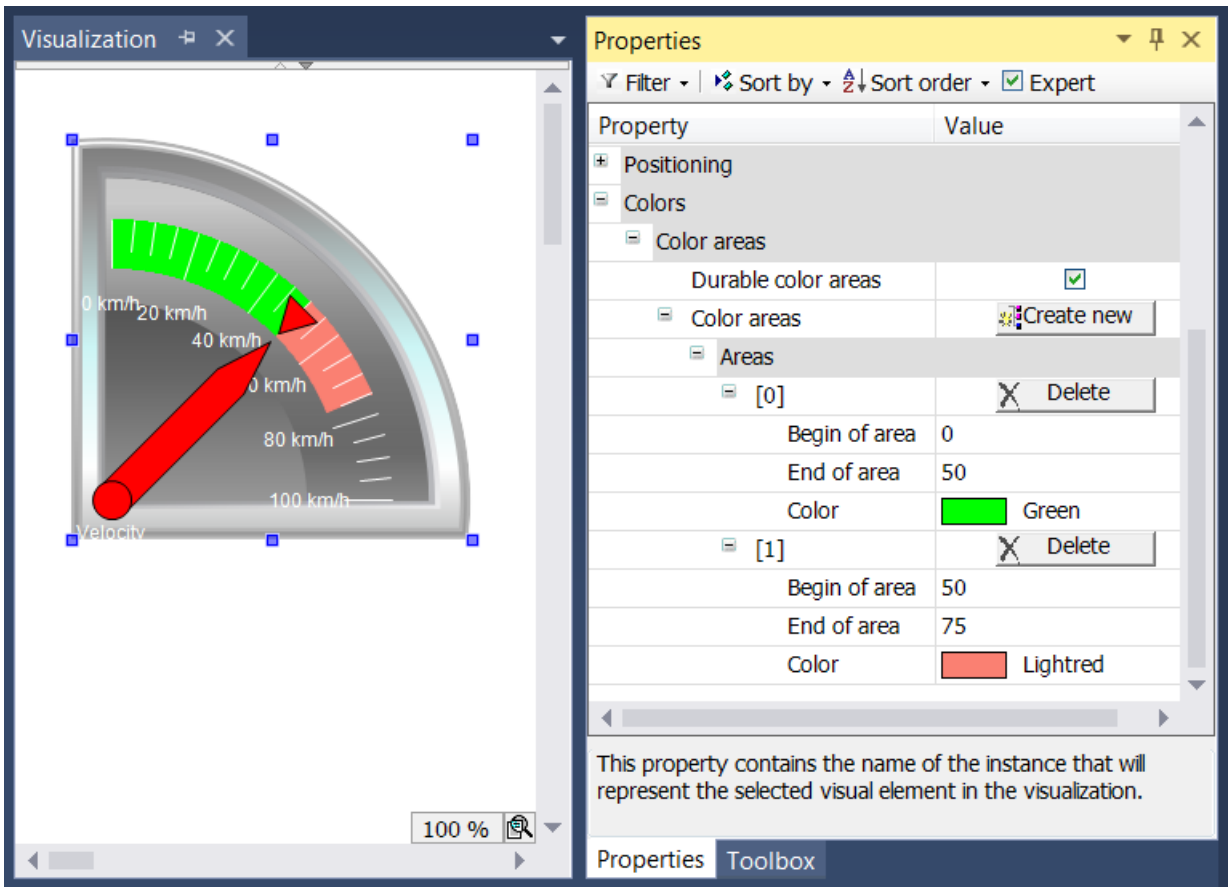
在指定标度之后，在“Label”（标签）部分中可以对标度标签进行格式化。



通过将标签从“Outside”（外部）改为“Inside”（内部），可将标度标记移至标度的内部。在箭头底部的下方会显示“Unit”（单位）字段中的条目。在选择合适的字体和字体颜色之后，您可以调整标度标记的格式。标度的数值必须根据 C 语言的语法进行格式化。“%d”可用于整数，“%.Xf”可用于浮点数，其中“X”应被替换为所需的小数位数。下面 2 个输入字段中的值根据本部分前面的设置进行输入。只有在自动调整无法达到要求的结果时，才需要更改值。

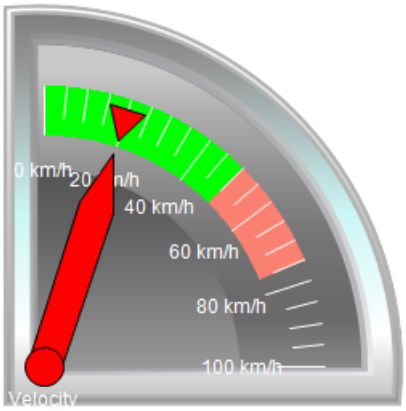
最后，在“Colors”（颜色）部分中，使用 

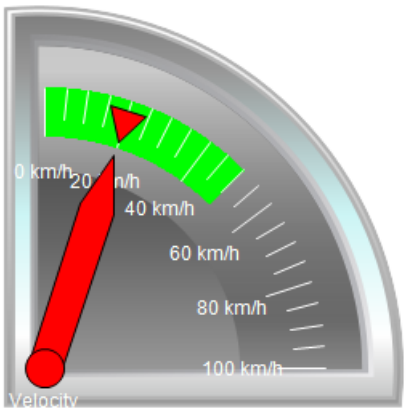
按钮创建颜色区域，从而可以对标度的某些区域进行着色。颜色区域按升序编号。在元素属性中，可为每个颜色区域分配其自己的输入字段。



“Begin of area”（区域开始）和“End of area”（区域结束）字段可用于指定标度的子部分。您可以从下拉菜单中选择颜色，使用 Delete 按钮可以将其再次删除。

复选框“Durable color areas”（持久颜色区域）的效果仅会在运行时显现。在第 1 种情况下，已勾选该复选框，而在第 2 种情况下，未勾选该复选框。

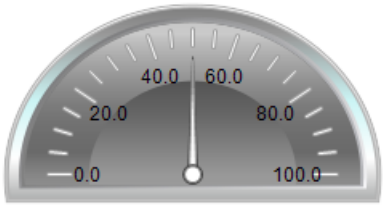




下部指针式仪表仅显示包含箭头的颜色区域，而上部元素则显示已创建的所有颜色区域，因为已勾选复选框“Durable color areas”（持久颜色区域）。

15.8.5.3 指针式仪表 180°

例如，指针式仪表可用于在可视化中添加转数计。箭头根据分配变量的值进行定位。元素采用预先配置的设计，背景颜色 [▶ 504]可以进行设置。可选的是，用户指定的背景图像 [▶ 504]可以替换该设计。可将标度细分为多个颜色区域 [▶ 506]。有关配置示例，请参见“指针式仪表的配置 [▶ 506]”部分。



属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在 View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用 … 按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

辅助设置

值	数值变量，其值显示为箭头的偏移量。

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

背景

如果没有使用用户指定的背景图像，则可使用以下属性：

背景颜色	为条形图选择 1 种颜色： • 黄色 • 红色 • 绿色 • 蓝色 • 灰色

如果使用用户指定的背景图像，则可使用以下属性：

图像	您可以在此处通过指定图像文件的名称或其 ID，从图像池中分配图像。
透明度颜色	对于具有透明背景的图像，您可以选择 1 种要显示为透明的颜色。 ... 按钮可以打开颜色选择对话框。

箭头

箭头类型	许多不同的箭头类型可供选择： • 正常箭头 • 细箭头 • 宽箭头 • 细指针 • 3D 细箭头 • 3D 细指针
颜色	显示箭头的颜色
角度范围	您可以在此处选择 1 个 180° 部分： • 顶部 • 底部 • 居左 • 居右
附加箭头	如果启用该选项，则会在标度上指针的对面显示 1 个附加箭头。

标度

子标度位置	在标度环的外半径或内半径处可以显示子标度： • 外部 • 内部
标度类型	标度可以显示为： • 线 • 点 • 正方形
标度开始	标度下限值
标度结束	标度上限值
主标度	粗标度的 2 条线之间的距离
子标度	细标度的 2 条线之间的距离。如果不需要对粗标度进行进一步细分，则可将该值设置为 0。
标度线宽	标度线的宽度（单位：像素）
标度颜色	通过选择列表或标准颜色选择对话框中的 [...] 按钮可以设置颜色。
3D 标度	如果启用该选项，则会以三维方式显示标度。
显示标度	如果启用该选项，则会显示标度。
边框内部	如果启用该选项，则会使用内部边框绘制标度环。默认情况下，已禁用该选项。
边框外部	如果启用该选项，则会使用外部边框绘制标度环。默认情况下，已禁用该选项。

标签

标签	您可以在此处指定标度值是位于标度的外部还是内部。
单位	输入的文本可用作元素标签。例如，它在标度中心下方显示，可用于指定标度的单位。
字体	您可以在此处为单位和标度设置字体： • 标准 • 页首标题 • 大型 • 标题 • 说明 点击 [...] 按钮，打开用户定义的字体属性设置的对话框。
标度格式（C 语言语法）	使用 C 语言语法指定标度标签的格式。例如，在该字段中输入字符串“%3.2f s”，标度标签就会显示 3 位数字，其中 2 位是小数位，后面跟字母“s”。
标签的最大文本宽度	指定标度标签的最大宽度的值。该值通常会自动进行正确设置。只有在自动调整无法达到要求的结果时，才会使用该设置选项。
标签的文本高度	指定标度标签的高度的值。该值通常会自动进行正确设置。只有在自动调整无法达到要求的结果时，才会使用该设置选项。
字体颜色	通过选择列表或标准颜色选择对话框中的 [...] 按钮可以设置标签的颜色。

定位

箭头距离	箭头长度（单位：像素）
标签偏移	标签定位的距离（单位：像素）
单位偏移	文本定位的垂直距离（单位：像素）（在 Label（标签） → Unit（单位）下输入）

颜色

颜色根据由红/绿/蓝（RGB）组件组成的十六进制数进行定义。这 3 种颜色各有 256（0-255）个值，您可以在此处静态输入。从选择列表中或通过颜色选择对话框可以选择颜色，通过按钮 ... 可以打开颜色选择对话框。此外，还可以为每种颜色设置透明度级别（0：完全透明，255：完全不透明）。

颜色区域

持久颜色区域	如果启用该选项，则颜色区域将永久可见。该选项的效果仅在在线模式下可见。
[<n>]	点击 Create new 按钮，生成新的颜色区域。为每个颜色区域创建 1 个区域，其中涵盖相应的设置。 [<n>]：数字表示该区域的索引。点击“Delete”（删除），可删除相应的颜色区域及其设置。

区域 [<n>]

区域开始	颜色范围的开始。它必须在规定的标度 [▶ 489]范围内。
区域结束	颜色范围的结束。它必须在规定的标度 [▶ 489]范围内。
颜色	条形区域的颜色

Windows CE 不支持透明度。

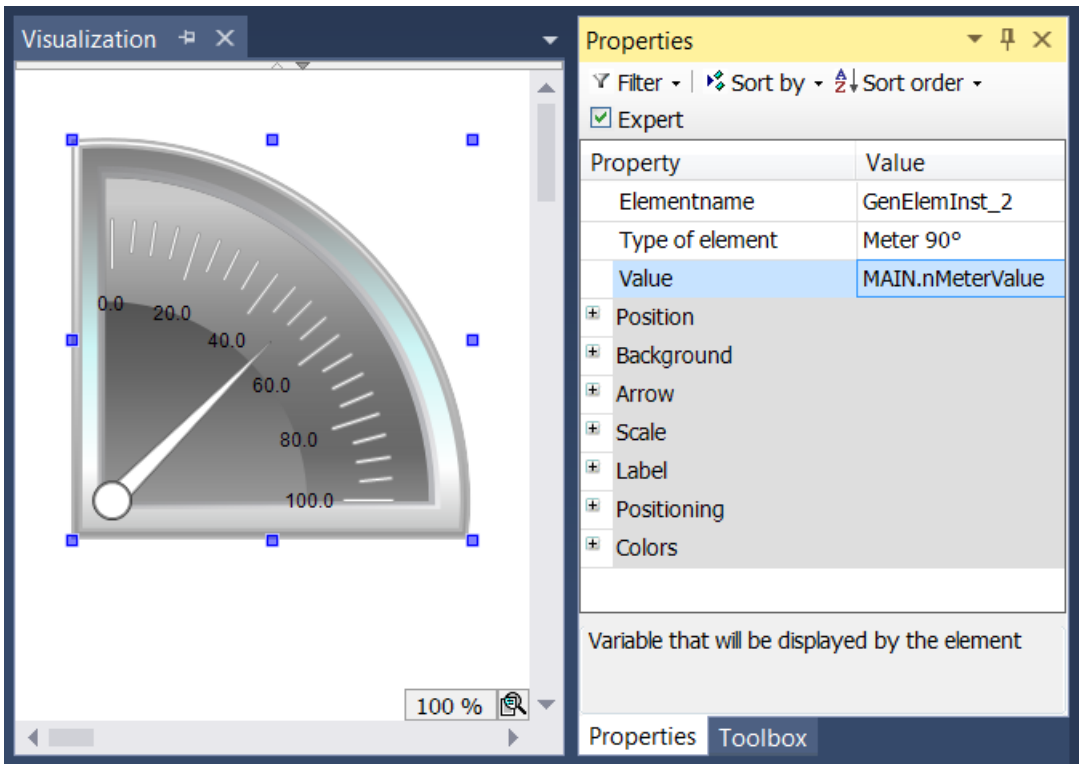
访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

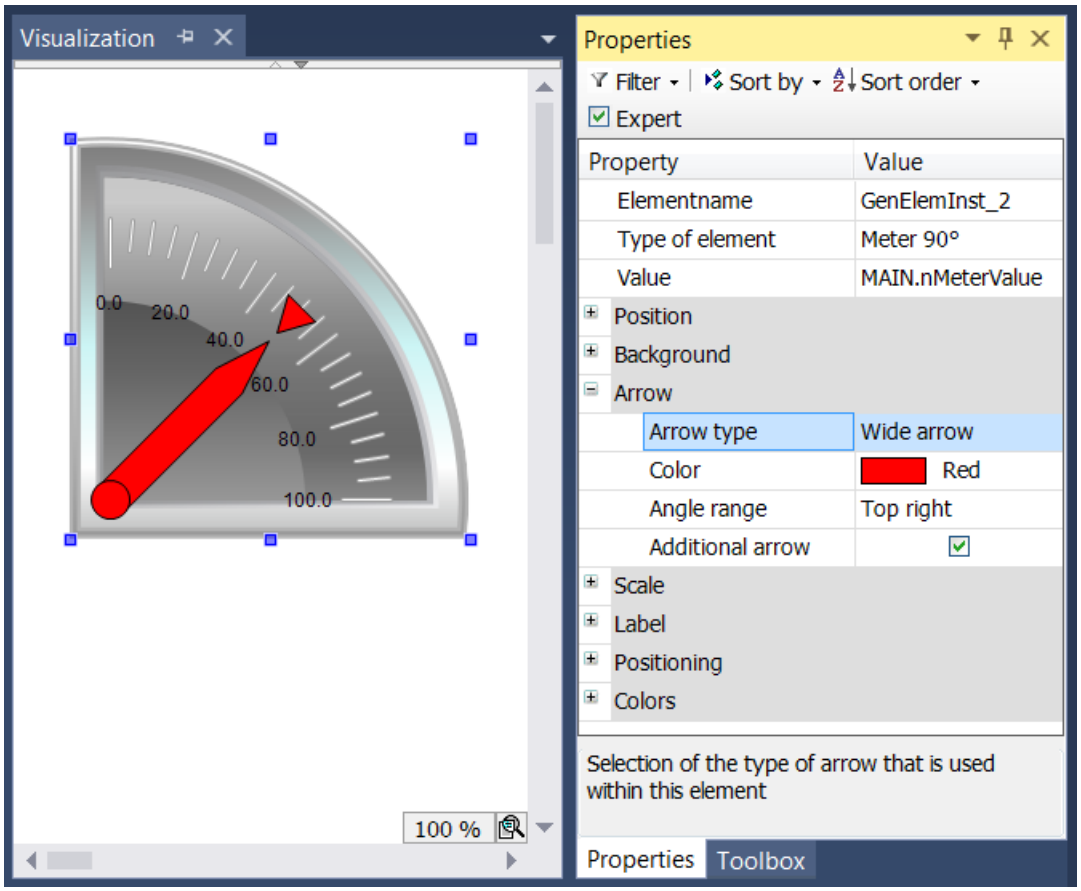
15.8.5.3.1 指针式仪表的配置

下一部分将根据 1 个示例解释指针式仪表的配置。本示例适用于所有可用的指针式仪表。



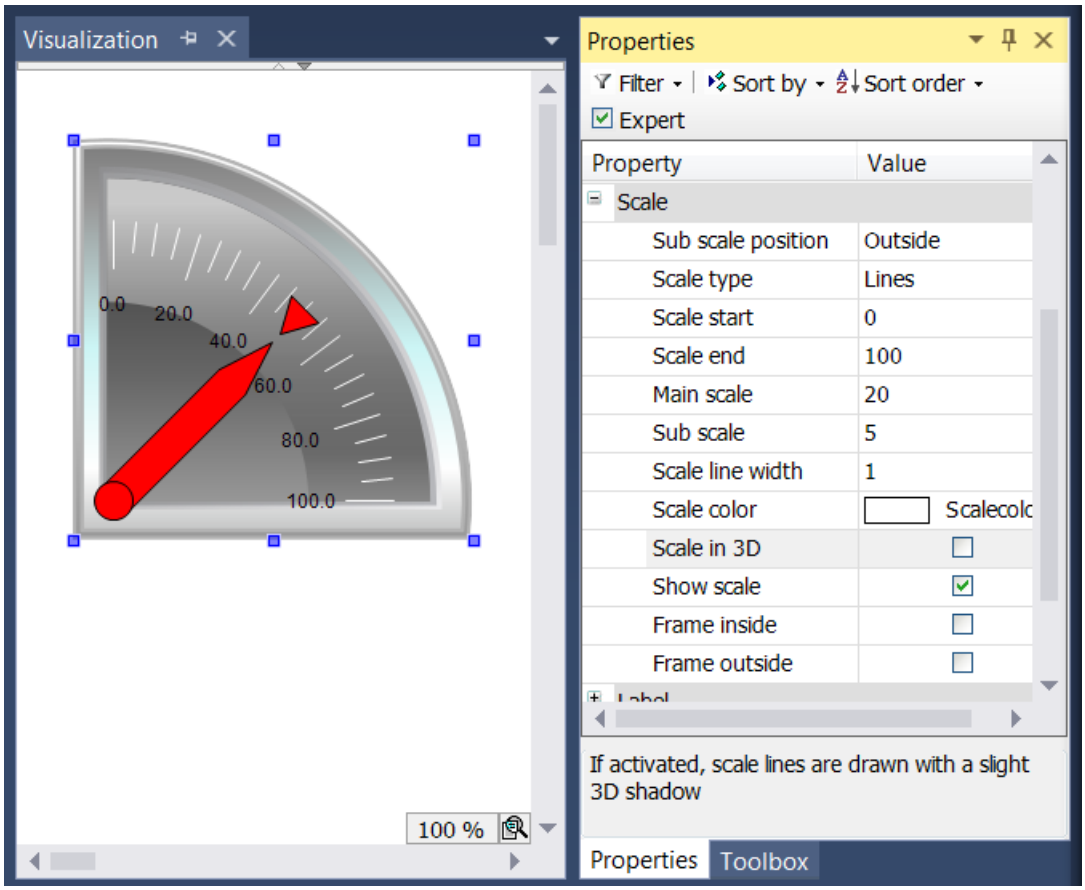
在指针式仪表元素的元素属性中的“Value”（值）下必须指定要关联的输入变量（在本示例中为名称为“nMeterValue”的变量）。在点击进入输入字段之后，即可使用 [...] 按钮。它可用于在项目中搜索输入变量。务必在项目树中指定输入变量及其完整路径。

在元素属性的“Arrow”（箭头）部分中可以指定标度显示的方向和大小以及箭头的颜色和外观。



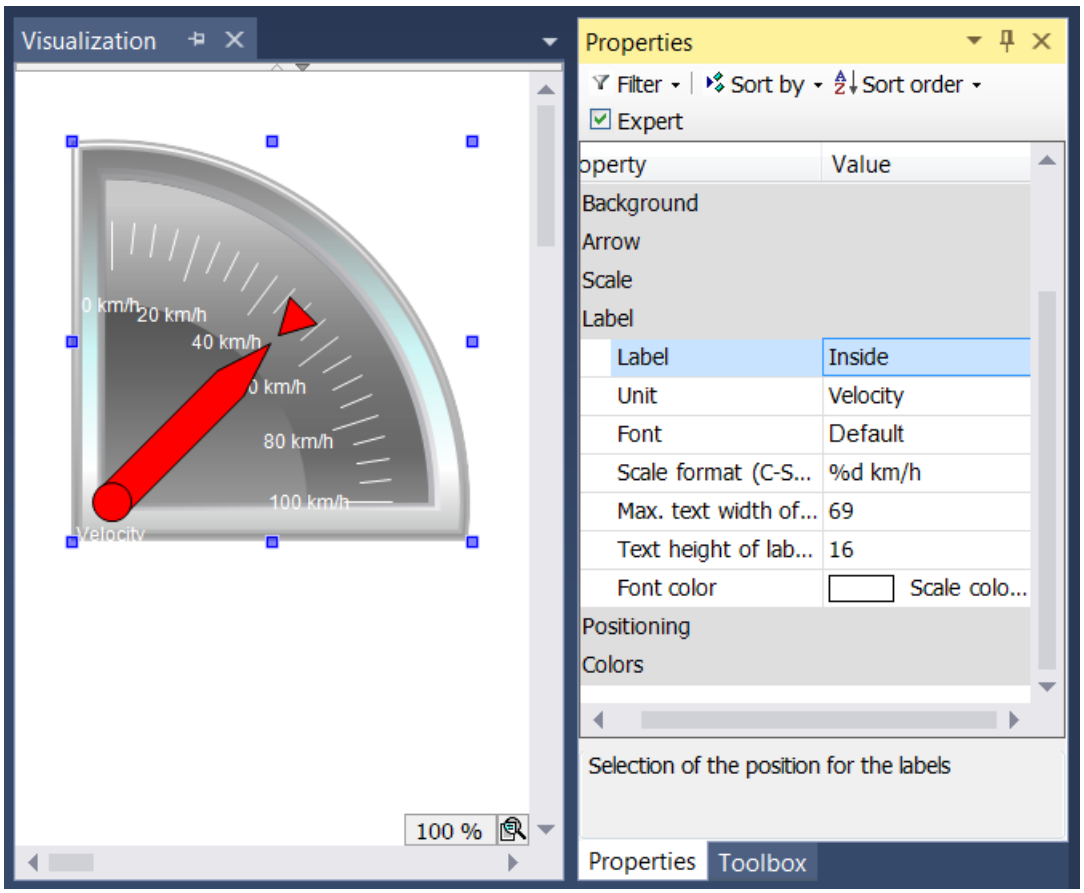
对于名称为指针式仪表的元素，可以设置“Pointer start”（指针开始）和“Pointer end”（指针结束）。它们表示标度左边缘或右边缘与水平线之间的角度（单位：度）。该角度在数学上定向（即逆时针方向），因此，“Pointer start”（指针开始）字段中的值必须始终大于“Pointer end”（指针结束）字段中的值。由起始值和终止值组成的对是 360 度周期性的。

在 Scale（标度）部分中可以指定标度（主标度和子标度）的值范围。



在标度的左边缘会显示标度的起始值。因此，它必须小于标度的终止值，在标度的右边缘会显示终止值。与主标度相比，通过将距离值设置为 0 可省略细标度（子标度）。在这种情况下，不会显示子标度标记。由于在本示例中未勾选“Frame inside”（边框内部）和“Frame outside”（边框外部）复选框，因此，标度的内弧和外弧会被隐藏。

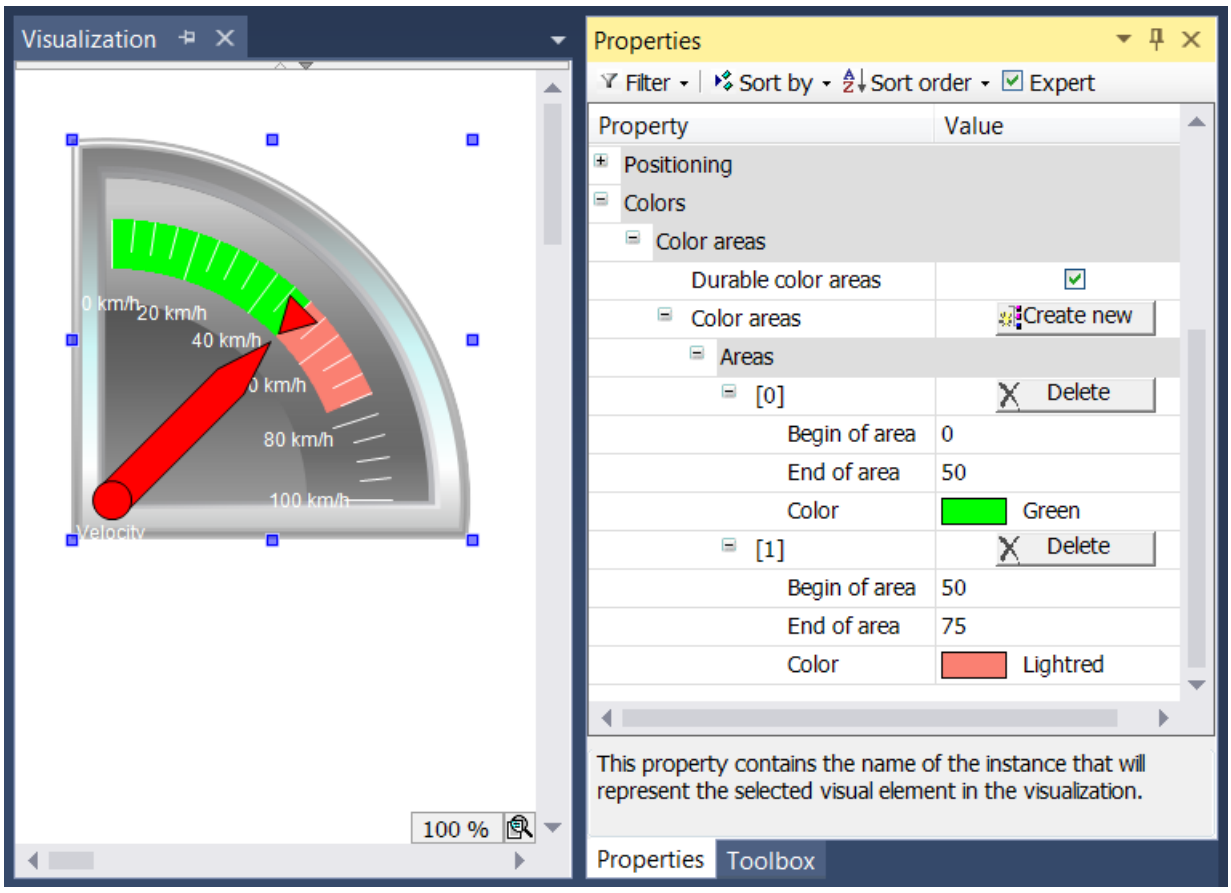
在指定标度之后，在“Label”（标签）部分中可以对标度标签进行格式化。



通过将标签从“Outside”（外部）改为“Inside”（内部），可将标度标记移至标度的内部。在箭头底部的下方会显示“Unit”（单位）字段中的条目。在选择合适的字体和字体颜色之后，您可以调整标度标记的格式。标度的数值必须根据 C 语言的语法进行格式化。“%d”可用于整数，“%.Xf”可用于浮点数，其中“X”应被替换为所需的小数位数。下面 2 个输入字段中的值根据本部分前面的设置进行输入。只有在自动调整无法达到要求的结果时，才需要更改值。

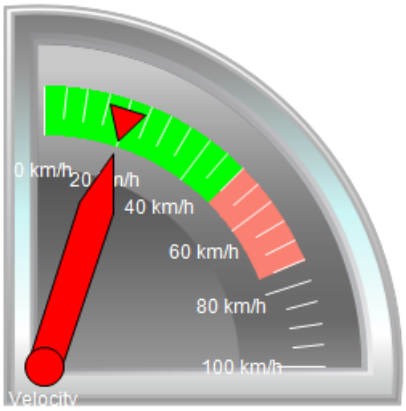
最后，在“Colors”（颜色）部分中，使用 

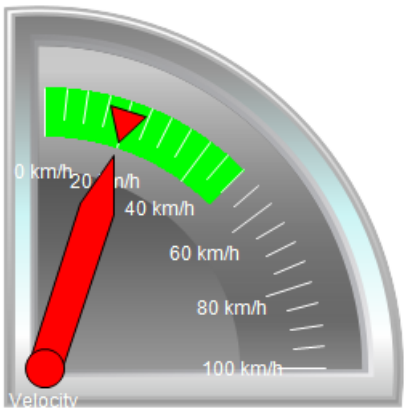
按钮创建颜色区域，从而可以对标度的某些区域进行着色。颜色区域按升序编号。在元素属性中，可为每个颜色区域分配其自己的输入字段。



“Begin of area”（区域开始）和“End of area”（区域结束）字段可用于指定标度的子部分。您可以从下拉菜单中选择颜色，使用 Delete 按钮可以将其再次删除。

复选框“Durable color areas”（持久颜色区域）的效果仅会在运行时显现。在第 1 种情况下，已勾选该复选框，而在第 2 种情况下，未勾选该复选框。

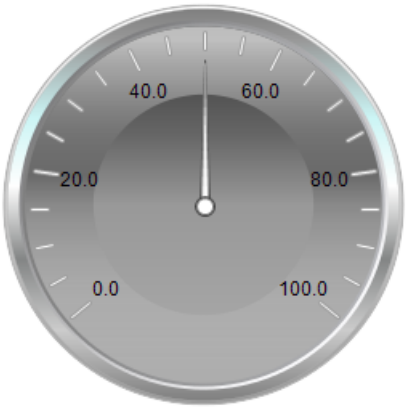




下部指针式仪表仅显示包含箭头的颜色区域，而上部元素则显示已创建的所有颜色区域，因为已勾选复选框“Durable color areas”（持久颜色区域）。

15.8.5.4 指针式仪表

例如，指针式仪表可用于在可视化中添加转数计。箭头根据分配变量的值进行定位。元素采用预先配置的设计，背景颜色 [▶ 512]可以进行设置。可选的是，用户指定的背景图像 [▶ 512]可以替换该设计。可将标度细分为多个颜色区域 [▶ 514]。有关配置示例，请参见“指针式仪表的配置 [▶ 514]”部分。



属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用 [...] 按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

辅助设置

值	数值变量，其值显示为箭头的偏移量。

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

背景

如果没有使用用户指定的背景图像，则可使用以下属性：

背景颜色	为条形图选择 1 种颜色： • 黄色 • 红色 • 绿色 • 蓝色 • 灰色

如果使用用户指定的背景图像，则可使用以下属性：

图像	您可以在此处通过指定图像文件的名称或其 ID，从图像池中分配图像。
透明度颜色	对于具有透明背景的图像，您可以选择 1 种要显示为透明的颜色。 ... 按钮可以打开颜色选择对话框。

箭头

箭头类型	许多不同的箭头类型可供选择： • 正常箭头 • 细箭头 • 宽箭头 • 细指针 • 3D 细箭头 • 3D 细指针
颜色	显示箭头的颜色
指针开始	此处指定的值被解释为角度（单位：度），其范围为沿着逆时针方向从标度的原点到标度区域的开始处。标度的原点在“3 点钟”位置。角度的值范围在 0° 到 360° 之间。旋转的中心在 X/Y 位置。
指针结束	此处指定的值被解释为角度（单位：度），其范围为沿着逆时针方向从标度的原点到标度区域的结束处。标度的原点在“3 点钟”位置。允许使用负值。旋转的中心在 X/Y 位置。从顺时针方向看，必须将指针结束处设置在指针开始处的右侧。角度的值范围在 0° 到最大值 350° 之间。
附加箭头	如果启用该选项，则会在标度上指针的对面显示 1 个附加箭头。

标度

子标度位置	在标度环的外半径或内半径处可以显示子标度： • 外部 • 内部
标度类型	标度可以显示为： • 线 • 点 • 正方形
标度开始	标度下限值
标度结束	标度上限值
主标度	粗标度的 2 条线之间的距离
子标度	细标度的 2 条线之间的距离。如果不需要对粗标度进行进一步细分，则可将该值设置为 0。
标度线宽	标度线的宽度（单位：像素）
标度颜色	通过选择列表或标准颜色选择对话框中的 [...] 按钮可以设置颜色。
3D 标度	如果启用该选项，则会以三维方式显示标度。
显示标度	如果启用该选项，则会显示标度。
边框内部	如果启用该选项，则会使用内部边框绘制标度环。默认情况下，已禁用该选项。
边框外部	如果启用该选项，则会使用外部边框绘制标度环。默认情况下，已禁用该选项。

标签

标签	您可以在此处指定标度值是位于标度的外部还是内部。
单位	输入的文本可用作元素标签。例如，它在标度中心下方显示，可用于指定标度的单位。
字体	您可以在此处为单位和标度设置字体： • 标准 • 页首标题 • 大型 • 标题 • 说明 点击 [...] 按钮，打开用户定义的字体属性设置的对话框。
标度格式（C 语言语法）	使用 C 语言语法指定标度标签的格式。例如，在该字段中输入字符串“%3.2f s”，标度标签就会显示 3 位数字，其中 2 位是小数位，后面跟字母“s”。
标签的最大文本宽度	指定标度标签的最大宽度的值。该值通常会自动进行正确设置。只有在自动调整无法达到要求的结果时，才会使用该设置选项。
标签的文本高度	指定标度标签的高度的值。该值通常会自动进行正确设置。只有在自动调整无法达到要求的结果时，才会使用该设置选项。
字体颜色	通过选择列表或标准颜色选择对话框中的 [...] 按钮可以设置标签的颜色。

定位

箭头距离	箭头长度（单位：像素）
标度距离	从标度线到中心的距离（单位：像素）。只有当您已定义自己的背景图像时，该属性才可用。
音阶长度	标度线的长度（单位：像素）。只有当您已定义自己的背景图像时，该属性才可用。
标签偏移	标签定位的距离（单位：像素）
单位偏移	文本定位的垂直距离（单位：像素）（在 Label（标签） → Unit（单位）下输入）
原点偏移	元素的偏移。它可以用来实现相对于背景图像的精确定位。

颜色

颜色根据由红/绿/蓝（RGB）组件组成的十六进制数进行定义。这 3 种颜色各有 256（0-255）个值，您可以在此处静态输入。从选择列表中或通过颜色选择对话框可以选择颜色，通过按钮  可以打开颜色选择对话框。此外，还可以为每种颜色设置透明度级别（0：完全透明，255：完全不透明）。

颜色区域

持久颜色区域	如果启用该选项，则颜色区域将永久可见。该选项的效果仅在在线模式下可见。
[<n>]	点击  按钮，生成新的颜色区域。为每个颜色区域创建 1 个区域，其中涵盖相应的设置。 [<n>]：数字表示该区域的索引。点击“Delete”（删除），可删除相应的颜色区域及其设置。

区域 [<n>]

区域开始	颜色范围的开始。它必须在规定的标度 [▶ 489]范围内。
区域结束	颜色范围的结束。它必须在规定的标度 [▶ 489]范围内。
颜色	条形区域的颜色

Windows CE 不支持透明度。

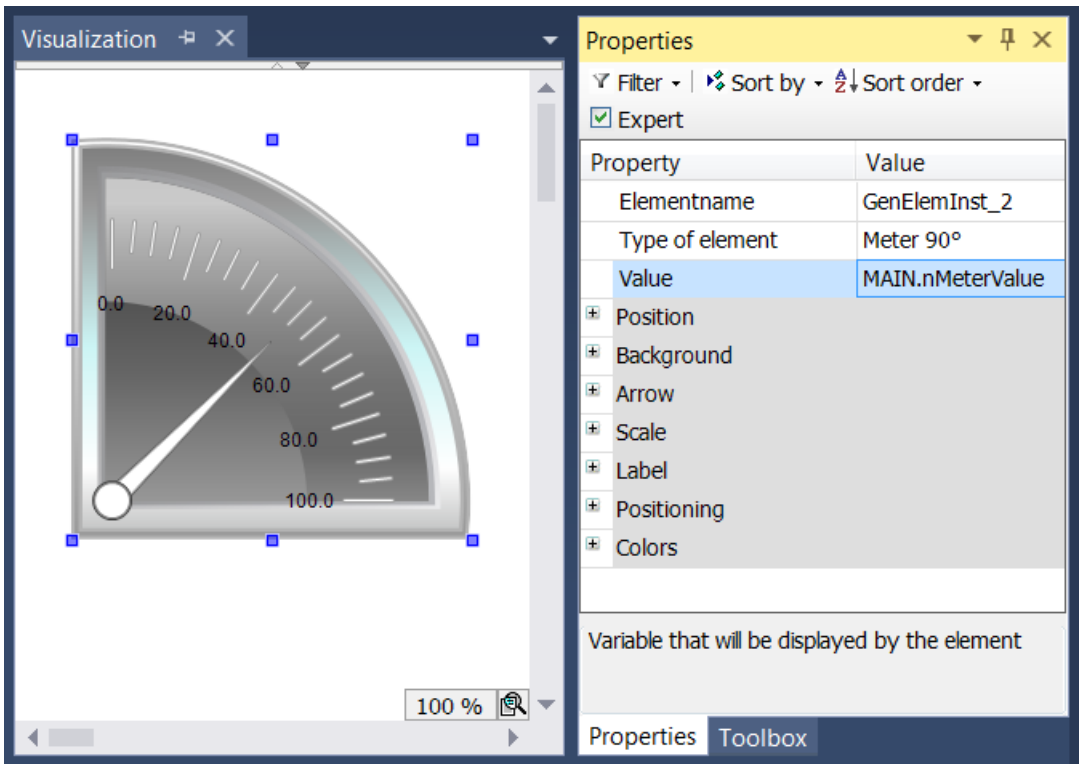
访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

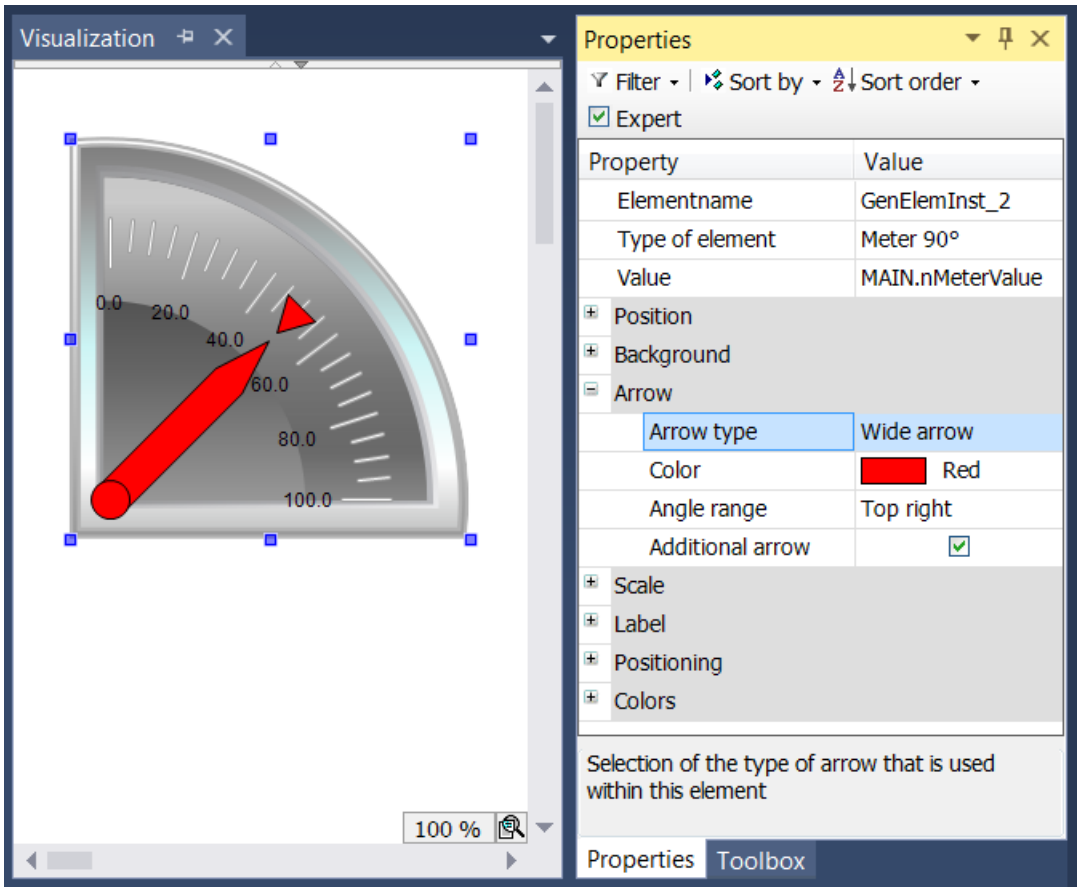
15.8.5.4.1 指针式仪表的配置

下一部分将根据 1 个示例解释指针式仪表的配置。本示例适用于所有可用的指针式仪表。



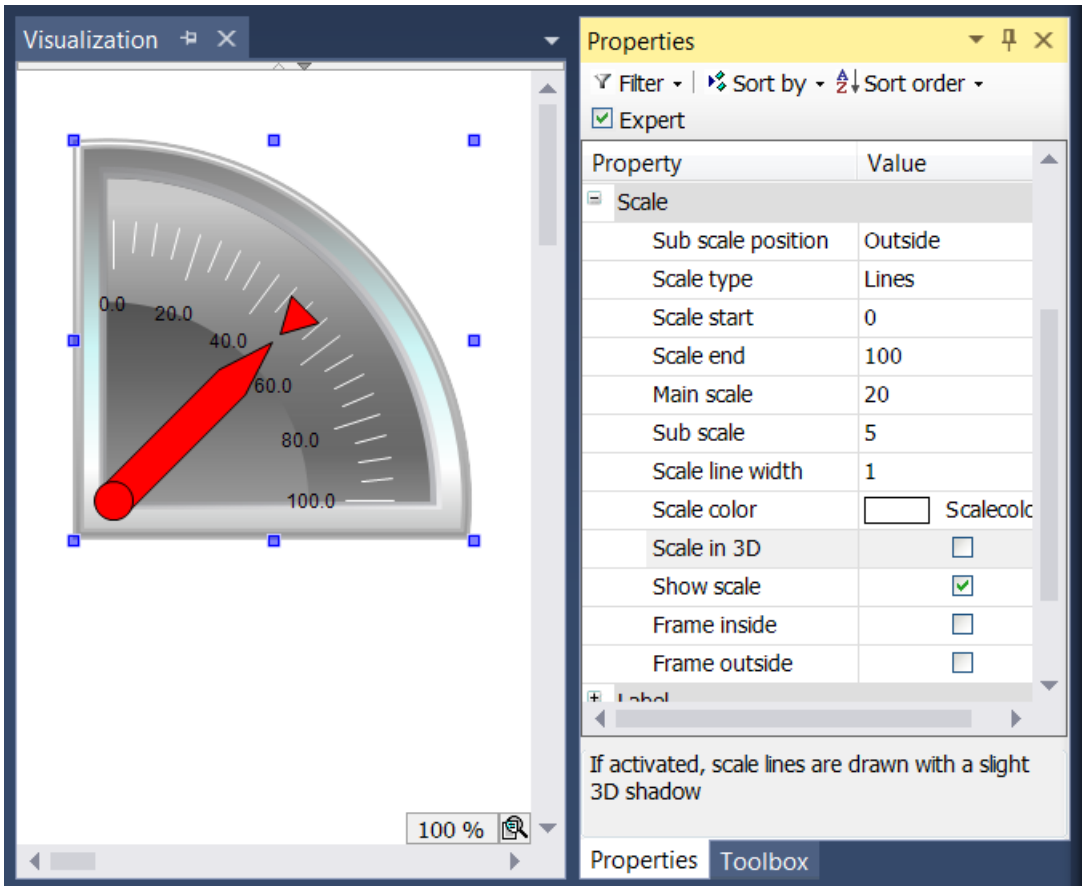
在指针式仪表元素的元素属性中的“Value”（值）下必须指定要关联的输入变量（在本示例中为名称为“nMeterValue”的变量）。在点击进入输入字段之后，即可使用 [...] 按钮。它可用于在项目中搜索输入变量。务必在项目树中指定输入变量及其完整路径。

在元素属性的“Arrow”（箭头）部分中可以指定标度显示的方向和大小以及箭头的颜色和外观。



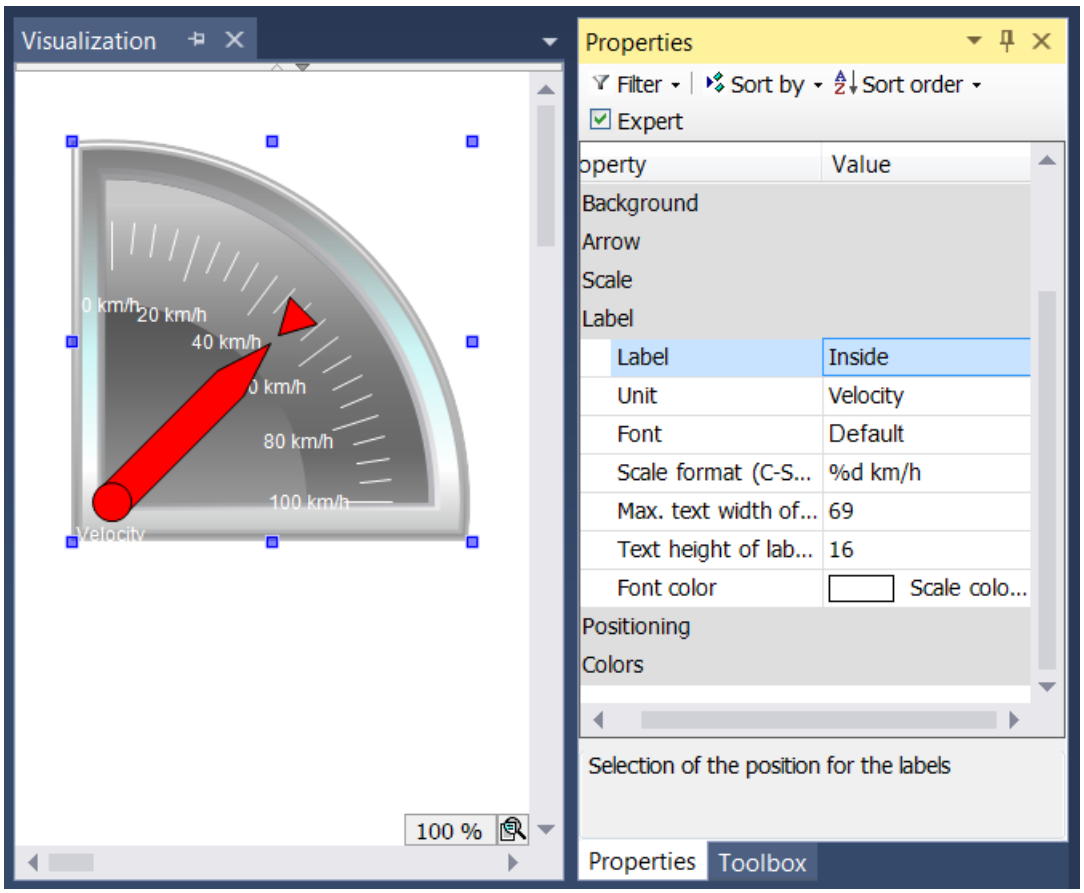
对于名称为指针式仪表的元素，可以设置“Pointer start”（指针开始）和“Pointer end”（指针结束）。它们表示标度左边缘或右边缘与水平线之间的角度（单位：度）。该角度在数学上定向（即逆时针方向），因此，“Pointer start”（指针开始）字段中的值必须始终大于“Pointer end”（指针结束）字段中的值。由起始值和终止值组成的对是 360 度周期性的。

在 Scale（标度）部分中可以指定标度（主标度和子标度）的值范围。



在标度的左边缘会显示标度的起始值。因此，它必须小于标度的终止值，在标度的右边缘会显示终止值。与主标度相比，通过将距离值设置为 0 可省略细标度（子标度）。在这种情况下，不会显示子标度标记。由于在本示例中未勾选“Frame inside”（边框内部）和“Frame outside”（边框外部）复选框，因此，标度的内弧和外弧会被隐藏。

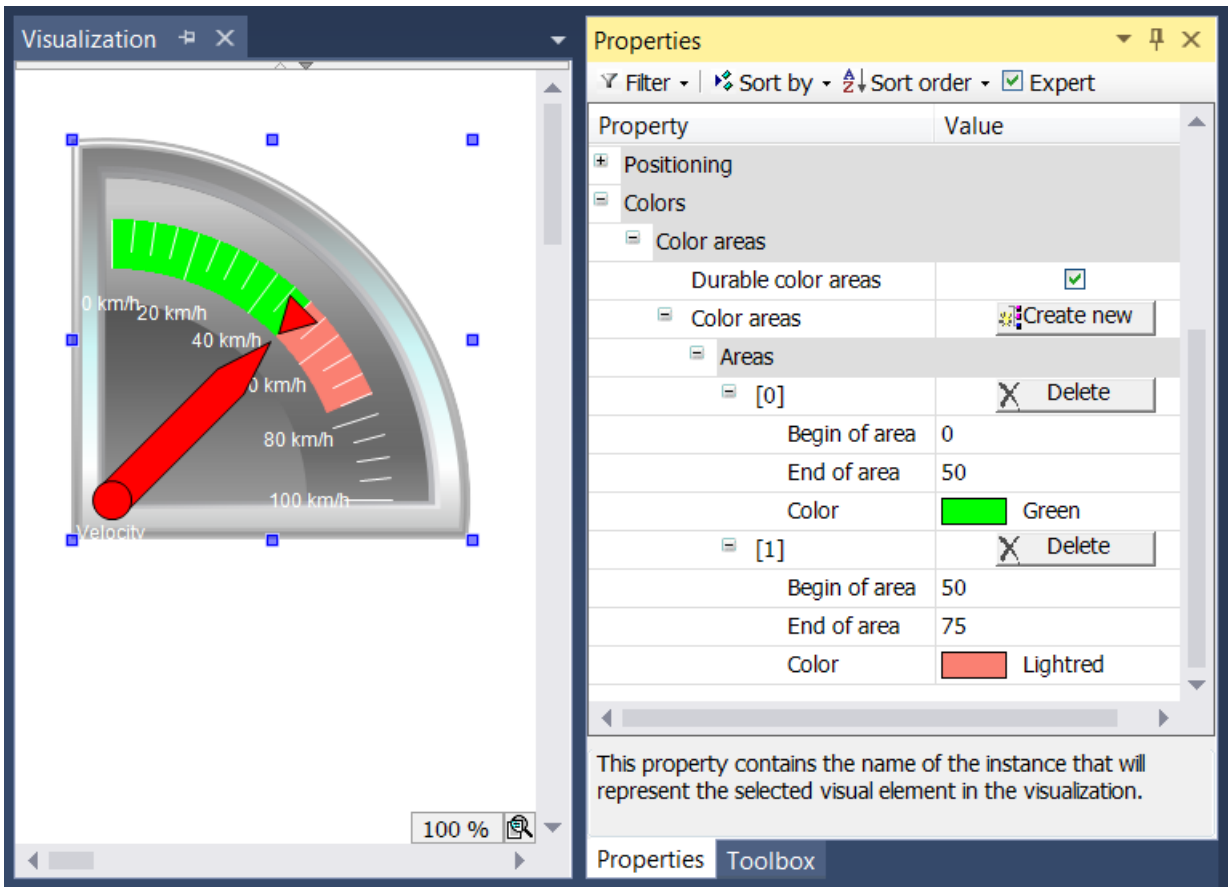
在指定标度之后，在“Label”（标签）部分中可以对标度标签进行格式化。



通过将标签从“Outside”（外部）改为“Inside”（内部），可将标度标记移至标度的内部。在箭头底部的下方会显示“Unit”（单位）字段中的条目。在选择合适的字体和字体颜色之后，您可以调整标度标记的格式。标度的数值必须根据 C 语言的语法进行格式化。“%d”可用于整数，“%.Xf”可用于浮点数，其中“X”应被替换为所需的小数位数。下面 2 个输入字段中的值根据本部分前面的设置进行输入。只有在自动调整无法达到要求的结果时，才需要更改值。

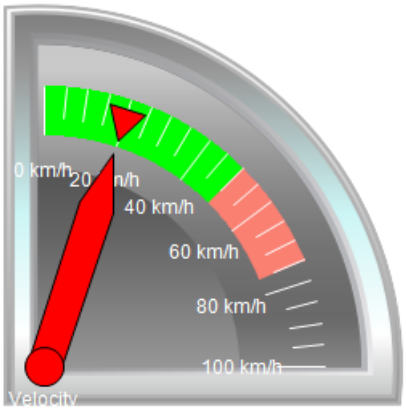
最后，在“Colors”（颜色）部分中，使用 

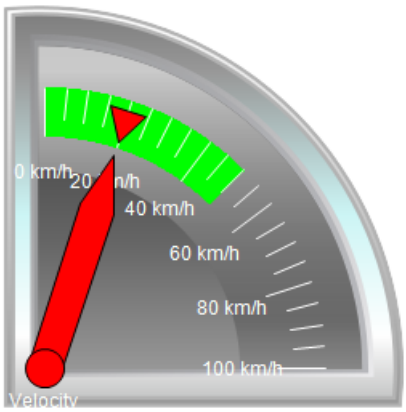
按钮创建颜色区域，从而可以对标度的某些区域进行着色。颜色区域按升序编号。在元素属性中，可为每个颜色区域分配其自己的输入字段。



“Begin of area”（区域开始）和“End of area”（区域结束）字段可用于指定标度的子部分。您可以从下拉菜单中选择颜色，使用 Delete 按钮可以将其再次删除。

复选框“Durable color areas”（持久颜色区域）的效果仅会在运行时显现。在第 1 种情况下，已勾选该复选框，而在第 2 种情况下，未勾选该复选框。

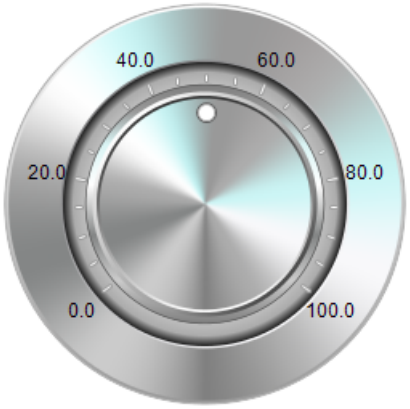




下部指针式仪表仅显示包含箭头的颜色区域，而上部元素则显示已创建的所有颜色区域，因为已勾选复选框“Durable color areas”（持久颜色区域）。

15.8.5.5 电位计

该元素可用于为可视化页面添加采用预定义设计的电位计。该操作元素使用箭头或指针来指示可通过用户输入移动的变量值。



属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用 … 按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

辅助设置

变量	数值变量，包含电位计的位置。

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

背景

如果没有使用用户指定的背景图像，则可使用以下属性：

背景颜色	为条形图选择 1 种颜色： • 黄色 • 红色 • 绿色 • 蓝色 • 灰色

如果使用用户指定的背景图像，则可使用以下属性：

图像	您可以在此处通过指定图像文件的名称或其 ID，从图像池中分配图像。
透明度颜色	对于具有透明背景的图像，您可以选择 1 种要显示为透明的颜色。 ... 按钮可以打开颜色选择对话框。

箭头

箭头类型	许多不同的箭头类型可供选择： • 正常箭头 • 细箭头 • 宽箭头 • 细指针 • 3D 细箭头 • 3D 细指针
颜色	显示箭头的颜色
指针开始	此处指定的值被解释为角度（单位：度），其范围为沿着逆时针方向从标度的原点到标度区域的开始处。标度的原点在“3 点钟”位置。角度的值范围在 0° 到 360° 之间。旋转的中心在 X/Y 位置。
指针结束	此处指定的值被解释为角度（单位：度），其范围为沿着逆时针方向从标度的原点到标度区域的结束处。标度的原点在“3 点钟”位置。允许使用负值。旋转的中心在 X/Y 位置。从顺时针方向看，必须将指针结束处设置在指针开始处的右侧。角度的值范围在 0° 到最大值 350° 之间。

标度

子标度位置	在标度环的外半径或内半径处可以显示子标度： • 外部 • 内部
标度类型	标度可以显示为： • 线 • 点 • 正方形
标度开始	标度下限值
标度结束	标度上限值
主标度	粗标度的 2 条线之间的距离
子标度	细标度的 2 条线之间的距离。如果不需要对粗标度进行进一步细分，则可将该值设置为 0。
标度线宽	标度线的宽度（单位：像素）
标度颜色	通过选择列表或标准颜色选择对话框中的 [...] 按钮可以设置颜色。
3D 标度	如果启用该选项，则会以三维方式显示标度。
显示标度	如果启用该选项，则会显示标度。
边框内部	如果启用该选项，则会使用内部边框绘制标度环。默认情况下，已禁用该选项。
边框外部	如果启用该选项，则会使用外部边框绘制标度环。默认情况下，已禁用该选项。

标签

标签	您可以在此处指定标度值是位于标度的外部还是内部。
单位	输入的文本可用作元素标签。例如，它在标度中心下方显示，可用于指定标度的单位。
字体	您可以在此处为单位和标度设置字体： • 标准 • 页首标题 • 大型 • 标题 • 说明 点击 [...] 按钮，打开用户定义的字体属性设置的对话框。
标度格式（C 语言语法）	使用 C 语言语法指定标度标签的格式。例如，在该字段中输入字符串“%3.2f s”，标度标签就会显示 3 位数字，其中 2 位是小数位，后面跟字母“s”。
标签的最大文本宽度	指定标度标签的最大宽度的值。该值通常会自动进行正确设置。只有在自动调整无法达到要求的结果时，才会使用该设置选项。
标签的文本高度	指定标度标签的高度的值。该值通常会自动进行正确设置。只有在自动调整无法达到要求的结果时，才会使用该设置选项。
字体颜色	通过选择列表或标准颜色选择对话框中的 [...] 按钮可以设置标签的颜色。

定位

箭头距离	箭头长度（单位：像素）
标度距离	从标度线到中心的距离（单位：像素）。只有当您已定义自己的背景图像时，该属性才可用。
音阶长度	标度线的长度（单位：像素）。只有当您已定义自己的背景图像时，该属性才可用。
标签偏移	标签定位的距离（单位：像素）
单位偏移	文本定位的垂直距离（单位：像素）（在 Label（标签） → Unit（单位）下输入）
原点偏移	元素的偏移。它可以用来实现相对于背景图像的精确定位。

颜色

颜色根据由红/绿/蓝（RGB）组件组成的十六进制数进行定义。这 3 种颜色各有 256（0-255）个值，您可以在此处静态输入。从选择列表中或通过颜色选择对话框可以选择颜色，通过按钮  可以打开颜色选择对话框。此外，还可以为每种颜色设置透明度级别（0：完全透明，255：完全不透明）。

颜色区域

持久颜色区域	如果启用该选项，则颜色区域将永久可见。该选项的效果仅在在线模式下可见。
[<n>]	点击  按钮，生成新的颜色区域。为每个颜色区域创建 1 个区域，其中涵盖相应的设置。 [<n>]：数字表示该区域的索引。点击“Delete”（删除），可删除相应的颜色区域及其设置。

区域 [<n>]

区域开始	颜色范围的开始。它必须在规定的标度 [▶ 489]范围内。
区域结束	颜色范围的结束。它必须在规定的标度 [▶ 489]范围内。
颜色	条形区域的颜色

Windows CE 不支持透明度。

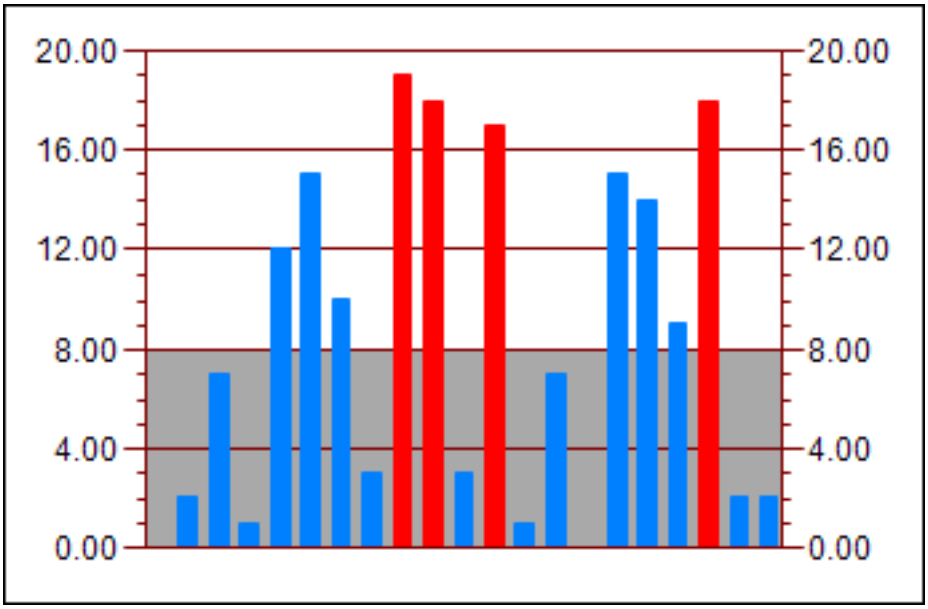
访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

15.8.5.6 直方图

该元素可用于为可视化页面添加直方图，以显示数组的值。在此处可以指定最小和最大显示值。对于某些值范围，可以定义特殊的颜色 [▶ 525]。有关配置示例，请参见“直方图的配置 [▶ 525]”部分。



属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用  按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

辅助设置

数据数组	数组类型的变量，其数据将可视化。

数组的子范围

使用子范围	如果启用该选项，则只使用已分配数据数组的子范围
起始索引	为已分配数据数组的第 1 个数据集建立索引。
结束索引	为已分配数据数组的最后一个数据集建立索引。

显示类型	您可以在此处选择直方图中的数据的显示类型： • 条 • 线 • 曲线
线宽	直方图中表示数据的曲线的厚度。仅在选择“Curve”（曲线）显示类型时，此设置才有效。
显示水平线	如果启用该选项，则会在主标度处绘制水平线。
相对条宽	条或行的相对宽度（单位：像素）

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

标度

标度开始	标度下限值
标度结束	标度上限值
主标度	粗标度的 2 条线之间的距离
子标度	细标度的 2 条线之间的距离。如果不需要对粗标度进行进一步细分，则可将该值设置为 0。
标度颜色	通过选择列表或标准颜色选择对话框中的  按钮可以设置颜色。
基线	将直方图的基线向上移动的数值。

标签

单位	输入的文本可用作元素标签。例如，它在标度中心下方显示，可用于指定标度的单位。
字体	您可以在此处为单位和标度设置字体： • 标准 • 页首标题 • 大型 • 标题 • 说明 点击  按钮，打开用户定义的字体属性设置的对话框。
标度格式（C 语言语法）	使用 C 语言语法指定标度标签的格式。例如，在该字段中输入字符串“%3.2f s”，标度标签就会显示 3 位数字，其中 2 位是小数位，后面跟字母“s”。
标签的最大文本宽度	指定标度标签的最大宽度的值。该值通常会自动进行正确设置。只有在自动调整无法达到要求的结果时，才会使用该设置选项。
标签的文本高度	指定标度标签的高度的值。该值通常会自动进行正确设置。只有在自动调整无法达到要求的结果时，才会使用该设置选项。
字体颜色	通过选择列表或标准颜色选择对话框中的  按钮可以设置标签的颜色。

颜色

颜色根据由红/绿/蓝（RGB）组件组成的十六进制数进行定义。这 3 种颜色各有 256（0-255）个值，您可以在此处静态输入。从选择列表中或通过颜色选择对话框可以选择颜色，通过按钮 ... 可以打开颜色选择对话框。此外，还可以为每种颜色设置透明度级别（0：完全透明，255：完全不透明）。

图形颜色	图形的颜色
报警颜色 • 报警值 • 报警条件 • 报警颜色	• 报警值 您可以在此处指定报警的阈值。 • 报警条件 如果数组元素的当前值满足报警条件，则会设置报警。如已设置“Less”（小于），则在值小于阈值时触发报警。如已设置“More”（大于），则在值大于阈值时触发报警。 • 报警颜色 您可以在此处选择在发生报警时用于显示单个条形的颜色。
使用颜色区域	如果启用该选项，则会显示在颜色区域下定义的颜色区域。
颜色区域 • [<n>]	点击 Create new 按钮，生成新的颜色区域。为每个颜色区域创建 1 个区域，其中涵盖相应的设置。 [<n>]：数字表示该区域的索引。点击“Delete”（删除），可删除相应的颜色区域及其设置。

区域 [<n>]

区域开始	颜色范围的开始。它必须在规定的标度 [▶ 489]范围内。
区域结束	颜色范围的结束。它必须在规定的标度 [▶ 489]范围内。
颜色	条形区域的颜色

Windows CE 不支持透明度。

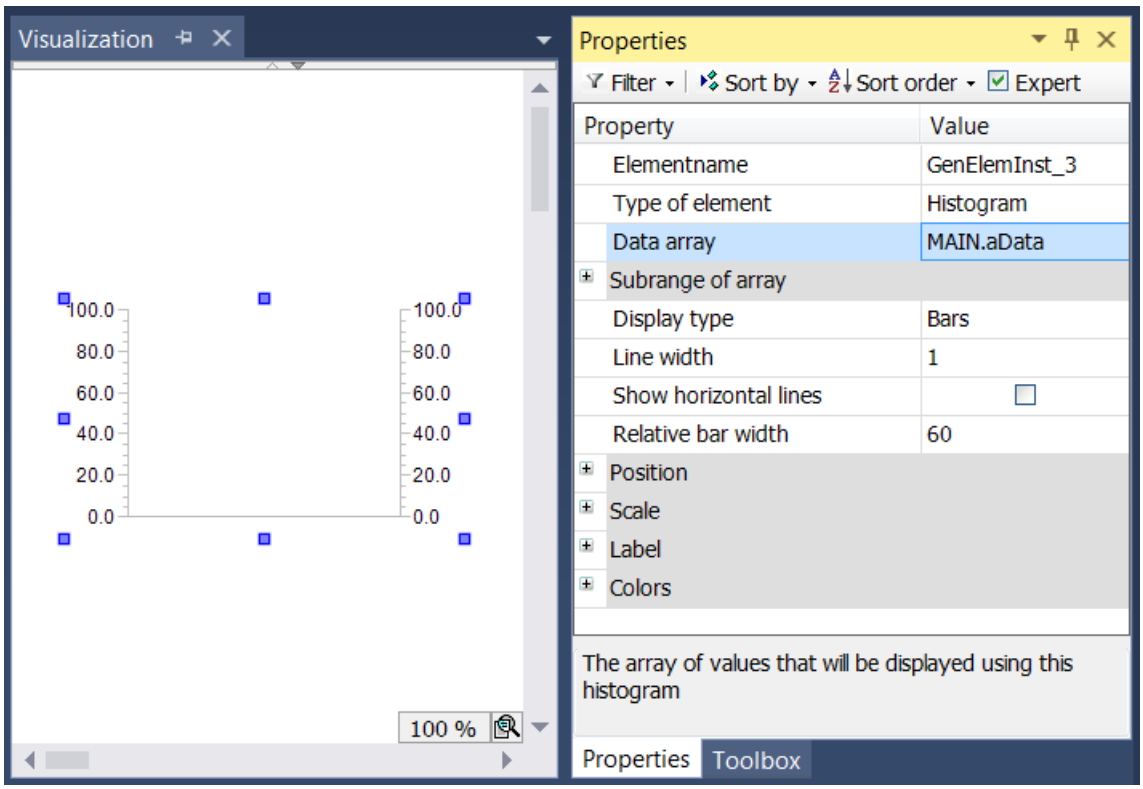
访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

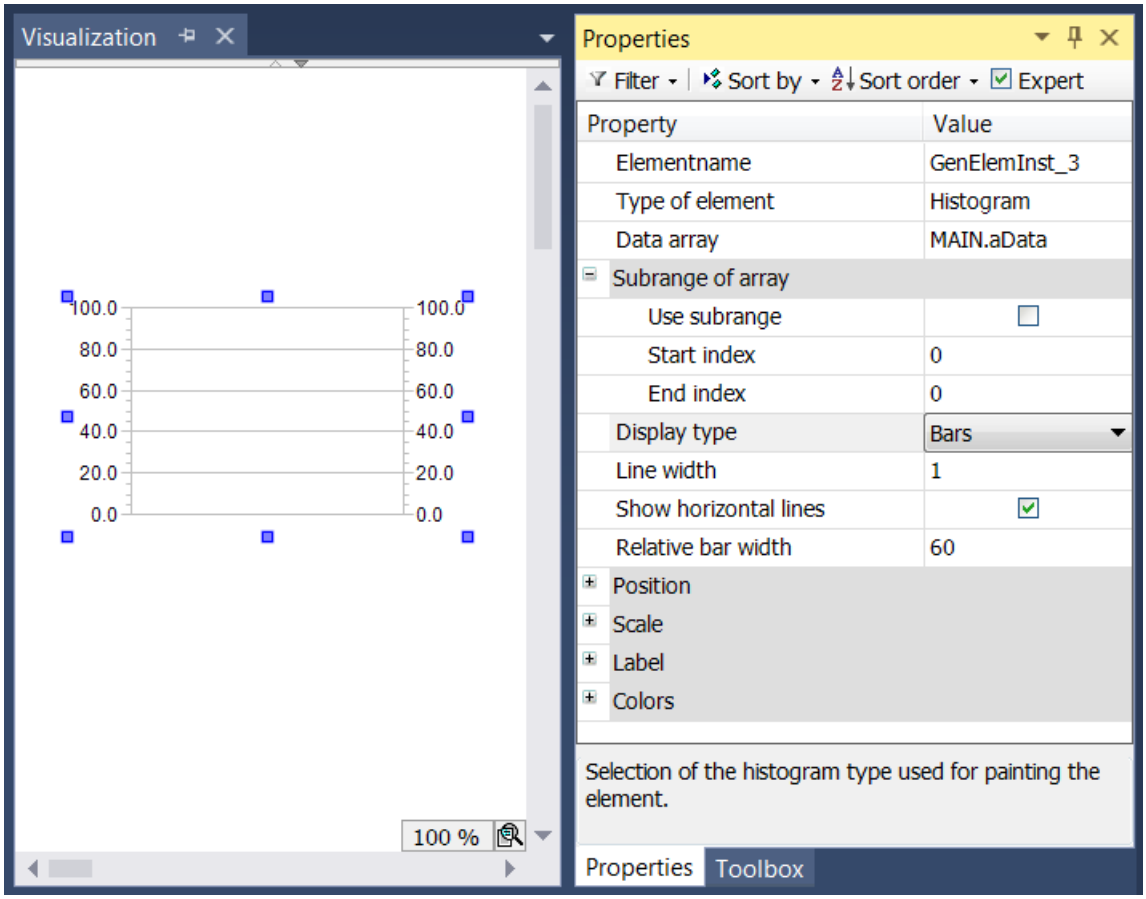
未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

15.8.5.6.1 直方图的配置

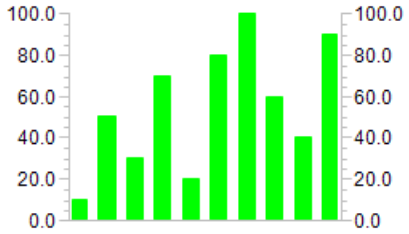
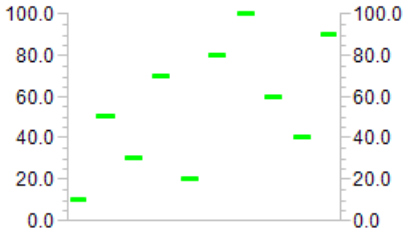
下一部分将根据 1 个示例解释直方图的配置。



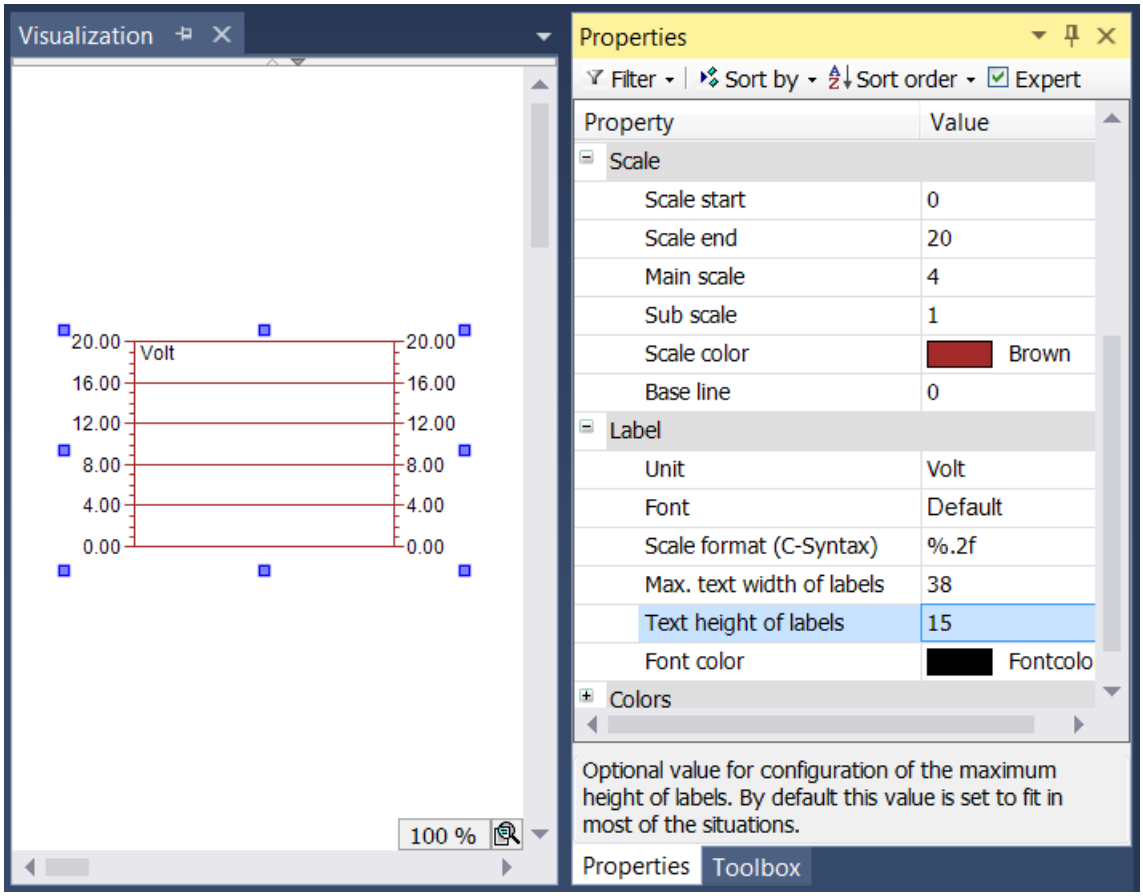
在直方图元素的元素属性中必须指定分配的数据数组（在本示例中为名称为“aData”的数组）。在点击进入“Data array”（数据数组）属性的输入字段之后，即可使用 ... 按钮。它可用于在项目中搜索变量。务必在设备树中指定输入变量及其完整路径。



如果不显示数组的所有元素，而只显示从“Start index”（起始索引）到“End index”（结束索引）之间的元素，则可使用“Use subrange”（使用子范围）选项。此外，数组数据的显示类型可以以 3 种不同的形式显示：

条	
线	
曲线	

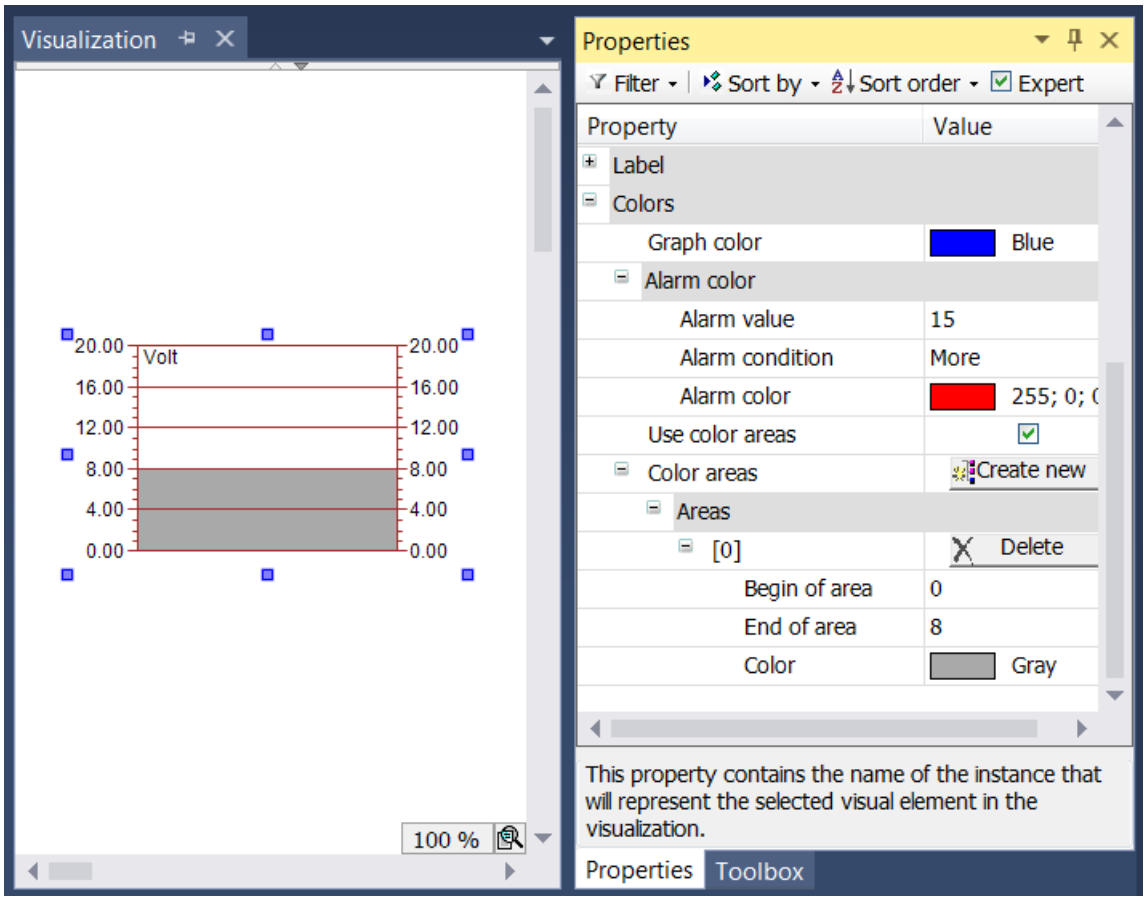
在本直方图示例中，在主标度处绘制水平线。



在标度属性中可以定义标度的数值范围以及主标度和子标度。标度范围受”Scale start“（标度开始）和”Scale end“（标度结束）中指定值的限制。因此，”Scale start“（标度开始）的值必须小于”Scale end“（标度结束）的值。Main scale（主标度）和 Sub scale（子标度）的值可被设置为 0，以禁用标度线

的显示：如果主标度的值被设置为 0，则无论将子标度的值设置为多少，都不会绘制子标度线。如果子标度的值被设置为 0，那么，只会绘制主标度的标度线。如要更改标度颜色，可点击标度颜色的字段。1 个对话框打开，您可以从中选择颜色。

在 Label（标签）部分中可以指定标度显示。Unit（单位）输入字段中的条目可用于指定标度单位。它会在直方图的左上角显示。在选择合适的字体和字体颜色之后，您可以调整标签格式。务必按照 C 语言的语法指定数值的格式。“%d”可用于整数值，“%.Xf”可用于浮点数，其中“X”应被替换为所需的小数位数。



最后，在“Alarm color”（报警颜色）部分中可以指定元素颜色。首先，为直方图选择颜色。在“Alarm color”（报警颜色）子部分中可以定义 1 种报警颜色，如果数组元素的当前值满足报警条件，则会以该报警颜色显示单个条形。该条件由报警值的定义所引起，值必须不超过报警值（如果报警条件被设置为“More”（大于）），或者值必须不低于最小值（如果报警条件被设置为“Less”（小于））。

使用 Create new 按钮创建颜色区域，可为标度的子区域分配 1 个特定的颜色。颜色区域按升序编号。在元素属性中，可为每个颜色区域分配其自己的输入字段。

在“Begin of area”（区域开始）和“End of area”（区域结束）下可以指定子区域的界限。通过下拉菜单可以选择颜色。在创建颜色区域之后，点击相应的 Delete 按钮，即可将其删除。

15.8.6 特殊控制

15.8.6.1 花朵等待图标

该动画元素可用于表示系统繁忙或系统正在等待数据。

属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用 ... 按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

颜色

颜色根据由红/绿/蓝（RGB）组件组成的十六进制数进行定义。这 3 种颜色各有 256（0-255）个值，您可以

在此处静态输入。从选择列表中或通过颜色选择对话框可以选择颜色，通过按钮 ... 可以打开颜色选择对话框。此外，还可以为每种颜色设置透明度级别（0：完全透明，255：完全不透明）。

边框颜色	为元素选择边框颜色。
填充颜色	为元素选择填充颜色。

Windows CE 不支持透明度。

外观

Appearance（外观）下的设置是对边框和元素填充的静态定义。

线宽	以像素为单位定义边框线宽。0 的代码与 1 相同，并将线宽设置为 1 像素。如果不需要边框，则将线条类型设置为不可见。
填充类型	定义填充颜色的填充类型： • 填充：填充颜色可见 • 不可见：填充颜色不可见
线条类型	为轮廓定义以下 1 种线条类型： • 实线 • 虚线 • 点 • 点划线 • 双点划线 • 不可见：轮廓不可见。

图标颜色	您可以在此处选择用于显示花朵图标的颜色。
图标线宽	您可以在此处指定 1 个数字来表示图标线条的宽度。0 的代码与 1 相同，并将线宽设置为 1 像素。

状态变量

这些是对元素在在线模式下的可用性的动态定义。

不可见	Boolean 变量。如果返回 TRUE，则元素在在线模式下不可见。

访问权限

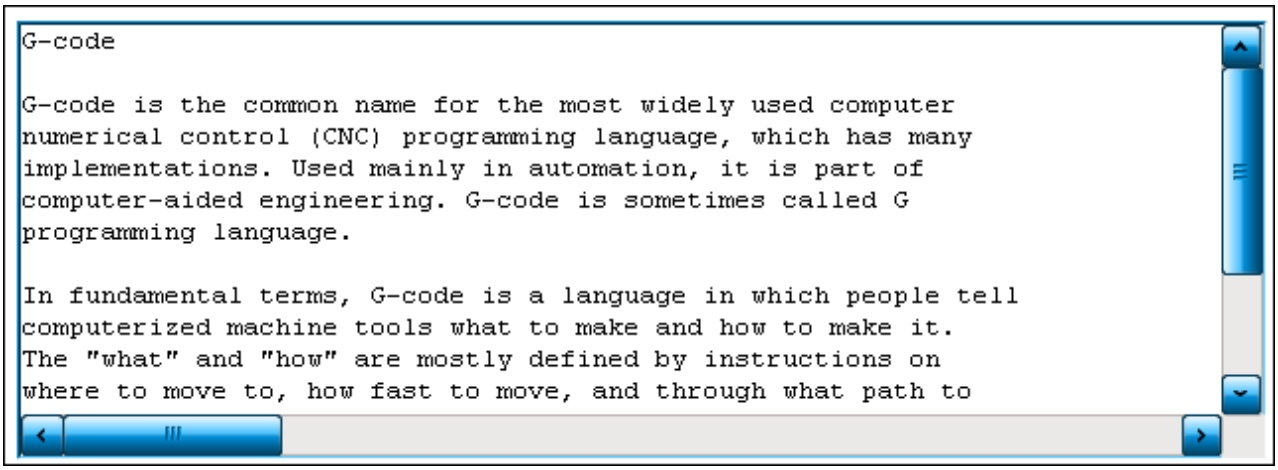
该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

15.8.6.2 文本编辑器

可视化元素文本编辑器可用于显示和编辑控制器上的 ASCII 或 Unicode 文本文件的内容。

- 导航的用户输入 [▶ 533]
- 配置举例 [▶ 533]



文本编辑器元素只能与 PLC HMI [▶ 551] 和/或 PLC HMI Web [▶ 555] 组合使用。

属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在 View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用 ... 按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

字体

名称	在系统中安装的字体的名称。在文本编辑器中输出的文本将使用这种字体。务必使用 Courier New 等非比例字体。
大小	字体大小 示例：12

控制变量

文件

文件名	STRING 变量，包含文件名和路径（必要时）
打开	Boolean 变量，用于控制在“File name”（文件名）中指定的文件的打开。如果该变量被设置为 TRUE，则会打开文件。
关闭	Boolean 变量，用于控制文件的关闭。如果该变量被设置为 TRUE，则会关闭文件。
保存	Boolean 变量，用于控制文件的保存。如果该变量被设置为 TRUE，则会保存文件。
新建	Boolean 变量，用于控制新文件的生成。如果该变量被设置为 TRUE，则会使用在“File name”（文件名）中定义的名字创建 1 个新文件。

编辑

搜索	STRING 变量，包含搜索词
查找	Boolean 变量，可控制对搜索词的搜索。如果该变量被设置为 TRUE，则会开始搜索。
查找下一个匹配项	Boolean 变量，用于控制对搜索词的下一个匹配项的搜索。如果该变量被设置为 TRUE，则从当前位置开始搜索。

插入符位置

行	整数变量，只要“Trigger setting”（触发器设置）为 FALSE，则包含当前行号。
列	整数变量，只要“Trigger setting”（触发器设置）为 FALSE，则包含当前列号
位置	整数变量，只要“Trigger setting”（触发器设置）为 FALSE，则包含连续编号中的当前插入符位置。
触发器设置	Boolean 变量，可控制插入符位置。如果该变量为 FALSE，则“Row”（行）、“Column”（列）和“Position”（位置）中的变量包含当前位置。如果该变量为 TRUE，则将插入符设置为在“Row”（行）和“Row”（列）中指定的位置。

选择

起始位置	整数变量，包含连续编号中的文本选择的起始位置。
结束位置	整数变量，包含连续编号中的文本选择的结束位置。
起始行号	整数变量，包含在开始文本选择时的行号。
起始列索引	整数变量，包含在开始文本选择时的列索引。
结束行号	整数变量，包含在结束文本选择时的行号。
结束列索引	整数变量，包含在结束文本选择时的列索引。
待选行	整数变量，用于确定要选择的行
触发器选择	Boolean 变量，用于控制文本选择。如果该变量被设置为 TRUE，则会按照“Row to be selected”（待选行）中的定义来设置文本选择。

错误处理

错误编号的变量	整数变量，包含出错时的错误编号。这些编号在“VisuElemTextEditor”库的 GVL_ErrorCodes 部分中进行声明。如要获取与错误编号相关的错误文本，可调用库中包含的函数 VisuFctTextEditorGetErrorText()。

为内容更改的变量	Boolean 变量。如果变量值为 TRUE，则文本编辑器的内容已更改。
访问模式的变量	Boolean 变量。如果变量值为 TRUE，则文件仅具有读取访问权限。如果变量值为 FALSE，则文件具有读取和写入访问权限。

最大行长	整数变量，用于指定 1 行的最大长度
编辑模式	您可以在此处选择编辑模式： • 只读 仅可读取文件。在该模式下，如果控制器正在运行，则编辑器会显示浅灰色背景。 • 读/写 可以读取和写入文件。

新文件

字符编码	您可以在此处指定创建新文件时的字符编码： • ASCII • Unicode（小端模式） • Unicode（大端模式）
行结束符	如果创建新文件，则必须在此处定义行结束符： • CR/LF：Windows 系统的标准 • LF：UNIX 系统的标准 如果打开现有文件，则会自动识别并使用要打开的文件的行结束符。

访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

15.8.6.2.1 在线模式下导航的用户输入

文本编辑器支持众所周知的用户输入，以便在文本中进行导航。这些功能仅可在在线模式下使用。

[Left arrow]	将插入符向左移动 1 个字符。如果当前插入符位置在行首，则会将其移动到上一行（如果有）的末尾。
[Right arrow]	将插入符向右移动 1 个字符。如果当前插入符位置在行尾，则会将其移动到下一行（如果有）的开头。
[Up arrow]	将插入符移动到上一行。
[Down arrow]	将插入符移动到下一行。
[Home]	将插入符移动到当前行首。
[End]	将插入符移动到当前行尾。
[PgUp]	显示上一页。
[PgDn]	显示下一页。
[Shift] + [Left arrow] [Shift] + [Right arrow]	文本选择
在选择文本时，[Del]	所选文本会被删除。如果未选择任何内容，则会删除位于插入符位置右侧的字符。
[Enter]	创建 1 个新行，并将插入符设置为新行的开头。
[Ctrl] + [Tab]	制表符会被插入。目前，它表示为 1 个空格。
[Tab]	将焦点移动到下一个可视化。

15.8.6.2.2 配置文本编辑器

如要访问文本编辑器 [▶ 530]支持的控制变量，则需要在可视化中添加附加的可视化元素，并通过控制变量将其与文本编辑器关联起来。

下面以文本编辑器的加载功能为例，对该程序进行说明：

1. 例如，在 MAIN 的 IEC 代码中声明元素“Load”（加载）的控制变量：

```
PROGRAMM MAIN
VAR
    bOpen : BOOL;
END_VAR
```

2. 例如，在 MAIN 中的 IEC 代码中声明文本编辑器文件名的变量：

```
PROGRAMM MAIN
VAR
    bOpen : BOOL;
    sFileName : STRING;
END_VAR
```

3. 为可视化及其配置添加矩形元素：
   - 文本 → 文本：加载
   - 输入配置 → 按键 → 变量：MAIN.bOpen
4. 为可视化及其配置添加文本编辑器元素：
   - 控制变量 → 文件 → 文件名：MAIN.sFileName
   - 控制变量 → 文件 → 打开：MAIN.bOpen

文本编辑器的属性中的所有控制变量都可以通过这种方式进行关联。

错误消息

如要输出在控制变量 → 错误处理 → 错误编号的变量中可用的错误编号的错误文本，可对 IEC 代码中的函数调用 VisuFctTextEditorGetErrorText() 进行编程：

1. 声明所需的变量：

```
PROGRAMM MAIN
VAR
…
nErrorCode : USINT;
sErrorMessage : STRING(255);
END_VAR
```

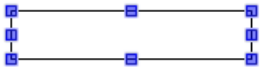
2. 执行函数调用：

```
sErrorMessage := VisuFctTextEditorGetErrorText(nErrorCode);
```

3. 在可视化中输出错误消息，例如，在具有以下属性的矩形元素中：
   - 文本 → 文本：%s
   - 文本变量 → 文本变量：MAIN.sErrorMessage

15.8.6.3 ActiveX 元素

ActiveX 元素可以集成现有的 ActiveX 组件。



ActiveX 控制器只能与集成的可视化 [▶ 550]组合使用，而不能与 PLC HMI [▶ 551] 或 PLC HMI Web [▶ 555] 组合使用。

属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在 View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用 … 按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

辅助设置

元素	在此处可以分配已安装的 ActiveX 组件。使用 … 按钮，通过输入向导选择组件。输入字符串类型的变量，描述组件的名称或 ID。

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

绝对移动

通过 1 个整数变量改变元素左上角的 x 和 y 位置（像素），从而可以移动元素。这里使用的是绝对坐标值。

运动 • X • Y	X：此处输入的整数变量可以定义元素左上角当前的 x 位置（单位：像素）。它可用于沿 x 方向移动元素。（正值会将元素从左至右移动）。 Y：此处输入的整数变量可以定义元素左上角当前的 y 位置（单位：像素）。它可用于沿 y 方向移动元素。（正值会将元素从上至下移动）。

状态变量

这些是对元素在在线模式下的可用性的动态定义。

不可见	Boolean 变量。如果返回 TRUE，则元素在在线模式下不可见。

首次调用

在该属性下可以定义在初始化过程中执行的方法调用。仅在第 1 个周期中执行它们。

使用 Create new 按钮，创建新的方法调用和相关参数。 Delete 按钮可用于删除相应的方法调用及所有设置，例如，参数和结果参数。

[<Number>]	为方法调用建立索引的编号
方法	要调用的方法的名称
参数	参数变量
结果参数	要包含结果的变量。基本上，每个方法都有 1 个结果参数。

循环调用

在该属性下可以定义在每个周期中执行的方法调用。在可视化的更新间隔期间可以调用它们。

使用 Create new 按钮，创建新的方法调用和相关参数。 Delete 按钮可用于删除相应的方法调用及所有设置，例如，参数和结果参数。

[<Number>]	为方法调用建立索引的编号
方法	要调用的方法的名称
参数	参数变量
结果参数	要包含结果的变量。基本上，每个方法都有 1 个结果参数。

条件调用

在该属性下可以定义仅在特定条件下执行的方法调用。在可视化的更新间隔期间可以调用它们。与循环调用相比，在属性“Call condition”（调用条件）下会定义调用条件。只有在出现调用条件的上升沿时，才会执行它们。

使用 Create new 按钮，创建新的方法调用和相关参数。 Delete 按钮可用于删除相应的方法调用及所有设置，例如，参数和结果参数。

[<Number>]	为方法调用建立索引的编号
方法	要调用的方法的名称
调用条件	Boolean 变量。仅在出现该变量的上升沿时调用 1 次分配的方法。
参数	参数变量
结果参数	要包含结果的变量。基本上，每个方法都有 1 个结果参数。

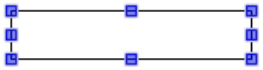
访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

15.8.6.4 Webbrowser

Webbrowser 元素会显示通过 URL 地址指定的内容。除了 HTML 网页之外，它还可以加载视频和 PDF 文档。有关配置示例，请参见“配置 [▶ 538]”部分。



属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在 View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用  按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

绝对移动

通过 1 个整数变量改变元素左上角的 x 和 y 位置（像素），从而可以移动元素。这里使用的是绝对坐标值。

运动 • X • Y	X：此处输入的整数变量可以定义元素左上角当前的 x 位置（单位：像素）。它可用于沿 x 方向移动元素。（正值会将元素从左至右移动）。 Y：此处输入的整数变量可以定义元素左上角当前的 y 位置（单位：像素）。它可用于沿 y 方向移动元素。（正值会将元素从上至下移动）。

状态变量

这些是对元素在在线模式下的可用性的动态定义。

不可见	Boolean 变量。如果返回 TRUE，则元素在在线模式下不可见。

控制变量

URL	字符串类型的变量，用于存储要打开的网站的 URL 地址 示例：sUrlAddress : STRING := ,'http://www.beckhoff.com/';
显示	Boolean 变量。如果变量包含 1 个上升沿，则可视化会调用在 URL 中指定的网页，并在元素的边框中显示其内容。
后退	Boolean 变量。如果变量包含 1 个上升沿，则可视化会显示先前显示页面的内容。
前进	Boolean 变量。如果变量包含 1 个上升沿，则可视化会显示在后退导航之前的内容。

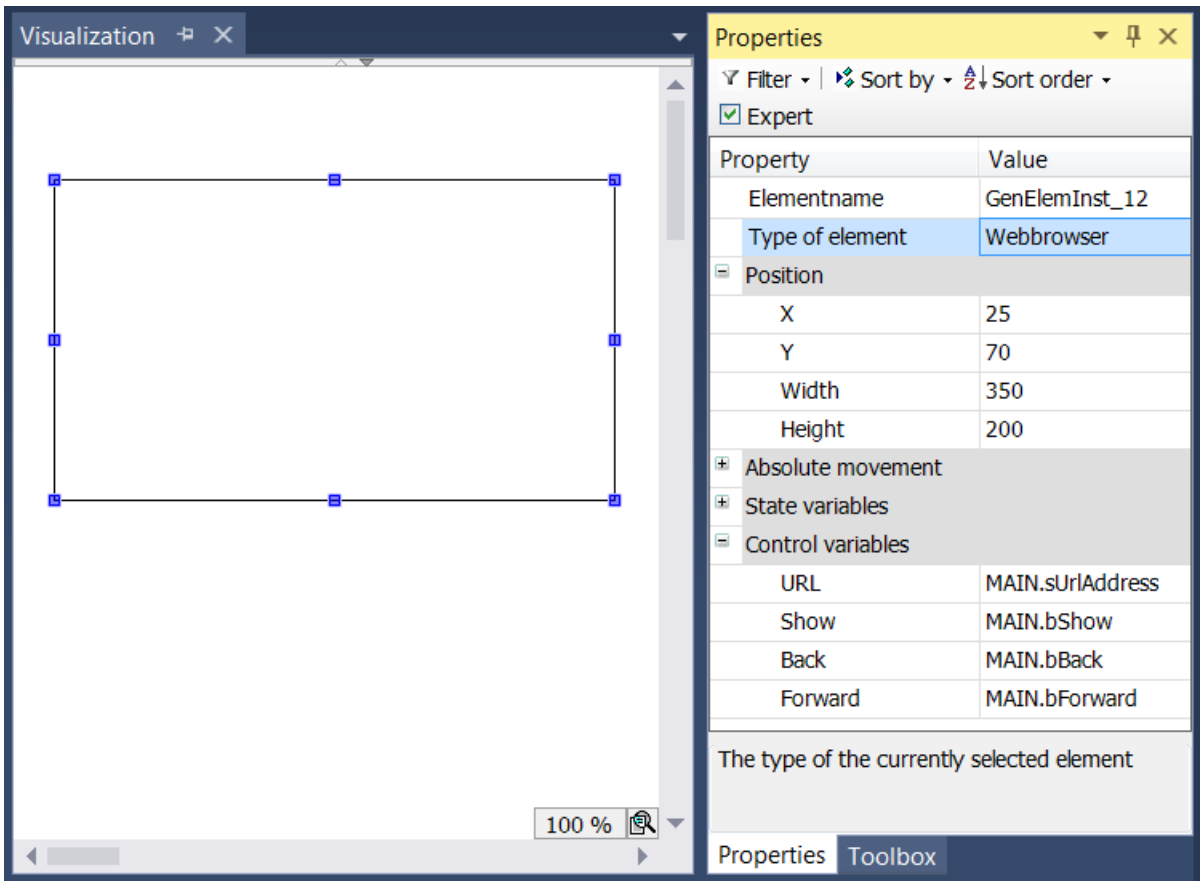
访问权限

该设置与单个元素的访问权限有关。点击打开访问权限对话框 [▶ 384]。只有在 PLC 项目中添加用户管理 [▶ 360]后，该设置才可用。可提供以下状态消息：

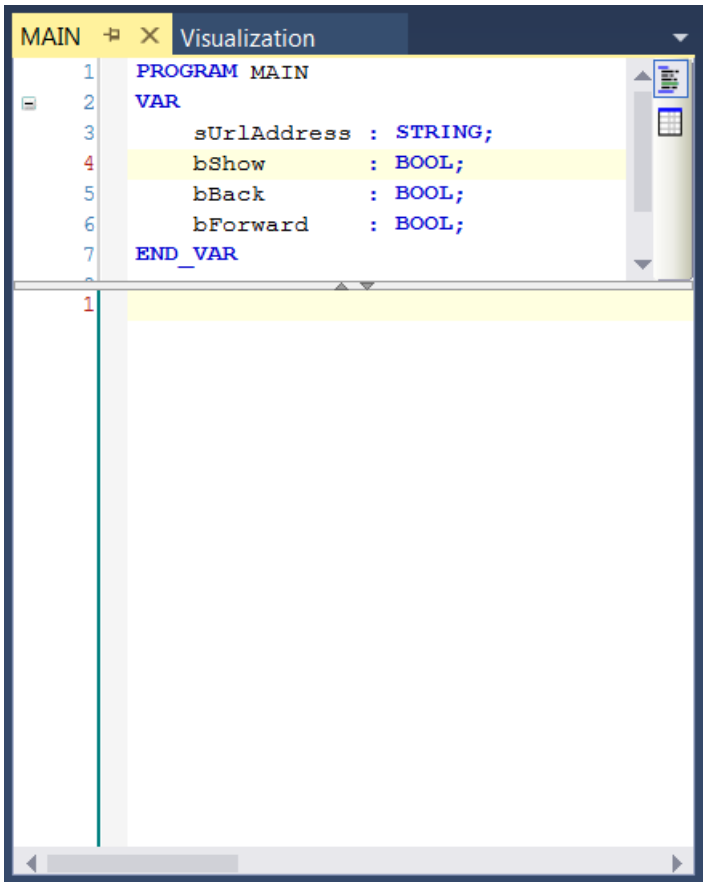
未设置。所有权限。	如果元素显示为可用于所有组，则设置默认消息。
已发布权限：有限的权限。	如果元素在至少 1 个组中显示有限的行为，则设置该消息。

15.8.6.4.1 配置

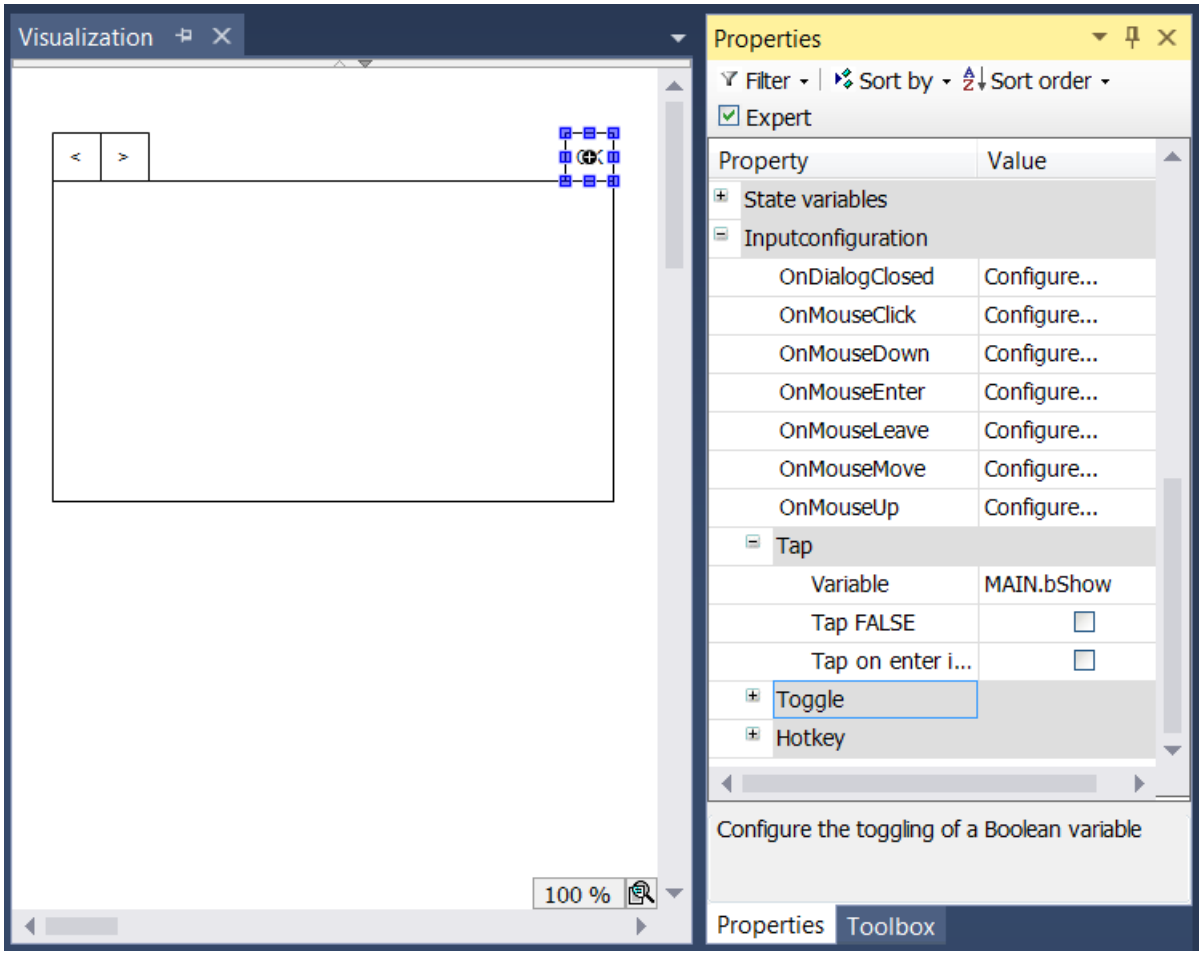
下一部分将根据 1 个示例解释 Webbrowser 元素的配置。



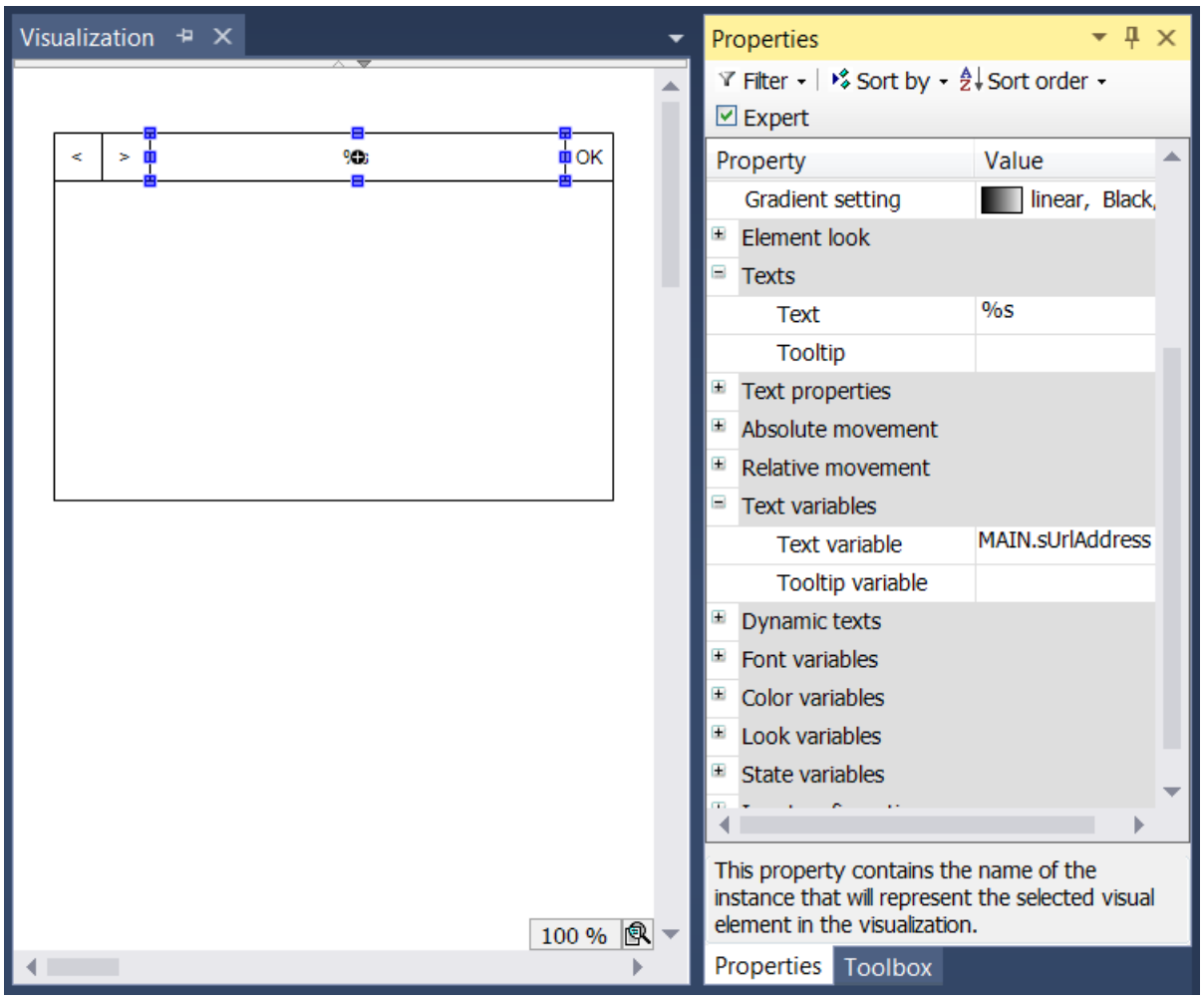
在将 Webbrowser 元素从工具箱拖到可视化页面 [▶ 367]之后，通过属性“Position”（位置）可以修改该元素的位置和大小。此外，在“控制变量 [▶ 538]”中还可以指定用于操作浏览器的变量。在本示例中，输入的是在“MAIN”程序中声明的变量“sUrlAddress”、“bShow”、“bBack”和“bForward”：



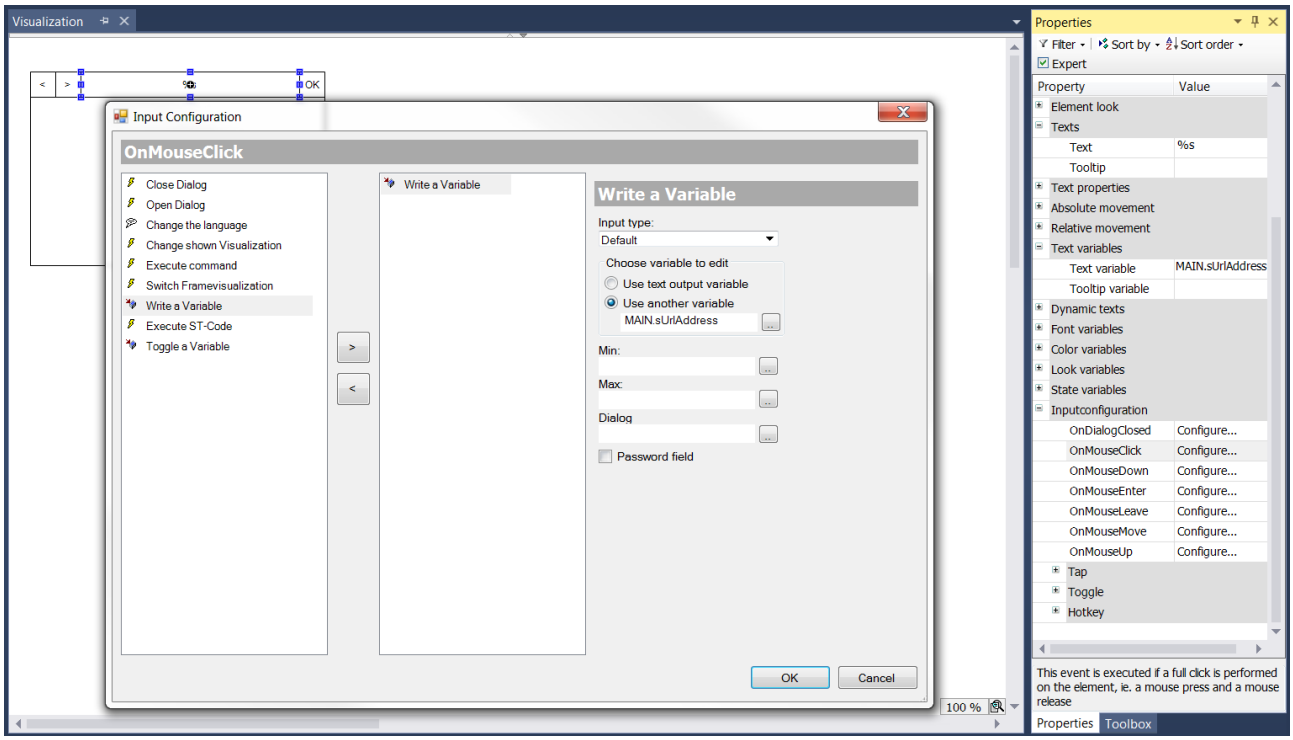
为了能够更改在“MAIN”程序中声明的可视化中的变量，应将前 3 个矩形元素拖到可视化中。在输入配置中，带有文本“<”的第 1 个元素与用于切换功能的“bBack”相关联，带有文本“>”的第 2 个元素与“bForward”相关联，带有文本“OK”的第 3 个元素与“bShow”相关联。



然后，添加第 4 个矩形元素，以便能够更改网络浏览器的 URL 地址。在属性“Text”（文本）中输入占位符“%s”，并在“Text variable”（文本变量）中输入“sUrlAddress”，以便元素可以显示地址。

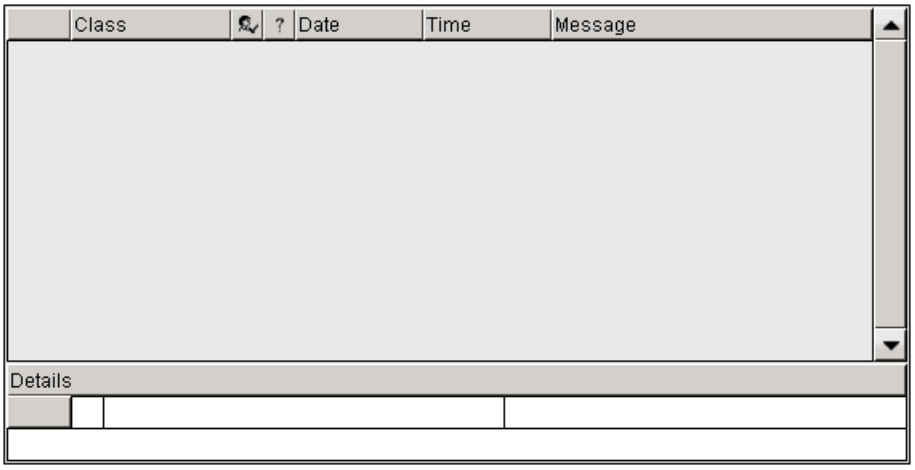


在输入配置中使用“OnMouseClick”事件，以便能够在可视化中更改 URL 地址。在这里添加“Write variable”（写入变量）动作，并与变量“sUrlAddress”相关联。



15.8.6.5 事件表格

元素事件表格可用于在可视化页面上以表格形式显示来自 TcEventLogger 和 TwinCAT 3 EventLogger 的事件。通过功能块“FB_AdsReadEvents”(TcEventLogger) 或“FB_ReadTc3Events [▶ 544]”(TwinCAT 3 EventLogger) 读取事件，并通过数组将其转发给可视化元素。有关元素配置的示例，请参见“事件表格配置 [▶ 546]”部分。Build 4020.0 及以上版本可提供该元素。事件表格可与 Build 4026.0 及以上版本的 TwinCAT 3 EventLogger 一起使用。



属性编辑器

在属性编辑器 [▶ 353]中可以对可视化元素的所有属性（除了对齐方式和顺序 [▶ 345]之外）进行配置。默认情况下，该编辑器会在可视化编辑器旁边打开，或者通过“Properties”（属性）命令（作为标准配置，在 View（视图）菜单中可以找到该命令）也可以明确打开该编辑器。

通过编辑“Value”（值）字段可以修改属性。为此，根据元素类型可以在该字段中提供输入字段、选择列表、对话框或可以激活的复选框。在以下情况下可以打开值字段

- 在双击之后，
- 在单击进入所选字段之后，
- 如果字段已被选中，则通过空格键进行操作。

如果已经分配变量，

- 只需输入其名称即可。
- 使用  按钮打开输入助手，以选择变量。Variables（变量）类别列出了已在项目中定义的所有变量。

借助默认值、排序和筛选功能，属性列表中的工作可以变得更加简单。

元素属性

所有元素属性及其说明如下。

元素名称	元素名称可以更改。标准名称为“GenElemInst_x”。“x”代表 1 个序号。
元素类型	在此处输入元素类型。对于 3 个元素组，通过更改元素类型可以在相应元素之间进行切换： • 矩形、圆角矩形和椭圆形 [▶ 433] • 多边形、折线或贝塞尔曲线 [▶ 447] • 拨码开关、电源开关、按压式开关、带 LED 的按压式开关、摇杆开关 [▶ 484]

消息数据数组	为此处使用的功能块分配输出变量“aEvents”：如果使用的是 TcEventLogger，则分配“FB_AdsReadEvents”；如果使用的是 TwinCAT 3 EventsLogger，则分配“FB_ReadTc3Events [▶ 544]”。将活跃事件保存在数组中。

位置

您可以在此处定义元素的位置（X/Y 坐标）和大小（宽度和高度），单位：像素。原点在窗口的左上角。正 x 轴在右侧，正 y 轴朝下。如果对值进行编辑，则会在可视化编辑器 [▶ 344]中同时修改显示的元素。

X	水平位置（单位：像素） - X=0 为窗口的左边缘。
Y	垂直位置（单位：像素） - Y=0 为窗口的上边缘。
宽度	元素的宽度（单位：像素）
高度	元素的高度（单位：像素）

列

元素 TcEventTable 包含 1 个有以下 7 个列的表格：

- 索引：索引表示事件发生顺序的编号。
- 类：类表示事件的类型。它是在使用 TcEventLogger 创建事件时定义的。当使用 TwinCAT 3 EventLogger 时，可在这里对消息和警报进行区分。
- 确认状态：可以创建具有确认要求的事件（仅适用于 TwinCAT 3 EventLogger 中的警报）。可在程序代码中或元素中进行确认。该列中会显示素是否（仍）待确认的状态。
- 重置状态：事件激活后可在程序代码中进行重置（TwinCAT 3 EventLogger：仅限警报）。该行表示事件是否已重置。
- 日期、时间：“Date”（日期）和“Time”（时间）这两列会显示事件发生的日期和时间。
- 消息：最后一列会显示实际事件文本。

列	7 个列的设置选项
行高	行高（单位：像素）
滚动条宽度	滚动条的宽度（单位：像素）
排序顺序	您可以在此处设置事件在元素中显示的顺序： • 最新优先 • 最旧优先

列

列宽	列宽（单位：像素）
文本属性	您可以在此处修改列的文本属性： • 水平对齐方式 • 垂直对齐方式 • 字体 • 字体颜色

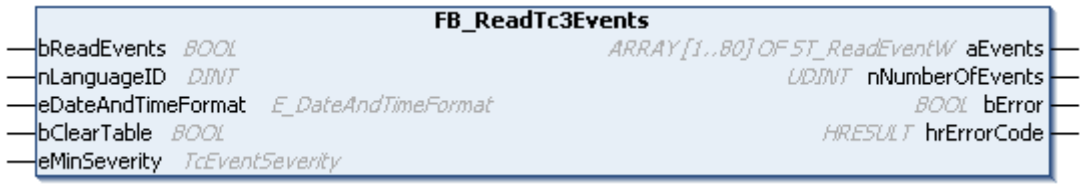
详细信息属性

除了显示事件的表格之外，元素 TcEventTable 还有 1 个详细信息字段。如果在运行时选择了表格中的某个事件，则会在详细信息字段中显示该事件的更多信息。可以通过“确认”按钮  确认该事件（TwinCAT 3 EventLogger：仅限警报）。

一般文本属性	您可以在此处为消息单元格以外的所有详细信息单元格设置文本属性： • 水平对齐方式 • 垂直对齐方式 • 字体 • 字体颜色
消息文本属性	您可以在此处为消息单元格设置文本属性： • 水平对齐方式 • 垂直对齐方式 • 字体 • 字体颜色

15.8.6.5.1 FB_ReadTc3Events

TwinCAT 3.1 Build 4026 及以上版本可用



该功能块可令 TwinCAT 3 EventLogger 的事件显示在可视化元素事件表格 [▶ 542]中。FB_ReadTc3Events 位于 VisuElemEventTable 库中，当激活事件表格时，该库将自动添加到项目中（参见“事件表格的配置 [▶ 546]”）。若要使用 FB_ReadTc3Events，必须在项目中添加 TargetVisualization 或 WebVisualization 对象。

功能块 FB_ReadTc3Events 会查询 TC3 EventLogger 发生的事件，并以数组 aEvents 的形式提供这些事件。在内部，FB_ListenerBase2 的实现便用于此目的。若要在事件表格中显示消息，必须在其消息数据数组 [▶ 542]属性中输入数组 aEvents。

输入

```
VAR_INPUT
    bReadEvents          : BOOL;
    nLanguageID          : DINT;
    eDateAndTimeFormat   : E_DateAndTimeFormat;
    bClearTable          : BOOL;
    eMinSeverity         : TcEventSeverity;
END_VAR
```

名称	类型	描述
bReadEvents	BOOL	此输入用于启用待读取的事件。
nLanguageID	DINT	定义要检索的事件文本的翻译。
eDateAndTimeFormat	E_DateAndTimeFormat	定义时间戳的格式。提供以下选项： • de_DE - 德国符号：dd.MM.yyyy hh:mm:ss（24 小时制） • en_GB - 英国符号：dd/MM/yyyy hh:mm:ss（24 小时制） • en_US - 美国符号：MM/dd/yyyy hh:mm:ss（12 小时制）
bClearTable	BOOL	通过该输入端的正沿清除事件表格。不会从表格中删除未确认或未重置的警报。
eMinSeverity	TcEventSeverity	事件表格中仅会显示严重性至少与此处指定的严重性相同的事件。

输出

```
VAR_OUTPUT
    aEvents             : ARRAY [1..80] OF ST_ReadEventW;
    nNumberOfEvents     : UDINT;
    bError              : BOOL;
    hrErrorCode         : HRESULT;
END_VAR
```

名称	类型	描述
aEvents	ARRAY [1..80] OF ST_ReadEventW	功能块使用该数组提供已评估的事件。该数组最多可以存储 80 条消息。如果事件的数量较多，则会覆盖事件较早的消息。
nNumberOfEvents	UDINT	表示已触发多少个事件。
bError	BOOL	如果在事件轮询或处理期间发生错误，将会设置此输出。
hrErrorCode	HRESULT	表示已发生错误的错误代码。

方法

SetFilter

您可以将先前配置的 FB_TcEventFilter（链接）类型的筛选器传递给该方法，以指定要评估的事件并在事件表格中显示它们。筛选从调用“SetFilter”方法时开始生效，对已经发生的事件没有影响。通过调用以“NULL”为输入参数的“SetFilter”方法，可以删除筛选。与其他情况一样，取消筛选也适用于新事件。

如果未调用该方法，事件表格将显示所有事件。

```
VAR_INPUT
    ipEventFilter    : I_TcEventFilter;
END_VAR
```

名称	类型	描述
ipEventFilter	I_TcEventFilter	用于指定哪些 TwinCAT 3 EventLogger 事件将由功能块 FB_ReadTc3Events 进行处理的筛选器配置。

通过创建多个 FB_ReadTc3Events 实例，并将事件表格与 aEvents 的引用关联起来（在每种情况下都会将 aEvents 分配给所需的实例），可以在不同的筛选器视图之间进行切换。

示例：

```
PROGRAM MAIN
VAR
    // FB_ReadTc3Events
    fbReadTc3Events     : FB_ReadTc3Events;
    bReadEvents         : BOOL := TRUE;
    nLanguage           : DINT := 1031;
    eDateAndTimeFormat  : Tc2_Utilities.E_DateAndTimeFormat :=
Tc2_Utilities.E_DateAndTimeFormat.de_DE;
    bClear              : BOOL;
    eSeverityMin        : TcEventSeverity;
    bError              : BOOL;
    hr                  : HRESULT;
    hrOccuredErrorCode  : HRESULT;

    // EventFilter
    bFilterInit         : BOOL;
    fbEventFilter       : FB_TcEventFilter;
END_VAR
```

```
// Configure EventFilter (filter possibility one, not necessary for use of FB_ReadTc3Events)
IF NOT bFilterInit THEN
    bFilterInit := TRUE;
    fbEventFilter.Clear();
    fbEventFilter.EventClass.EqualTo(TC_EVENTS.EventClass1.Message1.uuidEventClass).OR_OP().EventCla
ss.EqualTo(TC_EVENTS.EventClass2.Alarm2.uuidEventClass).AND_OP().Severity.GreaterThan(0);
    hr := fbReadTc3Events.SetFilter(ipEventFilter := fbEventFilter);
    IF FAILED(hr) THEN
        hrOccuredErrorCode := hr;
    END_IF
END_IF
```

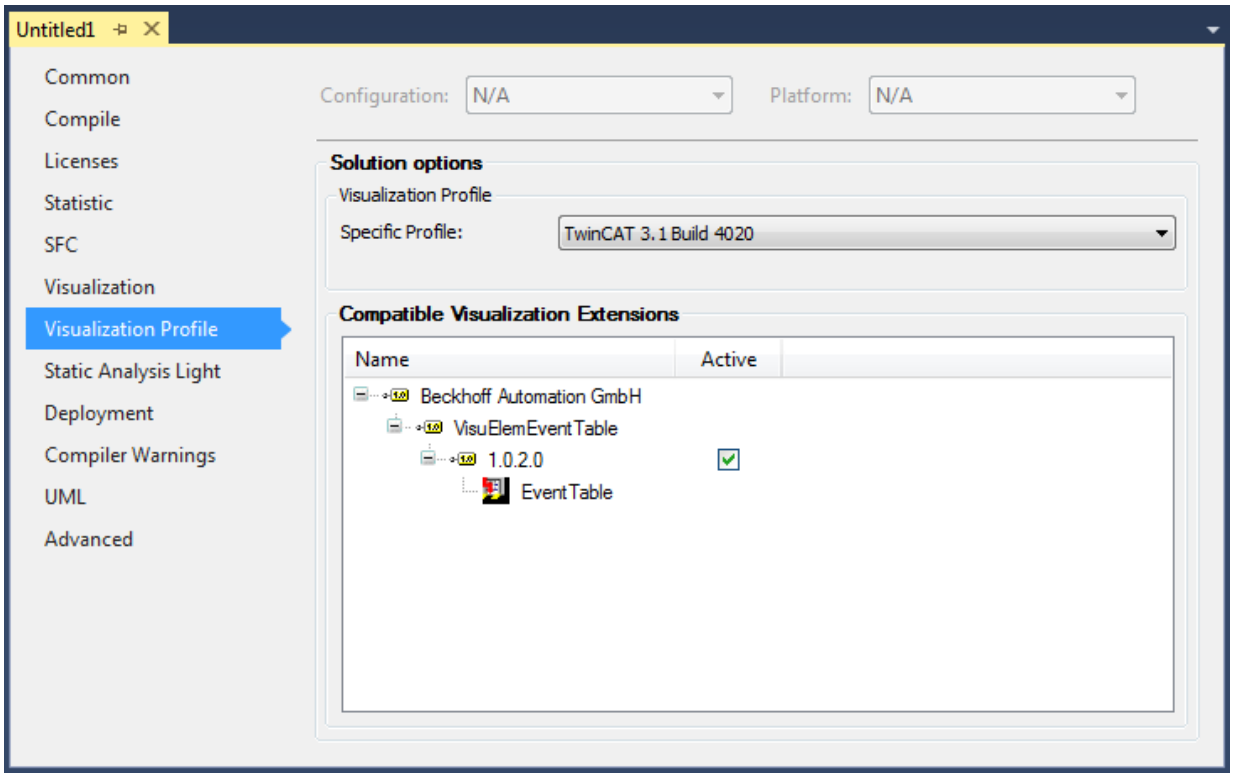
```
// Call fbReadTc3Events
fbReadTc3Events (
    bReadEvents         := bReadEvents,
    nLanguageID         := nLanguage,
    eDateAndTimeFormat  := eDateAndTimeFormat,
    bClearTable         := bClear,
    eMinSeverity        := eSeverityMin,          // Set event filter for severity via input (filter
possibility two)
    nNumberOfEvents     => nEvents1,
    bError              => bError1,
    hrErrorCode         => );
IF fbReadTc3Events.bError THEN
    hrOccuredErrorCode  := fbReadTc3Events.hrErrorCode;
END_IF
```

要求

开发环境	目标平台	需要包含的 PLC 库
TwinCAT v3.1.4026	PC 或 CX（x86、x64、ARM）	VisuElemEventTable

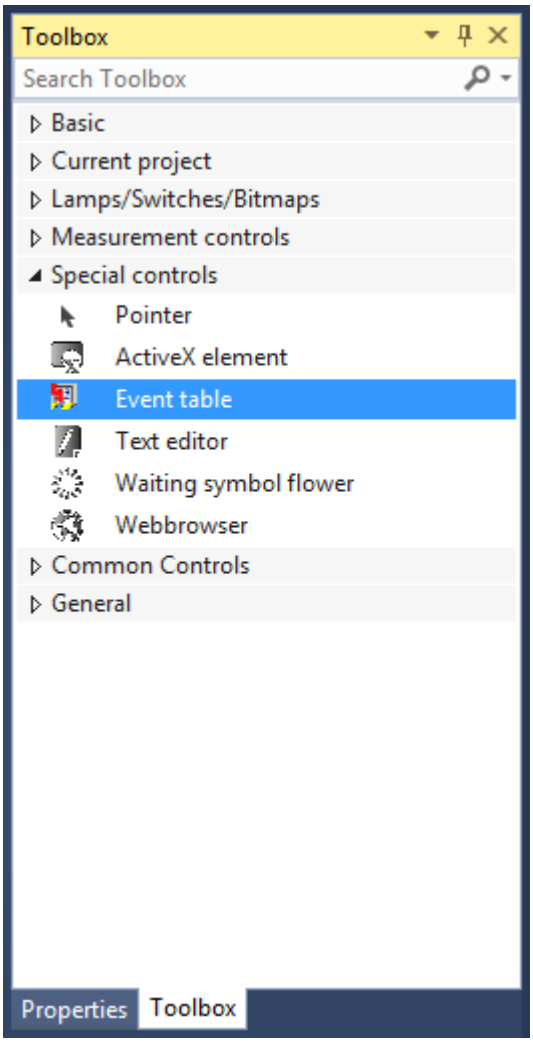
15.8.6.5.2 事件表格的配置

1. 若要在可视化页面中添加可视化元素“Event table”（事件表格），请在 PLC 项目设置 [▶ 367]中的“Visualization Profile”（可视化配置文件）类别下为该元素选择相应的扩展名。



2. 在激活此设置后重新启动 TwinCAT 项目。

⇨ 然后，该元素就会出现在工具箱 [▶ 352]的“Special controls [▶ 528]”（特殊控制）类别中，并且可在可视化页面上使用。



可视化元素“Event table”（事件表格）与功能块“FB_AdsReadEvents”(TcEventLogger) 或“FB_ReadTc3Events [▶ 544]”(TwinCAT 3 EventLogger) 结合使用。

FB_AdsReadEvents 和 TcEventLogger

功能块“FB_AdsReadEvents”位于“Tc2_Utilities”库中。本示例中，该功能块在程序“MAIN”中进行声明和调用。由于要读取本地 TcEventLogger 的消息，因此，可以在输入端“sNetId”输入 1 个空字符串。

```
PROGRAM MAIN
VAR
    fbAdsReadEvents      : FB_AdsReadEvents;
    bReadEvents          : BOOL;
END_VAR
```

```
fbAdsReadEvents (
    sNetId               := ,`,
    bReadEvents          := bReadEvents,
    nLanguageId          := 1031,
    eDateAndTimeFormat   := E_DateAndTimeFormat.de_DE,
    tRefreshTime         := T#1S,
    tTimeout             := T#5s);
```

FB_ReadTc3Events 和 TwinCAT 3 EventLogger

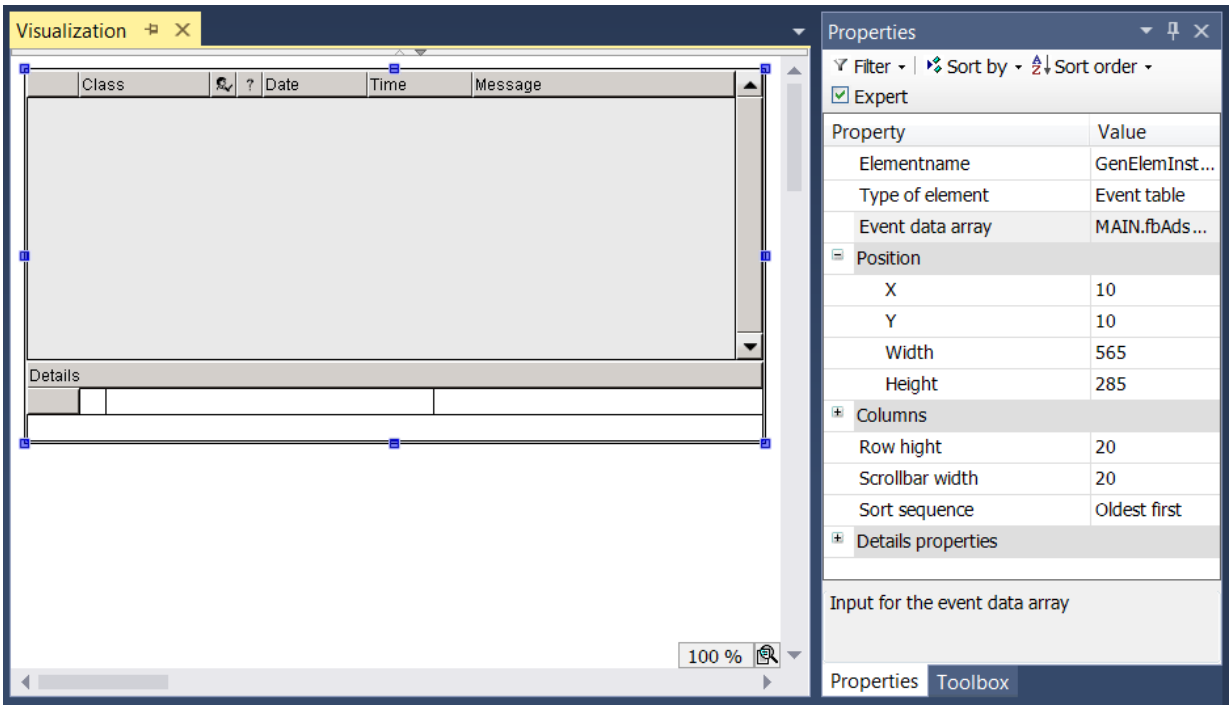
功能块“FB_ReadTc3Events [▶ 544]”位于“VisuElemEventTable”库中，当激活事件表格时，该库将自动添加到项目中。若要将事件表格与 FB_ReadTc3Events 和 TwinCAT 3 EventLogger 一同使用，必须在项目中添加 TargetVisualization 或 WebVisualization 对象。本示例中，该功能块在程序“MAIN”中进行声明和调用。

```
PROGRAM MAIN
VAR
    fbReadTc3Events      : FB_ReadTc3Events;
    bReadEvents          : BOOL;
    bClear               : BOOL;
END_VAR
```

```
fbReadTc3Events (
    bReadEvents          := bReadEvents          ,
    nLanguageID          := 1031,
    eDateAndTimeFormat   := E_DateAndTimeFormat.de_DE,
    bClearTable          := bClear,);
```

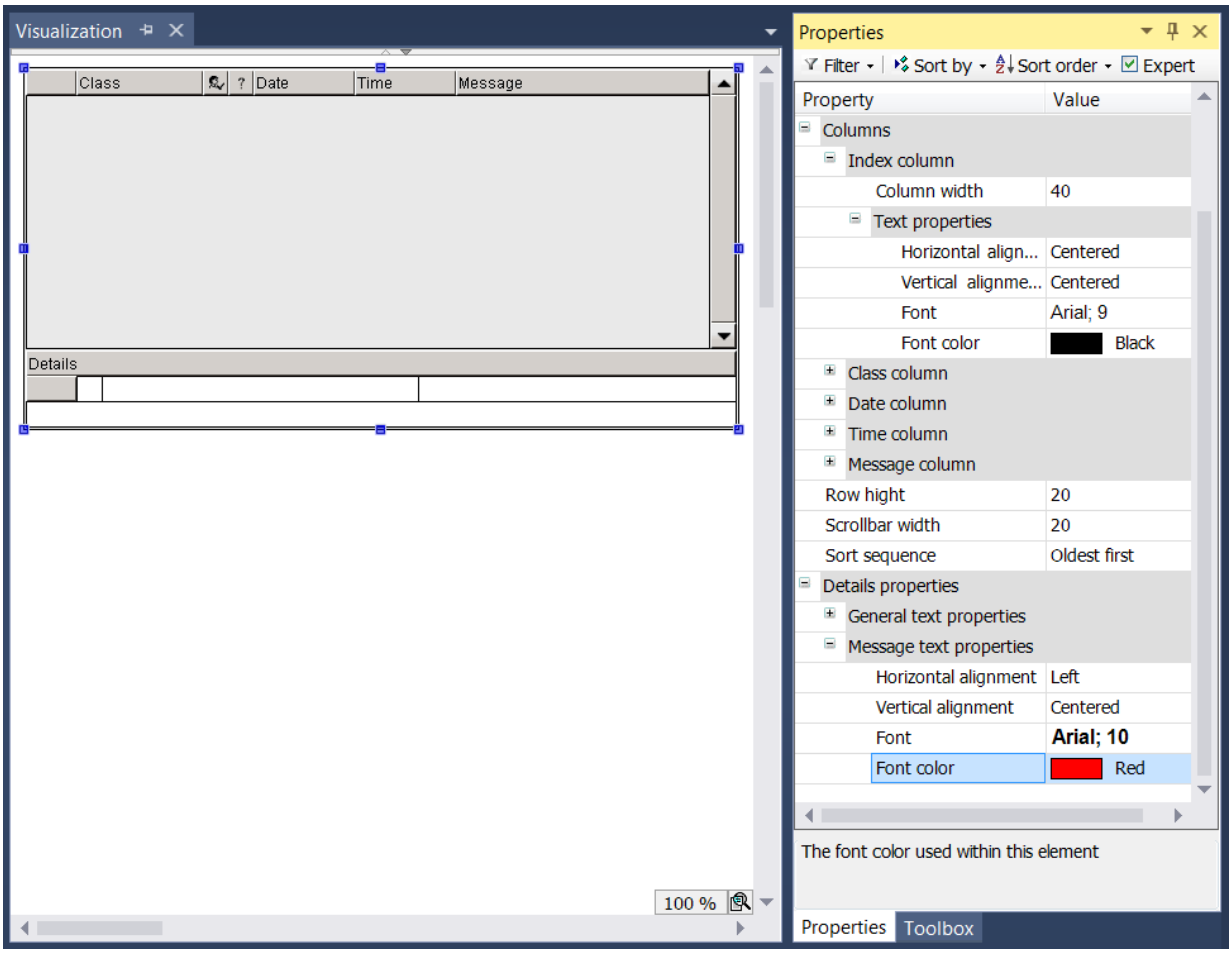
下一步：

1. 将 EventTable 元素添加到可视化页面后，在“Properties”（属性）中输入保存事件的“fbAdsReadEvents”或“fbReadTc3Events”实例的数组“aEvents”。
2. 将大小设置为 565x285 像素。

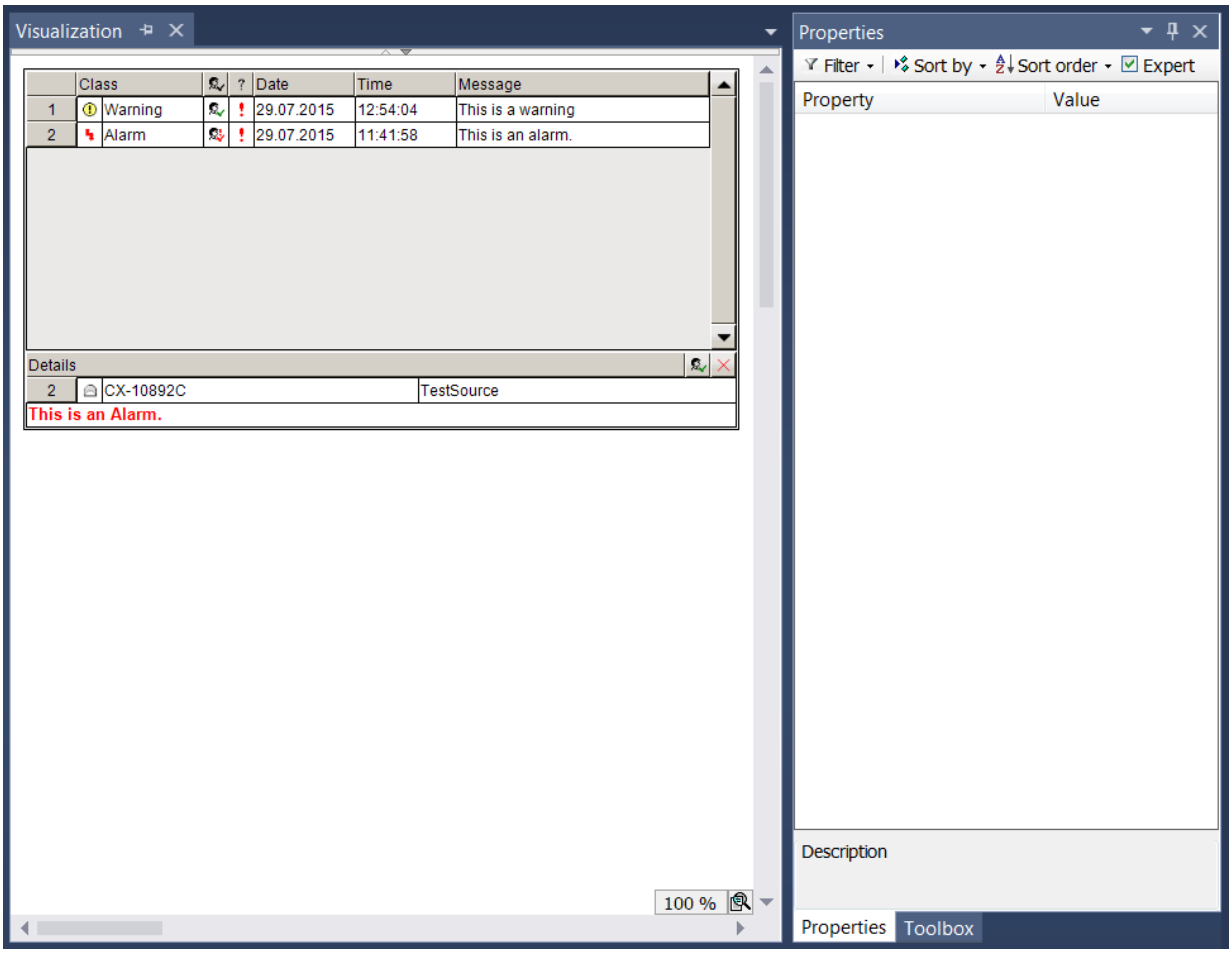


3. 若要使索引列的文本居中显示，请在“Columns”（列） → “Index column”（索引列）类别中将文本属性“Horizontal alignment”（水平对齐方式）更改为“Central”（居中）。

4. 在“Detail properties”（详细信息属性）中调整消息文本的字体大小和颜色，以便与其他信息区分开来。



⇨ 在运行时，如果触发了 2 个示例消息（来自 TcEventLogger），可视化元素将如下图所示：



15.9 可视化变体

在 PLC 项目中，3 种不同的可视化变体是有区别的：

- 集成的可视化 [▶ 550] - 可视化在编程系统上的 TwinCAT 开发环境中运行
- PLC HMI [▶ 551] - 可视化在控制计算机或第 3 台计算机上运行，无需开发环境
- PLC HMI Web [▶ 555] - 可视化在浏览器中运行

尽管有 3 种不同的变体，但只需要进行 1 次工程设计，因此所有可视化的外观都是相同的。只需添加 TargetVisualization [▶ 554] 和 WebVisualization [▶ 556] 对象，即可启用 PLC HMI 和 PLC HMI Web。在“可用性 [▶ 558]”部分可以查看各个 可视化元素 [▶ 369] 的可用性以及不同变体的功能

15.9.1 集成的可视化

出于诊断目的，最好仅在编程系统内运行可视化，而无需在控制器上加载可视化代码。如果在可视化管理器 [▶ 354]下没有添加“TargetVisualization [▶ 554]”或“WebVisualization [▶ 556]”客户端对象，则会自动使用此集成的可视化。在这种情况下，则不会生成可视化代码并将其加载到控制器上。不过，这也涉及到一些限制，具体如下。

对表达式、监控的限制

诊断可视化仅支持表达式，这可由编程系统的监控机制处理。它们包括：

- 正常变量访问，例如，MAIN.fbTest.nCounter
- 复杂访问，如下所列：
  - 访问标量数据类型的数组，其中 1 个变量用作索引（a[i]）
  - 访问复杂数据类型（结构、功能块、数组）的数组，其中 1 个变量用作索引（a[i].x）

- 访问所有数据类型的多维数组，带有 1 个或多个变量索引（a[i, 1, j].x）
- 访问带有常量索引的数组（a[3]）
- 如上所述的访问，其中简单运算符用于索引括号内的计算（a[i + 3]）
- 上述复合表达式的嵌套组合（a[i + 4 * j].aInner[j * 3].x）

- 索引计算中支持的运算符：+、-、*、/、MOD
- 指针监控，例如，p^.x
- 除以下情况外，不支持方法或函数调用：
  - 所有默认字符串函数
  - 所有类型转换函数，例如，INT_TO_DWORD
  - 所有运算符，例如，SEL、MIN……

对输入的限制

在输入动作“Run ST code”（运行 ST 代码）中，仅支持 1 个赋值列表。

示例：

```
PLC_PRG.n := 20 * PLC_PRG.m;
// nicht erlaubt
IF PLC_PRG.n < MAX_COUNT THEN
PLC_PRG.n := PLC_PRG.n + 1;
END_IF
//statt dessen folgendes verwenden:
PLC_PRG.n := MIN(MAX_COUNT, PLC_PRG.n + 1);
```

如果使用赋值列表，则在下一个周期之前不会分配左侧的值。无法在下一行立即处理。

可视化接口

在可视化的接口定义中不可以使用“Interface”（接口）类型。

15.9.2 PLC HMI

PLC HMI 是运行时系统的扩展，无需开发环境，即可在控制计算机或第 3 台计算机上执行可视化。根据现有的可视化对象可以创建可视化代码，并将其下载到控制计算机上。避免使用开发环境，这样可以显著节省内存。这对小型计算机十分有用。

下文将介绍以下主题：

- 调试 PLC HMI [▶ 551]
- PLC HMI 客户端的远程操作 [▶ 554]
- TargetVisualization 对象的编辑器 [▶ 554]

调试 PLC HMI

步骤 1：启用 PLC HMI

对象“TargetVisualization”（ ）启用 PLC HMI。通过上下文菜单命令 **Add > TargetVisualization**（添加 > TargetVisualization）可以将其添加到 PLC 项目树中的“Visualization Manager”（可视化管理器）对象中（另请参见 PLC 文档：创建可视化 > 可视化对象）。

通过 TargetVisualization 对象，在解决方案中创建可视化任务“VISU_TASK”，并在 PLC 项目中引用该任务。引用可用于调用可视化代码。因此，在添加对象之后，您必须重新激活配置。

删除 TargetVisualization 对象

如果您删除 TargetVisualization 对象，并且没有添加其他 WebVisualization 对象，则您必须删除 TwinCAT 项目树中 **System > Tasks**（系统 > 任务）下的任务“VISU_TASK”。在集成的可视化中不需要该任务。（另请参见对象 WebVisualization 的编辑器和集成的可视化）。

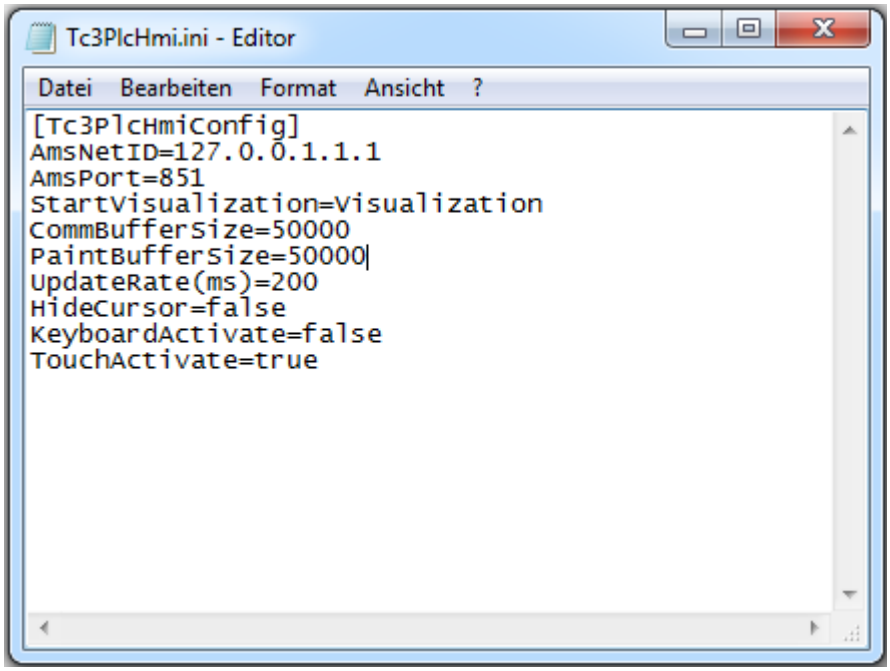
步骤 2：配置 PLC HMI 客户端

只有在您使用 Build 4022.0 以下版本或想要启动与运行时设备远程连接的 PLC HMI 客户端时，才需要执行步骤 2。自 Build 4022.0 或更高版本开始，系统会自动生成 .ini 文件并在文件夹 *C:\TwinCAT\3.1\Boot\Plc* 中进行更新。自 Build 4026.0 或更高版本开始，系统会自动生成 .ini 文件并在文件夹 *C:\ProgramData\Beckhoff\TwinCAT\3.1\Boot\Plc* 中进行更新。

为了在客户端和执行相应的可视化代码的设备之间建立连接，您必须调整 Tc3PlcHmi.ini 文件。

对于 Build 4022.0 以下版本，.ini 文件在文件夹 *C:\TwinCAT\3.1\Components\Plc\Tc3PlcHmi* 中可用；对于 Build 4022.0 及以上版本，该文件在文件夹 *C:\TwinCAT\3.1\Boot\Plc* 中可用；对于 Build 4026.0 及以上版本，该文件在文件夹 *C:\Program Files (x86)\Beckhoff\TwinCAT\3.1\Components\Plc\Tc3PlcHmi* 中可用。

.ini 文件的示例：



AMSNETID	执行可视化代码的设备的 AmsNetID。 预设值：127.0.0.1.1.1
AmsPort	可视化所属的 PLC 项目的 AmsPort。 预设值：851
StartVisualization	作为起始页打开的可视化对象的名称。 预设值：可视化
CommBufferSize	可视化为该 PLC HMI 客户端分配且可用于通信的内存大小（以字节为单位）。 预设值：50000
PaintBufferSize	可视化为该 PLC HMI 客户端分配且可用于绘图动作的内存大小（以字节为单位）。 预设值：50000
UpdateRate(ms)	再次查询客户端数据的更新速率（以毫秒为单位）。 预设值：200
HideCursor	可以隐藏光标的设置。 预设值：FALSE
KeyboardActivate	启用硬件键盘输入的设置。如果禁用此设置，则会自动使用软件键盘。 预设值：FALSE
TouchActivate	启用触控输入的设置。 预设值：TRUE

步骤 3：将 PLC HMI 设置为启动应用程序

只有在您使用 Build 4024.0 以下版本或想要启动与运行时设备远程连接的 PLC HMI 客户端时，才需要执行步骤 3。自 Build 4024 或更高版本开始，PLC HMI 客户端会在运行时设备上自动本地启动。

如果 PLC HMI 要在使用启动项目启动计算机时自动启动，则在*启动*文件夹中必须有 1 个指向 Tc3PlcHmi.exe 应用程序的链接。

为此，可执行以下步骤：

1. 打开目录 *C:\TwinCAT\3.1\Target\StartUp*。
2. 通过上下文菜单命令 **New**（新建），添加新链接。
3. 输入 *C:\TwinCAT\3.1\Components\Plc\Tc3PlcHmi\Tc3PlcHmi.exe* 作为存储位置。
4. 确认此对话框和后续对话框。

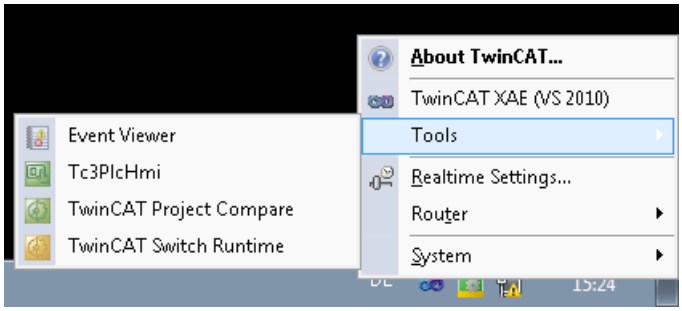
针对倍福 CE 设备，执行以下步骤：

1. 在 **Start** > **StartMan**（开始 > StartMan）下，启动倍福启动管理器。
2. 通过 **New**（新建）按钮，添加新项目。
3. 将项目命名为“Tc3PlcHmi”，并选择类型“ShellCommand”。
4. 确认对话框。
5. 在 **Startup Options**（启动选项）下选择“Autostart”（自动启动），并在 **Delay**（延迟时间）下输入 1 个时间，以便仅在 PLC 项目已启动时打开客户端。
6. 切换到 **Shell Command**（Shell 命令）选项卡。
7. 在 **Enter Shell command**（输入 Shell 命令）字段中，输入“\Hard Disk\TwinCAT\3.1\Components\Plc\Tc3PlcHmi\X.exe”。使用在指定路径下存储的客户端 Exe 的名称替换“X”。例如，这可能在 ARM 和 ATOM 设备之间有所不同。
8. 确认对话框。

步骤 4：启动 PLC HMI 客户端

只有在您使用 Build 4024.0 以下版本或想要启动与运行时设备远程连接的 PLC HMI 客户端时，才需要执行步骤 4。自 Build 4024 或更高版本开始，PLC HMI 客户端会在运行时设备上自动本地启动。

PLC HMI 客户端借助 Tc3PlcHmi.exe 应用程序启动。它位于目录 *C:\TwinCAT\3.1\Components\Plc\Tc3PlcHmi* 下，但也可以链接到任何所需的位置。如果您在 *C:\TwinCAT\3.1\Target\StartMenuAdmin\Tools* 目录下创建 1 个链接，则您可以通过 **Tools**（工具）下的上下文菜单中的 TwinCAT 图标启动应用程序。



如果已连接开发 PC，还可以在开发环境中显示可视化。不过，它并不等同于集成的可视化，而是同样基于 PLC HMI 客户端。

对于倍福 CE 设备，您必须在启动客户端之前在可视化管理器中激活 1 个设置：该设置可将所有 svg 格式的图像文件自动转换为 bmp 格式。此步骤是必需的，因为在 CE 下，PLC HMI 客户端仅支持 bmp 格式的图像文件。由于 PLC HMI Web 客户端会继续使用 svg 格式，因此，这 2 种图像文件格式都会加载到目标系统上。在目录 *\Hard Disk\TwinCAT\3.1\Components\Plc\Tc3PlcHmi* 下可以找到 CE 的 PLC HMI 客户端。

另请参见：

- PLC 文档：创建可视化 > 可视化管理器 > 设置
- PLC 文档：创建可视化 > 可视化变体 > 集成的可视化

- 文档 TC3 PLC HMI Web

PLC HMI 客户端的远程操作

在第 3 台计算机（既不是开发计算机，也不是控制计算机）上也可以远程操作 PLC HMI 客户端。为此，必须满足以下要求：

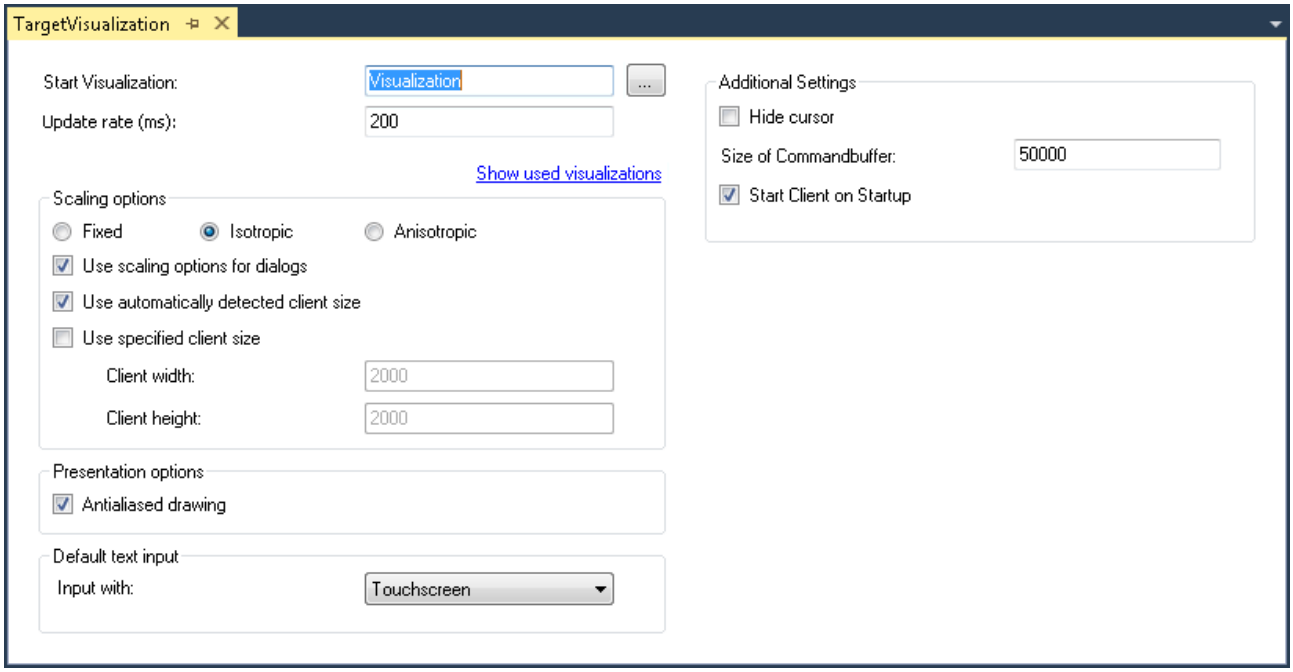
- 系统已安装 TwinCAT 3 Build 4018.0 ADS 或更高版本。
- 与执行可视化代码的控制计算机建立 ADS 通信（**TwinCAT Icon > Router > Edit Routes > Add…**）（TwinCAT 图标 > 路由器 > 编辑路由 > 添加……）。
- 从开发计算机或控制计算机已将 Tc3PlcHmi 文件夹复制到第 3 个系统。务必手动添加文件夹的路径。
- 在运行客户端的系统上已调整 Tc3PlcHmi.ini 文件。

TargetVisualization 对象的编辑器

您可以在 PLC 项目树中的“Visualization Manager”（可视化管理器）对象下方添加

“TargetVisualization”对象（ ），该对象可以启用 PLC HMI 并包含其设置。双击该对象，以便在编辑器窗口中编辑设置。

自 Build 4022.0 开始，“TargetVisualization”对象中的设置将自动应用到 .ini 文件中。如果您希望使用较旧的版本或启动与运行时设备远程连接的 PLC HMI 客户端，则您必须手动更改 .ini 文件中的设置。



启动可视化	在启动 PLC HMI 时作为第 1 页打开的可视化对象的名称。默认情况下，在这里已经输入了 1 个可视化对象。输入助手可用于选择不同的可视化对象。如果 PLC 项目只包含 1 个可视化对象，则会自动将其用作启动可视化。
更新速率（ms）	PLC HMI 中数据的更新速率（以毫秒为单位）。
显示已使用的可视化	用于打开可视化管理器的标准对话框的按钮：您可以在此处选择要用于 PLC HMI 的可视化。（另请参见 PLC 文档：创建可视化 > 可视化管理器 > 可视化 [▶ 359]）

缩放选项

固定	无论屏幕大小如何，都会保持可视化的大小。
各向同性	可视化的大小取决于屏幕的大小。可视化保持其比例。
各向异性	可视化的大小取决于屏幕的大小。可视化无法保持其比例。
使用对话框的缩放选项	对话框、小键盘和数字键盘均使用与可视化相同的缩放因子进行缩放。如果已创建与可视化相匹配的对话框，这将非常有利。
使用自动确定的客户端大小	PLC HMI 填充客户端屏幕。
使用指定的客户端大小	PLC HMI 填充的屏幕区域由以下尺寸决定。 • 客户端高度：以像素为单位的高度 • 客户端宽度：以像素为单位的宽度

显示选项

具有抗锯齿特性的字符	如果要在编程系统的可视化编辑器窗口中绘制可视化时使用抗锯齿功能，可激活此选项。（离线或在线）

标准文本输入

只有在可视化元素的输入配置中选择“Standard”（标准）输入类型时，才会激活该设置。在这种情况下，将使用在可视化管理器中定义的默认文本条目。

触摸屏	如果默认使用触摸屏操作目标设备，可选择此选项。
键盘	如果默认使用键盘操作目标设备，可选择此选项。

高级设置

隐藏鼠标指针	可以隐藏光标的设置。
命令缓冲区的大小	可视化为该 PLC HMI 客户端分配且可用于通信的内存大小（以字节为单位）。
在启动时启动客户端	PLC HMI 客户端会在运行时系统上自动本地启动。

15.9.3 PLC HMI Web

PLC HMI Web 可在任何网络浏览器中显示可视化。它以 Java 脚本的形式实现，可从网络服务器查询显示信息。只有显示中的变化才会循环传输。在下载可视化项目时，对于 PLC HMI Web 所需的所有文件，TC3.1.4026.0 版本以下的文件会传输到目录 *C:\TwinCAT\3.1\Boot\Plc\Port_851\Visu* 下，TC3.1.4026.0 及以上版本的文件会传输到目录 *C:\ProgramData\Beckhoff\TwinCAT\3.1\Boot\Plc\Port_851\Visu* 下。这包括 Java 脚本、可视化的基本 HTML 页面（HTM 文件）以及可视化中所需的所有图像。

目前，只能为可通过端口 851 访问的 PLC 项目配置 PLC HMI Web。

下文将介绍以下主题：

- 要求 [▶ 555]
- PLC HMI Web 的调试 [▶ 556]
- WebVisualization 对象的编辑器 [▶ 556]

要求

- 在服务器端，必须对网络服务器进行相应配置。
- 在客户端，必须至少使用 Microsoft Internet Explorer 10 或最新版本的 Mozilla Firefox、Google Chrome 或 Safari。

违反数据安全规定

为了最大限度地降低违反数据安全规定的风险，建议对运行应用程序的系统采取以下组织和技术措施：

- 尽可能避免将 PLC 和控制网络暴露在开放网络和互联网中。
- 使用 VPN 等附加数据链路层进行远程访问，并安装防火墙机制进行保护。
- 限制授权人员访问，在首次调试期间更改任何默认密码，然后定期更改密码。

调试 PLC HMI Web

步骤 1：配置互联网信息服务（IIS）

PLC HMI Web 使用 Microsoft IIS 作为网络服务器。IIS 必须进行相应配置。配置工作由可从倍福网站下载的 TF1810 | TC3 PLC HMI Web 安装程序完成。

步骤 2：启用 PLC HMI Web

“WebVisualization”对象（ ）可启用 PLC HMI Web。通过上下文菜单命令 **Add > WebVisualization**（添加 > WebVisualization），您可以将其添加到 PLC 项目树中的“Visualization Manager”（可视化管理器）对象中（另请参见 PLC 文档：创建可视化 > 可视化对象）。

通过 WebVisualization 对象，在解决方案中创建可视化任务“VISU_TASK”，并在项目中引用该任务。引用可用于调用可视化代码。因此，在添加对象之后，您必须重新激活配置。

删除 WebVisualization 对象

如果您删除 WebVisualization 对象，并且没有添加其他 TargetVisualization 对象，则您必须删除 TwinCAT 项目树中 **System > Tasks**（系统 > 任务）下的任务“VISU_TASK”。在集成的可视化中不需要该任务。（另请参见 TF1800：TargetVisualization 对象的编辑器和 PLC：集成的可视化）

步骤 3：调用 PLC HMI Web

为了调用可视化的开始页面，可在网页浏览器中输入以下地址：https://device_name/Tc3PlcHmiWeb/Port_851/Visu/webvisu.htm

示例：https://localhost/Tc3PlcHmiWeb/Port_851/Visu/webvisu.htm

“webvisu”是在 PLC HMI Web 设置中定义的可视化的 HTML 开始页面。在调用之后，它可用于在浏览器中显示在管理器中定义的启动可视化。然后，您可以在浏览器中进行可视化操作。

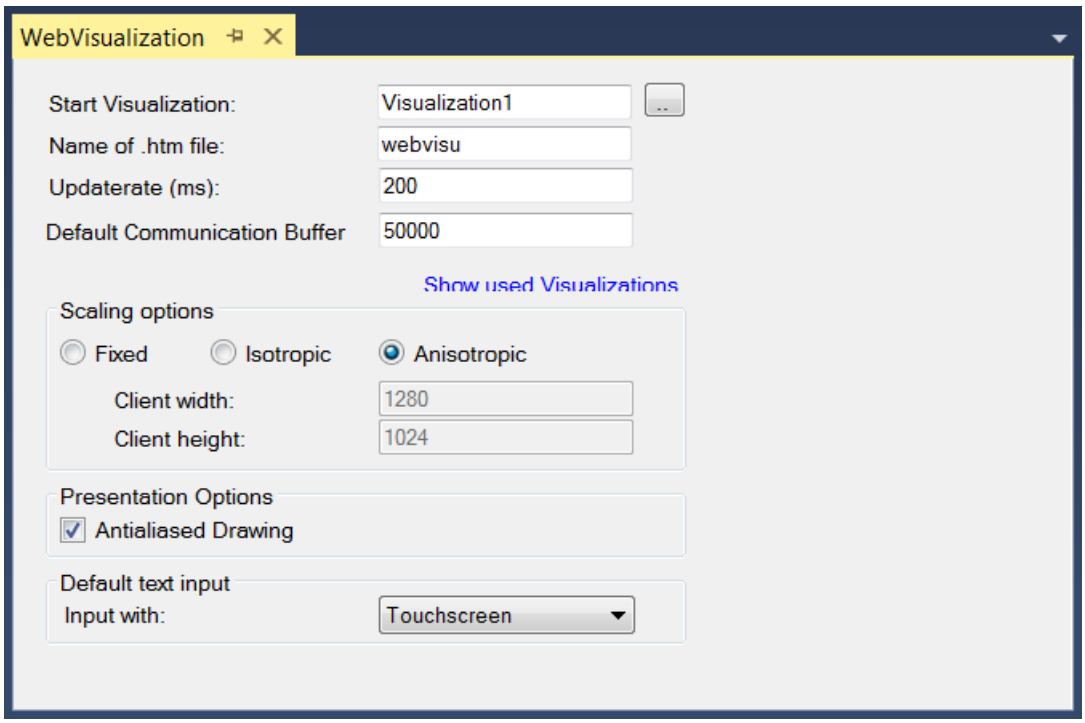
您还可以在调用 PLC HMI Web 时为其命名，以便以后在应用程序中专门对其进行处理。为此，可在 URL 后输入参数 ClientName=<Name>。

示例：https://localhost/Tc3PlcHmiWeb/Port_851/Visu/webvisu.htm?Clientname=V_ClientXY

WebVisualization 对象的编辑器

您可以在 PLC 项目树中的“Visualization Manager”（可视化管理器）对象下方添加

“WebVisualization”对象（ ），该对象可以启用 PLC HMI Web 并包含网络可视化的设置。双击该对象，以便在编辑器窗口中编辑设置。



启动可视化	在启动 PLC HMI Web 时自动显示的可视化的名称。在这里，“可视化”应作为标准输入。输入助手可用于选择不同的可视化。
.htm 文件的名称	可视化的基本 HMTL 页面的名称，也必须作为网络浏览器中的地址输入。 示例： https://localhost/Tc3PlcHmiWeb/Port_851/Visu/webvisu.htm
Updaterate（ms）	网络浏览器中数据的更新速率（以毫秒为单位）。
默认通信缓冲区	通信缓冲区的大小（以字节为单位）。指定网络客户端和网络浏览器之间数据传输的最大可用内存。
显示已使用的可视化	用于打开可视化管理器的标准对话框的按钮：您可以在此处选择要用于 PLC HMI Web 的可视化。（另请参见 PLC 文档：创建可视化 > 可视化管理器 > 可视化 [▶ 359]）

缩放选项

固定	无论浏览器窗口的大小如何，都会保持可视化的大小。
各向同性	可视化的大小取决于浏览器窗口的大小。不过，可视化会保持其比例。
各向异性	可视化的大小取决于浏览器窗口的大小。可视化无法保持其比例。
客户端大小	通过以下设置可以定义 PLC HMI Web 的显示大小： • 客户端高度：以像素为单位的高度 • 客户端宽度：以像素为单位的宽度

显示选项

抗锯齿图	如果要在编程系统的可视化编辑器窗口中绘制可视化时使用抗锯齿功能，可激活此选项。（离线或在线）

默认文本输入

只有在可视化元素的输入配置中选择“Standard”（标准）输入类型时，才会激活该设置。在这种情况下，将使用在可视化管理器中定义的默认文本条目。

触摸屏	如果默认使用触摸屏操作网络客户端，可选择此选项。
键盘	如果默认使用键盘操作网络客户端，可选择此选项。

15.9.4 可用性

下文将介绍不同可视化类型的单个可视化元素和功能的可用性。

可视化元素

	集成的可视化	PLC HMI	PLC HMI CE	PLC HMI Web
通用控制 [▶ 385]	✓	✓	✓	✓
基础 [▶ 433]	✓	✓	✓	✓
灯/开关/位图 [▶ 479]	✓	✓	✓	✓
测量设备 [▶ 488]	✓	✓	✓	✓
特殊控制 [▶ 528]				
• 文本编辑器 [▶ 530]	✗	✓	✓	✓
• ActiveX 元素 [▶ 535]	✓	✗	✗	✗
• Webbrowser [▶ 537]	✓	✗	✗	✗
• TcEventTable [▶ 542]	✓	✓	✓	✓

功能

	集成的可视化	PLC HMI	PLC HMI CE	PLC HMI Web
用户管理 [▶ 360]	✗	✓	✓	✓
CurrentVisu [▶ 355], CurrentLanguage [▶ 563]	✗	✓	✓	✓
更改语言 [▶ 563]	✓	✓	✓	✓
内部旋转 [▶ 396]	✓	✓	✗	✓
渐变类型 [▶ 434]	✓	✓	✗	✓
透明度 [▶ 434]	✓	✓	✗	✓
文本格式 [▶ 386]	✓	✓	✗	✓

支持的图像格式

	集成的可视化	PLC HMI	PLC HMI CE	PLC HMI Web
SVG	✓	✓	✗	✓
BMP	✓	✓	✓	✓
JPG	✓	✓	✓	✓
PNG	✓	✓	✓	✓

版本高于 3.0.0.0 的可视化样式 [▶ 355]在内部使用 SVG 格式的图像文件。在可视化管理器中可以启用将图像转换为 [▶ 356]功能，以便在 PLC HMI CE 中使用这些样式。该功能可将图像文件转换为 BMP 或 PNG。借助设置传输 svg 图像和转换图像 [▶ 356]，您可以将转换图像文件和原始图像文件传输到目标系统。然后，PLC HMI Web 客户端会自动使用 SVG 变体，PLC HMI CE 客户端会自动使用 BMP 或 PNG 变体。

15.10 应用程序提示

15.10.1 处理可视化页面

TwinCAT PLC Control 中可视化引用和占位符的概念可以被 TwinCAT 3 中的类似概念所取代。

另一种可视化的引用

您可以在另一个页面中添加可视化页面 [▶ 367]，并以这种方式引用它。为此，您应使用边框 [▶ 470]元素。这样，由其他各种页面可以组成 1 个可视化页面。1 个边框元素可以包含 1 个或多个可视化页面的引用。在边框选择 [▶ 478]对话框中可以定义这些可视化页面。

占位符的接口

每个可视化页面 [▶ 367]都可以通过接口编辑器 [▶ 345]提供 1 个接口，在该接口中，可以采用与功能块类似的方式定义输入变量。这些输入参数可用作占位符。在可视化页面的实例（引用）中，必须将它们替换为本地对象中的特殊应用程序的值或表达式。

替换必须在集成可视化实例的边框元素的属性 [▶ 393]中进行。请注意，必须为可视化实例的输入变量分配有效变量。如果在接口编辑器中更改变量，则会为每个实例的占位符打开对话框“更新边框参数 [▶ 478]”。您可以在此处添加或编辑占位符。

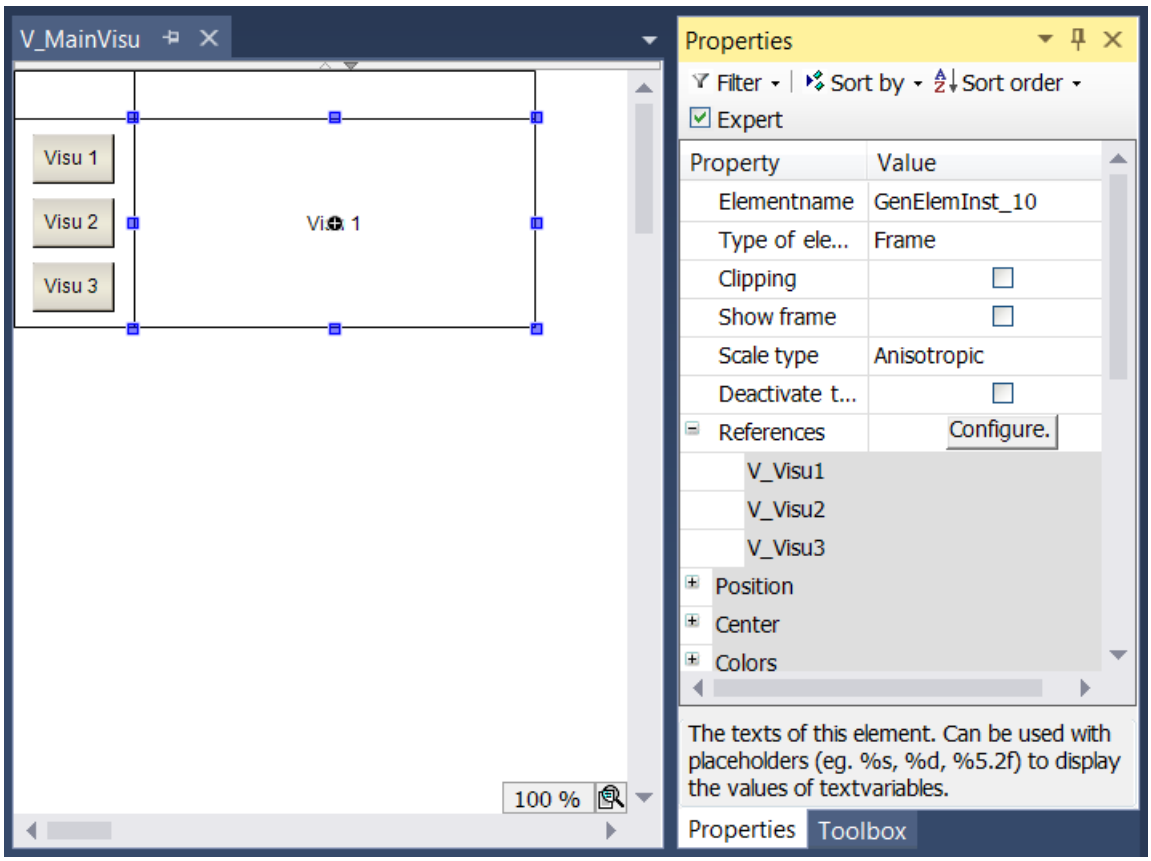
在边框内的可视化页面之间切换

如果 1 个边框元素包含多个可视化参照，则可对另一个可视化元素的用户输入进行配置，使其在边框中切换显示这些参照。为此，输入配置 [▶ 399]可提供动作“切换边框可视化 [▶ 379]”。这样可以在 1 个基本的可视化页面上切换多个其他可视化页面。

示例

可视化页面“V_MainView”有 1 个选择菜单，由 3 个按钮和 1 个边框元素组成。为每个按钮分配 1 个可视化页面。在在线模式下按下按钮时，页面将在边框元素中显示。

1. 创建带有 3 个按钮的选择菜单
2. 插入边框元素
3. 通过边框选择对话框，可为该边框分配 3 种可视化，并在这 3 种可视化之间切换显示。
4. 对于每个按钮，通过输入配置 [▶ 399]可以为相应的可视化页面添加 1 个“切换边框可视化 [▶ 379]”类型的“OnMouseClick”动作。



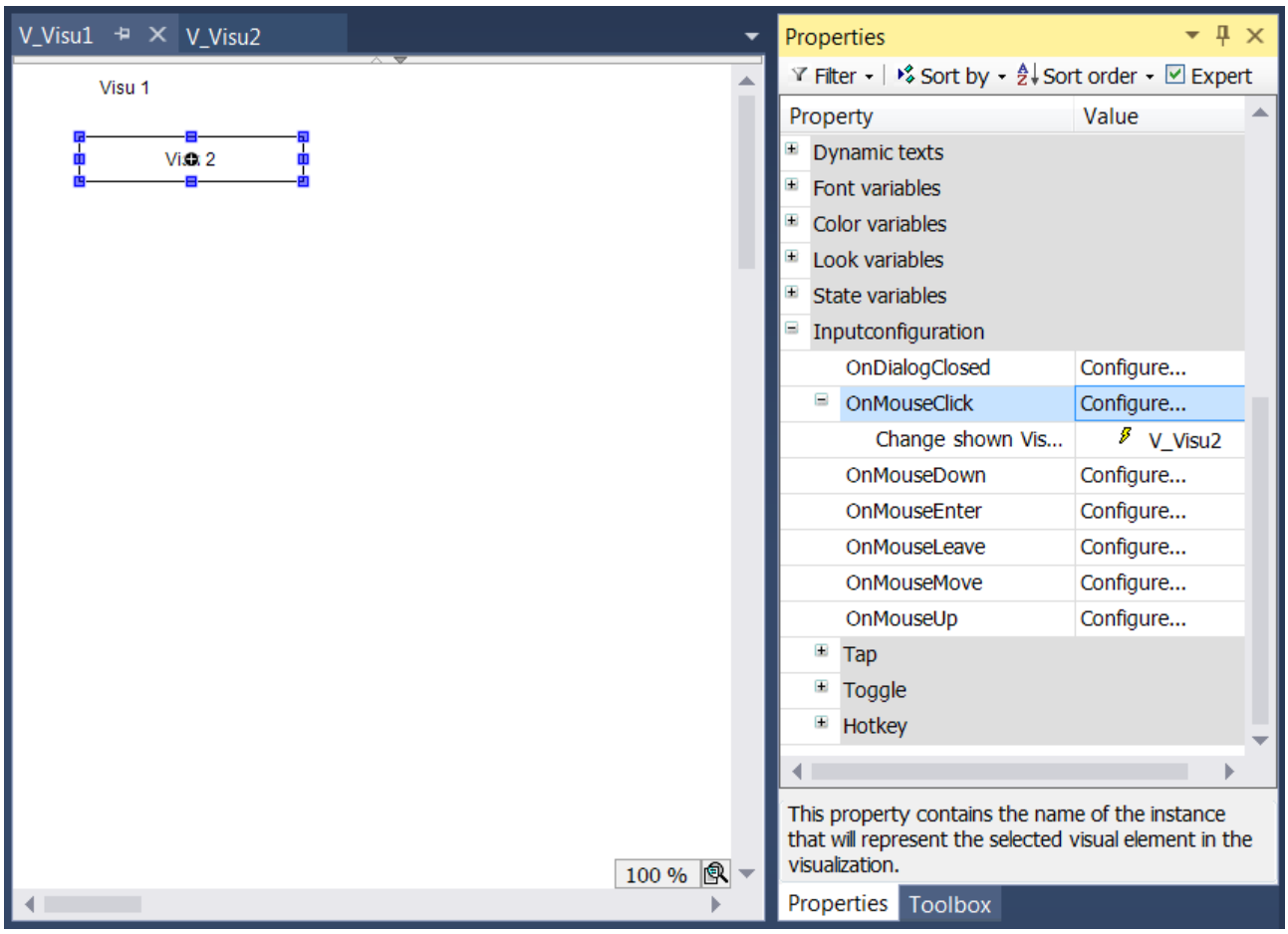
在可视化页面之间切换

除了边框元素内的切换选项之外，您还可以更改当前可见的整个可视化页面。为此，输入配置 [▶ 399]可提供另一个名称为“更改显示可视化 [▶ 376]“的动作。

示例

创建 2 个可视化页面，它们的名称分别为”V_Visu1“和”V_Visu2“。每个页面都包含 1 个按钮，用于切换到相应的其他可视化页面。

1. 在 PLC 项目中创建 2 个可视化对象，它们的名称分别为”V_Visu1“和”V_Visu2“。
2. 在这 2 个可视化页面上添加标签，以便它们可以相互区分。
   - ”V_Visu1“：在标签属性中输入文本”Visu 1“。
   - ”V_Visu2“：在标签属性中输入文本”Visu 2“。
3. 在两侧添加 1 个矩形元素。
   - ”V_Visu1“：在矩形属性中输入文本”Visu 2“。
   - ”V_Visu2“：在矩形属性中输入文本”Visu 1“。
4. 为这 2 个矩形配置 1 个”OnMouseClick [▶ 399]“事件，动作为”更改显示可视化 [▶ 376]“。
   - ”V_Visu1“：将可视化”V_Visu2“分配给动作。
   - ”V_Visu2“：将可视化”V_Visu1“分配给动作。



CurrentVisu 变量

通过变量 CurrentVisu [▶ 355] 还可以进行页面切换。在分配可视化页面之后，该页面上的所有活动客户端都会自动刷新。

分配变量：

```
VisuElems.CurrentVisu := sVisuName;
```

分配文本：

```
VisuElems.CurrentVisu := ‚Visualization‘;
```

15.10.2 文本和语言

作为 1 项基本原则，在 PLC HMI 中会借助文本列表 [▶ 127]来管理文本和语言。ANSI 和 Unicode [▶ 355] 可以作为字符编码变体。在可视化中，通过 2 种不同的方式可以对元素 [▶ 369]进行标注：

- 静态文本
- 动态文本

静态文本

通过在元素属性中的“Texts”（文本）类别下输入文本，可为可视化元素分配静态文本。如果选中元素并按下空格键，也可以直接在编辑器窗口中输入它。在运行时不能更改该文本。仅可更改文本语言。

在文本列表“GlobalTextList [▶ 130]”中会自动保存每个静态文本。在该文本列表中，也可以管理其他语言的翻译。

GlobalTextList

ID	Default	de	en
0	%s		
1	Activate	Aktivieren	Activate
2	Deactivate	Deaktivieren	Deactivate

动态文本

在元素属性中的“Dynamic texts”（动态文本）类别下输入动态文本。为此，首先必须从 PLC 项目中的列表中选择文本列表的名称。在这里不能使用“GlobalTextList [▶ 130]”，只能使用用户创建的文本列表 [▶ 127]。此外，还必须为文本条目的 ID 指定 1 个变量，通过该变量可以在运行时更改文本。

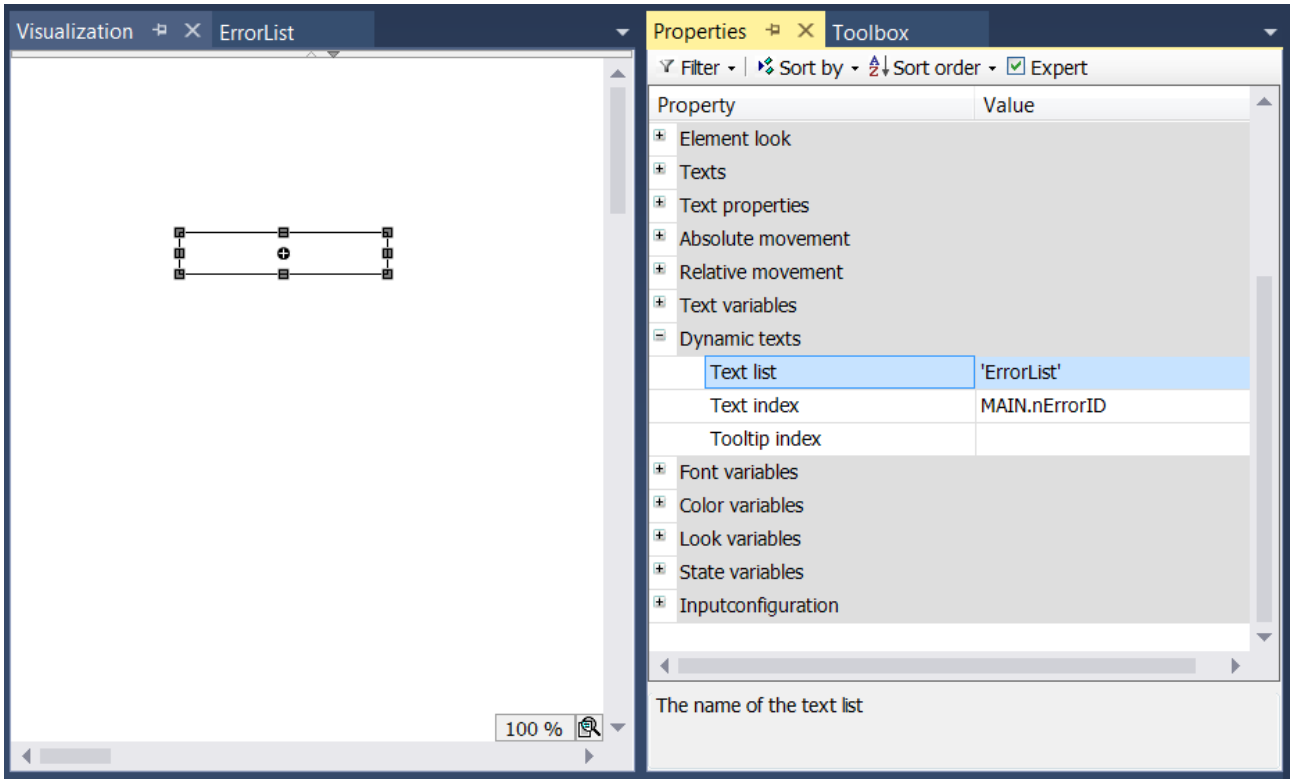
示例

例如，创建 1 个名称为“ErrorList”的文本列表。在此列表中输入一些错误消息及其翻译。

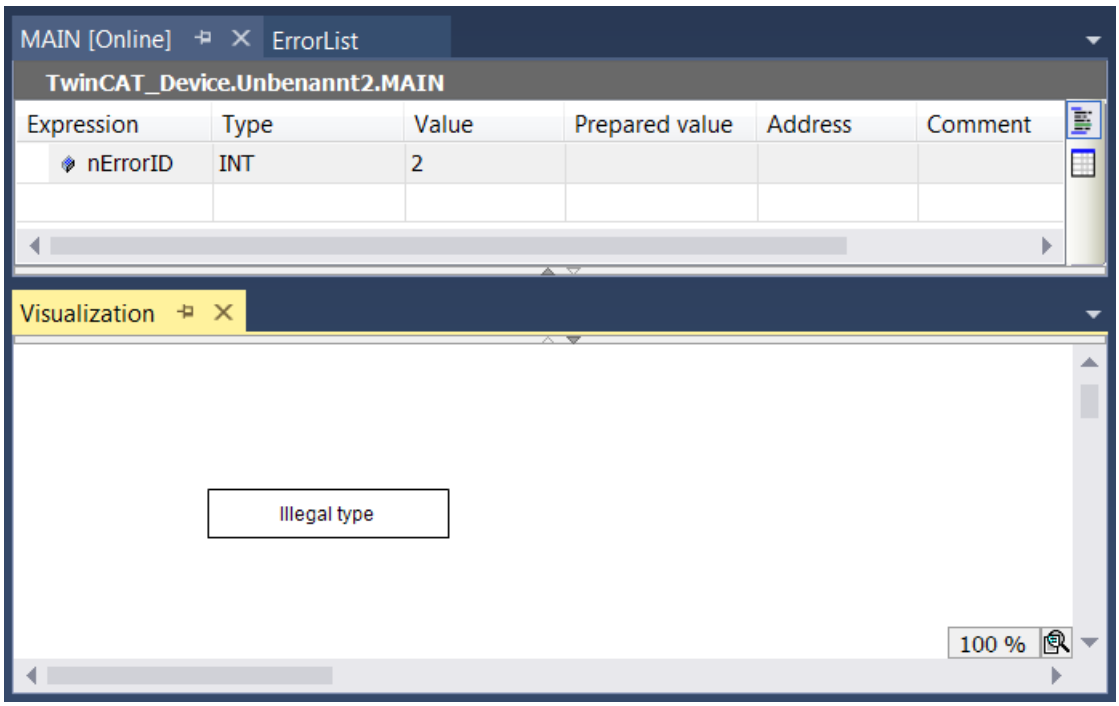
ErrorList

ID	Default	de	en
0	Wrong argument	Falsches Argument	Wrong argument
1	Bad format	Ungültiges Format	Bad format
2	Illegal type	Ungültiger Typ	Illegal type
3	Bad result	Ungültiges Ergebnis	Bad result
4	Wrong data type	Ungültiger Datentyp	Wrong data type

为可视化添加 1 个矩形元素 [▶ 433]，名称为“Visualization”（可视化）。在该元素的属性中，选择“动态文本 [▶ 397]”下的文本列表“ErrorList”，以及在“MAIN”中声明的变量“nErrorId”。



在运行时中，通过将变量“nErrorId”设置为 1 个介于大于或等于 0 至小于或等于 4 之间的值，可以在矩形中显示存储在文本列表“ErrorList”中的错误消息之一。在本示例中，“nErrorID”的值为 2。



语言切换

如果当前使用的文本列表 [▶ 127]定义了多种语言的文本翻译，则可以指定在可视化开始时使用的语言 [▶ 356]。此外，您还可以借助可视化按钮，在运行时更改语言。为此，可使用元素设置中的“Input configuration”（输入配置）类别下的“更改语言 [▶ 375]”动作。

“VisuElems [▶ 366]”库中的变量“CURRENTLANGUAGE”可用于查询当前使用的语言。该变量还可用于更改程序代码中的语言：

```
fbTrigger(CLK := bPressed);
IF fbTrigger.Q THEN
    IF VisuElems.CURRENTLANGUAGE = 'de' THEN
        VisuElems.CURRENTLANGUAGE := 'en';
    ELSE
        VisuElems.CURRENTLANGUAGE := 'de';
    END_IF
END_IF
```

如果文本列表中没有与当前设置语言匹配的条目，则使用“Default”（默认值）下的条目。

在可视化开始时的默认语言的定义以及变量“VisuElems.CURRENTLANGUAGE”的使用只能与 PLC HMI [▶ 551] 和/或 PLC HMI Web [▶ 555] 一起使用。通过集成的可视化 [▶ 550]，在启动时会自动使用文本版“标准”。在集成的可视化中，还可以通过可视化按钮在运行时进行语言更改。

格式化文本

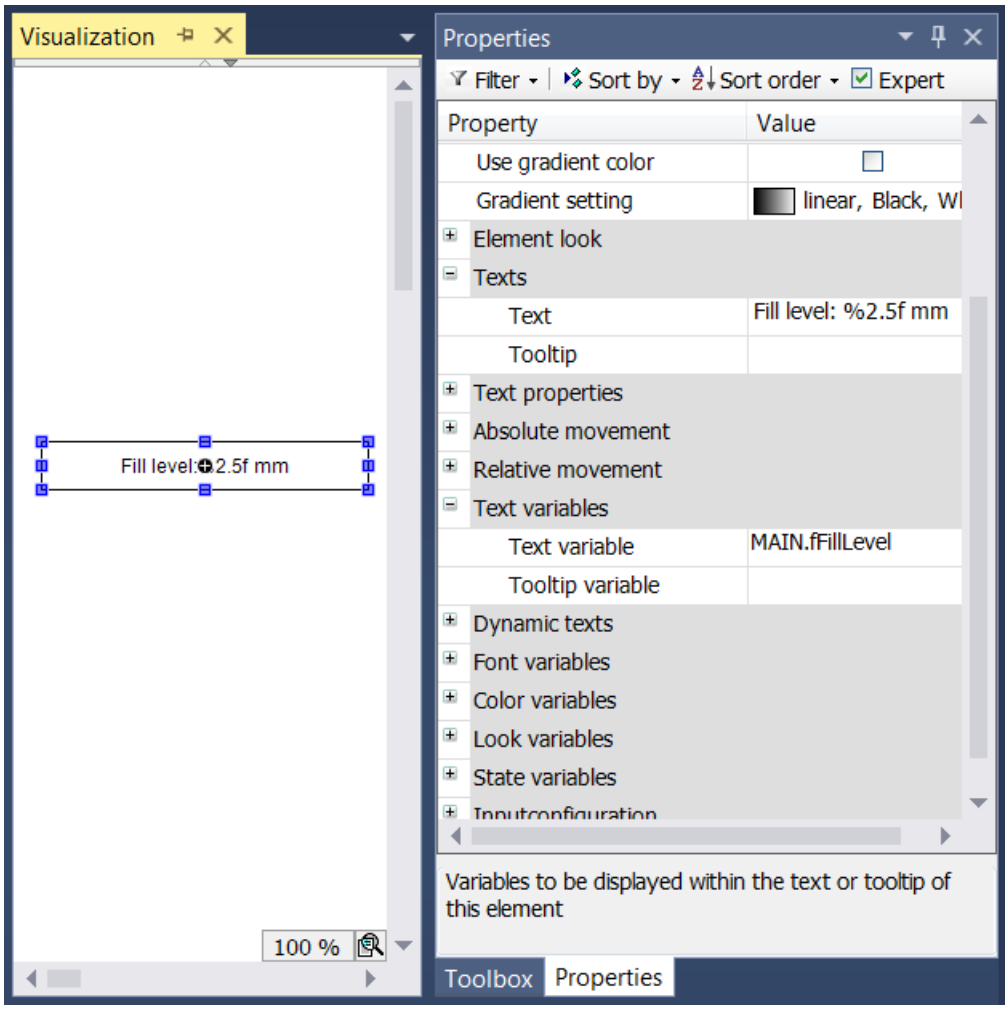
除了在元素属性 [▶ 353]中的“Texts”（文本）类别下输入纯文本之外，您还可以在在线模式下为文本显示添加格式化信息。格式化规范总是以“%”开头，后面跟 1 个决定格式化类型的字符。它可以单独使用，也可以与实际文本组合使用。

在元素属性中的“Text variables”（文本变量）类别下应该指定在元素中要作为文本输出的变量。如要显示作为输入参数传输至可视化功能块的变量的实例名称，请可使用编译指示属性“parameterstringof [▶ 758]”。

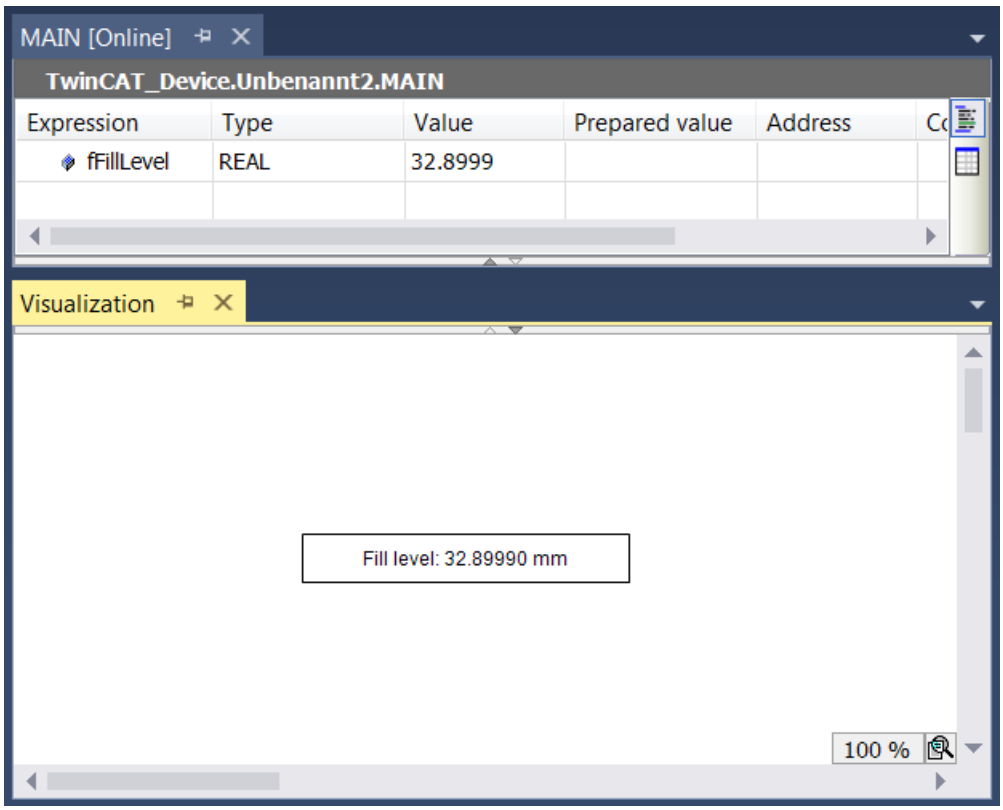
%b	二进制数
%c	单个字符
%d，%i	十进制数
%f	REAL 值
%o	无符号八进制数（无前缀零）
%s	字符串
%u	无符号十进制数
%x	无符号十六进制数（无前缀“0x”）。

示例

在矩形元素属性 [▶ 353]的“Texts”（文本）类别的“Text”（文本）属性字段中输入“Fill level: %2.5f mm”。在“Text variables”（文本变量）类别下的“Text variable”（文本变量）输入字段中可以指定要使用的变量。在本示例中，它是“fFillLevel”。在 MAIN 程序中将其声明为 REAL。



在运行时，矩形元素如下所示：



如要显示百分号 % 以及上述格式化规范之一，可输入“%%”。

示例

在在线模式下，输入“Rate in %%: %s”可获得以下信息：“Rate in %: 12”（如果文本变量当前返回“12”）。

系统时间的输出

在在线模式下，“%t”和方括号内的以下特殊占位符的组合会被当前系统时间取代。占位符可定义显示格式。

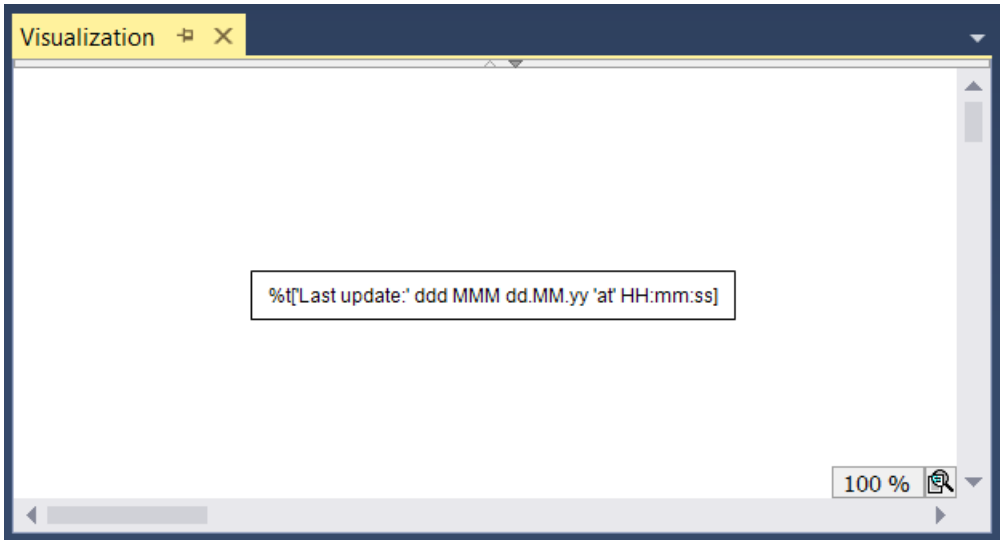
导入 TwinCAT PLC 2 Control 项目：如果导入的是旧项目，则先前使用的时间格式 %t 会被自动转换为新的 %t[] 格式，但不再支持以下占位符：%U、%W、%z、%Z。

有效占位符：

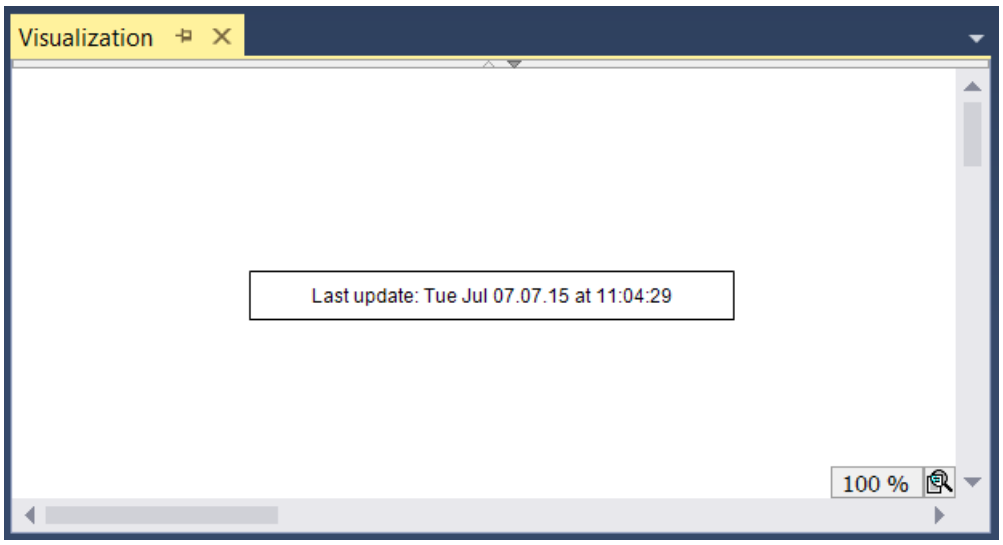
ddd	星期名称的缩写，例如，“Wed”
dddd	星期名称，例如，“Wednesday”
ddddd	以数字表示的 1 个星期中的某一天（0 - 6；星期日为 0）
MMM	月份名称的缩写，例如，“Feb”
MMMM	月份名称，例如，“February”
d	以数字表示的 1 个月中的某一天（1 - 31），例如，“8”。
dd	以数字表示的 1 个月中的某一天（01 - 31），例如，“08”。
M	以数字表示的月份（1 - 12），例如，“4”。
MM	以数字表示的月份（01 - 12），例如，“04”。
jjj	以数字表示的 1 年中的某一天（001-366），例如，“067”
y	未指明世纪的年份（0-99），例如，“9”
yy	未指明世纪的年份（00-99），例如，“09”
yyyy	指明世纪的年份，例如，“2009”
HH	小时，24 小时制（01-24），例如，“16”。
hh	小时，12 小时制（01-12），例如，“4”表示下午 4 点
m	分钟（0-59），无前缀零，例如，“6”。
mm	分钟（00-59），带前缀零，例如，“06”。
s	秒（0-59），无前缀零，例如，“6”。
ss	秒（00-59），带前缀零，例如，“06”
ms	毫秒（0-999），无前缀零，例如，“322”
t	以 12 小时制显示的 ID：A（小时数 <12）或 P（小时数 >12），例如，如果时间为上午 9 点，则显示“A”
tt	以 12 小时制显示的 ID：AM（小时数 <12）或 PM（小时数 >12），例如，如果时间为上午 9 点，则显示“AM”
’ ’	包含上述占位符之一的字符串必须用单引号括起来。格式字符串中的所有其他文本可以不带引号，例如，“update”，因为其中包含 1 个“d”和 1 个“t”。

示例

在矩形元素属性 [▶ 353]的“Texts”（文本）类别的“Text”（文本）属性字段中输入“%t['Last update:' ddd MMM dd.MM.yy 'at' HH:mm:ss]”。



在运行时，矩形元素如下所示：



字体和对齐方式

在元素属性 [▶ 353]中可以定义元素文本的字体以及水平和垂直对齐方式。请参见相应元素的“字体变量”和“文本属性”类别。

15.10.3 图像

在 PLC 项目的图像池 [▶ 134]中可以管理来自外部图像文件的图像。基本上，通过 3 种不同的方式可以将它们添加到可视化页面 [▶ 367]：

- 可视化元素“图像 [▶ 461]”
- 可视化元素“图像切换器 [▶ 479]“
- 可视化页面的背景图像 [▶ 345]

可视化元素“图像”

元素”图像 [▶ 461]“允许在可视化页面上静态显示外部图像文件或动态显示不同的外部图像文件。为此，您可以在元素设置中输入 1 个静态图像 ID [▶ 462] 或 1 个字符串变量 [▶ 399]，在其中存储要在运行时显示的图像的 ID。

可视化元素”图像切换器“

”图像切换器 [▶ 479]“元素可用于创建 1 个用户定义的按钮，它可能具有与标准交换机 [▶ 479]类似的按钮或切换功能。在元素属性 [▶ 480]中可以为“开启”、“关闭”和“按下” 3 种状态定义不同的图像。

可视化页面的背景图像

您可以为每个可视化页面设置背景图像 [▶ 345]。客户端的可视化页面的大小 [▶ 367]可能会根据该背景图像进行自动调整。

在项目属性 [▶ 367]中的“Visualization”（可视化）类别中可以定义文件夹，它可以提供图像文件供可视化使用。

15.10.4 在线模式下的键盘操作

每个设备都支持一些标准的键盘快捷键，以便在在线模式下对可视化进行键盘操作。

标准键盘快捷键：

按键	动作
[Tab]	根据插入的时间顺序选择下一个元素；在表格中，每个单元格均可供选择；如果选中 1 个边框元素，则会将该选择传递到它所包含的各个元素。
[Shift + Tab]	选择前一个元素；顺序与 [Tab] 相反
[Enter]	对所选元素执行输入动作
[Arrow keys]	按箭头键指定的方向选择下一个元素

对于集成的可视化，使用命令“Activate keyboard operation”（激活键盘操作）可以明确激活或停用键盘操作。最好这样做，因为只要启用可视化的键盘操作，就不会执行键盘快捷键给出的其他命令。

您可以选择在应用程序代码中处理键盘事件。

16 引用编程

16.1 编程语言及其编辑器

针对您在创建 POU 时所选择的实现语言，您可以在编辑器中对 POU 进行编程。TwinCAT 3 PLC 为 ST 提供了文本编辑器，为 SFC、FBD/LD/IL（文本）和 CFC 提供了图形编辑器。

双击 PLC 项目树中的 POU 或使用上下文菜单中的命令**Open**（打开），您可以打开编辑器。

每个编程语言编辑器均由 2 个子窗口组成：

- 上部可用于声明编辑器中的声明，根据设置的不同，声明可以采用文本形式或表格形式。
- 您可以在下部插入相应语言的实现代码。

在 TwinCAT 选项的相应选项卡中可以为整个项目配置每个编辑器的显示和行为。

16.1.1 声明编辑器

在声明编辑器中，您可以在变量列表和 POU 中声明变量。

如果声明编辑器与编程语言编辑器一起使用，它将在编程语言编辑器上方的窗口中打开。

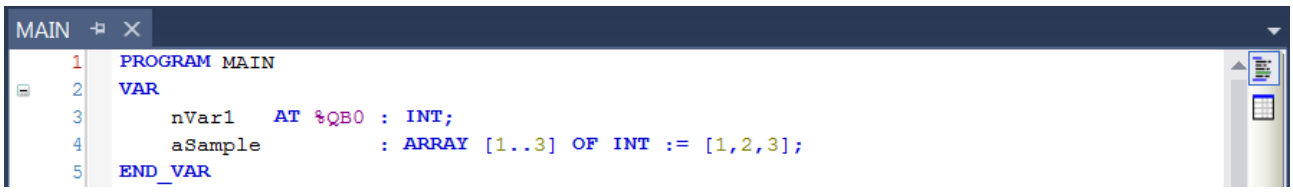
声明编辑器提供 2 种可能的视图：文本视图（ ）和表格视图（ ）。在对话框 **Tools** > **Options** > **TwinCAT** > **PLC Environment** > **Declaration editor**（工具 > 选项 > TwinCAT > PLC 环境 > 声明编辑器）中，您可以定义是否仅提供文本视图或表格视图，或者用户是否可以通过编辑器窗口右侧的按钮在 2 种视图之间进行选择。

在声明编辑器的文本视图中可以进行矩形选择。有关矩形选择的快捷键，请参见 ST 编辑器 [▶ 571]章节。

基于文本的声明编辑器

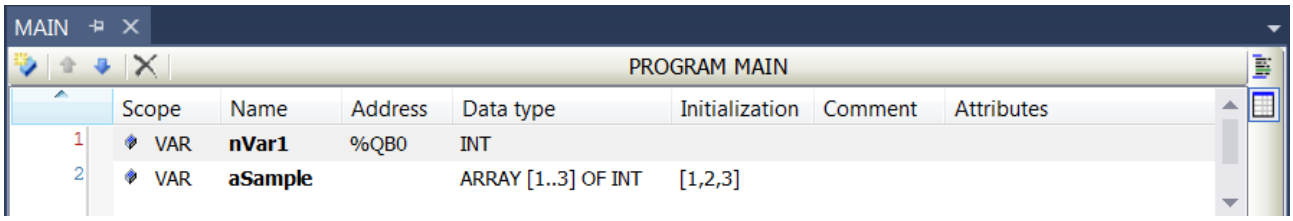
通过对话框 **Tools** > **Options** > **TwinCAT** > **PLC Environment** > **Text editor**（工具 > 选项 > TwinCAT > PLC 环境 > 文本编辑器）中的设置，您可以配置文本编辑器的行为和外观。这些设置涉及颜色、行号、制表符宽度、缩进等。在文本编辑器中，您可以使用普通的 Windows 功能以及 IntelliMouse 功能（如适用）。

您也可以在文本声明编辑器中使用注释（请参见 ST 注释 [▶ 580]）。



表格声明编辑器

在表格声明编辑器中，您可以在带有以下各列的表格中插入变量声明：范围、名称、地址、数据类型、初始化、注释和（编译指示）属性。



在线模式下的声明编辑器

在在线模式下，您可以看到编辑器的表格视图。标题始终包含当前对象路径：<device name>.<project name>.<object name>。与离线模式不同的是，该表格还包含 **Value**（值）和 **Prepared value**（准备值）列。

值	在控制器上显示实际值，从而提供监控功能。 如果表达式是 1 个具有超过 1000 个元素的数组，则您可以指定要监控的数组索引的范围。为此，可双击 **Data type**（数据类型）列，打开对话框 Monitoring area [▶ 207]（监控区域）。在该对话框中，可输入已声明的数组范围作为监控的有效范围。每个数组最多可显示 20000 个元素。 通过指定起始和结束索引，您可以定义要显示的数组索引的范围。如要在保持范围不变的情况下更方便地移动该区域，您可以使用为此目的而耦合的滚动条。如要在链接滚动条  和非链接滚动条  之间切换，可点击滚动条右侧的图标。在非耦合状态下，您可以随意增大或缩小要显示的区域的范围。
准备值	包含您可能已准备用于强制或写入 [▶ 198]的值。 如果您双击 **Prepared value**（准备值）列中的字段，则您可以直接输入 1 个用于写入或强制的值。 对于枚举，则会打开 1 个组合框，您可以从中选择 1 个值。 通过 Boolean 变量，您可以借助 **[Enter]** 或 **[Space]** 键切换（转换）准备值。 如果表达式（变量）属于结构化数据类型，例如，功能块实例或数组变量，则可在其前面加上加号或减号。 在上下文菜单中，您可以调整浮点数的表示格式和继承层次结构的表示格式。

另请参见：

- 声明变量 [▶ 60]
- 声明部分中的强制 [▶ 200]
- 文档 TC3 用户界面：选项对话框 - 声明编辑器 [▶ 899]
- 文档 TC3 用户界面：准备对话框值 [▶ 893]

16.1.2 图形编辑器中的常用函数

FBD、LD、CFC 和 SFC 的图形编辑器的实现部分在右下角有 1 个工具栏：

	返回正常编辑模式：鼠标指针会变为默认的箭头形状。您可以再次在编辑器窗口中选择和编辑元素。
	平移工具：鼠标指针会变为 2 个交叉箭头的形状。现在，您可以点击编辑器窗口中的任意位置，按住鼠标按钮，然后，在窗口中移动 FBD/LD/IL 编辑器或 CFC 图表的可见区域。
	放大镜工具：在编辑器窗口的右下角会打开 1 个额外的放大镜窗口，鼠标指针会变为 2 个交叉箭头的形状。现在，只要您将鼠标指针移到图表上，放大镜就会以 100% 的大小显示指针周围的图表区域。如果您点击进入该窗口，放大镜就会关闭，其中显示的图表部分会以 100% 的大小显示。因此，如果您想保留已设置的缩放因子，您应该使用  返回正常编辑模式。
	缩放工具：该按钮可打开 1 个子菜单，用于选择缩放因子。选择 3 个点 ... 可打开缩放对话框，您可以在其中输入不同的值。在按钮的左侧始终会显示当前的缩放因子。

使用鼠标滚轮放大/缩小：如果您在移动鼠标滚轮的同时按下 **[Ctrl]** 按钮，则缩放因子会以 10% 的幅度变化。

每个图形编辑器都有 1 个工具箱（**Toolbox**（工具箱）视图），默认位于编辑器窗口的右侧。工具箱中包含一些可用的元素，您可以用鼠标将这些元素拖到编辑器窗口中相应的插入位置。TwinCAT 用灰色菱形、三角形或箭头形位置标记来突出显示相应的插入位置。只要您将鼠标指针移到其中 1 个标记上，该标记就会变成绿色。当您松开鼠标按钮后，TwinCAT 就会在当前位置插入元素。

您也可以用鼠标在编辑器中移动元素。

您可以将功能块声明拖到 FBD、LD 和 CFC 图形编辑器中的编辑器窗口中。为此，可选择完整的声明（变量名和数据类型），并将其拖到编辑器窗口中的合适位置。通过梯形图，您还可以将 Boolean 声明拖到编辑器中，并将其作为触点插入。

另请参见：

- SFC 编辑器 [▶ 580]

- FBD/LD/IL 编辑器 [▶ 594]
- CFC 编辑器 [▶ 607]

16.1.3 结构化文本和扩展结构化文本（ExST）

16.1.3.1 ST 编辑器

ST 编辑器是 1 个文本编辑器，可用于实现结构化文本（ST）和扩展结构化文本（ExST）中的代码。

行号位于编辑器的左侧边缘。在输入编程元素时，**List components**（列出组件）功能和输入助手（**F2**）将为您提供帮助。如果光标位于变量上，则 TwinCAT 会在工具提示中显示变量声明的相关信息。

您可以使用以下快捷键执行矩形选择：

- **[Shift] + [Alt] + [Right arrow]**：所选区域将向右扩展 1 个位置。
- **[Shift] + [Alt] + [Left arrow]**：所选区域将向左扩展 1 个位置。
- **[Shift] + [Alt] + [Up arrow]**：所选区域将向上扩展 1 行。
- **[Shift] + [Alt] + [Down arrow]**：所选区域将向下扩展 1 个位置。

在按住 **Ctrl** 键的同时转动鼠标滚轮，您可以放大编辑器窗口。

在 TwinCAT 选项的 **PLC Environment > Text editor**（PLC 环境 > 文本编辑器）类别中，您可以配置编辑器的行为（例如，括号、鼠标动作、制表符）和外观。

为了在编辑器内对字符串进行增量搜索，可使用快捷键 **[Ctrl] + [Shift] + [i]** 打开编辑器底部的输入字段。只要您开始输入字符串，编辑器就会以彩色突出显示相应的匹配项。在输入字段的右侧会显示找到的匹配数。使用箭头按钮或快捷键 **[Alt] + [Page up]** 或 **[Alt] + [Page down]**，您可以将光标置于找到的位置上。

当您将光标置于符号名称上时，编辑器中使用该符号的所有位置都会突出显示。引用与交叉引用列表中的查询结果相对应。对于非常大的项目，这可能会导致输入延迟。在这种情况下，您可以在文本编辑器的选项中禁用该功能。

TwinCAT 可以在编辑的过程中识别语法错误，并用红色波浪线在错误处加下划线。前提条件是，您需要在 TwinCAT 选项的 **PLC Environment > Smart Coding**（PLC 环境 > 智能编码）类别中激活相应的选项。在编译过程中检测到的任何语法错误都会在 **Error list**（错误列表）（**View**（视图）菜单）中显示。

另请参见：

- 编程结构化文本（ST） [▶ 113]
- ST 表达式 [▶ 572]
- 赋值 [▶ 573]
- 指令 [▶ 576]
- TC3 用户界面文档：对话框选项 - 文本编辑器 [▶ 914]

16.1.3.2 在线模式下的 ST 编辑器

在在线模式下，TwinCAT 会显示在 ST 编辑器中使用的变量和表达式。您还可以使用编写和强制变量与表达式以及调试功能（断点、单步处理）。

如果您在 ST 编程中使用赋值作为表达式，则在 1 行内不会再出现断点位置。

另请参见：

- 编程对象中的监控 [▶ 205]
- 使用断点 [▶ 195]
- 使用监视列表 [▶ 209]
- 强制和写入变量值 [▶ 198]
- 测试 PLC 项目和故障排除 [▶ 195]
- 逐步处理程序（步进） [▶ 197]

- 流程控制 [▶ 202]
- 使用调用堆栈确定当前处理位置 [▶ 204]

16.1.3.3 ST 表达式

表达式是 1 种在求值后返回值的结构。

表达式由运算符和操作数组成。操作数可以是常量、变量、函数调用或其他表达式。

示例：

2014	(* 常量 *)
nVar	(* 变量 *)
F_Fct (a, b)	(* 函数调用 *)
(x*y)/z	(* 表达式 *)

另请参见：

- 运算符 [▶ 635]
- 操作数 [▶ 686]

表达式的求值

根据特定的绑定规则处理运算符，从而执行表达式的求值。TwinCAT 首先会处理具有最强绑定关系的运算符。然后，从左到右依次处理绑定强度相同的运算符。

操作	符号	绑定强度
加括号	(表达式)	最强绑定关系
函数调用	函数名称(参数列表) 采用该语法的所有运算符： <operator> ()	
乘方	EXPT	
否定 构建。补充	- NOT	
乘法 除法 模数	* / MOD	
加法 减法	+ -	
比较	<、>、<=、>=	
等于 不等于	= <>	
Bool AND	AND AND_THEN	
Bool XOR Bool OR	XOR OR OR_ELSE	最弱绑定关系

示例：

在下面的示例中，使用了 AND_THEN [▶ 652] 和 OR [▶ 652] 运算符。请注意，OR 的绑定关系最弱，只有当 AND_THEN 运算符的第 1 个操作数为 TRUE 时，TwinCAT 才会在 AND_THEN 运算符的其他操作数处执行表达式。

因此，此处所示的 4 个表达式的结果如下：

- 这 4 个表达式都没有解除引用指针“pSample”。背景是使用 AND_THEN 运算符，而解除引用将发生在 AND_THEN 运算符的其他操作数上。不过，由于 AND_THEN 运算符的第 1 个操作数已经返回 FALSE（因为“pSample”的值为 0），所以不会执行后面的 AND_THEN 操作数和指针解除引用。
使用 AND_THEN 运算符可以避免在运行时出现空指针异常。如果使用 AND 运算符而不是 AND_THEN 运算符，则会解除引用指针“pSample”作为操作数中的空指针，从而导致空指针异常。
- 在表达式 1 和 2 中，OR 作为 AND_THEN 运算之后的运算符。因此，如果“pSample”为 0 且“bTest”为 TRUE，则计数器“nCounter1”和“nCounter2”就会递增。表达式的缩写是“IF <FALSE> OR TRUE THEN”，返回 TRUE。
- 另一方面，在表达式 3 和 4 中，如果“pSample”为 0 且“bTest”为 TRUE，则“nCounter3”和“nCounter4”不会递增，因为第 1 个操作数“pSample <> 0”返回 FALSE。由于 AND_THEN 和括号的作用，不再检查其他操作数或运算符。表达式的缩写形式是“IF <FALSE> AND_THEN <...>”，返回 FALSE。

```
PROGRAM MAIN
VAR
    pSample    : POINTER TO INT;
    bTest      : BOOL := TRUE;
    nCounter1  : INT;
    nCounter2  : INT;
    nCounter3  : INT;
    nCounter4  : INT;
END_VAR
```

```
// Expression 1
IF (pSample <> 0) AND_THEN (pSample^ = 250) AND_THEN (pSample^ <> 300) OR bTest THEN
    nCounter1 := nCounter1 + 1;   // increasing if (pSample = 0) and (bTest = TRUE)
END_IF

// Expression 2
IF ((pSample <> 0) AND_THEN (pSample^ = 250) AND_THEN (pSample^ <> 300)) OR bTest THEN
    nCounter2 := nCounter2 + 1;   // increasing if (pSample = 0) and (bTest = TRUE)
END_IF

// Expression 3
IF (pSample <> 0) AND_THEN ((pSample^ = 250) AND_THEN (pSample^ <> 300) OR bTest) THEN
    nCounter3 := nCounter3 + 1;   // not increasing if (pSample = 0) and (bTest = TRUE)
END_IF

// Expression 4
IF (pSample <> 0) AND_THEN ((pSample^ = 250) OR bTest) THEN
    nCounter4 := nCounter4 + 1;   // not increasing if (pSample = 0) and (bTest = TRUE)
END_IF
```

16.1.3.4 赋值

16.1.3.4.1 ST 赋值运算符

语法：

<operand> := <expression>

该赋值运算符的功能与 MOVE 运算符相同。

另请参见：

- MOVE [▶ 645]

16.1.3.4.2 输出的 ST 赋值运算符

赋值运算符 => 将函数、功能块或方法的输出赋值给变量。运算符右侧的空间可以为空。

语法：

<output> => <variable>

示例：

```
bFBCompOutput1 => bVar1;
bFBCompOutput2 => ;
```

bFBCompOutput1 和 bFBCompOutput2 是功能块的输出。bFBCompOutput1 的值被分配给变量 bVar1。

16.1.3.4.3 ExST 赋值 S=

如果设置赋值的操作数切换为 TRUE，则该赋值会导致运算符左侧的变量被赋值为 TRUE。变量已设定。

语法：

<variable name> S= <operand name> ;

变量和操作数的数据类型均为 BOOL。

示例：

```
PROGRAM MAIN
VAR
    bOperand     : BOOL := FALSE;
    bSetVariable : BOOL := FALSE;
END_VAR
```

```
bSetVariable S= bOperand;
```

如果操作数 bOperand 从 FALSE 切换到 TRUE，则变量 bSetVariable 将被赋值为 TRUE。但是，即使操作数继续改变状态，变量也会保持该状态。

多重赋值

如果代码行内有多重赋值，则不会从右到左处理各个赋值，而是将所有赋值都指向代码行末尾的操作数。

示例：

```
FUNCTION F_Sample: BOOL
VAR_INPUT
    bIn : BOOL;
END_VAR
```

```
IF bIn = TRUE THEN
    F_Sample := TRUE;
    RETURN;
END_IF
```

```
PROGRAM MAIN
VAR
    bSetVariable   : BOOL;
    bResetVariable : BOOL := TRUE;
    bVar           : BOOL;
END_VAR
```

```
bSetVariable S= bResetVariable R= F_Sample(bIn := bVar);
```

bResetVariable 接收 F_Sample 返回值的 R= 赋值。bSetVariable 接收 F_Sample 返回值的 S= 赋值，而不是 bResetVariable。

16.1.3.4.4 ExST 赋值 R=

如果重置赋值的操作数切换为 TRUE，则该赋值会导致运算符左侧的变量被赋值为 FALSE。变量已重置。

语法：

<variable name> R= <operand name> ;

变量和操作数的数据类型均为 BOOL。

示例：

```
PROGRAM MAIN
VAR
    bOperand       : BOOL := FALSE;
    bResetVariable : BOOL := TRUE;
END_VAR
```

```
bResetVariable R= bOperand;
```

如果操作数 bOperand 从 FALSE 切换为 TRUE，则变量 bResetVariable 将被赋值为 FALSE。但是，即使操作数一直改变状态，变量也会保持其状态。

多重赋值

如果代码行内有多重赋值，则不会从右到左处理各个赋值，而是将所有赋值都指向代码行末尾的操作数。

示例：

```
FUNCTION F_Sample: BOOL
VAR_INPUT
    bIn : BOOL;
END_VAR
```

```
IF bIn = TRUE THEN
    F_Sample := TRUE;
    RETURN;
END_IF
```

```
PROGRAM MAIN
VAR
    bSetVariable   : BOOL;
    bResetVariable : BOOL := TRUE;
    bVar           : BOOL;
END_VAR
```

```
bSetVariable S= bResetVariable R= F_Sample(bIn := bVar);
```

bResetVariable 接收 F_Sample 返回值的 R= 赋值。bSetVariable 接收 F_Sample 返回值的 S= 赋值，而不是 bResetVariable。

16.1.3.4.5 ExST 赋值作为表达式

在 ExST 中，TwinCAT 允许将赋值用作表达式，这是对 IEC 61131-3 标准的扩展。

示例：

`nVarInt1 := nVarInt2 := nVarInt3 + 9;`	(*nVarInt1 和 nVarInt2 获得 nVarInt3 的值 + 9*)
`fVarReal1 := fVarReal2 := nVarInt;`	(*fVarReal1 和 fVarReal2 获得 nVarInt 的值*)
`nVarInt:= fVarReal1:= nVarInt;`	(*赋值不正确，数据类型不匹配！*)

`IF b := (i = 1) THEN`
`  i := i + 1;`
`END_IF`
	(*b 获得 Boolean 表达式 i = 1 的值，然后在 if 查询中进行检查*)

16.1.3.4.6 赋值运算符 REF=

运算符生成 1 个指向值的引用 [▶ 712]（指针）。

语法：

<variable name> REF= <variable name>

示例：

```
PROGRAM MAIN
VAR
    refA  : REFERENCE TO ST_Sample;
    stA   : ST_Sample;
    refB  : REFERENCE TO ST_Sample;
    stB1  : ST_Sample;
    stB2  : ST_Sample;
END_VAR
```

```
refA REF= stA;   // represents => refA  := ADR(stA);
refB REF= stB1;  // represents => refB  := ADR(stB1);
refA := refB;    // represents => refA^ := refB^; (value assignment of refB as refA and refB are
implicitly dereferenced)
refB := stB2;    // represents => refB^ := stB2; (value assignment of stB2 as refB is implicitly
dereferenced)
END_VAR
```

另请参见：

- 引用 [▶ 712]

16.1.3.5 指令

16.1.3.5.1 ST 指令 IF

IF 语句用于测试 1 个条件，并在满足该条件时执行后续指令。

条件被编码为返回 Boolean 值的表达式 [▶ 572]。如果表达式返回 TRUE，则满足该条件并执行 THEN 后面的相关语句。如果表达式返回 FALSE，则会对之后标有 ELSIF 的条件进行评估。如果 ELSIF 条件返回 TRUE，则执行相关 THEN 后面的指令。如果所有条件均为 FALSE，则执行 ELSE 后面的语句。

因此，最多只能执行 IF 语句的 1 个分支。ELSIF 分支和 ELSE 分支是可选的。

语法

```
IF <condition> THEN
        <statements>
(ELSIF <condition> THEN
        <statements>)*
(ELSE
        <statements>)?
END_IF;

// ( ... )* None, once or several times
// ( ... )? Optional
```

示例：

```
PROGRAM MAIN
VAR
    nTemp       : INT;
    bHeatingOn  : BOOL;
    bOpenWindow : BOOL;
END_VAR
```

```
IF nTemp < 17 THEN
    bHeatingOn  := TRUE;
ELSIF nTemp > 25 THEN
    bOpenWindow := TRUE;
ELSE
    bHeatingOn  := FALSE;
    bOpenWindow := FALSE;
END_IF;
```

在运行时，程序按如下方式运行：

如果表达式 nTemp < 17 的求值 = TRUE，则执行之后的指令并开启加热。如果表达式 nTemp < 17 的求值 = FALSE，则对之后的 ELSIF 条件 nTemp > 25 进行求值。如果为 TRUE，则执行 ELSIF 下的语句并打开窗口。如果所有条件均为 FALSE，则执行 ELSE 下的语句。关闭加热，关闭窗户。

另请参见：

- ST 表达式 [▶ 572]

16.1.3.5.2 ST 指令 FOR

FOR 循环可用于执行具有特定重试次数的指令。

语法：

```
FOR <counter> := <start value> TO <end value> {BY <increment> } DO
        <instructions>
END_FOR;
```

大括号 {} 内的部分是可选的。

TwinCAT 执行 <instructions>，直到 <counter> 不大于或（如果步长增量为负数）小于 <end value> 为止。在执行 <instructions> 之前检查这一点。

每当 <instructions> 执行完毕，<counter> 就会自动按步长 <increment> 递增。步长 <increment> 可以是任意整数。如果您未指定步长，则使用默认步长 1。

示例：

```
FOR nCounter := 1 TO 5 BY 1 DO
    nVar1 := nVar1*2;
END_FOR;
nErg := nVar1;
```

如果您已将 nVar1 设置为 1，则在 FOR 循环结束后，nVar1 的值为 32。

FOR 循环的终止值

<end value> 不能与计数器数据类型的上限值相同。

除了 IEC 61131-3 标准之外，您还可以在 FOR 循环中使用 CONTINUE 指令。

另请参见：

- ExST 指令 CONTINUE [▶ 579]
- 整数数据类型 [▶ 699]

16.1.3.5.3 ST 指令 CASE

CASE 指令可用于在 1 个结构中将具有相同条件变量的多个条件指令分组。

语法：

```
CASE <Var1> OF
<value1>:<instruction1>
<value2>:<instruction2>
<value3, value4, value5>:<instruction3>
<value6 ... value10>:<instruction4>
...
<value n>:<instruction n>
{ELSE <ELSE-instruction>}
END_CASE;
```

大括号 {} 内的部分是可选的。

CASE 指令的处理方案：

- 如果变量 <Var1> 的值为 <value i>，则执行 <instruction i>。
- 如果变量 <Var1> 没有任何指定值，则执行 <ELSE instruction>。
- 如果您想要对变量的多个值执行相同的指令，则您可以用逗号分隔这些值。

示例：

```
CASE nVar OF
     1,5 : bVar1 := TRUE;
           bVar3 := FALSE;

     2 : bVar2 := FALSE;
         bVar3 := TRUE;

10..20 : bVar1 := TRUE;
         bVar3 = TRUE;
ELSE
         bVar1 := NOT bVar1;
         bVar2 := bVar1 OR bVar2;
END_CASE;
```

16.1.3.5.4 ST 指令 WHILE

与 FOR 循环一样，WHILE 循环可用于重复执行指令，直到终止条件适用。WHILE 循环的终止条件是 Boolean 表达式。

语法：

```
WHILE <boolean expression> DO
      <instructions>
END_WHILE;
```

只要 Boolean 表达式 <boolean expression> 返回 TRUE，TwinCAT 就会重复执行 <instructions>。如果 Boolean 表达式在第 1 次求值时为 FALSE，则 TwinCAT 永远不会执行指令。如果 Boolean 表达式从未取值 FALSE，则指令将连续重复，从而导致运行时错误。

示例：

```
WHILE nCounter <> 0 DO
    nVar1 := nVar1*2
    nCounter := nCounter-1;
END_WHILE;
```

您必须通过编程确保不会创建无限循环。

从某种意义上说，WHILE 循环和 REPEAT 循环比 FOR 循环更强大，因为在运行循环之前，您不需要知道迭代次数。因此，在某些情况下，只能使用这 2 种类型的循环。但是，如果您知道迭代次数，则最好使用 FOR 循环，以避免无限循环。

除了 IEC 61131-3 标准之外，您还可以在 WHILE 循环中使用 CONTINUE 指令。

另请参见：

- ExST 指令 CONTINUE [▶ 579]
- ST 指令 FOR [▶ 576]

16.1.3.5.5 ST 指令 REPEAT

您使用 REPEAT 循环的方法与使用 WHILE 循环的方法相同，但不同之处在于，TwinCAT 直到执行循环之后才会检查终止条件。这种行为的后果是，无论终止条件是什么，REPEAT 循环都将至少运行 1 次。

语法：

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

TwinCAT 重复执行 <instructions>，直到 <boolean expression> 返回 TRUE。

如果 Boolean 表达式在第 1 次求值时为 TRUE，则 TwinCAT 会执行 1 次指令。如果 Boolean 表达式从未取值 TRUE，则指令将连续重复，从而导致运行时错误。

示例：

```
REPEAT
    nVar1 := nVar1*2;
    nCounter := nCounter-1;
UNTIL
    nCounter = 0
END_REPEAT;
```

从某种意义上说，WHILE 循环和 REPEAT 循环比 FOR 循环更强大，因为在运行循环之前，您不需要知道迭代次数。在某些情况下，您只能使用这 2 种类型的循环。但是，如果您知道迭代次数，则最好使用 FOR 循环，以避免无限循环。

除了 IEC 61131-3 标准之外，您还可以在 WHILE 循环中使用 CONTINUE 指令。

另请参见：

- ExST 指令 CONTINUE [▶ 579]
- ST 指令 FOR [▶ 576]
- ST 指令 WHILE [▶ 578]

16.1.3.5.6 ST 指令 RETURN

您可以使用 RETURN 指令退出功能块。例如，您可以让它依赖于某个条件。

示例：

```
IF bVar1 = TRUE THEN
    RETURN;
END_IF;
nVar2 := nVar2 + 1;
```

如果 bVar1 的值为 TRUE，则立即退出功能块，而且 TwinCAT 不会执行指令 nVar2 := nVar2 + 1;。

另请参见：

- ST 指令 IF [▶ 576]

16.1.3.5.7 ST 指令 JMP

JMP 指令可用于执行向 1 个以标签标记的行的无条件跳转。

语法：

```
<label>: <instructions>
JMP <label>;
```

<label> 是 1 个可自由选择的唯一标识符，您可以将它放在 1 行的开头。在执行 JMP 指令时，可触发跳回到带有 <label> 的行。

示例：

```
nVar1 := 0;
_label1 : nVar1 := nVar1+1;
(*instructions*)
IF (nVar1 < 10) THEN
    JMP _label1;
END_IF;
```

您必须通过编程确保不会创建无限循环。例如，您可以在跳转和条件之间建立关联。

16.1.3.5.8 ST 指令 EXIT

尽管有其他终止条件，但是，在 FOR、WHILE 或 REPEAT 循环中使用 EXIT 指令可退出循环。

另请参见：

- ST 指令 FOR [▶ 576]
- ST 指令 REPEAT [▶ 578]
- ST 指令 WHILE [▶ 578]

16.1.3.5.9 ExST 指令 CONTINUE

CONTINUE 是扩展结构化文本（ExST）指令。

在 FOR、WHILE 和 REPEAT 循环中使用该指令，可触发跳转到下一个循环的起点。

示例：

```
FOR nCounter :=1 TO 5 BY 1 DO
    nInt1:=nInt1/2;
    IF nInt1=0 THEN
        CONTINUE; (* to avoid a division by zero *)
    END_IF
    nVar1:=nVar1/nInt1; (* executed, if nInt1 is not 0 *)
END_FOR;
nRes:=nVar1;
```

另请参见：

- ST 指令 FOR [▶ 576]
- ST 指令 WHILE [▶ 578]
- ST 指令 REPEAT [▶ 578]

16.1.3.5.10 ST 调用功能块

语法：

<FB-instance>(<FB input variable>:=<value or adress>|, <other FB input variables>);

示例：

```
fbTMR : TON;
fbTMR (IN := %OX5, PT := T#300ms);
bVarA := fbTMR.Q;
```

计时器功能块 TON 在 fbTMR : TON; 中实例化，并通过参数 IN 和 PT 的赋值进行调用。

输出 Q 使用 fbTMR.Q 进行寻址，并被赋值给变量 bVarA。

另请参见：

- 对象功能块 [▶ 76]

16.1.3.5.11 ST 注释

注释	描述	示例：
单行	以 // 开头，在行尾结束。	`// Dies ist ein Kommentar`
多行	以 (* 并以此结尾 *) 开头。	`(* Dies ist ein mehrzeiliger Kommentar *)`
嵌套	以 (* 并以此结尾 *) 开头。附加 (*... *) 注释可能会包含在该注释中。	`(* a:=fbTest.out; (* 1.Kommentar *) b:=b+1; (* 2.Kommentar *)*)`

此外，您还可以使用菜单命令或快捷键注释/注释掉 1 个或多个标记的行：

- 命令注释选择 [▶ 817]
- 命令取消注释选择 [▶ 818]

工具提示注释

您可以在 POU 声明上方插入注释，为 POU 创建工具提示。

示例：

```
// Das ist ein Beispiel-Funktionsbaustein
FUNCTION_BLOCK FB_Sample
VAR_INPUT
END_VAR
```

16.1.4 顺序功能图（SFC）

16.1.4.1 SFC 编辑器

SFC 编辑器是 1 个图形编辑器。新添加的 SFC POU 包含 1 个 init 步骤和 1 个后续转换。

在 SFC 编辑器中，您可以使用 **SFC** 菜单、上下文菜单或 **Toolbox**（工具箱）视图中的命令将各个元素插入图表中。

在使用菜单命令进行插入时，可以插入到当前所选位置的元素可供选择。

在插入与多个动作和转换并行的分支之前，您需要通过多项选择来选择这些动作和转换。

您还可以将 **Toolbox**（工具箱）视图中的 SFC 元素拖放到图表中。只要您将元素拖到编辑器中，TwinCAT 就会用灰色矩形标记所有可能的插入位置。如果您将鼠标移到可能的插入位置上，矩形的颜色会变为绿色。如果您松开鼠标按钮，则会在此位置插入元素。

如果您使用拖放的方式插入分支，则您必须用鼠标标记分支的起点和终点。您可以在插入位置松开鼠标，从而标记分支的起点。矩形的颜色会变为红色。点击第 2 个插入位置，即可定义分支的终点。然后，TwinCAT 会在开始标记和结束标记之间的对象周围插入 1 个分支。

在复制用于调用动作对象或转换对象的步骤和转换元素时，您可以设置 2 种不同的复制模式。在复制的过程中，您可以复制引用或者“嵌入”和复制引用对象。

在 TwinCAT 选项的 **TwinCAT > PLC Environment > SFC editor**（TwinCAT > PLC 环境 > SFC 编辑器）类别中，您可以定义编辑器的行为和外观。

另请参见：

- 图形编辑器中的常用函数 [▶ 570]
- 用顺序功能图（SFC）编程 [▶ 114]
- TC3 用户界面文档：SFC [▶ 929]
- TC3 用户界面文档：对话框选项 - SFC 编辑器 [▶ 907]

16.1.4.2 在线模式下的 SFC 编辑器

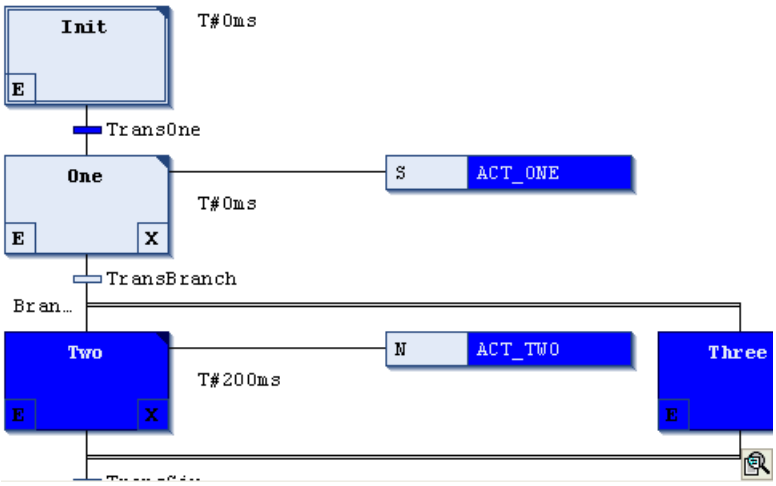
在 SFC 编辑器中，您可以显示在控制器上在运行时使用的变量和表达式。您还可以编写和强制变量和表达式。调试功能（断点、逐步执行等）尚不可用。

在 SFC 编辑器的选项中，您可以对 SFC 元素和属性的在线表示进行设置。

如果您已明确声明 SFC 标志，则在在线模式下，这些标志将在声明部分中显示。在脱机模式下，则不会显示它们。

请注意 SFC 图中的元素处理顺序。

在在线模式下，TwinCAT 会以蓝色显示活动步骤。



另请参见：

- 隐式变量 [▶ 583]
- SFC 中的处理顺序 [▶ 582]

16.1.4.3 SFC 中的处理顺序

元素的基本行为

- 活动步骤：活动步骤是指当前正在执行动作的步骤。在在线模式下，TwinCAT 会以蓝色显示活动步骤。
- 初始步骤：在调用 SFC POU 后的第 1 个周期中，系统会自动激活初始步骤并执行步骤动作。
- TwinCAT 至少会执行 2 次 IEC 动作。第 1 次在激活步骤之后，第 2 次（但仅限在下一个周期中）是在停用步骤之后。
- 替代分支：如果分支前面的步骤处于活动状态，则 TwinCAT 将从左到右评估每个替代分支的第 1 个转换。在第 1 个分支中，如果 TwinCAT 找到 1 个返回 TRUE 的转换，则会激活下一个步骤。
- 平行分支：如果分支前面的步骤（在水平双线之前）处于活动状态，且分支前的转换返回 TRUE，则 TwinCAT 会在所有分支中激活第 1 个步骤。然后，同时处理这些分支。当分支中的所有最后步骤均处于活动状态且在双线之后的转换返回 TRUE 时，系统才会激活在分支终点后（在水平双线之后）的步骤。

处理顺序

1. **重置 IEC 动作**
   TwinCAT 会重置动作限定符（N、R、S、L、D、P、SD、DS、SL）的内部动作控制标志。这些是控制 IEC 动作的标志。但是，在动作中调用的标记不会被重置！
2. **执行退出动作**
   然后，TwinCAT 会检查所有步骤是否满足执行退出动作的条件。检查顺序与 SFC 图中的顺序相对应：从上到下，从左到右。
   在停用步骤时，即在前一个周期中已执行其输入和步骤动作（如果可用）且后续步骤的条件返回 TRUE 时，TwinCAT 会执行退出动作。
3. **执行进入动作**
   然后，TwinCAT 会检查所有步骤是否满足执行进入动作的条件。检查顺序与 SFC 图中的顺序相对应：从上到下，从左到右。如果是这种情况，则 TwinCAT 会执行进入动作。
   一旦处理到达步骤之前的转换，并且该转换返回 TRUE，即步骤被激活，则 TwinCAT 会执行进入动作。
4. **时间检查，执行步骤动作**
   TwinCAT 会按照 SFC 图中的顺序对所有步骤执行动作：
   - TwinCAT 会将活动步骤的经过时间复制到相应的隐式步骤变量 <step name>.t 中。
   - 如果出现超时，则 TwinCAT 会设置相应的错误标志。
   - 对于非 IEC 步骤，TwinCAT 现在会执行步骤动作。
5. **执行 IEC 动作**
   TwinCAT 会按字母顺序执行 IEC 动作。这需要在动作列表中完成 2 个周期。在第 1 次运行中，TwinCAT 会执行在上一个周期中停用的所有步骤的 IEC 动作。在第 2 次运行中，则会执行处于活动状态的所有步骤的 IEC 动作。
6. **转换检查，激活后续步骤**
   对转换进行评估：如果 1 个步骤在当前周期内处于活动状态，且后续转换返回 TRUE（且已经过可能定义的最短步骤时间），则会激活后续步骤。

在执行动作时，请注意以下几点：
在同一个周期内，可能会将 1 个动作执行多次，因为它与多个进程相关联。（示例：1 个 SFC 包含 2 个 IEC 动作 A 和 B，它们都在 SFC 中进行编程，并且都调用 1 个 IEC 动作 C。在这种情况下，A 和 B 可以在同一个周期的 IEC 动作中都处于活动状态，而 IEC 动作 C 可以在 2 个动作中都处于活动状态。在这种情况下，C 将被调用 2 次。）
如果在 SFC 图的不同层次同时使用相同的 IEC 动作，则可能会在执行过程中产生不可预测的效果，因此会发出相应的错误消息。在处理使用旧版本编程系统创建的项目时，可能会出现这些问题。

请注意，使用隐式变量可以监控或控制步骤和动作的处理状态。

另请参见：

- 隐式变量 [▶ 583]
- SFC 中的动作限定符 [▶ 583]

16.1.4.4 SFC 中的动作限定符

您可以为限定符分配 IEC 步骤。限定符描述了如何执行与步骤相关的动作。

限定符由系统库的功能块 SFCActionControl 处理。SFC 插件会自动将库集成到项目中。

可用的限定符：

N	非存储	只要步骤处于活动状态，动作就处于活动状态。
R	覆盖重置	动作已停用。
S	设置（存储）	一旦步骤进入活动状态，TwinCAT 就会立即执行动作。即使步骤已停用，仍会继续执行动作，直到重置为止。
L	时间限制	一旦步骤进入活动状态，TwinCAT 就会立即执行动作。动作将一直执行，直到步骤进入非活动状态或经过指定的时间间隔。
D	时间延迟	在步骤进入活动状态后，直到经过给定的延迟时间并且步骤仍处于活动状态时，TwinCAT 才会开始执行动作。动作将一直执行，直到步骤停用。
P	脉冲	TwinCAT 会精确地执行 2 次动作：1 次是在步骤进入活动状态时，另一次是在随后的周期中。
SD	存储和时间延迟	在步骤进入活动状态后，直到经过指定的延迟时间，TwinCAT 才会开始执行动作。动作将一直执行，直到它接收到重置。
DS	延迟和存储	在步骤进入活动状态后，直到经过给定的延迟时间并且步骤仍处于活动状态时，TwinCAT 才会开始执行动作。动作将一直执行，直到它接收到重置。
SL	存储和时间限制	一旦激活步骤，TwinCAT 就会立即执行动作。动作将一直执行，直到经过指定的时间或接收到重置。

您必须以 TIME 常量的格式为限定符 L、D、SD、DS 和 SL 指定时间规范。

在停用 IEC 动作后，该动作将再执行 1 次。这意味着 TwinCAT 至少要执行 2 次这样的动作。这也涉及带有 P 限定符的动作。

另请参见：

- 用顺序功能图（SFC）编程 [▶ 114]

16.1.4.5 隐式变量

每个 SFC 对象都会提供隐式变量，您可以使用这些变量在运行时监控步骤和 IEC 动作的状态。TwinCAT 会为每个步骤和 IEC 动作自动创建这些隐式变量。

隐式变量是 SFCStepType（用于步骤）或 SFCActionType（用于动作）类型的结构实例。这些变量具有元素的名称，例如，“step 1”表示步骤名称为“step1”的步骤。结构组件描述了 1 个步骤或动作的状态，或者在 1 个活动步骤中经过的时间。

步骤和动作状态

隐式变量声明的语法：

<stepname>:SFCStepType;

_<actionname>:SFCActionType;

下列隐式变量可用于步骤或 IEC 动作状态：

步骤	
<stepname>.x	显示当前周期中的激活状态。 如果 <stepname>.x = TRUE，则 TwinCAT 会在当前周期中执行该步骤。
<stepname>._x	显示下一个周期的激活状态。 如果 <stepname>._x = TRUE 且 <stepname>.x = FALSE，则 TwinCAT 会在下一个周期中执行该步骤，即 在周期开始时，将 <stepname>._x 复制到 <stepname>.x 中。
<stepname>.t	标志 t 返回当前时间间隔，该时间间隔是步骤进入活动状态后的时间间隔。这只适用于步骤，无论在步骤属性中是否定义了最短时间。 另请参见 SFC 标志 SFCError。
<stepname>._t	仅供内部使用
IEC 动作	
_<actionname>.x	如果已执行动作，则为 TRUE。
_<actionname>._x	如果已激活动作，则为 TRUE。

您可以使用上述变量来强制某个步骤的特定状态值，也就是将某个步骤设置为活动状态。但请注意，这会导致 SFC 处于失控状态。

访问隐式变量

访问的语法：

在 POU 中，您可以直接分配隐式变量：<variable name>:=<step name>.<implicit variable> 或 <variable name>:=_<action name>.<implicit variable>

示例：

```
status := step1._x;
```

来自另一个具有 POU 名称的功能块：<variable name>:=<POU name>.<step name>.<implicit variable> 或 <variable name>:=<POU name>._<action name>.<implicit variable>

示例：

```
status := SFC_prog.step1._x;
```

另请参见：

- SFC 元素属性 [▶ 593]
- SFC 标志 [▶ 584]

16.1.4.6 SFC 标志

SFC 标志是隐式生成的变量，具有预定义的名称。您可以使用它们来影响 SFC 图的处理。例如，您可以使用这些标志来指示超时或重置步骤顺序。例如，您还可以激活点动模式，有选择地切换转换。如要访问这些变量，您必须声明并激活它们。

SFC 标志

名称	数据类型	描述
SFCInit	Bool	TRUE：TwinCAT 将序列重置为初始步骤。其他 SFC 标志也会被重置（初始化）。只要变量为 TRUE，初始步骤就会保持设置（活动），但不会执行。只有您将 SFCInit 重新设置为 FALSE 时，才能继续正常处理功能块。
SFCReset	Bool	行为类似于 SFCInit。然而，相比之下，TwinCAT 会在初始化后继续处理初始步骤。例如，您可以在 init 步骤中直接将 SFCReset-yyyyy 标志设置为 FALSE。
SFCError	Bool	如果 SFC 图发生超时，则会变为 TRUE。如果在第 1 次超时后，在程序中再次发生超时，在您先前没有重置变量 SFCError 的情况下，则将不再记录这种情况。SFCError 的声明是控制时序的其他标志变量（SFCErrorStep、SFCErrorPOU、SFCQuitError）发挥作用的前提条件。
SFCEnableLimit	Bool	用于 SFCError 分步骤对超时控制进行特定激活（TRUE）和停用（FALSE）。在声明和激活该变量（SFC 设置）时，您还必须将其设置为 TRUE，以便 SFCError 发挥作用，否则超时将被忽略。例如，在调试或手动操作期间，这可能会很有用。如果您未声明变量，则 SFCError 会自动发挥作用。 前提条件是声明 SFCError。
SFCErrorStep	字符串	保存导致超时的步骤的名称，该超时由 SFCError 记录。在 SFCQuitError 重置记录的超时之前，将一直存储该名称。 前提条件是声明 SFCError。
SFCErrorPOU	字符串	如果发生超时，则保存发生由 SFCError 记录的超时的功能块的名称。在 SFCQuitError 重置记录的超时之前，将一直存储该名称。 前提条件是声明 SFCError。
SFCQuitError	Bool	只要该 Boolean 变量为 TRUE，TwinCAT 就会暂停执行 SFC 图。在变量 SFCError 中存储的可能的超时将被重置。如果您将变量重置为 FALSE，则活动步骤中的所有先前时间都将被重置。 前提条件是声明 SFCError。
SFCPause	Bool	只要该变量为 TRUE，TwinCAT 就会暂停执行 SFC 图。
SFCTrans	Bool	在发生转换时，变为 TRUE。
SFCCurrentStep	字符串	显示当前活动步骤的名称，与时间监控无关。在平行分支中，始终会存储最右侧分支的步骤名称。
SFCTip, SFCTipMode	Bool	允许 SFC 功能块使用“提示模式”。 如果您使用 SFCTipMode=TRUE 激活提示模式，则您只有将 SFCTip 设置为 TRUE，才能切换到下一个步骤。只要将 SFCTipMode 设置为 FALSE，您也可以通过转换功能进行切换。
SFCErrorAnalyzation		以字符串的形式包含导致 SFCError 的总值 TRUE（一步超时）的所有变量。为此，必须启用 SFCError。 SFCErrorAnalyzation 隐式使用 Tc2_System 库中的功能块 AnaylzeExpression。
SFCError AnalyzationTable		以表格的形式包含导致 SFCError 的总值 TRUE（一步超时）的所有变量。为此，必须启用 SFCError。 SFCErrorAnalyzationTable 隐式使用 Tc2_System 库中的功能块 AnaylzeExpressionTable。

隐式创建 SFC 标志

如果您已激活相应的选项，则 TwinCAT 会自动声明 SFC 标志。您可以在 **Properties**（属性）视图中为单个 SFC POU 设置该选项，也可以在项目属性的 **SFC** 类别中为项目中的所有 SFC POU 设置该选项。

只有在您尚未激活 **Use default SFC settings**（使用默认 SFC 设置）选项的情况下，单个 POU 的 SFC 标志的 SFC 设置才有效。如果您已激活该选项，则在项目属性中定义的设置将适用。

您在 SFC 设置对话框中声明的 SFC 标志仅在 SFC 功能块的在线视图中可见！

明确创建 SFC 标志

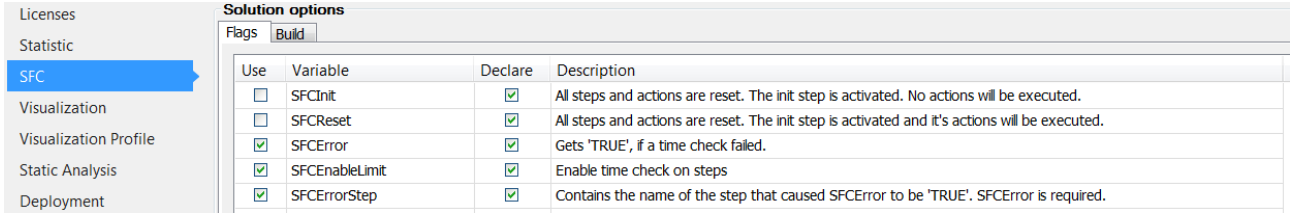
只有在启用来自另一个功能块的写入访问时，才需要手动声明。在这种情况下，请注意以下几点：如果您已在全局变量列表中声明标志，则您必须在 SFC 设置对话框中停用其“Declare”（声明）设置。否则将隐式声明 1 个本地 SFC 标志，然后 TwinCAT 将使用该标志，而不是全局变量！

SFCError 的应用示例

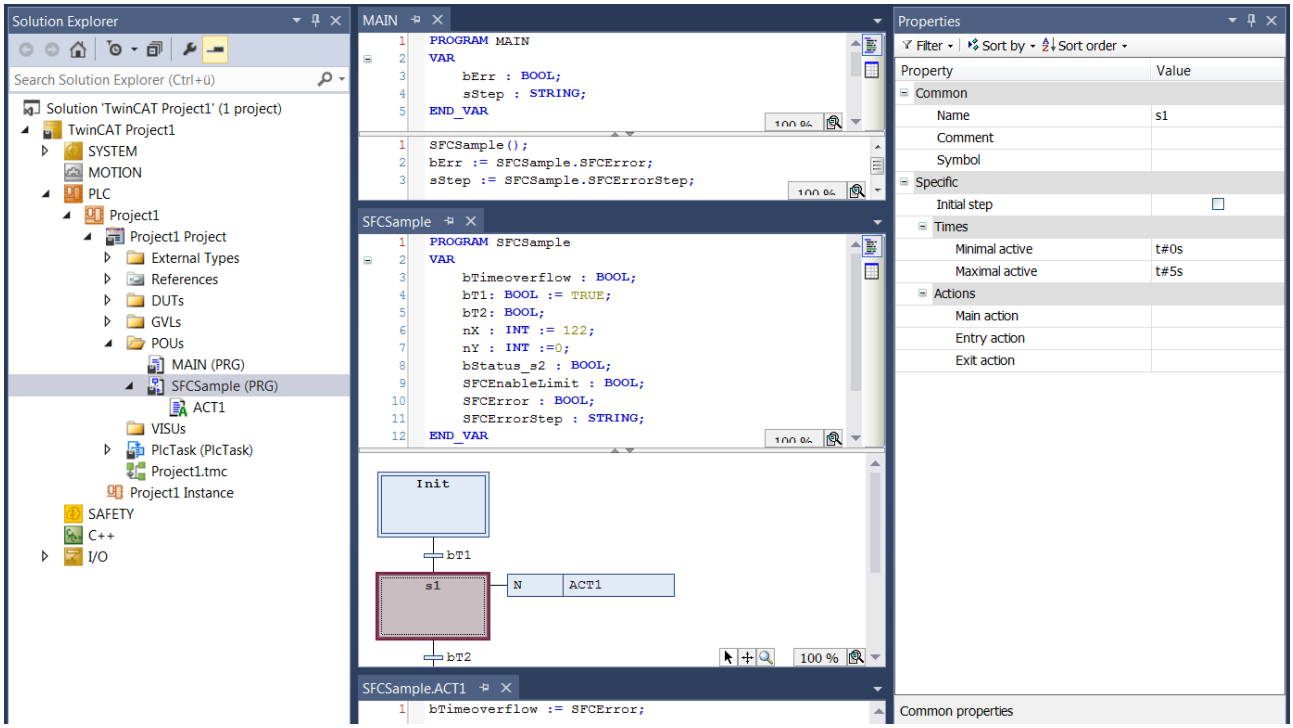
您已创建 1 个名为“SFCSample”的 SFC 功能块，其中包含 1 个步骤“s1”。您已在步骤属性中定义时间限制。请参见图“SFC 功能块 SFCSample 的在线视图”。

如果由于某种原因，步骤 s1 的活动时间超出其时间属性中所允许的时间（超时），则 TwinCAT 会设置 SFC 标志SFCError，PLC 程序可以访问该标志。

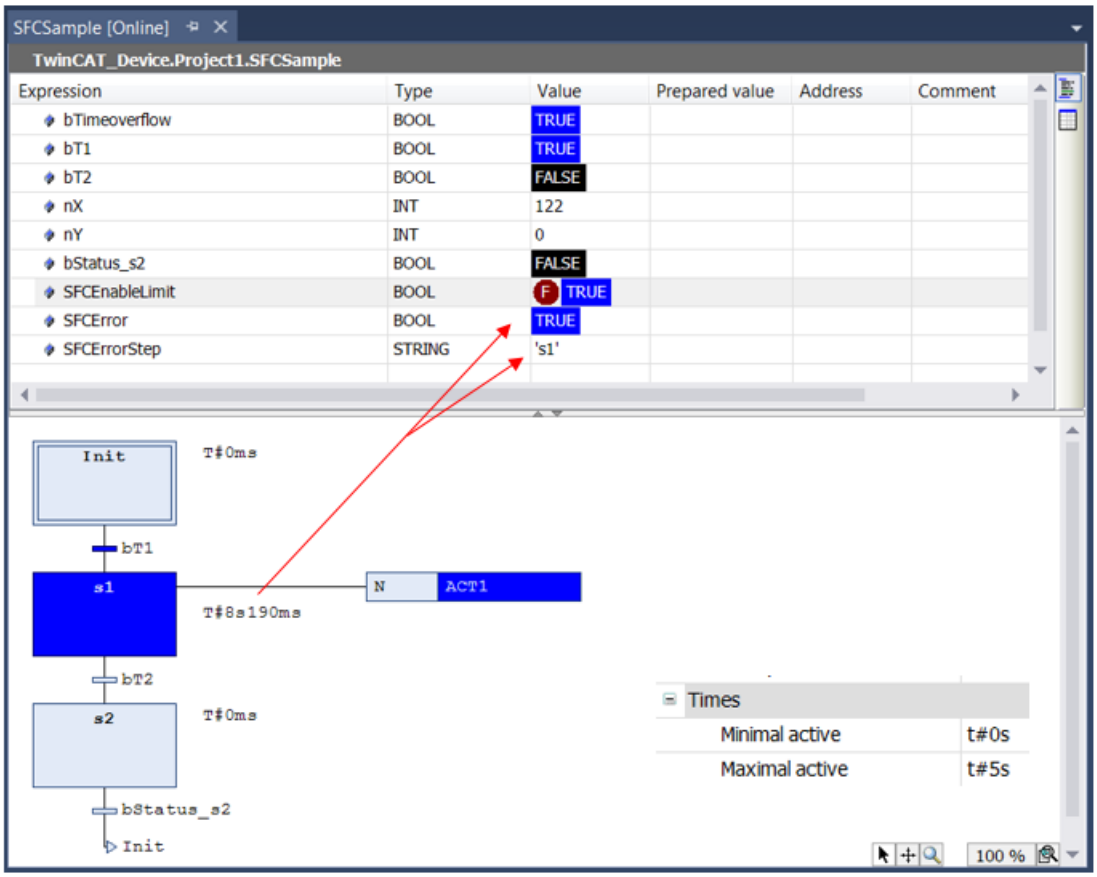
如要允许该访问，则您必须在 SFC 设置中激活并声明 SFC 标志。如果您仅声明它，则会在 SFCSample 的在线视图中的声明部分显示 SFC 标志，但它没有任何功能。



现在，您可以在功能块内（例如，在动作中）或功能块外寻址 SFC 标志。



SFC 功能块 SFCSample 的在线视图：



如果在 SFCSample 内发生超时，则 SFCError 会变为 TRUE。

请注意，您可以使用标志 SFCErrorAnalyzation 和 SFCErrorAnalyzationTable 来确定表达式中会导致 SFCError 的值 TRUE 的组件。

访问标志

访问的语法：

在 POU 中，您可以直接分配标志：<variable name>:=<SFC flag>

示例：

```
checkerror:=SFCerror;
```

来自另一个具有 POU 名称的功能块：<variable name>:=<POU name>.<SFC flag>

示例：

```
checkerror:=SFC_prog.SFCerror;
```

如果您需要从另一个块进行写入访问，则您还必须在 SFC 功能块中将 SFC 标志明确声明为 VAR_INPUT 变量，或者在 GVL 中进行全局声明。

示例：

局部声明：

```
PROGRAM SFC_prog
VAR_INPUT
    SFCinit:BOOL;
END_VAR
```

或全局变量列表中的全局声明：

```
VAR_GLOBAL
    SFCinit:BOOL;
END_VAR

PROGRAM PLC_PRG
VAR
```

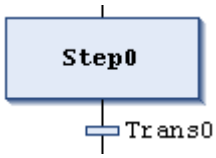
```
    setinit: BOOL;
END_VAR
SFC_prog.SFCinit:=setinit;  //Schreibzugriff auf SFCinit in SFC_prog
```

16.1.4.7 元素

16.1.4.7.1 SFC 元素步骤和转换

步骤符号：  ，转换符号： 

TwinCAT 总是将步骤和转换作为组合插入。插入不带转换的步骤或不带步骤的转换将会导致编译错误。您可以通过双击名称来更改名称。



在“父级”功能块的范围内，步骤名称必须是唯一的。在使用也在 SFC 中进行编程的动作时，请记住这一点。

请注意，通过 **Init step**（Init 步骤）命令或在 SFC 元素属性中设置相应属性，您可以使某个步骤成为初始步骤。

每个步骤均由步骤属性定义，您可以根据在 **Properties**（属性）视图中设置的选项来查看和编辑这些属性。

您必须添加您想要在步骤处于活动状态时执行的动作。它们分为“IEC 动作”和“步骤动作”。有关详细信息，请参见 SFC 元素的**动作**部分。

如果转换包含多个指令，则用户有责任将所需表达式分配给转换变量。

在转换矩形的右上角会用 1 个小三角形表示由转换或属性对象组成的转换。

另请参见：

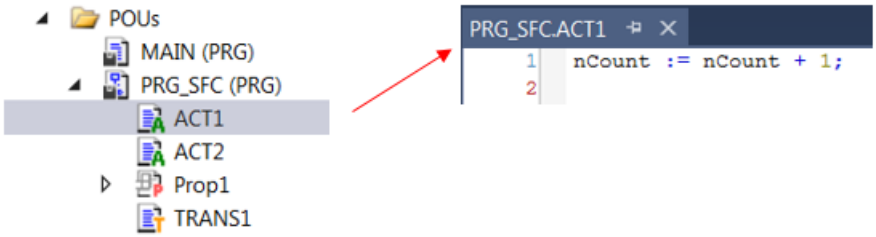
- 用顺序功能图（SFC）编程 [▶ 114]
- 对象转换 [▶ 79]
- SFC 元素属性 [▶ 593]
- SFC 元素动作 [▶ 588]
- 方法调用 - 虚拟函数调用 [▶ 184]
- TC3 用户界面文档：命令：插入步骤转换 [▶ 929]
- TC3 用户界面文档：命令：初始化步骤 [▶ 929]

16.1.4.7.2 SFC 元素动作

符号： 

1 个动作包含 1 个或多个使用有效编程语言编写的指令。您可以为 1 个步骤分配 1 个动作。

您在 SFC 步骤中使用的动作必须在项目中被创建为功能块。



例外：当您将 IEC 动作作为动作关联添加到步骤中时，您也可以指定 1 个 Boolean 变量，而不是动作对象。每次执行动作时，该变量的值都会在 FALSE 和 TRUE 之间切换。

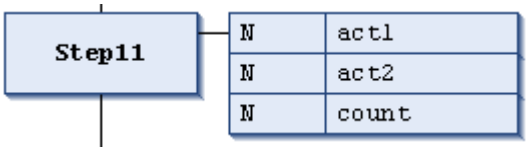
在“父级”功能块的范围内，步骤名称必须明确定义。在 SFC 中编写的动作不得包含与分配该动作的步骤同名的步骤。

它们分为“IEC 动作”和“步骤动作”。

1. IEC 动作

IEC 动作是指符合 IEC 61131-3 标准的动作。它们根据其限定符执行。IEC 动作至少要执行 2 次：第 1 次是在步骤进入活动状态时，第 2 次是在步骤停用时。如果您为 1 个步骤分配多个动作，则会从上到下依次处理动作列表。

每个动作框的第 1 列包含限定符，第 2 列包含动作名称。您可以直接编辑这 2 项内容。



与步骤动作不同，您可以针对 IEC 动作使用不同的限定符。与步骤动作相比，另一个不同之处在于每个 IEC 动作都有 1 个控制标志。这意味着 TwinCAT 仅会执行 1 次动作，即使该动作同时被另一个步骤调用也是如此。对于步骤动作，这一点无法保证。

您可以使用 SFC 菜单中的 **Insert action association**（插入动作关联）命令，为步骤分配 IEC 动作。

相关 Boolean 变量

每次调用 SFC 功能块时，都会设置或重置 1 个相关 Boolean 变量。这意味着，无论相关步骤是否处于活动状态，每次都会为它重新分配值 TRUE 或 FALSE。

如果在不同的 SFC 功能块中将同一个全局 Boolean 变量关联为 IEC 动作，这可能会导致不良的覆盖效果。

另请参见：

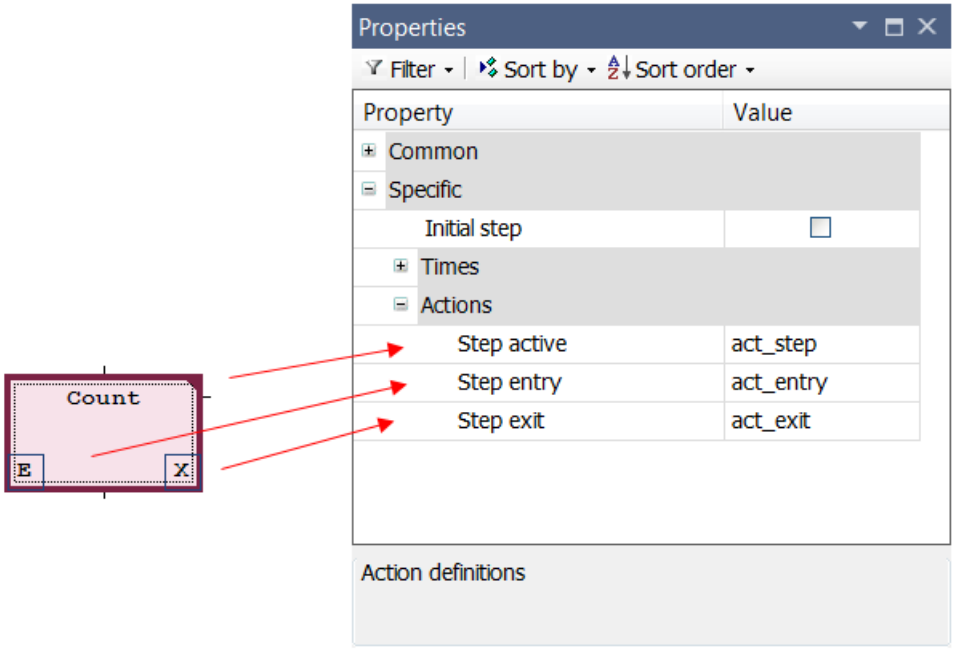
- TC3 用户界面文档：命令：插入动作关联 [▶ 932]
- SFC 中的动作限定符 [▶ 583]

2. 步骤动作

您可以使用这些动作来扩展 IEC 标准。

- **输入动作：**
  在激活步骤后和执行主要动作前，TwinCAT 会执行 1 次该动作。
  通过元素属性 **Entry action**（进入动作）（**Step entry**）（进入步骤），您可以从步骤中引用新动作或在 SFC 对象下已创建的动作。使用 **Add entry action**（添加进入动作），您还可以为步骤添加新动作。步骤框左下角的“E”表示进入动作。
- **主要动作：**
  在激活步骤并执行任何进入动作后，TwinCAT 会执行该动作。不过，与 IEC 动作（见上文）不同的是，当再次停用步骤时，不会将它执行第 2 次。此外，您不可以在此处使用限定符。
  如要添加现有动作，可使用元素属性 **Main action**（主要动作）（**Step active**）（活动步骤）。步骤框右上角的填充三角形表示主要动作。

- **退出动作：**
  如果已停用步骤，TwinCAT 会执行 1 次该动作。但请注意，这不再在同一个周期内执行，而是在下一个周期开始时执行！
  通过元素属性 **Exit action**（退出动作）（**Step exit**）（退出步骤），您可以从步骤中引用新动作或在 SFC 对象下已创建的动作。使用 **Add exit action**（添加退出动作），您还可以为步骤添加新动作。步骤框右下角的“X”表示退出动作。



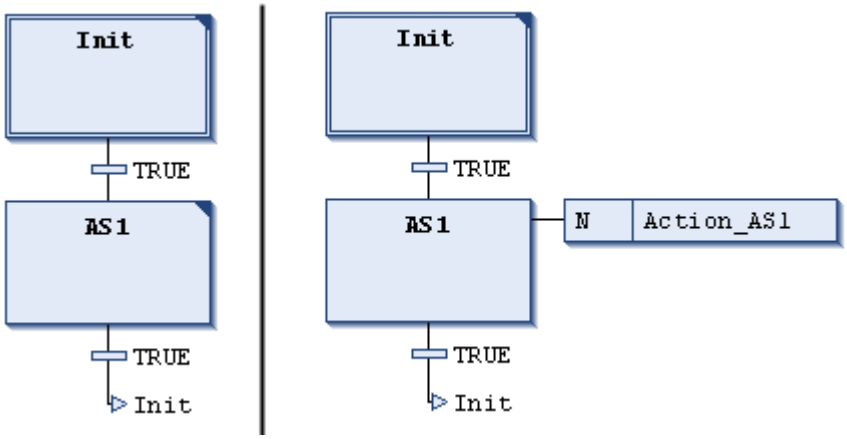
另请参见：

- SFC 元素属性 [▶ 593]

IEC 动作与步骤动作之间的区别

版本

步骤动作与带有限定符“N”的 IEC 动作之间的主要区别在于，IEC 动作（至少）要执行 2 次：第 1 次是在步骤进入活动状态时，第 2 次是在步骤停用时。请参见下面的示例：



您为步骤 SFC1 添加了 Action_SFC1 动作，可作为步骤动作（左），也可作为带有限定符 N 的 IEC 动作。由于在这 2 种情况下都会触发 2 次转换，因此需要 2 个 PLC 周期才能再次到达初始化步骤。假设 Action_SFC1 使初始化为 0 的计数器变量 nCounter 递增。在再次激活 Init 步骤后，nCounter 在左侧示例中的值为 1。不过，在右侧示例中，该值为 2，因为由于 SFC1 的停用，需要第 2 次执行 IEC 动作。

复制

另一个区别是，您可以“嵌入”步骤动作。在这种情况下，只能从相关步骤中调用它们。当您复制该步骤时，TwinCAT 会自动生成新的动作对象并复制实现代码。

当您在步骤中插入第 1 个动作或使用步骤属性 **Duplicate or copy**（复制或拷贝）时，您可以定义是否“嵌入”步骤动作。一般来说，您也可以在 TwinCAT 选项的 **SFC editor**（SFC 编辑器）类别中预先设置该行为。

Boolean 变量

此外，您可以为 IEC 动作指定 1 个 Boolean 变量，而不是动作对象。步骤动作则无法做到这一点。

另请参见：

- SFC 元素属性 [▶ 593]

16.1.4.7.3 SFC 元素分支

符号： 

分支可用于在 SFC 中对并行或替代序列进行编程。

对于替代分支，TwinCAT 总是只执行其中 1 个分支，具体取决于先前的转换条件。平行分支可同时执行。

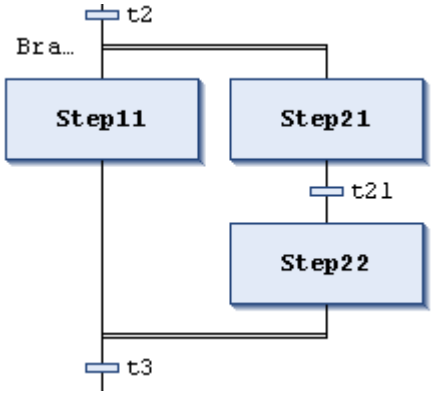
另请参见：

- 用顺序功能图（SFC）编程 [▶ 114]
- SFC 中的处理顺序 [▶ 582]
- TC3 用户界面文档：命令：在右侧插入分支 [▶ 931]

平行分支

在平行分支中，分支必须以步骤开始和结束。平行分支可能包含更多分支。

分支前后的水平线为双线。



在在线模式下处理：如果前一个转换（在所示示例中为 t2）为 TRUE，则所有平行分支中的第 1 个步骤（Step11 和 Step21）都处于活动状态。TwinCAT 会并行处理各个分支，然后才会评估后续转换（t3）。

系统会自动将分支标签 <n> 添加到构成分支起点的水平线上。您可以将此标记指定为跳转目标。

请注意，您可以使用 **Alternative** （替代）命令将平行分支转换为替代分支。

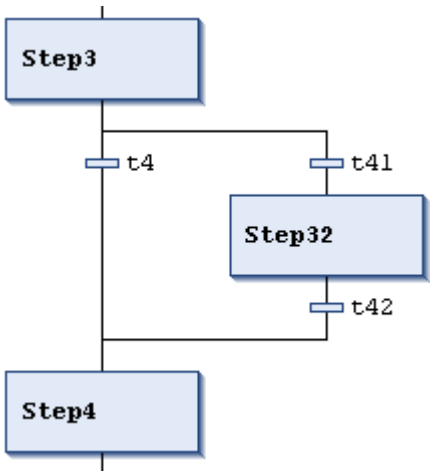
另请参见：

- TC3 用户界面文档：命令：替代 [▶ 930]

替代分支

分支前后的水平线为单线。

在替代分支中，分支必须以转换开始和结束。这些分支可能包含更多分支。



如果分支前面的步骤处于活动状态，则 TwinCAT 将从左到右评估每个替代分支的第 1 个转换。在第 1 个返回 TRUE 的转换中会“打开”相应的分支，即转换后的步骤将被激活。

请注意，您可以使用 Parallel （平行）命令将替代分支转换为平行分支。

另请参见：

- TC3 用户界面文档：命令：平行 [▶ 930]

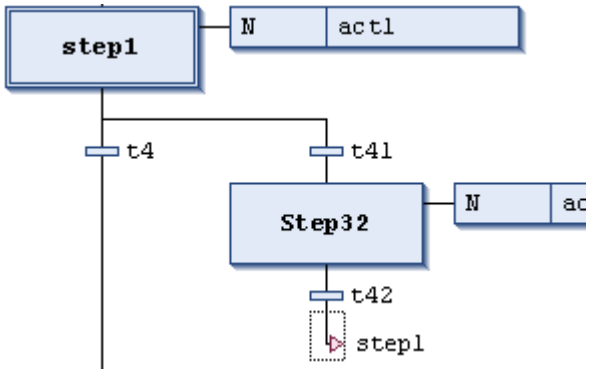
16.1.4.7.4 SFC 元素跳转

符号：

跳转可定义当跳转之前的转换变为 TRUE 时，下一步要执行的步骤。跳转可能十分必要，因为执行线可能不会交叉，也无法向上引导。

除了图表末尾处的强制跳转之外，您仅可在分支终点处输入跳转。

跳转目标由添加的文本字符串定义，您可以直接对其进行编辑。跳转目标可以是步骤名称或平行分支的标记。



另请参见：

- 用顺序功能图（SFC）编程 [▶ 114]
- TC3 用户界面文档：命令：插入跳转 [▶ 933]

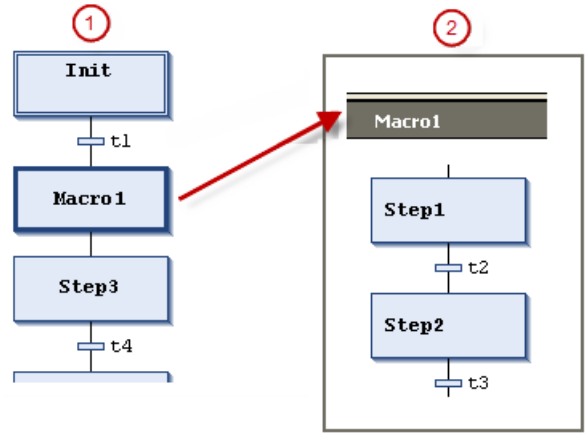
16.1.4.7.5 SFC 元素宏

符号：

宏包含 SFC 图的一部分，这部分在编辑器的主视图中没有详细显示。

宏的使用不会影响处理流程。例如，宏仅用于隐藏图表的特定部分，以提高清晰度。

双击宏框或使用 SFC 菜单中的 Show macro（显示宏）命令，可打开宏编辑器。您可以在此处以与 SFC 编辑器的主视图相同的方式进行编程。如要关闭宏编辑器，可使用 SFC 菜单中的 Exit macro（退出宏）命令。

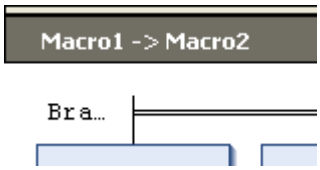


① SFC 编辑器中的主视图

② 宏编辑器中的 Macro1 视图

宏可能包含更多宏。宏编辑器的标题栏始终会显示当前在图表中打开的宏的路径。

示例：



另请参见：

- 用顺序功能图（SFC）编程 [▶ 114]
- TC3 用户界面文档：命令：显示宏 [▶ 934]
- TC3 用户界面文档：命令：退出宏 [▶ 934]

16.1.4.7.6 SFC 元素属性

您可以在 Properties（属性）视图中编辑 SFC 元素的属性。使用 View > Properties Window（视图 > 属性窗口）命令打开该视图。显示哪些属性取决于当前所选元素。

在 SFC 图中元素旁边显示的属性取决于 View（视图）选项卡中的 TwinCAT > PLC Environment > SFC editor（TwinCAT > PLC 环境 > SFC 编辑器）类别中的 TwinCAT 选项设置。

一般信息

属性	值描述
名称	元素名称，默认为 <element><consecutive number>，例如，步骤名称“Step0”、“Step1”，分支名称“Branch0”等。
注释	元素注释（文本），例如，计数器重置。您可以使用 [Ctrl] + [Enter] 插入换行符。
符号	对于每个 SFC 元素，TwinCAT 都会创建 1 个与该元素同名的隐式变量。

特定信息

属性	值描述
Init 步骤	☑：该选项仅适用于当前被定义为初始步骤的步骤。默认情况下，这是 SFC 图中的第 1 个步骤。 如果您已为另一个步骤激活该属性，则您必须为先前具有该属性的步骤停用该属性，以避免编译错误。
复制或拷贝	该选项适用于包含步骤动作（进入动作、主要动作或退出动作）的步骤，以及与转换对象相关联的转换。 ☑：在您复制步骤/转换时，系统会为每个被调用的动作/转换创建 1 个新对象。它包含复制对象的实现代码的副本。 ☐：在您复制步骤或转换时，系统会为相关的动作/转换保留与相关对象的链接。系统不会创建新对象。步骤或转换的源端和副本会调用相同的动作/转换。 在 PLC 项目树中会显示嵌入对象，其名称中带有下划线。 您可以使用 **Change duplication > Set**（更改复制 > 设置）和 **Remove**（删除）命令“嵌入”在 SFC 功能块中调用的所有步骤动作或转换，或者取消嵌入。
时间 • 最短活动时间 • 最长活动时间	步骤处于活动状态的最短时间，即使后续转换为 TRUE。 步骤可能处于活动状态的最长时间。如果超过该时间，TwinCAT 会将隐式变量 SFCError 设置为 TRUE。 根据 IEC 语法（例如，t#8 s）或 TIME 变量确定的时间范围；默认值：t#0s。
动作 • 进入动作 • 步骤动作 • 退出动作	• 进入动作：在激活步骤后，TwinCAT 会执行该动作。 • 步骤动作：如果步骤处于活动状态且已执行任何进入动作，则 TwinCAT 会执行该动作。 • 退出动作：如果停用步骤，TwinCAT 会在下一个周期中执行该动作。 请注意处理顺序。

您可以使用相应的隐式 SFC 变量和标志来获取有关步骤或动作状态的信息，或者有关超时的信息。

另请参见：

- 隐式变量 [▶ 583]
- TC3 用户界面文档：对话框选项 - SFC 编辑器 [▶ 907]

16.1.5 功能块图/梯形图/指令表（FBD/LD/IL）

16.1.5.1 FBD/LD/IL 编辑器

FBD/LD/IL 编辑器是编程语言 FBD、LD 和 IL 的组合编辑器。

如果需要，可通过 TwinCAT 选项启用 IL。（**Tools > Options > TwinCAT > PLC Environment > FBD, LD and IL > IL**）（工具 > 选项 > TwinCAT > PLC 环境 > FBD、LD 和 IL > IL）

有 1 组通用的命令和元素，TwinCAT 可以在内部自动转换这 3 种编程语言。

实现部分的代码在所有 3 种语言中都使用网络进行构建。

FBD/LD/IL 菜单包含在编辑器中运行的命令。

在离线和在线模式下，您随时可以使用命令在 3 种编辑器视图之间切换。

FBD/LD/IL 编辑器的行为由 **Tools > Options**（工具 > 选项）菜单中的 **TwinCAT > PLC Environment > FBD, LD and IL**（TwinCAT > PLC 环境 > FBD、LD 和 IL）类别中的设置决定。

TwinCAT 无法转换一些特殊元素，因此只能以相应的语言显示它们。同样，在 IL 和 FBD 之间无法明确转换一些结构，因此，在将这些结构转换回 FBD 时会对它们进行“标准化”处理，即取消。这会涉及到：表达式的否定以及功能块输入端和输出端的显式/隐式赋值。

另请参见：

- 所有图形编辑器的一般功能 [▶ 570]
- TC3 用户界面文档：FBD/LD/IL [▶ 951]
- TC3 用户界面文档：对话框选项 - FBD、LD 和 IL [▶ 900]

FBD 和 LD 编辑器

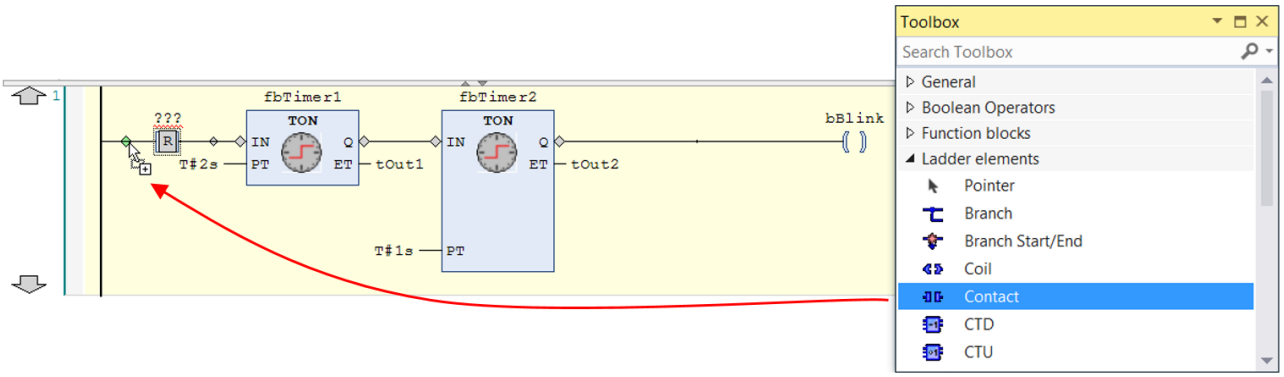
插入和排列项目：

您可以用鼠标拖动元素，将它们从 **Toolbox**（工具箱）视图移动到编辑器的实现部分。或者，您也可以使用上下文菜单或菜单 **FBD/LD/IL** 的命令。

在 TwinCAT 选项的 **TwinCAT > PLC Environment > FBD, LD and IL**（TwinCAT > PLC 环境 > FBD、LD 和 IL）类别中，您可以定义与显示和接口有关的设置。

当您用鼠标在编辑器中的网络上拖动元素时，所有可能的插入位置都会以灰色菱形、三角形或箭头形位置标记显示出来。当您将鼠标指针置于其中 1 个标记上方时，该标记会变成绿色，当您松开鼠标按钮后，TwinCAT 会在该位置插入元素。

示例：



当您从 **Toolbox**（工具箱）或网络中拖动 1 个功能块或运算符到网络左边缘的 2 个箭头之一时，将会发生以下情况：TwinCAT 会自动创建 1 个新网络，并将该元素插入其中。

如要替换 1 个元素，可使用鼠标拖动 1 个合适的其他元素到该元素的位置。在编辑器中，TwinCAT 通过文本字段识别可被新元素替换的元素，例如，“Replace”（替换）或“Append input”（附加输入）。

如要剪切、复制、粘贴和删除元素，您可以使用 **Edit**（编辑）菜单的常用命令。在按下 **[Ctrl]** 键时，您也可以通过拖放进行复制。

仅可将具有 EN/ENO 功能的运算符插入到 FBD 和 LD 编辑器中。

选择元素：

通过点击编辑器中的功能块或连接线并使焦点位于其上，您可以选择该功能块或连接线。在按下 **[Ctrl]** 按钮时，您可以进行多项选择。所选元素以红色阴影显示。

工具提示：

当光标指向特定元素（例如，变量或输入）时，就会出现工具提示，显示该元素的相关信息。

对于带有红色波浪线的元素，工具提示会显示相应的预编译错误消息。

在编辑器中导航：

使用下述按钮和命令，您可以将焦点移动到编辑器内不同的光标位置。跨网络也可以实现不同位置之间的切换。	
[←] [→]	沿着信号流的方向（即从左到右，反之亦然），切换到相邻的光标位置。
[↑][↓]	如果相邻位置属于同一个逻辑组，则切换到当前位置上方或下方的下一个光标位置。例如，1 个逻辑组由功能块的所有连接组成。 如果不存在这样的逻辑组：切换到下一个上部或下部相邻元素的第 1 个光标位置。在元素并行连接的情况下，将沿着第 1 个分支进行导航。
[Ctrl] + [Pos 1]	切换到第 1 个网络；已选中。
[Ctrl] + [End]	切换到最后一个网络；已选中。
[Image up]	向上滚动 1 页。该页的顶部网络已选中。
[Image down]	向下滚动 1 页。该页的底部网络已选中。
命令“Go To...”（转至……）	切换到特定网络。

打开功能块：

如果在编辑器中插入功能块，您可以通过双击或使用上下文菜单中的 Go To（转至）命令来双击其实现。

另请参见：

- 功能块图（FBD） [▶ 99]
- 编程功能块图（FBD） [▶ 101]
- 梯形图（LD） [▶ 100]
- 编程梯形图（LD） [▶ 102]
- 元素 [▶ 602]
- 在线模式下的 FBD/LD/IL 编辑器 [▶ 598]
- TC3 用户界面文档：命令：转至 [▶ 963]

IL 编辑器

插入和排列项目：

您可以使用菜单 FBD/LD/IL 或上下文菜单的命令插入元素。您也可以通过拖放将新网络从 Toolbox（工具箱）添加到编辑器的实现部分。

如要剪切、复制、粘贴和删除元素，您可以使用 Edit（编辑）菜单的常用命令。在按下 [Ctrl] 键时，您也可以通过拖放进行复制。

请注意，仅可将具有 EN/ENO 功能的运算符插入到 FBD 和 LD 编辑器中。

每个程序行都应被输入到 1 个表格行中。

IL 编辑器中的网络结构：

第 1 行：网络标题 要求：在 TwinCAT 选项中已激活该选项		
第 2 行：网络注释 要求：在 TwinCAT 选项中已激活该选项		
自第 3 行起：		
列	**内容**	**描述**
1	运算符	包含 IL 运算符（LD、ST、CAL、AND、OR 等）或函数名称。如果您调用 1 个功能块，您还必须在此处指定相应的参数；在这种情况下，您必须在前面的字段中输入 := 或 =>。
2	操作数	精确包含 1 个操作数或 1 个标签的名称。 如果有多个操作数，您必须在多个行中输入它们，并在各个操作数的后面直接插入逗号。（请参见下文示例）
3	地址	包含为操作数声明所定义的操作数地址。 不可编辑 您可以使用 Show symbol address（显示符号地址）选项来启用/禁用显示。为此，可选择命令 **Tools > Options**（工具 > 选项）并在类别 **TwinCAT > PLC Environment > FBD, LD and IL**（TwinCAT > PLC 环境 > FBD、LD 和 IL）中选择 **General**（常规）选项卡。
4	符号注释	包含在声明中可能为操作数输入的注释。 不可编辑 通过选择命令 **Tools > Options**（工具 > 选项）并在类别 **TwinCAT > PLC Environment > FBD, LD and IL**（TwinCAT > PLC 环境 > FBD、LD 和 IL）中选择 **General**（常规）选项卡，您可以使用选项 **Show symbol comment**（显示符号注释）启用/禁用显示。
5	操作数注释	与当前程序行有关的注释。 通过选择命令 **Tools > Options**（工具 > 选项）并在类别 **TwinCAT > PLC Environment > FBD, LD and IL**（TwinCAT > PLC 环境 > FBD、LD 和 IL）中选择 **General**（常规）选项卡，您可以使用选项 **Show operand comment**（显示操作数注释）启用/禁用显示。

示例：

```
CAL          fbTonInst1(
        IN:= bVar,
        PT:= tTime1,
        ET=> tOut1)
LD           fbTonInst1.Q          gets TRUE, delay time (PT) aft…
ST           fbTonInst2.IN         starts timer with rising edge,…
CAL          fbTonInst2(
        PT:= tTime2,
         Q=> bReady,               %Q*                              for fbTonInst2
        ET=> tOut2)
```

在编辑器中导航：

按键/命令	光标移动
[↑][↓]	跳转到上面/下面的字段。
[Tab]	向右跳转到该行中的下一个字段。
[Shift] + [Tab]	向左跳转到该行中的前一个字段
[Space]	打开所选字段的编辑边框。或者，您也可以用鼠标点击字段。如果适用，Input Assistant（输入助手）对话框的按钮可供使用。
[Ctrl] + [Enter]	在当前行的下面插入 1 个新行。
[Del]	删除当前行。
[Ctrl] + [Pos 1]	将焦点设置到文档开头并标记第 1 个网络。
[Ctrl] + [End]	将焦点移至文档末尾并标记最后一个网络。
[Image down]	向上滚动页面并标记最上面的矩形。
[Image up]	向下滚动页面并标记最上面的矩形。

另请参见：

- 指令表（IL） [▶ 100]
- 编程指令表（IL） [▶ 103]
- IL 中的修饰符和运算符 [▶ 599]
- 在线模式下的 FBD/LD/IL 编辑器 [▶ 598]

16.1.5.2 在线模式下的 FBD/LD/IL 编辑器

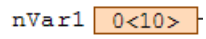
在在线模式下，编辑器会在变量名后显示每个变量的当前值。您可以编写/强制和设置断点。

如果当前为强制变量，则在强制值前直接用 F 表示。如果为写入或强制而准备 1 个值，则该值将紧接着在当前值后面的括号 <value> 中显示。

示例：

强制变量：

准备值



在梯形图（LD）的在线视图中，连接线采用颜色编码：值为 TRUE 的连接线显示为蓝色粗线，值为 FALSE 的连接线显示为黑色粗线。未知值或模拟值的连接线正常显示（黑色细线）。

请注意，连接线的值根据监控变量计算得出。这不是真正的过程控制。

断点

可能的断点位置为变量值可能发生变化的位置（指令）、程序分支的位置或调用另一个功能块的位置。

可能的断点位置

- 在整个网络上：将断点设置在网络中第 1 个可用的位置。
- 在功能块框上，如果功能块包含 1 个赋值的话。不适用于运算符功能块，例如，ADD、DIV。
- 在赋值处。
- 在功能块的末尾，在返回至调用功能块的位置。在在线模式下，此时会自动出现 1 个空网络，该网络以“RET”标记，而不是网络编号。

目前，您无法直接在网络中的第 1 个功能块上设置断点。不过，如果您在整个网络上设置了断点，则该断点标记会被自动转移到在线模式下的第 1 个功能块。

方法中的断点：TwinCAT 会在所有可以调用的方法中自动设置断点。这意味着，如果调用 1 个由接口管理的方法，则会在实现该接口的功能块中出现的所有方法以及使用该方法的所有派生功能块中设置断点。如果通过指向功能块的指针调用 1 个方法，则 TwinCAT 会在该功能块的方法以及使用该方法的所有派生功能块中设置断点。

另请参见：

- 强制和写入变量值 [▶ 198]
- 使用断点 [▶ 195]

16.1.5.3 IL 中的修饰符和运算符

修饰符：

修饰符	与运算符组合	描述
C	JMP、CAL、RET	只有在前面表达式的结果为 TRUE 时，才会执行指令。
N	JMPC、CALC、RETC	只有在前面表达式的结果为 FALSE 时，才会执行指令。
N	否则	操作数（非累加器）的否定。

运算符和可能的修饰符：

运算符	N	含义	示例
LD	N	将操作数的（否定）值加载到累加器上。	LD ivar
ST	N	将累加器的（否定）内容存储到操作数中。	ST iErg
S		如果累加器的内容为 TRUE，则将操作数（BOOL 类型）设置为 TRUE。	S bVar1
R		如果累加器的内容为 TRUE，则将操作数（BOOL 类型）设置为 FALSE。	R bVar1
AND	N,(	累加器值和（否定）操作数的按位与	AND bVar2
OR	N,(	累加器值和（否定）操作数的按位或	OR xVar
XOR	N,(	累加器值和（否定）操作数的按位异或	XOR N,(bVar1,bVar2)
NOT		累加器值的按位取反	
ADD	(	累加器值和操作数的相加。结果将被写入累加器。	ADD ivar1
SUB	(	从累加器值中减去操作数。结果将被写入累加器。	SUB iVar2
MUL	(	累加器值与操作数的相乘。结果将被写入累加器。	MUL ivar2
DIV	(	累加器值除以操作数。结果将被写入累加器。	DIV 44
GT	(	检查累加器值是否大于操作数的值。结果（BOOL）将被写入累加器。>	GT 23
GE	(	检查累加器值是否大于或等于操作数的值。结果（BOOL）将被写入累加器。	GE iVar2
EQ	(	检查累加器值是否等于操作数的值。结果（BOOL）将被写入累加器。	EQ iVar2
NE	(	检查累加器值是否等于操作数的值。结果（BOOL）将被写入累加器。	NE iVar1
LE	(	检查累加器值是否大于或等于操作数的值。结果（BOOL）将被写入累加器。	LE 5
LT	(	检查累加器值是否小于操作数的值。结果（BOOL）将被写入累加器。	LT cVar1
JMP	CN	无条件（条件）跳转到指定标签	JMPN next
CAL	CN	(条件）调用程序或功能块（如果累加器值为 TRUE）	CAL prog1
RET		退出块，返回至调用功能块	RET
RET	C	如果累加器值为 TRUE：退出功能块，返回至调用功能块	RETC
RET	CN	如果累加器值为 FALSE：退出功能块，返回至调用功能块	RETCN
)		评估延迟操作	

示例：

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

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

2   m1:

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

应用	描述	示例

1 个运算符的多个操作数	可能性
	• 您在连续行中输入操作数，在第 2 列中用逗号隔开。
	• 您在连续行中重复运算符。
		版本 1：
		`LD 2`
		`ADD 3,`
		`4,`
		`6,`
		`ST nVar`
		版本 2：
		`LD 2`
		`ADD 3`
		`ADD 4`
		`ADD 6`
		`ST nVar`

复杂操作数	对于复杂操作数，在第 1 列中指定左括号（。在下面各行输入操作数之后，在单独 1 行的第 1 列中输入右括号。	字符串每个周期旋转 1 个字符：
		`LD sRotate`
		`RIGHT( sRotate`
		`LEN`
		`SUB 1`
		`)`
		`CONCAT( sRotate`
		`LEFT 1`
		`)`
		`ST sRotate`

功能块调用，子程序调用	第 1 列：运算符 CAL 或 CALC
	第 2 列：功能块实例或程序的名称以及左括号（。如果后面没有参数，则在此处输入右括号）。
	后续行：
	第 1 列：参数名称后面跟：= 表示输入参数，或 => 表示输出参数
	第 2 列：参数值后面加逗号，如果后面跟其他参数的话。在最后一个参数之后，输入右括号）。
	此处不能使用复杂表达式，这是 IEC 标准的限制。您必须在调用前将此类结构分配给功能块或程序。
		`CAL ProgToCall(`
		`nCounter:= 1,`
		`nDecrement:= 1000,`
		`nError=> nResult)`
		`LD ProgToCall.bError`
		`ST bErr`

函数调用	第 1 行：第 1 列：LD
	第 2 列：输入变量
	第 2 行：第 1 列：函数名称 第 2 列：其他输入参数，用逗号隔开。
	TwinCAT 将返回值写入累加器中。
	第 3 行：第 1 列：ST 第 2 列：写入返回值的变量
		`LD nVar7`
		`F_GeomAverage 25`
		`ST nAve`

动作调用	像功能块或程序调用一样。 动作名称会被附加到 FB 实例或程序名称中。	`CAL ProgToCall.ResetAction`
跳转	第 1 列：运算符 JMP 或 JMPC。 第 2 列：目标网络的标签名称。 对于无条件跳转，前面的指令序列必须以下列命令之一结束：ST、STN、S、R、CAL、RET、JMP 对于条件跳转，执行情况取决于加载值。	`LD bVar1` `JMPC Label1`

另请参见：

- 指令表（IL） [▶ 100]

- 编程指令表（IL） [▶ 103]

16.1.5.4 元素

16.1.5.4.1 FBD/LD/IL 元素网络

符号： 

网络是 FBD 或 LD 程序的基本单元。在 FBD/LD/IL 编辑器中，网络以列表的形式排列在彼此下面。每个网络的左侧都有 1 个序列号，可能包含以下内容：逻辑和算术表达式；程序、函数和功能块调用；跳转或返回指令。

IL 程序至少包括 1 个网络。该网络可能包含程序的所有 IL 指令。

您可以为每个网络分配标题、注释或标记。在 TwinCAT 选项的 **TwinCAT > PLC Environment > FBD, LD and IL**（TwinCAT > PLC 环境 > FBD、LD 和 IL）类别中，您可以定义是否在编辑器中显示网络标题、网络注释以及各个网络之间的分隔线。

如要输入网络标题，可点击网络的第 1 行。如要输入注释，可点击网络的第 2 行。

另请参见：

- TC3 用户界面文档：对话框选项 - FBD、LD 和 IL [▶ 900]
- TC3 用户界面文档：命令：插入网络 [▶ 952]

16.1.5.4.2 FBD/LD/IL 元素功能块

符号： 

块及其调用可代表其他函数，例如，IEC 功能块、IEC 函数、库块、运算符。

1 个功能块可以有任何输入和输出。

如果功能块与图像文件相关联，则会在功能块内显示功能块符号。前提条件是，在 TwinCAT 选项的 **TwinCAT > PLC Environment > FBD, LD and IL**（TwinCAT > PLC 环境 > FBD、LD 和 IL）类别中启用 **Show function block icon**（显示功能块图标）选项。

如果您已更改功能块接口，则您可以通过 **FBD/LD/IL** 菜单中的 **Update parameters**（更新参数）命令来更新功能块参数，而无需重新插入功能块。

另请参见：

- TC3 用户界面文档：命令：更新参数 [▶ 962]
- TC3 用户界面文档：对话框选项 - FBD、LD 和 IL [▶ 900]
- TC3 用户界面文档：命令：插入框 [▶ 953]

16.1.5.4.3 FBD/LD/IL 元素赋值

符号：

FBD 编辑器会将新赋值显示为 1 行，后面跟 3 个问号。LD 编辑器会将新赋值显示为 1 个线圈，上面有 3 个问号。

在插入之后，您可以用变量名替换 **???** 占位符，从而将来自左侧的信号分配给该变量。**input assistant**（输入助手）可用于此目的。

在 IL 中，使用运算符 LD 和 ST 对赋值进行编程。

另请参见：

- FBD/LD/IL 编辑器 [▶ 594]
- TC3 用户界面文档：命令：插入赋值 [▶ 953]

16.1.5.4.4 带 EN/ENO 的 FBD/LD/IL 元素功能块

符号： 

该元素仅限在 FBD 和 LD 编辑器中可用。

该功能块通常与 FBD/LD/IL **功能块**相对应。此外，该功能块还有 1 个 EN 输入端和 1 个 ENO 输出端。EN 和 ENO 的数据类型均为 BOOL。

EN 输入端和 ENO 输出端的功能：如果在调用功能块时 EN 输入端的值为 FALSE，则不执行在功能块中定义的操作。否则，即如果 EN 为 TRUE，则执行这些操作。ENO 输出端的值与 EN 输入端的值相同。

另请参见：

- FBD/LD/IL 元素功能块 [▶ 602]
- TC3 用户界面文档：命令：插入带 EN/ENO 的框 [▶ 953]

16.1.5.4.5 FBD/LD/IL 元素输入端

符号： 

输入端的最大数量取决于功能块类型。

新添加的输入端的初始赋值为 ???。您可以用变量或常量替换 ???。

另请参见：

- TC3 用户界面文档：命令：插入输入 [▶ 955]

16.1.5.4.6 FBD/LD/IL 元素标签

标签是 FBD 和 LD 中网络的可选标识符，您可以用它来指定跳转目标。

当您在网络中插入标签时，您会将其添加为可编辑的 Label:（标签：）字段。

另请参见：

- TC3 用户界面文档：命令：插入标签 [▶ 954]

16.1.5.4.7 FBD/LD/IL 元素跳转

符号： 

在 FBD 或 LD 中，根据当前光标位置，您可以直接在输入端前、输出端后或网络末端插入跳转。

您可以直接在跳转元素后输入 1 个标签作为跳转目标。

在 IL 中，JMP 指令可用于对跳转进行编程。

另请参见：

- FBD/LD/IL 元素标签 [▶ 603]
- TC3 用户界面文档：命令：插入跳转 [▶ 954]

16.1.5.4.8 FBD/LD/IL 元素返回

当 RETURN 元素的输入端变为 TRUE 时，该元素可用于立即停止执行功能块。

在 FBD 或 LD 网络中，您可以将返回指令置于在与前面元素平行或在其之后的位置。

在 IL 中，RET 指令可用于此目的。

另请参见：

- IL 中的修饰符和运算符 [▶ 599]
- TC3 用户界面文档：命令：插入返回 [▶ 955]

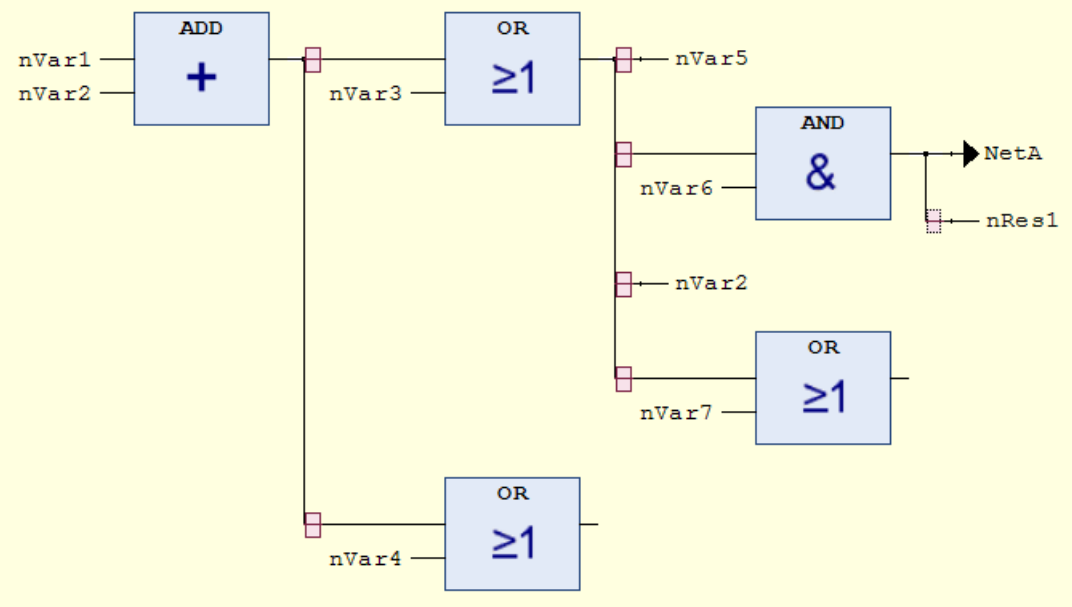
16.1.5.4.9 FBD/LD/IL 元素线路分支

符号：

该元素在 LD 和 FBD 编辑器中可用，表示开放线路分支。线路分支从当前光标位置将处理线分成 2 个子网络，从上到下依次执行它们。您可以对每个子网络进行进一步分支处理，在 1 个网络内创建多个分支。

每个子网的分支点都有 1 个标记符号（矩形），您可以选择它来执行其他命令。

复制、剪切和粘贴命令不适用于子网络。



如要删除子网，可首先删除网络的所有元素，然后删除子网标记符号。

另请参见：

- TC3 用户界面文档：命令：插入分支 [▶ 960]
- TC3 用户界面文档：命令：在上方插入分支 [▶ 960]
- TC3 用户界面文档：命令：在下方插入分支 [▶ 961]

16.1.5.4.10 FBD/LD/IL 元素执行

符号：

该元素是 1 个功能块，可用于在 FBD 和 LD 编辑器中直接输入 ST 代码。

您可以使用鼠标将 **Execute**（执行）元素从 **Toolbox**（工具箱）视图拖到您的 POU 的实现部分。点击功能块中的 **Enter ST Code here ...**（在此处输入 ST 代码……）字段，可打开 1 个输入字段，您可以在此输入多行 ST 代码。

16.1.5.4.11 LD 元素触点

符号：  ，在编辑器中： 

该元素仅限在 LD 编辑器中可用。

触点从左到右传递 TRUE（ON）或 FALSE（OFF）信号，直到信号最终到达网络右侧的线圈。为此，可为触点分配 1 个包含信号的 Boolean 变量。为此，可将触点上方的 ??? 占位符替换为 Boolean 变量的名称。

您可以将多个触点串联或并联起来。如果使用 2 个并联触点，则仅限在其中 1 个触点的值为 TRUE 时，才能向右传递 ON。如果将触点串联起来，则必须将所有触点设置为 TRUE，才能从串联的最后一个触点向右传递 ON。因此，您可以使用 LD 对并联和串联电路进行编程。

如果变量值为 FALSE，则否定触点  会传递信号 TRUE。您可以使用 **Negation**（否定）命令或 **Toolbox**（工具箱）视图中的否定触点来添加已插入的触点。

如果您将光标置于所选网络中的触点上并按下鼠标左键，在网络中会出现 Convert to Coil（转换为线圈）按钮。如果您现在按住鼠标按钮将光标移到该按钮上，并在该按钮上松开鼠标按钮，TwinCAT 会将触点转换为线圈。

另请参见：

- TC3 用户界面文档：命令：否定 [▶ 959]
- TC3 用户界面文档：命令：插入接触点 [▶ 956]
- TC3 用户界面文档：命令：插入否定的接触点 [▶ 957]
- TC3 用户界面文档：命令：插入接触点（右）[▶ 951]
- TC3 用户界面文档：命令：插入平行接触点（上）[▶ 957]
- TC3 用户界面文档：命令：插入平行接触点（下）[▶ 956]

16.1.5.4.12 LD 元素线圈

符号：  ，在编辑器中： 

该元素仅限在 LD 编辑器中可用。

线圈可获取从左侧传递过来的值，并将其存储到分配给它的 Boolean 变量中。其输入端可以有值 TRUE（ON）或 FALSE（OFF）。

您仅可将网络中的多个线圈并联起来。

在否定线圈  中，输入信号的否定值存储在分配给线圈的 Boolean 变量中。

设置线圈和重置线圈

符号：  、  ，在编辑器中：  、 

设置线圈：如果 TRUE 值到达设置线圈，则该线圈会保留 TRUE 值。只要 PLC 程序处于运行状态，此时就不再覆盖该值。

重置线圈：如果 TRUE 值到达重置线圈，则该线圈会保留 FALSE 值。只要 PLC 程序处于运行状态，此时就不再覆盖该值。

您可以使用 **FBD/LD/IL** 菜单中的 **Set/Reset**（设置/重置）命令将插入的线圈定义为设置线圈或重置线圈，或者从 **Tools** （工具）视图中将其作为设置线圈或重置线圈插入。

另请参见：

- TC3 用户界面文档：命令：设置/重置 [▶ 960]
- TC3 用户界面文档：命令：插入线圈 [▶ 955]
- TC3 用户界面文档：命令：插入复位线圈 [▶ 956]

16.1.5.4.13 LD 元素线路分支起点/终点

符号： 

该元素可用于封闭线路分支。

另请参见：

- 封闭线路分支 [▶ 606]
- TC3 用户界面文档：命令：设置分支开始点 [▶ 961]
- TC3 用户界面文档：命令：设置分支结束点 [▶ 961]

16.1.5.4.14 封闭线路分支

封闭线路分支仅限在 LD 中可用。它包含 1 个起点和 1 个终点。它可用于实现逻辑元素的并行求值。

插入封闭线路分支：

- TC3 用户界面文档：命令：插入平行接触点（下）[▶ 956]
- TC3 用户界面文档：命令：插入平行接触点（上）[▶ 957]
- TC3 用户界面文档：命令：设置分支结束点 [▶ 961]
- TC3 用户界面文档：命令：设置分支开始点 [▶ 961]

触点处的封闭线路分支

如果您已标记 1 个或多个触点并执行 **Insert Contact Parallel**（插入并联触点）命令，则会插入 1 个带有简单垂直线的平行分支。在此类分支中，信号流会通过 2 个分支。这是 2 个分支的 OR 结构。

功能块上的封闭线路分支

如果您已标记 1 个功能块并执行“Insert contact parallel”（插入并联触点）命令，则会插入 1 个带有双垂直线的平行分支。这表明实现了短路求值（SCE）。如果某个条件为 TRUE，则 SCE 允许绕过带有 Boolean 输出端的功能块的执行。在 LD 编辑器中，通过与功能块分支平行排列的分支可以表示该条件。通过该分支中的 1 个或多个并联或顺序连接的触点可以定义短路条件。

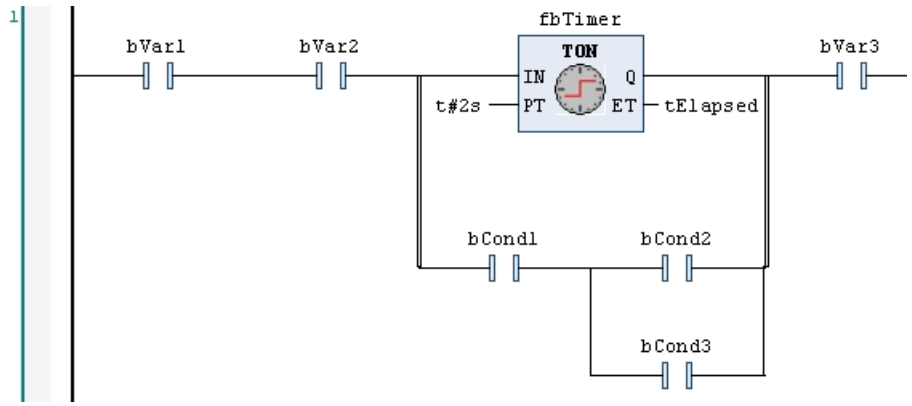
运作：

首先，考虑不包含功能块的分支。如果 TwinCAT 确定其中 1 个分支的值为 TRUE，则不会在平行分支中调用功能块。在这种情况下，会立即将应用于功能块输入端的值传输到输出端。如果 TwinCAT 确定短路求值条件为 FALSE，则会调用功能块并传递其处理的 Boolean 结果。如果所有分支均包含功能块，则会从上到下对它们进行处理，并通过逻辑 OR 运算将它们的输出端关联起来。如果没有包含功能块的分支，则执行正常的 OR 运算。

示例：

功能块实例 fbTimer（TON）有 1 个 Boolean 输入端和 1 个 Boolean 输出端。如果平行线路分支中的条件被确定为 TRUE，则省略 fbTimer 的执行。条件值由关联触点 bCond1、bCond2 和 bCond3 的 OR 和 AND 运算产生。

如果关联触点 bCond1、bCond2 和 bCond3 的条件值为 FALSE，则执行 fbTimer。



(1) 显示带有双垂直线的连接表示需要进行短路求值的结构。

(2) 显示带有单垂直线的连接表示 OR 结构。

上文介绍的 LD 示例以 ST 代码形式显示如下。bIN 和 bOUT 分别是平行线路分支的输入端（分割点）和输出端（重合点）的 Boolean 值。

```
bIN := bVar1 AND bVar2;

IF ((bIN AND bCond1) AND (bCond2 OR bCond3)) THEN
    bOUT := bIN;
ELSE
    fbTimer(IN := bIN, PT := {p 10}t#2s);
    tElapsed := fbTimer.ET;
    bOUT := fbTimer.Q;
END_IF

bRes := bOUT AND bVar3;
```

16.1.6 连续功能图（CFC）和面向页面的 CFC

从外部看，功能块图由输入端和输出端组成，数据在输入端和输出端之间进行处理。从内部看，功能图由表示数据（信号）并显示分配运算符如何在 ST 中运作的块及其连接组成。整体行为由调用其他编程块或库块的插入功能块的行为组成。

连续功能图（CFC）实现语言中的代码主要说明了数据在系统中的流动。因此，连续功能图也被称为块状图。

在面向页面的 CFC 编辑器中，您可以将编程块连接在一起，并创建分布在各个页面上的描述性功能图。面向页面的编辑器的行为与 CFC 编辑器类似，但增加了其他功能：

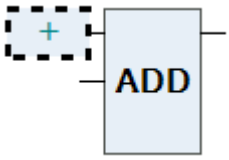
- 创建页面
- 设置页面大小
- 在页面导航器中复制和粘贴页面
- 复制实现语言 CFC 的编程块的实现并将其粘贴到页面中
- 以清晰且节省空间的方式在页边空白处排列输入端、输出端和连接标记
- 使用连接标记进行跨页连接

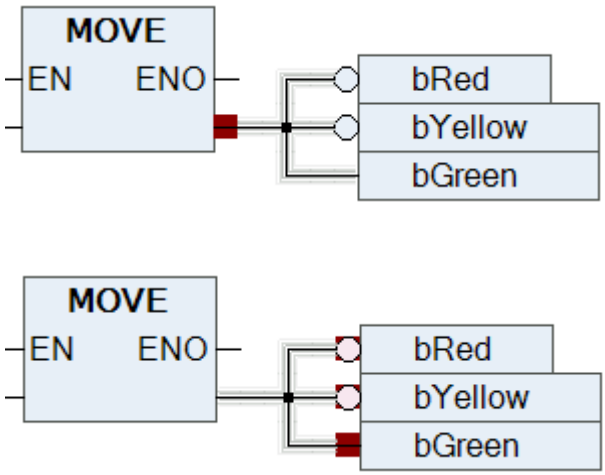
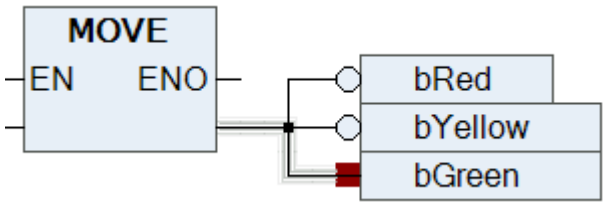
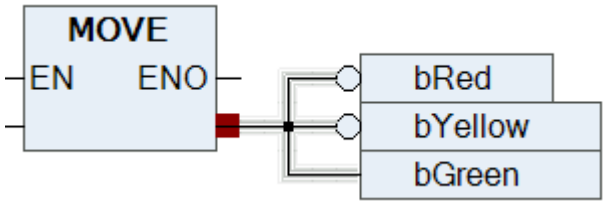
16.1.6.1 CFC 编辑器

配置编辑器

在 TwinCAT 选项的 **TwinCAT > PLC Programming Environment > CFC Editor**（TwinCAT > PLC 编程环境 > CFC 编辑器）类别中，您可以配置整个项目的外观、行为和打印。例如，在 **View**（视图）选项卡中，您可以根据数据类型配置连接线的颜色。

编辑

光标符号	要求：在 Toolbox（工具箱）视图中已选择 Pointer（指针）。 该图标表示您可以在编辑器中进行编辑。选择元素或连接以移动它们或选择命令。
光标符号	要求：在 Toolbox（工具箱）视图中已选择 1 个元素。 在编辑器中点击鼠标，即可插入所选元素。或者，您也可以将元素拖放到编辑器中。
将功能块实例从声明中拖放到编辑器中	要求：在 CFC 的声明中已选择声明行。 实例作为功能块插入，并带有块名称、块类型和所有连接。
将变量从声明中拖放到编辑器中的块连接中	变量作为输入或输出插入，并与焦点块连接进行连接。 提示：光标指示您是否正在关注变量的有效位置。 
将变量从声明部分拖放到编辑器中	要求：在声明中已选择相应的元素。 • 功能块实例：创建具有相应数据类型的功能块。 • 声明 VAR_INPUT 或 CONSTANT：插入 1 个输入元素。 • 声明 VAR_OUTPUT：插入 1 个输出元素。 • 声明 VAR、VAR_GLOBAL：在插入位置打开 1 个窗口，您可以选择要插入输入元素还是输出元素。 当您将变量从声明部分拖放到现有的可替换元素上时，现有元素就会被替换。
将功能块或编程块从项目树或库管理器中拖放到编辑器中	插入具有相应类型的功能块元素。 • 如果您将功能块拖放到现有连接线上，且功能块的输入端和输出端均与该线的数据类型兼容，则会在该线上插入功能块。在这种情况下，它的第 1 个匹配输入端和输出端会与先前由连接线连接的元素连接在一起。 • 当您将功能块拖放到现有功能块上时，现有功能块就会被替换。
通过拖放的方式重新排列功能块内的输入端和输出端的顺序。	要求：选择要重新排序到另一个位置的输入端或输出端的文本字段。
[Ctrl] + 点击编程区域	要求：在 Toolbox（工具箱）视图中已选择 1 个元素。 只要您按住 [Ctrl]，每当您点击编程区域时，都会创建 1 个所选元素。

[Ctrl] + 右箭头	要求：在 CFC 程序中，为 1 个元素**恰好**选择 1 个输出引脚。 移动选区，以便在连接线的终点选择输入引脚。如果存在多个引脚，则选择所有引脚。 示例： 
[Ctrl] + 左箭头	要求：在 CFC 程序中，为 1 个元素**恰好**选择 1 个输入引脚。 移动选区，以便在连接线的起点选择输出引脚。如果存在多个引脚，则选择所有引脚。 示例：  

另请参见：

- 所有图形编辑器的一般功能 [▶ 570]

连接

您可以在元素引脚之间插入连接线。插入的连接线具有自动路由功能，因此，它们可以自动优化并尽可能缩短。检查连接线是否存在碰撞现象。

从 1 个引脚拖放到另一个引脚中	在 2 个元素引脚之间插入 1 个连接线。
从 1 个引脚拖放到 1 个功能块中	您可以在引脚或连接文本字段上执行放下操作。 对于可扩展运算符（例如，ADD），您也可以在功能块内进行放下操作。以下行为适用： 如果仍然存在未连接的输入引脚，则会连接最上面的空闲引脚。如果不再存在未连接的输入引脚，则会在底部自动插入 1 个新引脚。
命令 **Connect selected pins**（连接所选引脚）	要求：您已选择多个引脚。引脚被标记为红色。
移动插入的元素，以便它与另一个元素的引脚相接触。	要求：选项 **Enable AutoConnect**（启用自动连接）已启用。 互相接触的引脚会自动彼此连接。
	连接图标位于编辑器的右上角。绿色图标表示无碰撞连接。红色图标表示碰撞。点击图标可打开 1 个包含碰撞处理命令的菜单，例如，**Show next collision**（显示下一个碰撞）命令。
	要求：您已选择 1 个连接和 **Connection mark**（连接标记）命令。 连接不是 1 条长连接线，而是用连接标记来表示。

另请参见：

- 命令：连接所选引脚 [▶ 944]
- 命令：显示下一个冲突 [▶ 945]
- 命令：连接标记 [▶ 948]

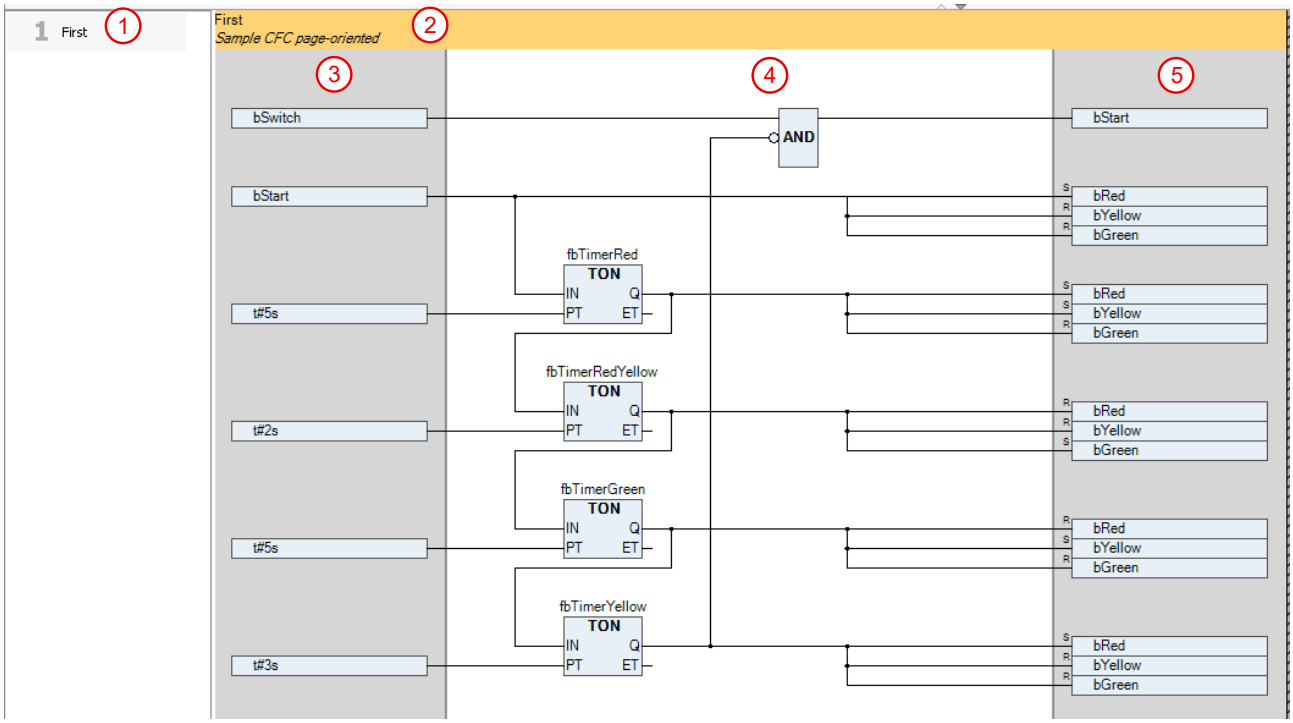
另请参见：

- 所有图形编辑器的一般功能 [▶ 570]
- CFC 语言编程 [▶ 104]
- 按数据流自动执行顺序 [▶ 110]
- 元素 [▶ 617]
- TC3 用户界面文档：CFC [▶ 938]
- TC3 用户界面文档：对话框选项 - CFC 编辑器 [▶ 897]

16.1.6.2 面向页面的 CFC 编辑器

您不能将使用**面向页面的连续功能图（CFC）**实现语言创建的编程块转换为使用**连续功能图（CFC）**实现语言的编程块，反之亦然。

使用 **Copy**（复制）和 **Paste**（粘贴）命令（通过剪贴板）或者通过拖放的方式，您可以在这 2 个编辑器之间复制元素。



1. 页面导航器
2. 带有页面名称和页面描述的页面标题
3. 为输入端和汇端连接标记预留的左侧空白区
4. 程序区
5. 为输出端和源端连接标记预留的右侧空白区

编辑

您可以将 **page**（页面）元素从 **toolbox**（工具箱）拖到页面导航中，然后插入另一个页面。

您可以在页面导航中选择现有页面，并使用 **Copy**（复制）和 **Paste**（粘贴）命令对其进行复制。

您可以使用 **Edit page size**（编辑页面大小）命令更改页面的大小。

您可以使用 **source connection mark**（源端连接标记）和 **sink connection mark**（汇端连接标记）元素创建跨页面的连接。当您将连接线从输入引脚或输出引脚拖到页边空白处时，系统会自动创建 1 个新的连接标记。**List components**（列出组件）功能可提供所有已定义的源端连接标记的概览。

当您在编辑器中选中 1 个元素后，您可以使用箭头键将选区从 1 个元素移动到下一个元素，从而在电路中进行导航。如果您已选择 1 个连接标记，但仍按下箭头键，则会选择下一页/上一页的相应连接标记。

您可以使用 **Copy**（复制）和 **Paste**（粘贴）命令将网络或元素从 CFC 编程块中传输到面向页面的 CFC 的程序区中。或者，您也可以使用拖放功能。

执行顺序

根据页面在编辑器的页面导航器中的排序，系统会自动设置执行顺序。在页面中，面向页面的 CFC 对象的行为与 CFC 对象类似。因此，您可以在 **Automatic data flow mode**（自动数据流模式）和 **Explicit execution order mode**（显式执行顺序模式）之间切换。

使用面向页面的 CFC 的附加命令

- 命令：编辑页面大小 [▶ 939]
- 命令：编辑工作表 [▶ 938]

另请参见：

- 所有图形编辑器的一般功能 [▶ 570]
- CFC 语言编程 [▶ 104]
- 按数据流自动执行顺序 [▶ 110]

- 元素 [▶ 617]
- TC3 用户界面文档：CFC [▶ 938]
- TC3 用户界面文档：对话框选项 - CFC 编辑器 [▶ 897]

16.1.6.3 在线模式下的 CFC 编辑器

在在线模式下，您可以在 CFC 编辑器中监控控制器变量值并修改它们。您还可以使用 TwinCAT 调试功能，例如，断点和逐步执行。

监控

与往常一样，您可以在声明部分和实现部分（使用内联监控）观察值。

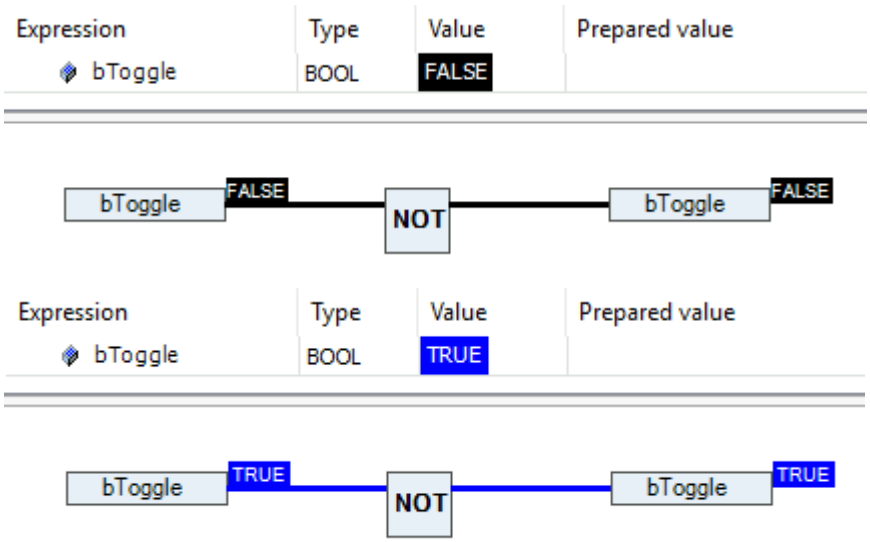
只有在功能块实例已打开的情况下，才能对功能块进行内联监控。在基本实现视图中不会显示任何值。

监控 Boolean 变量

Boolean 变量之间的连接根据其实际值以不同的颜色显示：蓝色表示 TRUE，黑色表示 FALSE。元素引脚用实际值进行装饰。

示例

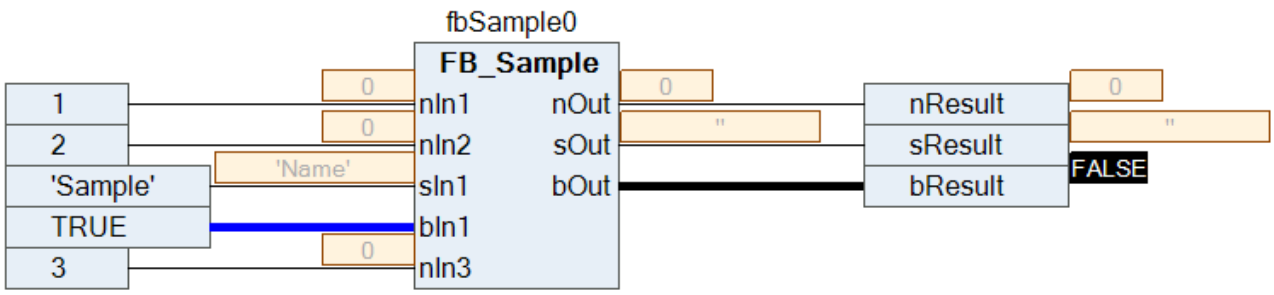
应用程序包含 1 个 CFC 编程块。1 个内部 Boolean 变量可以在这里切换：在每个总线周期，变量 bToggle 都会将其状态从 TRUE 更改为 FALSE。



监控标量变量

对于标量变量，元素引脚用实际值进行装饰。

示例



强制和写入变量值

在在线模式下，您可以在声明编辑器中为强制或写入映射变量而准备 1 个值。

如果您在 TwinCAT 选项的 **TwinCAT > PLC Environment > CFC editor**（TwinCAT > PLC 环境 > CFC 编辑器）类别下已启用 **Prepare values in implementation part**（在实现部分中准备值）选项，则您也可以在实现部分中准备值。为此，可双击每个元素旁边的监控框或直接双击元素，打开 **Prepare value**（准备值）对话框。对于 Boolean 变量，不会出现对话框，但每当鼠标点击元素时，在变量旁边显示的值都会在 TRUE 和 FALSE 之间切换。

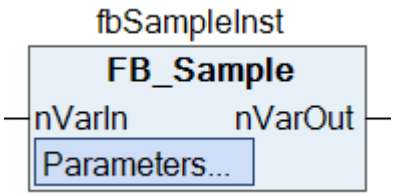
在尖括号内会显示准备值。在执行写入或强制之后，在监控框中会出现 1 个红色的“F”。

更改功能块实例的常量输入参数

在在线模式下，还可更改 VAR_CONSTANT 类型功能块实例的输入参数。参数会相应地进行调整。在退出之后，您可以使用 **Accept prepared parameter values**（接受准备参数值）命令将这些参数应用到您的项目中。

✓ CFC 编辑器处于活动状态。功能块实例化，其声明中带有 VAR_INPUT CONSTANT 变量。

1. 通过在编辑器中调用功能块实例，打开 POU。
2. 登录控制器。
3. 点击其中 1 个功能块实例的 **Parameter**（参数）字段。



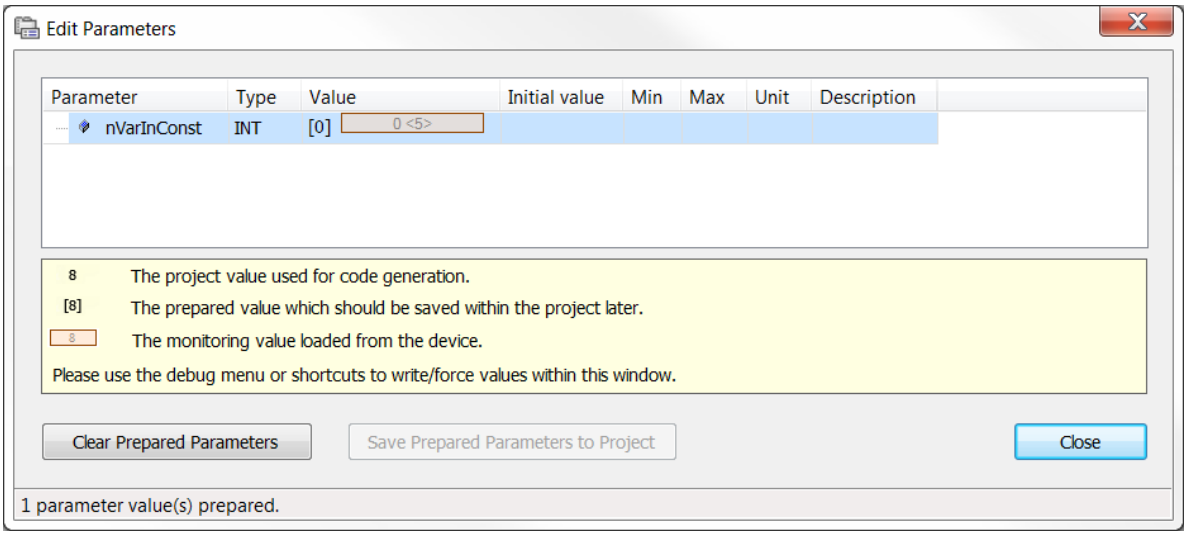
⇨ **Edit Parameters**（编辑参数）对话框会打开。

4. 在 **Value**（值）列中，点击进入参数的内联监控字段。

⇨ **Prepare Value**（准备值）对话框会打开。

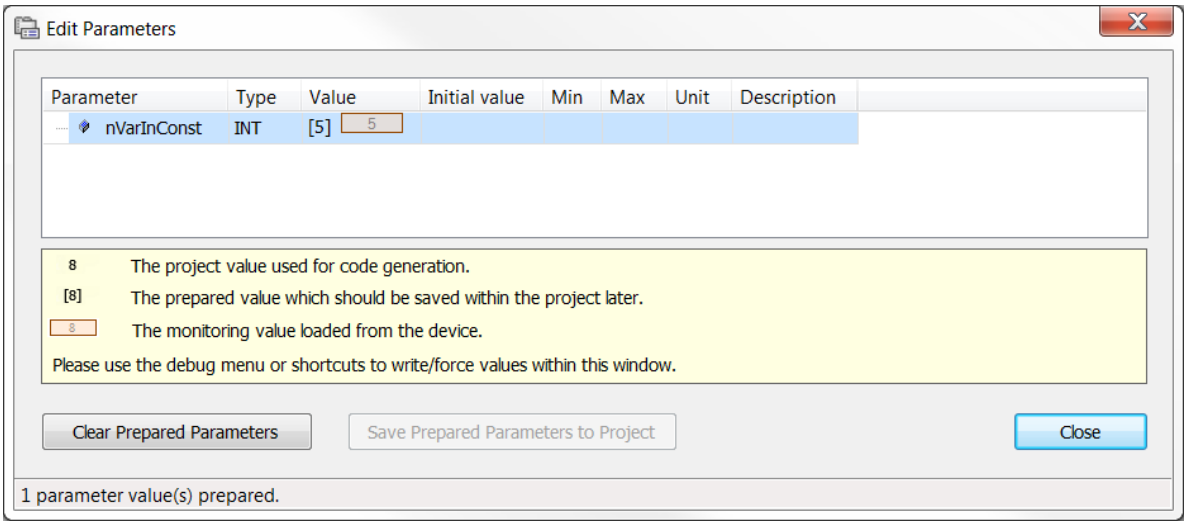
5. 在 **Prepare a new value for the next write or force operation**（为下一次写入或强制操作准备新值）字段中，输入新值。
6. 选择 **OK**（确定），确认输入。

⇨ 在当前值旁边的尖括号中会显示准备值（在本示例中为 <5>）

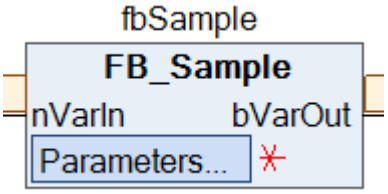


7. 使用 **PLC** 菜单或工具栏中的 **Write values**（写入值）命令，写入准备值。

⇨ 准备值已写入。参数已更改并在方括号中显示。



在功能块实例的参数字段旁，用 1 个红色的叉表示 2 个值的差值



8. 关闭 **Edit Parameters**（编辑参数）对话框。
9. 退出。
10. 点击其中 1 个功能块实例的 **Parameter**（参数）字段，或选择功能块实例并使用 **CFC** 菜单中的 **Edit Parameters**（编辑参数）命令。
    ⇨ **Edit Parameters**（编辑参数）对话框会再次打开。
11. 选择 **Save Prepared Parameters to Project**（将准备参数保存到项目中）命令。

⇨ 更改的参数值将被传输到项目中。参数字段旁的星号会消失。

另请参见：

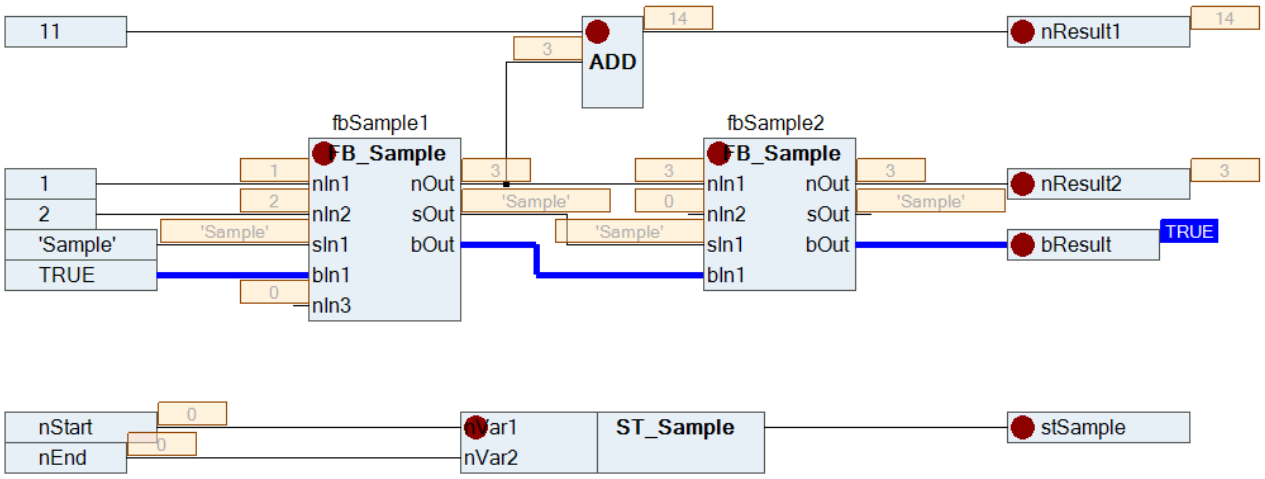
- TC3 用户界面文档：命令：编辑参数 [▶ 949]
- TC3 用户界面文档：命令：将预备参数保存到项目中 [▶ 951]

断点位置

可能的断点位置

- 元素 **Output（输出）**
  写入变量。
- 元素 **Function block（功能块）**
  调用编程块。
- 元素 **RETURN（返回）**
  程序流程分支。
- 元素 **Compositor（合成器）**
  写入树项目。

从 **Debug**（调试）菜单中选择 **Toggle Breakpoint**（切换断点）命令，设置断点或删除现有断点。块状图中的红色圆圈代表已激活的断点。



系统会在所有可调用的方法自动设置断点。
因此，如果调用了通过接口定义的方法，则会在实现该接口的功能块的所有方法中设置断点。这也适用于定义方法的所有派生功能块。

逐步执行编程块

您可以在调试模式下逐步处理编程块。调用的编程块在内部由 1 个 RETURN 补充，它位于数字为 0 的元素之前的开头和最后一个元素之后的末尾。在逐步处理过程中，可自动跳转到这些内容。

另请参见：

- 监控值 [▶ 205]
- 强制和写入变量值 [▶ 198]
- 使用断点 [▶ 195]
- 逐步处理程序（步进） [▶ 197]

16.1.6.4 为 CFC 编辑器定义热键

在 Options（选项）中，您可以在 Environment > Keyboard（环境 > 键盘）下为 CFC 编辑器中的许多命令定义单独的热键，以简化编辑工作：

命令	热键命令
全选	CFC.SelectAll
插入元素：	
插入方框。 如要选择方框，打开 **Input Assistant**（ 输入助手）对话框。	CFC.InsertBoxFromInputAssistant
插入空方框。	CFC.InsertBox
插入带 EN/ENO 的方框。 如要选择方框，打开 **Input Assistant**（ 输入助手）对话框。	CFC.InsertBoxWithENENOFromInputAssistant
插入输入端。 插入 1 个输入元素。	CFC.InsertInput
插入输出端。 插入 1 个输出元素。	CFC.InsertOutput
插入跳转。	CFC.InsertJump
编辑已插入的元素：	
否定	CFC.Negate
在设置、重置、REF 和无之间切换。	CFC.SwitchSRRefNone
重置引脚。	CFC.ResetPins

另请参见

- TC3 用户界面文档：CFC [▶ 938]
- TC3 用户界面文档：自定义键盘快捷键

16.1.6.5 元素

16.1.6.5.1 CFC 元素页面

符号： 

该元素会在编辑器中插入 1 个新页面。它仅在面向页面的 CFC 编辑器中可用。根据其位置自动分配页码。您可以在橙色标题中输入页面名称和描述。使用 **Edit page size**（编辑页面大小）命令，调整页面大小。

另请参见：

- TC3 用户界面文档：命令：编辑页面大小 [▶ 939]

16.1.6.5.2 CFC 元素控制点

符号： 

在您调整线路路由之前，您可以使用控制点来固定连接的点。为此，可将元素拖到连接线上的所需位置。带控制点的连接线不再自动路由。

另请参见：

- CFC 语言编程 [▶ 104]
- TC3 用户界面文档：命令：创建控制点 [▶ 948]
- TC3 用户界面文档：命令：删除控制点 [▶ 948]

16.1.6.5.3 CFC 元素输入

符号：

默认情况下，TwinCAT 会添加 1 个文本为 **???** 的输入元素。您可以直接点击该字段进行编辑，并输入常量值或变量名。或者，您也可以点击 … 打开输入助手，从而选择变量。

16.1.6.5.4 CFC 元素输出

符号：

默认情况下，TwinCAT 会添加 1 个文本为 **???** 的输出元素。您可以直接点击该字段进行编辑，并输入常量值或变量名。或者，您也可以点击 … 打开输入助手，从而选择变量。

16.1.6.5.5 CFC 元素功能块

符号：

您可以使用该元素插入运算符、函数、功能块或程序。默认情况下，TwinCAT 会添加 1 个名称为 **???** 的元素。您可以直接点击该字段进行编辑，并输入功能块名称。或者，您也可以点击 … 打开输入助手，并选择 1 个功能块。

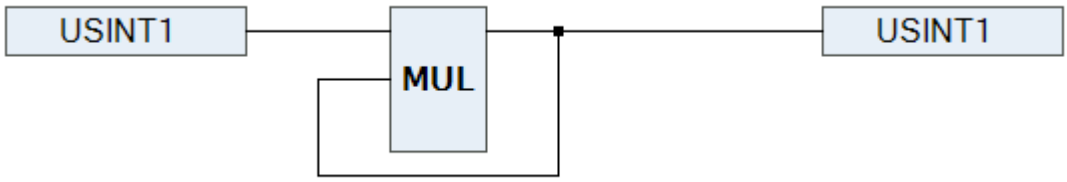
就功能块而言，TwinCAT 还会在功能块符号 **(???)** 上方显示 1 个输入字段。您必须将此名称替换为功能块实例的名称。如果您使用常量输入参数对功能块进行实例化，在功能块元素的左下角会显示 **Parameter...**（参数……）字段。您点击该字段可以编辑参数。

如要替换现有方框，只需用所需的新名称替换当前插入的标识符即可。请注意，TwinCAT 会根据 POU 的定义调整输入和输出引脚的数量，因此会删除任何现有的分配。

由于在 CFC 中允许反馈，因此在功能块（在本示例中为 temp_USINT）的输出端会生成具有数据类型的输入变量的隐式变量。如果功能块的操作结果是 1 个超出输入变量数据类型的数值范围的值，则会将溢出值写入隐式变量。实际输出变量会接收隐式变量的值（即溢出值），而不是操作的实际结果（请参见示例）。

示例

在功能块输出端隐式生成的变量：



隐式生成的代码：

```
temp_USINT := USINT1 * temp_USINT;
UDINT1     := temp_USINT;
```

另请参见：

- TC3 用户界面文档：命令：编辑参数 [▶ 949]

16.1.6.5.6 CFC 元素跳转

符号：

您可以使用该元素指定应该继续进行程序处理的位置。您必须用 1 个标记来定义该目标位置。为此，可在输入字段 **???** 中输入标记的名称。如果您已插入相应的标记，则您可以使用输入向导（ … ）进行选择。

另请参见：

- CFC 元素标记 [▶ 619]

16.1.6.5.7 CFC 元素标记

符号：

标记定义了程序在处理过程中使用跳转元素跳转到的位置。

在在线模式下，TwinCAT 会在 CFC 功能块的末尾自动插入 1 个 RETURN 标记。

另请参见：

- CFC 元素跳转 [▶ 618]

16.1.6.5.8 CFC 元素返回

符号：

使用该元素退出功能块。

请注意，在在线模式下，系统会在 CFC 编辑器的第 1 行之前和最后一个元素之后自动插入返回元素。通过逐步处理，TwinCAT 会在功能块退出之前自动跳转到返回元素。

16.1.6.5.9 CFC 元素合成器

符号： 

合成器元素可用于处理结构组件。结构的各个组件可用作输入端。为此，您必须为合成器元素分配与相应结构相同的名称（替换 ???）。

合成器元素与选择器元素相对应。

另请参见：

- CFC 元素选择器 [▶ 619]

16.1.6.5.10 CFC 元素选择器

符号： 

选择器元素可用于处理结构组件。结构的各个组件可用作输出端。为此，您必须为选择器元素分配与相应结构相同的名称（替换 ???）。

选择器元素与合成器元素相对应。

另请参见：

- CFC 元素合成器 [▶ 619]

16.1.6.5.11 CFC 元素注释

符号：

您可以使用该元素在 CFC 编辑器中输入注释。用注释文本替换元素中的占位符文本。如要插入换行符，可使用按键组合 [Ctrl] + [Enter]。

16.1.6.5.12 CFC 元素连接标记源端/目标端

符号： 、

您可以使用连接标记来代替元素之间的连接线。这有助于更加轻松地阅读复杂的图表。

如要建立有效的连接，您必须将 **Connection mark - source**（连接标记 - 源端）元素与 1 个元素的输出端关联，并将 **Connection mark - target**（连接标记 - 目标端）元素与另一个元素的输入端关联。这 2 个标记的名称必须相同。名称不区分大小写。

命名说明：

- 连接标记的默认名称是 C-<nr>。<nr> 是 1 个连续的数字，从 1 开始。
- 您可以更改默认名称。确保源端标记和目标端标记具有相同的名称。
- 如果您更改源端标记的名称，则系统会自动对目标端名称重命名。
- 如果您更改目标端标记的名称，源端名称将会保留。

请注意自动转换现有连接的命令。

另请参见：

- TC3 用户界面文档：命令：连接标记 [▶ 948]

16.1.6.5.13 CFC 元素功能块输入端

符号： 

根据功能块类型，您可以为插入的功能块元素添加更多输入端。为此，可选择功能块元素，并将功能块输入端元素拖到功能块主体上。

在按下 [Ctrl] 键的同时，您可以将输入端或输出端连接拖到功能块上的不同位置。

另请参见：

- CFC 元素功能块输出端 [▶ 620]

16.1.6.5.14 CFC 元素功能块输出端

符号： 

根据功能块类型，您可以为插入的功能块元素添加更多输出端。为此，可选择功能块元素，并将功能块输出端元素拖到功能块主体上。

在按下 [Ctrl] 键的同时，您可以将输入端或输出端连接拖到功能块上的不同位置。

另请参见：

- CFC 元素功能块输入端 [▶ 620]

16.1.7 梯形图（LD）（测试版）

16.1.7.1 梯形图编辑器

TC3.1 Build 4026 及以上的测试版本可提供新的梯形图编辑器。

梯形图编辑器是 1 种基于网络的编辑器，适用于 IEC 编程语言梯形图（LD）。在实现部分，您需要在 1 个或多个网络元素中对“电路图”进行编程。为此，您可以从工具箱中或通过菜单命令插入所需的元素，将它们关联起来，并为它们提供输入和输出变量及修饰符。

您可以赋予每个网络标题、注释和/或跳转标签。您可以注释掉网络。

插入和替换元素的方法是，用鼠标从工具箱中拖动 1 个元素到所提供的插入位置。

当您将元素拖到实现部分上时，插入位置将显示以下符号：

- 现有元素符号内带灰色背景的正方形
- 连接线上的菱形
- 指向上方或下方的三角形，用于在上方或下方插入

1 个可能的位置会“亮起”，鼠标指针会带有 1 个加号 ，当您松开鼠标按钮后，就会插入元素。

编辑器中当前所选区域会以红色突出显示并显示出轮廓。

在上下文菜单中设有用于修改（边缘检测、设置/重置、EN/ENO）、删除、重构、搜索等的合适命令。

在在线模式 [▶ 621]下，您可以使用断点、写入值和强制值进行监控和故障排除。

在 TwinCAT 选项的 Ladder [▶ 912]（梯形图）编辑器类别中定义了编辑器中的表示方法。例如，这会涉及到注释、地址的显示或者网络是否具有换行符。

在 LD/FBD 编辑器中创建的程序可读入梯形图编辑器并进行进一步编辑。

另请参见：

- 在梯形图编辑器中编程 [▶ 117]
- 用户界面文档：梯形图编辑器 [▶ 964]
- 用户界面文档：对话框：选项 - 梯形图编辑器 [▶ 912]

16.1.7.2 在线模式下的梯形图编辑器

在在线模式下，编辑器可监控值并支持当前值的写入和强制。您可以设置断点，采用颜色编码表示的连接可以实现某种流程控制。

监控

在在线模式下，在编辑器中每个变量的旁边会显示其实际值。常量变量会带有 1 个绿色的 C 符号。在 TwinCAT 选项的 Ladder editor [▶ 912]（梯形图编辑器）类别中定义了值的表示方法。

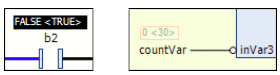
值的写入/强制

如果当前为强制变量，则在强制值前直接用 F 表示。如果为写入或强制而准备 1 个值，则该值将直接在实际值后面的尖括号中显示：*<value>*。

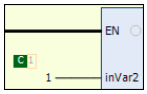
示例：

强制变量：

准备值：



常量值：

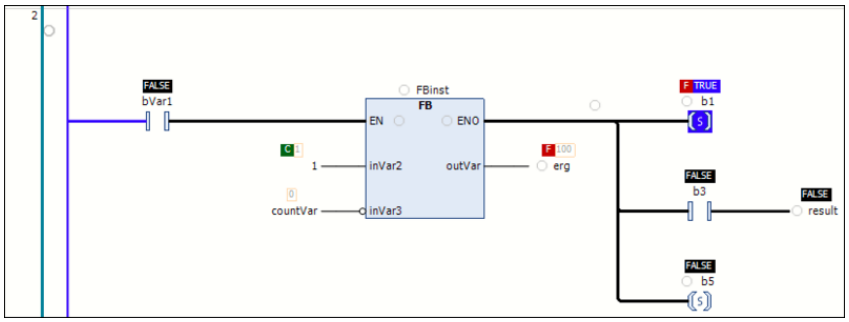


采用颜色编码表示连接

在梯形图的在线视图中，连接线以彩色显示：值为 *TRUE* 的连接线用蓝色粗线表示，值为 *FALSE* 的连接线用黑色粗线表示。未知值或模拟值的连接线正常显示（黑色细线）。

连接线的值不是从监控变量中读取，而是在编程系统中计算得出。这不是真正的流程控制。

示例：连接线和断点位置



断点

基本上，可能的断点位置为变量值可能发生变化的位置（语句）、程序分支的位置或调用另一个编程块的位置。

在编辑器中，可能的断点位置用灰色空心圆圈表示。设置的断点显示为红色实心圆圈。请参见上图：“示例：连接线和断点位置”。

可能的断点位置

- 在可调用框（功能块、函数、程序、动作、方法）上。无法使用运算符框（例如：*ADD*、*DIV*）
- 在赋值处
- 在平行分支前
- 在方框的末尾，在返回至调用框的位置。在在线模式下，此时会自动出现 1 个空网络，该网络以 **RET** 标记，而不是网络编号。
- 在方框的 **EN** 输入端和 **ENO** 输出端
- 在整个网络上。仅表示在网络中设置了断点。不能在整个网络上设置断点。

方法中的断点：TwinCAT 会在所有可以调用的方法中自动设置断点。这意味着，如果调用 1 个由接口管理的方法，则会在实现该接口的功能块中出现的所有方法以及使用该方法的所有派生功能块中设置断点。如果通过指向功能块的指针调用 1 个方法，则 TwinCAT 会在该功能块的方法以及使用该方法的所有派生功能块中设置断点。

16.1.7.3 分支

线路分支将处理线分成 2 个或多个分支，从上到下依次执行它们。您可以对每个分支进行进一步分支处理，从而在 1 个网络内创建多个分支。

每个分支都会在分支点上出现 1 个标记符号（小矩形），您可以选择它来执行子网的进一步动作/命令。

有关更多信息，请参见 在梯形图编辑器中编程 [▶ 117]

16.1.7.4 在梯形图编辑器中导航

您可以使用下述按键和按键组合在编辑器中的光标位置之间切换。跨网络也可以实现切换。

→ ←	沿着信号流的方向（即从左到右，反之亦然），切换到相邻的光标位置。
↑ ↓	如果相邻位置属于同一个逻辑组，则切换到当前位置上方或下方的下一个光标位置。 例如，1 个逻辑组由方框的所有连接组成。如果不存在这样的逻辑组：移动到下一个上部或下部相邻元素的第 1 个光标位置。在元素并行连接的情况下，将沿着第 1 个分支进行导航。
当前未实现。 Ctrl + Pos1	切换到第 1 个网络；已选中。
Ctrl + End	切换到最后一个网络；已选中。
向上翻页 ↑	向上滚动 1 页。 该页的顶部网络已选中。
向下翻页 ↓	向下滚动 1 页。 该页的底部网络已选中。

16.1.7.5 元素

在可用于插入各个元素的菜单命令的帮助页中介绍了在梯形图编辑器中可用的元素。

16.2 变量

变量的范围定义了您可以如何以及在何处使用该变量。您可以在变量声明中指定有效范围。

16.2.1 局部变量 - VAR

局部变量在关键字 VAR 和 END_VAR 之间的编程对象的声明部分进行声明。

您可以使用属性关键字扩展局部变量。

您可以访问局部变量，以便通过实例路径从编程对象外部进行读取。无法从编程对象外部进行写入访问；编译器会将此操作显示为错误。

为了保持预期的数据封装，强烈建议不要从 POU 外部访问 POU 的局部变量，无论是在读取模式下还是在写入模式下均是如此。（其他高级语言编译器也会将对局部变量的读取访问操作输出为错误）。此外，使用库功能块时，无法保证功能块的局部变量在后续更新过程中保持不变。这意味着在库更新之后，可能无法再对应用程序项目进行正确编译。

这里还需要注意静态分析中的 SA0102 规则，它决定了从外部对局部变量进行读取访问的权限。

示例：

```
VAR
    nVar1 : INT;
END_VAR
```

另请参见：

- 文档 TE1200 | PLC 静态分析：配置 > 规则
- 变量类型 - 属性关键字 [▶ 635]

16.2.2 输入变量 - VAR_INPUT

输入变量是功能块的输入变量。

VAR_INPUT 变量在关键字 VAR_INPUT 和 END_VAR 之间的编程对象的声明部分进行声明。

您可以使用属性关键字扩展输入变量。

示例：

```
VAR_INPUT
    nIn1 : INT; //1st input variable
END_VAR
```

另请参见：

- 变量类型 - 属性关键字 [▶ 635]

16.2.3 输出变量 - VAR_OUTPUT

输出变量是功能块的输出变量。

VAR_OUTPUT 变量在关键字 VAR_OUTPUT 和 END_VAR 之间的编程对象的声明部分进行声明。TwinCAT 会将这些变量的值返回至调用功能块。您可以在那里查询值并继续使用它们。

您可以使用属性关键字扩展输出变量。

示例：

```
VAR_OUTPUT
  nOut1 : INT; //1st output variable
END_VAR
```

函数和方法中的输出变量

根据 IEC 61131-3 标准，函数（和方法）可以有附加输出。您必须在调用函数时分配附加输出，如以下示例所示。

示例：

```
F_Fun(nIn1 := 1, nIn2 := 2, nOut1 => nLoc1, nOut2 => nLoc2);
```

另请参见：

- 变量类型 - 属性关键字 [▶ 635]

16.2.4 输入/输出变量 - VAR_IN_OUT，VAR_IN_OUT CONSTANT

VAR_IN_OUT 变量是输入和输出变量，是功能块接口的一部分，可作为形式传递参数。

在调用功能块时必须分配功能块的 VAR_IN_OUT 变量。

语法：

```
<keyword> <POU name>
VAR_IN_OUT
    <variable name> : <data type> ( := <initialization value> )? ;
END_VAR
<keyword> : FUNCTION | FUNCTION_BLOCK | METHOD | PRG
```

在编程块 PRG、FUNCTION_BLOCK、METHOD 或 FUNCTION 的 VAR_IN_OUT 声明部分中可以对输入和输出变量进行声明。您可以选择分配 1 个已声明数据类型的常量作为初始化值。可以读取和写入 VAR_IN_OUT 变量。

使用：

- **调用：**当调用编程块时，形式 VAR_IN_OUT 变量会收到实际变量（被称为传递变量）作为参数。在参数传递过程中，在运行时不会创建副本，但形式变量会收到从外部传输的实际变量的引用。引用变量的内部值是实际值的内存地址（作为指针传输，按引用调用）。不能直接指定常量（字面量）、常量变量或位变量作为参数。
- **编程块内的读取/写入访问：**在编程块内对变量的任何写入访问都会对传输的变量产生影响。因此，在退出编程块时，所做的任何更改都会保留下来。这意味着编程块使用其 VAR_IN_OUT 变量的方式与调用编程块使用其变量的方式相同。始终允许读取访问。
- **从外部读取/写入访问：**不能通过 <function block instance name>.<variable name> 从外部直接读取或写入 VAR_IN_OUT 变量。这仅适用于 VAR_INPUT 和 VAR_OUTPUT 变量。
- **传输字符串变量：**如果将字符串变量作为参数传输，则实际变量和形式变量应该具有相同的长度。否则，传输的字符串可能会被无意篡改。VAR_OUTPUT CONSTANT 参数不会出现这个问题。
- **传输位变量：**不能直接将位变量传输到 VAR_IN_OUT 变量，而需要 1 个中间变量。
- **传输属性：**不允许。

将字符串传输到 VAR_IN_OUT CONSTANT

如果将字符串作为变量或常量传输到形式 VAR_IN_OUT CONSTANT 变量，则字符串长度可能会改变。不过，最多只能处理 VAR_IN_OUT CONSTANT 变量的字符串长度。有关更多信息，请参见本页底部。

无法传输属性

无法将任何属性传输到 VAR_IN_OUT 或 VAR_IN_OUT CONSTANT 变量。

例外：如果属性的返回类型为 REFERENCE，则可以对 VAR_IN_OUT（CONSTANT）赋值。但请注意，属性只能通过 REFERENCE 返回 1 个在属性调用后仍然有效的地址。例如，如果属性返回对临时变量的引用，则情况不会是这样。

示例：

功能块 FB_Sample

```
FUNCTION_BLOCK FB_Sample
VAR_IN_OUT
    bInOut    : BOOL;
END_VAR
```

MAIN 程序：

```
VAR
    bTest     : BOOL;
    fbSample  : FB_Sample;
END_VAR
```

```
fbSample(bInOut := bTest);  // OK
fbSample();                 // NOK: not possible as the VAR_IN_OUT variable is not assigned in this
FB call
fbSample.bInOut := bTest;   // NOK: direct access to VAR_IN_OUT variable from the outside not
possible
```

将位变量分配给 VAR_IN_OUT 输入端的变通方法示例：

声明位变量（bBit0）：

```
VAR_GLOBAL
    bBit0 AT %MX0.1 : BOOL;
    bTemp           : BOOL;
END_VAR
```

带有 VAR_IN_OUT 输入端 bInOut 的功能块：

```
FUNCTION_BLOCK FB_Test
VAR_INPUT
    bIn    : BOOL;
END_VAR
VAR_IN_OUT
    bInOut : BOOL;
END_VAR
```

```
IF bIn THEN
    bInOut := TRUE;
END_IF
```

调用功能块的程序。将位变量直接分配给 VAR_IN_OUT 输入端（错误）和使用中间变量分配（变通方法）：

```
PROGRAM MAIN
VAR
    bIn      : BOOL;
    fbTest1  : FB_Test;
    fbTest2  : FB_Test;
END_VAR
```

```
// Error C0201: Type 'BIT' doesn't correspond to the type 'BOOL' of VAR_IN_OUT 'bInOut'
fbTest1(bIn := bIn, bInOut := bBit0);

// Workaround
//bTemp := bBit0;
//fbTest2(bIn := bIn, bInOut := bTemp);
//bBit0 := bTemp;
```

传输变量 VAR_IN_OUT CONSTANT

作为常量传输参数，VAR_IN_OUT CONSTANT 变量可供读取，但不能写入。

语法：

```
<keyword> <POU name>
VAR_IN_OUT CONSTANT
    <variable name> : <data type>; // formal parameter
END_VAR
<keyword> : FUNCTION | FUNCTION_BLOCK | METHOD | PRG
```

声明 VAR_IN_OUT CONSTANT 变量，而不分配初始化值。

使用：

- 允许使用所有数据类型。
- 不允许对 VAR_IN_OUT CONSTANT 变量进行写入访问。
- 不允许传输属性。
- 如果 VAR_IN_OUT CONSTANT 变量为 STRING/WSTRING 类型，则在调用编程块时可以传输变量、常量变量或常量（字面量）。对于传输的变量/常量的字符串长度没有限制，长度也不取决于 VAR_IN_OUT CONSTANT 变量的字符串长度。请注意，最多只能处理 VAR_IN_OUT CONSTANT 变量的字符串长度。
- 如果 VAR_IN_OUT CONSTANT 变量不是 STRING/WSTRING 类型，则在调用编程块时可以传输变量。如果在 PLC 项目属性的 **Compile**（编译）类别中禁用编译器选项 **Replace constants**（替换常量），则可以传输常量变量。

如果在 PLC 项目属性的 **Compile**（编译）类别中启用编译器选项 **Replace constants**（替换常量），则基本数据类型为 STRING 以外的常量或基本数据类型为 STRING 以外的常量变量的参数传输会产生编译器错误。

示例：

在代码中，通过各种 VAR_IN_OUT 变量可以将字符串传输到 F_Manipulate 函数。如果将字面量传输到 VAR_IN_OUT 变量，则会发出编译器错误。在将字面量传输到 VAR_IN_OUT CONSTANT 变量时以及在传输字符串变量时，系统会生成校正代码。

同样应该注意到的是，不能将太短的 STRING 变量传输到 VAR_IN_OUT 变量（编译器错误），但可以将此类变量传输到 VAR_IN_OUT CONSTANT 变量。

函数 F_Manipulate：

```
FUNCTION F_Manipulate : BOOL
VAR_IN_OUT
    sReadWrite   : STRING(16);  (* Can be read or written here in POU *)
    nReadWrite   : DWORD;       (* Can be read or written here in POU *)
END_VAR
VAR_IN_OUT CONSTANT
    cReadOnly    : STRING(16);  (* Constant string variable can only be read here in POU *)
END_VAR
```

```
sReadWrite := 'String_from_POU';
nReadWrite := STRING_TO_DWORD(cReadOnly);
```

MAIN 程序：

```
PROGRAM MAIN
VAR
    sVar10 : STRING(10) := '1234567890';
    sVar16 : STRING(16) := '1234567890123456';
    sVar20 : STRING(20) := '12345678901234567890';
    nVar   : DWORD;
END_VAR
```

```
// The following line of code causes the compiler error:
// VAR_IN_OUT parameter 'sReadWrite' of 'F_Manipulate' needs variable with write access as input.
F_Manipulate(sReadWrite := '1234567890123456', cReadOnly := '1234567890123456', nReadWrite := nVar);

// The following line of code causes the compiler error:
// String variable 'sVar10' too short for VAR_IN_OUT parameter 'sReadWrite' of 'F_Manipulate'
F_Manipulate(sReadWrite := sVar10, cReadOnly := sVar10, nReadWrite := nVar);

// Correct code
F_Manipulate(sReadWrite := sVar16, cReadOnly := '1234567890', nReadWrite := nVar);
F_Manipulate(sReadWrite := sVar16, cReadOnly := '1234567890123456', nReadWrite := nVar);
```

```
F_Manipulate(sReadWrite := sVar16, cReadOnly := '12345678901234567890', nReadWrite := nVar);
F_Manipulate(sReadWrite := sVar16, cReadOnly := sVar10, nReadWrite := nVar);
F_Manipulate(sReadWrite := sVar16, cReadOnly := sVar16, nReadWrite := nVar);
F_Manipulate(sReadWrite := sVar20, cReadOnly := sVar20, nReadWrite := nVar);
```

16.2.5 全局变量 - VAR_GLOBAL

全局变量是在整个项目中已知的”正常“变量、常量或保留变量。

全局变量在关键字 VAR_GLOBAL 和 END_VAR 之间的全局变量列表中进行声明。

如果变量名前缀带点，例如“.nGlobVar1”，则系统会检测到全局变量。

如果在功能块中局部声明的变量与全局变量同名，则该变量在功能块内具有优先权。

全局变量始终在 POU 的局部变量之前进行初始化。

示例：

```
VAR_GLOBAL
    nVarGlob1 : INT;
END_VAR
```

另请参见：

- 对象全局变量列表 [▶ 66]
- 全局命名空间 [▶ 654]

16.2.6 临时变量 - VAR_TEMP

该功能是 IEC 61131-3 标准的扩展。

临时变量在关键字 VAR_TEMP 和 END_VAR 之间进行局部声明。

VAR_TEMP 声明只能在程序和功能块中使用。

每次调用功能块时，TwinCAT 都会对临时变量进行重新初始化。

应用程序只能访问程序或功能块的实现部分中的临时变量。

示例：

```
VAR_TEMP
    nVarTmp1 : INT; //1st temporary variable
END_VAR
```

16.2.7 静态变量 - VAR_STAT

该功能是 IEC 61131-3 标准的扩展。

您可以在关键字 VAR_STAT 和 END_VAR 之间对静态变量进行局部声明。在首次调用相应功能块时，TwinCAT 会对静态变量进行初始化。

您只能在声明变量的命名空间内访问静态变量（就像 C 语言中的静态变量一样）。不过，当应用程序退出功能块时，静态变量会保留其值。例如，您可以使用静态变量作为函数调用的计数器。

您可以使用属性关键字扩展静态变量。

静态变量只存在 1 次。这也适用于功能块或功能块方法的静态变量，即使对功能块多次进行实例化也是如此。

示例：

```
VAR_STAT
    nVarStat1 : INT;
END_VAR
```

另请参见：

- 变量类型 - 属性关键字 [▶ 635]

16.2.8 外部变量 - VAR_EXTERNAL

外部变量是被“导入”到功能块中的全局变量。

您可以在关键字 VAR_EXTERNAL 和 END_VAR 之间对变量进行声明。如果全局变量不存在，则会发出错误消息。

在 TwinCAT 3 PLC 中，无需将变量声明为外部变量，即可在 POU 中使用它们。该关键字的存在是为了确保遵守 IEC 61131-3 标准。

语法：

```
<POU keyword> <POU name>
VAR_EXTERNAL
    <variable name> : <data type>;
END_VAR
```

不允许初始化。

确保您仅在全局变量列表中对已分配的变量进行寻址（使用 AT %I 或 AT %Q）。对局部变量实例的额外寻址会导致过程映像中的重复。

示例：

```
FUNCTION_BLOCK FB_Sample
VAR_EXTERNAL
    nVarExt1 : INT;     // 1st external variable
END_VAR
```

另请参见：

- 对象全局变量列表 [▶ 66]

16.2.9 实例变量 - VAR_INST

TwinCAT 不像 VAR 变量那样在方法堆栈上创建方法的 VAR_INST 变量，而是在功能块实例的堆栈上进行创建。这意味着 VAR_INST 变量的行为与功能块实例的其他变量一样，不会在每次调用方法时进行重新初始化。

仅可在功能块的方法中使用 VAR_INST 变量，而且，仅可在方法中访问此类变量。您可以在方法的声明部分监控实例变量的变量值。

实例变量不能使用属性关键字进行扩展。

示例：

```
METHOD MethLast : INT
VAR_INPUT
    nVar  : INT;
END_VAR
VAR_INST
    nLast : INT := 0;
END_VAR
```

```
MethLast := nLast;
nLast    := nVar;
```

另请参见：

- 对象方法 [▶ 82]

16.2.10 常量变量 - CONSTANT

常量变量在全局变量列表或编程对象的声明部分进行声明。在实现过程中，通过实例路径可以对常量变量进行读取访问，但不能进行写入访问。

语法

```
<scope> CONSTANT
    <identifier> : <data type> := <initial value> ;
END_VAR

<scope> : VAR | VAR_INPUT | VAR_STAT | VAR_GLOBAL
<data type>: <elementary data type> | <user defined data type> | <function block>
<initial value> : <literal value> | <identifier> | <expression>
```

在声明常量变量时，一定要指定初始化值。之后，再不能写入常量。

示例：

声明：

```
VAR CONSTANT
    cTaxFactor : REAL := 1.19;
END_VAR
```

调用：

```
rPrice := rValue * cTaxFactor;
```

您只能在只读实现中访问常量变量。常量变量位于分配运算符的右侧。

另请参见：

- 输入/输出变量 - VAR_IN_OUT，VAR_IN_OUT CONSTANT [▶ 624]
- 操作数 [▶ 686]

16.2.11 通用常量变量 - VAR_GENERIC CONSTANT

通用常量是功能块的 VAR_GENERIC CONSTANT 声明部分中的变量，在声明功能块实例之前不会对其进行定义。

TC3.1 Build 4026 及以上可用

语法

功能块的声明：

```
FUNCTION_BLOCK <function block name>
VAR_GENERIC CONSTANT
    <generic constant name> : <integer data type> := <initial value>; //Initialwert wird
überschrieben
END_VAR
```

与往常一样，在功能块中可以使用通用常量（整数数据类型）。例如，您可以使用常量来定义数组的大小或字符串的长度。初始值仅需要用于编译检查。在运行时，该值会被覆盖。

功能块实例的声明：

```
PROGRAM MAIN
VAR
    <fb instance name> : <function block name> < <literal> >;
    <fb instance name> : <function block name> <( <expression> )>;
END_VAR
```

在声明功能块实例时，会为常量分配 1 个仅对该实例有效的特定值。为此，将值（<literal>）附加到作为数据类型的功能块的尖括号中。

此外，还可以附加表达式（<expression>）。不过，必须用圆括号将其括起来，因为在表达式中允许使用诸如 < 或 > 等符号。否则，再不能保证代码的唯一性。

示例

带可参数化数组变量的通用功能块

下面的代码展示了如何定义 1 个可以处理任意长度数组的功能块。该功能块有 1 个通用但长度恒定的数组。所谓“恒定”，是指虽然每个功能块实例的数组长度各不相同，但是，它在对象的生命周期内是恒定的。

例如，这种结构对希望实现通用库功能块的库程序员很有帮助。

```
FUNCTION_BLOCK FB_Sample
VAR_GENERIC CONSTANT
    nMaxLen : UDINT := 1;
END_VAR
VAR
    aSample : ARRAY[0..nMaxLen-1] OF BYTE
END_VAR
```

```
PROGRAM MAIN
VAR CONSTANT
    cConst    : DINT := 10;
END_VAR
VAR
    fbSample1 : FB_Sample<100>;
    fbSample2 : FB_Sample<(2*cConst)>;
    aSample   : ARRAY[0..5] OF FB_Sample<10>;
END_VAR
```

继承

带有通用常量的功能块也可以使用继承结构 EXTENDS 和 IMPLEMENTS。

在实现时，应确保先插入通用常量的声明，然后再插入 EXTENDS 和 IMPLEMENTS。这样做的原因是，通用常量也可以与基类一起使用。

示例

```
INTERFACE I_Sample
```

```
FUNCTION_BLOCK FB_Sample
VAR_GENERIC CONSTANT
    nMaxLen   : UDINT := 1;
END_VAR
IMPLEMENTS I_Sample
VAR
    aSample   : ARRAY[0..nMaxLen-1] OF BYTE
END_VAR
```

```
FUNCTION_BLOCK FB_Sub_1 EXTENDS FB_Sample<100>
```

```
FUNCTION_BLOCK FB_Sub_2
VAR_GENERIC CONSTANT
    nMaxLen2  : UDINT := 1;
END_VAR
EXTENDS FB_Sample<nMaxLen2>
```

```
PROGRAM MAIN
VAR
    fbSample1 : FB_Sample<10>;
    fbSub1    : FB_Sub_1;
    fbSub2    : FB_Sub_2<20>;
END_VAR
```

16.2.12 剩余变量 - PERSISTENT，RETAIN

在正常程序运行时之外，剩余变量仍可保留其值。在 PLC 项目中，您可以将剩余变量声明为 RETAIN 变量，或者更严格地声明为 PERSISTENT 变量。

实现 RETAIN 变量的全部功能的先决条件是，在控制器（NovRam）上有相应的内存区。只有在 TwinCAT 关闭时，才会写入持久变量。这通常需要相应的 UPS。例外：使用 FB_WritePersistentData 功能块也可以写入持久变量。

如果相应的内存区不存在，则在断电时会丢失 RETAIN 和 PERSISTENT 变量的值。

AT 声明不得与 VAR RETAIN 或 VAR PERSISTENT 组合使用。

持久变量

在编程对象的声明部分，通过在变量类型的关键字（VAR、VAR_GLOBAL 等）后添加关键字 PERSISTENT，您可以对持久变量进行声明。

在失控终止、冷重置或重新下载 PLC 项目后，PERSISTENT 变量会保留其值。
当程序重新启动时，系统将继续以存储的值运行。在这种情况下，TwinCAT 会使用明确指定的初始值或默认初始化对“普通”变量进行重新初始化。
换句话说，TwinCAT 仅会在重置原点期间对 PERSISTENT 变量进行重新初始化。

持久变量的 1 个应用示例是运行小时计数器，该计数器需要在断电后以及再次下载 PLC 项目时继续计数。

显示 PERSISTENT 变量行为的概览表格

在在线命令之后	VAR PERSISTENT
冷重置	保留值
重置原点	对值重新初始化
下载	保留值
在线更改	保留值

示例：

持久变量列表：

```
VAR_GLOBAL PERSISTENT
    nVarPers1 : DINT; (* 1. Persistent variable *)
    bVarPers2 : BOOL; (* 2. Persistent variable *)
END_VAR
```

- 避免在持久变量列表中使用 POINTER TO 数据类型，因为在再次下载 PLC 项目时，地址值可能会发生变化！TwinCAT 会发出相应的编译器警告。
- 在函数中将局部变量声明为 PERSISTENT 没有任何效果。不能以这种方式使用数据持久性。
- 通过编译指示“TcInitOnReset [▶ 766]”可以影响**冷重置**期间的行为

RETAIN 变量

在编程对象的声明部分，通过在变量类型的关键字（VAR、VAR_GLOBAL 等）后添加关键字 RETAIN，您可以对 RETAIN 变量进行声明。

声明为 RETAIN 的变量取决于目标系统，但通常在 1 个单独的内存区中进行管理，务必保护该内存区不会发生断电。所谓的保留处理程序可确保在 PLC 周期结束时写入 RETAIN 变量，并且仅在 NovRam 的相应区域中写入。在 C/C++ 文档的“保留数据”章节中介绍了保留处理程序的处理方法。

在失控终止（断电）后，RETAIN 变量会保留其值。当程序重新启动时，系统将继续以存储的值运行。在这种情况下，TwinCAT 会使用明确指定的初始值或默认初始化对“普通”变量进行重新初始化。
在重置原点时，TwinCAT 会对 RETAIN 变量进行重新初始化。

1 种可能的应用是生产工厂的零件计数器，它在断电后仍能继续计数。

显示 RETAIN 变量行为的概览表格

在在线命令之后	VAR RETAIN
冷重置	保留值
重置原点	对值重新初始化
下载	保留值

示例：

在 POU 中：

```
VAR RETAIN
    nRem1 : INT;
END_VAR
```

在 GVL 中：

```
VAR_GLOBAL RETAIN
    nVarRem1 : INT;
END_VAR
```

- 如果您在程序或功能块中将局部变量声明为 RETAIN，则 TwinCAT 会将该特定变量存储在保留区域（就像全局 RETAIN 变量一样）。
- 如果您在函数中将局部变量声明为 RETAIN，则无效。TwinCAT 不会将变量存储在保留区域。

完整概览表格

RETAIN 变量的保留程度自动包含在 PERSISTENT 变量的保留程度中。

在在线命令之后	VAR	VAR RETAIN	VAR PERSISTENT
冷重置	对值重新初始化	保留值	保留值
重置原点	对值重新初始化	对值重新初始化	对值重新初始化
下载	对值重新初始化	保留值	保留值
在线更改	保留值	保留值	保留值

另请参见：

- 数据持久性 [▶ 151]
- 重置 PLC 项目 [▶ 201]

16.2.13 SUPER

SUPER 是 1 个用于面向对象的编程的特殊变量。

SUPER 是 1 个功能块的指针，指向创建该功能块的基本功能块实例。因此，SUPER 指针也允许访问基本功能块（基类）方法的实现。系统会为每个功能块自动提供 1 个 SUPER 指针。

您仅可在方法实现以及相关的功能块实现中使用 SUPER。

解除引用指针：SUPER^

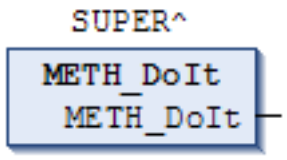
使用 SUPER 指针：使用 SUPER 关键字调用基类的实现或在基类实例中有效的方法。

示例：

ST：

```
SUPER^();                   // Call of FB-body of base class
SUPER^.METH_DoIt();         // Call of method METH_DoIt that is implemented in base class
```

FBD/CFC/LD：



对于指令表（IL），未实现 SUPER。

使用 SUPER 和 THIS 指针：

功能块 FB_Base：

```
FUNCTION_BLOCK FB_Base
VAR_OUTPUT
    nCnt : INT;
END_VAR
```

方法 FB_Base.METH_DoIt：

```
METHOD METH_DoIt : BOOL
```

```
nCnt := -1;
```

方法 FB_Base.METH_DoAlso：

```
METHOD METH_DoAlso : BOOL
```

```
METH_DoAlso := TRUE;
```

功能块 FB_1：

```
FUNCTION_BLOCK FB_1 EXTENDS FB_Base
VAR_OUTPUT
    nBase: INT;
END_VAR
```

```
THIS^.METH_DoIt();      // Call of the methods of FB_1
THIS^.METH_DoAlso();

SUPER^.METH_DoIt();     // Call of the methods of FB_Base
SUPER^.METH_DoAlso();
nBase := SUPER^.nCnt;
```

方法 FB_1.METH_DoIt：

```
METHOD METH_DoIt : BOOL
```

```
nCnt := 1111;
METH_DoIt := TRUE;
```

方法 FB_1.METH_DoAlso：

```
METHOD METH_DoAlso : BOOL
```

```
nCnt := 123;
METH_DoAlso := FALSE;
```

MAIN 程序：

```
PROGRAM MAIN
VAR
    fbMyBase : FB_Base;
    fbMyFB_1 : FB_1;
    nTHIS    : INT;
    nBase    : INT;
END_VAR
```

```
fbMyBase();
nBase := fbmyBase.nCnt;
fbMyFB_1();
nTHIS := fbMyFB_1.nCnt;
```

另请参见：

- 数据类型 [▶ 697]
- THIS [▶ 633]

16.2.14 THIS

THIS 是 1 个用于面向对象的编程的特殊变量。

THIS 是功能块的指针，指向其自身功能块实例。系统会为每个功能块自动提供 1 个 THIS 指针。

您仅可在方法和功能块中使用 THIS。THIS 可用于 **Keywords**（关键字）类别的 **input assistant**（输入助手）中的实现。

解除引用指针：THIS^

使用 THIS 指针：

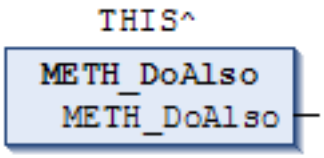
- 如果功能块变量被方法中的局部变量遮蔽，则 THIS 指针可用于设置功能块变量。请参见下文示例（1）
- THIS 指针还可用于通过函数调用为自己的 FB 实例分配 1 个指针。（请参见下文示例（2））

示例：

ST：

```
THIS^.METH_DoIt();
```

FBD/CFC/LD：



对于指令表（IL），未实现 THIS。

示例：

（1）功能块变量 nVarB 被局部变量 nVarB 遮蔽。

```
FUNCTION_BLOCK FB_A
VAR_INPUT
    nVarA: INT;
END_VAR

nVarA := 1;

FUNCTION_BLOCK FB_B EXTENDS FB_A
VAR_INPUT
    nVarB : INT := 0;
END_VAR

nVarA := 11;
nVarB := 2;

METHOD DoIt : BOOL
VAR_INPUT
END_VAR
VAR
    nVarB : INT;
END_VAR

nVarB := 22; // The local variable nVarB is set.
THIS^.nVarB := 222; // The function block variable nVarB is set even though nVarB is obscured.

PROGRAM MAIN
VAR
   fbMyfbB : FB_B;
END_VAR

fbMyfbB(nVarA:=0, nVarB:= 0);
fbMyfbB.DoIt();
```

（2）函数调用需要引用自己的 FB 实例。

```
FUNCTION F_FunA
VAR_INPUT
    fbMyFbA : FB_A;
END_VAR
...;

FUNCTION_BLOCK FB_A
VAR_INPUT
    nVarA: INT;
END_VAR
...;

FUNCTION_BLOCK FB_B EXTENDS FB_A
VAR_INPUT
    nVarB: INT := 0;
END_VAR

nVarA := 11;
nVarB := 2;

METHOD DoIt : BOOL
VAR_INPUT
END_VAR
VAR
```

```
    nVarB: INT;
END_VAR

nVarB := 22;  //The local variable nVarB is set.
F_FunA(fbMyFbA := THIS^);  //F_FunA is called via THIS^.

PROGRAM MAIN
VAR
    fbMyFbB: FB_B;
END_VAR

fbMyFbB(nVarA:=0 , nVarB:= 0);
fbMyFbB.DoIt();
```

另请参见：

- 指针 [▶ 708]
- SUPER [▶ 632]

16.2.15 变量类型 - 属性关键字

在变量声明中，您可以为变量类型添加以下关键字：

- RETAIN：用于 RETAIN 类型的剩余变量
- PERSISTENT：用于 PERSISTENT 类型的剩余变量
- CONSTANT：用于常量

另请参见：

- 常量 [▶ 686]
- 剩余变量 - RETAIN, PERSISTENT [▶ 630]

16.3 运算符

TwinCAT 3 PLC 支持 IEC 61131-3 标准的所有运算符。这些运算符在整个项目中均为隐式已知的。除了 IEC 运算符之外，TwinCAT 3 PLC 还支持一些在 IEC 61131-3 标准中没有描述的运算符。

运算符在功能块中的使用类似于函数。

浮点数据类型的操作

对于浮点数据类型的操作，结果取决于所使用的目标系统硬件。

存在上溢或下溢的操作

对于数据类型中存在上溢或下溢的操作，计算结果取决于所使用的目标系统硬件。

有关 ST 运算符处理顺序（绑定强度）的信息，请参见“ST 表达式 [▶ 572]”部分。

数据类型中的上溢/下溢

TwinCAT 3 编译器为相应的目标设备生成代码，并始终以目标系统指定的原生大小计算中间结果。例如，在 x86 和 ARM 系统中至少使用 32 位中间值，而在 x64 系统中则始终使用 64 位中间值。这在计算速度方面有很大优势，通常能够产生预期的结果。不过，这也意味着数据类型中的上溢或下溢可能不会被截断。

示例 1

该加法运算的结果不会被截断，dwVar 中的结果为 65536。

```
VAR
    nVarWORD  : WORD;
    nVarDWORD : DWORD;
END_VAR
```

```
nVarWORD  := 65535;
nVarDWORD := nVarWORD + 1;
```

示例 2

数据类型中的上溢和下溢不会被截断，在 32 位和 64 位硬件上，2 次比较的结果（bVar1、bVar2）均为 FALSE。

```
VAR
    nVar1 : WORD;
    nVar2 : WORD;
    bVar1 : BOOL;
    bVar2 : BOOL;
END_VAR
```

```
nVar1 := 65535;
nVar2 := 0;
bVar1 := (nVar1 + 1) = nVar2;
bVar2 := (nVar2 - 1) = nVar1;
```

示例 3

对 nVar3 的赋值会将值截断为目标数据类型 WORD，结果 bVar1 为 TRUE。

```
VAR
    nVar1 : WORD;
    nVar2 : WORD;
    nVar3 : WORD;
    bVar1 : BOOL;
END_VAR
```

```
nVar1 := 65535;
nVar2 := 0;
nVar3 := (nVar1 + 1);
bVar1 := (nVar3 = nVar2);
```

示例 4

为了强制编译器截断中间结果，可添加 1 个转换。

类型转换可确保 2 次比较仅比较 16 位，并且 2 次比较的结果（bVar1、bVar2）均为 TRUE。

```
VAR
    nVar1 : WORD;
    nVar2 : WORD;
    bVar1 : BOOL;
    bVar2 : BOOL;
END_VAR
```

```
nVar1 := 65535;
nVar2 := 0;
bVar1 := (TO_WORD(nVar1 + 1) = nVar2);
bVar2 := (TO_WORD(nVar2 - 1) = nVar1);
```

地址运算符

- ADR [▶ 639]
- BITADR [▶ 640]
- 内容运算符 [▶ 640]

算术运算符

- ADD [▶ 641]
- SUB [▶ 642]
- MUL [▶ 643]
- DIV [▶ 643]
- MOD [▶ 644]
- MOVE [▶ 645]
- INDEXOF [▶ 645]
- SIZEOF [▶ 645]
- XSIZEOF [▶ 646]

调用运算符

- CAL [▶ 646]

选择运算符

- LIMIT [▶ 646]
- MAX [▶ 647]
- MIN [▶ 647]
- MUX [▶ 648]
- SEL [▶ 648]

位移运算符

- ROL [▶ 650]
- ROR [▶ 651]
- SHL [▶ 649]
- SHR [▶ 650]

位串运算符

- AND [▶ 651]
- AND_THEN [▶ 652]
- NOT [▶ 652]
- OR [▶ 652]
- OR ELSE [▶ 653]
- XOR [▶ 653]

命名空间运算符

命名空间运算符是 IEC 61131-3 运算符的扩展。即使您在项目中多次使用相同的变量或模块名称，它们也提供了使对变量或模块的访问具有唯一性的选项。

- 库命名空间 [▶ 654]
- 枚举命名空间 [▶ 655]
- 全局命名空间 [▶ 654]
- 全局变量列表的命名空间 [▶ 654]

数字运算符

- ABS [▶ 655]
- ACOS [▶ 655]
- ASIN [▶ 655]
- ATAN [▶ 656]
- COS [▶ 656]
- EXP [▶ 656]
- EXPT [▶ 657]
- LN [▶ 657]
- LOG [▶ 658]
- SIN [▶ 658]
- SQRT [▶ 658]
- TAN [▶ 659]

类型转换运算符

类型转换有隐式类型转换和显式类型转换之分。

隐式类型转换

从“较小”类型到“较大”类型的转换，例如从 BYTE 转换为 INT 或从 WORD 转换为 DINT，可以显式进行，但也可以隐式进行，无需调用转换运算符。

显式类型转换

可以显式调用类型转换运算符。对于从一种基本类型到另一种基本类型的类型转换以及重载，可以使用下述类型转换运算符。

类型转换：

```
<elementary data type>_TO_<another elementary data type>
```

重载转换：

```
TO_<elementary data type>
```

基本数据类型：

```
<elementary data type> =
__UXINT | __XINT | __XWORD | BIT | BOOL | BYTE | DATE | DINT | DT | DWORD | INT | LDATE | LDT | LINT
| LREAL | LTIME | LTOD | LWORD | REAL | SINT | TIME | TOD | UDINT | UINT | ULINT | USINT | WORD
```

关键字 TIME_OF_DAY 和 DATE_AND_TIME 是数据类型 TOD 和 DT 的另一种拼写形式。

- TIME_OF_DAY = TOD
- DATE_AND_TIME = DT

TIME_OF_DAY 和 DATE_AND_TIME 没有被映射为类型转换命令。请改用数据类型 TOD 和 DT。

如果超出值范围，则不会定义结果

如果类型转换运算符的输入值超出输出数据类型的值范围，则不会定义运算结果，并且运算结果与平台息息相关。例如，当负运算数值从 LREAL 转换为目标数据类型 UINT 时，就会出现这种情况。也可能会出现异常错误！

信息可能会丢失

如果将较大的数据类型转换为较小的数据类型，可能会导致信息丢失。

转换为 STRING 或 WSTRING 时的字符串操作

如果将类型转换为 STRING 或 WSTRING，将以左对齐的字符串存储类型值，如果该字符串过长，则会将其截断。因此，请声明足够长的返回变量，以便类型转换操作符 <type>_TO_STRING 和 <type>_TO_WSTRING 能够容纳字符串而无需进行截断。

- 重载 [▶ 660]
- 整数转换 [▶ 663]
- Boolean 转换 [▶ 661]
- --- FEHLENDER LINK ---
- --- FEHLENDER LINK ---
- 浮点数转换 [▶ 664]
- 字符串转换 [▶ 665]
- --- FEHLENDER LINK ---
- 日期和时间转换 [▶ 669]
- TRUNC [▶ 671]
- TRUNC_INT [▶ 671]

比较运算符

比较运算符是 Boolean 运算符，用于比较 2 个输入（第 1 个操作数和第 2 个操作数）。

- EQ [▶ 672]
- GE [▶ 672]
- GT [▶ 672]
- LE [▶ 673]
- LT [▶ 673]
- NE [▶ 673]

其他运算符

- __DELETE [▶ 676]
- __ISVALIDREF [▶ 677]
- __NEW [▶ 674]
- __QUERYINTERFACE [▶ 677]
- __QUERYPOINTER [▶ 678]
- __VARINFO [▶ 683]
- __POUNAME [▶ 685]
- __POSITION [▶ 685]

16.3.1 地址运算符

16.3.1.1 ADR

该运算符是 IEC 61131-3 标准的扩展。

ADR 以 PVOID、DWORD（32 位系统）或 LWORD（32 位和 64 位系统）（取决于运行时系统）的形式提供其参数的地址。为了确保独立于运行时系统架构，建议使用 PVOID 作为数据类型。

⚠ 谨慎

通过在线更改改变地址内容

如果您使用在线更改，则可能会改变地址内容。因此，POINTER 变量可能会指向 1 个无效的内存区。为了避免出现问题，请确保 TwinCAT 在发生在线更改时会更新指针的值。

语法：

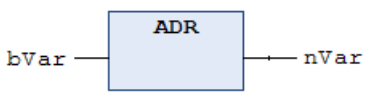
```
VAR
    <address name> : PVOID | DWORD | LWORD | __XWORD | POINTER TO < basis data type>;
END_VAR
<address name> := ADR(<variable name>);
```

示例：

ST：

```
nVar := ADR(bVAR);
```

FBD：



使用不同的数据类型的示例：

```
FUNCTION_BLOCK FB_Address
VAR
    nVar      : INT := 10;
    pNumber   : POINTER TO INT;
    nAddress1 : PVOID;
    nAddress2 : DWORD;
```

```
    nAddress3 : LWORD;
    nAddress4 : __XWORD;
END_VAR
```

```
pNumber   := ADR(nVar);   // pNumber wird der Adresse von nVar zugewiesen
nAddress1 := ADR(nVar);   // PVOID   : für 32- und 64-Bit-Systeme
nAddress2 := ADR(nVar);   // DWORD   : nur für 32-Bit-Systeme
nAddress3 := ADR(nVar);   // LWORD   : für 32- und 64-Bit-Systeme
nAddress4 := ADR(nVar);   // __XWORD : für 32- und 64-Bit-Systeme
```

另请参见：

- 指针 [▶ 708]
- 特殊数据类型 XINT、UXINT、XWORD 和 PVOID [▶ 708]

16.3.1.2 内容运算符

该运算符是 IEC 61131-3 标准的扩展。

该运算符允许解除引用指针。在指针标识符上附加运算符 ^。

⚠ 谨慎

通过在线更改改变地址内容

如果您使用在线更改，则可能会改变地址内容。

示例：

ST：

```
pSample : POINTER TO INT;
nInt1   : INT;
nInt2   : INT;
pSample := ADR(nInt1);
nInt2   := pSample^;
```

16.3.1.3 BITADR

该运算符是 IEC 61131-3 标准的扩展。

BITADR 以 DWORD 形式返回段内的位偏移。

该 DWORD 中的最高值半字节（4 位）描述了内存区：

标志：16x40000000

输入：16x80000000

输出：16xC0000000

⚠ 谨慎

通过在线更改改变地址内容

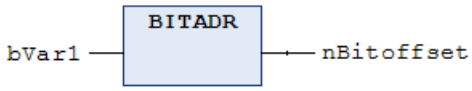
如果您使用在线更改，则可能会改变地址内容。

示例：

ST：

```
VAR
    bVar1 AT %IX2.3 : BOOL;
    nBitoffset : DWORD;
END_VAR

nBitoffset := BITADR(bVar1); (*Result if byte addressing=TRUE: 16x80000013, if byte addressing = FAL
SE : 16x80000023*)
```

FBD：



16.3.2 算术运算符

16.3.2.1 ADD

该 IEC 运算符进行变量的加法运算。

允许的数据类型：__UXINT、__XINT、__XWORD、BYTE、DATE、DATE_AND_TIME、DINT、DT、DWORD、INT、LDATE、LDATE_AND_TIME、LDT、LINT、LREAL、LTIME、LTOD、LWORD、REAL、SINT、TIME、TIME_OF_DAY、TOD、UDINT、UINT、ULINT、USINT、WORD

时间数据类型的可能组合：

- TIME + TIME = TIME
- TIME + LTIME = LTIME
- LTIME + LTIME = LTIME

日期和时间数据类型的可能组合：

- TOD + TIME = TOD
- DT + TIME = DT
- TOD + LTIME = LTOD
- DT + LTIME = LDT
- LTOD + TIME = LTOD
- LDT + LTIME = LDT
- LTOD + LTIME = LTOD
- LDT + LTIME = LDT

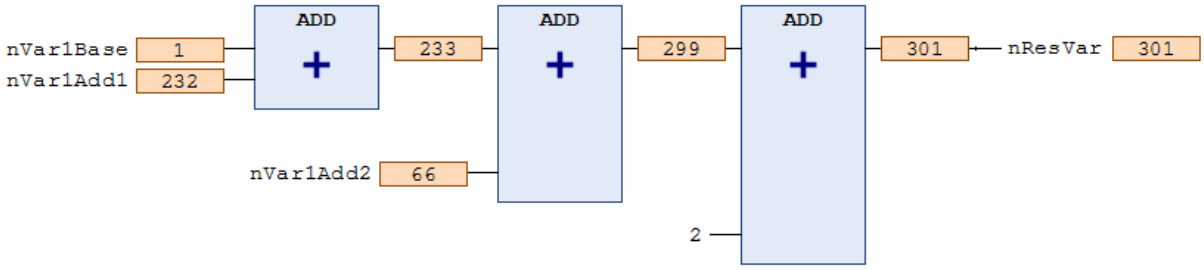
FBD/LD 编辑器中的特殊功能：通过功能块输入端，您可以扩展 ADD 运算符。附加功能块输入端的数量有限。

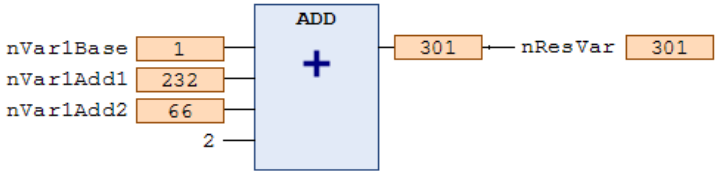
示例：

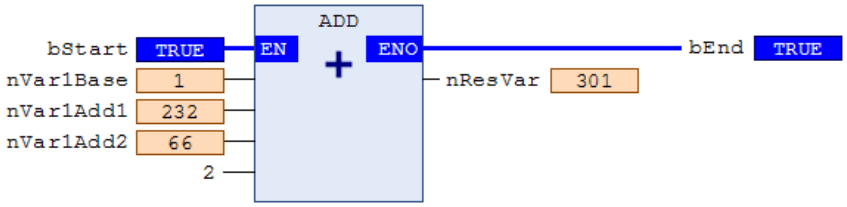
ST：

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

FBD：







16.3.2.2 SUB

该 IEC 运算符进行变量的减法运算。

允许的数据类型：BYTE、WORD、DWORD、LWORD、SINT、USINT、INT、UINT、DINT、UDINT、LINT、ULINT、REAL、LREAL、TIME、TIME_OF_DAY、TOD、LTIME、LTIME_OF_DAY、LTOD、DATE、DATE_AND_TIME、DT、LDATE、LDATE_AND_TIME、LDT

时间数据类型的可能组合：

- TIME - TIME = TIME
- LTIME - LTIME = LTIME

日期和时间数据类型的可能组合：

- DATE - DATE = TIME
- LDATE - LDATE = LTIME
- TOD - TIME = TOD
- LTOD - LTIME = LTOD
- TOD - TOD = TIME
- LTOD - LTOD = LTIME
- DT - TIME = DT
- LDT - LTIME = LDT
- DT - DT = TIME
- LDT - LDT = LTIME

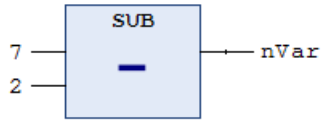
未定义负 TIME/LTIME 值。

示例：

ST：

```
nVar := 7-2;
```

FBD：



16.3.2.3 MUL

该 IEC 运算符用于变量的乘法运算。

允许的数据类型：BYTE、WORD、DWORD、LWORD、SINT、USINT、INT、UINT、DINT、UDINT、LINT、ULINT、REAL、LREAL、TIME

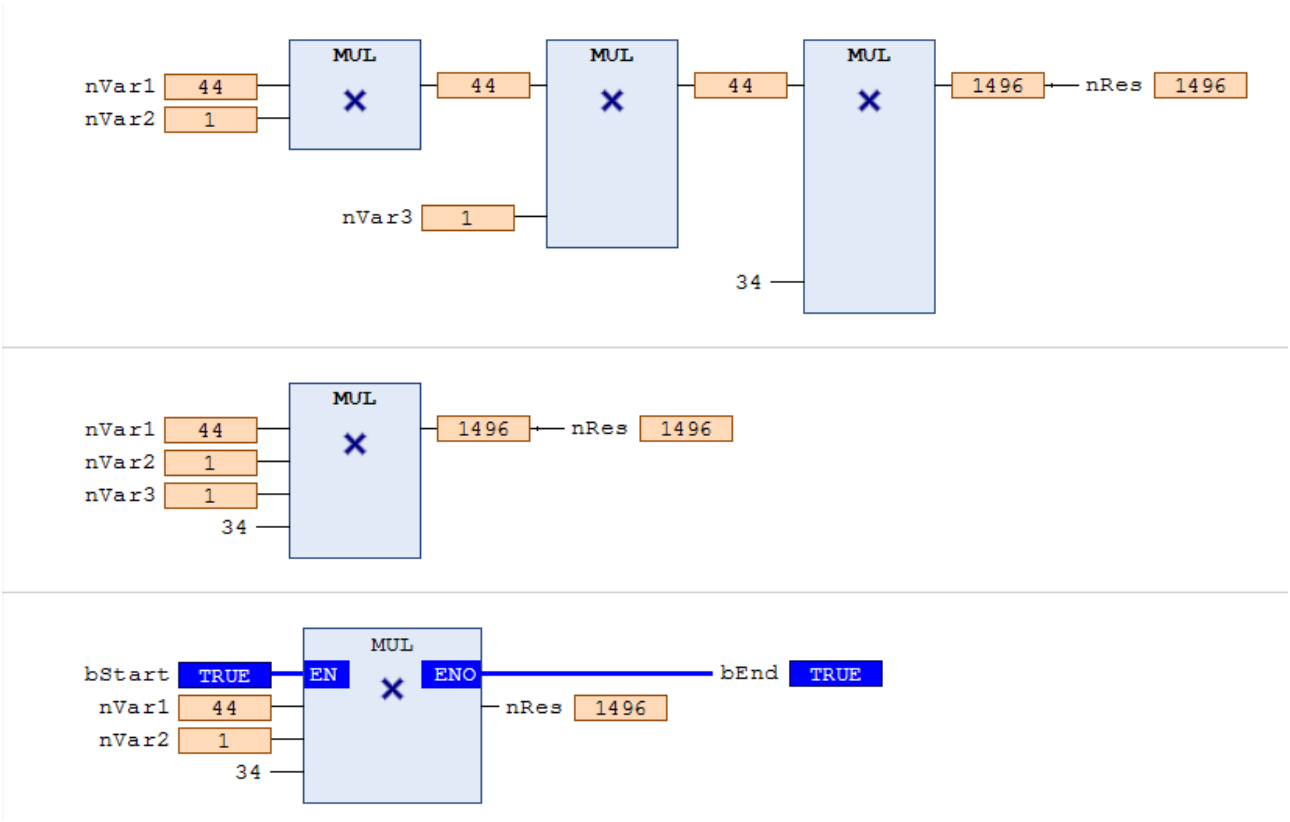
FBD/LD 编辑器中的特殊功能：通过附加的功能块输入端，您可以扩展 MUL 运算符。附加功能块输入端的数量有限。

示例：

ST：

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

FBD：



16.3.2.4 DIV

该 IEC 运算符用于变量的除法运算。

允许的数据类型：BYTE、WORD、DWORD、LWORD、SINT、USINT、INT、UINT、DINT、UDINT、LINT、ULINT、REAL、LREAL、TIME

在 TwinCAT 中，除以 0 总是会导致异常，而且，相应的任务会停止。

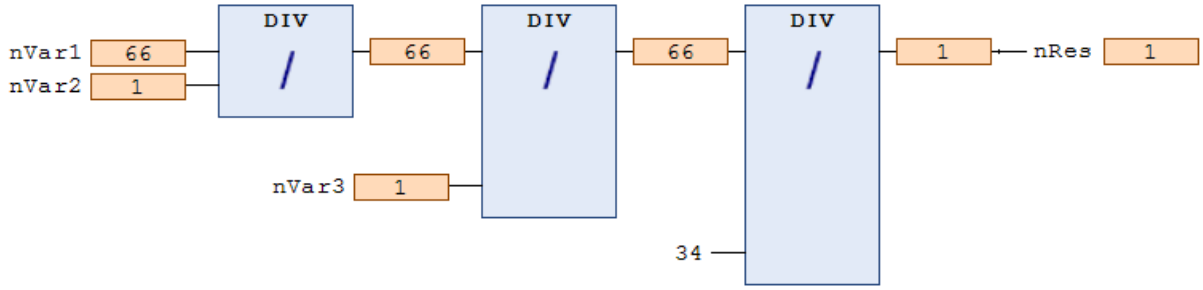
示例：

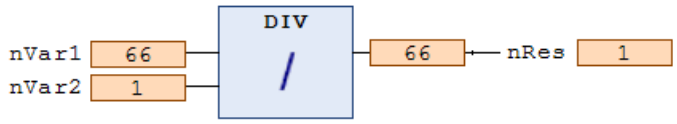
ST：

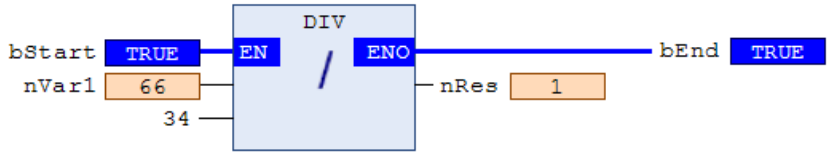
```
nVar := 8/2;
```

FBD：

1. DIV 功能块系列，2. 单个 DIV 功能块，3. 带有 EN/ENO 参数的 DIV 功能块







请注意，在运行时期间，您可以使用隐式监控函数 CheckDivInt、CheckDivLint、CheckDivLReal 和 CheckDivLReal 监控除数为 0 的可能性。

另请参见：

- 除法检查（POU：CheckDivDInt、CheckDivLInt、CheckDivReal、CheckDivLReal） [▶ 155]
-
-
-

16.3.2.5 MOD

该 IEC 运算符用于模除运算。

函数的结果是除法的整数余数。

允许的数据类型：BYTE、WORD、DWORD、LWORD、SINT、USINT、INT、UINT、DINT、UDINT、LINT、ULINT

在 TwinCAT 中，除以 0 总是会导致异常，而且，相应的任务会停止。
不过，Mod 0 的结果是 0。

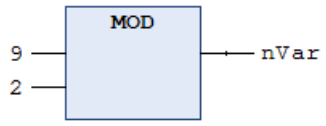
示例：

结果：nVar 为 1。

ST：

```
nVar := 9 MOD 2;
```

FBD：



16.3.2.6 MOVE

该 IEC 运算符用于将 1 个变量分配给另一个相应类型的变量。

由于 MOVE 在 CFC、FBD 和 LD 编辑器中可作为功能块使用，因此您可以将 EN/ENO 功能应用于此处的变量赋值。

示例：

结果：var2 接收到 var1 的值。

CFC 与 EN/ENO 功能组合使用：

如果 en_i 为 TRUE，则 TwinCAT 会将变量 var1 的值分配给变量 var2。

ST：

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

这对应于：

```
ivar2 := ivar1;
```

16.3.2.7 INDEXOF

该运算符是 IEC 61131-3 标准的扩展。

在 TwinCAT 中，可以使用 ADR 运算符代替 INDEXOF 运算符，以获取指向功能块索引的指针。

另请参见：

- ADR [▶ 639]

16.3.2.8 SIZEOF

该运算符是 IEC 61131-3 标准的扩展。

该运算符用于确定指定变量 x 所需的字节数。SIZEOF 运算符总是返回 1 个无符号值。返回变量的类型与检测到的变量 x 的大小相适应。

SIZEOF（x）的返回值	常量的数据类型，TwinCAT 将其隐式用于所找到的大小。
0 <= x 的大小 < 256	USINT
256 <= x 的大小 < 65536	UINT
65536 <= x 的大小 < 4294967296	UDINT
4294967296 <= x 的大小	ULINT

示例：

结果：nVar 为 10。

ST：

```
aArr1 : ARRAY[0..4] OF INT;
nVar  : INT;

nVar := SIZEOF(aArr1); (*nVar := USINT#10;*)
```

16.3.2.9 XSIZEOF

TC3.1 Build 4026 及以上可用

该运算符是 IEC 61131-3 标准的扩展。

XSIZEOF 运算符可确定在传递的变量或数据类型中所需的字节数。

总是返回 1 个无符号值。返回值 <return value> 的数据类型指定如下：在 64 位平台上，类型为 ULINT；在所有其他平台上，则为 UDINT。为了生成可在所有平台上运行的代码，可以使用 __UXINT 数据类型声明返回值。

语法：

```
<return value> := XSIZEOF( <variable> );
```

示例：

```
PROGRAM MAIN
VAR
    nReturnValue : __UXINT;             // Datentyp bei 64-bit-Plattformen: ULINT
    aData1       : ARRAY[0..4] OF INT;
END_VAR
```

```
nReturnValue := XSIZEOF(aData1);
```

结果：

```
nReturnValue = 10
```

在给 __UXINT 类型的变量赋值时，建议使用运算符 XSIZEOF，而不是运算符 SIZEOF [▶ 645]。这是因为在使用 XSIZEOF 时，返回值的数据类型取决于平台。因此，不会出现在使用 SIZEOF 运算符时所出现的问题。

16.3.3 调用运算符

16.3.3.1 CAL

该 IEC 运算符用于调用 1 个功能块。

CAL 调用 IL 中功能块的实例。

语法：

CAL <function block> (<input variable 1> := <value>, <input variable N> := <value>)

示例：

调用功能块实例 fbInst，将输入变量 nVar1、bVar2 赋值为 0 或 TRUE。

```
CAL fbInst(nVar1 := 0, bVar2 := TRUE)
```

16.3.4 选择运算符

16.3.4.1 LIMIT

该 IEC 选择运算符用于限制。

语法： OUT := LIMIT(Min, IN, Max)

意思：OUT := MIN(MAX (IN, Min), Max)

Max 是结果的上限，Min 是结果的下限。如果 IN 值超过上限 Max，则 LIMIT 会返回 Max。如果 IN 低于 Min，则结果为 Min。

IN 和 OUT 的允许数据类型：全部

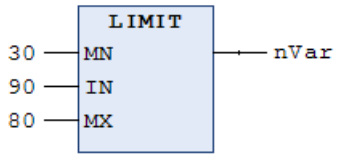
示例：

结果：nVar 为 80。

ST：

```
nVar := LIMIT(30,90,80);
```

FBD：



16.3.4.2 MAX

该 IEC 运算符用于最大值函数。它会返回传输的输入值中的最大值。

语法：OUT := MAX(IN0, IN1,<further inputs>)

允许的数据类型：全部

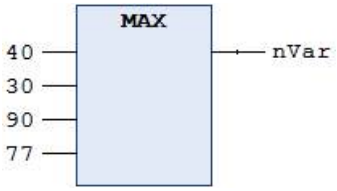
示例：

结果：nVar 为 90。

ST：

```
nVar := MAX(40,30,90,77);
```

FBD：



16.3.4.3 MIN

该 IEC 运算符用于最小值函数。它会返回传输的输入值中的最小值。

语法：OUT := MIN(IN0,IN1,<further inputs>)

允许的数据类型：全部

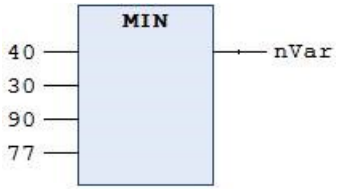
示例：

结果：nVar 为 30。

ST：

```
nVar := MIN(40,30,90,77);
```

FBD：



16.3.4.4 MUX

该 IEC 运算符用作多路复用器。

语法： OUT := MUX(K, IN0,...,INn)

这意味着 OUT = IN_K

K 的允许数据类型：BYTE、WORD、DWORD、LWORD、SINT、USINT、INT、UINT、DINT、LINT、ULINT、UDINT

IN0、……、INn 和 OUT：任何相同的数据类型。确保在所有 3 个位置都使用相同类型的变量，特别是在使用用户定义的数据类型时。编译器会检查类型相等的情况，并发布编译错误。特别是不支持将功能块实例分配给接口（变量）。

MUX 从 1 组值中选择第 K 个值。第 1 个值对应于 K=0。如果 K 大于附加输入端的数量（n），则 TwinCAT 会传递最后一个值（INn）。

为了实现运行时优化，TwinCAT 仅会计算 IN_K 前面的表达式。

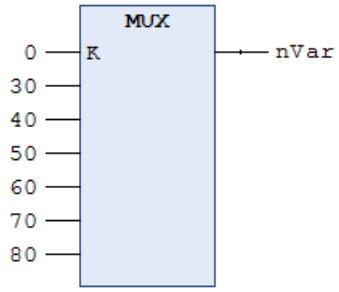
示例：

结果：nVar 为 30。

ST：

```
nVar := MUX(0,30,40,50,60,70,80);
```

FBD：



16.3.4.5 SEL

该 IEC 运算符用于按位选择。

语法：

OUT := SEL(G, IN0, IN1) 意思：

OUT := IN0；如果 G = FALSE

OUT := IN1；如果 G = TRUE

允许的数据类型：

IN0、……、INn 和 OUT：任何相同的数据类型。确保在所有 3 个位置都使用相同类型的变量，特别是在使用用户定义的数据类型时。编译器会检查类型相等的情况，并发布编译错误。特别是不支持将功能块实例分配给接口（变量）。

G：BOOL

如果 G = TRUE，则 TwinCAT 不会计算 IN0 前面的表达式。如果 G = FALSE，则 TwinCAT 不会计算 IN1 前面的表达式。

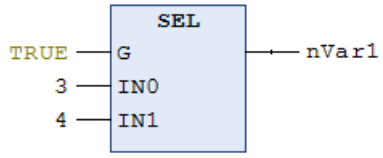
使用图形化编程语言，如果上游连接功能块、跳转、返回、分支或边缘检测，则 IN0 和 IN1 处的表达式计算与输入 G 无关。

示例：

ST：

```
nVar1 := SEL(TRUE,3,4); (*Result: 4*)
```

FBD：



16.3.5 位移运算符

16.3.5.1 SHL

该 IEC 运算符用于将操作数按位向左移动。

语法： erg := SHL (in, n)

in：向左移动的操作数。

n：向左移动的位数。

如果 n 超过数据类型的宽度，则 BYTE、WORD、DWORD 和 LWORD 操作数的填充方式取决于目标系统。目标系统会使用零或 n MOD <register width> 进行填充。

请注意，TwinCAT 在进行算术运算时考虑的位数由输入变量的数据类型决定。

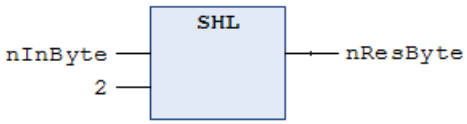
示例：

尽管输入变量 nInByte 和 nInWord 的值相同，但 nResByte 和 nResWord 的结果却不同，这是因为输入变量的数据类型不同。

ST：

```
PROGRAM Shl_st
VAR
    nInByte  :  BYTE:=16#45; (*2#01000101*)
    nInWord  :  WORD:=16#0045; (*2#0000000001000101*)
    nResByte :  BYTE;
    nResWord :  WORD;
    nVar : BYTE  := 2;
END_VAR

nResByte := SHL(nInByte,nVar); (*Result is 16#14, 2#00010100*)
nResWord := SHL(nInWord,nVar); (*Result is 16#0114, 2#0000000100010100*)
```

FBD：



16.3.5.2 SHR

该 IEC 运算符用于将操作数按位向右移动。

语法： erg := SHR (in, n)

in：向右移动的操作数。

n：向右移动的位数。

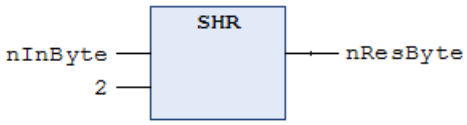
如果 n 超过数据类型的宽度，则 BYTE、WORD、DWORD 和 LWORD 操作数的填充方式取决于目标系统。目标系统会使用零或 n MOD <register width> 进行填充。

示例：

ST：

```
PROGRAM Shr_st
VAR
    nInByte  :  BYTE:=16#45; (*2#01000101*)
    nInWord  :  WORD:=16#0045; (*2#0000000001000101*)
    nResByte :  BYTE;
    nResWord :  WORD;
    nVar : BYTE := 2;
END_VAR

nResByte := SHR(nInByte,nVar); (*Result is 16#11, 2#00010001*)
nResWord := SHR(nInWord,nVar); (*Result is 16#0011, 2#0000000000010001*)
```

FBD：



16.3.5.3 ROL

该 IEC 运算符用于将操作数按位向左旋转。

允许的数据类型：BYTE、WORD、DWORD、LWORD

语法： erg := ROL (in, n)

TwinCAT 向左移动 n 次，每次移动 1 位，同时将最左侧位置的位添加到右侧。

TwinCAT 在进行计算时考虑的位数由输入变量的数据类型决定。如果这是常量，则 TwinCAT 会考虑最小可能的数据类型。输出变量的数据类型对算术运算没有影响。

示例：

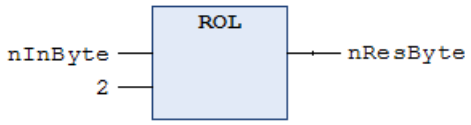
尽管输入变量 nInByte 和 nInWord 的值相同，但根据输入变量的数据类型，nResByte 和 nResWord 的结果却不同。

ST：

```
PROGRAM Rol_st
VAR
    nInByte  : BYTE := 16#45;
    nInWord  : WORD := 16#45;
    nResByte : BYTE;
```

```
    nResWord : WORD;
    nVar : BYTE := 2;
END_VAR

nResByte := ROL(nInByte,nVar); (*Result: 16#15*)
nResWord := ROL(nInWord,nVar); (*Result: 16#0114*)
```

FBD：



16.3.5.4 ROR

该 IEC 运算符用于将操作数按位向右旋转。

允许的数据类型：BYTE、WORD、DWORD、LWORD

语法： erg := ROR (in, n)

TwinCAT 向右移动 n 次，每次移动 1 位，同时将最右侧位置的位添加到左侧。

TwinCAT 在进行计算时考虑的位数由输入变量的数据类型决定。如果这是常量，则 TwinCAT 会考虑最小可能的数据类型。输出变量的数据类型对算术运算没有影响。

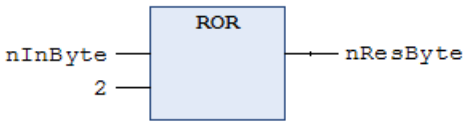
示例：

尽管输入变量 nInByte 和 nInWord 的值相同，但根据输入变量的数据类型，nResByte 和 nResWord 的结果却不同。

ST：

```
PROGRAM Ror_st
VAR
    nInByte  : BYTE:=16#45;
    nInWord  : WORD:=16#45;
    nResByte : BYTE;
    nResWord : WORD;
    nVar : BYTE :=2;
END_VAR

nResByte := ROR(nInByte,nVar); (*Result: 16#51*)
nResWord := ROR(nInWord,nVar); (*Result: 16#4011*)
```

FBD：



16.3.6 位串运算符

16.3.6.1 AND

该 IEC 运算符用于位操作数的按位与运算。

如果输入位为 1，则输出位为 1，否则为 0。

允许的数据类型：BOOL、BYTE、WORD、DWORD、LWORD

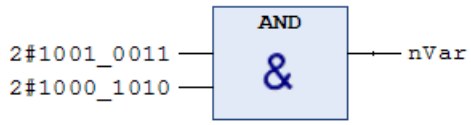
示例：

结果：nVar 为 2#1000_0010。

ST：

```
nVar := 2#1001_0011 AND 2#1000_1010
```

FBD：



16.3.6.2 AND_THEN

该运算符是 IEC 61131-3 标准的扩展。

运算符 AND_THEN 仅可在结构化文本编程中进行以下运算：对 BOOL 和 BIT 类型的操作数进行与运算，并进行短路求值。这意味着：

如果所有操作数均为 TRUE，则运算结果也为 TRUE，否则为 FALSE。

但是：只有当 AND_THEN 运算符的第 1 个操作数为 TRUE 时，TwinCAT 才会执行其他操作数的表达式。在 IF (ptr < > 0 AND_THEN ptr ^ = 99) THEN... 这样的条件下，这可以避免空指针的问题。

相反，如果使用 IEC 运算符 AND，则 TwinCAT 始终会对所有操作数进行求值。

另请参见：

- AND [▶ 651]

16.3.6.3 NOT

该 IEC 运算符用于位操作数的按位非运算。

如果相应的输入位为 0，则输出位为 1，反之亦然。

允许的数据类型：BOOL、BYTE、WORD、DWORD、LWORD

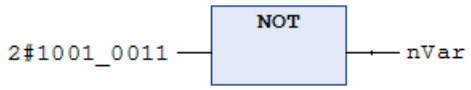
示例：

结果：nVar 为 2#0110_1100。

ST：

```
nVar := NOT 2#1001_0011
```

FBD：



16.3.6.4 OR

该 IEC 运算符用于位操作数的按位或运算。

如果至少 1 个输入位为 1，则输出位为 1，否则为 0。

允许的数据类型：BOOL、BYTE、WORD、DWORD、LWORD

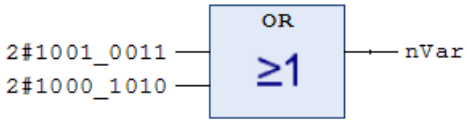
示例：

结果：nVar 为 2#1001_1011。

ST：

```
nVar := 2#1001_0011 OR 2#1000_1010
```

FBD：



16.3.6.5 OR_ELSE

该运算符是 IEC 61131-3 标准的扩展。

运算符 OR_ELSE 仅适用于结构化文本编程：对 BOOL 和 BIT 类型的操作数进行或运算，并进行短路求值。

这意味着：

如果至少 1 个操作数为 TRUE，则运算结果也为 TRUE，否则为 FALSE。

与使用 IEC 运算符 OR 相比，在使用 OR_ELSE 时，如果其中 1 个操作数的求值结果为 TRUE，则不再执行所有其他操作数的表达式。

示例：

```
Function_Block FB_Sample
VAR
    nCounter : INT;
END_VAR

METHOD TestMethod : BOOL
nCounter := nCounter + 1;
TestMethod := TRUE;

PROGRAM MAIN
VAR
    fbSampleOr : FB_Sample;
    fbSampleOrElse : FB_Sample;
    bResult : BOOL;
    bVar : BOOL;
END_VAR

bResult : bVar OR fbSampleOr.TestMethod();
// Counter of fbSampleOr increases as the method is executed

bResult := bVar OR_ELSE fbSampleOrElse.TestMethod();
//Counter of fbSampleOrElse does not increases as the method is not executed
```

bVar 为 TRUE，因此，在使用 OR 和 OR_ELSE 时，结果 bResult 也为 TRUE。

这 2 个逻辑运算符之间的区别在于，只有在使用 OR 时才会执行第 2 个操作数（方法的调用）。在这种情况下，功能块实例 fbSampleOr 的计数器将会递增。

由于第 1 个操作数 bVar1 产生的结果已为 TRUE，所以，在使用 OR_ELSE 时不再执行第 2 个操作数，因此不会调用功能块实例 fbSampleOrElse 的方法，计数器也不会递增。

另请参见：

- OR [▶ 652]

16.3.6.6 XOR

该 IEC 运算符用于位操作数的按位异或运算。

如果 2 个输入位中只有 1 个返回值为 1，则输出位被设置为 1。如果 2 个输入端都为 1 或都为 0，则输出端变为 0。

允许的数据类型：BOOL、BYTE、WORD、DWORD、LWORD

请注意扩展形式的异或功能块的以下行为，即，如果存在 2 个以上输入端：TwinCAT 会成对地比较输入端，然后比较各自的结果（这符合标准，但不一定符合期望）。

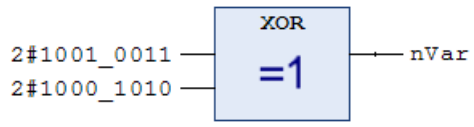
示例：

结果：nVar 为 2#0001_1001。

ST：

```
nVar := 2#1001_0011 XOR 2#1000_1010
```

FBD：



16.3.7 命名空间运算符

命名空间运算符是 IEC 61131-3 运算符的扩展。即使您在项目中多次使用相同的变量或模块名称，它们也提供了使对变量或模块的访问具有唯一性的选项。

16.3.7.1 全局命名空间

该运算符是 IEC 61131-3 标准的扩展。

以点 . 开头的实例路径总是会打开 1 个全局命名空间。因此，如果存在 1 个与全局变量同名的局部变量 <varname>，则会将 <varname> 用于寻址全局变量。

16.3.7.2 全局变量列表的命名空间

该运算符是 IEC 61131-3 标准的扩展。

您可以使用全局变量列表（GVL）的名称作为在列表中定义的变量的命名空间标识符。这样可以在不同的全局变量列表中使用具有相同名称的变量，并且仍然可以对特定变量进行唯一访问。在变量名前应加上全局变量列表的名称，中间用点隔开。

语法：<global variable list name>.<variable>

示例：

```
GVL1.nVar := GVL2.nVar;
```

全局变量列表 GVL1 和 GVL2 包含 1 个变量 nVar。TwinCAT 将全局变量 nVar 从列表 GVL2 复制到列表 GVL1 中的 nVar 处。

如果您引用的变量已在多个全局变量列表中进行声明，而该变量前面没有列表名称，则会出现错误消息。

16.3.7.3 库命名空间

该运算符是 IEC 61131-3 标准的扩展。

语法：

<library namespace>.<library function block identifier>

库功能块标识符由库命名空间（前缀以点作为分隔符）扩展，以便以明确的限定方式访问库功能块。通常，命名空间和库名是相同的。

示例：

在项目中集成 1 个库，其中包含 1 个功能块 FB_Sample。但是，1 个具有相同名称的功能块已在项目中进行本地实例化。使用 libA.fbSample 作为库块的名称，以便访问库块，而不是本地功能块。

```
nVar1 := fbSample(nIn := 12); // Call of the project function block FB_Sample
nVar2 := lib.fbSample(nIn := 22); // Call of the library function block FB_Sample
```

您可以为命名空间定义不同的标识符。为此，作为库开发人员，您在创建库项目时可以在项目信息中输入命名空间。或者，您也可以在 **Properties**（属性）视图中的库管理器中为库创建 PLC 项目时指定特殊的命名空间。

16.3.7.4 枚举命名空间

该运算符是 IEC 61131-3 标准的扩展。

您可以使用枚举的 TYPE 名称来明确访问枚举常量。这样，您可以在多个枚举中使用具有相同名称的常量。

枚举名称的前缀为常量名称，以点分隔。

语法：< enumeration name>.<constant name>

示例：

常量 eBlue 是 E_Colors 枚举和 E_Feelings 枚举的组件。

```
eColor := E_Colors.eBlue;  // Access to coomponent Blue in enumeration Colors
eFeeling := E_Feelings.eBlue; // Acces to component Blue in enumeration Feeling
```

16.3.8 数字运算符

16.3.8.1 ABS

该 IEC 运算符会返回 1 个数字的绝对值。

输入和输出变量及数字常量的允许数据类型：任何数字基本数据类型

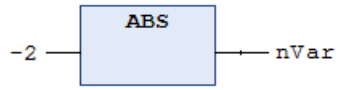
示例：

结果：nVar 为 2。

ST：

```
nVar := ABS(-2);
```

FBD：



16.3.8.2 ACOS

该 IEC 运算符会返回 1 个数字的反余弦值（余弦值的倒数）。该值以弧度为单位进行计算。

以弧度为单位指定角度的输入变量的允许数据类型：任何数字基本数据类型

输出变量的允许数据类型：REAL 和 LREAL

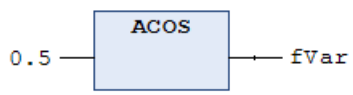
示例：

结果：fVar 为 1.0472。

ST：

```
fVar := ACOS(0.5);
```

FBD：



16.3.8.3 ASIN

该 IEC 运算符会返回 1 个数字的反正弦值（正弦值的倒数）。

输入变量的允许数据类型：任何数字基本数据类型

输出变量的允许数据类型：REAL 和 LREAL

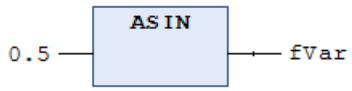
示例：

结果：fVar 为 0.523599。

ST：

```
fVar := ASIN(0.5);
```

FBD：



16.3.8.4 ATAN

该 IEC 运算符会返回 1 个数字的反正切值（正切值的倒数）。该值以弧度为单位进行计算。

以弧度为单位指定角度的输入变量的允许数据类型：任何数字基本数据类型

输出变量的允许数据类型：REAL 和 LREAL

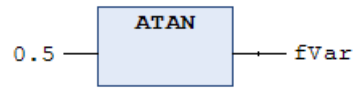
示例：

结果：fVar 为 0.463648。

ST：

```
fVar := ATAN(0.5);
```

FBD：



16.3.8.5 COS

该 IEC 运算符会返回 1 个数字的余弦值。

以弧度为单位指定角度的输入变量的允许数据类型：任何数字基本数据类型

输出变量的允许数据类型：REAL 和 LREAL

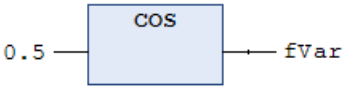
输入值的允许范围为 -2^{63} 至 $+2^{63}$。
在 x86 和 x64 系统上，如果输入值超出范围，则函数会返回输入值。

示例：

ST：

```
fVar := COS(0.5);
```

FBD：



结果：fVar 为 0.877583。

16.3.8.6 EXP

该 IEC 运算符会返回指数函数。

输入变量的允许数据类型：任何数字基本数据类型

输出变量的允许数据类型：REAL 和 LREAL

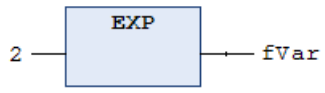
示例：

结果：fVar 为 7.389056099。

ST：

```
fVar := EXP(2);
```

FBD：



16.3.8.7 EXPT

该 IEC 运算符将 1 个数字与另一个数字进行指数运算，并返回底数的指数幂：power = base^{exponent}。底数和指数都是输入值（参数）。如果底数为 0 且指数同时为负数，则幂函数没有定义。不过，这种情况下的行为取决于平台。

语法：EXPT(<base>,<exponent>)

输入值的允许数据类型：基本数值数据类型（SINT、USINT、INT、UINT、DINT、UDINT、LINT、ULINT、REAL、LREAL、BYTE、WORD、DWORD、LWORD）

返回值的允许数据类型：浮点数类型（REAL、LREAL）。

如果结果存储在 REAL 变量中，则会为从 LREAL 到 REAL 的转换生成警告。--> 返回类型总是 LREAL。

例如，为以下 PLC 代码生成警告：

real1 := EXPT(real2,INT#2);

示例：

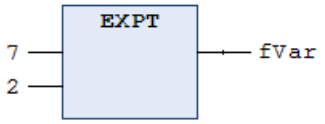
a）带有字面量的幂函数

结果：fVar 为 49。

ST：

```
fVar := EXPT(7,2);
```

FBD：



b）带有变量的幂函数

结果：fPow 为 128。

```
PROGRAM MAIN
VAR
    fPow      : LREAL;
    nBase     : INT := 2;
    nExponent : INT := 7;
END_VAR

fPow := EXPT(nBase, nExponent);
```

16.3.8.8 LN

该 IEC 运算符会返回 1 个数字的自然对数。

输入变量的允许数据类型：任何数字基本数据类型

输出变量的允许数据类型：REAL 和 LREAL

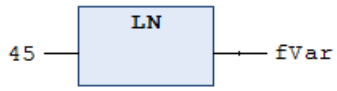
示例：

结果：fVar 为 3.80666。

ST：

```
fVar := LN(45);
```

FBD：



16.3.8.9 LOG

该 IEC 运算符会返回 1 个数字的以 10 为底的对数。

输入变量的允许数据类型：任何数字基本数据类型

输出变量的允许数据类型：REAL 或 LREAL

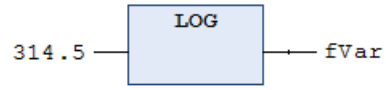
示例：

结果：fVar 为 2.49762。

ST：

```
fVar := LOG(314.5);
```

FBD：



16.3.8.10 SIN

该 IEC 运算符会返回 1 个数字的正弦值。

以弧度为单位指定角度的输入变量的允许数据类型：任何数字基本数据类型

输出变量的允许数据类型：REAL 和 LREAL

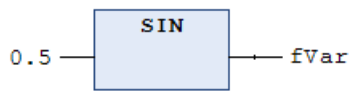
输入值的允许范围为 -2^{63} 至 $+2^{63}$。
在 x86 和 x64 系统上，如果输入值超出范围，则函数会返回输入值。

示例：

ST：

```
fVar := SIN(0.5);
```

FBD：



结果：fVar 为 0.479426。

16.3.8.11 SQRT

该 IEC 运算符会返回 1 个数字的平方根。

输入变量的允许数据类型：任何数字基本数据类型

输出变量的允许数据类型：REAL 或 LREAL

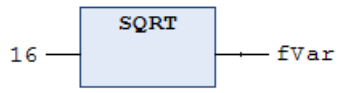
示例：

结果：fVar 为 4。

ST：

```
fVar := SQRT(16);
```

FBD：



16.3.8.12 TAN

该 IEC 运算符会返回 1 个数字的正切值。

以弧度为单位指定角度的输入变量的允许数据类型：任何数字基本数据类型

输出变量的允许数据类型：REAL 和 LREAL

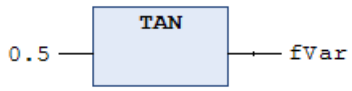
示例：

结果：fVar 为 0.546302。

ST：

```
fVar := TAN(0.5);
```

FBD：



16.3.9 类型转换运算符

类型转换有隐式类型转换和显式类型转换之分。

隐式类型转换

从“较小”类型到“较大”类型的转换，例如从 BYTE 转换为 INT 或从 WORD 转换为 DINT，可以显式进行，但也可以隐式进行，无需调用转换运算符。

显式类型转换

可以显式调用类型转换运算符。对于从一种基本类型到另一种基本类型的类型转换以及重载，可以使用下述类型转换运算符。

类型转换：

```
<elementary data type>_TO_<another elementary data type>
```

重载转换：

```
TO_<elementary data type>
```

基本数据类型：

```
<elementary data type> =
__UXINT | __XINT | __XWORD | BIT | BOOL | BYTE | DATE | DINT | DT | DWORD | INT | LDATE | LDT | LINT
| LREAL | LTIME | LTOD | LWORD | REAL | SINT | TIME | TOD | UDINT | UINT | ULINT | USINT | WORD
```

关键字 TIME_OF_DAY 和 DATE_AND_TIME 是数据类型 TOD 和 DT 的另一种拼写形式。

- TIME_OF_DAY = TOD
- DATE_AND_TIME = DT

TIME_OF_DAY 和 DATE_AND_TIME 没有被映射为类型转换命令。请改用数据类型 TOD 和 DT。

如果超出值范围，则不会定义结果

如果类型转换运算符的输入值超出输出数据类型的值范围，则不会定义运算结果，并且运算结果与平台息息相关。例如，当负运算数值从 LREAL 转换为目标数据类型 UINT 时，就会出现这种情况。也可能会出现异常错误！

信息可能会丢失

如果将较大的数据类型转换为较小的数据类型，可能会导致信息丢失。

转换为 STRING 或 WSTRING 时的字符串操作

如果将类型转换为 STRING 或 WSTRING，将以左对齐的字符串存储类型值，如果该字符串过长，则会将其截断。因此，请声明足够长的返回变量，以便类型转换操作符 <type>_TO_STRING 和 <type>_TO_WSTRING 能够容纳字符串而无需进行截断。

另请参见：

- 数据类型 [▶ 697]

16.3.9.1 重载

如果超出值范围，则不会定义结果

如果类型转换运算符的输入值超出输出数据类型的值范围，则不会定义运算结果，并且运算结果与平台息息相关。例如，当负运算数值从 LREAL 转换为目标数据类型 UINT 时，就会出现这种情况。也可能会出现异常错误！

信息可能会丢失

如果将较大的数据类型转换为较小的数据类型，可能会导致信息丢失。

舍入逻辑

边界情况的舍入逻辑取决于目标系统或目标系统的 FPU（浮点单元）。例如，-1.5 这样的值在不同的控制器上可能会有不同的转换方式。

通过应用程序拦截值范围溢出，以编写与目标系统无关的代码。

该运算符将值转换为其他数据类型，仅明确指定目标数据类型，而不指定初始数据类型（运算数的数据类型）（“重载转换”）。

类型转换的规则也适用于重载。

超载不属于 IEC 61131-3 的一部分。如果您想要严格按照 IEC61131-3 进行编程，请使用下文所述的 <type>_TO_ <another type> 方案的运算符。

语法：

```
<Variablenname> := <TO Operator> ( <Operand> );
<Operand> = <Variablenname> | <Literal>
```

运算符：

```
TO___UXINT
TO___XINT
TO___XWORD
TO_BIT
TO_BYTE
TO_BOOL
TO_DATE
TO_DINT
TO_DT
TO_DWORD
TO_INT
TO_LDATE
TO_LDT
TO_LINT
TO_LREAL
TO_LTIME
TO_LTOD
TO_LWORD
TO_REAL
TO_SINT
TO_STRING
TO_TIME
TO_TOD
TO_UDINT
TO_UINT
TO_ULINT
TO_USINT
TO_WORD
TO_WSTRING
```

示例：

实现语言 ST：

```
VAR
    nNumber1 : INT;
    nNumber2 : INT;
    fNumber3 : REAL     := 123.456;
    bTruth   : BOOL;
    sText1   : STRING;
    sText2   : STRING   := 'Hello World!';
    wsText   : WSTRING;
    dEvent   : DATE     := D#2019-9-3;
    nEvent   : UINT;
    nData    : __UXINT;
END_VAR
```

```
nNumber1 := TO_INT(4.22);          // Result: 4
nNumber2 := TO_INT(fNumber1);      // Result: 123
bTruth   := TO_BOOL(1);            // Result: TRUE
sText1   := TO_STRING(342);        // Result: '342'
wsText   := TO_WSTRING(sText2);    // Result: "Hello World!"
nEvent   := TO_UINT(dEvent);       // Result: 44288
dEvent   := TO___UXINT(nNumber2);  // Result: 123
```

另请参见：

- 数据类型 [▶ 697]

16.3.9.2 Boolean 转换

如果超出值范围，则不会定义结果

如果类型转换运算符的输入值超出输出数据类型的值范围，则不会定义运算结果，并且运算结果与平台息息相关。例如，当负运算数值从 LREAL 转换为目标数据类型 UINT 时，就会出现这种情况。也可能会出现异常错误！

信息可能会丢失

如果将较大的数据类型转换为较小的数据类型，可能会导致信息丢失。

转换为 STRING 或 WSTRING 时的字符串操作

如果将类型转换为 STRING 或 WSTRING，将以左对齐的字符串存储类型值，如果该字符串过长，则会将其截断。因此，请声明足够长的返回变量，以便类型转换操作符 <type>_TO_STRING 和 <type>_TO_WSTRING 能够容纳字符串而无需进行截断。

<type>_TO_BOOL

该运算符用于将其他数据类型转换为 BOOL 数据类型。

您可以指定任意变量或字面数值作为运算数。

语法：

```
<Datentyp>_TO_BOOL (<Variablenname> | <Literal>)
```

如果运算数不等于 0，则结果为 TRUE。如果运算数等于 0，则结果为 FALSE。
对于数据类型 STRING，如果运算数值为“TRUE”或“true”，则结果为 TRUE。否则，即使运算数值为“True”，也会返回结果“FALSE”。

示例：

ST 代码	结果
`bVar := BYTE_TO_BOOL(2#11010101);`	TRUE
`bVar := INT_TO_BOOL(0);`	FALSE
`bVar := TIME_TO_BOOL(T#5ms);`	TRUE
`bVar := STRING_TO_BOOL('TRUE');`	TRUE
`bVar := STRING_TO_BOOL('true');`	TRUE
`bVar := STRING_TO_BOOL('True');`	FALSE

BOOL_TO_<type>

该运算符用于将 BOOL 数据类型转换为其他数据类型。

您可以指定任意变量或字面数值作为运算数。

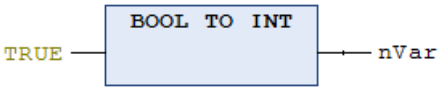
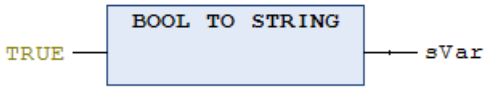
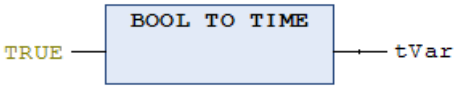
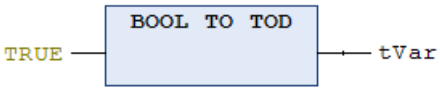
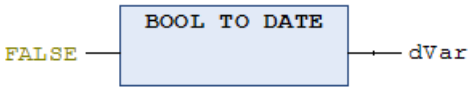
语法：

```
BOOL_TO_<Datentyp> (<Variablenname> | <Literal>)
```

对于数值数据类型，如果运算数为 TRUE，则结果为 1；如果运算数为 FALSE，则结果为 0。对于数据类型 STRING，结果为“TRUE”或“FALSE”。

示例：

ST 代码	结果
`nVar := BOOL_TO_INT(TRUE);`	1
`sVar := BOOL_TO_STRING(TRUE);`	'TRUE'
`tVar := BOOL_TO_TIME(TRUE);`	T#1ms
`tVar := BOOL_TO_TOD(TRUE);`	TOD#00:00:00.001
`dVar := BOOL_TO_DATE(FALSE);`	D#1970
`dtVar := BOOL_TO_DT(TRUE);`	DT#1970-01-01-00:00:01

FBD 代码	结果
	1
	'TRUE'
	T#1ms
	TOD#00:00:00.001
	D#1970-1-1

另请参见：

- 数据类型 > BOOL [▶ 698]

16.3.9.3 整数转换

如果超出值范围，则不会定义结果

如果类型转换运算符的输入值超出输出数据类型的值范围，则不会定义运算结果，并且运算结果与平台息息相关。例如，当负运算数值从 LREAL 转换为目标数据类型 UINT 时，就会出现这种情况。也可能会出现异常错误！

信息可能会丢失

如果将较大的数据类型转换为较小的数据类型，可能会导致信息丢失。

转换为 STRING 或 WSTRING 时的字符串操作

如果将类型转换为 STRING 或 WSTRING，将以左对齐的字符串存储类型值，如果该字符串过长，则会将其截断。因此，请声明足够长的返回变量，以便类型转换操作符 <type>_TO_STRING 和 <type>_TO_WSTRING 能够容纳字符串而无需进行截断。

这些操作符可以进行以下转换：

- 将整数数据类型转换为指定的数据类型，
- 将其他数据类型转换为整数数据类型，
- 或将整数数据类型转换为其他整数数据类型。

您可以指定任意变量或字面数值作为运算数。

如果将较大的数据类型转换为较小的数据类型，高位（前端）字节会被截断。如果将较小的数据类型转换为较大的数据类型，高位字节将以 0 填补。

语法：

```
<INT-Datentyp>_TO_<Datentyp> (<Variablenname> | <Literal>)
<Datentyp>_TO_<INT-Datentyp> (<Variablenname> | <Literal>)
<INT-Datentyp>_TO_<INT-Datentyp> (<Variablenname> | <Literal>)
```

整数数据类型：

- BYTE
- WORD、DWORD、LWORD
- SINT、INT、DINT、LINT
- USINT、UINT、UDINT、ULINT

示例：

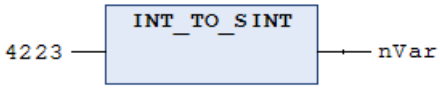
结果：nVar 为 127。

ST：

```
nVar := INT_TO_SINT(4223);
```

如果您将整数 4223（十六进制表示法为 16#107f）保存在 SINT 变量中，则该变量将包含数字 127（十六进制表示法为 16#7f）。

FBD：



另请参见：

- 数据类型 > 整数数据类型 [▶ 699]

16.3.9.4 浮点数转换

如果超出值范围，则不会定义结果

如果类型转换运算符的输入值超出输出数据类型的值范围，则不会定义运算结果，并且运算结果与平台息息相关。例如，当负运算数值从 LREAL 转换为目标数据类型 UINT 时，就会出现这种情况。也可能会出现异常错误！

信息可能会丢失

如果将较大的数据类型转换为较小的数据类型，可能会导致信息丢失。

如果浮点数在目标数据类型的值范围内，则转换在所有系统上均以相同的方式进行。

如果浮点数超出范围限值，则不考虑数字的前几个字节。

舍入逻辑

边界情况的舍入逻辑取决于目标系统或目标系统的 FPU（浮点单元）。例如，-1.5 这样的值在不同的控制器上可能会有不同的转换方式。

通过应用程序拦截值范围溢出，以编写与目标系统无关的代码。

<type>_TO_REAL/<type>_TO_LREAL

该运算符用于将其他数据类型转换为 REAL 或 LREAL 数据类型。

您可以指定任意变量或字面数值作为运算数。

如有必要，会在转换过程中进行四舍五入。

语法：

```
<Datentyp>_TO_REAL (<Variablenname> | <Literal>)
<Datentyp>_TO_LREAL (<Variablenname> | <Literal>)
```

REAL_TO_<type>/LREAL_TO_<type>

该运算符用于将 REAL 或 LREAL 数据类型转换为其他数据类型。

您可以指定任意变量或字面数值作为运算数。

语法：

```
REAL_TO_<Datentyp> (<Variablenname> | <Literal>)
LREAL_TO_<Datentyp> (<Variablenname> | <Literal>)
```

舍入：

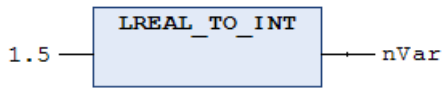
TwinCAT 会将运算数的浮点值向上或向下舍入为一个整数值，并将该值转换为相应的数据类型。将小数点后的数字 1 至 4 向下舍入，数字 5 至 9 向上舍入。此舍入不包括 STRING、BOOL、REAL 和 LREAL 等数据类型。

转换为字符串：

将浮点数转换为字符串时，尾数的小数位数限值为 6。如果数值 < 1，则尾数 m 适用于以下规定：1 <= m < 10。如果尾数的小数点后位数较多，则四舍五入到第 6 位，然后再进行转换。

此外，声明的返回值字符串变量可能过短。这种情况下，会在右侧截断返回字符串。

示例：

ST 代码	结果
`nVar1 := REAL_TO_INT(1.5);`	2
`nVar2 := REAL_TO_INT(1.4);`	1
`nVar1 := REAL_TO_INT(-1.5);`	-2
`nVar2 := REAL_TO_INT(-1.4);`	-1
FBD 代码	结果
	2

另请参见：

- 数据类型 > REAL/LREAL [▶ 700]

16.3.9.5 字符串转换

如果超出值范围，则不会定义结果

如果类型转换运算符的输入值超出输出数据类型的值范围，则不会定义运算结果，并且运算结果与平台息息相关。例如，当负运算数值从 LREAL 转换为目标数据类型 UINT 时，就会出现这种情况。也可能会出现异常错误！

信息可能会丢失

如果将较大的数据类型转换为较小的数据类型，可能会导致信息丢失。

转换为 STRING 或 WSTRING 时的字符串操作

如果将类型转换为 STRING 或 WSTRING，将以左对齐的字符串存储类型值，如果该字符串过长，则会将其截断。因此，请声明足够长的返回变量，以便类型转换操作符 <type>_TO_STRING 和 <type>_TO_WSTRING 能够容纳字符串而无需进行截断。

<type>_TO_STRING

该运算符用于将其他数据类型转换为 STRING 或 WSTRING 数据类型。

您可以指定任意变量或字面数值作为运算数。

语法：

```
<Datentyp>_TO_STRING (<Variablenname> | <Literal>)
<Datentyp>_TO_WSTRING (<Variablenname> | <Literal>)
```

STRING_TO_<type>

该运算符用于将 STRING 或 WSTRING 数据类型转换为其他数据类型。

您可以指定任意变量或字面数值作为运算数。

语法：

```
STRING_TO_<Datentyp> (<Variablenname> | <Literal>)
WTRING_TO_<Datentyp> (<Variablenname> | <Literal>)
```

您必须根据 IEC 61131-3 标准指定 STRING 类型的运算数。该值必须等于目标数据类型的有效常量（字面量）。它适用于指数值、无限值、前缀、分组字符（“_”）和逗号。在数字后面可以添加其他字符，例如，23xy。在数字前面不可以添加其他字符。

运算数必须代表目标数据类型的有效值。

将 STRING/WSTRING 转换为 BOOL

运算符：

```
STRING_TO_BOOL (<Variablenname> | <Literal>)
WSTRING_TO_BOOL (<Variablenname> | <Literal>)
```

只有当运算数值为“TRUE”或“true”时，才会返回 TRUE。否则，“True”会返回 FALSE。

示例：

ST 代码	结果
`bVar := STRING_TO_BOOL('TRUE');`	TRUE
`bVar := STRING_TO_BOOL('true');`	TRUE
`bVar := STRING_TO_BOOL('True');`	FALSE
`nVar := STRING_TO_WORD('abc34');`	0
`nVar := STRING_TO_WORD('34abc');`	34
`tVar := STRING_TO_TIME('T#127ms');`	T#127ms
`fVar := STRING_TO_REAL('1.234');`	1.234
`nVar := STRING_TO_BYTE('500');`	244

用于枚举变量的 TO_STRING/TO_WSTRING

如果您想要查询枚举组件的文本标识符，例如，为了在文本输出中进一步处理该标识符，请在枚举声明上方添加 属性“to_string” [▶ 774]。在实现部分，您可以将转换函数 TO_STRING 或 TO_WSTRING 应用于枚举变量或枚举的组件。然后会返回枚举组件的名称。

属性“to_string”

TC3.1 Build 4024 及以上版本可用

转换函数 TO_STRING/TO_WSTRING 也可应用于未使用属性“to_string”声明的枚举。在这种情况下，将会返回枚举组件的数值。

示例：

枚举 E_Sample

```
{attribute 'qualified_only'}
{attribute 'strict'}
{attribute 'to_string'}
TYPE E_Sample :
(
    eInit := 0,
    eStart,
    eStop
);
END_TYPE
```

MAIN 程序

```
PROGRAM MAIN
VAR
    eSample         : E_Sample;
    nCurrentValue   : INT;
    sCurrentValue   : STRING;
    wsCurrentValue  : WSTRING;

    sComponent      : STRING;
    wsComponent     : WSTRING;
END_VAR
```

```
nCurrentValue  := eSample;
sCurrentValue  := TO_STRING(eSample);
wsCurrentValue := TO_WSTRING(eSample);

sComponent     := TO_STRING(E_Sample.eStart);
wsComponent    := TO_WSTRING(E_Sample.eStop);
```

赋值/转换函数的结果：

- nCurrentValue 的值：0
- sCurrentValue 的值：’eInit’
- wsCurrentValue 的值：″eInit″
- sComponent 的值：’eStart’
- wsComponent 的值：″eStop″

如果没有使用属性“to_string”声明枚举，则会出现以下结果：

- nCurrentValue 的值：0
- sCurrentValue 的值：’0’
- wsCurrentValue 的值：″0″
- sComponent 的值：’1’
- wsComponent 的值：″2″

另请参见：

- 数据类型 > STRING [▶ 700]

16.3.9.6 时间转换

如果超出值范围，则不会定义结果

如果类型转换运算符的输入值超出输出数据类型的值范围，则不会定义运算结果，并且运算结果与平台息息相关。例如，当负运算数值从 LREAL 转换为目标数据类型 UINT 时，就会出现这种情况。也可能会出现异常错误！

信息可能会丢失

如果将较大的数据类型转换为较小的数据类型，可能会导致信息丢失。

转换为 STRING 或 WSTRING 时的字符串操作

如果将类型转换为 STRING 或 WSTRING，将以左对齐的字符串存储类型值，如果该字符串过长，则会将其截断。因此，请声明足够长的返回变量，以便类型转换操作符 <type>_TO_STRING 和 <type>_TO_WSTRING 能够容纳字符串而无需进行截断。

TIME_TO_<type>

该运算符用于将 TIME 或 LTIME 数据类型转换为其他数据类型。

您可以指定任意变量或字面数值作为运算数。

语法：

```
TIME_TO_<Datentyp> (<Variablenname> | <Literal>)
LTIME_TO_<Datentyp> (<Variablenname> | <Literal>)
```

将 TIME/LTIME 转换为 BOOL

运算符：

```
TIME_TO_BOOL (<Variablenname> | <Literal>)
LTIME_TO_BOOL (<Variablenname> | <Literal>)
```

当运算数值可解析为“0”时，运算符将返回 FALSE。

示例：

ST 代码	结果
`TIME_TO_BOOL(T#0MS);`	FALSE
`TIME_TO_BOOL(T#0NS);`	FALSE
`TIME_TO_BOOL(T#1MS);`	TRUE
`TIME_TO_BOOL(T#1NS);`	TRUE

将 TIME/LTIME 转换为 STRING/WSTRING

运算符：

```
TIME_TO_STRING (<Variablenname> | <Literal>)
LTIME_TO_STRING (<Variablenname> | <Literal>)
```

```
TIME_TO_WSTRING (<Variablenname> | <Literal>)
LTIME_TO_WSTRING (<Variablenname> | <Literal>)
```

示例：

ST 代码	结果
`TIME_TO_STRING(T#0MS);`	'T#0MS'
`LTIME_TO_WSTRING(T#0US);`	"T#0US"

16.3.9.7 日期和时间转换

如果超出值范围，则不会定义结果

如果类型转换运算符的输入值超出输出数据类型的值范围，则不会定义运算结果，并且运算结果与平台息息相关。例如，当负运算数值从 LREAL 转换为目标数据类型 UINT 时，就会出现这种情况。也可能会出现异常错误！

信息可能会丢失

如果将较大的数据类型转换为较小的数据类型，可能会导致信息丢失。

转换为 STRING 或 WSTRING 时的字符串操作

如果将类型转换为 STRING 或 WSTRING，将以左对齐的字符串存储类型值，如果该字符串过长，则会将其截断。因此，请声明足够长的返回变量，以便类型转换操作符 <type>_TO_STRING 和 <type>_TO_WSTRING 能够容纳字符串而无需进行截断。

TIME/TOD_TO_<type>

该运算符用于将日期和时间规格转换为其他数据类型。

您可以指定任意变量或字面数值作为运算数。

语法：

```
DATE_TO_<Datentyp> (<Variablenname> | <Literal>)
DT_TO_<Datentyp> (<Variablenname> | <Literal>)
TOD_TO_<Datentyp> (<Variablenname> | <Literal>)
```

```
LDATE_TO_<Datentyp> (<Variablenname> | <Literal>)
LDT_TO_<Datentyp> (<Variablenname> | <Literal>)
LTOD_TO_<Datentyp> (<Variablenname> | <Literal>)
```

DATE 和 DT 数据类型使用相同的内部存储格式，均以 DWORD 的形式存储。DATE 的分辨率为 1 天。DT 的分辨率为 1 秒。两者都从 1970 年 1 月 1 日开始。TOD 以 DWORD 形式存储，分辨率为 1 毫秒。

示例：

ST 代码	结果
`sVar := TIME_TO_STRING(T#12ms);`	'T#12ms'
`nVar := TIME_TO_DWORD(T#5m);`	300000
`nVar := TOD_TO_SINT(TOD#00:00:00.012);`	12

将日期和时间转换为 BOOL

运算符：

```
DATE_TO_BOOL (<Variablenname> | <Literal>)
DT_TO_BOOL (<Variablenname> | <Literal>)
TOD_TO_BOOL (<Variablenname> | <Literal>)
```

```
LDATE_TO_BOOL (<Variablenname> | <Literal>)
LDT_TO_BOOL (<Variablenname> | <Literal>)
LTOD_TO_BOOL (<Variablenname> | <Literal>)
```

当运算数值可解析为“0”时，运算符将返回 FALSE。

示例：

ST 代码	结果
DATE_TO_BOOL(D#1970-1-1);	FALSE
DT_TO_BOOL(DT#1970-1-1-0:0:0);	FALSE
TOD_TO_BOOL(TOD#0:0:0);	FALSE
DATE_TO_BOOL(D#2019-9-1);	TRUE
DT_TO_BOOL(DT#2019-9-1-12:0:0);	TRUE
TOD_TO_BOOL(TOD#12:0:0);	TRUE

将日期和时间转换为整数

运算符：

```
DATE_TO_<INT-Datentyp> (<Variablenname> | <Literal>)
DT_TO_<INT-Datentyp> (<Variablenname> | <Literal>)
TOD_TO_<INT-Datentyp> (<Variablenname> | <Literal>)
```

```
LDATE_TO_<INT-Datentyp> (<Variablenname> | <Literal>)
LDT_TO_<INT-Datentyp> (<Variablenname> | <Literal>)
LTOD_TO_<INT-Datentyp> (<Variablenname> | <Literal>)
```

DATE 和 DT 数据类型使用相同的内部存储格式，即 DWORD。DATE 的分辨率为 1 天。DT 的分辨率为 1 秒。两者都从 1970 年 1 月 1 日开始。

TOD 以 DWORD 形式存储，分辨率为 1 毫秒。

示例：

ST 代码	结果
DT_TO_DINT(DT#1970-1-1-0:0:0);	0
DATE_TO_DINT(D#1970-1-1);	0
TOD_TO_DINT(TOD#0:0:0);	0
DT_TO_DINT(DT#1970-1-1-0:0:1);	1
DATE_TO_DINT(D#1970-1-2);	86400
DT_TO_DINT(DT#2019-9-1-12:0:0.0);	1567339200
DATE_TO_DINT(D#2019-9-1);	1567339200
TOD_TO_DINT(TOD#12:0:0);	43200000

将日期和时间转换为 STRING/WSTRING

运算符：

```
DATE_TO_<STRING/WSTRING> (<Variablenname> | <Literal>)
DT_TO_<STRING/WSTRING> (<Variablenname> | <Literal>)
TOD_TO_<STRING/WSTRING> (<Variablenname> | <Literal>)
```

```
LDATE_TO_<STRING/WSTRING> (<Variablenname> | <Literal>)
LDT_TO_<STRING/WSTRING> (<Variablenname> | <Literal>)
LTOD_TO_<STRING/WSTRING> (<Variablenname> | <Literal>)
```

将 DATE、DATE_AND_TIME、TIME_OF_DAY、DT 或 TOD 类型的运算数传递给运算符进行日期和时间转换时，将按其常量符号（文字符号）进行转换。根据 IEC 61131-3 标准的规定，生成的字符串包含关键字 D#、DT# 或 TOD#，还包含其大小及其日期和时间单元。

示例：

ST 代码	结果
TIME_TO_STRING(T#12ms);	'T#12ms'
DT_TO_STRING(DT#1998-02-13-14:20);	'DT#1998-02-13-14:20:00'

另请参见：

- 数据类型 > 日期和时间数据类型 [▶ 702]

16.3.9.8 TRUNC

如果超出值范围，则不会定义结果

如果类型转换运算符的输入值超出输出数据类型的值范围，则不会定义运算结果，并且运算结果与平台息息相关。例如，当负运算数值从 LREAL 转换为目标数据类型 UINT 时，就会出现这种情况。也可能会出现异常错误！

信息可能会丢失

如果将较大的数据类型转换为较小的数据类型，可能会导致信息丢失。

该 IEC 运算符会将 REAL 数据类型转换为 DINT 数据类型。TwinCAT 仅使用数字的整数部分。

在 TwinCAT 2.x PLC Control 中，TRUNC 运算符会将 REAL 转换为 INT。如果您导入 TwinCAT 2.x PLC 项目，TwinCAT 会自动将 TRUNC 替换为 TRUNC_INT。

示例：

ST 代码	结果
`TRUNC(1.9)`	1
`TRUNC(-1.4)`	-1

16.3.9.9 TRUNC_INT

如果超出值范围，则不会定义结果

如果类型转换运算符的输入值超出输出数据类型的值范围，则不会定义运算结果，并且运算结果与平台息息相关。例如，当负运算数值从 LREAL 转换为目标数据类型 UINT 时，就会出现这种情况。也可能会出现异常错误！

信息可能会丢失

如果将较大的数据类型转换为较小的数据类型，可能会导致信息丢失。

该 IEC 运算符用于将 REAL 数据类型转换为 INT 数据类型。TwinCAT 仅使用数字的整数部分的量。

TRUNC_INT 与 TwinCAT 2.x PLC 中的运算符 TRUNC 相对应，在导入 TwinCAT 2.x PLC 项目时会自动用它进行替代。请注意 TRUNC 函数的变化。

示例：

ST 代码	Result
`TRUNC_INT(1.9)`	1
`TRUNC_INT(-1.4)`	-1

16.3.10 比较运算符

比较运算符是 Boolean 运算符，用于比较 2 个输入（第 1 个操作数和第 2 个操作数）。

16.3.10.1 EQ

该 IEC 运算符用于“相等”函数。

操作数的允许数据类型：任何基本数据类型。

如果操作数相同，则运算符会返回 TRUE，否则返回 FALSE。

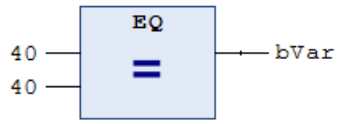
示例：

结果：bVar 为 TRUE。

ST：

```
bVar := 40 = 40;
```

FBD：



16.3.10.2 GE

该 IEC 运算符用于“大于或等于”函数。

允许的操作数数据类型：任何基本数据类型。

如果第 1 个操作数大于第 2 个操作数或者与第 2 个操作数大小相同，则运算符会返回 TRUE，否则返回 FALSE。

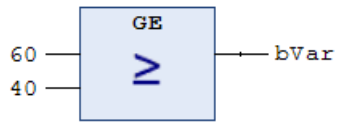
示例：

结果：bVar 为 TRUE。

ST：

```
bVar := 60 >= 40;
```

FBD：



16.3.10.3 GT

该 IEC 运算符用于“大于”函数。

允许的操作数数据类型：任何基本数据类型。

如果第 1 个操作数大于第 2 个操作数，则运算符会返回 TRUE，否则返回 FALSE。

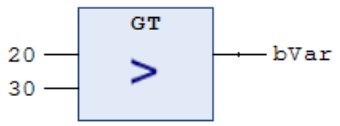
示例：

结果：bVar 为 FALSE。

ST：

```
bVar := 20 > 30;
```

FBD：



16.3.10.4 LE

该 IEC 运算符用于“小于或等于”函数。

允许的操作数数据类型：任何基本数据类型。

如果第 1 个操作数小于第 2 个操作数或者与第 2 个操作数大小相同，则运算符会返回 TRUE，否则返回 FALSE。

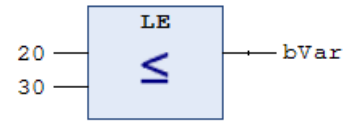
示例：

结果：bVar 为 TRUE。

ST：

```
bVar := 20 <= 30;
```

FBD：



16.3.10.5 LT

该 IEC 运算符用于“小于”函数。

允许的操作数数据类型：任何基本数据类型。

如果第 1 个操作数小于第 2 个操作数，则运算符会返回 TRUE，否则返回 FALSE。

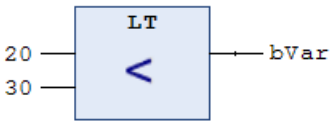
示例：

结果：bVar 为 TRUE。

ST：

```
bVar := 20 < 30;
```

FBD：



16.3.10.6 NE

该 IEC 运算符用于“不相等”函数。

操作数的允许数据类型：任何基本数据类型。

如果操作数不相同，则运算符会返回 TRUE，否则返回 FALSE。

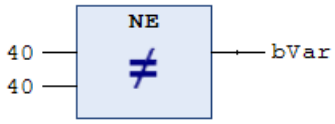
示例：

结果：bVar 为 FALSE。

ST：

```
bVar := 40 <> 40;
```

FBD：



16.3.11 其他运算符

16.3.11.1 __NEW

该运算符是 IEC 61131-3 标准的扩展。

__NEW 运算符动态保留功能块周围的内存，以实例化用户定义的数据类型或标准数据类型的数组。运算符会返回 1 个合适类型的指针。

语法：

```
<pointer name> := __NEW( <type> ( , <size> )? );
__DELETE( <pointer name> );
```

```
<type> : <function block> | <data unit type> | <standard data type>
// (...)? : Optional
```

示例：

```
pScalarType := __NEW(ScalarType, length);
```

__NEW 运算符创建 1 个 <type> 类型的实例，并返回 1 个指向该实例的指针。然后，调用该实例的初始化。如果 <type> 是标准标量数据类型，则还会对可选操作数 <size> 进行求值。在这种情况下，运算符会生成 1 个类型为 <standard data type> 且长度为 <size>（元素数量）的数组。如果内存分配尝试失败，则 __NEW 返回值为 0。

在赋值中使用 __NEW 运算符（":="），否则会发出错误消息。

通过在线更改无法进行类型更改

使用 __NEW 动态创建实例的功能块或用户定义的数据类型会占用固定的内存区。使用在线更改功能不能更改其数据布局。这意味着不能添加任何新变量，不能删除任何变量，也不能更改任何类型。这样可以确保指向该对象的指针在在线更改之后仍然有效。

因此，__NEW 运算符仅可应用于库中的功能块/DUT，以及带有“enable_dynamic_creation”属性 [▶ 740]的功能块/DUT。如果更改此类功能块/DUT 的接口，则 TwinCAT 会发出错误消息。

从路由器内存池中分配动态内存。

TwinCAT 路由器的状态信息

Tc2_Utilities 库中的功能块 FB_GetRouterStatusInfo 可用于从 PLC 中读取 TwinCAT 路由器的状态信息，例如，可用的路由器内存。

使用 __NEW 保留的动态内存必须使用 __DELETE 进行显式释放。此操作必须在同一个任务周期内完成，或者最迟在 PLC 关闭之前完成。否则，可能会出现所谓的内存泄漏。因此，其他程序部分无法再保留内存，这可能会导致系统不稳定。建议使用 PLC 库 Tc3_DynamicMemory，以简化动态内存的处理。

分配 STRING 变量：

如果您想要分配长度为 cLength 的 STRING，可使用 BYTE 代替 STRING 作为数据类型。

```
VAR CONSTANT
    cLength : UINT := 150;
END_VAR
VAR
    bNew    : BOOL;
    pString : POINTER TO STRING(cLength);
    bDelete : BOOL;
END_VAR
```

```
IF bNew AND (pSTRING = 0) THEN
    pString := __NEW(BYTE, cLength);
    bNew    := FALSE;
END_IF

IF bDelete THEN
    __DELETE(pSTRING);
    bDelete := FALSE;
END_IF
```

带结构的示例：

结构 ST_sample：

```
{attribute 'enable_dynamic_creation'}
TYPE ST_Sample :
STRUCT
    a,b,c,d,e,f : INT;
END_STRUCT
END_TYPE
```

MAIN 程序：

```
PROGRAM MAIN
VAR
    pDut    : POINTER TO ST_Sample;
    bNew    : BOOL := TRUE;
    bDelete : BOOL;
END_VAR
```

```
IF bNew AND (pDut = 0) THEN
    pDut := __NEW(ST_Sample);
    bNew := FALSE;
END_IF

IF bDelete THEN
    __DELETE(pDut);
    bDelete := FALSE;
END_IF
```

带功能块的示例：

功能块 FB_Dynamic：

```
{attribute 'enable_dynamic_creation'}
FUNCTION_BLOCK FB_Dynamic
VAR
    nCounts : INT;
END_VAR
```

方法增加：

```
METHOD Increase : INT
VAR_INPUT
    nCountStep : INT;
END_VAR
```

```
nCounts := nCounts + nCountStep;
Increase := nCounts;
```

MAIN 程序：

```
PROGRAM MAIN
VAR
    pFB       : POINTER TO FB_Dynamic;
    nResult   : INT;
    nIncrease : INT := 5;
    bNew      : BOOL;
    bDelete   : BOOL;
END_VAR
```

```
IF bNew AND (pFB = 0) THEN
    pFB := __NEW(FB_Dynamic);
    bNew := FALSE;
END_IF

IF (pFB <> 0) THEN
    nResult := pFB^.Increase(nCountStep := nIncrease);
END_IF

IF bDelete THEN
    __DELETE(pFB);
    bDelete := FALSE;
END_IF
```

带数组的示例：

MAIN 程序：

```
PROGRAM MAIN
VAR
    bNew        : BOOL := TRUE;
    bDelete     : BOOL;
    pArrayBytes : POINTER TO BYTE;
    nTest       : INT;
END_VAR
```

```
IF bNew AND (pArrayBytes = 0) THEN
    pArrayBytes := __NEW(BYTE, 25);
    bNew := FALSE;
END_IF

IF (pArrayBytes <> 0)THEN
    pArrayBytes[24] := 125;  // writing a value to the last array element
    nTest := pArrayBytes[24]
END_IF

IF bDelete THEN
    __DELETE(pArrayBytes);
    bDelete := FALSE;
END_IF
```

16.3.11.2 __DELETE

该运算符是 IEC 61131-3 标准的扩展。

该运算符会释放由运算符 __NEW 动态生成的实例内存。运算符 DELETE 没有返回值，在该操作后，操作数将被设置为 0。

语法： __DELETE (<Pointer>)

如果指针指向 1 个功能块，则 TwinCAT 会在指针被设置为 0 之前调用相应的方法 FB_exit。

处理动态内存

始终指定要释放内存的实例的确切类型。
对于继承，应指定派生功能块（类型为 POINTER TO FB_Sub 的变量），而不是基本功能块（类型为 POINTER TO FB_Base 的变量）。
如果基本功能块没有实现 FB_exit 函数，那么，在稍后调用 __DELETE（pfbBase）时，就不会调用 FB_exit！

示例：

功能块 FB_Dynamic：

```
FUNCTION_BLOCK FB_Dynamic
VAR_INPUT
    nIn1, nIn2 : INT;
END_VAR
VAR_OUTPUT
    nOut : INT;
END_VAR
VAR
    nTest1 : INT := 1234;
    _inc   : INT := 0;
    _dut   : POINTER TO DUT;
    bNeu   : BOOL;
END_VAR
```

```
nOut := nIn1 + nIn2;
```

方法 FB_exit：

```
METHOD FB_exit : BOOL
VAR_INPUT
    bInCopyCode : BOOL;
END_VAR
```

```
__DELETE(_dut);
```

方法 FB_init：

```
METHOD FB_init : BOOL
VAR_INPUT
    bInitRetains : BOOL;
    bInCopyCode  : BOOL;
END_VAR
```

```
_dut := __NEW(DUT);
```

方法 INC：

```
METHOD INC : INT
VAR_INPUT
END_VAR
```

```
_inc := _inc + 1;
INC  := _inc;
```

MAIN 程序：

```
PROGRAM MAIN
VAR
    pFB     : POINTER TO FB_Dynamic;
    bInit   : BOOL := TRUE;
    bDelete : BOOL;
    nLoc    : INT;
END_VAR
```

```
IF (bInit) THEN
    pFB := __NEW(FB_Dynamic);
    bInit := FALSE;
END_IF

IF (pFB <> 0) THEN
    pFB^(nIn1 := 1, nIn2 := nLoc, nOut => nLoc);
    pFB^.INC();
END_IF

IF (bDelete) THEN
    __DELETE(pFB);
END_IF
```

16.3.11.3 __ISVALIDREF

该运算符是 IEC 61131-3 标准的扩展。

该运算符用于检查引用是否指向 1 个值。因此，该检查类似于指针变量中的“不等于 0”检查。

有关该运算符的应用说明和使用示例，请参见关于数据类型 REFERENCE [▶ 712] 的描述。

检查接口变量

运算符 __ISVALIDREF 仅可用于 REFERENCE TO 类型的操作数。此运算符不能用于检查接口变量。如要检查接口变量是否已分配给 1 个功能块实例，您可以检查接口变量是否等于 0（IF iSample <> 0 THEN …）。

16.3.11.4 __QUERYINTERFACE

该运算符是 IEC 61131-3 标准的扩展。

在运行时，该运算符会执行从 1 个接口引用到另一个接口引用的类型转换。该运算符会返回 BOOL 类型的结果。TRUE 表示 TwinCAT 已成功执行转换。

语法： __QUERYINTERFACE(<ITF_Source>,<ITF_Dest>);

1. 操作数：接口变量或 FB 实例

2. 操作数：具有所需目标类型的接口变量

显式转换的前提条件是 ITF_Source 和 ITF_Dest 均为 __System.IQueryInterface 的导数。该接口隐式可用，无需任何库。

示例：

接口：

```
INTERFACE I_Base EXTENDS __System.IQueryInterface
METHOD BaseMethod : BOOL
```

```
INTERFACE I_Sub1 EXTENDS I_Base
METHOD SubMethod1 : BOOL
```

```
INTERFACE I_Sub2 EXTENDS I_Base
METHOD SubMethod2 : BOOL
```

```
INTERFACE I_Sample EXTENDS __System.IQueryInterface
METHOD SampleMethod : BOOL
```

功能块：

```
FUNCTION_BLOCK FB_1 IMPLEMENTS I_Sub1
METHOD BaseMethod : BOOL
    BaseMethod := TRUE;
METHOD SubMethod1 : BOOL
    SubMethod1 := TRUE;
```

```
FUNCTION_BLOCK FB_2 IMPLEMENTS I_Sub2
METHOD BaseMethod : BOOL
    BaseMethod := FALSE;
METHOD SubMethod2 : BOOL
    SubMethod2 := TRUE;
```

```
FUNCTION_BLOCK FB_3 IMPLEMENTS I_Base, I_Sample
METHOD BaseMethod : BOOL
    BaseMethod   := FALSE;
METHOD SampleMethod : BOOL
    SampleMethod := FALSE;
```

程序：

```
PROGRAM MAIN
VAR
    fb1           : FB_1;
    fb2           : FB_2;
    fb3           : FB_3;
    iBase1        : I_Base   := fb1;
    iBase2        : I_Base   := fb2;
    iBase3        : I_Base   := fb3;
    iSub1         : I_Sub1   := 0;
    iSub2         : I_Sub2   := 0;
    iSample       : I_Sample := 0;
    bResult1      : BOOL;
    bResult2      : BOOL;
    bResult3      : BOOL;
    bResult4      : BOOL;
    bResult5      : BOOL;
END_VAR
```

```
// Result: bResult1 = TRUE as the conversion is successful => iSub1 references fb1
// Explanation: iBase1 references the object fb1 of type FB_1 that implements the interface I_Sub1
bResult1 := __QUERYINTERFACE(iBase1, iSub1);

// Result: bResult2 = FALSE as the conversion is not successful => iSub2 = 0
// Explanation: iBase1 references the object fb1 of type FB_1 that does not implement the interface
I_Sub2
bResult2 := __QUERYINTERFACE(iBase1, iSub2);

// Result: bResult3 = FALSE as the conversion is not successful => iSub1 = 0
// Explanation: iBase2 references the object fb2 of type FB_2 that does not implement the interface
I_Sub1
bResult3 := __QUERYINTERFACE(iBase2, iSub1);

// Result: bResult4 = TRUE as the conversion is successful => iSub2 references fb2
// Explanation: iBase2 references the object fb2 of type FB_2 that implements the interface I_Sub2
bResult4 := __QUERYINTERFACE(iBase2, iSub2);

// Result: bResult5 = TRUE as the conversion is successful => iSample references fb3
// Explanation: iBase3 references the object fb3 of type FB_3 that implements the interface I_Sample
bResult5 := __QUERYINTERFACE(iBase3, iSample);
```

16.3.11.5 __QUERYPOINTER

该运算符是 IEC61131-3 的扩展。

该运算符可实现在运行时将功能块的接口引用类型转换为指针。该运算符会返回 BOOL 类型的结果。TRUE 表示 TwinCAT 已成功执行转换。

由于兼容性的问题，要转换的指针的定义必须是基本接口 __SYSTEM.IQueryInterface 的扩展。

语法： __QUERYPOINTER (<ITF_Source>, <Pointer_Dest>)

分配给运算符的第 1 个操作数是接口引用或具有所需目标类型的 FB 实例，第 2 个操作数是指针。在处理 __QUERYPOINTER 之后，Pointer_Dest 包含指向功能块的引用或实例的指针，接口引用 ITF_Source 当前指向该功能块。Pointer_Dest 没有分型，可以被转换为任何类型。确保类型正确。例如，接口可以提供 1 种返回类型代码的方法。

示例：

接口：

```
INTERFACE I_Base EXTENDS __System.IQueryInterface
METHOD Base : BOOL
```

```
INTERFACE I_Derived EXTENDS I_Base
METHOD Derived : BOOL
```

功能块：

```
FUNCTION_BLOCK FB_Variante IMPLEMENTS I_Derived
METHOD Base : BOOL
METHOD Derived : BOOL
```

程序：

```
PROGRAM MAIN
VAR
    iDerived   : I_Derived;
    fbVariante : FB_Variante;
    bResult    : BOOL;
    bTest      : BOOL;
    pFB        : POINTER TO FB_Variante;
END_VAR
```

```
iDerived := fbVariante;
bResult  := __QUERYPOINTER(iDerived, pFB);

IF bResult THEN
    bTest := pFB^.Derived();
END_IF
```

16.3.11.6 __TRY、__CATCH、__FINALLY、__ENDTRY

此类运算符是 IEC 61131-3 标准的扩展，用于 IEC 代码中的目标异常处理。

TC3.1 Build 4024 及以上版本适用于 32 位运行时系统

TC3.1 Build 4026 及以上版本适用于 64 位运行时系统

语法：

```
__TRY
    <try_statements>

__CATCH(exc)
    <catch_statements>

__FINALLY
    <finally_statements>

__ENDTRY

<further_statements>
```

如果在运算符 __Try 下出现的指令产生异常，则 PLC 程序不会停止。相反，它会执行 _CATCH 下的指令，从而启动异常处理。然后，执行 __FINALLY 下的指令。异常处理以 __ENDTRY 结束。随后，PLC 程序会执行后续指令（在 __ENDTRY 之后的指令）。

触发异常的指令下的 _TRY 块的指令将不再执行。这意味着，一旦放弃异常，则会中止进一步执行 _TRY 块，并执行 _CATCH 下的指令。

始终会执行 _FINALLY 下的指令，即，即使 _TRY 下的指令没有抛出任何异常也是如此。

异常的 IEC 变量具有 数据类型 __SYSTEM.ExceptionCode [▶ 681]。

注意

由于在 x86 目标系统上拦截浮点异常而导致的设备停机

由于 x86 平台的技术限制，由浮点运算所引起的拦截异常可能会产生意想不到的严重后果：

由于堆栈溢出，系统可能会陷入无法恢复的状态或者可能会发生设备停机。也有可能出现这样的情况，即设备并没有立即进入停滞状态，而是随后在堆栈上执行进一步操作时停转。

- 在 x86 目标系统的 try-catch 块中使用浮点运算时，可使用隐式检查函数。

示例

下面示例中的 _TRY 块包含 1 个指针访问和 1 个除法。在第 1 个周期中，在该块内抛出 1 个“访问违规”异常，因为指针 *pSample* 此时仍然是 1 个空指针。使用空指针会导致异常。在第 2 个周期中，在 _TRY 块内又抛出 1 个异常 - 这次是除以 0 所引起的异常。

这 2 个异常的原因均可在 __CATCH 语句中解决：通过指针 *pSample* 的值校正可以解决第 1 个异常，通过除数 *nDivisor* 的值校正可以解决第 2 个异常。

异常处理的工作原理和工作方式可以在运行时理解如下：

- 由于通常会导致运行时异常和相应的停止执行程序的错误访问会在异常处理期间被拦截，因此 TC 在运行时仍然处于运行模式，并且程序继续执行。
- 由于拦截了 2 个异常，计数器 *nCounter_CATCH* 会递增 2 次，因此值为 2。
- 由于在每个周期内都会执行下述指令，因此在每个周期内递增的计数器都具有相同的值。该值与迄今为止的周期数相对应。
  - TRY-CATCH 块前的指令 = 计数器 *nCounter1* 的递增
  - _TRY 块起始处的指令 = 计数器 *nCounter_TRY1* 的递增
  - _FINALLY 下的指令 = 计数器 *nCounter_FINALLY* 的递增
  - TRY-CATCH 块后的指令 = 计数器 *nCounter2* 的递增
- 由于 2 次抛出异常而导致中止执行 _TRY 块 2 次，因此 _TRY 指令在 2 个周期内无法结束。因此，*nCounter_TRY2* 的变量值比 *nCounter_TRY1* 小 2。
- 由于这 2 个异常的原因均在 _CATCH 指令中得到解决（为 *pSample* 分配 1 个有效地址，为 *nDivisor* 分配 1 个不为 0 的值），因此这 2 个异常分别只出现 1 次。
- 这 2 个异常会被保存在全局数组的示例中，以便创建异常历史记录。因此，在这 2 个异常之后，相应的索引变量 *nExcIndex* 为 2，数组 *aExceptionHistory* 的值如下：
  - *aExceptionHistory*[0] = RTSEXCPT_ACCESS_VIOLATION
  - *aExceptionHistory*[1] = RTSEXCPT_DIVIDEBYZERO
  - *aExceptionHistory*[2] 至 *aExceptionHistory*[10] = RTSEXCPT_NOEXCEPTION
- 在这 2 个异常之后，变量 *exc* 已经返回默认值 RTSEXCPT_NOEXCEPTION，变量 *lastExc* 表示最后发生的异常代码 RTSEXCPT_DIVIDEBYZERO。

全局变量列表“GVL_Exc”：

```
{attribute 'qualified_only'}
VAR_GLOBAL CONSTANT
    cMaxExc             : UINT := 10;
END_VAR
VAR_GLOBAL
    nExcIndex           : UINT;
    aExceptionHistory   : ARRAY[0..cMaxExc] OF __SYSTEM.ExceptionCode;
END_VAR
```

函数“F_SaveExceptionCode”：

```
FUNCTION F_SaveExceptionCode
VAR_INPUT
    excInput            : __SYSTEM.ExceptionCode;
END_VAR
```

```
// Log the thrown exception into the global exception history array
IF GVL_Exc.nExcIndex <= GVL_Exc.cMaxExc THEN
    GVL_Exc.aExceptionHistory[GVL_Exc.nExcIndex] := excInput;
    GVL_Exc.nExcIndex := GVL_Exc.nExcIndex + 1;
END_IF
```

“MAIN”程序：

```
PROGRAM MAIN
VAR
    nCounter1           : INT;
    nCounter2           : INT;
    nCounter_TRY1       : INT;
    nCounter_TRY2       : INT;
    nCounter_CATCH      : INT;
    nCounter_FINALLY    : INT;

    exc                 : __SYSTEM.ExceptionCode;
    lastExc             : __SYSTEM.ExceptionCode;

    pSample             : POINTER TO BOOL;
    bVar                : BOOL;
    nSample             : INT := 100;
    nDivisor            : INT;
END_VAR
```

```
// Counter 1
nCounter1 := nCounter1 + 1;

// TRY-CATCH block
__TRY
    nCounter_TRY1 := nCounter_TRY1 + 1;
    pSample^      := TRUE;                // 1. cycle: null pointer access leads to "access
violation" exception
    nSample       := nSample/nDivisor;    // 2. cycle: division by zero leads to "divide by zero"
exception
    nCounter_TRY2 := nCounter_TRY2 + 1;

__CATCH(exc)
    nCounter_CATCH := nCounter_CATCH + 1;

    // Exception logging
    lastExc := exc;
    F_SaveExceptionCode(excInput := exc);

    // Correct the faulty variable values
    IF (exc = __SYSTEM.ExceptionCode.RTSEXCPT_ACCESS_VIOLATION) AND (pSample = 0) THEN
        pSample  := ADR(bVar);
    ELSIF ((exc = __SYSTEM.ExceptionCode.RTSEXCPT_DIVIDEBYZERO) OR (exc =
__SYSTEM.ExceptionCode.RTSEXCPT_FPU_DIVIDEBYZERO)) AND (nDivisor = 0) THEN
        nDivisor := 1;
    END_IF

__FINALLY
    nCounter_FINALLY := nCounter_FINALLY + 1;

__ENDTRY

// Counter 2
nCounter2 := nCounter2 + 1;
```

16.3.11.6.1 数据类型 __SYSTEM.ExceptionCode

表示异常代码的 IEC 变量的数据类型为 __SYSTEM.ExceptionCode。例如，__TRY、__CATCH、__FINALLY、__ENDTRY [▶ 679] 就能捕获此类异常，可将其用于有针对性地处理异常。

例如，还可以使用 ExceptionCode 数据类型对应用程序代码本身出现的错误或异常进行分类，然后记录这些错误或异常等。请参见本页底部的示例。

数据类型 __SYSTEM.ExceptionCode：

```
TYPE ExceptionCode :
(
    RTSEXCPT_UNKNOWN                                := 16#FFFFFFFF,
    RTSEXCPT_NOEXCEPTION                            := 16#00000000,
    RTSEXCPT_WATCHDOG                               := 16#00000010,
    RTSEXCPT_HARDWAREWATCHDOG                       := 16#00000011,
    RTSEXCPT_IO_CONFIG_ERROR                        := 16#00000012,
    RTSEXCPT_PROGRAMCHECKSUM                        := 16#00000013,
    RTSEXCPT_FIELDBUS_ERROR                         := 16#00000014,
    RTSEXCPT_IOUPDATE_ERROR                         := 16#00000015,
    RTSEXCPT_CYCLE_TIME_EXCEED                      := 16#00000016,
    RTSEXCPT_ONLCHANGE_PROGRAM_EXCEEDED             := 16#00000017,
    RTSEXCPT_UNRESOLVED_EXTREFS                     := 16#00000018,
```

```
    RTSEXCPT_DOWNLOAD_REJECTED                      := 16#00000019,
    RTSEXCPT_BOOTPROJECT_REJECTED_DUE_RETAIN_ERROR  := 16#0000001A,
    RTSEXCPT_LOADBOOTPROJECT_FAILED                 := 16#0000001B,
    RTSEXCPT_OUT_OF_MEMORY                          := 16#0000001C,
    RTSEXCPT_RETAIN_MEMORY_ERROR                    := 16#0000001D,
    RTSEXCPT_BOOTPROJECT_CRASH                      := 16#0000001E,
    RTSEXCPT_BOOTPROJECTTARGETMISMATCH              := 16#00000021,
    RTSEXCPT_SCHEDULEERROR                          := 16#00000022,
    RTSEXCPT_FILE_CHECKSUM_ERR                      := 16#00000023,
    RTSEXCPT_RETAIN_IDENTITY_MISMATCH               := 16#00000024,
    RTSEXCPT_IEC_TASK_CONFIG_ERROR                  := 16#00000025,
    RTSEXCPT_APP_TARGET_MISMATCH                    := 16#00000026,
    RTSEXCPT_ILLEGAL_INSTRUCTION                    := 16#00000050,
    RTSEXCPT_ACCESS_VIOLATION                       := 16#00000051,
    RTSEXCPT_PRIV_INSTRUCTION                       := 16#00000052,
    RTSEXCPT_IN_PAGE_ERROR                          := 16#00000053,
    RTSEXCPT_STACK_OVERFLOW                         := 16#00000054,
    RTSEXCPT_INVALID_DISPOSITION                    := 16#00000055,
    RTSEXCPT_INVALID_HANDLE                         := 16#00000056,
    RTSEXCPT_GUARD_PAGE                             := 16#00000057,
    RTSEXCPT_DOUBLE_FAULT                           := 16#00000058,
    RTSEXCPT_INVALID_OPCODE                         := 16#00000059,
    RTSEXCPT_MISALIGNMENT                           := 16#00000100,
    RTSEXCPT_ARRAYBOUNDS                            := 16#00000101,
    RTSEXCPT_DIVIDEBYZERO                           := 16#00000102,
    RTSEXCPT_OVERFLOW                               := 16#00000103,
    RTSEXCPT_NONCONTINUABLE                         := 16#00000104,
    RTSEXCPT_PROCESSORLOAD_WATCHDOG                 := 16#00000105,
    RTSEXCPT_FPU_ERROR                              := 16#00000150,
    RTSEXCPT_FPU_DENORMAL_OPERAND                   := 16#00000151,
    RTSEXCPT_FPU_DIVIDEBYZERO                       := 16#00000152,
    RTSEXCPT_FPU_INEXACT_RESULT                     := 16#00000153,
    RTSEXCPT_FPU_INVALID_OPERATION                  := 16#00000154,
    RTSEXCPT_FPU_OVERFLOW                           := 16#00000155,
    RTSEXCPT_FPU_STACK_CHECK                        := 16#00000156,
    RTSEXCPT_FPU_UNDERFLOW                          := 16#00000157,
    RTSEXCPT_VENDOR_EXCEPTION_BASE                  := 16#00002000,
    RTSEXCPT_USER_EXCEPTION_BASE                    := 16#00010000
) UDINT ;
END_TYPE
```

错误/异常分类的实现示例

全局变量列表“GVL_Exc”：

```
VAR_GLOBAL
    nCheckBounds              : INT;
    nCheckBounds_OutOfBounds  : INT;
    myException               : __SYSTEM.ExceptionCode;
END_VAR
```

“CheckBounds”h函数：

```
// Implicitly generated code : DO NOT EDIT
FUNCTION CheckBounds : DINT
VAR_INPUT
    index, lower, upper       : DINT;
END_VAR
```

```
// Only an implementation suggestion
{noflow}
GVL_Exc.nCheckBounds := GVL_Exc.nCheckBounds + 1;

// Index too low
IF index < lower THEN
    CheckBounds                     := lower;
    GVL_Exc.myException             := __SYSTEM.ExceptionCode.RTSEXCPT_ARRAYBOUNDS;
    GVL_Exc.nCheckBounds_OutOfBounds := GVL_Exc.nCheckBounds_OutOfBounds + 1;

// Index too high
ELSIF index > upper THEN
    CheckBounds                     := upper;
    GVL_Exc.myException             := __SYSTEM.ExceptionCode.RTSEXCPT_ARRAYBOUNDS;
    GVL_Exc.nCheckBounds_OutOfBounds := GVL_Exc.nCheckBounds_OutOfBounds + 1;

// Index OK
ELSE
    CheckBounds := index;
END_IF
```

另请参见：

- 边界检查（POU CheckBounds） [▶ 153]

16.3.11.7 __VARINFO

该运算符是 IEC 61131-3 标准的扩展。该运算符会返回有关变量的信息。您可以将该信息作为数据结构保存在数据类型为 __SYSTEM.VAR_INFO 的变量中。

声明中的语法：

```
<name of the info variable> : __SYSTEM.VAR_INFO; // Data structure for info variable
```

调用的语法：

```
<name of the info variable> := __VARINFO( <variable name> ); // Call of the operator
```

示例：

在运行时，变量 MyVarInfo 包含有关变量 nVar 的信息。

```
VAR
    MyVarInfo : __SYSTEM.VAR_INFO;
    nVar      : INT;
END_VAR
MyVarInfo := __VARINFO(nVar);
```

数据类型 SYSTEM.VAR_INFO

数据类型为 __SYSTEM.VAR_INFO 的变量包含：

名称	数据类型	初始化	描述
ByteAddress	DWORD	0	变量的地址 示例：16#072E35EC 注意：在对变量 <variable name>.<bit index> 进行位访问期间，应指定包含该位的变量地址。
ByteOffset	DWORD	0	变量地址的偏移量，以字节为单位。 示例：13936 字节。 注意：如果变量是全局变量，则偏移量相对于区域的起点。如果变量是函数或方法中的局部变量，则偏移量相对于当前堆栈帧。如果变量是功能块中的局部变量，则偏移量相对于功能块实例。
区域	DINT	0	运行时系统中的内存区编号。 示例：-1。这意味着变量不是全局存储在内存中，而是相对于某个实例或在堆栈上存储。 注意：内存区与设备相关。
BitNo	INT	0	以字节为单位的位数 示例：16#00FF 字节 注意：如果变量不是整数数据类型：BitNo = -1 = 16#FFFF
BitSize	INT	0	变量的内存大小，以位为单位 示例：16 位
BitAddress	UDINT	0	变量的位地址 要求：变量位于输入内存区 I、输出内存区 Q 或标志内存区 M。否则，未定义该值。
TypeClass	TYPE_CLASS	TYPE_BOOL	变量的数据类型类 示例：TYPE_INT、TYPE_ARRAY 注意：对于用户定义的数据类型或功能块实例，系统会输出数据类型类 TYPE_USERDEF。
TypeName	STRING(79)	‘’	变量的数据类型名称为 STRING(79) 注意：对于用户定义的数据类型，应指定功能块名称或 DUT 名称。 示例：’INT’、’ARRAY’
NumElements	UDINT	0	数组元素的数量 要求：变量的数据类型为 ARRAY。 示例：8
BaseTypeClass	TYPE_CLASS	TYPE_BOOL	数组元素的初级基本数据类型。 要求：变量的数据类型为 ARRAY。 示例：TYPE_INT with arrA : ARRAY [1..2,1..2,1..2] OF INT;
ElemBitSize	UDINT	0	数组元素的内存大小，以位为单位 要求：变量的数据类型为 ARRAY。 示例：16 bits at arrA : ARRAY [1..2,1..2,1..2] OF INT;

MemoryArea	MEMORY_AREA	MEM_MEMORY	有关内存区的信息： • MEM_GLOBAL：全局内存区 例如，在区域 0 中 • MEM_LOCAL：局部内存区 在区域 -1 中 • MEM_MEMORY：标记内存区 %M 例如，在区域 1 的 16#10 中 • MEM_INPUT：输入内存区 %I 例如，在区域 2 的 16#04 中 • MEM_OUTPUT：输出内存区 %Q 例如，在区域 3 的 16#08 中 • MEM_RETAIN：保留内存区 例如，在区域 0 的 16#20 中 示例：MEM_GLOBAL
符号	STRING(39)	‘’	变量名为 STRING(39) 示例：’iCounter’、’arrA’
注释	STRING(79)	‘’	变量声明的注释 示例：’Counts the calls’ 或 ’Stores the A data’

16.3.11.8 __POUNAME

TC3.1 Build 4026 及以上可用。

该运算符是 IEC 61131-1 标准的扩展。

该运算符在运行时会返回包含 __POUNAME 运算符的程序组织单元（POU）的名称。为此，必须在声明部分或实现部分中将运算符分配给 STRING 类型的变量。

__POUNAME 的结果取决于使用它的位置：

- 在程序内：程序名称
- 在函数内：函数名称
- 在功能块内：功能块名称
- 在方法内：方法名称，用功能块名称限定
- 在属性中的 Get/Set 访问器内部：属性名称，用功能块名称限定 + Get/Set
- 在 GVL 内：GVL 的名称
- 在结构内：结构名称
- 在数据结构 UNION 内：UNION 的名称

示例：

```
PROGRAM MAIN
VAR
    sPouNameDecl : STRING := __POUNAME();  //Liefert 'MAIN'
    sPouNameImpl : STRING;
END_VAR
```

```
sPouNameImpl:= __POUNAME();                //Liefert 'MAIN'
```

16.3.11.9 __POSITION

TC3.1 Build 4026 及以上可用。

该运算符是 IEC 61131-1 标准的扩展。

该运算符在运行时会返回变量在编程块的声明部分或实现部分中的位置。为此，必须在声明部分或实现部分中将运算符 __POSITION 分配给 STRING 类型的变量。

__POSITION 的结果：

- 声明部分：Line <line number> (Decl)
- 实现部分：Line <line number>, Column <Column number> (Impl)

示例：

```
PROGRAM MAIN
VAR
    sPositionDecl  : STRING := __POSITION();   //Liefert die Zeilennummer dieser Deklaration
    sPositionImpl  : STRING;
END_VAR
```

```
sPositionImpl := __POSITION();                 //Liefert Zeilen- und Spaltennummer dieser Zuweisung
```

16.4 操作数

常量和字面量

常量是不变值的标识符。您可以在编程块中局部声明常量，也可以在全局变量列表中全局声明常量。为此，声明部分使用关键字 CONSTANT 进行扩展。

常量也是表示诸如整数或浮点数等基本类型值的字符串，例如，16#FFFF_FFFF、T#5s 或 -1.234 E-5。为了区别它们，此类常量也被称为字面量、字面常量或未命名常量。有逻辑字面量（TRUE、FALSE）或数字字面量（3.1415、T#5s），也有字符字面量（'Hello world!'、"black"）。

语法声明：

```
<scope> CONSTANT
    <identifier>:<data type> := <initial value>;
END_VAR
```

```
<scope>         : VAR | VAR_INPUT | VAR_STAT | VAR_GLOBAL
<data type>     : <elementary data type | user defined data type | function block >
<initial value> : literal value | identifier | expression
```

允许的初始值：

- 字面量，例如 TRUE、FALSE、16#FFFF_FFFF
- 在别处声明的命名常量。
- 由字面量组成的简单表达式，也可与诸如 + - * 等简单运算符组合使用。

不能将输入或函数调用指定为初始值。

示例：

```
VAR_GLOBAL CONSTANT
    cMax     : INT := 100;
    cSpecial : INT := cMax - 10;
END_VAR
```

仅在声明中描述常量。初始值的分配是强制性的。在实现中，常量为只读，因此常量总是出现在指令中分配运算符的右侧。

在编译代码时，常量将被初始值替换。此外，还必须能够在编译时计算初始值。

结构化或用户定义类型的常量仅在运行时进行计算。在程序启动时会计算 1 次程序或 GVL 中的结构化常量。每次调用函数或方法时，都会计算函数或方法中的结构化常量。因此，结构化常量的初始化可能取决于输入或执行函数调用。

另请参见：

- BOOL 常量 [▶ 687]
- 数字常量 [▶ 687]
- REAL/LREAL 常量 [▶ 688]
- STRING 常量 [▶ 688]
- TIME/LTIME 常量 [▶ 689]

- 日期和时间常量 [▶ 690]
- 类型常量/类型字面量 [▶ 693]

变量

您可以在功能块的声明部分中局部声明变量，也可以在全局变量列表中声明变量。

您何时可以使用变量取决于其数据类型。

另请参见：

- 访问数组、结构和功能块的变量 [▶ 693]
- 对变量的位访问 [▶ 693]

其他

- 地址 [▶ 695]
- 函数 [▶ 697]

另请参见：

- 声明变量 [▶ 60]
- 使用输入向导 [▶ 123]
- 常量变量 - CONSTANT [▶ 629]
- ST 表达式 [▶ 572]

16.4.1 BOOL 常量

BOOL 常量的真值为 TRUE（1）和 FALSE（0）。

另请参见：

- BOOL [▶ 698]

16.4.2 数字常量

数值可以是二进制数、八进制数、十进制数或十六进制数。如果整数值不是十进制数，则必须在整数常量之前写入其基数，后跟井字符号（#）。对于十六进制数，10 至 15 的数字与往常一样用字母 A-F 表示。

您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
DINT#16#A1	类型化数据类型 DINT# 和基数 16# 组合。

该数值的类型可以是 BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL 或 LREAL。

不允许从“较大”类型隐式转换为“较小”类型。您不能简单地将 DINT 变量用作 INT 变量。为此，您必须使用类型转换函数。

由于数字常量通常被视为整数值，因此，在除法中必须将常量指定为浮点数，以避免丢失其余部分。例如：除法 1/10 的结果为 0，除法 1.0/10 的结果为 0.1。

另请参见：

- 运算符 [▶ 635]

- 类型常量/类型字面量 [▶ 693]

16.4.3 REAL/LREAL 常量

您可以将浮点数指定为点符号形式或带有尾数和指数的指数符号形式的 REAL 和 LREAL 常量。根据国际单位制（英语），点可作为十进制符号。

语法指数表达式：

```
<significand> e | E <exponent>
```

```
exponent :  -44..38    // REAL
exponent : -324..308   // LREAL
```

示例：

7.4	十进制数；带逗号的 7,4 会返回编译器错误。
1/3.0	0.333333343 的十进制小数 注意：在进行整数类型的除法运算时，结果仍然是整数类型。采取截断的方式。例如，1/3 返回的结果是 0。
1.64e+009	指数表达式

REAL 字面量

-3.402823e+38	最小数
-1E-44	最大负数
1.0E-44	最小正数
3.402823e+38	最大数

LREAL 字面量

-1.7976931348623157E+308	最小数
-4.94065645841247E-324	最大负数
4.94065645841247E-324	最小正数
1.7976931348623157E+308	最大数

特大字面量的分配

如要分配 1 个大于 ULINT 上限的整数字面量，可设置逗号或指定显式类型转换。如果没有浮点数这样的规范，则可能会丢失信息。
示例：

```
fMyReal : REAL := 3400000000000000000000.0;
fMyReal : REAL := REAL#340000000000000000000000;
```

另请参见：

- REAL/LREAL [▶ 700]

16.4.4 STRING 常量

STRING 常量是 1 个用单引号括起来的字符串。这些字符根据 Windows 1252 字符集进行编码。作为 Windows-1252 的子集，ISO/IEC 8859-1 的字符集是受支持的。STRING 常量可以包含空格和变音符，因为这些字符是字符集的一部分。它也被称为字符字面量或简单称为字符串。

示例：

```
'Hello World!'
```

如果 STRING 常量包含美元符号（$），则后面的 2 个字符将根据 Windows-1252 编码被解释为十六进制代码。该代码也与 ASCII 码相对应。另请注意特殊情况。

十六进制编码

带 $ 代码的字符串	解释
$<8-bit code>	8 位代码：根据 ISO/IEC 8859-1 解释的 2 位十六进制数。
'$41'	A
'$A9'	©
'$40'	@
'$0D'	控制字符断行，相当于 '$R'。
'$0A'	控制字符新行，相当于 '$L' 和 '$SN'。

特殊情况

带 $ 代码的字符串	解释
'$L'、'$l'	控制字符换行，相当于 '$0A'。
'$N'、'$n'	控制字符新行，相当于 $0A'。
'$P'、'$p'	控制字符换页
'$R'、'$r'	控制字符断行，相当于 '$0D'。
'$T'、'$t'	控制字符选项卡
'$$'	美元符号：$
'$''	单引号：'

示例：常量声明

```
VAR CONSTANT
    sConstA : STRING := 'Hello Allgäu';
    sConstB : STRING := 'Hello Allgäu $21'; // Hello Allgäu!
END_VAR
```

16.4.5 TIME/LTIME 常量

您可以使用 TIME 常量来处理标准计时器模块。常量的大小为 32 位，因此以毫秒为单位进行解析。

此外，时间常量 LTIME 可作为高精度计时器的时基。LTIME 常量的大小为 64 位，因此以纳秒为单位进行解析。

TIME 常量

语法：

```
<time keyword> # <length of time>
```

```
<time keyword>   : TIME | time | T | t
<length of time> : ( <number of days>d )? ( <number of hours>h )? ( <number of minutes>m )?
( <number of seconds>s )? (<number of milliseconds>ms)?    // ( ...)? Optional
```

时间单位的顺序不得更改。不必使用所有单位。

时间单位：

- D | d：天
- H | h：小时
- M | m：分钟
- S | s：秒钟
- MS | ms：毫秒

示例：ST 赋值中的正确时间常量

```
VAR
    tLength0 : TIME := T#14ms;
    tLength1 : TIME := T#100s12ms;      // Overflow in the highest unit is allowed.
    tLength2 : TIME := T#12h34m15s;
    tCompare : TIME;
    bOK      : BOOL;
    tLongest := T#49D17H2M47S295MS;    // 4294967295
END_VAR
```

```
IF tLength < T#15MS THEN
    IF tCompare < tLength1 THEN
        bOK := TRUE;
    END_IF;
END_IF
```

示例：时间常量的不正确使用

tIncorrect1 := t#5m68s;	在较低位置溢出
tIncorrect2 := 15ms;	时间标识符 T# 缺失
tIncorrect3 := t#4ms13d;	时间单位的顺序不正确

LTIME 常量

语法：

```
<long time keyword> # <length of high resolution time>
```

```
<long time keyword>              : LTIME | ltime
<length of high resolution time> : <length of time> ( <number of microseconds>us )? ( <number of
nanoseconds>ns )?    // ( ...)? Optional
```

对于 LTIME 常量，您可以使用与 TIME 常量相同的时间单位。此外，您还可以指定微秒和纳秒，因为时间规范是以更高的时间解析度计算的。在内部，LTIME 字面量可被视为数据类型 LWORD，因此该值以纳秒为单位进行解析。

其他时间单位：

- US | us：微秒
- NS | ns：纳秒

示例：ST 赋值中的正确时间常量

```
VAR
    tLength0 : TIME := LTIME#1000d15h23m12s34ms2us44ns;
    tLength1 : TIME := LTIME#3445343m3424732874823ns;
END_VAR
```

16.4.6 日期和时间常量

32 位日期“DATE”

使用 DATE (D) 关键字指定日期。

语法：

```
<date keyword>#<year>-<month>-<day>
```

```
<date keyword> : DATE | date | D | d
<year>  : 1970-2106
<month> : 1-12
<day>   : 1-31
```

DATE 字面量在内部可被视为 DWORD 数据类型，它与 DATE#2106-2-7 的上限值对应。

示例：

```
PROGRAM MAIN
VAR
    dStart    : DATE := DATE#2018-8-8;
    dEnd      : DATE := D#2018-8-31;
    dCompare  : DATE := date#1996-05-06;
    bInTime   : BOOL;

    dEarliest : DATE := d#1970-1-1;      // = 0
    dLatest   : DATE := DATE#2106-2-7;   // = 4294967295
END_VAR
```

```
IF dStart < dCompare THEN
    IF dCompare < dEnd THEN
        bInTime := TRUE;
    END_IF;
END_IF
```

64 位日期“LDATE”

使用 LDATE (LD) 关键字指定日期。

语法：

```
<date keyword>#<year>-<month>-<day>
```

```
<date keyword> : LDATE | ldate | LD | ld
<year>  : 1970-2554
<month> : 1-12
<day>   : 1-31
```

LDATE 字面量在内部以 LWORD 数据类型进行处理，对应的上限值是 DATE#2554-7-21。

示例：

```
PROGRAM MAIN
VAR
    dStart    : LDATE := LDATE#2018-8-8;
    dEnd      : LDATE := ldate#2018-8-31;
    dCompare  : LDATE := LD#1996-05-06;
    bInTime   : BOOL;

    dEarliest : LDATE := ld#1970-1-1;      // = 0
    dLatest   : LDATE := LDATE#2554-7-21;
END_VAR
```

```
IF dStart < dCompare THEN
    IF dCompare < dEnd THEN
        bInTime := TRUE;
    END_IF;
END_IF
```

32 位日期和时间规范“DATE_AND_TIME”

使用 DATE_AND_TIME (DT) 关键字指定日期和时间。

语法：

```
<date and time keyword>#<date and time value>
```

```
<date and time keyword>: DATE_AND_TIME | date_and_time | DT | dt
<date and time value>: <year>-<month>-<day>-<hour>:<minute>:<second>
<year>: 1970-2106
<month>: 1-12
<day>: 1-31
<hour>: 0-24
<minute>: 0-59
<second>: 0-59
```

DATE_AND_TIME 字面量在内部可作为 DWORD 数据类型进行处理。时间以秒为单位进行处理，因此取值范围为 1970 年 1 月 1 日 00:00 至 2106 年 2 月 7 日 6:28:15。

示例：

```
PROGRAM MAIN
VAR
    dtDate0    : DATE_AND_TIME := DATE_AND_TIME#1996-05-06-15:36:30;
    dtDate1    : DATE_AND_TIME := DT#1972-03-29-00:00:00;
    dtDate2    : DT            := DT#2018-08-08-13:33:20.5;

    dtEarliest : DATE_AND_TIME := DATE_AND_TIME#1979-1-1-00:00:00;    // 0
    dtLatest   : DATE_AND_TIME := DATE_AND_TIME#2106-2-7-6:28:15;     // 4294967295
END_VAR
```

64 位日期和时间规格“LDATE_AND_TIME”

使用 LDATE_AND_TIME (LDT) 关键字指定日期和时间。

语法：

```
<date and time keyword>#<long date and time value>
```

```
<date and time keyword>  : LDATE_AND_TIME | ldate_and_time | LDT | ldt
<date and time value>      : <year>-<month>-<day>-<hour>:<minute>:<second>
<year>      : 1970-2554
<month>    : 1-12
<day>       : 1-31
<hour>      : 0-24
<minute>  : 0-59
<second>  : 0-59
```

LDATE_AND_TIME 字面量在内部以 LWORD 数据类型进行处理。时间以秒为单位进行处理，因此取值范围为 1970 年 1 月 1 日 00:00 至 2554 年 7 月 21 日 23:59:59.999999999。

示例：

```
PROGRAM MAIN
VAR
    dtDate0    : LDATE_AND_TIME := LDATE_AND_TIME#1996-05-06-15:36:30;
    dtDate1    : LDATE_AND_TIME := LDT#1972-03-29-00:00:00;
    dtDate2    : LDT            := LDT#2018-08-08-13:33:20.5;

    dtEarliest : LDATE_AND_TIME := LDT#1979-1-1-00:00:00;    // 0
    dtLatest   : LDATE_AND_TIME := LDT#2554-7-21-23:59:59;
END_VAR
```

32 位日期“TIME_OF_DAY”

使用 TIME_OF_DAY (TOD) 关键字指定时间。

语法：

```
<time keyword>#<time value>
```

```
<time keyword> : TIME_OF_DAY | time_of_day | TOD | tod
<time value>   : <hour>:<minute>:<second>
<hour>   : 0-23
<minute> : 0-59
<second> : 0.000-59.999
```

您还可以使用秒的小数部分来指定秒的分数。TIME_OF_DAY 字面量在内部以DWORD 进行处理，因此该值以毫秒为单位进行解析。

示例：

```
PROGRAM MAIN
VAR
    tdClockTime0 : TIME_OF_DAY := TIME_OF_DAY#15:36:30.123;
    tdClockTime1 : TOD         := TOD#12:34:56.789;

    tdEarliest   : TIME_OF_DAY := TIME_OF_DAY#0:0:0.000;
    tdLatest     : TIME_OF_DAY := TIME_OF_DAY#23:59:59.999;
END_VAR
```

64 位日期“LTIME_OF_DAY”

使用关键字 LTIME_OF_DAY (LTOD) 指定时间。

语法：

```
<time keyword>#<time value>
```

```
<time keyword> : LTIME_OF_DAY | ltime_of_day | LTOD | ltod
<time value>   : <hour>:<minute>:<second>
<hour>   : 0-23
<minute> : 0-59
<second> : 0.000-59.999999999
```

您还可以使用秒的小数部分来指定秒的分数。LTIME_OF_DAY 字面量在内部以 LWORD 进行处理，因此该值以纳秒为单位进行解析。

示例：

```
PROGRAM MAIN
VAR
    tdClockTime0 : LTIME_OF_DAY := LTIME_OF_DAY#15:36:30.123;
    tdClockTime1 : LTOD         := LTOD#12:34:56.7890123456;
```

```
    tdEarliest    : LTIME_OF_DAY := LTIME_OF_DAY#0:0:0.000;
    tdLatest      : LTIME_OF_DAY := LTIME_OF_DAY#23:59:59.999999999;
END_VAR
```

另请参见：

- 日期和时间数据类型 [▶ 702]

16.4.7 类型字面量

除了 REAL/LREAL 常量（始终使用 LREAL）之外，TwinCAT 在涉及 IEC 常量的计算中会使用最小可能的数据类型。如果您想要使用不同的数据类型，则您可以使用类型字面量（类型常量），而不必显式声明常量。为指定类型的常量分配前缀。

语法：<Type>#<Literal>

<Type> 可指定所需的数据类型；可能的条目有：BOOL、SINT、USINT、BYTE、INT、UINT、WORD、DINT、UDINT、DWORD、REAL、LREAL。您必须将类型大写。

<Literal> 可指定常量。输入必须与在 <Type> 下指定的数据类型相匹配。

示例：

```
var1:=DINT#34;
```

如果 TwinCAT 无法在不丢失数据的情况下将常量传输到目标类型，则会发出错误消息。

在您可以使用普通常量的任何地方，您都可以使用类型常量。

16.4.8 访问数组、结构和块中的变量

访问的符号：

- 二维数组组件：<Array-Name> [<1st dimension>, <2nd dimension>]
- 结构变量：<structure name> . <component name>
- 功能块或程序变量：<function block name> | <project name > . <variable name>

另请参见：

- 数组 [▶ 714]
- 结构 [▶ 719]
- 对象功能块 [▶ 76]
- 对象程序 [▶ 77]

16.4.9 对变量的位访问

通过索引访问，您可以对整数变量中的各个位进行寻址。您可以使用结构变量或功能块实例，以符号的方式对各个位进行寻址。

对整数变量中的位进行索引访问

您可以对整数变量中的各个位进行寻址。为此，可将要寻址的位的索引附加到变量中，中间用点隔开。任何常量均可指定位索引。索引以 0 为基础。

语法：

```
<integer variable name> . <index>
```

```
<integer data typ> = BYTE | WORD | DWORD | LWORD | SINT | USINT | INT | UINT | DINT | UDINT | LINT |
ULINT
```

如果变量的类型不被允许，则 TwinCAT 会发出以下错误消息：用于直接索引的数据类型 <type> 无效。如果索引大于变量的位宽，则 TwinCAT 会提示以下错误：索引 <n> 超出变量 <name> 的有效范围。

示例索引访问：

在程序中，变量 nVarA 的第 3 位被设置为变量 nVarB 的值。

```
PROGRAM MAIN
VAR
    nVarA : WORD := 16#FFFF;
bVarB : BOOL := 0;
END_VAR
```

```
// Index access in an integer variable
nVarA.2 := bVarB;
```

结果：nVarA = 2#1111_1111_1111_1011 = 16#FFFB

示例变量作为索引：

常量 cEnable 可充当索引，以访问变量 nVar 的第 3 位。

```
// GVL declaration
VAR_GLOBAL CONSTANT
    cEnable : USINT := 2;
END_VAR
```

```
PROGRAM MAIN
VAR
    nVar    : INT   := 0;
END_VAR
```

```
// Constant as index
nVar.cEnable := TRUE; // Third bit in nVar is set TRUE
```

结果：nVar = 4

由位数定义的变量的可访问性

请注意，定义位数的变量（在上述示例中 nEnable）必须可以通过变量名直接访问，且前面不能有任何命名空间。
因此，举例来说，如果在局部范围（与 nVar 同级）或不具有“qualified only” [▶ 761]属性的 GVL 上的全局范围中声明变量，则允许进行位访问。
如果在具有“qualified_only”属性的 GVL 上声明变量，则只有通过指定全局变量列表名称（GVL.nEnable）才能访问 nEnable。对全局变量的这种访问不能用于位访问（不可能：nVar.GVL.nEnable := TRUE;）。

结构变量中的符号位访问

通过数据类型 BIT，您可以用名称指定各个位，并将它们组合成 1 个结构。然后，用组件名称对位进行寻址。

示例

结构的类型声明：

```
TYPE ST_ControllerData :
STRUCT
    nStatus_OperationEnabled : BIT;
    nStatus_SwitchOnActive   : BIT;
    nStatus_EnableOperation  : BIT;
    nStatus_Error            : BIT;
    nStatus_VoltageEnabled   : BIT;
    nStatus_QuickStop        : BIT;
    nStatus_SwitchOnLocked   : BIT;
    nStatus_Warning          : BIT;
END_STRUCT
END_TYPE
```

位的声明和写入访问：

```
PROGRAM MAIN
VAR
    stControllerDrive1 : ST_ControllerData;
END_VAR
```

```
// Symbolic bit access to nStatus_ EnableOperation
stControllerDrive1.nStatus_EnableOperation := TRUE;
```

功能块实例中的符号位访问

在功能块中，您可以为各个位声明变量。

示例

结构的类型声明：

```
FUNCTION_BLOCK FB_Controller
VAR_INPUT
    nSwitchOnActive   : BIT;
    nEnableOperation  : BIT;
    nVoltageEnabled   : BIT;
    nQuickStop        : BIT;
    nSwitchOnLocked   : BIT;
END_VAR
VAR_OUTPUT
    nOperationEnabled : BIT;
    nError            : BIT;
    nWarning          : BIT;
END_VAR
VAR
END_VAR
```

```
PROGRAM MAIN
VAR
    fbController : FB_Controller;
END_VAR
```

```
// Symbolic bit access to nSwitchOnActive
fbController(nSwitchOnActive:= TRUE);
```

另请参见：

- BIT [▶ 700]
- 整数数据类型 [▶ 699]
- 结构中的位访问 [▶ 720]

16.4.10 地址

⚠ 谨慎

通过在线更改改变地址内容

如果您使用地址指针，则在在线更改时可能会改变地址内容！

自动寻址

建议不要对已分配变量使用直接寻址，而应使用 * 占位符。
如果使用占位符 *（%I*、%Q* 或 %M*），TwinCAT 将自动执行灵活的优化寻址。

语法：

```
<identifier> AT <address> : <data type>;
```

如果已经指定地址，则可以通过特殊字符串表示内存中的位置和大小。地址用百分号 % 标记，然后是内存区前缀、可选大小前缀以及内存位置。

```
%<memory area prefix> ( <size prefix> )? <memory position>
```

```
<memory area prefix> : I | Q | M
<size prefix>        : X | B | W | D
<memory position>    : * | <number> ( .<number> )*
```

内存区前缀

I	用于输入的输入内存区 通过输入驱动器（“传感器”）进行物理输入
Q	用于输出的输出内存区 通过输出驱动器（“执行器”）进行物理输出
M	标志内存区

大小前缀

X	单个位
B	字节（8 位）
W	字（16 位）
D	双字（32 位）

示例：

```
IbSensor1 AT%I*    : BOOL;
IbSensor2 AT%IX7.5 : BOOL;
```

如果您没有明确指定单个位地址，则会逐字节分配 Boolean 变量。示例：bVar AT %QB0 的值变化涉及从 QX0.0 至 QX0.7 的区域。

示例

变量声明：

IbSensor AT%I* : BOOL;	在指定地址时，指定的是占位符 *，而不是内存位置。这使得 TwinCAT 能够自动执行灵活的优化寻址。
InInput AT%IW0 : WORD;	变量声明，1 个输入字的地址指定
ObActuator AT%QB0 : BOOL;	Boolean 变量声明 注意：对于 Boolean 变量，如果没有指定单个位地址，则会在内部分配 1 个字节。因此，ObActuator 的值变化会影响从 QX0.0 至 QX0.7 的范围。
IbSensor AT%IX7.5 : BOOL;	Boolean 变量声明，明确指定单个位地址。在访问时，只读取输入位 7.5。

其他地址：

%QX7.5 %Q7.5	输出位 7.5 的单个位地址
%IW215	输入字 215 的字地址
%QB7	输出字节 7 的字节地址
%MD48	标志区中内存位置 48 的双字地址
%IW2.5.7.1	解释取决于当前控制器配置（见下文）

直接地址可能出现内存区重叠

为了在 PLC 项目中分配有效地址，您必须知道过程映像中的所需位置。为此，您首先必须定义内存区和所需大小。在选择内存位置时，您必须遵守下方表格所示的内存中不同大小的分配，以便排除内存区重叠的情况。

DWord	单词	字节	位
D0	W0	B0	X0.0
		B1	X1.0
	W1	B2	X2.0
		B3	X3.0
D1	W2	B4	X4.0
		B5	X5.0
	W3	B6	X6.0
		B7	X7.0
D2	W4	B8	X8.0
.....			

示例：内存区重叠

1. W0 包含 B0 和 B1。如果您在 W0 处放置 1 个 Word（单词）变量，并在 B1 处放置 1 个 Boolean 变量，则内存区会重叠。

2. W3 包含 B6 和 B7。如果您在 W3 处放置 1 个 Word（单词）变量，并在 B6 处放置 1 个 Boolean 变量，则内存区会重叠。

另请参见：

- 对 PLC 项目进行编程 > 声明变量 > AT 声明 [▶ 63]

16.4.11 函数

在 ST 中，您可以使用函数调用作为操作数。

示例：

```
nResult := F_Add(7,5) + 3;
```

另请参见：

- 对象函数 [▶ 73]

TIME() 函数

该函数会返回 1 个数据类型为 TIME 的值。
TIME() 函数不表示绝对引用点，但是，通过计算至少 2 个 TIME() 返回值之间的差值，可以将该函数用于相对时间测量。

ST 中的示例：

下面的示例包含 1 个 TON 块（fbTimer），该块应用了 5 秒的时间（tTimerValue），并在程序开始时启动。R_TRIG 块（fbTrigger）可以检测到计时器激活的上升沿，因此，TIME() 函数的返回值首次被缓冲（tTimeReturn1）。当经过计时器的时间间隔时，TIME() 函数的第 2 个返回值将被缓冲（tTimeReturn2）。通过至少 2 个存储的 TIME() 返回值之间的差值（tDifference）可以计算相关的 TIME() 调用之间的相对时间。在这种情况下，计时器开始和结束之间的时间按 5 秒进行计算。

```
PROGRAM MAIN
VAR
    bStart          : BOOL := TRUE;
    fbTimer         : TON;
    tTimerValue     : TIME := T#5S;
    fbTrigger       : R_TRIG;
    tTimeReturn1    : TIME;
    tTimeReturn2    : TIME;
    tDifference     : TIME;
END_VAR

//=================================================

fbTimer(IN := bStart, PT := tTimerValue);
fbTrigger(CLK := fbTimer.IN);

IF fbTrigger.Q THEN
    tTimeReturn1  := TIME();
END_IF

IF fbTimer.Q THEN
    bStart        := FALSE;
    tTimeReturn2  := TIME();
    tDifference   := tTimeReturn2 - tTimeReturn1;    // The difference will be T#5s
END_IF
```

16.5 数据类型

在编程中，变量由其名称标识，它在目标系统的内存中有 1 个地址。因此，变量名是对分配的存储空间进行寻址的标识符。变量的大小由其数据类型决定。这会指定为变量保留多少存储空间，以及如何解释内存中的值。数据类型还可以决定允许进行哪些操作。

在 TwinCAT 中还可以实例化功能块。功能块实例占用内存的方式与变量类似。内存要求由功能块决定。

以下几组数据类型可供使用：

标准数据类型

标准数据类型是基本数据类型或字符串数据类型。

```
<standard data type> : __UXINT | __XINT | __XWORD | BIT | BOOL | BYTE | DATE | DATE_AND_TIME | DINT
| DT | DWORD | INT | LDATE | LDATE_AND_TIME | LDT | LINT | LREAL | LTIME | LTOD | LWORD | REAL |
SINT |STRING | TIME | TOD | TIME_OF_DAY | UDINT | UINT | ULINT | USINT | WORD | WSTRING
```

另请参见：

- BOOL [▶ 698]
- 整数数据类型 [▶ 699]
- REAL/LREAL [▶ 700]
- STRING [▶ 700]
- WSTRING [▶ 701]
- TIME/LTIME [▶ 702]
- 日期和时间数据类型 [▶ 702]
- 特殊数据类型 XINT、UXINT、XWORD 和 PVOID [▶ 708]

IEC 61131-3 标准的扩展

另请参见：

- BIT [▶ 700]
- 指针 [▶ 708]
- 引用 [▶ 712]
- UNION [▶ 725]
- ANY 和 ANY <type> [▶ 704]
- 数据类型 __SYSTEM.ExceptionCode [▶ 710]

用户定义的数据类型

您可以根据默认的预定义数据类型或现有的数据类型来声明您自己的数据类型。

此类数据类型被称为用户定义或用户特定的数据类型。数据类型可作为单独的 DUT 对象进行组织，或者可在编程对象的声明部分中进行声明。根据它们的用途和语法，还可以对它们加以区分。

用户定义的数据类型	声明	另请参见
别名	DUT 对象	别名 [▶ 724]
数组	编程对象	数组 [▶ 714]
枚举	DUT 对象、编程对象	枚举 [▶ 721]
指针	编程对象	指针 [▶ 708]
Reference	编程对象	引用 [▶ 712]
结构	DUT 对象	结构 [▶ 719]
子范围类型	编程对象	子范围类型 [▶ 699]
联合	DUT 对象	UNION [▶ 725]

注意命名标识符的建议。

另请参见：

- 标识符 [▶ 782]

16.5.1 BOOL

数据类型	值	内存空间
BOOL	TRUE (1)、FALSE (0)	8 位

另请参见：

- BOOL 常量 [▶ 687]

16.5.2 整数数据类型

在 TwinCAT 中，以下整数数据类型可供使用。

数据类型	下限	上限	内存空间
BYTE	0	255	8 位
WORD	0	65535	16 位
DWORD	0	4294967295	32 位
LWORD	0	$2^{64}-1$	64 位
SINT	-128	127	8 位
USINT	0	255	8 位
INT	-32768	32767	16 位
UINT	0	65535	16 位
DINT	-2147483648	2147483647	32 位
UDINT	0	4294967295	32 位
LINT	-2^{63}	$2^{63}-1$	64 位
ULINT	0	$2^{64}-1$	64 位

将较大类型转换为较小类型时，可能会丢失信息。

另请参见：

- 数字常量 [▶ 687]

16.5.3 子范围类型

子范围类型是 1 种数据类型，其值范围仅涵盖基本类型的 1 个子集。
只有整数类型可以作为基本类型。

语法：<Name> : <Inttype> (<ug>..<og>)

<Name>	有效的 IEC 标识符
<Inttype>	子范围的数据类型 （SINT、USINT、INT、UINT、DINT、UDINT、BYTE、WORD、DWORD、LINT、ULINT、LWORD）。
<ug>	范围的下限：常量，必须与基本数据类型兼容。下限本身也包含在该范围内。
<og>	范围的上限：常量，必须与基本数据类型兼容。上限本身也包含在该范围内。

示例：

```
VAR
    nVarA: INT (-4095..4095);
    nVarB : UINT (0..10000);
END_VAR
```

如果您在声明或实现中为子范围类型分配 1 个值，而该值不在此范围内（例如，nVarA: = 5000），则 TwinCAT 会发出错误消息。

请注意，您可以选择使用隐式监控函数 CheckRangeSigned 和 CheckRangeUnsigned 在运行时监控子范围类型的字段边界。

另请参见：

- 范围/LRange 检查（POU：CheckRangeSigned、CheckRangeUnsigned、CheckLRangeSigned、CheckLRangeUnsigned） [▶ 155]

-

16.5.4 BIT

对于结构或功能块中的各个变量，您仅可使用数据类型 BIT。可能的值为 TRUE (1) 和 FALSE (0)。

BIT 元素需要 1 位内存空间，您可以使用它对使用其名称的结构或功能块的各个位进行寻址。按顺序声明的 BIT 元素可合并为字节。与 BOOL 类型（每个类型至少占用 8 位）相比，这样可以使您优化内存使用。不过，位访问所需的时间要长得多。因此，只有当您想要以指定格式定义数据时，才应使用数据类型 BIT。

另请参见：

- 结构 [▶ 719]
- 对象功能块 [▶ 76]

16.5.5 REAL/LREAL

数据类型 REAL 和 LREAL 是符合 IEEE 754 标准的浮点类型。对于在点表示法或指数表示法中使用十进制数和浮点数而言，它们必不可少。

数据类型	下限值	上限值	最小绝对值	存储空间
REAL	-3.402823e+38	3.402823e+38	1.0e-44	32 位
LREAL	-1.7976931348623158e+308	1.7976931348623158e+308	4.94065645841247e-324	64 位

示例

```
PROGRAM MAIN
VAR
    fMaxReal     : REAL  := 3.402823E+38;              // Largest REAL number
    fPosMinReal  : REAL  := 1.0E-44;                   // Smallest positive REAL number
    fNegMaxReal  : REAL  := -1.0E-44;                  // Largest negative REAL number
    fMinReal     : REAL  := -3.402823E+38;             // Smallest REAL number

    fMaxLreal    : LREAL := 1.7976931348623157E+308;   // Largest LREAL number
    fPosMinLreal : LREAL := 4.94065645841247E-324;     // Smallest positive LREAL number
    fNegMaxLreal : LREAL := -4.94065645841247E-324;    // Largest negative LREAL number
    fMinLreal    : LREAL := -1.7976931348623157E+308;  // Smallest LREAL number
END_VAR
```

如果 REAL/LREAL 数的值在整数值范围之外，则在将数据类型从 REAL 或 LREAL 转换为 SINT、USINT、INT、UINT、DINT、UDINT、LINT 或 ULINT 时会产生未定义的结果。

特大字面量的分配

为了分配 1 个大于 ULINT 上限的整数字面量，必须设置逗号或指定显式类型转换。如果没有浮点数这样的规范，则可能会丢失信息。
示例：

```
fMyReal : REAL := 3400000000000000000000.0;
fMyReal : REAL := REAL#340000000000000000000;
```

另请参见：

- REAL/LREAL 常量 [▶ 688]

16.5.6 STRING

将 STRING 数据类型解析为 Latin-1 或 UTF-8编码。

常见属性和区别如下。

STRING 数据类型的变量可以接受任意字符串。声明中的存储空间预留大小指的是字节数，用圆括号或方括号括起来。如果没有指定大小，则默认预留 80 字节。还要为最后一个终止符/0 预留 1 个字节。

通常情况下，TwinCAT 不会限制字符串长度，但字符串函数仅可处理长度在 1 至 255 个字符之间的字符串。如果初始化变量时所使用的字面值长度 超出了变量类型允许的最大长度，则会相应地从右侧截断字面值。

如果 STRING 数据类型的某一变量因应用程序重置而重新初始化，初始值最后一个终止符/0 字符之后的（旧）字符串内容不会被覆盖。此规则既适用于使用初始化值进行的初始化，也适用于使用默认初始化值 0 进行的初始化。

Latin-1

Latin-1 编码包括 1 个受限字符集，其中还包括所有 ASCII 字符。

使用 Latin-1 编码时，STRING 变量所需的存储空间始终为每个字符 1 字节外加 1 个附加字节。因此，以标准 STRING(80) 声明为例，需要 81 个字节。

示例：为 46 个字符预留 46+1 字节存储空间的字符串声明。

```
sVar : STRING(46) := 'This is a string with memory for 46 characters.';
```

UTF-8

UTF-8 编码 包含了最大的可能字符集。

Unicode 字符的 UTF-8 编码与使用 ASCII 字符的 Latin-1 编码相同。 所有超出部分的字符在转储中会有所不同。

在这种情况下，请在变量声明中添加编译指示 {attribute 'TcEncoding':='UTF-8'}。因此，变量的内容在监控中也会以UTF-8编码显示。如需更多详细信息，请参见“TcEncoding”属性的 [▶ 765]相关描述。

只能使用专门为 UTF-8 编码字符串发布的字符串处理函数。请参见 UTF8Len() 等。

使用 UTF-8 编码时，1 个字符的长度不得超过 4 个字节。但是，声明中规定的字符串最大长度始终是指为 STRING 预留的字节数。这意味着，字符的数量和所需的存储空间之间不再是 1:1 的关系。

示例：字符串声明时预留了46+1个字节的存储空间，可以存储最多46个字符。实际能存储的字符数量取决于字符本身的类型。

```
{attribute 'TcEncoding':='UTF-8'}
sVar : STRING(46) := 'This is a string with memory for up to 46 characters.';
```

另请参见：

- STRING 常量 [▶ 688]
- WSTRING [▶ 701]

16.5.7 WSTRING

根据 IEC 61131-3 标准，WSTRING 数据类型采用的是 UCS-2 编码。

字符串以带有双引号的 WSTRING 来进行标识。可以选择指定大小和初始化。如果未指定大小，则默认预留 80 个字符 (WORD)的空间。还要为存储结束符0 预留 1 个 WORD。

如果一个WSTRING类型的变量在应用程序重置时被重新初始化，初始值最后一个 0 字 (WORD) 之后的（旧）字符串的字节对不会被覆盖。此规则既适用于使用初始化值进行的初始化，也适用于使用默认初始化值 0 进行的初始化。

UCS-2

与 Latin-1 编码相比，UTCS-2 编码还包括 1 个扩展字符集，但它仍然比 UTF-8 编码小。另请参见 STRING [▶ 700] 数据类型的相关描述。

UCS-2 编码是指每个字符完全按照 2 字节的固定长度进行编码。UCS-2 包括 U+0000 至 U+D7FF 以及 U+E000 至 U+FFFF 的码位字符。
字符串以 0 结尾。

WSTRING 数据类型的每个字符需要 1 个 WORD，并且需要额外1个WORD来存储结束符0。例如，对于一个标准的WSTRING(80)声明，总共需要162个字节的内存空间。

示例：

```
wsVar : WSTRING := "This is a WString.";
```

另请参见：

- STRING [▶ 700]
- STRING 常量 [▶ 688]

16.5.8 TIME/LTIME

时间数据类型 TIME 的内部处理方式类似于 UDINT（32 位）。这会导致以毫秒为单位进行解析。

时间数据类型 LTIME 的内部处理方式类似于 ULINT（64 位）。您可以将该数据类型用作以纳秒为单位进行解析的高精度计时器的时基。

数据类型	下限值	上限值	存储空间	解析
TIME	0	4294967295 (49d17h2m47s295ms)	32 位	毫秒
LTIME	0	213503d23h34m33s709ms551us615ns	64 位	纳秒

时间声明可以包含对 TIME 或 LTIME 常量有效的时间单位。

示例：

```
VAR
    tTime  : TIME  := T#1d2h30m40s500ms
    tLTime : LTIME := LTIME# 100d2h30m40s500ms600us700ns
END_VAR
```

另请参见：

- TIME/LTIME 常量 [▶ 689]
- 日期和时间数据类型 [▶ 702]
- 日期和时间常量 [▶ 690]
- 日期和时间转换 [▶ 669]

16.5.9 日期和时间数据类型

DATE、DATE_AND_TIME (DT) 和 TIME_OF_DAY (TOD) 数据类型的内部处理类似于 UDINT（32 位值）。

数据类型	下限值	上限值	存储空间	解析
DATE	0 = D#1970-01-01 (01.01.1970)	4294967295 = D#2106-02-07 (07.02.2106)	32 位	秒，但仅显示日期。
DATE_AND_TIME DT	0 = DT#1970-1-1-0:0:0 (01.01.1970, 00:00 h)	4294967295 = DT#2106-02-07-06:28:15 (07.02.2106, 6:28:15)	32 位	秒钟
TIME_OF_DAY TOD	0 = TOD#0:0:0 (00:00:00:000 h)	86399999 = TOD#23:59:59.999 (23:59:59.999 h)	32 位	毫秒

LDATE、LDATE_AND_TIME (LDT) 和 LTIME_OF_DAY (LTOD) 的内部处理方式类似于 ULINT（64 位）。

数据类型	下限值	上限值	存储空间	解析
LDATE	0 = LD#1970-1-1 (01.01.1970)	2^{64}-1 = LD#2554-7-21 (21.07. 2554)	64 位	纳秒，但仅显示日期。
LDATE_AND_TIME LDT	0 = LDT#1970-1-1-0:0:0 (01.01.1970, 00:00 h)	2^{64}-1 = LDT#2554-7-21-23:34:33.709551615 (21.07.2554, 23:34:33.709551615 h)	64 位	纳秒
LTIME_OF_DAY LTOD	0 = LTOD#0:0:0 (00:00:00:000 h)	86399999999999 = LTOD#23:59:59.999999999 (23:59:59.999999999 h)	64 位	纳秒

要求

数据类型 LDATE、LDATE_AND_TIME (LDT) 和 LTIME_OF_DAY (LTOD) 需要 TwinCAT 3.1.4026.0 或以上版本。

示例：

```
VAR

    // Date
    dLowerLimit   : DATE           := DATE#1970-1-1;
    dUpperLimit   : DATE           := DATE#2106-2-7;
    dAppointment  : DATE           := D#2020-2-7;

    // Date and time
    dtLowerLimit  : DATE_AND_TIME  := DATE_AND_TIME#1970-1-1-0:0:0;
    dtUpperLimit  : DATE_AND_TIME  := DATE_AND_TIME#2106-02-07-06:28:15;
    dtAppointment : DT             := DT#2020-2-7-12:55:1.234;

    // Time of day
    tdLowerLimit  : TIME_OF_DAY    := TIME_OF_DAY#0:0:0;
    tdUpperLimit  : TIME_OF_DAY    := TIME_OF_DAY#23:59:59.999;
    tdAppointment : TOD            := TOD#12:3:4.567;

    // Long date
    dLowerLimit   : LDATE          := LDATE#1970-1-1;
    dUpperLimit   : LDATE          := LDATE#2106-2-7;
    dAppointment  : LDATE          := LD#2020-2-7;

    // Long date and time
    dtLowerLimit  : LDATE_AND_TIME := LDATE_AND_TIME#1970-1-1-0:0:0;
    dtUpperLimit  : LDATE_AND_TIME := LDATE_AND_TIME#2262-4-10-23:59:59.99999999;
    dtAppointment : LDT            := LDT#2020-2-7-12:55:1.234567891;

    // Long time of day
    tdLowerLimit  : LTIME_OF_DAY   := LTIME_OF_DAY#0:0:0;
    tdUpperLimit  : LTIME_OF_DAY   := LTIME_OF_DAY#23:59:59.999999999;
    tdAppointment : LTOD           := LTOD#12:3:4.567890123;

END_VAR
```

另请参见：

- 日期和时间常量 [▶ 690]
- TIME/LTIME [▶ 702]
- TIME/LTIME 常量 [▶ 689]
- DATE/DT_TO_<type>
- Tc2_Utilities PLC 库中的 FileTime 数据类型 T_FILETIME64
- Tc2_EtherCAT PLC 库中的 DC Time 数据类型 T_DCTIME64

16.5.10 ANY 和 ANY_<type>

在实现函数、方法或功能块（Build 4026 及以上版本）时，您可以将输入（VAR_INPUT）声明为具有通用 IEC 日期类型 ANY 或 ANY_<type> 的变量。因此，您可以实现调用参数因数据类型而异的调用。

在运行时，通过输入变量的编程块内的预定义结构，您可以查询传输的值及其数据类型。

编译器会在内部用下述数据结构替换输入变量的类型，但不能直接传输值。相反，传输的是指向实际值的指针，因此仅可传输 1 个变量。因此，数据类型仅在调用时才会具体化。因此，使用具有不同数据类型的各个参数可以进行此类编程块的调用。

在调用函数、功能块或方法时，不能为数据类型为 ANY 或 ANY_<type> 的输入分配任何常量或任何属性。反之，不能为属性分配数据类型为 ANY 或 ANY_<type> 的任何变量。

支持以下所示的通用 IEC 数据类型。该表格显示了哪些通用数据类型允许使用哪些基本数据类型。

通用数据类型			基本数据类型
ANY	ANY_BIT		• BYTE • WORD • DWORD • LWORD
	ANY_DATE		• DATE_AND_TIME、DT • DATE • TIME_OF_DAY、TOD • LDATE • LDATE_AND_TIME、LDT • LTIME_OF_DAY、LTOD
	ANY_NUM	ANY_REAL	• REAL • LREAL
		ANY_INT	• USINT • UINT • UDINT • ULINT • SINT • INT • DINT • LINT
	ANY_STRING		• STRING • WSTRING

带有“ANY”和“ANY_<type>”的内部数据结构

在编译代码时，在内部会用以下结构替换 ANY 数据类型的输入变量。在运行时将结构元素分配给实际调用参数。

```
TYPE AnyType :
STRUCT
      // the type of the actual parameter
  typeclass  : __SYSTEM.TYPE_CLASS ;
      // the pointer to the actual parameter
  pvalue     : POINTER TO BYTE;
      // the size of the data, to which the pointer points
  diSize     : DINT;
END_STRUCT
END_TYPE
```

通过该结构，您可以在编程块内访问输入变量，例如，查询传输值。

声明

语法描述指的是精确包含 1 个参数（1 个输入参数）的编程块。

语法

```
FUNCTION | FUNCTION_BLOCK | METHOD <POU name>
( : <return data type> )?
VAR_INPUT
    <input variable name> : <generic data type>;
END_VAR
<generic data type> = ANY | ANY_BIT | ANY_DATE |
ANY_NUM | ANY_REAL | ANY_INT | ANY_STRING
```

调用

语法描述指的是精确包含 1 个参数的编程块，向该编程块传输 1 个参数。参数的数据类型使输入变量的通用数据类型具体化。例如，可以将类型为 BYTE、WORD、DWORD、LWORD 的参数传输到 ANY_BIT 输入变量。

函数调用语法

```
<variable name> := <function name> ( <argument name> );
<argument name> : variable with valid data type
```

功能块调用语法

```
<function block name> ( <input variable name> := <argument name> );
```

方法调用语法

```
<function block name> . <method name> ( <input variable name> := <argument name> );
```

示例 1：将基本数据类型传输到具有通用数据类型的输入端

```
FUNCTION F_ComputeAny : BOOL
VAR_INPUT
    anyInput1         : ANY; // valid data type see table
END_VAR
```

```
FUNCTION_BLOCK FB_ComputeAny
VAR_INPUT
    anyInput1     : ANY;
END_VAR
```

```
FUNCTION_BLOCK FB_ComputeMethod
METHOD ComputeAny : BOOL
VAR_INPUT
    anyInput1     : ANY_INT; // valid data types are SINT, INT, DINT,INT, USINT, UINT, UDINT, ULINT
END_VAR
```

```
PROGRAM PLC_PRG
VAR
    fbComputeAnyByte  : FB_ComputeAny;
    fbComputeAnyInt   : FB_ComputeAny;

    fbComputeM1       : FB_ComputeMethod;
    fbComputeM2       : FB_ComputeMethod;

    nByte             : BYTE := 16#AB;
    nInt              : INT  := -1234;
    bResultByte       : BOOL;
    bResultInt        : BOOL;
END_VAR
```

```
bResultByte := F_ComputeAny(nByte);
bResultInt  := F_ComputeAny(nInt);

fbComputeAnyByte(anyInput1 := nByte);
fbComputeAnyInt(anyInput1 := nInt);

fbComputeM1.methComputeAny(anyInput1 := nByte);
fbComputeM2.methComputeAny(anyInput1 := nInt);
```

示例 2：将基本数据类型传输到具有通用数据类型的输入端

函数调用的传输参数具有不同的数据类型。

```
FUNCTION F_AnyBitFunc    : BOOL
VAR_INPUT
    value : ANY_BIT;
```

```
END_VAR

FUNCTION F_AnyDateFunc   : BOOL
VAR_INPUT
    value : ANY_DATE;
END_VAR

FUNCTION F_AnyFunc       : BOOL
VAR_INPUT
    value : ANY;
END_VAR

FUNCTION F_AnyIntFunc    : BOOL
VAR_INPUT
    value : ANY_INT;
END_VAR

FUNCTION F_AnyNumFunc    : BOOL
VAR_INPUT
    value : ANY_NUM;
END_VAR

FUNCTION F_AnyRealFunc   : BOOL
VAR_INPUT
    value : ANY_REAL;
END_VAR

FUNCTION F_AnyStringFunc : BOOL
VAR_INPUT
    value : ANY_STRING;
END_VAR
```

```
PROGRAM MAIN
VAR
    bBOOL            : BOOL           := TRUE;
    nBYTE            : BYTE           := 16#AB;
    nWORD            : WORD           := 16#1234;
    nDWORD           : DWORD          := 16#6789ABCD;
    nLWORD           : LWORD          := 16#0123456789ABCDEF;
    sSTRING          : STRING         := 'xyz';
    wsWSTRING        : WSTRING        := "abc";
    dtDATEANDTIME    : DATE_AND_TIME  := DT#2017-02-20-11:07:00;
    dDATE            : DATE           := D#2017-02-20;
    tdTIMEOFDAY      : TIME_OF_DAY    := TOD#11:07:00;
    fREAL            : REAL           := 42.24;
    fLREAL           : LREAL          := 24.42;
    nUSINT           : USINT          := 12;
    nUINT            : UINT           := 1234;
    nUDINT           : UDINT          := 12345;
    nULINT           : ULINT          := 123456;
    nSINT            : SINT           := -12;
    nINT             : INT            := -1234;
    nDINT            : DINT           := -12345;
    nLINT            : LINT           := -123456;
END_VAR
```

```
F_AnyFunc(bBOOL);
F_AnyFunc(nBYTE);
F_AnyFunc(nWORD);
F_AnyFunc(nDWORD);
F_AnyFunc(nLWORD);
F_AnyFunc(sSTRING);
F_AnyFunc(wsWSTRING);
F_AnyFunc(dtDATEANDTIME);
F_AnyFunc(tdTIMEOFDAY);
F_AnyFunc(fREAL);
F_AnyFunc(fLREAL);
F_AnyFunc(nUSINT);
F_AnyFunc(nUINT);
F_AnyFunc(nUDINT);
F_AnyFunc(nULINT);
F_AnyFunc(nSINT);
F_AnyFunc(nINT);
F_AnyFunc(nDINT);
F_AnyFunc(nLINT);

F_AnyBitFunc(nBYTE);
F_AnyBitFunc(nWORD);
F_AnyBitFunc(nDWORD);
F_AnyBitFunc(nLWORD);
```

```
F_AnyStringFunc(sSTRING);
F_AnyStringFunc(wsWSTRING);

F_AnyDateFunc(dtDATEANDTIME);
F_AnyDateFunc(dDATE);
F_AnyDateFunc(tdTIMEOFDAY);

F_AnyNumFunc(fREAL);
F_AnyNumFunc(fLREAL);
F_AnyNumFunc(nUSINT);
F_AnyNumFunc(nUINT);
F_AnyNumFunc(nUDINT);
F_AnyNumFunc(nULINT);
F_AnyNumFunc(nSINT);
F_AnyNumFunc(nINT);
F_AnyNumFunc(nDINT);
F_AnyNumFunc(nLINT);

F_AnyRealFunc(fREAL);
F_AnyRealFunc(fLREAL);

F_AnyIntFunc(nUSINT);
F_AnyIntFunc(nUINT);
F_AnyIntFunc(nUDINT);
F_AnyIntFunc(nULINT);
F_AnyIntFunc(nSINT);
F_AnyIntFunc(nINT);
F_AnyIntFunc(nDINT);
F_AnyIntFunc(nLINT);
```

示例 3：比较 2 个传输变量

该函数对 2 个传输变量进行比较，以确定它们是否属于同一个类型且具有相同的值。

```
FUNCTION F_GenericCompare : BOOL
VAR_INPUT
    any1 : ANY;
    any2 : ANY;
END_VAR
VAR
    nCount: DINT;
END_VAR
```

```
IF any1.typeclass <> any2.typeclass THEN
   RETURN;
END_IF

IF any1.diSize <> any2.diSize THEN
    RETURN;
END_IF

// Byte comparison
FOR nCount := 0 TO any1.diSize-1 DO
    IF any1.pvalue[nCount] <> any2.pvalue[nCount] THEN
        RETURN;
    END_IF
END_FOR

F_GenericCompare := TRUE;
```

示例 4：确定传输的数据类型

该函数检查传输的变量是 REAL 类型，还是 LREAL 类型。如果是这种情况，则会将变量的值四舍五入。

```
// function to round transfer parameters of the type REAl and LREAL (other types are detected as
invalid)
FUNCTION F_RoundFloatingValue : INT
VAR_INPUT
    anyIn        : ANY;              // input variable of the type ANY
END_VAR
VAR
    pAnyReal     : POINTER TO REAL;  // pointer to a variable of the type REAL
    pAnyLReal    : POINTER TO LREAL; // pointer to a variable of the type LREAL
END_VAR
VAR_OUTPUT
    bInvalidType : BOOL;             // output variable with value TRUE if the transferred
parameter has an invalid type
END_VAR
```

```
// round floating value for a transfer parameter of the type REAL
IF (anyIn.TypeClass = __SYSTEM.TYPE_CLASS.TYPE_REAL) THEN
    pAnyReal              := anyIn.pValue;
    F_RoundFloatingValue  := REAL_TO_INT(pAnyReal^);

// round floating value for a transfer parameter of the type LREAL
ELSIF (anyIn.TypeClass = __SYSTEM.TYPE_CLASS.TYPE_LREAL) THEN
    pAnyLReal             := anyIn.pValue;
    F_RoundFloatingValue  := LREAL_TO_INT(pAnyLReal^);

// inform about invalid type if the transfer parameter is not of the type REAL or LREAL
ELSE
    bInvalidType          := TRUE;
END_IF
```

16.5.11 特殊数据类型 XINT、UXINT、XWORD 和 PVOID

根据目标系统的不同，具有这些数据类型的变量会被转换为与平台兼容的数据类型。

TwinCAT 支持地址寄存器宽度在 32 至 64 位之间的系统。为了使 IEC 代码尽可能独立于目标系统，您可以使用下面列出的“伪”数据类型 UXINT、XINT、XWORD 和 PVOID。编译器会检查当前使用的目标系统类型，并将这些数据类型转换为相应的标准数据类型。此外，类型转换运算符可用于该数据类型的变量。

以下“伪”数据类型可供使用：

	64 位平台上的类型转换	32 位平台上的类型转换
XINT 或 __XINT	LINT	DINT
UXINT 或 __UXINT	ULINT	UDINT
XWORD 或 __XWORD	LWORD	DWORD
PVOID	UXINT	

16.5.12 指针

在运行时，指针会保存诸如变量或功能块实例等对象的内存地址。

语法

```
<pointer name>: POINTER TO <data type> | <data unit type> | <function block name>;
```

示例

```
FUNCTION_BLOCK FB_Sample
VAR
    pSample : POINTER TO INT;
    nVar1 : INT := 5;
    nVar2 : INT;
END_VAR
```

```
pSample := ADR(nVar1);  // pointer pSample is assigned to address of nVar1
nVar2 := pSample^;      //
value 5 of nVar1 is assigned to variable nVar2 by dereferencing of pointer pSample
```

解除引用指针意味着获取指针所指向的值。为指针标识符附加内容运算符 ^ 即可解除引用指针，例如，在上面的示例中的 pSample^。如要将对象的地址分配给指针，可使用地址运算符 ADR：ADR(nVar1)。

在在线模式下，您可以使用转至引用 [▶ 816]命令，从指针跳转到引用对象的声明位置（TC3.1 Build 4026 及以上版本可用）。

如果使用指向已分配的输入变量的指针，则访问会被解释为写入访问。在生成代码期间，这会导致编译器发出“<Pointer Name> 不是有效赋值目标”的警告。

示例：pTest := ADR(nInput);

如果您需要该类型的结构，则您必须首先将输入值（nInput）复制到具有写入访问权限的变量中。

对指针的索引访问

TwinCAT 允许对 POINTER 类型的变量以及 STRING 或 WSTRING 数据类型的变量进行索引访问 [ ]。

为指针标识符附加括号运算符 [] 也可访问指针所指向的数据，例如 pData[i]。指针的基本数据类型决定了索引组件的数据类型和大小。通过在指针地址上添加与索引相关的偏移量 i * SIZEOF(<base type>)，以算数方式对指针进行索引访问。同时，隐式解除引用指针。计算方式：pData[i] :=(pData + i * SIZEOF(INT))^;

对 STRING 的索引访问

如果您对 STRING 类型的变量使用索引访问，则您会获得在索引表达式偏移量处的字符。结果的类型为 BYTE。例如，sData[i] 会以 SINT（ASCII）格式返回字符串 sData 中的第 i 个字符。

索引访问 WSTRING

如果您对 WSTRING 类型的变量使用索引访问，则您会获得在索引表达式偏移量处的字符。结果的类型为 WORD。例如，wsData[i] 会以 INT（Unicode）格式返回字符串中的第 i 个字符。

减法指针

如果指针是 64 位指针，那么，即使在 64 位平台上，2 个指针之间的差值结果也是 DWORD 类型的值。

请注意使用引用的可能性。使用引用的优点是，类型安全可以得到保证。指针则不是这样。

通过隐式监控函数 CheckPointer，可在运行时检查指针的内存访问。

在在线更改期间，自动更新指针/引用

TwinCAT 3.1 Build 4026 及以上可用

以下描述涉及指针和引用。不过，为了便于阅读，下面仅使用指针一词。

功能：

在在线更改期间，所有 PLC 指针的值均会自动更新，以便相关指针可以指向在线更改之前的同一个变量或同一个对象。这意味着，即使指针指向的变量在在线更改期间被移动到不同的内存位置，指针在在线更改后也仍然有效。

运行模式：

在在线更改期间会检查每个指针是否指向 PLC 符号。为了确定此类情况，必须满足以下要求。

- 情况 1：确定符号
  - 如果该符号仍然存在于在线更改后存在的新符号中，并且该符号的类型没有改变，则指针将被移动到已找到的符号上。指针或地址值已更新。
  - 如果该符号在新符号中不再存在，或者如果符号的类型已改变，则指针将被设置为零。
- 情况 2：未确定符号
  - 如果指针指向的地址在在线更改之前没有找到任何符号（例如，由于指针指向 PLC 外部的内存区），则指针值在在线更改期间不会改变。

要求：

- 该函数需要指针所指向的变量的（ADS）符号描述。如果没有变量的符号描述，例如，由于使用 属性 “hide” [▶ 744]，则不考虑指针。
- 指针必须指向 PLC 内存中的变量/对象。在在线更改期间，指向 PLC 内存外部的内存区的指针不会改变。

附加更新选项：

所述功能在 TwinCAT 3.1 Build 4026 及以上版本中可用。任何先前用于更新指针的手动实现机制仍然可以使用。例如，可以将所需的目标周期性地分配给指针。无需从项目中删除这些代码行。同样，也没有必要保留或实现这些代码行。

另请参见：

- 引用 [▶ 712]
- 指针检查（POU CheckPointer） [▶ 156]

16.5.13 数据类型 __SYSTEM.ExceptionCode

表示异常代码的 IEC 变量的数据类型为 __SYSTEM.ExceptionCode。例如，__TRY、__CATCH、__FINALLY、__ENDTRY [▶ 679] 就能捕获此类异常，可将其用于有针对性地处理异常。

例如，还可以使用 ExceptionCode 数据类型对应用程序代码本身出现的错误或异常进行分类，然后记录这些错误或异常等。请参见本页底部的示例。

数据类型 __SYSTEM.ExceptionCode：

```
TYPE ExceptionCode :
(
    RTSEXCPT_UNKNOWN                                := 16#FFFFFFFF,
    RTSEXCPT_NOEXCEPTION                            := 16#00000000,
    RTSEXCPT_WATCHDOG                               := 16#00000010,
    RTSEXCPT_HARDWAREWATCHDOG                       := 16#00000011,
    RTSEXCPT_IO_CONFIG_ERROR                        := 16#00000012,
    RTSEXCPT_PROGRAMCHECKSUM                        := 16#00000013,
    RTSEXCPT_FIELDBUS_ERROR                         := 16#00000014,
    RTSEXCPT_IOUPDATE_ERROR                         := 16#00000015,
    RTSEXCPT_CYCLE_TIME_EXCEED                      := 16#00000016,
    RTSEXCPT_ONLCHANGE_PROGRAM_EXCEEDED             := 16#00000017,
    RTSEXCPT_UNRESOLVED_EXTREFS                     := 16#00000018,
    RTSEXCPT_DOWNLOAD_REJECTED                      := 16#00000019,
    RTSEXCPT_BOOTPROJECT_REJECTED_DUE_RETAIN_ERROR  := 16#0000001A,
    RTSEXCPT_LOADBOOTPROJECT_FAILED                 := 16#0000001B,
    RTSEXCPT_OUT_OF_MEMORY                          := 16#0000001C,
    RTSEXCPT_RETAIN_MEMORY_ERROR                    := 16#0000001D,
    RTSEXCPT_BOOTPROJECT_CRASH                      := 16#0000001E,
    RTSEXCPT_BOOTPROJECTTARGETMISMATCH              := 16#00000021,
    RTSEXCPT_SCHEDULEERROR                          := 16#00000022,
    RTSEXCPT_FILE_CHECKSUM_ERR                      := 16#00000023,
    RTSEXCPT_RETAIN_IDENTITY_MISMATCH               := 16#00000024,
    RTSEXCPT_IEC_TASK_CONFIG_ERROR                  := 16#00000025,
    RTSEXCPT_APP_TARGET_MISMATCH                    := 16#00000026,
    RTSEXCPT_ILLEGAL_INSTRUCTION                    := 16#00000050,
    RTSEXCPT_ACCESS_VIOLATION                       := 16#00000051,
    RTSEXCPT_PRIV_INSTRUCTION                       := 16#00000052,
    RTSEXCPT_IN_PAGE_ERROR                          := 16#00000053,
    RTSEXCPT_STACK_OVERFLOW                         := 16#00000054,
    RTSEXCPT_INVALID_DISPOSITION                    := 16#00000055,
    RTSEXCPT_INVALID_HANDLE                         := 16#00000056,
    RTSEXCPT_GUARD_PAGE                             := 16#00000057,
    RTSEXCPT_DOUBLE_FAULT                           := 16#00000058,
    RTSEXCPT_INVALID_OPCODE                         := 16#00000059,
    RTSEXCPT_MISALIGNMENT                           := 16#00000100,
    RTSEXCPT_ARRAYBOUNDS                            := 16#00000101,
    RTSEXCPT_DIVIDEBYZERO                           := 16#00000102,
    RTSEXCPT_OVERFLOW                               := 16#00000103,
    RTSEXCPT_NONCONTINUABLE                         := 16#00000104,
    RTSEXCPT_PROCESSORLOAD_WATCHDOG                 := 16#00000105,
    RTSEXCPT_FPU_ERROR                              := 16#00000150,
    RTSEXCPT_FPU_DENORMAL_OPERAND                   := 16#00000151,
    RTSEXCPT_FPU_DIVIDEBYZERO                       := 16#00000152,
    RTSEXCPT_FPU_INEXACT_RESULT                     := 16#00000153,
    RTSEXCPT_FPU_INVALID_OPERATION                  := 16#00000154,
    RTSEXCPT_FPU_OVERFLOW                           := 16#00000155,
    RTSEXCPT_FPU_STACK_CHECK                        := 16#00000156,
    RTSEXCPT_FPU_UNDERFLOW                          := 16#00000157,
    RTSEXCPT_VENDOR_EXCEPTION_BASE                  := 16#00002000,
    RTSEXCPT_USER_EXCEPTION_BASE                    := 16#00010000
) UDINT ;
END_TYPE
```

错误/异常分类的实现示例

全局变量列表“GVL_Exc”：

```
VAR_GLOBAL
    nCheckBounds               : INT;
    nCheckBounds_OutOfBounds   : INT;
    myException                : __SYSTEM.ExceptionCode;
END_VAR
```

“CheckBounds”h函数：

```
// Implicitly generated code : DO NOT EDIT
FUNCTION CheckBounds : DINT
VAR_INPUT
    index, lower, upper        : DINT;
END_VAR
```

```
// Only an implementation suggestion
{noflow}
GVL_Exc.nCheckBounds := GVL_Exc.nCheckBounds + 1;

// Index too low
IF index < lower THEN
    CheckBounds                       := lower;
    GVL_Exc.myException               := __SYSTEM.ExceptionCode.RTSEXCPT_ARRAYBOUNDS;
    GVL_Exc.nCheckBounds_OutOfBounds  := GVL_Exc.nCheckBounds_OutOfBounds + 1;

// Index too high
ELSIF index > upper THEN
    CheckBounds                       := upper;
    GVL_Exc.myException               := __SYSTEM.ExceptionCode.RTSEXCPT_ARRAYBOUNDS;
    GVL_Exc.nCheckBounds_OutOfBounds  := GVL_Exc.nCheckBounds_OutOfBounds + 1;

// Index OK
ELSE
    CheckBounds := index;
END_IF
```

另请参见：

- 边界检查（POU CheckBounds） [▶ 153]

16.5.14 接口指针 / 接口

接口指针（也被称为接口变量）可存储实现接口的函数表或功能块实例的地址。通过这种间接方式，使用接口指针可以调用由接口定义的所有方法和属性。因此，这些方法和属性必须由其实例被分配给接口指针的功能块来实现。

在使用之前，必须始终将实现相应接口的功能块实例分配给接口指针。

建议在使用之前检查接口指针是否有效（<> 0）。

语法：

```
<Kennzeichner> : <Interfacetyp>;
```

```
FUNCTION_BLOCK <Funktionsbausteintyp> IMPLEMENTS <Interfacetyp>
```

示例：

```
FUNCTION_BLOCK FB_Sample IMPLEMENTS I_Sample
```

```
VAR
    ipSample : I_Sample;
    fbSample : FB_Sample;
    nResult  : INT;
END_VAR
```

```
ipSample := fbSample;

// calling a method of this interface
IF ipSample <> 0 THEN
    nResult := ipSample.Add(3, 6); // result is 9
END_IF
```

有关使用接口和接口指针的详细信息，请参见面向对象的编程 > 对象接口 [▶ 94] 部分。

使用运算符 __QUERYINTERFACE() [▶ 677] 可以进行类型转换。

16.5.15 引用

引用隐式指向另一个对象。在访问期间对引用进行隐式解除引用，因此不需要像指针那样使用特殊内容运算符 ^。

语法

```
<identifier> : REFERENCE TO <data type> ;
<data type>  : base type of the reference
```

示例声明：

```
PROGRAM_MAIN
VAR
    refInt  : REFERENCE TO INT;
    nA      : INT;
    nB      : INT;
END_VAR
```

现在，您可以使用 refInt 作为 INT 类型变量的“别名”。

赋值：

您必须借助赋值运算符 REF= [▶ 712]，通过单独的赋值运算来设置引用的地址。除非输入为 REFERENCE TO 且在调用过程中传输该输入。在这种情况下，可使用普通分配运算符 := 代替分配运算符 REF=。

```
FUNCTION_BLOCK FB_Sample
VAR_INPUT
    refInput1  : REFERENCE TO INT;
    refInput2  : REFERENCE TO INT;
END_VAR
```

```
PROGRAM MAIN
VAR
    fbSample   : FB_Sample;
    n1         : INT;
    n2         : INT;
END_VAR
```

```
fbSample.refInput1 REF= n1;
fbSample(refInput2   := n2);
```

借助特殊运算符，您可以检查引用是否指向有效值（例如，不等于 0）（请参见“检查引用的有效性 [▶ 713]”）。

应用示例：

```
refInt REF= nA;         // refInt points now to nA
refInt   := 12;         // nA has got the value 12
nB       := refInt * 2; // nB has got the value 24
refInt REF= nB;         // refInt points now to nB
refInt   := nA / 2;     // nB has got the value 6
refInt REF= 0;          // explicit initialisation of the reference
```

TwinCAT 对引用进行初始化（为 0）。

分配运算符 REF=

运算符生成 1 个指向值的引用 [▶ 712]（指针）。

语法：

<variable name> REF= <variable name>

示例：

```
PROGRAM MAIN
VAR
    refA  : REFERENCE TO ST_Sample;
    stA   : ST_Sample;
    refB  : REFERENCE TO ST_Sample;
    stB1  : ST_Sample;
    stB2  : ST_Sample;
END_VAR
```

```
refA REF= stA;   // represents => refA  := ADR(stA);
refB REF= stB1;  // represents => refB  := ADR(stB1);
refA := refB;    // represents => refA^ := refB^; (value assignment of refB as refA and refB are
implicitly dereferenced)
refB := stB2;    // represents => refB^ := stB2; (value assignment of stB2 as refB is implicitly
dereferenced)
END_VAR
```

无效声明

```
PROGRAM MAIN
VAR
     aTest      : ARRAY[0..9] OF REFERENCE TO INT;
     pTest      : POINTER TO REFERENCE TO INT;
     refTestRef : REFERENCE TO REFERENCE TO INT;
     refTestBit : REFERENCE TO BIT;
END_VAR
```

引用类型不可用作数组、指针或引用的基本类型。此外，引用不可指向位变量。此类结构会产生编译器报错。

引用和指针的比较

与指针相比，引用具有以下优点：

- 更易于使用：
  引用可以直接访问被引用对象的内容，而无需解除引用。
- 语法在传输值时更简洁和易懂的语法：
  调用功能块时，该功能块可传输不使用地址操作符的引用，直接传递变量本身，而不是传递变量的地址。
  示例：FB_Test1(refInput := nValue);
  而无需： FB_Test2(pInput := ADR(nValue));
- 类型安全：
  在分配 2 个引用时，编译器会检查基本类型是否匹配。如果是指针，则不会检查这一点。

检查引用的有效性

您可以使用运算符 __ISVALIDREF 检查引用是否指向有效值，即 1 个不等于 0 的值。

语法：

```
<boolesche Variable> := __ISVALIDREF(<mit REFERENCE TO <datatype> deklarierter Kennzeichner>);
```

如果引用指向 1 个有效值，则 Boolean 变量变为 TRUE，否则变为 FALSE。

示例

```
PROGRAM_MAIN
VAR
    nVar      : INT;
    refInt1   : REFERENCE TO INT;
    refInt2   : REFERENCE TO INT;
    bTestRef1 : BOOL := FALSE;
    bTestRef2 : BOOL := FALSE;
END_VAR
```

```
nVar       := nVar + 1;
refInt1  REF= nVar;
refInt2  REF= 0;
bTestRef1  := __ISVALIDREF(refInt1); (* becomes TRUE, because refInt1 points to nVar, which is non-
zero *)
bTestRef2  := __ISVALIDREF(refInt2); (* becomes FALSE, because refInt2 is set to 0 *)
```

隐式监控函数 Checkpointer 对 REFERENCE 类型变量的作用与指针变量相同。

在在线更改期间，自动更新指针/引用

TwinCAT 3.1 Build 4026 及以上可用

以下描述涉及指针和引用。不过，为了便于阅读，下面仅使用指针一词。

功能：

在在线更改期间，所有 PLC 指针的值均会自动更新，以便相关指针可以指向在线更改之前的同一个变量或同一个对象。这意味着，即使指针指向的变量在在线更改期间被移动到不同的内存位置，指针在在线更改后也仍然有效。

运行模式：

在在线更改期间会检查每个指针是否指向 PLC 符号。为了确定此类情况，必须满足以下要求。

- 情况 1：确定符号
  - 如果该符号仍然存在于在线更改后存在的新符号中，并且该符号的类型没有改变，则指针将被移动到已找到的符号上。指针或地址值已更新。
  - 如果该符号在新符号中不再存在，或者如果符号的类型已改变，则指针将被设置为零。
- 情况 2：未确定符号
  - 如果指针指向的地址在在线更改之前没有找到任何符号（例如，由于指针指向 PLC 外部的内存区），则指针值在在线更改期间不会改变。

要求：

- 该函数需要指针所指向的变量的（ADS）符号描述。如果没有变量的符号描述，例如，由于使用 属性 “hide” [▶ 744]，则不考虑指针。
- 指针必须指向 PLC 内存中的变量/对象。在在线更改期间，指向 PLC 内存外部的内存区的指针不会改变。

附加更新选项：

所述功能在 TwinCAT 3.1 Build 4026 及以上版本中可用。任何先前用于更新指针的手动实现机制仍然可以使用。例如，可以将所需的目标周期性地分配给指针。无需从项目中删除这些代码行。同样，也没有必要保留或实现这些代码行。

另请参见：

- 指针检查（POU CheckPointer） [▶ 156]

16.5.16 数组

数组是相同数据类型的数据元素的集合。TwinCAT 支持长度固定或可变的一维和多维数组。

16.5.16.1 固定长度的数组

可在 POU 的声明部分或全局变量列表中对数组进行声明。

声明一维数组的语法

```
<variable name> : ARRAY[ <dimension> ] OF <data type> ( := <initialization> )? ;
```

```
<dimension> : <lower index bound>..<upper index bound>
<data type> : elementary data types | user defined data types | function block types
// (...)? : Optional
```

声明多维数组的语法

```
<variable name> : ARRAY[ <1st dimension> ( , <next dimension> )
+ ] OF <data type> ( := <initialization> )? ;
```

```
<1st dimension> : <1st lower index bound>..<1st upper index bound>
<next dimension> : <next lower index bound>..<next upper index bound>
<data type> : elementary data types | user defined data types | function block types
// (...)+ : One or more further dimensions
// (...)? : Optional
```

索引限值为整数，最大数据类型为 DINT。

数据访问的语法

```
<variable name>[ <index of 1st dimension> ( , <index of next dimension> )* ]
// (...)* : 0, one or more further dimensions
```

请注意，您可以选择使用隐式监控函数 CheckBounds 在运行时监控数组越界的情况。

另请参见：

- 边界检查（POU CheckBounds） [▶ 153]

设置大型数组的监控范围

如果 1 个数组有超过 1000 个元素，在线视图中会默认显示 1000 个元素。通过双击在线视图声明部分中的数组声明，可以更改该监控范围。然后，系统会打开 1 个对话框，您可以在其中设置所需监控范围的开始和结束索引。

请注意，显示质量会随着同时显示的元素数量的增加（最多 20,000 个元素）而下降。

示例：一维数组

声明 1：

由 10 个整数元素组成的一维数组

索引下限：0
索引上限：9

```
VAR
    aCounter : ARRAY[0..9] OF INT;
END_VAR
```

初始化 1：

```
aCounter : ARRAY[0..9] OF INT := [0, 10, 20, 30, 40, 50, 60, 70, 80, 90];
```

数据访问 1：

将值 20 分配给局部变量。

```
nLocalVariable := aCounter[2];
```

声明 2：

```
VAR CONSTANT
    cMin    : INT := 0;
    cMax    : INT := 5;
END_VAR
VAR
    aSample : ARRAY [cMin..cMax] OF BOOL;
END_VAR
```

数据访问 2：

通过索引变量访问数组。在访问数组之前，系统会检查索引变量的值是否在有效的数组边界之内。

```
IF nIndex >= cMin AND nIndex <= cMax THEN
    bValue := aSample[nIndex];
END_IF
```

示例：二维数组

声明：

第 1 个维度：1 至 2
第 2 个维度：3 至 4

```
VAR
    aCardGame : ARRAY[1..2, 3..4] OF INT;
END_VAR
```

初始化：

```
aCardGame : ARRAY[1..2, 3..4] OF INT := [2(10),2(20)]; // Short notation for [10, 10, 20, 20]
```

数据访问：

```
nLocal1 := aCardGame[1, 3]; // Assignment of 10
nLocal2 := aCardGame[2, 4]; // Assignment of 20
```

示例：三维数组

声明：

第 1 个维度：1 至 2
第 2 个维度：3 至 4
第 3 个维度：5 至 6
2 * 2 * 2 = 8 个数组元素

```
VAR
    aCardGame : ARRAY[1..2, 3..4, 5..6] OF INT;
END_VAR
```

初始化 1：

```
aCardGame : ARRAY[1..2, 3..4, 5..6] OF INT := [10, 20, 30, 40, 50, 60, 70, 80];
```

数据访问 1：

```
nLocal1 := aCardGame[1, 3, 5]; // Assignment of 10
nLocal2 := aCardGame[1, 3, 6]; // Assignment of 20
nLocal3 := aCardGame[1, 4, 5]; // Assignment of 30
nLocal4 := aCardGame[1, 4, 6]; // Assignment of 40
nLocal5 := aCardGame[2, 3, 5]; // Assignment of 50
nLocal6 := aCardGame[2, 3, 6]; // Assignment of 60
nLocal7 := aCardGame[2, 4, 5]; // Assignment of 70
nLocal8 := aCardGame[2, 4, 6]; // Assignment of 80
```

初始化 2：

```
aCardGame : ARRAY[1..2, 3..4, 5..6] OF INT := [2(10), 2(20), 2(30), 2(40)]; // Short notation for [1
0, 10, 20, 20, 30, 30, 40, 40]
```

数据访问 2：

```
nLocal1 := aCardGame[1, 3, 5]; // Assignment of 10
nLocal2 := aCardGame[1, 3, 6]; // Assignment of 10
nLocal3 := aCardGame[1, 4, 5]; // Assignment of 20
nLocal4 := aCardGame[1, 4, 6]; // Assignment of 20
nLocal5 := aCardGame[2, 3, 5]; // Assignment of 30
nLocal6 := aCardGame[2, 3, 6]; // Assignment of 30
nLocal7 := aCardGame[2, 4, 5]; // Assignment of 40
nLocal8 := aCardGame[2, 4, 6]; // Assignment of 40
```

示例：用户自定义结构类型的三维数组

声明：

数组 aData 总共由 3 * 3 * 10 = 90 个数据类型为 ST_Data 的数组元素组成。

```
TYPE ST_Data
STRUCT
    n1 : INT;
    n2 : INT;
    n3 : DWORD;
END_STRUCT
END_TYPE
```

```
PROGRAM MAIN
VAR
    aData : ARRAY[1..3, 1..3, 1..10] OF ST_Data;
END_VAR
```

部分初始化：

在本示例中，只有前 3 个元素进行了显式初始化。没有显式分配初始化值的元素则使用基本数据类型的默认值进行内部初始化。因此，从元素 aData[2, 1, 1] 开始，使用 0 对结构组件进行初始化。

```
aData : ARRAY[1..3, 1..3, 1..10] OF ST_Data := [(n1 := 1, n2 := 10, n3 := 16#00FF),
                                                (n1 := 2, n2 := 20, n3 := 16#FF00),
                                                (n1 := 3, n2 := 30, n3 := 16#FFFF)];
```

数据访问：

```
nLocal1 := aData[1,1,1].n1; // Assignment of 1
nLocal2 := aData[1,1,3].n3; // Assignment of 16#FFFF
```

示例：功能块的数组

声明：

数组 aObjects 由 4 个元素组成。每个元素实例化 1 个功能块 FB_Object。

```
FUNCTION BLOCK FB_Object
VAR
    nCounter : INT;
END_VAR
```

```
PROGRAM MAIN
VAR
    aObjects : ARRAY[1..4] OF FB_Object;
END_VAR
```

函数调用：

```
aObjects[2]();
```

示例：带有 FB_init 方法的功能块的数组

声明：

使用 FB_init 方法实现 FB_Sample

```
FUNCTION_BLOCK FB_Sample
VAR
    _nId : INT;
    _fIn : LREAL;
END_VAR
```

功能块 FB_Sample 有 1 个 FB_init 方法，该方法需要 2 个参数。

```
METHOD FB_init : BOOL
VAR_INPUT
    bInitRetains : BOOL;
    bInCopyCode  : BOOL;
    nId          : INT;
    fIn          : LREAL;
END_VAR
```

```
_nId := nId;
_fIn := fIN;
```

带初始化的数组声明：

```
PROGRAM MAIN
VAR
    fbSample : FB_Sample (nId := 11, fIn := 33.44);
    aSample  : ARRAY[0..1, 0..1] OF FB_Sample [(nId := 12, fIn := 11.22),
                                               (nId := 13, fIn := 22.33),
                                               (nId := 14, fIn := 33.55),
                                               (nId := 15, fIn := 11.22)];
END_VAR
```

16.5.16.2 数组的数组

“数组的数组”声明是多维数组的另一种表示方法。不再声明元素的大小，而是对元素的集合进行嵌套。任何嵌套深度均可使用。

声明的语法

```
<variable name> : ARRAY[<first>] ( OF ARRAY[<next>] )+ OF <data type> ( := <initialization> )? ;
```

```
<first> : <first lower index bound>..<first upper index bound>
<next> : <lower index bound>..<upper index bound> // one or more arrays
<data type> : elementary data types | user defined data types | function block types
// (...)+ : One or more further arrays
// (...)? : Optional
```

数据访问的语法

```
<variable name>[<index of first array>] ( [<index of next array>] )+ ;
```

```
// (...)* : 0, one or more further arrays
```

示例

变量 aiPoints 和 ai2Boxes 将相同的数据元素组合在一起，但是声明和数据访问的拼写不同。

```
PROGRAM MAIN
VAR
    aPoints : ARRAY[1..2,1..3] OF INT := [1,2,3,4,5,6];
    a2Boxes : ARRAY[1..2] OF ARRAY[1..3] OF INT := [ [1, 2, 3], [ 4, 5, 6]];
    a3Boxes : ARRAY[1..2] OF ARRAY[1..3] OF ARRAY[1..4] OF INT := [ [ [1, 2, 3, 4], [5, 6, 7, 8], [9
, 10, 11, 12] ], [ [13, 14, 15, 16], [ 17, 18, 19, 20], [21, 22, 23, 24] ] ];
    a4Boxes : ARRAY[1..2] OF ARRAY[1..3] OF ARRAY[1..4] OF ARRAY[1..5] OF INT;
END_VAR
```

```
aPoints[1, 2] := 1200;
a2Boxes[1][2] := 1200;
```

aPoints	ARRAY [1..2, 1..3] OF INT	
aPoints[1, 1]	INT	1
aPoints[1, 2]	INT	1200
aPoints[1, 3]	INT	3
aPoints[2, 1]	INT	4
aPoints[2, 2]	INT	5
aPoints[2, 3]	INT	6
a2Boxes	ARRAY [1..2] OF ARRAY [1..3] OF INT	
a2Boxes[1]	ARRAY [1..3] OF INT	
a2Boxes[1][1]	INT	1
a2Boxes[1][2]	INT	1200
a2Boxes[1][3]	INT	3
a2Boxes[2]	ARRAY [1..3] OF INT	
a2Boxes[2][1]	INT	4
a2Boxes[2][2]	INT	5
a2Boxes[2][3]	INT	6

16.5.16.3 可变长度的数组

在功能块、函数或方法中，在声明部分 VAR_IN_OUT 中可以对可变长度的数组进行声明。运算符 LOWER_BOUND 和 UPPER_BOUND 可用于确定在运行时实际使用的数组的索引限值。LOWER_BOUND 会返回下限值，UPPER_BOUND 会返回上限值。

只有静态声明的数组才能被传递给可变长度的数组。不得传递使用 __NEW 运算符创建的动态数组。

声明可变长度的一维数组的语法

```
<variable name> : ARRAY[*] OF <data type> ( := <initialization> )? ;
```

```
<data type> : elementary data types | user defined data types | function block types
// (...)? : Optional
```

声明可变长度的多维数组的语法

```
<variable name> : ARRAY[* ( , * )+ ] OF <data type> ( := <initialization> )? ;
```

```
<data type> : elementary data types | user defined data types | function block types
// (...)+ : One or more further dimensions
// (...)? : Optional
```

计算限值索引的运算符的语法

```
LOWER_BOUND( <variable name> , <dimension number> )
UPPER_BOUND( <variable name> , <dimension number> )
```

示例

函数 F_SUM 将数组元素的整数值相加，并将计算出的和作为结果返回。总和根据在运行时存在的所有数组元素计算得出。由于只有在运行时才能知道数组元素的实际数目，因此将局部变量声明为可变长度的一维数组。您可以将不同固定长度的数组传递给该加法函数。

```
FUNCTION F_Sum : DINT;
VAR_IN_OUT
    aData     : ARRAY [*] OF INT;
END_VAR
VAR
    i, nSum   : DINT;
END_VAR
```

```
nSum := 0;

FOR i := LOWER_BOUND(aData,1) TO UPPER_BOUND(aData,1) DO
    nSum := nSum + aData[i];
END_FOR;

F_Sum := nSum;
```

16.5.17 结构

结构是 1 种用户定义的数据类型，它将多个任意数据类型的变量组合成 1 个逻辑单元。在结构中声明的变量被称为组件。

使用 PLC 项目树的上下文菜单中的命令 **Add** > **DUT**（添加 > DUT），您可以在项目中创建的 DUT 对象中声明结构。

语法：

```
TYPE <structure name> :
STRUCT
    (<variable declaration optional with initialization>)+
END_STRUCT
END_TYPE
```

<structure name> 是在整个项目中有效的标识符，您可以像使用标准数据类型一样使用它。此外，您还可以声明任意多的变量（至少 1 个），而且，您可以选择通过初始化对它们进行补充。

此外，您还可以嵌套结构。这意味着您要用现有的结构类型来声明结构组件。唯一的限制是，您不得为变量（结构组件）分配任何地址（此处不允许使用 AT 声明）。

示例：结构声明

```
TYPE ST_POLYGONLINE :
STRUCT
    aStart : ARRAY[1..2] OF INT := [-99, -99];
    aPoint1 : ARRAY[1..2] OF INT;
    aPoint2 : ARRAY[1..2] OF INT;
    aPoint3 : ARRAY[1..2] OF INT;
    aPoint4 : ARRAY[1..2] OF INT;
    aEnd : ARRAY[1..2] OF INT := [99, 99];
END_STRUCT
END_TYPE
```

8 字节对齐

TwinCAT 3 引入了 8 字节对齐方式。如果数据作为整个内存块与其他控制器或软件组件进行交换，请确保对齐方式正确（请参见 对齐方式 [▶ 731]）。

扩展类型声明

从现有的结构开始，声明另一个结构。除了自身的组件外，扩展结构还具有与基本结构相同的结构组件。

语法

```
TYPE <structure name> EXTENDS <basis structure> :
STRUCT
    (<variable declaration optional with initialization>)+
END_STRUCT
END_TYPE
```

示例：结构声明

```
TYPE ST_PENTAGON EXTENDS ST_POLYGONLINE :
STRUCT
    aPoint5 : ARRAY[1..2] OF INT;
END_STRUCT
END_TYPE
```

结构的声明和初始化

示例

```
PROGRAM Line
VAR
    stPolygon  : ST_POLYGONLINE := (aStart:=[1,1], aPoint1:=[5,2], aPoint2:=[7,3], aPoint3:=[8,5],
aiPoint4:=[5,7], aEnd:=[1,1]);
    stPentagon : ST_PENTAGON := (aStart:=[0,0], aPoint1:=[1,1], aPoint2:=[2,2], aPoint3:=[3,3],
aPoint4:=[4,4], aPoint5:=[5,5], aEnd:=[0,0]);
END_VAR
```

您不能对变量使用初始化。有关如何初始化结构数组的示例，请参见数组数据类型的帮助页面。

另请参见：

- 对象 DUT [▶ 68]

访问结构组件

根据以下语法，您可以访问结构组件：

```
<variable name>.<component name>
```

示例

```
PROGRAM Polygon
VAR
    stPolygon : ST_POLYGONLINE := := (aStart:=[1,1], aPoint1:=[5,2], aPoint2:=[7,3], aPoint3:=[8,5],
aiPoint4:=[5,7], aEnd:=[1,1]);
    nPoint    : INT;
END_VAR
// Assigns 5 to nPoint
```

```
nPoint := stPolygon.aPoint1[1];
```

Ergebnis: nPoint = 5

结构变量中的符号位访问

您可以用数据类型为 BIT 的变量声明 1 个结构，以便将各个位组合成 1 个逻辑单元。这样，您可以通过名称（而不是位索引）对各个位进行符号寻址。

语法声明：

```
TYPE <structure name> :
STRUCT
    ( <bit name> : BIT; )+
END_STRUCT
END_TYPE
```

位访问语法：

<Structure name>.<Bit name>

示例：

结构声明

```
TYPE ST_CONTROL :
STRUCT
    bitOperationEnabled  : BIT;
    bitSwitchOnActive    : BIT;
    bitEnableOperation   : BIT;
    bioterror            : BIT;
    bitVoltageEnabled    : BIT;
    bitQuickStop         : BIT;
    bitSwitchOnLocked    : BIT;
    bitWarning           : BIT;
END_STRUCT
END_TYPE
```

位访问

```
FUNCTION_BLOCK FB_Controller
VAR_INPUT
    bStart : BOOL;
END_VAR
VAR_OUTPUT
END_VAR
VAR
    stControl : ST_CONTROL;
END_VAR

IF bStart = TRUE THEN
// Symbolic bit access
stControl.bitEnableOperation := TRUE;
END_IF
```

```
PROGRAM MAIN
VAR
fbController : FB_Controller;
END_VAR
```

```
fbController();
fbController.bStart := TRUE;
```

BIT 变量的引用和指针是无效声明，基本类型为 BIT 的数组组件也是如此。

另请参见：

- BIT [▶ 700]
- 数组 [▶ 714]

还请参阅有关此

对象 DUT [▶ 68]

16.5.18 枚举

枚举是 1 种用户定义的数据类型，由以逗号分隔的一系列组件组成，这些组件也被称为枚举值，用于声明用户定义的变量。此外，您还可以像使用常量变量一样使用枚举组件，其标识符 <enumeration name>.<component name> 在项目中是全局已知的。

使用 PLC 项目树的上下文菜单中的命令 **Add** > **DUT**（添加 > DUT），您可以在项目中创建的 DUT 对象中声明枚举。

您添加到项目中的每个枚举都会在 TYPE 声明上方的行中被自动赋予 1 个属性“strict” [▶ 762]和 1 个属性“qualified only” [▶ 761]。您也可以显式添加或删除属性。

另请参见：

- 对象 DUT [▶ 68]
- 属性“to_string” [▶ 774]

- 枚举变量的 TO_STRING/TO_WSTRING

声明

语法

```
{attribute 'strict'}
TYPE <enumeration name>:
(
    <component declaration>,
    <component declaration>
) <basic data type> := <default variable initialization>
;
END_TYPE
```

{attribute 'strict'}	可选 该编译指示会导致执行严格的类型检查，如下所述。 该编译指示是可选的，但建议使用。
<enumeration name>	枚举的名称，可在代码中用作数据类型 示例：E_ColorBasic
<component declaration>	2 个或以上组件（任何等于或大于 2 的数字） 组件的值自动进行初始化：从 0 开始，值连续递增 1。不过，您也可以为各个组件显式分配固定的初始值。 示例：eYellow := 1
<basic data type>	可选 您可以显式声明以下基本数据类型之一： UINT | SINT | USINT | DINT | UDINT | LINT | ULINT | BYTE | WORD | DWORD | LWORD 如果没有声明显式基本数据类型，则会自动使用 INT 基础数据类型。
<default variable initialization>	可选 您可以将其中 1 个组件显式声明为初始组件。如果没有指定显式初始化，则会自动使用顶部组件执行初始化。

示例

```
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE E_ColorBasic :
(
    eRed,
    eYellow,
    eGreen := 10,
    eBlue,
    eBlack
) // Basic data type is INT, default initialization for all E_ColorBasic  variables is eYellow
;
END_TYPE
```

在该声明中，前 2 个组件会收到标准初始化值：eRed = 0、eYellow = 1，第 3 个组件的初始化值以不同的方式进行显示定义：eGreen = 10。然后，以下组件会收到标准初始化值：eBlue = 11、eBlack = 12。

具有显式基本数据类型的枚举

IEC 61131-3 标准的扩展

枚举声明的基本数据类型默认为 INT。不过，您也可以声明明确基于不同整数数据类型的枚举。

示例：使用 DWORD 基本数据类型的枚举

```
TYPE E_Color :
(
    eWhite  := 16#FFFFFF,
    eYellow := 16#FFFF00,
    eGreen  := 16#FF00FF,
    eBlue   := 16#0000FF,
    eBlack  := 16#000000
```

```
)DWORD := eBlack
; // Basic data type is DWORD, default initialization for all E_Color variables is eBlack
END_TYPE
```

严格的编程规则

对于 TwinCAT 3.1 Build 4026 及以上版本，为枚举的第 1 行自动添加编译指示 {attribute 'strict'}。

在添加编译指示 {attribute 'strict'} 之后，将会激活严格的编程规则。

下面的代码会被归类为编译器错误：

- 例如，使用枚举组件进行算术运算时，枚举变量不能用作 FOR 循环中的计数器变量。
- 为常量值的枚举部分赋值，而该常量值与枚举值不对应。
- 为非常量变量的枚举组件赋值，而该非常量变量的数据类型与枚举不同。

算术运算可能会导致将未声明的值分配给枚举组件。更好的编程风格是，使用 SWITCH/CASE 指令来处理组件值。

枚举变量的声明和初始化

语法

```
<variable name> : <enumeration name> ( := <initialization> )? ;
```

在声明具有用户定义的数据类型 <enumeration name> 的枚举变量时，您可以使用枚举组件对其进行初始化。

示例

```
PROGRAM MAIN
VAR
    eColorCar  : E_Color;
    eColorTaxi : E_Color := E_Color.eYellow;
END_VAR
```

使用 E_Color.eblack 对变量 eColorCar 进行初始化。这是 E_Color 类型的所有枚举变量的标准初始化，因此在类型声明中以这种方式进行定义。变量 eColorTaxi 有自己的初始化。

如果未指定初始化，则使用 0 进行初始化。

```
PROGRAM MAIN
VAR
    eColorFlower : E_ColorBasic;
    eColorTree   : E_ColorBasic := E_ColorBasic.eGreen;
END_VAR
```

使用 E_ColorBasic.eYellow 对变量 eColorFlower 进行初始化。这是 E_ColorBasic 类型的所有枚举变量的标准初始化。由于在枚举声明中没有指定用于初始化的组件，因此会自动使用值为 0 的组件进行初始化。这通常是枚举组件中的第 1 个。不过，这也可以是另一个不在第 1 位但使用 0 进行显式初始化的组件。变量 eColorTree 有 1 个显式初始化。

如果没有为类型和变量指定值，则适用以下规则：如果枚举包含 0 值，则该值为标准初始化；如果不包含，则为列表中的第 1 个组件。

示例

使用 0 组件进行初始化：

```
TYPE E_SampleA :
(
    e1 := 2,
    e2 := 0,
    e3
);
END_TYPE
```

```
PROGRAM MAIN
VAR
    eSampleA  : E_SampleA;
END_VAR
```

使用 E_SampleA.e2 对变量 eSampleA 进行初始化。

使用第 1 个组件进行初始化：

```
TYPE E_SampleB :
(
    e1 := 3,
    e2 := 1,
    e3
);
END_TYPE
```

```
PROGRAM MAIN
VAR
    eSampleB  : E_SampleB;
END_VAR
```

使用 E_SampleB.e1 对变量 eSampleB 进行初始化。

IEC 61131-3 标准的扩展

枚举组件也可以用作常量变量，其标识符为 <enumeration name>.<component name>。枚举组件在项目中是全局已知的，可对它们进行显式访问。因此，1 个组件名称可以在不同的枚举中使用。

示例：蓝色组件

```
PROGRAM MAIN
VAR
    eFlower   : E_ColorBasic;
    eColorCar : E_Color;
END_VAR
```

```
// unambiguous identifiers although the component names are identical
eFlower   := E_ColorBasic.eBlue;
eColorCar := E_Color.eBlue;

// invalid code
eFlower   := eBlue;
eColorCar := eBlue;
```

16.5.19 别名

别名是 1 种用户定义的数据类型，可用于为数据类型或功能块创建替代名称。

在项目中，您可以在使用 PLC 项目树的上下文菜单中的命令 **Add > DUT**（添加 > DUT）创建的 DUT 对象中声明别名。

语法：

```
TYPE <identifier> : <Assignment statement>;
END_TYPE
```

示例：

该示例显示了别名 T_Message 的声明。在程序中声明的 T_Message 类型的 PLC 变量始终是 1 个包含 50 个字符的字符串。

```
TYPE T_Message : STRING[50];
END_TYPE
```

声明：

```
sMessageA : T_Message;
```

程序：

```
sMessageA := 'This is a message';
```

另请参见：

- 对象 DUT [▶ 68]

16.5.20 UNION

UNION 是 1 种数据结构，通常包含不同的数据类型。在 1 个联合中，所有组件都有相同的偏移量，这意味着它们会占据相同的内存空间。

在项目中，您可以在使用 PLC 项目树的上下文菜单中的命令 **Add > DUT**（添加 > DUT）创建的 DUT 对象中声明联合。

语法：

```
TYPE <union name>:
UNION
        <Variable declaration 1>
        ……
        <Variable declaration n>
END_UNION
END_TYPE
```

示例 1：

在以下声明联合的示例中，对 uName.fA 的赋值也会影响 uName.nB 和 uName.nC。

声明：

```
TYPE U_Name:
UNION
    fA : LREAL;
    nB : LINT;
    nC : WORD;
END_UNION
END_TYPE
```

实例化：

```
VAR
    uName : U_Name;
END_VAR
```

赋值：

```
uName.fA := 1;
```

结果：

```
fA = 1
nB = 16#3FF0000000000000
nC = 0
```

示例 2：

在以下声明联合的示例中，对 u2Byte.nUINT 的赋值也会影响 u2Byte.a2Byte 和 u2Byte.aBits。

结构 ST_Bits 的声明：

```
TYPE ST_Bits :
STRUCT
    bBit1  : BIT;
    bBit2  : BIT;
    bBit3  : BIT;
    bBit4  : BIT;
    bBit5  : BIT;
    bBit6  : BIT;
    bBit7  : BIT;
    bBit8  : BIT;
END_STRUCT
END_TYPE
```

联合 U_2Byte 的声明：

```
TYPE U_2Byte :
UNION
    nUINT  : UINT;
    a2Byte : ARRAY[1..2] OF BYTE;
```

```
    aBits  : ARRAY[1..2] OF ST_Bits;
END_UNION
END_TYPE
```

联合的实例化：

```
VAR
    u2Byte : U_2Byte;
END_VAR
```

赋值 1：

```
u2Byte.nUINT := 5;
```

结果 1：

Expression	Type	Value
u2Byte	U_2Byte	
nUINT	UINT	5
a2Byte	ARRAY [1..2] OF BYTE	
a2Byte[1]	BYTE	5
a2Byte[2]	BYTE	0
aBits	ARRAY [1..2] OF ST_Bits	
aBits[1]	ST_Bits	
bBit1	BIT	TRUE
bBit2	BIT	FALSE
bBit3	BIT	TRUE
bBit4	BIT	FALSE
bBit5	BIT	FALSE
bBit6	BIT	FALSE
bBit7	BIT	FALSE
bBit8	BIT	FALSE
aBits[2]	ST_Bits	
bBit1	BIT	FALSE
bBit2	BIT	FALSE
bBit3	BIT	FALSE
bBit4	BIT	FALSE
bBit5	BIT	FALSE
bBit6	BIT	FALSE
bBit7	BIT	FALSE
bBit8	BIT	FALSE

赋值 2：

```
u2Byte.nUINT := 255;
```

结果 2：

Expression	Type	Value
u2Byte	U_2Byte	
nUINT	UINT	255
a2Byte	ARRAY [1..2] OF BYTE	
a2Byte[1]	BYTE	255
a2Byte[2]	BYTE	0
aBits	ARRAY [1..2] OF ST_Bits	
aBits[1]	ST_Bits	
bBit1	BIT	TRUE
bBit2	BIT	TRUE
bBit3	BIT	TRUE
bBit4	BIT	TRUE
bBit5	BIT	TRUE
bBit6	BIT	TRUE
bBit7	BIT	TRUE
bBit8	BIT	TRUE
aBits[2]	ST_Bits	
bBit1	BIT	FALSE
bBit2	BIT	FALSE
bBit3	BIT	FALSE
bBit4	BIT	FALSE
bBit5	BIT	FALSE
bBit6	BIT	FALSE
bBit7	BIT	FALSE
bBit8	BIT	FALSE

赋值 3：

```
u2Byte.nUINT := 256;
```

结果 3：

Expression	Type	Value
u2Byte	U_2Byte	
nUINT	UINT	256
a2Byte	ARRAY [1..2] OF BYTE	
a2Byte[1]	BYTE	0
a2Byte[2]	BYTE	1
aBits	ARRAY [1..2] OF ST_Bits	
aBits[1]	ST_Bits	
bBit1	BIT	FALSE
bBit2	BIT	FALSE
bBit3	BIT	FALSE
bBit4	BIT	FALSE
bBit5	BIT	FALSE
bBit6	BIT	FALSE
bBit7	BIT	FALSE
bBit8	BIT	FALSE
aBits[2]	ST_Bits	
bBit1	BIT	TRUE
bBit2	BIT	FALSE
bBit3	BIT	FALSE
bBit4	BIT	FALSE
bBit5	BIT	FALSE
bBit6	BIT	FALSE
bBit7	BIT	FALSE
bBit8	BIT	FALSE

另请参见：

- 对象 DUT [▶ 68]

16.6 全局数据类型

16.6.1 概述

TwinCAT 3 提供了 1 个用于管理数据类型的类型系统。类型系统由系统基本类型组成，可通过客户项目扩展自定义数据类型。（另请参见 TwinCAT 3 类型系统）

在本部分中，您将会了解 TwinCAT 系统为 PLC 提供的全局数据类型：

- PlcAppSystemInfo [▶ 728]
- PlcTaskSystemInfo [▶ 729]
- ST_LibVersion [▶ 730]

16.6.2 PlcAppSystemInfo

每个 PLC 都包含 1 个类型为“PlcAppSystemInfo”的实例，名称为“_AppInfo”。

相应的命名空间是”TwinCAT_SystemInfoVarList“。例如，在库中使用时，必须指定这一点。

```
TYPE PlcAppSystemInfo
STRUCT
    ObjId              : OTCID;
    TaskCnt            : UDINT;
    OnlineChangeCnt    : UDINT;
    Flags              : DWORD;
    AdsPort            : UINT;
    BootDataLoaded     : BOOL;
```

```
    OldBootData         : BOOL;
    AppTimestamp        : DT;
    KeepOutputsOnBP     : BOOL;
    ShutdownInProgress  : BOOL;
    LicensesPending     : BOOL;
    BSODOccured         : BOOL;
    LoggedIn            : BOOL;
    PersistentStatus    : EPlcPersistentStatus;

    TComSrvPtr          : ITComObjectServer;

    AppName             : STRING(63);
    ProjectName         : STRING(63);
END_STRUCT
END_TYPE
```

ObjId	PLC 项目实例的对象 ID
TaskCnt	运行时系统中的任务数
OnlineChangeCnt	自上次完整下载以来的在线更改次数
标志	保留
AdsPort	PLC 应用程序的 ADS 端口
BootDataLoaded	PERSISTENT 变量：LOADED（无错误）
OldBootData	PERSISTENT 变量：INVALID（由于没有有效的文件，所以加载了备份副本）
AppTimestamp	编译 PLC 应用程序的时间
KeepOutputsOnBP	该标志可以设置，并防止在达到断点时输出被清零。在这种情况下，任务继续运行。只有 PLC 代码的执行被中断。
ShutdownInProgress	如果 TwinCAT 系统正在关闭，该变量的值为 TRUE。TwinCAT 系统的某些部分可能已经关闭。
LicensesPending	如果尚未验证所有许可证加密狗提供的许可证，该变量的值为 TRUE。
BSODOccured	如果 Windows 处于 BSOD 状态，该变量的值为 TRUE。
LoggedIn	如果至少有 1 个 XAE 实例登录到项目中，该变量的值为 TRUE。
PersistentStatus	PERSISTENT 变量：恢复状态： PS_None：未恢复任何持久变量。 PS_All：在当前项目状态下持久存在的所有变量已恢复。 PS_Partial：恢复的变量少于在当前项目状态下持久存在的变量。
TComSrvPtr	指向 TcCOM 对象服务器的指针
AppName	TwinCAT 生成的名称，其中包含端口。
ProjectName	项目名称

与 TwinCAT 2 的区别

如果在 TwinCAT 2 下使用变量 runTimeNo，则必须转换相应的程序代码，以便在 TwinCAT 3 下应用。

示例：

- **TwinCAT 2 下的应用程序：** nPlcAdsPort := 801 + (SystemInfo.runTimeNo - 1) * 10;
- **TwinCAT 3 下的应用程序：** nPlcAdsPort := _AppInfo.AdsPort;

16.6.3 PlcTaskSystemInfo

每个 PLC 都包含 1 个该类型实例的数组。数组的名称是“_TaskInfo[]”。
通过使用相应任务的索引作为数组索引，您可以访问该类型的单个实例。使用 Tc2_System 库所提供的 GETCURTASKINDEXEX 函数可以读取任务索引。

相应的命名空间是“TwinCAT_SystemInfoVarList”。例如，在库中使用时，必须指定这一点。

```
{attribute 'Namespace' := 'PLC'}
TYPE PlcTaskSystemInfo
STRUCT
    ObjId                   : OTCID;
    CycleTime               : UDINT;
    Priority                : UINT;
    AdsPort                 : UINT;
```

```
    CycleCount              : UDINT;
    DcTaskTime              : LINT;
    LastExecTime            : UDINT;
    FirstCycle              : BOOL;
    CycleTimeExceeded       : BOOL;
    InCallAfterOutputUpdate : BOOL;
    RTViolation             : BOOL;

    TaskName                : STRING(63);
END_STRUCT
END_TYPE
```

ObjId	任务引用的对象 ID，从中调用 PLC 程序。
CycleTime	以 100 ns 的倍数设置任务循环时间
优先级	设置任务优先级
AdsPort	任务的 ADS 端口
CycleCount	周期计数器 请注意，周期计数器指的是实际系统任务，而不是 PLC 项目的任务引用。其背景是多个 PLC 项目或其他 TcCOM 模块可以共享 1 个任务，因此，这些模块可以访问同一个周期计数器。因此，在运行模式下重启 TwinCAT 期间（而不是在重置 PLC 项目时），周期计数器会被重置。
DcTaskTime	分布式时钟系统时间。它在任务运行时保持不变。
LastExecTime	最后一个周期所需的循环时间，以 100 ns 的倍数表示
FirstCycle	在第 1 个 PLC 任务周期内，该变量的值为 TRUE。
CycleTimeExceeded	该变量表示是否已超出设定的任务循环时间。 CycleTimeExceeded 变量的值会在每个周期内更新。如果前一个周期已在预定的时间范围内完成，则该变量被设置为 0，否则设为 1。
InCallAfterOutputUpdate	如果使用属性“TcCallAfterOutputUpdate”声明当前调用的起源，该变量的值为 TRUE。
RTViolation	如果在混合核（Windows + 单核上的实时）上已超过实时限制，该变量的值为 TRUE。在这种情况下，TRUE 值指的是正在运行相应任务的核。
TaskName	PLC 项目中的任务引用对象的名称（例如，“MyPlcProject_PlcTaskRef”）

示例：

```
VAR
    nTaskIdx     : DINT;
    nCycleTime   : UDINT;
END_VAR
```

```
nTaskIdx := GETCURTASKINDEXEX();
IF nTaskIdx > 0 THEN
    nCycleTime := _TaskInfo[nTaskIdx].CycleTime;
END_IF
```

16.6.4 ST_LibVersion

该结构定义了 PLC 库的版本。每个库都包含该数据类型的全局声明实例。

```
TYPE ST_LibVersion :
STRUCT
    iMajor     : UINT;
    iMinor     : UINT;
    iBuild     : UINT;
    iRevision  : UINT;
    nFlags     : DWORD
    sVersion   : STRING(23);
END_STRUCT
END_TYPE
```

iMajor	主要编号
iMinor	次要编号
iBuild	版本号
iRevision	修订版本号
nFlags	启用库 nFlags = 1，已启用库（复选标记已设置）， nFlags = 0，未启用库。
sVersion	完整版本作为字符串

16.7 对齐方式

在 TwinCAT 3 中引入 8 字节对齐方式。

系统	对齐方式
TwinCAT 2，x86	1 字节
TwinCAT 2，ARM	4 字节
TwinCAT 3	8 字节

如果数据作为整个内存块与其他控制器或软件组件（例如，可视化）进行交换，则需要特别注意。

内存位置

- 变量所在的内存位置由其数据类型大小和对齐方式指定。
  - 如果数据类型小于或等于系统的对齐方式，则变量的内存位置相当于数据类型大小的倍数。
  - 如果数据类型大于系统的对齐方式，则变量的内存位置相当于对齐方式的倍数。
- 结构化数据类型（结构、功能块）所在的内存位置由结构中最大数据类型的大小和对齐方式指定。
  - 如果最大数据类型小于或等于系统的对齐方式，则结构实例的内存位置相当于该数据类型大小的倍数。
  - 如果最大数据类型大于系统的对齐方式，则结构实例的内存位置相当于对齐方式的倍数。

内存空间

- 从上述内存位置来看，1 个变量所占的内存大小由其数据类型的大小决定。
- 结构化数据类型（结构、功能块）所占的内存空间，从上述内存位置开始，由结构中最大数据类型的大小和对齐方式指定。
  - 如果最大数据类型小于或等于系统的对齐方式，则结构实例所占的内存空间相当于该数据类型大小的倍数。
  - 如果最大数据类型大于系统的对齐方式，则结构实例所占的内存空间相当于对齐方式的倍数。

这些对齐规则可能会导致隐式填充字节。

请注意（例如，对于借助 Tc2 系统函数 MEMCMP 进行内存比较的情况），填充字节不会进行初始化。

功能块中最大的数据类型

每个功能块都隐式包含至少 1 个指针，用于存储虚拟方法表的地址。这意味着，在 64 位架构的目标系统中，功能块包含大小为 8 字节的数据类型。根据功能块的不同，这可能是其包含的最大数据类型，可决定内存位置和内存空间。

示例 1（TwinCAT 3）

```
TYPE ST_Test1 :
STRUCT
    fVar   : LREAL;  // 8 Byte
    nVar1  : DINT;   // 4 Byte
    nVar2  : SINT;   // 1 Byte
    (* 3 filler bytes to reach a limit corresponding to the largest data type (divisible by 8 byte)
```

```
*)
END_STRUCT
END_TYPE
```

```
TYPE ST_Test2 :
STRUCT
    nVar2  : SINT;   // 1 Byte
    (* 3 filler bytes to reach a limit corresponding to the following data type (divisible by 4 byte
) *)
    nVar1  : DINT;   // 4 Byte
    fVar   : LREAL;  // 8 Byte
END_STRUCT
END_TYPE
```

```
TYPE ST_Test3 :
STRUCT
    nVar2  : SINT;   // 1 Byte
    (* 7 filler bytes to reach a limit corresponding to the following data type (divisible by 8 byte
) *)
    fVar   : LREAL;  // 8 Byte
    nVar1  : DINT;   // 4 Byte
    (* 4 filler bytes to reach a limit corresponding to the largest data type (divisible by 8 byte)
*)
END_STRUCT
END_TYPE
```

由于 TwinCAT 3 采用 8 字节对齐方式，因此会添加隐式填充字节。虽然这些结构包含 3 个相同数据类型的变量，但它们的整体大小可能会有所不同。

Expression	Type	Value
stTest1	ST_Test1	
stTest2	ST_Test2	
stTest3	ST_Test3	
nSize1	UDINT	16
nSize2	UDINT	16
nSize3	UDINT	24

```
1  nSize1 16 :=SIZEOF(stTest1);
2  nSize2 16 :=SIZEOF(stTest2);
3  nSize3 24 :=SIZEOF(stTest3);
4  RETURN
```

在采用 1 字节对齐方式的系统中，这 3 个结构均有相同的 13 字节内存要求。

示例 2

```
TYPE ST_Test :
STRUCT
    nDWORD : DWORD;   // 4 Byte
    (* With an 8-byte alignment: 4 filler bytes *)
    nLWORD : LWORD;   // 8 Byte
END_STRUCT
END_TYPE
```

采用 4 字节对齐方式，该结构的大小为 12 字节。没有插入任何填充字节。原因：

- 变量 nLWORD 所在的内存位置相当于对齐方式的倍数，因为其数据类型（8 字节）大于对齐方式（4 字节）。
- 结构化数据类型所占的内存空间相当于对齐方式的倍数，因为结构中的最大数据类型（8 字节）大于对齐方式（4 字节）。

反之，如果采用 8 字节对齐方式，该结构的大小为 16 字节，因为在 2 个变量之间插入了 4 个填充字节。原因：

- 变量 nLWORD 所在的内存位置相当于其数据类型大小的倍数，因为其数据类型（8 字节）小于或等于对齐方式（8 字节）。
- 结构化数据类型所占的内存空间相当于最大数据类型大小的倍数，因为结构中的最大数据类型（8 字节）小于或等于对齐方式（8 字节）。

更多信息和示例项目：

- 如需定义数据结构的打包方式，另请参见 属性“pack mode” [▶ 755]。
- 在 PLC 示例页面 [▶ 1051]上，您还可以下载有关字节对齐主题的示例项目。

16.8 编译指示

编译指示指令会影响与编译或预编译过程有关的 1 个或多个变量的属性。为此可以使用各种类别的编译指示。

16.8.1 消息编译指示

消息编译指示可用于在编译过程中强制在消息窗口中输出消息。

您可以在 POU 的文本编辑器中的单独 1 行或现有 1 行中插入编译指示指令。

类型

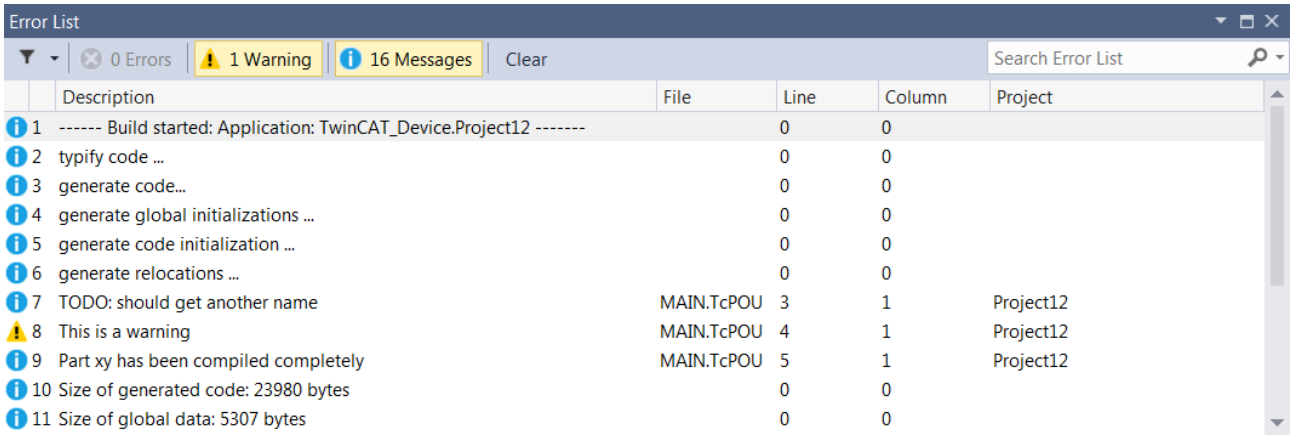
编译指示	消息类型
{text ‘textstring’}	文本：<text string> 的输出。
{info ‘textstring’}	信息：<text string> 的输出。
{warning ‘textstring’}	警告：<text string> 的输出。 与属性编译指示“obsolete”相反，您可以在本地为当前位置定义警告。
{error ‘textstring’}	错误：<text string> 的输出。

在 TwinCAT 消息窗口中，您可以使用命令 **Next Message**（下一条消息）或 **Previous Message**（上一条消息）访问 **Warning and Error**（警告和错误）中的 **Information**（信息）类别消息的源位置。这意味着您会到达在源代码中添加编译指示的位置。

示例：

```
VAR
    nVar : INT; {info 'TODO: should get another name'}
    bVar : BOOL;
    aTest : ARRAY [0..10] OF INT;
    nIdx : INT;
END_VAR

aTest[nIdx] := aTest[nIdx]+1;
nVar := nVar+1;

{warning 'This is a warning'}
{text 'Part xy has been compiled completely'}
```

消息窗口中的输出：



另请参见：

- 条件编译指示 [▶ 775]

16.8.2 属性编译指示

属性语法用于影响编译和预编译。

TwinCAT 支持许多预定义的属性编译指示。此外，您还可以使用自定义编译指示，在编译项目之前使用条件编译指示对其进行查询。

如果您定义自己的属性，请确保这些属性明确无误。例如，您可以为属性名称分配 1 个前缀来实现这一点。

属性在声明部分定义。例外：只有当动作或转换的实现语言是“结构化文本（ST）”时，您才能在 **Action**（动作）或 **Transition**（转换）对象中使用属性。由于动作和转换没有声明部分，因此您要在实现部分开始时定义属性。

16.8.2.1 用户定义的属性

自定义属性是任何项目或用户定义的属性，您可以将其应用于 POU、动作、数据类型定义和变量。在编译 PLC 项目之前，您可以使用条件编译指示查询自定义属性。

您可以使用 hasattribute 运算符通过条件编译指示查询自定义属性。

语法：{attribute 'attribute'}

POU 和动作的示例：

函数 F_Sample 的属性“vision”：

```
{attribute 'vision'}
FUNCTION F_Sample : INT
VAR_INPUT
    nSample : INT;
END_VAR
```

变量的示例：

变量 nVar 的属性“DoCount”：

```
PROGRAM MAIN
VAR
    {attribute 'DoCount'};
    nVar : INT;
    bVar : BOOL;
END_VAR
```

数据类型的示例：

数据类型 ST_MyType 的属性“aType”：

```
{attribute 'aType'}
TYPE ST_MyType :
STRUCT
    nTest : INT;
    bTest : BOOL;
END_STRUCT
END_TYPE
```

另请参见：

- 条件编译指示 [▶ 775]

16.8.2.2 属性“c++_compatible”

该编译指示使 PLC 编译器生成的 VTable 与 C++ 编译器生成的 VTable 二进制兼容。这样可以从在 C++ 中实现的 TcCom 模块访问在 PLC 中实现的接口方法。

语法： {attribute 'c++_compatible'}

插入位置：

在以下位置必须添加编译指示：

- 在接口声明上方一行
- 在各个接口方法的声明上方一行
- 在实现接口的功能块的声明上方一行
- 在从接口实现的方法的声明上方一行

示例：

实现 C++ 兼容接口的功能块的声明：

```
{attribute 'c++_compatible'}
FUNCTION_BLOCK FB_Sample IMPLEMENTS I_Sample
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
END_VAR
```

在接口中定义的方法的声明：

```
{attribute 'c++_compatible'}
{attribute 'minimal_input_size' := '4'}
{attribute 'pack_mode' := '4'}
METHOD Method1 : HRESULT
VAR_INPUT
    nParameter1 : INT;
END_VAR
```

另请参见：

- 属性“minimal_input_size” [▶ 749]
- 属性“pack_mode” [▶ 755]

16.8.2.3 属性“call_after_global_init_slot”

该编译指示会导致包含该属性的所有函数和程序在全局初始化后被调用。使用属性值确定调用顺序。

语法： {attribute 'call_after_global_init_slot' := '<slot>'}

<slot>：整数值，用于定义调用顺序中的排序值：值越小，调用越早。如果多个功能块具有相同的属性值，则仍然没有定义它们的调用顺序。

插入位置： 在函数和程序的声明部分上方第 1 行

如果某个方法具有该属性，则 TwinCAT 会确定相应功能块的所有实例，并调用指定插槽中的所有实例。在这种情况下，您不会影响实例的顺序。

另请参见：

- 方法 FB_init、FB_reinit 和 FB_exit [▶ 786]

16.8.2.4 属性“call_after_init”

该编译指示会导致在功能块实例的初始化之后隐式调用方法。严格来说，在处理初始分配之后、PLC 项目的任务启动之前会调用该方法。因此，它可以适当地响应用户的规范要求。您可以自由选择方法的名称（例外：FB_init、FB_reinit 和 FB_exit）。

从性能方面考虑，您必须在声明部分上方专用的第 1 行中为功能块和方法添加属性。

语法：{attribute 'call_after_init'}

插入位置：在方法和功能块的声明部分上方第 1 行

TwinCAT 会在方法 FB_init 之后、初始化表达式的变量值在实例声明中生效之后调用该方法。

派生功能块的 call_after_init

对于扩展另一个使用属性“call_after_init”的功能块的功能块，也必须为其分配该属性。

为了清楚起见，建议您使用相同的名称、签名和属性覆盖相应的方法。这需要调用 SUPER^.MyInit。

包含属性“call_after_init”的方法不得有任何输入（VAR_INPUT）。TwinCAT 会发出相应的编译错误。

调试

例如，在使用 {attribute' call_after_init'} 声明的方法中查找错误非常麻烦，因为设置断点无法达到预期效果。

示例：

定义：

```
{attribute 'call_after_init'}
FUNCTION_BLOCK FB_Sample
... <functionblock definition>
```

```
{attribute 'call_after_init'}
METHOD CallAfterInit
... <method definition>
```

例如，该定义在以下代码处理中实现了以下声明：

```
fbSample : FB_Sample := (nValue1 := 99);
```

代码处理：

```
fbSample.FB_Init();
fbSample.nValue1 := 99;
fbSample.CallAfterInit();
```

通过这种方式，CallAfterInit() 能够响应用户定义的初始化。

另请参见：

- 方法 FB_init、FB_reinit 和 FB_exit [▶ 786]

16.8.2.5 属性“call_after_online_change_slot”

该编译指示会导致包含该属性的所有函数和程序在在线更改后被调用。使用属性值 <slot> 确定调用顺序。

语法：{attribute 'call_after_online_change_slot' := '<slot>'}

<slot>：整数值，用于定义调用顺序中的排序值：值越小，调用越早。如果多个功能块具有相同的属性值，则仍然没有定义它们的调用顺序。

插入位置：在函数和程序的声明部分上方第 1 行

如果某个方法具有该属性，则 TwinCAT 会确定相关功能块的所有实例。TwinCAT 会调用指定插槽中的所有实例。在这种情况下，您不会影响实例的顺序。

由于在在线更改期间无法运行 PLC 程序，因此在这种情况下执行的任何代码都可能会导致抖动。因此，要尽可能减少要执行的代码量。

另请参见：

- TC3 用户界面文档：命令在线更改 [▶ 887]

16.8.2.6 属性“call_on_type_change”

该编译指示用于识别功能块 A 的方法，一旦 A 引用的功能块 B、C……的数据类型发生变化，该方法就会被调用。通过指针变量或 REFERENCE 类型的变量可以定义引用。您定义 A 所引用的功能块，其类型更改将会触发属性值中的方法调用。

语法：

{attribute 'call_on_type_change':= '<name of the first referenced function block>, <name of the second referenced function block>, <name of the ... referenced function block>'}

插入位置：在方法声明中的第 1 行上方一行

示例：

带有引用的功能块：

```
FUNCTION_BLOCK FB_A
...
VAR
    pVar   : POINTER TO FB_B;
    refVar : REFERENCE TO FB_C;
END_VAR
```

响应引用 FB_B 和 FB_C 中的类型更改的方法：

```
{attribute 'call_on_type_change' := 'FB_B, FB_C'}
METHOD METH_react_on_type_change : INT
VAR_INPUT
```

16.8.2.7 属性“conditionalshow”

该编译指示与库中包含的功能块和其他类型相关。

该编译指示的作用是，如果相关库作为 *.compiled 库安装，则在用户界面上不会显示包含该属性的完整功能块。但是，如果相关库作为 *.library 安装，则在用户界面上会显示功能块。

受影响的功能：

- 库管理
- 调试
- 输入助手
- 列出组件功能
- 监控
- 符号配置

这在您开发库时非常有用。作为库开发人员，您可以提供带有编译指示的功能块或变量。这样，您可以指定哪些标识符在包含在项目中后会被隐藏。如果您稍后错过隐藏的标识符，例如，在调试或进一步开发库时，您可以重新启用它们的可见性。

语法：{attribute 'conditionalshow'}

插入位置：在签名上方一行

示例

隐藏变量：

```
FUNCTION_BLOCK FB_Sample
PROGRAM_MAIN
VAR
{attribute 'conditionalshow'}
    nLocal    : INT;
    nCounter  : INT;
END_VAR
```

变量 nLocal 不可见。

隐藏功能块：

```
{attribute 'conditionalshow'}
FUNCTION_BLOCK FB_ Sample
VAR
    nLocal    : INT;
    nCounter  : INT;
END_VAR
```

功能块 FB_Sample 及其变量 nLocal 和 nCounter 均不可见。

另请参见：

- 属性“conditionalshow all locals” [▶ 738]
- 属性“hide” [▶ 744]
- 属性“hide all locals” [▶ 745]

16.8.2.8 属性“conditionalshow_all_locals”

该编译指示与库中包含的功能块和其他类型相关。

该编译指示的作用是，如果相关库作为 *.compiled 库安装，则在用户界面上不会显示任何局部变量。但是，如果相关库作为 *.library 安装，则在用户界面上会显示局部变量。

受影响的功能：

- 库管理
- 调试
- 输入助手
- **列出组件**功能
- 监控
- 符号配置

这在您开发库时非常有用。作为库开发人员，您可以提供带有编译指示的功能块或变量。这样，您可以指定哪些标识符在包含在项目中后会被隐藏。如果您稍后错过隐藏的标识符，例如，在调试或进一步开发库时，您可以重新启用它们的可见性。

语法：{attribute 'conditionalshow_all_locals'}

插入位置：在签名上方一行

示例：隐藏所有变量

```
{attribute 'conditionalshow_all_locals'}
FUNCTION_BLOCK FB_ Sample
VAR
    nLocal    : INT;
    nCounter  : INT;
END_VAR
```

功能块 FB_Sample 的局部变量 nLocal 和 nCounter 不可见。

另请参见：

- 属性“conditionalshow” [▶ 737]
- 属性“hide” [▶ 744]
- 属性“hide all locals” [▶ 745]

16.8.2.9 属性“const_replaced”、属性“const_non_replaced”

无论编译器选项 **Replace constants**（替换常量）的设置如何，{attribute 'const_non_replaced'} 属性都会导致代码中的常量被替换。该属性对标量类型的变量有影响，但对诸如数组和结构等复合类型的变量没有影响。

插入编译指示 {attribute 'const_non_replaced' }，以显式禁用 Replace Constants（替换常量）编译器选项。最好这样做，例如，以便在符号配置中提供常量。

语法：

{attribute 'const_replaced'}

{attribute 'const_non_replaced'}

插入位置：在全局变量的声明行上方一行。

示例：

在符号配置中可以使用常量 nTestCon 和 bTestCon，因为 Replace constants（替换常量）选项已禁用。

```
VAR_GLOBAL CONSTANT
    {attribute 'const_non_replaced'}
    nTestCon  :  INT  := 12;
    {attribute 'const_non_replaced'}
    bTestCon  :  BOOL := TRUE;
    fTestCon  :  REAL := 1.5;
END_VAR

VAR_GLOBAL
    nTestVar  :  INT  := 12;
    bTestVar  :  BOOL := TRUE;
END_VAR
```

另请参见：

- 文档 TC3 用户界面：属性命令（项目） > 类别编译 [▶ 844]

16.8.2.10 属性“dataflow”

在 FBD/LD/IL 编辑器中处理功能块期间，该编译指示可用于控制数据流。该属性可决定在功能块的哪个输入端或输出端连接下一个或上一个功能块。

在功能块的声明中，您仅可为 1 个输入端和 1 个输出端分配属性。

语法：{attribute 'dataflow'}

插入位置：在变量的声明行上方一行。

对于不带“dataflow”属性的功能块，TwinCAT 按以下方式确定数据流：首先，在相同数据类型的输出端和输入端之间建立连接。总是优先使用功能块的第 1 个输入或输出变量。如果没有相同数据类型的变量，则 TwinCAT 会连接最上面的输出端与相邻功能块的最上面的输入端。

示例：

FB 与前一个功能块之间的连接通过输入变量 i1 实现。FB 与后一个功能块之间的连接通过输出变量 outRes1 实现。

```
FUNCTION_BLOCK FB
VAR_INPUT
    r1 : REAL;
    {attribute 'dataflow'}
    i1 : INT;
    i2 : INT;
    r2 : REAL;
END_VAR
VAR_OUTPUT
    {attribute 'dataflow'}
    outRes1 : REAL;
    out1 : INT;
    g1 : INT;
    g2 : REAL;
END_VAR
```

另请参见：

- FBD/LD/IL [▶ 99]

16.8.2.11 属性“displaymode”

该编译指示定义了各个变量的显示模式。该设置可覆盖用于显示监控变量的全局设置，该设置由菜单 **Debug > Representation**（调试 > 表示法）中的命令定义。

语法：{attribute 'displaymode':=<displaymode>}

可能的定义如下：

- 二进制格式
  - {attribute 'displaymode':='bin'}
  - {attribute 'displaymode':='binary'}
- 十进制格式
  - {attribute 'displaymode':='dec'}
  - {attribute 'displaymode':='decimal'}
- 十六进制格式
  - {attribute 'displaymode':='hex'}
  - {attribute 'displaymode':='hexadecimal'}

插入位置：在变量的声明行上方一行

示例：

```
VAR
    {attribute 'displaymode':='hex'}
    nVar1 : DWORD;
END_VAR
```

另请参见：

- TC3 用户界面文档：命令：显示模式 - 二进制、十进制、十六进制 [▶ 894]

16.8.2.12 属性“enable_dynamic_creation”

为了将 __NEW 运算符与功能块或 DUT 一起使用，必须使用该编译指示。

语法：{attribute 'enable_dynamic_creation'}

插入位置：在功能块或 DUT 的声明部分上方第 1 行

另请参见：

- 引用编程：__NEW [▶ 674]

16.8.2.13 属性“estimated-stack-usage”

该编译指示会传输堆栈大小要求的估计值。

采用递归调用的方法不能承受堆栈检查，因为无法确定堆栈使用情况。因此，系统会对这些方法发出警告。如要抑制该警告，您可以使用属性为方法提供堆栈大小要求的估计值（以字节为单位）。然后，该方法可成功通过堆栈检查。

语法：{attribute 'estimated-stack-usage' := '<estimated stack size in bytes>'}

插入位置：在方法的声明部分上方第 1 行

示例：

```
{attribute 'estimated-stack-usage' := '99'} // 99 bytes
METHOD SampleMethod : INT
VAR_INPUT
END_VAR
```

发出的警告：

对于不带“estimated-stack-usage”属性的递归方法，发出的警告如下：

“C0298：由于从‘<FB.Method>’开始的递归调用，堆栈使用量的计算不完整”

在与上次构建项目相比没有变化的项目中，只有在使用重建项目命令时（而不是在使用构建项目命令时）才会发出警告。

设置堆栈大小：

在 TwinCAT 项目的实时设置（Settings tab > Global Task Config/Maximum Stack Size [KB]）（设置选项卡 > 全局任务配置/最大堆栈大小 [KB]）中可以配置最大可能的堆栈大小的值。例如，设置的最大值并非完全可供 PLC 项目使用，但 TwinCAT 运行时系统可以使用其中的一部分。

独立的 PLC 项目无堆栈计算

由于与系统管理器分离，因此无法为独立的 PLC 项目计算堆栈使用量。

递归方法调用

在相关实现过程中，1 个方法可以调用自身：直接借助 THIS 指针，或者借助分配的功能块的局部变量。

注意

由于可能的堆栈溢出而导致设备停机

意外的深度递归可能会导致堆栈溢出，从而导致设备停机。

- 递归主要用于处理递归数据类型，例如，链表。

示例：阶乘的计算

在功能块 FB_Factorial 中，通过不同的方式可以计算数字的阶乘。各种计算均使用单独的方法进行。

- 方法 Iterative：迭代
- 方法 Pragmaed：递归，抑制警告
- 方法 Recursive：递归

在重建项目时，只有方法 Recursive 会生成警告 C0298。

结构 ST_FactorialResult 用于保存值：

```
// Contains the data of the factorial calculation of nNumber.
TYPE ST_FactorialResult :
STRUCT
    nNumber    : USINT;
    nIterative : UDINT;
    nRecursive : UDINT;
    nPragmaed  : UDINT;
END_STRUCT
END_TYPE
```

功能块 FB_Factorial 用于计算阶乘：

```
// Factorial calculation in different ways
FUNCTION_BLOCK FB_Factorial
VAR
     nNumberIterative    : UINT;
END_VAR
```

属性 nNr 包含设置函数，用于为迭代法传输参数：

```
{attribute 'monitoring' := 'variable'}
PROPERTY nNr : UINT
```

```
nNumberIterative := nNr;
```

迭代计算法：

```
// Iterative calculation
METHOD PUBLIC Iterative : UDINT
VAR
    nCnt : UINT;
END_VAR
```

```
Iterative := 1;

IF nNumberIterative > 1 AND nNumberIterative <= 12 THEN
    FOR nCnt := 1 TO nNumberIterative DO
        Iterative := Iterative * nCnt;
    END_FOR;
    RETURN;
ELSE
    RETURN;
END_IF
```

带属性的递归计算法，用于抑制警告 C0298：

```
// Recursive calculation with suppressed warning
{attribute 'estimated-stack-usage' := '200'}
METHOD PUBLIC Pragmaed : UDINT
VAR_INPUT
    nNumber : USINT;
END_VAR
```

```
Pragmaed := 1;

IF nNumber > 1 AND nNumber <= 12 THEN
    Pragmaed := nNumber * THIS^.Pragmaed(nNumber := (nNumber - 1));
    RETURN;
ELSE
    RETURN;
END_IF
```

例如，用于指定估计堆栈大小要求的编译指示参数被设置为 200 字节。计算方法如下：

每次方法调用的堆栈使用量：
返回值 UDINT + 输入参数 USINT + 方法指针 = 4 字节 + 1 字节 + 8 字节 = 13 字节

最多调用 12 次的堆栈使用量：
12 * 13 字节 = 156 字节，四舍五入为 200 字节

递归计算法：

```
// Recursive calculation
METHOD PUBLIC Recursive : UDINT
VAR_INPUT
    nNumber : USINT;
END_VAR
```

```
Recursive := 1;
IF nNumber > 1 AND nNumber <= 12 THEN
    Recursive := nNumber * THIS^.Recursive(nNumber := (nNumber - 1) );
    RETURN;
ELSE
    RETURN;
END_IF
```

主程序 MAIN：

```
PROGRAM MAIN
VAR
    fbFactorial : FB_Factorial;
    stFactorial : ST_FactorialResult := (nNumber := 9);
END_VAR
```

```
// Iterativ
fbFactorial.nNr        := stFactorial.nNumber;
stFactorial.nIterative := fbFactorial.Iterative();

// Recursive
stFactorial.nRecursive := fbFactorial.Recursive(nNumber := stFactorial.nNumber);

// Pragmaed
stFactorial.nPragmaed  := fbFactorial.Pragmaed(nNumber := stFactorial.nNumber);
```

16.8.2.14 属性“ExpandFully”

该编译指示会导致用作引用可视化的输入变量的数组的组件在可视化属性对话框中可视化。

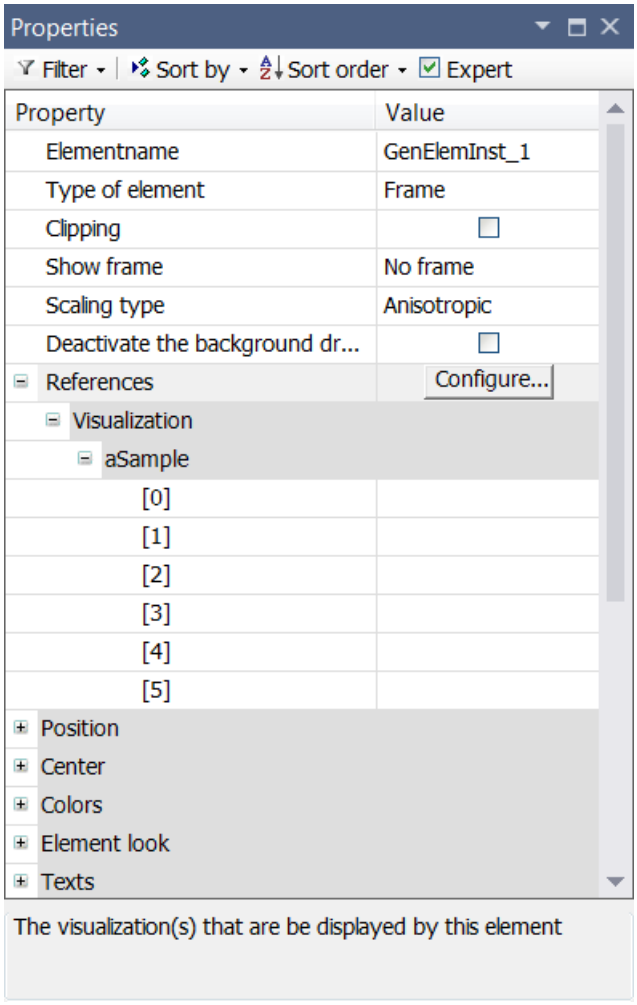
语法：{attribute 'ExpandFully'}

插入位置：在数组的声明行上方一行

示例：

在可视化 visu_main 中的 1 个边框中将插入可视化 visu。aSample 在 visu 的接口编辑器中被定义为输入变量，因此稍后可在 visu_main 中的边框属性对话框中进行赋值。如要在该属性对话框中也使用 aSample 的各个组件，您必须在 aSample 之前直接在 visu 的接口编辑器中插入“ExpandFully”属性。在 visu 的接口编辑器中的声明：

```
VAR_INPUT
    {attribute 'ExpandFully'}
    aSample : ARRAY[0..5] OF INT;
END_VAR
```



16.8.2.15 属性“global_init_slot”

该编译指示定义了编程块和全局变量列表的初始化顺序。GVL 或 POU 中的变量从上到下进行初始化。对于多个全局变量列表的情况，没有定义初始化顺序。

对于具有字面值的初始化，例如 1、“hello”、3.6 或基本数据类型的常量，初始化的顺序无关紧要。但是，如果初始化中存在相互依赖关系，则您必须设置初始化顺序。为此，您可以使用属性“global_init_slot”为 GVL 或 POU 分配 1 个已定义的初始化插槽。

常量在变量之前进行初始化，顺序与变量相同。在初始化期间，首先根据 <slot> 的值对签名（功能块、GVL）进行排序。然后生成用于初始化常量的代码，之后再生成用于初始化变量的代码。

因此，初始化可分为以下几个阶段：

1. 签名根据初始化插槽进行排序。通过属性“global_init_slot”可以对插槽进行隐式定义或显式定义。
2. 然后，对所有常量进行初始化。这是按照插槽的顺序完成的。没有常量的签名将被跳过。
3. 然后，再按照插槽的顺序对变量进行初始化。

语法： {attribute 'global_init_slot' := '<slot>'}

<slot>：整数值，用于定义调用顺序中的位置。POU（程序、功能块）的默认值为 50000。GVL 的默认值为 49990。静态变量的默认值为 49980。值越小，初始化越早。

插入位置： 该编译指示始终会影响整个 GVL 或 POU，因此必须位于 VAR_GLOBAL 声明或 POU 声明的上方。

如果为多个编程块分配了相同的“global_init_slot”属性值，则仍然没有定义它们的初始化顺序。为了避免影响 TwinCAT 3 的系统行为，请使用 40000 以上的值。

示例：

该项目包含 2 个全局变量列表 GVL1 和 GVL2。MAIN 程序使用这 2 个列表中的变量。对于变量 nA 的初始化，GVL1 使用变量 nB，该变量在 GVL2 中初始化的值为 1000：

GVL1：

```
VAR_GLOBAL   //49990
    nA : INT := GVL2.nB*3;
END_VAR
```

GVL2：

```
VAR_GLOBAL   //49990
    nB : INT := 1000;
    nC : INT := 10;
END_VAR
```

MAIN 程序：

```
PROGRAM MAIN  //50000
VAR
    nVar1 : INT := GVL1.nA;
    nVar2 : INT;
END_VAR
```

```
nVar1 := nVar + 1;
nVar2 := GVL2.nC;
```

在这种情况下，编译器会发出错误，因为在初始化 GVL2 之前，使用 GVL2.nB 对 GVL1.nA 进行初始化。通过将 global_init_slot 属性设置为初始化顺序中 GVL1 之前的 GVL2 的位置，您可以避免出现这种情况。

为此，GVL2 的 global_init_slot 值必须小于 GVL1 的值（默认值为 49990），且大于 40000（为系统功能保留）。
即：40001 <= GVL2 的 global_init_slot 值 <= 49989。

GVL2：

```
{attribute 'global_init_slot' := '40500'}
VAR_GLOBAL
    nB : INT := 1000;
    nC : INT := 10;
END_VAR
```

即使不使用编译指示，在 MAIN 的实现部分中使用 GVL2.nC 也无关紧要，因为这 2 个 GVL 总是在程序之前进行初始化。

16.8.2.16 属性“hide”

该编译指示可防止在 TwinCAT 界面中显示变量或程序元素。在在线模式下，它们在输入助手或声明部分中不可见，例如，您无法使用交叉引用搜索找到它们，也无法对它们使用调试功能（例如，步进或断点）。

请注意，使用“hide”或“hide_all_locals”属性声明的变量无法保存为持久变量。此外，对于“已隐藏的”变量，还可防止生成相关的过程映像（已分配的输入/输出）。此外，系统不会为这些变量生成（ADS）符号。换句话说，符号访问会被阻止。

“hide”的效果也会影响作为 .library 存在的库中的变量和签名。

语法：{attribute 'hide'}

插入位置：在签名上方一行、在变量的声明行上方一行（此处只有直接跟在编译指示后面的变量才会呈现不可见的状态），或者在 ST 动作/转换的情况下，直接在动作/转换的开始处

示例：

功能块 FB_Sample 使用属性 {attribute 'hide'}：

```
FUNCTION_BLOCK FB_Sample
VAR_INPUT
    nA          : INT;
    {attribute 'hide'}
    bInvisible : BOOL;
    bVisible    : BOOL;
END_VAR
VAR_OUTPUT
    nB : INT;
END_VAR
```

在主程序中定义了功能块 FB_Sample 的实例。

```
PROGRAM MAIN
VAR
    fbSample    : FB_Sample;
END_VAR
```

在实现 fbSample 的输入值的同时，在输入“fbSample”时会打开 **List components**（列出组件）功能。(在 MAIN 的实现部分）显示了变量 nA、bVisible 和 nB，但没有显示隐藏的变量 bInvisible。

编译指示 hide_all_locals [▶ 745] 可用于隐藏声明中的所有局部变量。

如果使用编译指示"隐藏编译库中使用的变量和签名"，则这些变量和签名不会在库管理器中显示。

另请参见：

- 属性“hide_all_locals” [▶ 745]
- 属性“conditionalshow” [▶ 737]
- 属性“conditionalshow all locals” [▶ 738]

16.8.2.17 属性“hide_all_locals”

该属性的作用与属性“hide” [▶ 744]类似。唯一的区别在于，“hide”属性只影响 1 个变量，而签名的所有局部变量都会受到“hide_all_locals”属性的影响。

请注意，使用“hide”或“hide_all_locals”属性声明的变量无法保存为持久变量。此外，对于“已隐藏的”变量，还可防止生成相关的过程映像（已分配的输入/输出）。此外，系统不会为这些变量生成（ADS）符号。换句话说，符号访问会被阻止。

语法：{attribute 'hide_all_locals'}

插入位置：在 POU 的声明部分上方第 1 行

示例：

功能块 FB_Sample 使用该属性：

```
{attribute 'hide_all_locals'}
FUNCTION_BLOCK FB_Sample
VAR_INPUT
    nA      : INT;
END_VAR
VAR_OUTPUT
    bB      : BOOL;
END_VAR
VAR
    nC,nD   : INT;
    bE      : BOOL;
END_VAR
```

在功能块 FB_Sample 中使用属性“hide_all_locals”时，属性“hide”的效果会影响该功能块的所有局部变量（nC、nD 和 bE）。

另请参见：

- 属性“hide” [▶ 744]

16.8.2.18 属性“init_namespace”

该编译指示会导致在项目中使用在库功能块中使用该编译指示声明的 STRING 或 WSTRING 类型的变量时，使用该库的当前命名空间对其进行初始化。

语法：{attribute 'init_namespace'}

插入位置：在库功能块中的变量的声明行上方一行

示例：

功能块 FB_Sample 具有所需的属性：

```
FUNCTION_BLOCK FB_Sample
VAR_OUTPUT
    {attribute 'init_namespace'}
    sNamespace : STRING;
END_VAR
```

在主程序 MAIN 中定义了功能块 FB_Sample 的实例 fbSample。

```
PROGRAM MAIN
VAR
    fbSample : FB_Sample;
    sMyNamespace : STRING;
END_VAR
sMyNamespace := fbSample.sNamespace;
```

使用当前命名空间对变量 sNamespace 进行初始化，例如，Tc3_TestLib。该值将被分配给主程序中的 sMyNamespace。

另请参见：

- 库管理器 [▶ 254]

16.8.2.19 属性“init_on_onlchange”

每次在线更改时，该编译指示都会导致应用该编译指示的变量进行初始化。

快速在线更改

对于微小更改（例如，在实现部分中，不需要切换变量），可执行“快速在线更改”。在这种情况下，仅会编译和重新加载修改后的功能块。特别是，在这种情况下不会生成初始化代码。这也意味着不会生成带“init_on_onlchange”属性的初始化变量代码。通常，这不会产生任何影响，因为该属性一般用于初始化带地址的变量，但变量在快速在线更改时无法更改其地址。

如要确保将 init_on_onlchange 属性应用于整个应用程序代码，可使用 no_fast_online_change 编译器定义停用 PLC 项目的快速在线更改。为此，可在 PLC 项目属性的 Compile [▶ 844]（编译）类别中插入定义。

“init_on_onlchange”属性对单个 FB 变量没有影响

属性“init_on_onlchange” [▶ 746] 仅适用于功能块的全局变量、程序变量和局部静态变量。

如要在在线更改期间重新初始化功能块，务必使用属性来声明功能块实例。对于功能块中的单个变量，不会对属性进行评估。

语法：{attribute 'init_on_onlchange' }

插入位置：在变量的声明行上方一行

16.8.2.20 属性“initialize_on_call”

该编译指示可应用于输入变量。每次调用功能块时，它都会导致功能块的输入变量进行初始化。如果受影响的输入变量需要 1 个指针，而该指针在在线更改过程中已被移除，则该变量使用 0 进行初始化。

语法：{attribute 'initialize_on_call'}

插入位置：始终位于整个功能块的声明部分中的第 1 行，此外，还可位于单个输入变量的声明上方一行

示例：

如果输入需要 1 个指针，且该输入与该属性相关联，则每次调用功能块时都要对该指针进行重新初始化。这可用于防止使用在在线更改期间可能已失效的指针。

```
{attribute 'initialize_on_call'}
FUNCTION_BLOCK FB_Test
VAR_INPUT
    {attribute 'initialize_on_call'}
    pSetpoint : POINTER TO LREAL := 0;
END_VAR
VAR_OUTPUT
END_VAR
```

在调用功能块时应该分配指针。

```
fbTest(pSetpoint := ADR(fSetpoint));
```

16.8.2.21 属性“instance-path”

该编译指示可应用于局部 STRING 变量，并会导致该局部 STRING 变量按照其所属 POU 的设备树路径依次进行初始化。这对错误消息十分有用。应用该编译指示需要对相应的 POU 应用属性“reflection”，并对 STRING 变量本身应用附加属性“no_init”。

语法： {attribute 'instance-path'}

插入位置： 在 STRING 变量的声明行上方一行

示例：

以下功能块包含属性“reflection”、“instance-path”和“noinit”。

```
{attribute 'reflection'}
FUNCTION_BLOCK FB_Sample
VAR
    {attribute 'instance-path'}
    {attribute 'noinit'}
    sPath : STRING;
END_VAR
```

在 MAIN 程序中定义了功能块 FB_Sample 的实例 fbSample：

```
PROGRAM MAIN
VAR
    fbSample : FB_Sample;
    sMyPath : STRING;
END_VAR
fbSample();
sMyPath := fbSample.sPath;
```

在实例 fbSample 进行初始化之后，字符串变量 sPath 将被分配为实例 fbSample 的路径，例如，“<project>.MAIN.fbSample”。在主程序中，该路径被分配给变量 sMyPath。

您可以根据需要定义字符串的长度（在必要时，可超过 255 个字符）。但请注意，如果将字符串分配给数据类型较小的变量，则该字符串会被截断（从后面截断）。

另请参见：

- 属性“reflection” [▶ 761]
- 属性“noinit” [▶ 754]

16.8.2.22 属性“is_connected”

“is_connected”编译指示用于识别 Boolean 功能块变量，该变量提供了关于功能块的分配输入在调用功能块实例时是否会接收到赋值的信息。

应用该编译指示的前提是，属性“reflection”已应用于受影响的功能块。

语法： {attribute 'is_connected' := '<input variable>'}

插入位置： 在单个 Boolean 功能块变量的声明上方一行

示例：

在功能块 FB 中，为每个输入变量（nIn1 和 nIn2）声明 1 个局部变量，并且该局部变量前面带有与输入变量规范相关的属性“is connected”。为功能块本身分配编译指示属性“reflection”。

在调用功能块的实例时，如果分配给它的输入已收到赋值，则局部变量变为 TRUE。

```
{attribute 'reflection'}
FUNCTION_BLOCK FB_Sample
VAR_INPUT
    nIn1: INT;
    nIn2: INT;
END_VAR
VAR
    {attribute 'is_connected' := 'nIn1'}
    bIn1Connected: BOOL;
    {attribute 'is_connected' := 'nIn2'}
    bIn2Connected: BOOL;
END_VAR
```

假设：在调用功能块实例时，nIn1 从外部接收到赋值；此时 nIn 2 没有接收到赋值。创建以下代码：

```
bIn1Connected := TRUE;
```

```
bIn2Connected := FALSE;
```

16.8.2.23 属性“linkalways”

该编译指示会导致编译器选择相关对象，因此始终包含在编译器信息中。这意味着对象始终会在 PLC 上进行编译和加载。在对象属性中，**Advanced**（高级）类别中的 **Always link**（始终链接）选项具有相同的效果。

语法：{attribute 'linkalways'}

插入位置：

POU：在 POU 的声明部分上方第 1 行

GVL：在声明部分中带有关键字 VAR_GLOBAL 的行上方一行

示例：

使用属性“linkalways”的 POU（此处：程序）的实现：

```
{attribute 'linkalways'}
PROGRAM PRG_Test
    bSample : BOOL;
END_VAR
```

使用属性“linkalways”的全局变量列表的实现：

```
{attribute 'linkalways'}
VAR_GLOBAL
    nVar1 : INT;
    nVar2 : INT;
END_VAR
```

16.8.2.24 memory_check 属性

该属性可限制由“**检查活跃应用程序内存**”命令触发的内存检查。它可通过 ignore 值将程序块签名的任意一个变量或所有变量从该检查中排除。

插入位置：

- 在任意一个变量的声明上方的一行中
- 在程序块声明部分上方的一行中

语法：

```
{attribute 'memory_check' := 'ignore'}
```

示例：

在内存检查过程中，不会考虑功能块 FB_01 的任何变量。

```
{attribute 'memory_check' := 'ignore'}
FUNCTION_BLOCK FB_01
VAR
```

```
    sVar : STRING;
    bVar : BOOL;
END_VAR
```

16.8.2.25 属性“minimal_input_size”

该编译指示定义了堆栈上输入的最小大小。为了实现与 32 位系统的 C++ 编译器的 C++ 兼容，每个输入在堆栈上至少占用 4 个字节。

语法：{attribute 'minimal_input_size'}

插入位置：在方法声明上方一行

示例：

实现 C++ 兼容接口的 PLC 功能块方法的声明：

```
{attribute 'c++_compatible'}
{attribute 'minimal_input_size' := '4'}
{attribute 'pack_mode' := '4'}
METHOD Method1 : HRESULT
VAR_INPUT
    nParameter1 : INT;
END_VAR
```

另请参见：

- 属性“c++_compatible” [▶ 735]

16.8.2.26 属性“monitoring”

该编译指示使您能够在 IEC 编辑器的在线视图或监视列表中监控属性值或函数调用。这有 2 个可能的属性值：“variable”和“call”。

语法：

{attribute 'monitoring' := 'variable'}

或

{attribute 'monitoring' := 'call'}

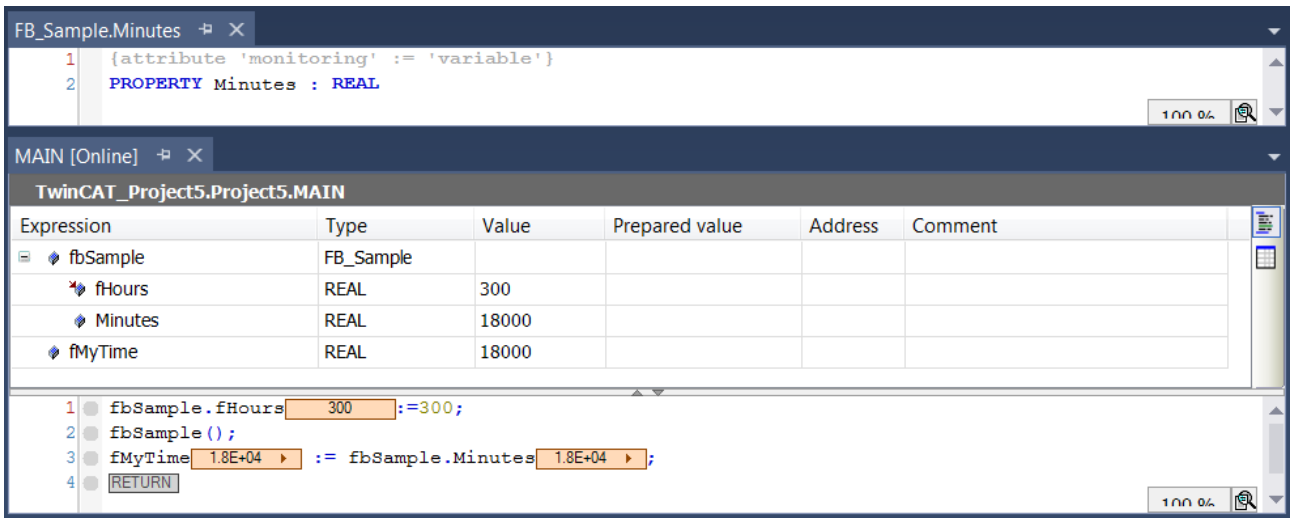
监控属性

在功能块或程序的在线视图中，除了局部变量之外，您还可以监控下级属性。因此，您可以监控 Get 和 Set 方法的值。

在属性的声明中添加 {attribute 'monitoring' := 'variable'} 或 {attribute 'monitoring' := 'call'} 编译指示。然后，属性的当前值会自动显示在 IEC 编辑器或监视列表中。

示例：

TwinCAT 在在线操作过程中在调用位置内联显示 Minutes 的值，因为 Minutes 属性的声明中包含编译指示 {attribute 'monitoring' := 'variable'}。



对于每个应用程序，应仔细检查哪个属性编译指示适合显示所需的值。这取决于是否在属性中执行涉及该变量的进一步操作。

编译指示 1 {attribute 'monitoring':='variable'}：

为属性创建 1 个隐式变量，当 PLC 程序调用 Set 或 Get 方法时，始终为该变量分配当前的属性值。在该变量中最后存储的值会在监控中显示。

编译指示 2 {attribute 'monitoring':='call'}：

您只能将该属性用于返回简单数据类型或指针的属性，不能将其用于结构化类型。通过直接调用属性可以读取或写入要监控的值。这意味着运行时系统的监控服务会执行属性函数的 Get 或 Set 方法。

如果您选择这种监控类型，而不是使用隐式变量（如使用 {attribute monitoring': ='variable'}），则您必须考虑可能的副作用。在属性函数中执行附加操作时，可能会产生此类副作用。

上下文菜单命令 **Add to Watch**（添加到监视）会导致将光标当前所在位置的变量直接添加到在线模式下的监视列表中。

不支持强制或写入函数。不过，您可以为每个函数添加 1 个额外的输入参数，将其用作内部强制标志，从而隐式实现强制。

在紧凑型运行时系统中无法进行函数监控。

注意：Windows CE 不是紧凑型运行时系统。紧凑型运行时系统不支持多任务处理，通常没有文件和操作系统。例如，紧凑型运行时系统的典型处理器是 ARM Cortex™-M。

16.8.2.27 属性“no_assign”、属性“no_assign_warning”

TwinCAT 3.1 Build 4026 及以上可用

在将功能块的 1 个实例分配给相同功能块的另一个实例时，“no_assign”编译指示会导致发出编译器错误。如果功能块中包含指针，应尽可能避免类似的分配，因为分配时复制的指针会导致问题。

“no_assign_warning”编译指示的作用与“no_assign”编译指示相同，但有一点不同：它发出的是编译器警告，而不是编译器错误。

语法： {attribute 'no_assign'} 或 {attribute 'no_assign_warning'}

插入位置： 在功能块的声明部分中的第 1 行

示例：

由于 FB_Test 功能块包含指针，因此应在 FB_Test 功能块的声明中添加“no_assign”属性，以避免分配功能块实例：

```
{attribute 'no_assign'}
FUNCTION_BLOCK FB_Test
VAR
    pVar    : POINTER TO LREAL;
…
```

分配功能块实例：

```
VAR_GLOBAL
    fbInst1 : FB_Test;
END_VAR
```

```
PROGRAM MAIN
VAR
    fbInst2 : FB_Test := fbInst1;
END_VAR
```

然后会输出以下编译器错误：

```
Assignment not allowed for type FB_Test
```

16.8.2.28 属性“no_check”

该编译指示可阻止为 POU 调用检查函数（隐式检查的 POU）。由于检查函数可能会影响程序的处理速度，因此将该属性应用于已经检查过或定期调用的功能块可能比较合理。

语法： {attribute 'no_check'}

插入位置： 在 POU 的声明部分中的第 1 行

- 对象用于隐式检查的 POU [▶ 151]

16.8.2.29 属性“no_copy”

如果实例（例如，POU）在在线更改期间在内存中发生移动，则需要重新分配该实例。实例中包含的变量值会被复制，因此在线更改后的变量值与在线更改前的变量值相同。如果在在线更改期间必须移动实例/变量，则会出现 1 个对话框，提供有关效果的信息，并允许中止在线更改。

该编译指示的效果是，在在线更改期间复制移动实例的过程中，不会复制实例中包含的变量的变量值，而是在在线更改过程中重新初始化变量。对于局部指针变量来说，这可能是有意义的，因为它指向的变量刚刚因在线更改而移动，因此具有修改后的地址。

语法： {attribute 'no_copy'}

插入位置： 在受影响变量的声明行上方一行

16.8.2.30 属性“no_explicit_call”

为 POU 添加该编译指示可防止任何直接调用该 POU 的情况。

例如，这与面向对象的功能块有关，这些功能块可以通过方法（方法功能块）和属性（属性功能块）进行专门控制。在这种情况下，对 FB 主体的调用将无效或不被允许。如果仍然直接调用使用属性“no_explicit_call”进行声明的功能块主体，则编译器会发出编译错误，以指明功能块的使用不正确。

语法： {attribute 'no_explicit_call' := '<text value>'}

插入位置： 在功能块的声明部分中的第 1 行。

示例：

在下面的示例中声明了 1 个功能块，它的 FB 主体可以被直接调用（“FB_DirectCallAllowed”）。还声明了 1 个功能块，它的 FB 主体不能被直接调用（FB_DirectCallNotAllowed）。因此，使用属性“no_explicit_call”对该功能块进行声明；为该属性添加注释文本：“do not call this POU directly”（请勿直接调用此 POU）。此外，为这 2 个功能块都添加了 1 个示例方法。

这 2 个功能块各被实例化 1 次并被直接调用。对于 FB 实例，也会调用示例方法。

在编译该程序时，如果直接调用实例 fbDirectCallNotAllowed，则会生成编译错误（在这种情况下：“do not call this POU directly”（请勿直接调用此 POU））。编译器不会阻止其他 3 个调用。

功能块 FB_DirectCallAllowed：

```
FUNCTION_BLOCK FB_DirectCallAllowed
VAR
END_VAR

METHOD SampleMethod
VAR_INPUT
END_VAR
```

功能块 FB_DirectCallNotAllowed：

```
{attribute 'no_explicit_call' := 'do not call this POU directly'}
FUNCTION_BLOCK FB_DirectCallNotAllowed
VAR
END_VAR
```

```
METHOD SampleMethod
VAR_INPUT
END_VAR
```

MAIN 程序：

```
PROGRAM MAIN
VAR
    fbDirectCallAllowed     : FB_DirectCallAllowed;
    fbDirectCallNotAllowed  : FB_DirectCallNotAllowed;
END_VAR
```

```
fbDirectCallAllowed();                  // => OK
fbDirectCallAllowed.SampleMethod();     // => OK

fbDirectCallNotAllowed();               // => NOK: generates compile error “do not call this POU
directly”
fbDirectCallNotAllowed.SampleMethod();  // => OK
```

16.8.2.31 属性“no_virtual_actions”

该编译指示可应用于从 SFC 中实现的功能块派生的功能块，并使用该基类的基本 SFC 序列。由此产生的动作会显示与方法相同的虚拟行为。这意味着派生类中基类中动作的实现可以被特定的自定义实现所取代。

如果您将该编译指示应用于基类，则您的动作会受到保护，不会被重载。

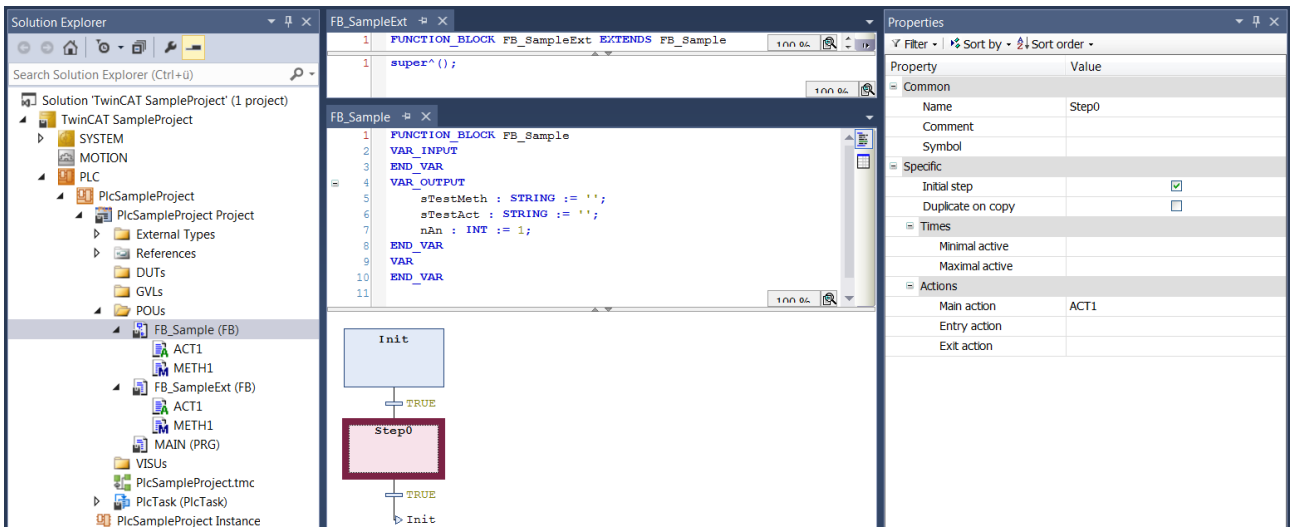
语法：{attribute 'no_virtual_actions'}

插入位置：在功能块的声明部分中的顶行

示例：

功能块 FB_Sample 是派生功能块 FB_SampleExt 的基类。

派生类 FB_SampleExt 使用特殊变量 SUPER 来调用以 SFC 编写的基类流程。



该流程的示例实现仅限于初始步骤，然后是具有相关步骤动作 ACT1 的单个步骤。该步骤具有相关步骤动作，可处理输出变量的分配。

```
nAN := nAn + 1; // Counting the action calls
sTestAct:='father_action';
METH1();  // Call of the method METH1 in order to set the string variable sTestMeth
```

在派生类 FB_SampleExt 的情况下，步骤动作会被 ACT1 的特殊实现所取代。ACT1 与原始版本的不同之处仅在于，将字符串“child_action”而不是“father_action”分配给变量 sTestAct。

此外，在基类中将字符串“father_method”分配给变量 sTestMeth 的方法METH1 会被覆盖，因此，现在已为 sTestMeth 分配值 sTestMeth。MAIN 程序调用功能块 FB_SampleExt 的实例“fbSampleExt”。正如预期的那样，字符串的值反映了派生类的动作和方法的调用：

MAIN [Online]

TwinCAT_SampleProject.PlcSampleProject.MAIN

Expression	Type	Value
fbSampleExt	FB_SampleExt	
sTestMeth	STRING	'child_method'
sTestAct	STRING	'child_action'
nAn	INT	185

但现在，您在基类前面加上编译指示 {attribute’ no_virtual_actions’}：

```
{attribute 'no_virtual_actions'}
FUNCTION_BLOCK FB_Sample...
```

这会改变行为：虽然派生类的实现会继续用于方法 METH1，但现在调用步骤动作会导致调用基类的动作 ACT1。因此，现在为 sTestAct 分配值“father_action”：

MAIN [Online]

TwinCAT_SampleProject.PlcSampleProject.MAIN

Expression	Type	Value
fbSampleExt	FB_SampleExt	
sTestMeth	STRING	'child_method'
sTestAct	STRING	'father_action'
nAn	INT	177

16.8.2.32 属性“no-exit”

该编译指示会抑制对功能块的某些实例调用 FB_exit 方法。

语法： {attribute ’no-exit’}

插入位置：在功能块实例的声明前一行

示例：

在方法 FB_exit 中添加功能块 FB_Sample。在 MAIN 程序中创建功能块 FB_Sample 的 2 个实例。

```
PROGRAM MAIN
VAR
    fbSample1 : FB_Sample;
    {attribute 'no-exit'}
    fbSample2 : FB_Sample;
END_VAR
```

调用了 fbSample1.FB_exit，但未调用 fbSample2.FB_exit。

另请参见：

- 方法 FB_init、FB_reinit 和 FB_exit [▶ 786]

16.8.2.33 属性“noflow”/“flow”

该编译指示可识别被排除在流程控制表示之外的区域。请参见 命令：流控制 [▶ 892]。

语法：

```
{noflow}
```

```
{flow}
```

插入位置：在不应由流程控制表示的区域上方和下方的行。

示例：

```
IF Test THEN
IF Test1 THEN
{noflow}
IF Test2 THEN
;
END_IF
{flow}
END_IF
END_IF
```

16.8.2.34 属性“noinit”

该编译指示可应用于未隐式初始化的变量。

语法：

```
{attribute 'no_init'}
```

```
{attribute 'no-init'}
```

```
{attribute 'noinit'}
```

插入位置：在受影响变量的声明行上方一行

示例：

```
PROGRAM MAIN
VAR
    nA : INT;
    {attribute 'no_init'}
    nB : INT;
END_VAR
```

如果重置相应的应用程序，则整数变量 nA 将再次隐式初始化为 0，而变量 nB 将保留其当前值。

16.8.2.35 属性“obsolete”

如果在项目中使用数据类型（结构、功能块等），则该编译指示会在编译期间对数据类型定义发出已定义的警告。例如，您可以用它来指明某个数据类型不再有效，因为接口可能已经改变，并且这应该反映在项目中。

与消息编译指示相比，该警告是为 1 个数据类型的所有实例集中定义的。

语法： {attribute 'obsolete' := 'user defined text'}

插入位置： 在数据类型定义的行或其上方一行。

示例：

在功能块 FB_Sample 的定义中插入该编译指示：

```
{attribute 'obsolete' := 'datatype FB_Sample() not valid!'}
FUNCTION_BLOCK FB_Sample
VAR_INPUT
    nVar : INT;
END_VAR
```

如果您使用 FB_Sample 作为数据类型，例如，在 fbSample : FB_Sample; 中，则在编译项目时会发出警告：“datatype FB_Sample not valid”（数据类型 FB_Sample 无效）。

另请参见：

- 消息编译指示： [▶ 733]

16.8.2.36 属性“pack_mode”

该编译指示定义了数据结构在分配过程中的打包方式。属性必须插入到数据结构的上方，并影响整个结构的打包。

语法： {attribute 'pack_mode' := '<pack mode value>'}

<pack mode value> 的可能值：

<pack mode value>	相关包类型	描述
0	对齐	将所有变量分配给字节地址；不会出现内存间隙。
1	1 字节对齐	
2	2 字节对齐	存在以下情况 • 将 1 字节变量分配给字节地址 • 将 2 字节变量分配给可被 2 整除的地址。可能出现的最大间隙为 1 字节 • 将 4 字节变量分配给可被 2 整除的地址。可能出现的最大间隙为 1 字节 • 将 8 字节变量分配给可被 2 整除的地址。可能出现的最大间隙为 1 字节 • 字符串始终在字节地址上。不会产生间隙。
4	4 字节对齐	存在以下情况 • 将 1 字节变量分配给字节地址 • 将 2 字节变量分配给可被 2 整除的地址。可能出现的最大间隙为 1 字节 • 将 4 字节变量分配给可被 4 整除的地址。可能出现的最大间隙为 3 字节 • 将 8 字节变量分配给可被 4 整除的地址。可能出现的最大间隙为 3 字节 • 字符串始终在字节地址上。不会产生间隙。
8	8 字节对齐	存在以下情况 • 将 1 字节变量分配给字节地址 • 将 2 字节变量分配给可被 2 整除的地址。可能出现的最大间隙为 1 字节 • 将 4 字节变量分配给可被 4 整除的地址。可能出现的最大间隙为 3 字节 • 将 8 字节变量分配给可被 8 整除的地址。可能出现的最大间隙为 7 字节 • 字符串始终在字节地址上。不会产生间隙。

插入位置：在数据结构的声明上方一行

根据结构配置的不同，各个模式之间的内存分割可能没有区别。在这种情况下，pack_mode = 4 结构的内存分配可以与 pack_mode = 8 结构的内存分配相对应。

结构的数组：如果结构分组为数组，则在结构的末尾插入字节，以便下一个结构再次对齐。

示例 1

```
{attribute 'pack_mode' := '1'}
TYPE ST_MyStruct:
STRUCT
    bEnable         : BOOL;
    nCounter        : INT;
    nMaxSize        : INT;
    bMaxSizeReached : BOOL;
    bReset          : BOOL;
END_STRUCT
END_TYPE
```

为数据类型为 ST_MyStruct 的变量的内存区分配”对齐“：例如，如果您的组件 bEnable 的内存地址是 0x0100，则组件 nCounter 的地址为 0x0101，nMaxSize 的地址为 0x0103，bMaxSizeReached 的地址为 0x0105，bReset 的地址为 0x0106。如果 ’pack_mode’ := ’2’，则 nCounter 为 0x0102，nMaxSize 为 0x0104，bMaxSizeReached 为 0x0106，bReset 为 0x0107。

示例 2

```
STRUCT
    bVar1 : BOOL  := 16#01;
    nVar2 : BYTE  := 16#11;
    nVar3 : WORD  := 16#22;
    nVar4 : BYTE  := 16#44;
    nVar5 : DWORD := 16#88776655;
    nVar6 : BYTE  := 16#99;
    nVar7 : BYTE  := 16#AA;
    nVar8 : DWORD := 16#AA;
END_TYPE
```

	pack_mode = 0		pack_mode = 1		pack_mode = 2		pack_mode = 4		pack_mode = 8	
	变量	值	变量	值	变量	值	变量	值	变量	值
0	Var1	01	Var1	01	Var1	01	Var1	01	Var1	01
1	Var2	11	Var2	11	Var2	11	Var2	11	Var2	11
2	Var3	22	Var3	22	Var3	22	Var3	22	Var3	22
3	...	00	...	00	...	00	...	00	...	00
4	Var4	44	Var4	44	Var4	44	Var4	44	Var4	44
5	Var5	55	Var5	55						
6	...	66	...	66	Var5	55				
7	...	77	...	77	...	66				
8	...	88	...	88	...	77	Var5	55	Var5	55
9	Var6	99	Var6	99	...	88	...	66	...	66
10	Var7	AA	Var7	AA	Var6	99	...	77	...	77
11	Var8	AA	Var8	AA	Var7	AA	...	88	...	88
12	...	00	...	00	Var8	AA	Var6	99	Var6	99
13	...	00	...	00	...	00	Var7	AA	Var7	AA
14	...	00	...	00	...	00				
15					...	00				
16							Var8	AA	Var8	AA
17							...	00	...	00
18							...	00	...	00
19							...	00	...	00
20										
21										
22										
23										
24										
25										
26										
27										
28										
29										
30										
31										

示例 3

```
STRUCT
    nVar1 : BYTE  := 16#01;
    nVar2 : LWORD := 16#11;
    nVar3 : BYTE  := 16#22;
    nVar4 : BYTE  := 16#44;
    nVar5 : DWORD := 16#88776655;
    nVar6 : BYTE  := 16#99;
    nVar7 : BYTE  := 16#AA;
    nVar8 : WORD  := 16#AA;
END_TYPE
```

	pack_mode = 0		pack_mode = 1		pack_mode = 2		pack_mode = 4		pack_mode = 8	
	变量	值	变量	值	变量	值	变量	值	变量	值
0	Var1	01	Var1	01	Var1	01	Var1	01	Var1	01
1	Var2	11	Var2	11						
2	...	00	...	00	Var2	11				
3	...	00	...	00	...	00				
4	...	00	...	00	...	00	Var2	11		
5	...	00	...	00	...	00	...	00		
6	...	00	...	00	...	00	...	00		
7	...	00	...	00	...	00	...	00		
8	...	00	...	00	...	00	...	00	Var2	11
9	Var3	22	Var3	22	...	00	...	00	...	00
10	Var4	44	Var4	44	Var3	22	...	00	...	00
11	Var5	55	Var5	55	Var4	44	...	00	...	00
12	...	66	...	66	Var5	55	Var3	22	...	00
13	...	77	...	77	...	66	Var4	44	...	00
14	...	88	...	88	...	77			...	00
15	Var6	99	Var6	99	...	88			...	00
16	Var7	AA	Var7	AA	Var6	99	Var5	55	Var3	22
17	Var8	AA	Var8	AA	Var7	AA	...	66	Var4	44
18	...	00	...	00	Var8	AA	...	77		
19					...	00	...	88		
20							Var6	99	Var5	55
21							Var7	AA	...	66
22							Var8	AA	...	77
23							...	00	...	88
24									Var6	99
25									Var7	AA
26									Var8	AA
27									...	00
28										
29										
30										
31										

16.8.2.37 属性“parameterstringof”

该编译指示可用于使用可视化访问变量的实例名。

语法：{attribute 'parameterstringof' := '<variable>'}

插入位置：在变量的声明行上方一行

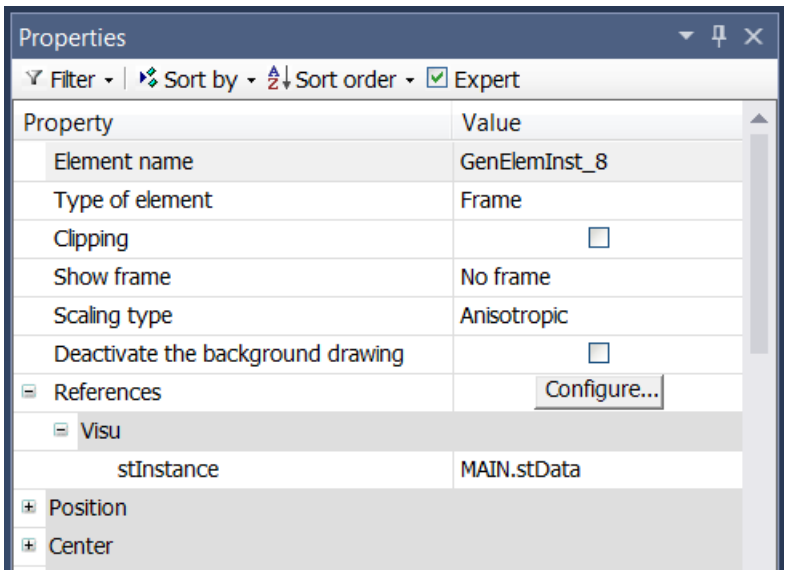
示例：

在主程序中，创建了用户定义的结构 ST_Data 的实例 stData：

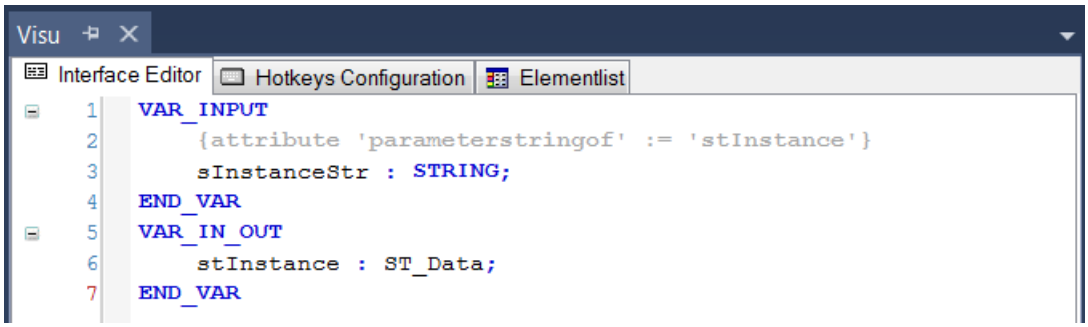
```
PROGRAM MAIN
VAR
    stData  : ST_Data;
END_VAR
```

该实例是可视化“Visu”的输入（在输入/输出参数 stInstance 中）。该可视化由另一个可视化“MainVisu”的边框引用。

“MainVisu”中的边框元素的设置如下所示：



在与可视化“Visu”相关联的接口编辑器中，对输入/输出变量 stInstance 和另一个输入变量 sInstanceStr 进行声明：



虽然 sInstanceStr 是 1 个输入变量，但在引用中并未将其列为输入（请参见上图）。这是因为变量 sInstanceStr 具有属性“parameterstringof”，因此会自动使用在属性中指定的变量名进行初始化。在本示例中，stInstance 是相应变量。因此，字符串变量 sInstanceStr 被设置为 MAIN.stData，现在可以在可视化“Visu”中使用，例如，用作占位符“%s”的文本变量。

16.8.2.38 属性“pin_presentation_order_inputs/outputs”

该编译指示在 CFC 和 FBD/LD 图形编辑器中进行评估，可使相关功能块的输入和输出以指定的顺序显示。通过以所需的顺序将输入/输出的名称分配给属性，可以对顺序进行编程。

语法：

```
{attribute 'pin_presentation_order_inputs' := '<First_Input_Name>, (<Next_Input_Name>, )*
( *, )? (<Next_Input_Name>,)* <Last_Input_Name>'}

{attribute 'pin_presentation_order_outputs' := '<First_Output_Name>, (<Next_Output_Name>,)*
( *, )? (<Next_Output_Name>,)* <Last_Output_Name>'}
```

- *
  可选的终端符号可作为所有未按显示顺序指定的输入/输出的占位符。如果终端符号缺失，则会将缺失的输入/输出附加在末尾。
- (...)?
  圆括号的内容是可选的。
- (...)*
  圆括号的内容是反复可选的，因此，它可以不出现，也可以出现 1 次或多次。

插入位置：在功能块的声明部分中的第 1 行

将属性“pin_presentation_order_inputs/outputs”与属性“pingroup”结合使用

如果使用编译指示 {attribute 'pingroup' := '<Group_Name>'}，则不会评估该编译指示。

示例：

1. 使用属性“pin_presentation_order_inputs/outputs”

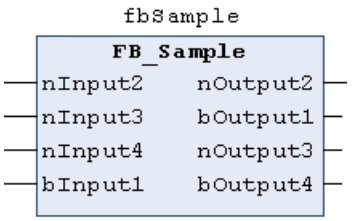
功能块 FB_Sample：

```
{attribute 'pin_presentation_order_inputs' := 'nInput2,*,bInput1'}
{attribute 'pin_presentation_order_outputs' := 'nOutput2,nOutput1'}
FUNCTION_BLOCK FB_Sample
VAR_INPUT
    bInput1 : BOOL;
    nInput2 : INT;
    nInput3 : INT;
    nInput4 : INT;
END_VAR
VAR_OUTPUT
    bOutput1 : BOOL;
    nOutput2 : INT;
    nOutput3 : INT;
    bOutput4 : BOOL;
END_VAR
```

程序 SampleProg：

```
PROGRAM SampleProg
VAR
    fbSample : FB_Sample;
END_VAR
```

在功能块实例 fbSample 的显示中，该编译指示会使输入和输出引脚进行如下排列：



另请参见：

- 属性“pingroup” [▶ 760]

16.8.2.39 属性“pingroup”

在功能块的声明中，该编译指示会将输入或输出引脚（参数）组合在一起。在 FBD/LD 编辑器中，以该方式定义的引脚组可以在添加的功能块中以折叠或展开的形式显示为 1 个单元。可分为几组，通过它们的名称可以进行区分。TwinCAT 会将每个功能块框的相应状态（折叠）与项目一起存储。

语法： {attribute 'pingroup' := '<group name>'}

插入位置： 在功能块的声明部分中受影响的输入或输出变量的声明上方一行。

示例：

功能块 FB_Sample：

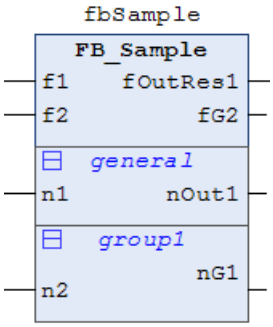
```
FUNCTION_BLOCK FB_Sample
VAR_INPUT
    f1 : REAL;
    {attribute 'pingroup' := 'general'}
    n1 : INT;
    {attribute 'pingroup' := 'group1'}
    n2 : INT;
    f2 : REAL;
END_VAR
VAR_OUTPUT
    fOutRes1 : REAL;
    {attribute 'pingroup' := 'general'}
```

```
    nOut1 : INT;
    {attribute 'pingroup' := 'group1'}
    nG1 : INT;
    fG2 : REAL;
END_VAR
```

程序 SampleProg：

```
PROGRAM SampleProg
VAR
    fbSample : FB_Sample;
END_VAR
```

在功能块实例 fbSample 的显示中，该编译指示会使输入和输出引脚进行如下分组：



另请参见：

- 属性“pin presentation order inputs/outputs” [▶ 759]

16.8.2.40 属性“qualified_only”

该编译指示的作用是，全局变量列表中的变量只能通过指定全局变量列表名称（例如，GVL.nVar）来寻址。这也适用于枚举类型的变量，它们只能通过指定枚举名称（例如，E_Sample.eMember）来访问。这对避免与局部变量混淆很有帮助。

请注意，GVL 只能作为 1 个整体分配属性。GVL 的单个变量范围不能用该属性进行声明。

语法：{attribute 'qualified_only'}

插入位置：在 GVL 中第 1 个 VAR_GLOBAL 上方一行

示例：

全局变量列表（GVL）：

```
{attribute 'qualified_only'}
VAR_GLOBAL
    nVar : INT;
END_VAR
```

在 POU 内，例如 MAIN，全局变量 nVar 只能使用前缀 GVL 进行寻址：

```
GVL.nVar := 5;
```

以下不完整的变量调用会产生错误：

```
nVar := 5;
```

16.8.2.41 属性“reflection”

该编译指示用于指明 TwinCAT 应该在其中搜索应用属性“instance-path”的局部 STRING 变量的功能块。编译器仅会在标有“reflection”的功能块中搜索具有这些属性的变量，因此需要的时间较少。

语法：{attribute 'reflection'}

请参见有关属性“instance-path”的描述，以查看示例。

另请参见：

- 属性“instance-path” [▶ 747]

16.8.2.42 属性“strict”

在下列情况下，属性“strict”会导致编译错误：

- 使用枚举类型的变量进行算术运算
- 将非枚举值的常量值分配给枚举类型的变量
- 将数据类型（而非枚举类型）的非常量值分配给枚举类型的变量

语法： {attribute 'strict'}

插入位置： 在 TYPE 定义上方一行

16.8.2.43 属性“subsequent”

该编译指示用于在内存位置上直接依次分配变量。如果列表发生变化，则会将整个变量列表分配到 1 个新的内存位置。该编译指示可用于程序和全局变量列表。

语法： {attribute 'subsequent'}

具有属性“subsequent”的程序中的 VAR_TEMP 会导致编译器错误。

如果 1 个变量在“RETAIN”列表中，则整个列表将被存储为“RETAIN”。

16.8.2.44 属性“Tc2GvlVarNames”

该编译指示的作用是，在 GVL 中声明的符号可以像在 TwinCAT 2 中一样通过 ADS 进行寻址（无需将 GVL 名称用作命名空间）。

语法： {attribute ‘Tc2GvlVarNames ‘}

示例：

```
{attribute 'Tc2GvlVarNames'}
VAR_GLOBAL
    nVar  AT %Q* : UINT;
END_VAR
```

16.8.2.45 属性“TcCallAfterOutputUpdate”

该编译指示定义了在输出更新后是否要执行程序。该属性取代了 TwinCAT 2 的 **IO at Task begin**（在任务开始时的 IO）选项的功能。

语法： {attribute 'TcCallAfterOutputUpdate' }

插入位置： 该属性必须添加到在输出更新后调用的所有程序 POU 中。

示例：

```
{attribute 'TcCallAfterOutputUpdate'}
PROGRAM MAIN
VAR
END_VAR
```

16.8.2.46 属性“TcContextId”

通过该编译指示，您可以定义由哪个任务更新已分配的变量。

语法： {attribute 'TcContextId' := '<TaskId>'}

引号中的占位符 <TaskId> 必须替换为任务 ID。在 **Result**（结果）表格中可以读取任务 ID。该表格在 **Context**（上下文）选项卡中显示，双击 PLC 过程映像（**<Project name> 实例**）即可打开。

插入位置：

- 在 GVL 中第 1 个 VAR_GLOBAL 上方一行
- 在 POU 的声明上方一行
- 在已分配变量的声明行上方一行

示例 1：

假设：任务 PlcTaskA 的 ID 为 0，任务 PlcTaskB 的 ID 为 1。

在相关变量声明上方一行中插入该编译指示。因此，它仅会影响后面的声明行。
变量 bVar1 由任务 PlcTaskA 更新，变量 bVar2 由任务 PlcTaskB 更新。

```
VAR_GLOBAL
    {attribute 'TcContextId':='0'}
    bVar1  AT%Q* : BOOL;
    {attribute 'TcContextId':='1'}
    bVar2  AT%Q* : BOOL;
END_VAR
```

示例 2：

假设：任务 PlcTaskA 的 ID 为 0，任务 PlcTaskB 的 ID 为 1。

在 GVL 中，将该编译指示插入 VAR_GLOBAL 上方一行中和变量声明上方一行中。
因为在它的声明行上方应用属性，所以变量 bVar3 由任务 PlcTaskB 更新。由于在第 1 个 VAR_GLOBAL 上方使用，因此所有其他变量（直到 bVar3）均由任务 PlcTaskA 更新。

```
{attribute 'TcContextId':='0'}
VAR_GLOBAL
    bVar1  AT%Q* : BOOL;     // => PlcTaskA
    bVar2  AT%Q* : BOOL;     // => PlcTaskA

    {attribute 'TcContextId':='1'}
    bVar3   AT%Q* : BOOL;    // => PlcTaskB

    bVar4  AT%Q* : BOOL;     // => PlcTaskA
END_VAR
```

示例 3：

假设：任务 PlcTaskA 的 ID 为 0，任务 PlcTaskB 的 ID 为 1。

在 GVL 中，将该编译指示插入第 1 个 VAR_GLOBAL 上方一行中和第 2 个 VAR_GLOBAL 上方一行中。
请注意，这种用法的目的不是让任务 PlcTaskB 更新变量 bVar4-bVar6。

背景：在 GVL 中，在 VAR_GLOBAL 上方插入该编译指示的情况仅在第 1 个 VAR_GLOBAL 上方有效。

所示的示例会导致以下分配：变量 bVar4 将第 2 个 VAR_GLOBAL 上方的编译指示解释为声明行上方的编译指示，因此变量 bVar4 由任务 PlcTaskB 更新。由于在第 1 个 VAR_GLOBAL 上方使用，因此所有其他变量（直到 bVar4）均由任务 PlcTaskA 更新。

为了将属性应用于整组变量（例如，bVar4-bVar6），建议为每组变量使用专用的 GVL。

```
{attribute 'TcContextId':='0'}
VAR_GLOBAL
bVar1 AT%Q* : BOOL; // => PlcTaskA
bVar2 AT%Q* : BOOL; // => PlcTaskA
bVar3 AT%Q* : BOOL; // => PlcTaskA
END_VAR
{attribute 'TcContextId':='1'}
VAR_GLOBAL
bVar4 AT%Q* : BOOL; // => PlcTaskB
bVar5 AT%Q* : BOOL; // => PlcTaskA
bVar6 AT%Q* : BOOL; // => PlcTaskA
END_VAR
```

16.8.2.47 属性“TcContextName”

通过该编译指示，您可以定义由哪个任务更新已分配的变量。

语法： {attribute 'TcContextName' := '<TaskName>'}

引号中的占位符 <TaskId> 必须替换为任务名称。

插入位置：

- 在 GVL 中第 1 个 VAR_GLOBAL 上方一行
- 在 POU 的声明上方一行
- 在已分配变量的声明行上方一行

示例 1：

在相关变量声明上方一行中插入该编译指示。因此，它仅会影响后面的声明行。
变量 bVar1 由任务 PlcTaskA 更新，变量 bVar2 由任务 PlcTaskB 更新。

```
VAR_GLOBAL
    {attribute 'TcContextName':='PlcTaskA'}
    bVar1  AT%Q* : BOOL;
    {attribute 'TcContextName':='PlcTaskB'}
    bVar2  AT%Q* : BOOL;
END_VAR
```

示例 2：

在 GVL 中，将该编译指示插入 VAR_GLOBAL 上方一行中和变量声明上方一行中。
因为在它的声明行上方应用属性，所以变量 bVar3 由任务 PlcTaskB 更新。由于在第 1 个 VAR_GLOBAL 上方使用，因此所有其他变量（直到 bVar3）均由任务 PlcTaskA 更新。

```
{attribute 'TcContextName':='PlcTaskA'}
VAR_GLOBAL
    bVar1  AT%Q* : BOOL;     // => PlcTaskA
    bVar2  AT%Q* : BOOL;     // => PlcTaskA

    {attribute 'TcContextName':='PlcTaskB'}
    bVar3  AT%Q* : BOOL;     // => PlcTaskB

    bVar4  AT%Q* : BOOL;     // => PlcTaskA
END_VAR
```

示例 3：

在 GVL 中，将该编译指示插入第 1 个 VAR_GLOBAL 上方一行中和第 2 个 VAR_GLOBAL 上方一行中。
请注意，这种用法的目的不是让任务 PlcTaskB 更新变量 bVar4-bVar6。

背景：在 GVL 中，在 VAR_GLOBAL 上方插入该编译指示的情况仅在第 1 个 VAR_GLOBAL 上方有效。

所示的示例会导致以下分配：变量 bVar4 将第 2 个 VAR_GLOBAL 上方的编译指示解释为声明行上方的编译指示，因此变量 bVar4 由任务 PlcTaskB 更新。由于在第 1 个 VAR_GLOBAL 上方使用，因此所有其他变量（直到 bVar4）均由任务 PlcTaskA 更新。

为了将属性应用于整组变量（例如，bVar4-bVar6），建议为每组变量使用专用的 GVL。

```
{attribute 'TcContextName':= 'PlcTaskA'}
VAR_GLOBAL
bVar1 AT%Q* : BOOL; // => PlcTaskA
bVar2 AT%Q* : BOOL; // => PlcTaskA
bVar3 AT%Q* : BOOL; // => PlcTaskA
END_VAR
{attribute 'TcContextName':='PlcTaskB'}
VAR_GLOBAL
bVar4 AT%Q* : BOOL; // => PlcTaskB
bVar5 AT%Q* : BOOL; // => PlcTaskA
bVar6 AT%Q* : BOOL; // => PlcTaskA
END_VAR
```

16.8.2.48 属性“TcDisplayScale”

该编译指示可用于缩放变量的显示值。

语法：{attribute ‘TcDisplayScale‘ := ’<Value>’}

以下值对 <Value> 有效：

- “+/-10”
- “0-10”

- “0-20”
- “4-20”
- “0-10(16)”
- “0-20(16)”
- “4-20(16)”
- “0.1° ”
- “0.01° ”
- “0-5”
- “0-30”
- “0-50”
- “+/-5”
- “+/-2.5”
- “+/-100”
- “0-5(16)”
- “0-30(16)”
- “0-50(16)”
- “+/-75mV”

插入位置：在变量的声明行上方一行

示例：

```
PROGRAM MAIN
VAR
    {attribute 'TcDisplayScale' := '0-10'}
    nVar  AT %Q* : UINT;
END_VAR
```

16.8.2.49 属性“TcEncoding”

通过该编译指示，您可以定义如何解释 STRING 类型的变量。

语法：{attribute 'TcEncoding' := 'UTF-8'}

插入位置：在 STRING 变量的声明行的上方一行

以该方式声明的 STRING 类型的变量使用 UTF-8 格式进行 Unicode 字符集的字符格式化。与每个 STRING 变量一样，这里也使用 0 终止符。

因此，为了支持特殊字符和不同语言的文本，字符集并不局限于数据类型为 STRING（或 WSTRING）的典型字符集。相反，UTF-8 格式的 Unicode 字符集与数据类型 STRING 结合使用。该格式通常用于 IT 领域中。

如果使用 ASCII 字符集，则 STRING 的典型格式与 STRING 的 UTF-8 格式之间没有区别。

对于 STRING 变量，单个字符总是存储在 1 个字节内，而对于 WSTRING 变量，单个字符总是存储在 2 个字节内。然而，对于 UTF-8 格式，单个字符存储在 1 至 4 个字节内，具体取决于所涉及的字符。因此，字符数往往与变量的字节大小不对应。

示例 1：

在相关变量声明上方一行中插入该编译指示。因此，它仅会影响后面的声明行。

```
VAR
    sMyStringText   : STRING;
    wsMyWStringText : WSTRING;
    {attribute 'TcEncoding':='UTF-8'}
    sMyUTF8Text     : STRING;
END_VAR
```

示例 2：

如果文本包含未在 ASCII 字符集中进行定义的字符，则字面量的分配需要 1 个帮助函数。

```
VAR
    sMyStringText   : STRING  := 'The dinner costs 30 €.';
    wsMyWStringText : WSTRING := "The dinner costs 30 €.";

    {attribute 'TcEncoding':='UTF-8'}
    sMyUTF8Text1 : STRING  := sLiteral_TO_UTF8('The dinner costs 30 €.');
    {attribute 'TcEncoding':='UTF-8'}
    sMyUTF8Text2 : STRING  := wsLiteral_TO_UTF8("The dinner costs 30 €.");
    {attribute 'TcEncoding':='UTF-8'}
    sMyUTF8Text3 : STRING  := 'Hello World.';

    // verfügbar ab TC3.1 Build 4026
    {attribute 'TcEncoding':='UTF-8'}
    sMyUTF8Text4 : STRING  := UTF8#'The dinner costs 30 €.';
END_VAR
```

在 TwinCAT 3 PLC Lib TC2 Utilities 文档中介绍了 STRING 变量的更多帮助函数以及 UTF-8 的转换可能性。

16.8.2.50 属性“TcHideSubItems”

该编译指示可用于定义要隐藏变量的子元素。然后，它们在解决方案资源管理器的过程映像中不再可见。

语法：{attribute 'TcHideSubItems'}

插入位置：在子元素的声明上方一行

示例：

```
TYPE DUT :
STRUCT
a : INT;
{attribute 'TcHideSubItems'}
b : ARRAY [0..20000] OF BYTE;
END_STRUCT
END_TYPE
```

在解决方案资源管理器中，属性“TcHideSubItems”只能创建变量“b”，而不再创建 b[0], b[1], b[2]... b[19999]。

16.8.2.51 属性“TcIgnorePersistent”

该编译指示可用于防止恢复变量的持久存储值。

语法：{attribute ‘TcIgnorePersistent ‘}

插入位置：在变量声明上方一行

16.8.2.52 属性“TcInitOnReset”

您可以使用该编译指示来定义在 **Reset cold**（冷重置）时是否要重新初始化持久变量。

语法：{attribute 'TcInitOnReset'}

插入位置：在受影响变量的声明行上方一行

TC3.1 Build 4024 及以上可用

示例：

```
VAR PERSISTENT
    nVar1 : UINT;
    {attribute 'TcInitOnReset'}
    nVar2 : UINT;
END_VAR
```

在 **Reset cold**（冷重置）的情况下，只有 nVar2 会被重新初始化。

16.8.2.53 属性“TcInitSymbol”

使用该编译指示，您可以定义变量是否用作 Init 符号。在构建过程之后，通过该属性选择的变量在 PLC 实例的“Symbol Initialization”（符号初始化）选项卡中可用。在“Value”（值）列中，您可以输入所需的值，这些值将在开始代码执行之前被分配给变量。这些值将在开始代码执行之前被复制到变量值中，并覆盖在变量声明期间可能已经指定的初始值（例如，nVar : INT := 123;）。

对于 TC3.1 Build 4024.4 及以上版本，在声明期间指定的初始值将作为初始值被纳入到表格中。有关此方面的更多信息，请参见下文。

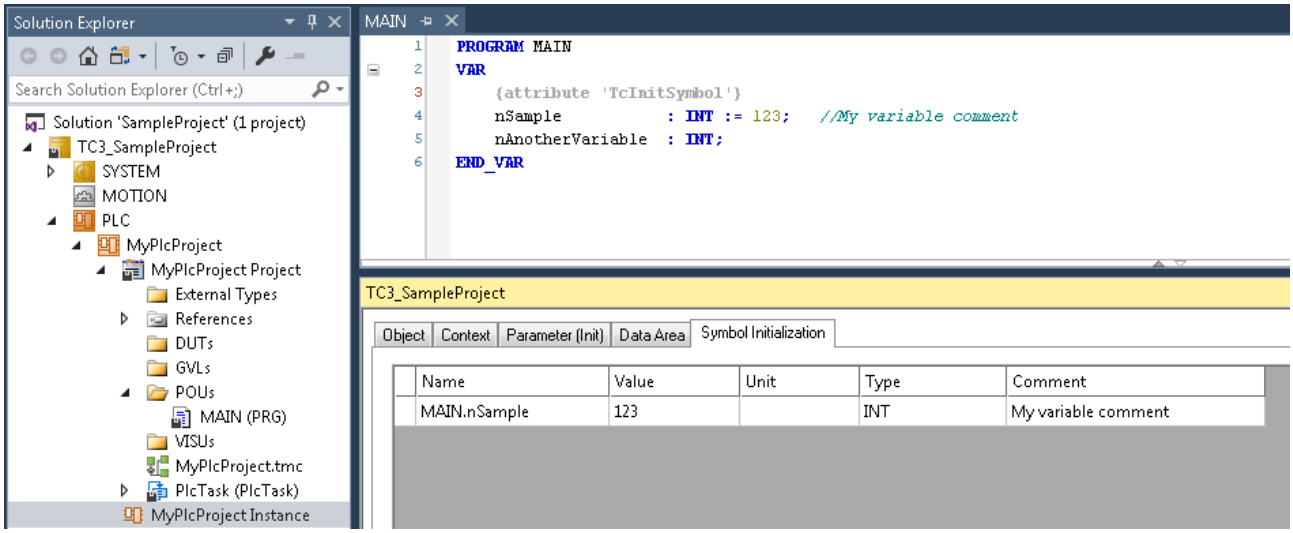
语法： {attribute 'TcInitSymbol'}

插入位置： 在变量的声明行上方一行

示例：

```
PROGRAM MAIN
VAR
    {attribute 'TcInitSymbol'}
    nSample           : INT := 123;   //My variable comment
    nAnotherVariable  : INT;
END_VAR
```

该声明会导致在 PLC 实例的 **Symbol initialization**（符号初始化）选项卡上显示如下内容：



列填充：

以下各列中的条目取自 PLC 编辑器中的变量声明（另请参见上方示例）：

- 名称（= 声明路径 + 符号名称）
- 值：只有在最初向表格中添加变量时才会采用变量声明的初始值；有关更多信息，请参见下文。
- 类型
- 注释

表格“Value”（值）列中的初始条目：

情景：使用属性“TcInitSymbol”声明 1 个变量。在首次编译该声明时，变量会被添加到符号初始化表格中。

对于 TC3.1 Build 4024.4 及以上版本，在将变量添加到表格中的过程中，在表格“Value”（值）列中会采用变量声明的初始值（在上述示例中为 123）。对于先前的 TC3.1 版本，使用的是零初始化，因此在“Value”（值）列中填充的值为 0。

在将变量添加到符号初始化表格中之后，您可以通过更改初始自动条目的方式在表格中输入所需的 Init 值。因此，上方示例中的变量 nVar 最初被添加到表格中的值为 123。如果您随后将表格值更改为 456，则在开始代码执行之前会将值 456 分配给变量 nVar。因此，变量声明的初始值（123）会被表格中的条目（456）所覆盖。

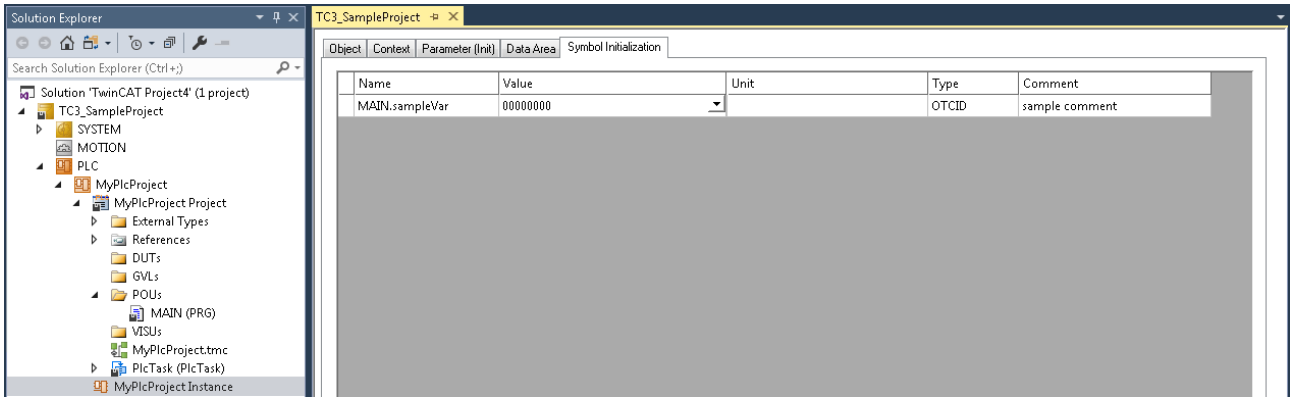
填充表格“Unit”（单元）列：

“Unit”（单元）列用于 OTCID 数据类型，系统会为这些数据类型自动填充 TcCOM 对象的名称，该名称属于“Value”（值）列中的 OTCID 值。

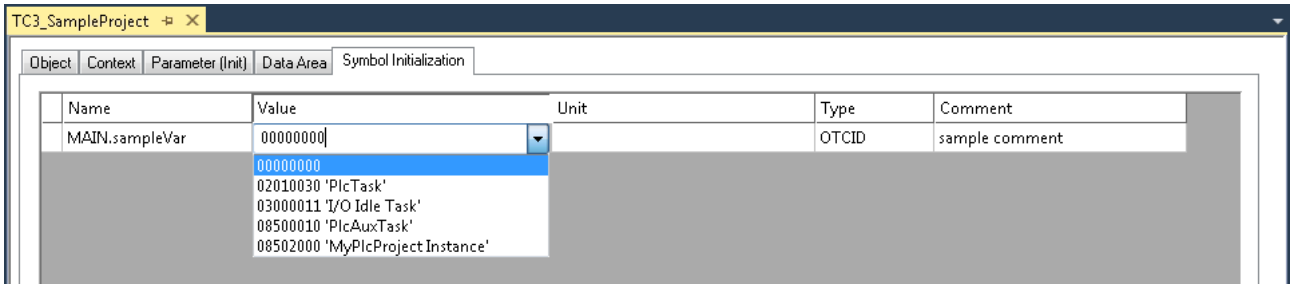
示例：

```
PROGRAM MAIN
VAR
    {attribute 'TcInitSymbol'}
    sampleVar         : OTCID;        //sample comment
END_VAR
```

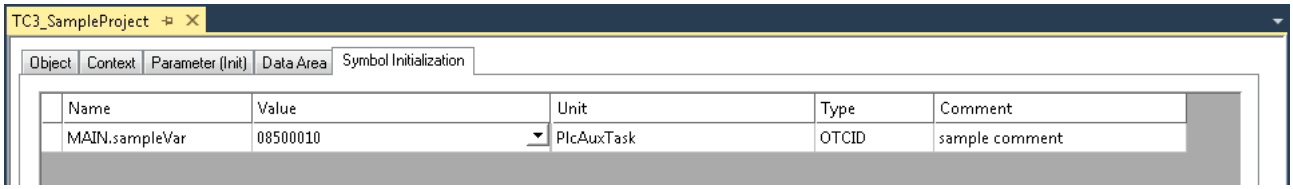
该声明会导致在 PLC 实例的 **Symbol initialization**（符号初始化）选项卡上显示如下内容：



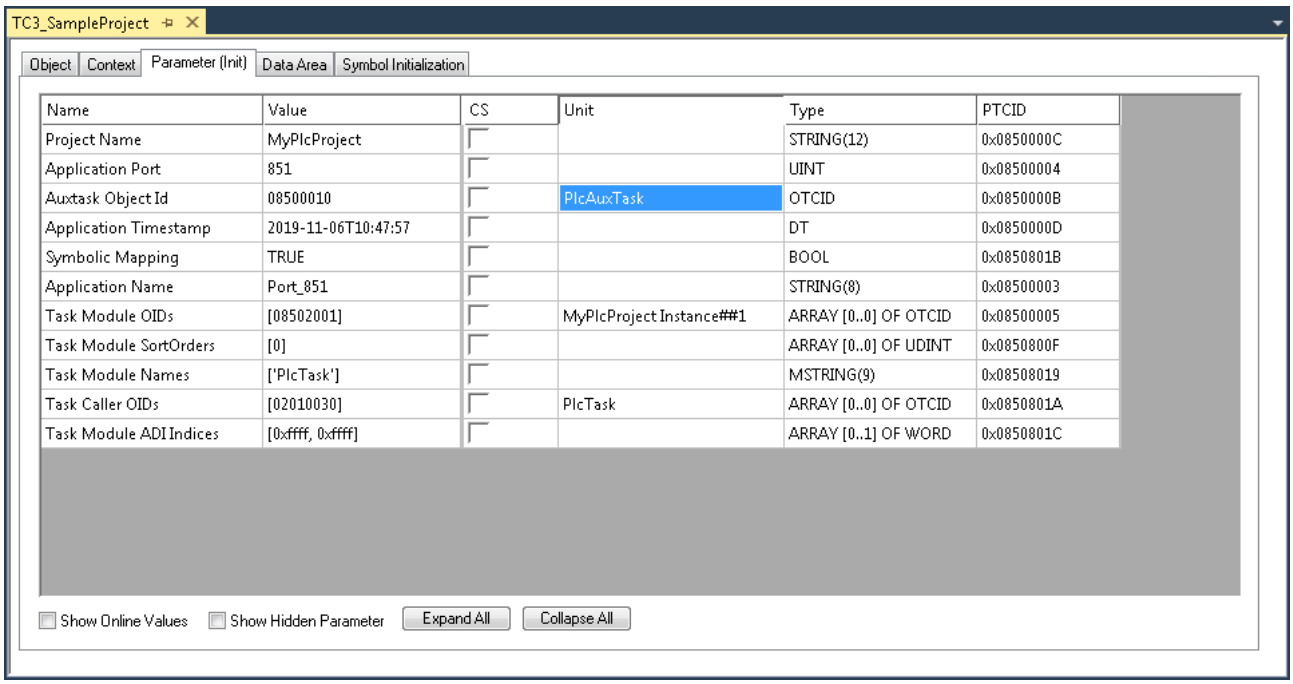
为了给变量赋值，您可以打开 1 个选择菜单，从可能的值中进行选择。



例如，如果您在这种情况下选择值对“08500010 ’PlcAuxTask’”，则可在“Value”（值）列中输入 ID（08500010），并在“Unit”（单元）列中输入名称（PlcAuxTask）。



值对“Value”（值）和“Unit”（单元）与 **Parameter (Init)**（参数（Init））选项卡上显示的值对相对应。



16.8.2.54 属性“TcLinkTo”/“TcLinkTo0S0”

这 2 个编译指示可用于将变量直接分配给另一个过程映像的输入和输出。

语法：

{attribute 'TcLinkTo' := '<I/O point name>'}

引号中的占位符 <I/O point name> 必须替换为输入或输出的名称。

{attribute 'TcLinkTo0S0' := '<x,y,z>'I/O point name'}

x：PLC 变量的位偏移

y：要连接的位数

z：目标变量的偏移，这些位将被映射到目标变量上

引号中的占位符 <I/O point name> 必须替换为输入或输出的名称。

链接功能块的已分配变量：

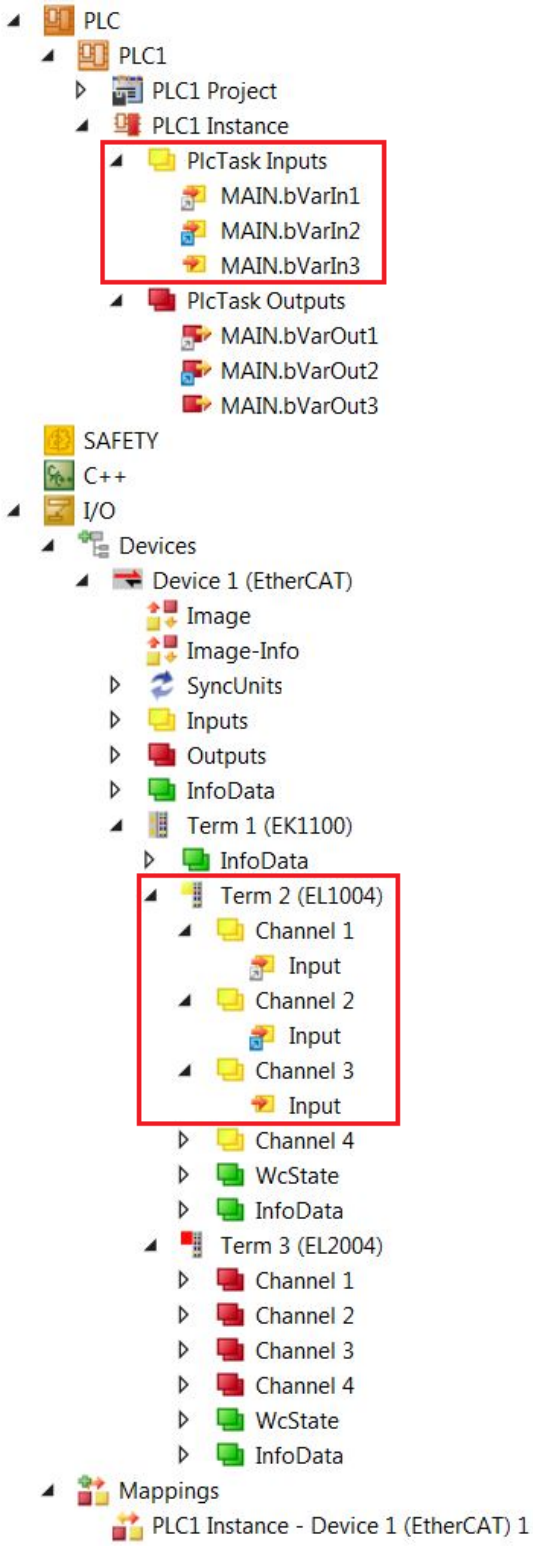
借助编译指示也可以链接功能块的已分配变量。对于功能块来说，这不是在声明已分配变量时进行的，而是在功能块实例的声明点进行的。有关此方面的更多信息，请参见下面的“使用属性的示例”部分。

链接方法的表示法：

如果对变量与输入或输出进行手动链接，则过程映像中的相关变量将会显示 1 个白色图标，链接通道也是如此。另一方面，如果通过属性“TcLinkTo”/“TcLinkTo0S0”链接变量，则相关图标为蓝色。未链接的变量没有图标。

```
PROGRAM MAIN
VAR
    bVarIn1    AT%I*  : BOOL;  // linked manually

    {attribute 'TcLinkTo' := 'TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 2 (EL1004)^Channel
2^Input'}
    bVarIn2    AT%I*  : BOOL;  // linked via attribute

    bVarIn3    AT%I*  : BOOL;  // not linked

    bVarOut1   AT%Q*  : BOOL;  // linked manually

    {attribute 'TcLinkTo' := 'TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 3 (EL2004)^Channel
2^Output'}
    bVarOut2   AT%Q*  : BOOL;  // linked via attribute
```

```
    bVarOut3   AT%Q*   : BOOL;  // not linked
END_VAR
```



使用属性的示例：

```
PROGRAM MAIN
VAR
    nVar1          : INT;

     {attribute 'TcLinkTo' := 'TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 5 (EL4132)^Channel1^Out
put'}
    // Link when using the full path of an input or output
    nVar2   AT%Q*   : INT;
```

```
    {attribute 'TcLinkTo' := '[1] := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 4 (EL2008)^Channe
l 4^Output;
                              [2] := TIID^Device 1
(EtherCAT)^Term 1 (EK1100)^Term 4 (EL2008)^Channel 5^Output'}
    aVar3   AT%Q*  : ARRAY [1..2] OF BOOL;

    {attribute 'TcLinkTo' := '.bIn  := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 3 (EL1008)^Chan
nel 2^Input;
                              .bOut := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 2 (EL2008)^Chan
nel 2^Output'}
    // Link when using function blocks (FB_Module with allocated input bIn and allocated output bOut
)
    fbVar4         : FB_Module;

    {attribute 'TcLinkTo' := '[1].bIn  := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 2 (EL1008)^C
hannel 4^Input;
                              [1].bOut := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 3 (EL2008)^C
hannel 4^Output;
                              [2].bIn  := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 2 (EL1008)^C
hannel 5^Input;
                              [2].bOut := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 3 (EL2008)^C
hannel 5^Output'}
    aModule         : ARRAY[1..2] OF FB_Module;

    {attribute 'TcLinkTo' := 'TIIB(2)^Channel 5^Output'}
    // Linking with the keyboard shortcuts
    bVar5   AT%Q*  : BOOL;

    {attribute 'TcLinkTo' := 'TIIB[TerminalXY]^Channel 5^Output'}
    // TerminalXY is the name of the terminal shown in the IO tree
    nVar6   AT%Q*  : INT;

    {attribute 'TcLinkToOSO' := '<1,2,3>TIIB(5)^Channel 2^Output'}
    // Link when using Offset/Size/Offset
    nVar7   AT%Q*  : BYTE;

    {attribute 'TcLinkToOSO' := '[1] := <1,2,3>TIIB(5)^Channel 2^Output;
                                 [2] := <4,5,6>TIIB(5)^Channel 3^Output'}
    aVar8   AT%Q*  : ARRAY [1..2] OF WORD;

    {attribute 'TcLinkTo' := '[0]
[0] := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 2 (EL2008)^InfoData^State;
                              [0]
[1] := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 3 (EL2008)^InfoData^State;
                              [0]
[2] := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 4 (EL2008)^InfoData^State;
                              [1]
[0] := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 5 (EL2008)^InfoData^State;
                              [1]
[1] := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 6 (EL2008)^InfoData^State;
                              [1]
[2] := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 7 (EL2008)^InfoData^State'}
    aVar9   AT%Q*  : ARRAY[0..1, 0..2] OF UINT;


    (*************************************************************************************)
      Available from TC3.1 Build 4026
    (*************************************************************************************)

    // ARRAY info on the left is "%d..%d" and on the right side "%d..#"
    // Link info will be generate from 1 TO 4 (4 times) and from 3 TO 6 on channel side
    {attribute 'TcLinkTo' := '[1..4] := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 2 (EL2008)^Cha
nnel 3..#^Output'}
    bVar10 AT %Q* : ARRAY[1..4] OF BOOL;

    {attribute 'TcLinkTo' := '[1..4] := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 4 (EL4014)^AO
Outputs Channel 1..#^Analog output'}
    bVar11 AT %Q* : ARRAY[1..4] OF INT;

    // Also additional syntax for EtherCAT PLC structures to PLC mappings -
 see "PLC" tab on EtherCAT terminals:
    // ARRAY info on the left is "%d..%d" and on the right side "%d..#" -
 channel index is added via "^^%d" and the value is 0-based
    {attribute 'TcLinkTo' := '[1..4] := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 5 (EL2819)^^ 0
..#'}
    bVar12 : ARRAY [1..16] OF MDP5001_280_700116F7;

END_VAR
```

所有可用快捷方式表格：

快捷方式	至
TIIC	节点 I/O 配置
TIID	节点 I/O 配置 ^I/O 设备或 节点 I/O 配置 TAB“I/O 设备”
TIRC	节点实时配置
TIRT	节点实时配置^附加任务或 节点“实时配置”TAB“附加任务”
TIRS	节点实时配置^实时设置或 节点“实时配置”TAB“实时设置”
TIPC	节点 PLC 配置
TINC	节点 NC 配置
TICC	节点 CNC 配置
TIAC	节点 CAM 配置
TING	节点轴
TINT	节点表格
TINS	节点 SAF 任务
TIIT(%d)	端子模块 %d（仅适用于 KL 总线）
TIIB(%d)	方框 %d
TIIF(%d)	设备 %d
TIIT[%s]	带有名称 %s 的端子模块（仅适用于 KL 总线）
TIIB[%s]	带有名称 %s 的方框
TIIF[%s]	带有名称 %s 的设备

16.8.2.55 属性“TcNcAxis”

该编译指示可用于将源代码外的 NC 轴与 Axis_Ref 类型的变量关联起来。

语法： {attribute 'TcNcAxis' := '<AxisName>'}

如 NC 部分所述，引号中的占位符 <AxisName> 必须替换为轴的名称。

插入位置： 在变量的声明行上方一行

示例：

```
PROGRAM MAIN
VAR
    {attribute 'TcNcAxis' := 'Axis1'}
    // Using the axis in the same POU of type Program.
    axis1     : AXIS_REF;

    {attribute 'TcNcAxis' := '.axis1:=Axis2;.axis2:=Axis3'}
    // Use of the axis in POU fbSample of type Function block.
    // The internal axis names (axis1 and axis2) of the POU fbSample must be assigned to the NC axes
 used by the function block instance (axis 2 and axis 3).
    fbSample  : FB_Sample;

    {attribute 'TcNcAxis' := '[0]:=Axis4;[1]:=Axis5;[2]:=Axis6'}
    // Using an axis in an array.
    aAxis     : ARRAY [0..2] OF AXIS_REF;
END_VAR
```

16.8.2.56 属性“TcNoSymbol”/“tc_no_symbol”

该编译指示可用于指定不为变量生成（ADS）符号。换句话说，符号访问会被阻止。如果将值“未使用”分配给该编译指示，则该编译指示仅对未使用的变量起作用。

请注意，使用属性”TcNoSymbol“/” tc_no_symbol“声明的变量无法持久存储。此外，对于未生成（ADS）符号的变量，还可防止生成相关的过程映像（已分配的输入/输出）。

语法： {attribute 'tc_no_symbol'} 或 {attribute 'TcNoSymbol'}
（符号”TcNoSymbol“仅适用于 TwinCAT 版本 3.1.4022 或以上版本）。

插入位置：在变量的声明行上方一行

示例：

```
VAR_GLOBAL
    {attribute 'tc_no_symbol'}
    nVar1   : INT;
    {attribute 'tc_no_symbol'}
    var2    : OTCID;
END_VAR
```

或者，对于 TwinCAT 版本 3.1.4022 以来的新符号：

```
VAR_GLOBAL
    {attribute 'TcNoSymbol'}
    nVar1   : INT;
    {attribute 'TcNoSymbol':= 'unused'}
    var2    : OTCID;
END_VAR
```

16.8.2.57 属性”TcRpcEnable“

该编译指示可用于激活远程过程调用（RPC）的方法。这样可以通过 ADS 调用该方法。TwinCAT OPC UA 服务器需要该编译指示才能将该方法作为 OPC UA 方法使用。

语法：{attribute 'TcRpcEnable'}

插入位置：在方法的声明部分上方第 1 行。

示例：

```
{attribute 'TcRpcEnable'}
METHOD M_Sum : INT
VAR_INPUT
    nInput1 : INT;
    nInput2 : INT;
END_VAR
```

更多示例：

- ADS：
  - 链接：TcAdsClient.InvokeRpcMethod Method (ITcAdsSymbol, Int32, .Object.)
  - 文档路径：TwinCAT 3 \ TE1000 XAE \ Technologies \ ADS \ TwinCAT ADS .NET \ TwinCAT.Ads Namespaces \ TwinCAT.Ads Namespace \ TcAdsClient Class \ TcAdsClient Methods \ TcAdsClient.InvokeRpcMethod Method \ TcAdsClient.InvokeRpcMethod Method (ITcAdsSymbol, Int32, .Object.)
- OPC UA：
  - 链接：
  - 文档路径：TwinCAT 3 \ TFxxxx | TC3 Functions \ TF6xxx - Connectivity \ TF6100 TC3 OPC UA \ Technical introduction \ Server \ PLC \ Method call

16.8.2.58 属性“TcRetain”

该编译指示会导致将下列变量创建为保留变量，并在失控终止（断电）后保留其值。因此，该编译指示是使用关键字 VAR RETAIN [▶ 631] 的替代方法。

所谓的保留处理程序可确保在 PLC 周期结束时写入保留变量，并且仅在 NovRam 的相应区域中写入。在 C/C++ 文档的“保留数据”章节中介绍了保留处理程序的处理方法。

如果要将自行定义的数据单元类型（DUT）用作保留变量，则数据类型必须在 TwinCAT 类型系统中可用。您可以使用选项 **Convert to Global Type**（转换为全局类型），也可以直接创建 STRUCT RETAIN 结构。但是，保留处理程序会处理该结构的所有出现情况。

保留数据不能用于整个 POU（功能块）。不过，可以使用 POU 的各个元素。

语法：{attribute 'TcRetain'}

插入位置：在变量的声明行上方一行

示例：

```
PROGRAM MAIN
VAR
    {attribute 'TcRetain'}
    nVar1   : INT;
END_VAR
```

16.8.2.59 属性“TcSwapDWord”

该编译指示可用于定义该已分配输出变量的字序应该改变（交换低位字和高位字）。如果存在该编译指示，则可为 PLC 过程映像内的输出勾选“Swap LOWORD and HIWORD”（交换 LOWORD 和 HIWORD）复选框。

语法：{attribute 'TcSwapDWord'}

插入位置：已分配输出变量的声明行上方一行（AT%Q）

示例：

```
VAR
    {attribute 'TcSwapWord'}
    nVar1  AT%Q* : WORD;
    {attribute 'TcSwapDWord'}
    nVar2  AT%Q* : DWORD;
END_VAR
```

16.8.2.60 属性“TcSwapWord”

该编译指示可用于定义该已分配输出变量的字节序应该改变（交换低字节和高字节）。如果存在该编译指示，则可为 PLC 过程映像内的输出勾选“Swap LOBYTE and HIBYTE”（交换 LOBYTE 和 HIBYTE）复选框。

语法：{attribute 'TcSwapWord'}

插入位置：已分配输出变量的声明行上方一行（AT%Q）

示例：

```
VAR
    {attribute 'TcSwapWord'}
    nVar1  AT%Q* : WORD;
    {attribute 'TcSwapDWord'}
    nVar2  AT%Q* : DWORD;
END_VAR
```

16.8.2.61 属性“to_string”

该编译指示会影响使用运算符 TO_STRING/TO_WSTRING 的枚举组件转换结果的输出方式：如果枚举声明中包含该编译指示，则枚举组件的名称将显示为字符串而不是数值。

语法：{attribute 'to_string'}

插入位置：枚举的声明上方一行。

TC3.1 Build 4024 及以上可用

示例：

枚举 E_Sample

```
{attribute 'qualified_only'}
{attribute 'strict'}
{attribute 'to_string'}
TYPE E_Sample :
(
    eInit := 0,
    eStart,
    eStop
);
END_TYPE
```

MAIN 程序

```
PROGRAM MAIN
VAR
    eSample         : E_Sample;
    nCurrentValue   : INT;
    sCurrentValue   : STRING;
    wsCurrentValue  : WSTRING;

    sComponent      : STRING;
    wsComponent     : WSTRING;
END_VAR
```

```
nCurrentValue  := eSample;
sCurrentValue  := TO_STRING(eSample);
wsCurrentValue := TO_WSTRING(eSample);

sComponent     := TO_STRING(E_Sample.eStart);
wsComponent    := TO_WSTRING(E_Sample.eStop);
```

赋值/转换函数的结果：

- nCurrentValue 的值：0
- sCurrentValue 的值：'eInit'
- wsCurrentValue 的值："eInit"
- sComponent 的值：'eStart'
- wsComponent 的值："eStop"

如果没有使用属性“to_string”声明枚举，则会出现以下结果：

- nCurrentValue 的值：0
- sCurrentValue 的值：'0'
- wsCurrentValue 的值："0"
- sComponent 的值：'1'
- wsComponent 的值："2"

另请参见：

- 枚举 [▶ 721]

16.8.2.62 'TcPersistent' 属性

该编译指示会导致将下列变量创建为持久变量，该变量可在意外终止（断电）后保留其值。因此，该编译指示是使用关键字 VAR PERSISTENT [▶ 631] 的替代方法。

语法： {attribute 'TcPersistent'}

插入位置： 在变量的声明行上方一行

示例：

```
PROGRAM MAIN
VAR
    {attribute 'TcPersistent'}
    nVar1   : INT;
END_VAR
```

16.8.3 条件编译指示

条件编译指示用于影响预编译过程或编译过程中的代码生成。编程语言 ST 支持这些编译指示。

对于 TC 3.1 build 4024 及以上版本，您可以在声明部分和实现部分使用条件编译指示。声明部分仅支持用于简单编译器定义的运算符“defined (<identifier>)”和“hasvalue (<identifier>, '<value>')”。下面列出的 2 个操作符的特殊用途不在此范围内。此外，不能借助条件编译指示来定义枚举的组件或别名的数据类型。

PLC 项目支持条件编译指示，但库不支持。在先前的 TC 3.1 版本中，在声明部分使用的编译指示没有经过评估。

通过条件编译指示，您可以影响在编译时是考虑声明代码还是实现代码。例如，您可以使其取决于某个编译器定义是否已定义或具有某个值、是否声明某个变量、是否存在某个功能块等。

编译指示	描述
代码中的定义： • {define <identifier>} • {define <identifier> '<value>'} 示例： • {define variantA} • {define variantA '123'} • {define variantA 'true'} **在 PLC 项目属性中的定义：** • <identifier> • <identifier> := '<value>' 示例： • variantA • variantA := '123' • variantA := 'true' • variantA := '123', variantB := '456'	稍后可使用定义的运算符查询定义。 此外，稍后还可使用 hasvalue 运算符查询和比较可选的指定值。
{undefine <identifier>}	<identifier> 的 {define} 指令被取消，然后，标识符再次变为“未定义”。如果指定的标识符当前未定义，则忽略该编译指示。
{IF <expr>}... {ELSIF <expr>}... {ELSE}... END_IF}	这些是条件编译的编译指示。 在编译过程中，指定的表达式 <expr> 必须保持不变；这些表达式将按照它们出现的顺序进行求值，直到其中 1 个表达式显示非零值为止。与指令链接的文本将被编译；其他行将被忽略。各部分的顺序已设定。ELSIF 和 ELSE 部分是可选的。ELSIF 部分可以多次使用。在常量 <expr> 中，您可以使用多个条件编译运算符。
<expr>}	您可以在条件编译的编译指示 {IF} 或 {ELSIF} 中的常量表达式 <expr> 中使用 1 个或多个运算符。

不同的范围

在 PLC 项目属性（编译类别）或上一个节点的系统管理器级别（编译器定义分组，请参见变体管理文档）中，您可以输入全局定义的定义作为编译器定义。此处定义的编译器定义在整个 PLC 项目中均有效。此外，您还可以在 PLC 项目属性中指定多个以逗号分隔的定义的定义。

而且，您可以在 PLC 元素的声明和实现部分定义本地编译器定义。这些内容在相关的声明或实现编辑器中均有效。

另请参见：

- 结构化文本（ST）、扩展结构化文本（ExST） [▶ 113]
- 变体管理

运算符已定义 (<identifier>)

该运算符会导致为表达式分配值 TRUE。前提条件是使用 {define} 指令定义 <identifier>，而随后没有使用 {undefine} 指令取消定义，在这种情况下会返回 FALSE。

示例 1：

要求：有 2 个 PLC 项目，即 Plc1 和 Plc2。变量 pdef1 在 Plc1 中由 {define} 指令定义，但在 Plc2 中没有定义。

```
{IF defined (pdef1)}
    (* This code is processed in Plc1 *)
    {info 'pdef1 defined'}
    hugo := hugo + SINT#1;
{ELSE}
    (* the following code is only processed in Plc2 *)
    {info 'pdef1 not defined'}
    hugo := hugo - SINT#1;
{END_IF}
```

此外，还包括 1 个消息编译指示的示例：在编译 PLC 项目时，在消息窗口中仅会显示已定义 pdef1 的信息，因为实际上已定义 pdef1。如果 pdef1 未定义，则会发出消息“pdef1 未定义”。

示例 2：

在下面的示例中，根据有效的编译器定义，使用不同的值对变量 sVariantUsed 进行初始化。此外，还要声明 1 个已分配的输入变量或 1 个输出变量。在任何情况下都要声明变量 nCounter，即与有效的编译器定义无关。

```
{IF defined (Variant1)}
    sVariantUsed          : STRING := 'Variant1';
    bOutput        AT%Q* : BOOL;
{ELSE}
    sVariantUsed          : STRING := 'NotVariant1';
    bInput         AT%I* : BOOL;
{END_IF}

nCounter                  : INT;
```

实现部分的运算符

下面介绍的运算符只能在实现部分使用，不能在声明部分求值。

运算符已定义 (variable: <variable>)

如果在当前范围内声明变量 <variable>，则该运算符会导致为表达式分配值 TRUE；否则会返回 FALSE。

示例：

要求：有 2 个 PLC 项目，即 Plc1 和 Plc2。在 Plc1 中声明变量 g_bTest，但在 Plc2 中没有声明。

```
{IF defined (variable: g_bTest)}
    (* the following code is only processed in Plc2*)
    g_bTest := x > 300;
{END_IF}
```

运算符已定义 (type: <identifier>)

如果使用标识符 <identifier> 声明数据类型，则该运算符会导致为表达式分配值 TRUE；否则会返回 FALSE。

示例：

要求：有 2 个 PLC 项目，即 Plc1 和 Plc2。在 Plc1 中声明数据类型 DUT，但在 Plc2 中没有声明。

```
{IF defined (type: DUT)}
    (* the following code is only processed in Plc1*)
    bDutDefined := TRUE;
{END_IF}
```

运算符已定义 (pou: <pou name>)

如果定义名称为 <pou name> 的功能块，则该运算符会导致为表达式分配值 TRUE；否则会返回 FALSE。

借助语法“pou: <pou name>.<method name>”，您还可以检查所提到的功能块是否具有所提到名称的方法或动作。

示例：

要求：有 2 个 PLC 项目，即 Plc1 和 Plc2。在 Plc1 中存在 CheckBounds 功能块，但在 Plc2 中不存在。

```
{IF defined (pou: CheckBounds)}
    (* the following code is only processed in Plc1 *)
    arrTest[CheckBounds(0,i,10)] := arrTest[CheckBounds(0,i,10)] + 1;
{ELSE}
    (* the following code is only processed in Plc2 *)
    arrTest[i] := arrTest[i]+1;
{END_IF}
```

运算符已定义 (IsLittleEndian)

如果 CPU 为“BigEndian（Motorola 字节序）”，则该运算符会导致将表达式设置为 FALSE。

运算符已定义 (IsFPUSupported)

如果表达式返回 TRUE，则代码生成器 FPU（浮点单元）会生成带有 REAL 值的计算代码。否则，TwinCAT 会模拟 FPU 运算，但速度会大大降低。

运算符 hasvalue (RegisterSize, '<register size>')

<register size>：CPU 寄存器的大小，以位为单位

如果 CPU 寄存器的大小等于 <register size>，则该运算符会使表达式返回 TRUE。

<register size> 的可能值

- C16x 为 16
- X86 64 位为 64
- X86 为 32

运算符 hasvalue (PackMode, '<pack mode value>')

选中的 PackMode 取决于设备描述，而非编译指示，后者可为单个 DUT 指定。

运算符 hasattribute (pou: <pou name>, '<attribute>')

如果在功能块 pou-name 的声明部分的第 1 行中指定属性 <attribute>，则该运算符会导致为表达式分配值 TRUE；否则会返回 FALSE。

示例：

要求：有 2 个 PLC 项目，即 Plc1 和 Plc2。在 Plc1 和 Plc2 中定义函数 fun1，但在 Plc1 中也为其分配了属性 vision。

在 Plc1 中：

```
{attribute 'vision'}
FUNCTION fun1 : INT
VAR_INPUT
  i : INT;
END_VAR
VAR
END_VAR
```

在 Plc2 中：

```
FUNCTION fun1 : INT
VAR_INPUT
  i : INT;
END_VAR
VAR
END_VAR
```

编译指示指令：

```
{IF hasattribute (pou: fun1, 'vision')}
(* the following code is only processed in Plc1 *)
ergvar := fun1(ivar);
{END_IF}
```

另请参见：

- 用户定义的属性 [▶ 734]

运算符 hasattribute (variable: <variable>, '<attribute>')

如果使用变量声明前一行中的指令 {attribute '<attribute>'} 为变量分配属性“<attribute>”，则该运算符会导致为表达式分配值 TRUE；否则会返回 FALSE。

示例：

要求：有 2 个 PLC 项目，即 Plc1 和 Plc2。在 Plc1 和 Plc2 中使用变量 g_globalInt，但在 Plc1 中也为其分配了属性“DoCount”。

在 Plc1 中声明 g_GlobalInt

```
VAR_GLOBAL
   {attribute 'DoCount'}
  g_globalInt : INT;
  g_multiType : STRING;
END_VAR
```

在 Plc2 中声明 g_GlobalInt

```
VAR_GLOBAL
  g_globalInt : INT;
  g_multiType : STRING;
END_VAR
```

编译指示指令：

```
{IF hasattribute (variable: g_globalInt, 'DoCount')}
    (* the following code is only processed in Plc1 *)
    g_globalInt := g_globalInt + 1;
{END_IF}
```

另请参见：

- 用户定义的属性 [▶ 734]

运算符 hastype (variable: <variable>, <type-spec>)

如果变量 <variable> 的数据类型为 <type-spec>，则该运算符会导致表达式返回值 TRUE；否则会返回 FALSE。

<type-spec> 的可能数据类型：

BOOL | BYTE | DATE | DATE_AND_TIME | DINT | DWORD | INT | LDATE | LDATE_AND_TIME | LINT | LREAL | LTIME | LTIME_OF_DAY | LWORD | REAL | SINT | STRING | TIME | TIME_OF_DAY | ULINT | UDINT | UINT | USINT | WORD | WSTRING

示例：

要求：有 2 个 PLC 项目，即 Plc1 和 Plc2。在 Plc1 中使用数据类型 LREAL 声明变量 g_multitype，而在 Plc2 中使用的数据类型为 STRING。

```
{IF (hastype (variable: g_multitype, LREAL))}
    (*  the following code is only processed in Plc1 *)
    g_multitype := (0.9 + g_multitype) * 1.1;
{ELSIF (hastype (variable: g_multitype, STRING))}
    (* the following code is only processed in Plc2 *)
    g_multitype := 'this is a multitalent';
{END_IF}
```

运算符 hasvalue (<define-ident>, '<char-string>')

如果定义具有标识符 <define-ident> 的变量，且该变量的值为 <char-string>，则该运算符会导致表达式返回值 TRUE；否则会返回 FALSE。

示例：

要求：有 2 个 PLC 项目，即 Plc1 和 Plc2。在 PLC 项目 Plc1 和 Plc2 中使用变量 test；在 Plc1 中，它的值为 1，在 Plc2 中，它的值为 2。

在 PLC 项目属性中通过 test := '1' 或者在 POU 的实现部分通过 {define test '1'} 可以设置编译器定义。

```
{IF hasvalue(test,'1')}
    (* the following code is only processed in Plc1 *)
    x := x + 1;
{ELSIF hasvalue(test,'2')}
    (* the following code is only processed in Plc2 *)
    x := x + 2;
{END_IF}
```

运算符 NOT <operator>

如果 <operator> 的逆运算返回值 TRUE，则会为表达式分配值 TRUE。<operator> 可以是在本章节中介绍的运算符之一。

示例：

要求：有 2 个 PLC 项目，即 Plc1 和 Plc2。PLC_PRG1 存在于 Plc1 和 Plc2 中；POU CheckBounds 仅存在于 Plc1 中。

```
{IF defined (pou: PLC_PRG1) AND NOT (defined (pou: CheckBounds))}
    (* the following code is only processed in Plc2 *)
    bANDNotTest := TRUE;
{END_IF}
```

运算符 <operator> AND <operator>

如果 2 个指定的运算符均返回 TRUE，则会为表达式分配值 TRUE。<operator> 可以是在本章节中介绍的运算符之一。

示例：

要求：有 2 个 PLC 项目，即 Plc1 和 Plc2。PLC_PRG1 存在于 Plc1 和 Plc2 中；POU CheckBounds 仅存在于 Plc1 中。

```
{IF defined (pou: PLC_PRG1) AND (defined (pou: CheckBounds))}
    (* the following code is only processed in Plc1,  *)
    bANDTest := TRUE;
{END_IF}
```

运算符 <operator> OR <operator>

如果 2 个运算符中的 1 个返回 TRUE，则表达式会返回 TRUE。<operator> 可以是在此处介绍的运算符之一。

示例：

要求：有 2 个 PLC 项目，即 Plc1 和 Plc2。PLC_PRG1 存在于 Plc1 和 Plc2 中；POU CheckBounds 仅存在于 Plc1 中。

```
{IF defined (pou: PLC_PRG1) OR (defined (pou: CheckBounds))}
    (* the following code is only processed in Plc1 and in Plc2 *)
    bORTest := TRUE;
{END_IF}
```

运算符 (<operator>)

() 将运算符括起来。

另请参见：

- 使用编译指示 [▶ 126]
- 用户定义的属性 [▶ 734]

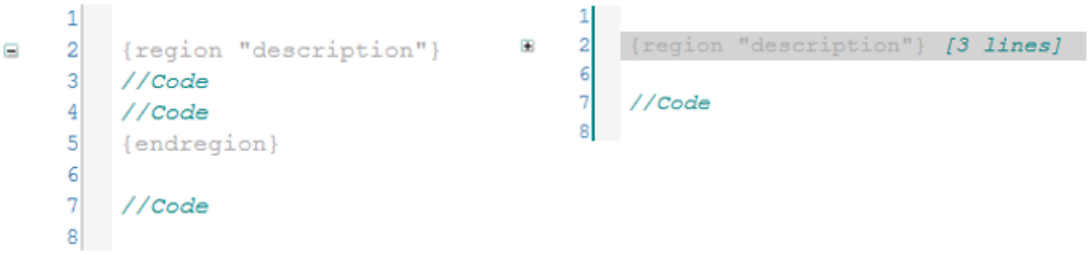
16.8.4 区域编译指示

该编译指示用于将文本编辑器中的多行合并为 1 个块。您可以为块分配 1 个名称。区域编译指示可以嵌套。

语法：{region "description"} … {endregion}

请务必遵循此语法，以便将编译指示考虑在内。

带有区域编译指示的代码：扩展和缩小视图



该编译指示可用于 ST 编辑器和所有声明编辑器。在选项中可以调整语法突出显示。

16.8.5 抑制警告的编译指示

16.8.5.1 编译指示“warning disable”、“warning restore”

编译指示 {warning disable <compiler ID>} 会导致某些警告受到抑制。

编译指示 {warning restore <compiler ID>} 会重新激活受到抑制的消息。

语法：

{warning disable <compiler ID>}

{warning restore <compiler ID>}

<compiler ID>：ID 位于警告消息的开头或 PLC 项目属性中的编译器警告概览中。

C0195 编译器警告的示例：

编译器警告：

```
C0195: Implicit conversion from signed type 'SINT' to unsigned type 'UINT': possible change of sign
```

将该编译指示应用于变量声明：

```
VAR
    {warning disable C0195}
    test1 : UINT := -1;
    {warning restore C0195}
    test2 : UINT := -1;
END_VAR
```

test1 不会生成警告，test2 会生成警告。

C0394 编译器警告的示例：

编译器警告：

```
C0394: Compatibility warning: FB_Exit is now called in F_Sample for FB instance fbSample which is
located on the stack. This has not been the case for code compiled with compiler versions <
3.1.4022.0.
```

将该编译指示应用于 POU 声明：

```
{warning disable C0394}
FUNCTION F_Sample : BOOL
VAR_INPUT
    bInput    : BOOL;
END_VAR
```

```
VAR
    fbSample : FB_Sample;   // The function block FB_Sample contains the method FB_exit
END_VAR
```

如果未使用该编译指示，则会对 fbSample 功能块实例发出上述警告。但是，如果使用了示例中提到的编译指示，就不会生成警告。

实现编辑器中的编译指示

如果您想要在实现编辑器中使用编译指示来抑制警告，目前在 ST 编辑器和 FBD/LD/IL 编辑器中都可以实现。

在 FBD/LD/IL 中，必须在标签 [▶ 603]中输入所需的编译指示。

另请参见：

- TC3 用户界面文档：“Properties”（属性）命令（PLC 项目） > 类别：编译器警告 [▶ 851]

16.8.5.2 属性“suppress_wrn_C0410”

对于带有 ID C0410 的编译器警告，必须使用属性“suppress_wrn_C0410”，而不是编译指示“warning disable”。

语法： {attribute 'suppress_wrn_C0410'}

示例：

属性的编译器警告：

```
C0410: COMPATIBILITY WARNING: A write access to a Property of type REFERENCE calls the SET-Accessor
for versions < 3.1.4022.0 and writes the reference, but calls the GET-Accessor for versions >=
3.1.4022.0 and writes the value! Use the operator REF= if you want to assign the reference.
```

将特性应用于属性：

```
{attribute 'suppress_wrn_C0410'}
PROPERTY refSample : REFERENCE TO BOOL
```

由于该特性的存在，将不会对属性 refSample 生成警告 C0410。

16.9 标识符

另请注意 TwinCAT 3 编程规范（“”部分）

标识符分配的规则

变量标识符规则

- 标识符必须包含以下字母和数字：
  - 0……9
  - A……Z
  - a……z
- TwinCAT 不区分大小写，也就是说：VAR1 和 var1 表示相同的变量。
- 标识符不得以数字开头。
- 标识符不得包含空格、特殊字符（!、&、ß、……）或连字符。
- 不过，允许使用下划线，TwinCAT 也能识别这些下划线。这意味着，例如，A_BCD 和 AB_CD 会被视为 2 个不同的标识符。单词之间通常不使用分隔符，例如，下划线。
- 您应避免使用双下划线（“__”），因为它们用于内部变量。
- 标识符没有长度限制。

标识符（命名空间）的多重使用规则

- 局部标识符只能使用 1 次。
- 标识符不得与关键字（例如，变量类型）相同。
- 全局标识符可重复使用。如果局部变量与全局变量同名，则局部变量在 POU 中具有优先权。
- 在全局变量列表中定义的变量可以与在另一个 GVL 中定义的变量同名。在变量的命名空间/范围方面，TwinCAT 提供以下标准扩展功能：
  - 运算符 . 全局命名空间：以点开头的实例路径总是会打开 1 个全局命名空间。如果存在 1 个与全局变量同名的局部变量（例如，nVar），则会将 .nVar 用于寻址全局变量。
  - 全局变量列表的名称可以明确定义所包含变量的命名空间。这样，您可以在不同的全局变量列表中声明具有相同名称的变量，并且仍然可以通过为列表名称添加前缀的方式对它们进行唯一寻址。
    示例：GVL1.nvar := GVL2.nVar; (*nVar from GVL2 is copied to nVar in GVL1*)
  - 在集成到项目中的库的全局变量列表中定义的变量，可以按照以下语法明确寻址：<namespace library>.<name of the GVL>.<variable name>。
    示例：
    GVL1.nVar := Lib1.GVL1.nVar (* nVar from GVL1 in library Lib1 is copied of nVar in GVL1*)
- 对于库，在通过库管理器插入时可以定义命名空间。这样，您可以通过 <namespace library>.<function block name|variable name> 明确寻址库功能块或库变量。对于嵌套库，请注意必须按顺序指定所有参与库的命名空间。
  示例：如果 Lib1 被 Lib0 引用，则包含在 Lib1 中的函数 F_Sample 将通过 Lib0.Lib1.F_Sample 寻址：nVar := Lib0.Lib1.F_Sample(nValue:=4); (*Return value of F_Sample is copied to variable nVar in the project*)

另请参见：

- 变量 [▶ 623]
- 数据类型 [▶ 697]

16.10 遮蔽规则

在 TwinCAT 中，原则上允许对不同的元素使用相同的标识符。例如，功能块和变量的名称可以相同。不过，为了防止混淆，应避免这样做。

反例：

在下面的代码片段中，局部功能块实例与函数具有相同的名称：

```
FUNCTION Sample : INT
```

```
FUNCTION_BLOCK FB_Sample
```

```
PROGRAM MAIN
VAR
    Sample : FB_Sample;
END_VAR
```

```
Sample();
```

在这种情况下，无法确定在程序中调用的是实例还是函数。

命名规范：

为了确保名称始终唯一，应遵循命名规范，例如，为变量指定特定的前缀。有关此类前缀的可能定义，请参见的 部分。

使用 TE1200 | PLC 静态分析可以自动检查命名规范。通过检查规则 SA0027，静态代码分析还可以检测名称 Sample 的重复使用情况，并将其报告为错误。

限定访问：

通过始终如一地在枚举和全局变量列表中使用“qualified_only” [▶ 761]属性以及使用合格库，可以避免出现模棱两可的情况。

遮蔽：

当相同的标识符用于不同的元素时，编译器基本上不会报告错误或警告。相反，编译器会按照特定的顺序在代码中搜索标识符的声明。如果已经找到 1 个声明，则编译器不会在其他地方进一步搜索可能的声明。如果还存在其他声明，则这些声明对编译器来说是“隐藏的”。下面介绍了隐藏规则，即编译器在搜索标识符的声明时使用的搜索顺序。在“模糊访问和限定访问”部分中，介绍了避免模糊访问和规避隐藏规则的方法。

应用程序中的搜索顺序

当编译器在应用程序代码中遇到单个标识符时，它会按照以下顺序查找相关声明：

1. 方法的局部变量
2. 功能块、程序或函数以及任何基本功能块中的局部变量
3. 功能块的本地方法
4. 项目中的全局变量，前提是在声明全局变量的变量列表中没有设置“qualified_only”属性。
5. 当库和变量列表都不需要限定访问权限时，引用库中的全局变量。
6. 项目中的功能块或类型名称（即：全局变量列表、功能块等的名称）
7. 库中的功能块或类型名称
8. 本地引用库和库所发布库的命名空间

库中的搜索顺序

当编译器在库代码中遇到单个标识符时，它会按照以下顺序查找相关声明：

1. 方法的局部变量
2. 功能块、程序或函数以及任何基本功能块中的局部变量
3. 功能块的本地方法
4. 本地库中的全局变量，前提是在声明全局变量的变量列表中没有设置“qualified_only”属性。
5. 当库和变量列表都不需要限定访问权限时，引用库中的全局变量。
6. 本地库中的功能块或类型名称（例如，全局变量列表、功能块等的名称）
7. 引用库中的功能块或类型名称
8. 本地引用库和本地吸引库所发布库的命名空间。

模糊访问和限定访问

尽管存在这些搜索顺序，但仍可能会出现模糊访问。例如，当 1 个同名变量出现在 2 个需要非限定访问的全局变量列表中时，就会出现这种情况。编译器会将这种情况报告为错误（例如：对名称 <variable> 的使用不明确））。

通过限定访问可以为这种歧义使用消除歧义，例如，通过全局变量列表的名称进行访问（例如：GVL.<Variable>）。

此外，限定访问总是可以用来绕过遮蔽规则。

- 全局变量列表的名称可用于对该列表中的变量进行唯一访问。
- 库名可用于对该库中的元素进行唯一访问。
- 指针 THIS [▶ 633] 可用于对功能块中的变量进行唯一访问，即使在功能块的方法中存在同名的局部变量也是如此。

如要随时查找标识符的声明点，可选择编辑器窗口的上下文菜单中的 命令转至定义 [▶ 816]。如果编译器产生了 1 条看似难以理解的错误消息，这一点尤其有用。

在实例路径中搜索

上述搜索顺序不适用于作为组件出现在实例路径中的标识符，也不适用于在调用中作为输入的标识符。

对于 yy.Komponente 类型的访问，这取决于 yy 所描述的实体，在那里可以找到 Komponente 的声明。

- 如果 yy 引用具有结构化数据类型（即 STRUCT 或 UNION 类型）的变量，则按此顺序搜索 Komponente：
  - 功能块的局部变量
  - 基本功能块的局部变量
  - 功能块的方法

- 基本功能块的方法

- 如果 yy 引用全局变量列表或程序，则仅在该列表中搜索 Komponente。
- 如果 yy 引用某个库的命名空间，则在该库中搜索 Komponente，搜索方式与上文“库中的搜索顺序”部分所述相同。

只有在第 2 种情况下，编译器才会决定是否允许访问找到的元素，即是否只能对变量进行本地访问，或者某个方法是否是私有的。如果不允许访问，则会发出错误。

另请参见：

- 标识符 [▶ 782]
- 属性“qualified only” [▶ 761]
- THIS [▶ 633]

16.11 关键字

在所有编辑器中，标识范围、数据类型或运算符的关键字必须用大写字母书写。

关键字不能用作变量名。

示例：

ABS、ACOS、ADD、ADR、AND、ANDN、ARRAY、ASIN、AT、ATAN

BITADR、BOOL、BY、BYTE

CAL、CALC、CALCN、CASE、CONSTANT、COS

DATE、DINT、DIV、DO、DT、DWORD

ELSE、ELSIF、END_CASE、END_FOR、END_IF、END_REPEAT、END_STRUCT、END_TYPE、END_VAR、END_WHILE、EQ、EXIT、EXP、EXPT

FALSE、FOR、FUNCTION、FUNCTION_BLOCK

GE、GT

IF、INDEXOF、INT

JMP、JMPC、JMPCN

LD、LDN、LE、LINT、LN、LOG、LREAL、LT、LTIME、LWORD

MAX、METHOD、MIN、MOD、MOVE、MUL、MUX

NE、NOT

OF、OR、ORN

PARAMS、PERSISTENT、POINTER、PROGRAM

R、READ_ONLY、READ_WRITE、REAL、REFERENCE、REPEAT、RET、RETAIN、RETC、RETCN、RETURN、ROL、ROR

S、SEL、SHL、SHR、SIN、SINT、SIZEOF、SUPER、SQRT、ST、STN、STRING、STRUCT、SUPER、SUB

TAN、THEN、THIS、TIME、TO、TOD、TRUE、TRUNC、TYPE

UDINT、UINT、ULINT、UNTIL、USINT

VAR、VAR_CONFIG、VAR_EXTERNAL、VAR_GLOBAL、VAR_IN_OUT、VAR_INPUT、VAR_OUTPUT、VAR_STAT、VAR_TEMP

WHILE、WORD、WSTRING

XOR、XORN

TwinCAT 会自动检查关键字是否正确使用，并在输入关键字时立即用红色波浪线标识错误。

当 TwinCAT 创建隐式代码时，通常会赋予变量和函数 1 个包含“_”的名称。系统会自动阻止在实现代码中使用双下划线。因此，系统内部标识符与程序员所分配的标识符之间不会发生冲突。

在 TwinCAT 导出格式中会使用以下关键字。因此，您不得将它们用作标识符：

- ACTION
- END_ACTION
- END_FUNCTION
- END_FUNCTION_BLOCK
- END_PROGRAM

16.12 方法 FB_init、FB_reinit 和 FB_exit

您可以显式使用这些方法来影响功能块变量的初始化和功能块终止时的行为。

- FB_init [▶ 787]
- FB_reinit [▶ 790]
- FB_exit [▶ 791]

另请参见有关不同操作情况 [▶ 792]的信息以及这些方法与派生功能块的行为 [▶ 795]。

隐式方法的返回值类型是 BOOL。即使没有进行求值，也不应更改返回类型。

请勿在方法中添加任何输出变量。您仅可定义附加输入变量。

不建议显式调用

方法 FB_init、FB_reinit 和 FB_exit 是在不同时间隐式调用的系统函数（有关更多相关信息，请参见 操作情况 [▶ 792]）。显式调用这些方法可能会产生意外后果，因此不建议使用。

在 FB_init/FB_reinit/FB_exit 中出现异常时自动核心转储

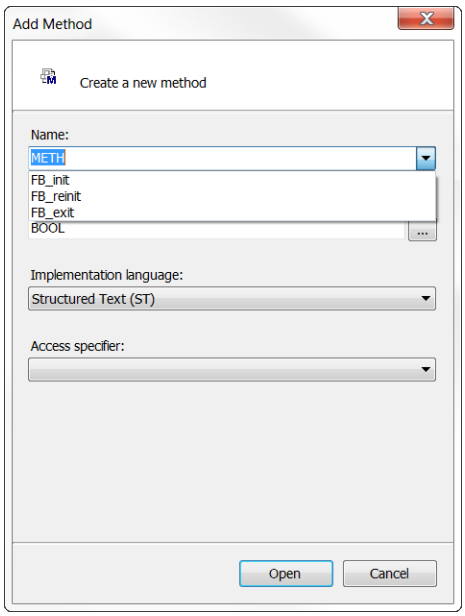
如果在 FB_init/FB_reinit/FB_exit 代码中出现异常错误（例如，由于编程错误而出现异常错误），运行时系统会自动在目标系统上存储核心转储（TC3.1 Build 4024.25 及以上）。该核心转储会以 *.core 文件的形式存储在目标系统的启动文件夹中（默认情况下，位于 *C:\TwinCAT\3.1\Boot\Plc* 下），可用于查找原因。

有关加载核心转储的更多信息，请参见：利用核心转储进行错误分析 [▶ 227]

添加方法

1. 在 **Solution Explorer**（解决方案资源管理器）的 PLC 项目树中，选择 1 个功能块。
2. 在上下文菜单中，选择命令 **Add** > **Method...**（添加 > 方法……）
   ⇨ 对话框 **Add Method**（添加方法）会打开。
3. 打开字段的下拉列表，可为方法命名。
4. 从方法 FB_init、FB_reinit 或 FB_exit 中选择 1 个。
5. 点击 **Open**（打开）。

⇨ 该方法被添加到 PLC 项目树中，并在编辑器中打开。系统会自动进行方法（返回值、参数）的定义。



返回值

隐式调用

如果对方法进行隐式调用，则系统将不会对返回值进行求值。即使您调整返回值，也不会使用隐式调用对其求值。

显式调用

如果对方法进行显式调用，则您可以对返回值进行求值。因此，您可以返回 1 个有意义的返回值。

另请参见：

- 对象方法 [▶ 82]
- 属性“call_after_init” [▶ 735]
- 属性“call_after_online_change_slot” [▶ 736]
- 属性“call_on_type_change” [▶ 737]
- 属性“no_copy” [▶ 751]
- 属性“no-exit” [▶ 753]

16.12.1 FB_init

FB_init 始终隐式可用，通常会在执行标准初始化（隐式调用）时被调用。您也可以显式声明方法，并在标准初始化代码中添加附加代码，以施加特定影响。

如果在执行过程中到达显式定义的初始化代码，则在自动零初始化和单独值初始化的情况下，功能块实例已经通过隐式初始化代码完全初始化。因此，不允许进行 SUPER^.FB_init 调用。

调试

在 FB_init 方法中查找错误非常费力，因为除了其他原因之外，设置断点无法达到预期效果。

在 FB_init/FB_reinit/FB_exit 中出现异常时自动核心转储

如果在 FB_init/FB_reinit/FB_exit 代码中出现异常错误（例如，由于编程错误而出现异常错误），运行时系统会自动在目标系统上存储核心转储（TC3.1 Build 4024.25 及以上）。该核心转储会以 *.core 文件的形式存储在目标系统的启动文件夹中（默认情况下，位于 *C:\TwinCAT\3.1\Boot\Plc* 下），可用于查找原因。

有关加载核心转储的更多信息，请参见：利用核心转储进行错误分析 [▶ 227]

注意初始化的顺序

请注意，在执行 FB_init 时，尚不知道分配给功能块实例的输入变量的值。如果您想要对 FB_init 的执行进行参数设置，则您可以在 FB_init 方法中声明额外的方法输入。
有关更多信息，请参见 操作情况 [▶ 792]。

显式调用 FB_init

不建议显式调用

方法 FB_init、FB_reinit 和 FB_exit 是在不同时间隐式调用的系统函数（有关更多相关信息，请参见 操作情况 [▶ 792]）。显式调用这些方法可能会产生意外后果，因此不建议使用。

原则上，也可以显式调用 FB_init 方法。然而，不建议采用这种做法。

在显式调用的情况下，不仅要再次执行 FB_init 方法的显式初始化代码，还要重复进行隐式初始化。例如，在自动零初始化和单独值初始化的情况下，所有变量都要重新初始化。

后者的效果是，对于在该实例中声明的 FB 实例也要执行这 2 个初始化步骤。该程序将在所包含的更多实例级别上继续进行。

如果 FB 实例在没有首先进行反初始化的情况下被重新初始化，这可能会导致在后续程序序列中出现问题，例如，如果 FB_init 方法包含动态内存分配或实例计数器的话。

由于此处的 FB 并非用于其预期目的，而且可能无法估计其问题后果，因此我们明确建议您不要显式调用 FB_init 方法。

方法 FB_init 的接口

```
METHOD FB_init : BOOL
VAR_INPUT
    bInitRetains : BOOL; // if TRUE, the retain variables are initialized (warm start / cold start)
    bInCopyCode  : BOOL; // if TRUE, the instance afterwards gets moved into the copy code (online c
hange)
END_VAR
```

通过对 FB_init 方法参数进行求值，您可以区分不同的操作情况，并在必要时调整实现方式。（请参见 操作情况 [▶ 792]）

方法参数	（首次/新）下载	在线更改	重新启动 TwinCAT（无新下载）
bInitRetains	TRUE	FALSE	FALSE
bInCopyCode	FALSE	TRUE	FALSE

您可以在 FB_init 方法中声明额外的方法输入。在这种情况下，您必须在功能块实例的声明中设置这些输入。

返回值

隐式调用

如果对方法进行隐式调用，则系统将不会对返回值进行求值。即使您调整返回值，也不会使用隐式调用对其求值。

显式调用

如果对方法进行显式调用，则您可以对返回值进行求值。因此，您可以返回 1 个有意义的返回值。

带有派生功能块的 FB_init

如果 1 个功能块是从另一个功能块派生出来的，则会为该功能块自动执行基本功能块的 FB_init 方法。如果显式添加派生功能块的 FB_init 方法，则会在基本功能块的 FB_init 方法之后执行该方法（请参见 派生功能块的行为 [▶ 796]）。

如果派生功能块的 FB_init 方法要以显式形式出现，则它必须定义与基本功能块的 FB_init 方法相同的参数。为了为派生实例设置特殊的初始化，还可以添加更多参数。

FB_init 方法不能与 C#、C++ 或 Java 中已知的构造函数的构造进行比较，因为 PLC 中的功能块不需要构造函数来初始化其声明的变量。这会在声明行中以隐式或显式的方式进行。

有关扩展其他功能块的功能块所产生的后果，请参见派生功能块的行为 [▶ 795]部分。

分配附加 FB_init 参数的示例

示例 1：

方法 FB_SerialDevice.FB_init

```
METHOD PUBLIC FB_init : BOOL
VAR_INPUT
    bInitRetains : BOOL; // if TRUE, the retain variables are initialized (warm start / cold start)
    bInCopyCode  : BOOL; // if TRUE, the instance afterwards gets moved into the copy code (online c
hange)
    nComNum      : INT;  // additional input: number of the COM-interface that is to be observed
END_VAR
```

功能块 FB_SerialDevice 的声明：

```
fbCom1  : FB_SerialDevice(nComNum := 1);
fbCom0  : FB_SerialDevice(nComNum := 0);
```

示例 2：

功能块 FB_Sample：

```
FUNCTION_BLOCK FB_Sample
VAR
    nStartValue : INT;
END_VAR
```

方法 FB_Sample.FB_init：

```
METHOD FB_init : BOOL
VAR_INPUT
    bInitRetains : BOOL; // if TRUE, the retain variables are initialized (warm start / cold start)
    bInCopyCode  : BOOL; // if TRUE, the instance afterwards gets moved into the copy code (online c
hange)
    nValue       : INT;  // parameter to initialize the start value
END_VAR
```

```
nStartValue := nValue;
```

功能块 FB_Sample 的声明：

```
PROGRAM MAIN
VAR
    fbSample1 : FB_Sample(123); // => fbSample1.nStartValue is set to 123
    fbSample2 : FB_Sample(456); // => fbSample2.nStartValue is set to 456
END_VAR
```

示例 3：

在下面的示例中，1 个输入变量和 1 个属性由 1 个功能块初始化，该功能块的 FB_init 方法 [▶ 787]带有 1 个附加参数。

功能块 FB_Sample：

```
FUNCTION_BLOCK FB_Sample
VAR_INPUT
    nInput           : INT;
END_VAR
VAR
    nLocalInitParam  : INT;
    nLocalProp       : INT;
END_VAR
```

方法 FB_Sample.FB_init：

```
METHOD FB_init : BOOL
VAR_INPUT
    bInitRetains : BOOL;  // if TRUE, the retain variables are initialized (warm start / cold start)
    bInCopyCode  : BOOL;  // if TRUE, the instance afterwards gets moved into the copy code (online
```

```
change)
    nInitParam   : INT;
END_VAR
```

```
nLocalInitParam := nInitParam;
```

属性 FB_Sample.nMyProperty 和相关的 Set 函数：

```
PROPERTY nMyProperty : INT
```

```
nLocalProp := nMyProperty;
```

MAIN 程序：

```
PROGRAM MAIN
VAR
    fbSample  : FB_Sample(nInitParam := 1) := (nInput := 2, nMyProperty := 3);
    aSample   : ARRAY[1..2] OF FB_Sample[(nInitParam := 4), (nInitParam := 7)]
                            := [(nInput := 5, nMyProperty := 6), (nInput := 8, nMyProperty := 9)];
END_VAR
```

初始化结果：

- fbSample
  - nInput = 2
  - nLocalInitParam = 1
  - nLocalProp = 3
- aSample[1]
  - nInput = 5
  - nLocalInitParam = 4
  - nLocalProp = 6
- aSample[2]
  - nInput = 8
  - nLocalInitParam = 7
  - nLocalProp = 9

另请参见：

- 属性“call_after_init” [▶ 735]
- 操作情况 [▶ 792]

16.12.2 FB_reinit

如果需要，您必须显式实现 FB_reinit。如果存在该方法，则会在复制相应功能块的实例之后自动调用该方法（隐式调用）。在在线更改期间会发生这种情况，在更改功能块声明（签名更改）以重新初始化新的实例块之后。

不建议显式调用

方法 FB_init、FB_reinit 和 FB_exit 是在不同时间隐式调用的系统函数（有关更多相关信息，请参见 操作情况 [▶ 792]）。显式调用这些方法可能会产生意外后果，因此不建议使用。

在 FB_init/FB_reinit/FB_exit 中出现异常时自动核心转储

如果在 FB_init/FB_reinit/FB_exit 代码中出现异常错误（例如，由于编程错误而出现异常错误），运行时系统会自动在目标系统上存储核心转储（TC3.1 Build 4024.25 及以上）。该核心转储会以 *.core 文件的形式存储在目标系统的启动文件夹中（默认情况下，位于 *C:\TwinCAT\3.1\Boot\Plc* 下），可用于查找原因。

有关加载核心转储的更多信息，请参见：利用核心转储进行错误分析 [▶ 227]

方法 FB_reinit 的接口

```
METHOD FB_reinit : BOOL
```

返回值

隐式调用

如果对方法进行隐式调用，则系统将不会对返回值进行求值。即使您调整返回值，也不会使用隐式调用对其求值。

显式调用

如果对方法进行显式调用，则您可以对返回值进行求值。因此，您可以返回 1 个有意义的返回值。

带有派生功能块的 FB_reinit

如要重新初始化功能块的基本实现，则必须为基本功能块显式调用 FB_reinit（通过 SUPER^.FB_reinit()）。可以对返回值进行评估。

16.12.3 FB_exit

如果需要，您必须显式实现 FB_exit。如果存在该方法，则会在控制器删除功能块实例的代码之前自动（隐式）调用该方法（例如，即使 TwinCAT 从运行模式切换到配置模式也是如此）。

不建议显式调用

方法 FB_init、FB_reinit 和 FB_exit 是在不同时间隐式调用的系统函数（有关更多相关信息，请参见 操作情况 [▶ 792]）。显式调用这些方法可能会产生意外后果，因此不建议使用。

在 FB_init/FB_reinit/FB_exit 中出现异常时自动核心转储

如果在 FB_init/FB_reinit/FB_exit 代码中出现异常错误（例如，由于编程错误而出现异常错误），运行时系统会自动在目标系统上存储核心转储（TC3.1 Build 4024.25 及以上）。该核心转储会以 *.core 文件的形式存储在目标系统的启动文件夹中（默认情况下，位于 *C:\TwinCAT\3.1\Boot\Plc* 下），可用于查找原因。

有关加载核心转储的更多信息，请参见：利用核心转储进行错误分析 [▶ 227]

FB_exit 方法的接口

```
METHOD FB_exit : BOOL
VAR_INPUT
    bInCopyCode : BOOL; // if TRUE, the exit method is called for exiting an instance that is copied
 afterwards (online change)
END_VAR
```

通过对 FB_exit 方法参数进行求值，您可以区分不同的操作情况，并在必要时调整实现方式。（请参见 操作情况 [▶ 792]）

方法参数	（首次/新）下载	在线更改
bInCopyCode	FALSE	TRUE

返回值

隐式调用

如果对方法进行隐式调用，则系统将不会对返回值进行求值。即使您调整返回值，也不会使用隐式调用对其求值。

显式调用

如果对方法进行显式调用，则您可以对返回值进行求值。因此，您可以返回 1 个有意义的返回值。

带有派生功能块的 FB_exit

如果 1 个功能块是从另一个功能块派生出来的，则会为该功能块自动执行基本功能块的 FB_exit 方法。如果显式添加派生功能块的 FB_exit 方法，则会首先执行该方法，然后再执行基本功能块的 FB_exit 方法（请参见 派生功能块的行为 [▶ 796]）。

另请参见：

- 属性“no-exit” [▶ 753]

16.12.4 操作情况

在不同时间隐式调用方法 FB_init、FB_reinit 和 FB_exit。有关这些不同操作情况的信息，请参见下文：

- 操作情况“首次下载”
- 操作情况“新下载”
- 操作情况“在线更改”

方法的调用行为根据操作情况而有所不同。通过 FB_init 和 FB_exit 查询方法参数“bInCopyCode”和“bInitRetains”，您可以区分方法内的操作情况。这样，您可以根据相关的操作情况调整实现方式。

操作情况 “首次下载”	操作情况 “新下载”	操作情况 “在线更改”
1. FB_init（隐式和显式初始化代码） 2. 通过功能块的实例声明进行的显式外部变量初始化 3. 使用属性“call_after_init”声明的方法	1. FB_exit 2. FB_init（隐式和显式初始化代码） 3. 通过功能块的实例声明进行的显式外部变量初始化 4. 使用属性“call_after_init”声明的方法	1. FB_exit 2. FB_init（隐式和显式初始化代码） 3. 通过功能块的实例声明进行的显式外部变量初始化 4. 使用属性“call_after_init”声明的方法 5. 复制程序 6. FB_reinit
方法参数： FB_init(bInitRetains := TRUE, bInCopyCode := FALSE);	方法参数： FB_exit(bInCopyCode := FALSE); FB_init(bInitRetains := TRUE, bInCopyCode := FALSE);	方法参数： FB_exit(bInCopyCode := TRUE); FB_init(bInitRetains := FALSE, bInCopyCode := TRUE);

操作情况“首次下载”

在将 PLC 项目下载到处于出厂状态的控制器时，必须通过初始化将所有变量的内存位置设置为所需的初始状态。在此过程中，将在功能块实例的数据区中填入所需的值。在这种情况下，通过为功能块显式实现 FB_init [▶ 787]，您可以有目的地影响初始化。

通过对 FB_init 方法参数“bInitRetains”（TRUE）和“bInCopyCode”（FALSE）求值，您可以明确检测到这种操作情况。(另见操作情况"在线更改")。

FB_init 方法不能与 C#、C++ 或 Java 中已知的构造函数的构造进行比较，因为 PLC 中的功能块不需要构造函数来初始化其声明的变量。这会在声明行中以隐式或显式的方式进行。

有关扩展其他功能块的功能块所产生的后果，请参见派生功能块的行为 [▶ 795]部分。

属性“call_after_init”作为 FB_init 的替代

在调用 FB_init 之后和 PLC 项目任务的第 1 个周期之前，在功能块实例的声明中会处理外部初始赋值。

```
fbTimer : TON := (PT := T#500MS);
```

只有在调用 FB_init 之后才会执行此类赋值。如果 TON 有 1 个 FB_init 方法，则 T#500MS 的分配时间值在该方法中是未知的。

如要控制此类赋值的效果，您可以将属性 {attribute 'call after init'} [▶ 735] 分配给功能块的方法。您必须在相应方法的声明部分上方和功能块主体的声明部分上方插入属性。

在处理外部初始赋值之后和 PLC 项目任务开始之前会调用该方法，因此它可以适当地响应用户的规范要求（使用外部初始化）。

示例

```
{attribute 'call_after_init'}
FUNCTION_BLOCK FB_Sample
VAR_INPUT
    nInput1   : INT;
    nInput2   : INT := 100;
END_VAR
```

```
METHOD FB_init : BOOL
VAR_INPUT
    bInitRetains    : BOOL; // if TRUE, the retain variables are initialized (warm start / cold
start)
    bInCopyCode     : BOOL; // if TRUE, the instance afterwards gets moved into the copy code (online
change)
END_VAR
```

```
{attribute 'call_after_init'}
METHOD MyCallAfterInit
```

```
PROGRAM MAIN
VAR
    fbSample  : FB_Sample := (nInput2 := 700);
END_VAR
```

在下载过程中，依次进行以下调用：

1. FB_init	初始化实例

	1. 隐式初始化代码（变量的隐式零初始化和显式内部值初始化）
	`nInput1 := 0;`
	`nInput2 := 100;`
	2. 显式初始化代码（在 FB_init 中显式定义的初始化代码）

	`fbSample.FB_init(bInitRetains := TRUE, bInCopyCode := FALSE);`
	通过对 FB_init 方法参数求值，您可以明确检测到这种操作情况。 例如，通过 FB_init 方法，您可以借助附加的初始化代码对变量进行初始化，或者将某些变量在内存中的位置通知 PLC 项目的其他部分。
2. 通过功能块的实例声明进行的显式外部变量初始化	处理外部初始赋值
	`nInput2 := 700;`
	如果功能块的输入变量已赋值，则会复制它们。如果变量没有从外部赋值，则保留原始值。
3. 使用属性“call_after_init”声明的方法	替代初始化
	`fbSample.MyCallAfterInit();`
	为了明确检测到该操作情况，您可以事先将 bInCopyCode 的值复制到 FB_init 中的辅助变量中，并在“call_after_init”方法中对该辅助变量进行求值。 您可以使用该属性作为 FB_init 的替代，或者，例如，检查显式外部变量初始化的效果。 让实现尽可能独立。您也可随时从 PLC 项目调用该方法，将功能块实例恢复到初始状态。

操作情况“新下载”

当再次下载 PLC 项目时，控制器上已经存在的项目可能会被替换。因此，必须首先以可控的方式释放现有功能块的内存空间。为此，您可以使用 FB_exit [▶ 791] 方法。如果已经创建该方法，则将在删除旧项目之前调用该方法。随后，系统会将新项目加载到控制器上并调用 FB_init。例如，在 FB_exit 中，您可以将外部资源（使用套接字或文件句柄）设置为已定义的状态，或者释放动态分配的内存（__NEW-，或者在本例中为__DELETE-Operator）。

通过对 FB_exit 方法参数“bInCopyCode”（FALSE）求值，您可以明确检测到这种操作情况。

操作情况“在线更改”

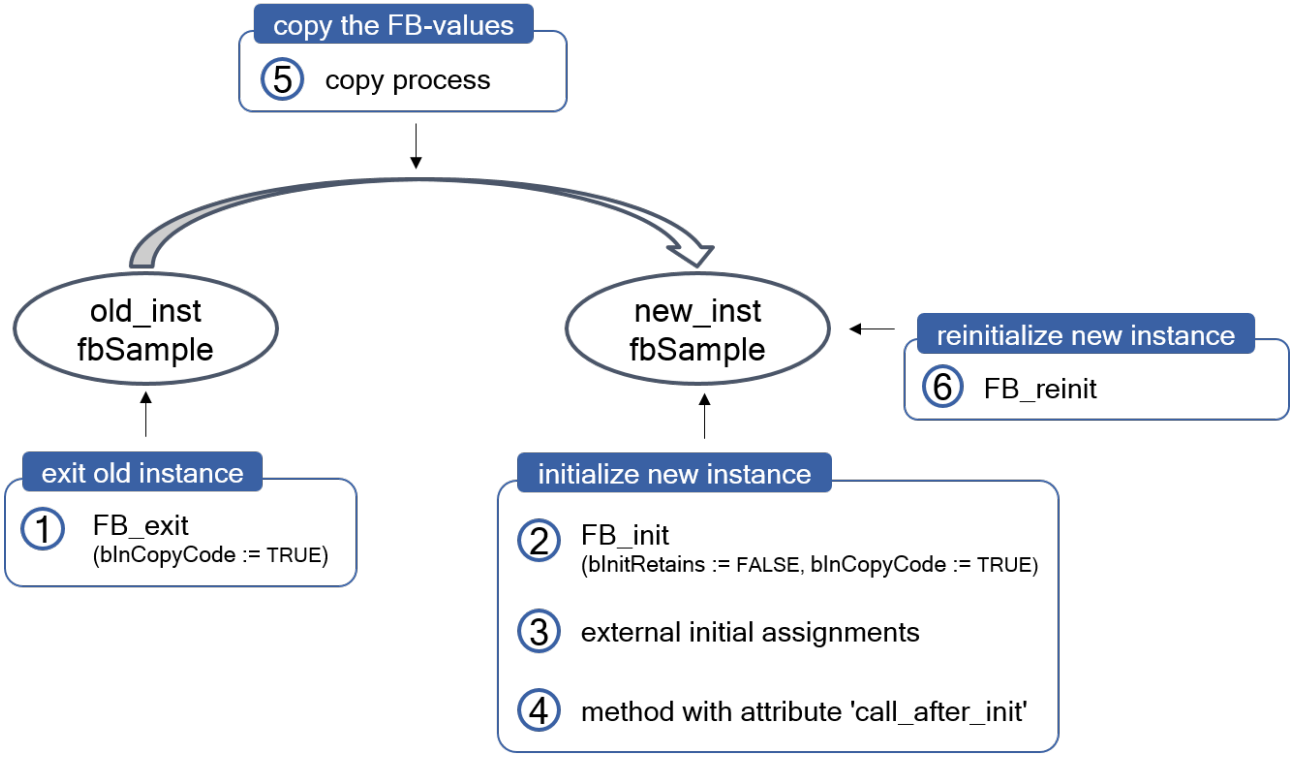
在在线更改期间，您可以通过方法 FB_exit、FB_init 和 FB_reinit 来影响功能块实例的初始化。作为在线更改的一部分，在当前控制系统中会跟踪在离线操作期间对 PLC 项目所做的更改。因此，功能块的“旧”实例会尽可能顺利地被其“新”同类所取代。

如果在登录 PLC 项目之前没有对功能块的声明部分进行更改（即仅对实现部分进行更改），则不会替换数据区。仅会替换代码块。在这种情况下，不会调用方法 FB_exit、FB_init 和 FB_reinit。

如果您对功能块的声明进行更改并导致上述复制过程，则您会在在线更改期间收到 1 条关于“可能的意外影响”的消息。消息框 **Details**（详细信息）包含要复制的所有实例的列表。

在 FB_init 和 FB_exit 方法的代码中，通过对参数“bInitRetains”（FALSE）和“bInCopyCode”（TRUE）求值，可以确定当前是否正在执行与功能块实例有关的在线更改，并将它们转移到不同的内存位置。

在在线更改过程中，依次进行以下调用：



1. FB_exit	退出旧实例
	`old_inst.FB_exit(bInCopyCode := TRUE);`
	通过对 FB_exit 方法参数求值，您可以明确检测到这种操作情况。 在离开“旧”实例时，您可以调用 FB_exit，以便在复制之前触发某些清除任务。这样，您可以为下一次复制操作准备数据，并影响“新”实例的状态。PLC 项目的其他部分可以通知您内存中即将发生的位置变化。 请特别注意 POINTER 或 REFERENCE 类型的变量。在在线更改之后，这些变量可能不再指向所需的存储位置。（注意：您可以使用 属性“call on type change” [▶ 737] 来响应引用功能块的数据类型更改）。 接口变量（INTERFACE）由编译器单独处理，并在在线更改期间进行相应调整。新实例可能会不加改变地接受外部资源（例如，套接字、文件或其他句柄）。在许多情况下，在在线更改过程中不需要单独处理它们。（请参见操作情况“新下载”）
2. FB_init	初始化新实例
	1. 隐式初始化代码（变量的隐式零初始化和显式内部值初始化） 2. 显式初始化代码（在 FB_init 中显式定义的初始化代码）
	`new_inst.FB_init(bInitRetains := FALSE, bInCopyCode := TRUE);`
	通过对 FB_init 方法参数求值，您可以明确检测到这种操作情况。 FB_init 调用会在复制操作之后进行，可用于执行在线更改的特定操作。例如，您可以将某些变量在内存中的新位置通知 PLC 项目的其他部分。
3. 通过功能块的实例声明进行的显式外部变量初始化	处理外部初始赋值
	`new_inst : <FB-Name> := (<Variable>:=<value>);`
	如果功能块的输入变量已赋值，则会复制它们。如果变量没有从外部赋值，则保留原始值。
4. 使用属性“call_after_init”声明的方法	替代初始化
	`new_inst.<Methodenname der gekennzeichneten Methode>();`
	为了明确检测到该操作情况，您可以事先将 bInCopyCode 的值复制到 FB_init 中的辅助变量中，并在“call_after_init”方法中对该辅助变量进行求值。 例如，您可以使用该属性作为 FB_init 的替代。 让实现尽可能独立。您也可随时从 PLC 项目调用该方法，将功能块实例恢复到初始状态。
5. 复制过程	复制功能块值（复制代码）
	`copy(&old_inst, &new_inst);`
	保留现有值。为此，将它们从旧实例复制到新实例。
6. FB_reinit	重新初始化新实例
	`new_inst.FB_reinit();`
	在复制操作之后会调用该方法。它将实例的变量设置为已定义的值。 例如，您可以对内存中“新”位置的变量进行相应的初始化，或者将某些变量在内存中的新位置通知 PLC 项目的其他部分。 实现的处理始终应该独立于在线更改。您也可随时从 PLC 项目调用该方法，将功能块实例恢复到初始状态。

属性 {attribute 'no copy'} [▶ 751] 可用于防止在在线更改过程中复制功能块的特定变量。在这种情况下，变量始终会保留初始值。

另请参见：

- 属性“call_after_init” [▶ 735]
- 属性“call_on_type_change” [▶ 737]
- 属性“no copy” [▶ 751]
- 属性“no-exit” [▶ 753]

16.12.5 派生功能块的行为

在派生功能块的上下文中使用方法 FB_init、FB_reinit 和 FB_exit 时，请注意以下几点。

带有派生功能块的 FB_init

如果 1 个功能块是从另一个功能块派生出来的，则会为该功能块自动执行基本功能块的 FB_init 方法。如果显式添加派生功能块的 FB_init 方法，则会在基本功能块的 FB_init 方法之后执行该方法（请参见 派生功能块的行为 [▶ 796]）。

如果派生功能块的 FB_init 方法要以显式形式出现，则它必须定义与基本功能块的 FB_init 方法相同的参数。为了为派生实例设置特殊的初始化，还可以添加更多参数。

带有派生功能块的 FB_reinit

如要重新初始化功能块的基本实现，则必须为基本功能块显式调用 FB_reinit（通过 SUPER^.FB_reinit()）。可以对返回值进行评估。

带有派生功能块的 FB_exit

如果 1 个功能块是从另一个功能块派生出来的，则会为该功能块自动执行基本功能块的 FB_exit 方法。如果显式添加派生功能块的 FB_exit 方法，则会首先执行该方法，然后再执行基本功能块的 FB_exit 方法（请参见 派生功能块的行为 [▶ 796]）。

派生功能块的调用顺序示例

功能块 FB_Base、FB_Sub 和 FB_SubSub 是相互派生的。以下规则适用：

- FB_Sub 扩展 FB_Base
- FB_SubSub 扩展 FB_Sub

情况：

- 所有 3 个功能块都有自己的 FB_init、FB_reinit 和 FB_exit 方法。
- 功能块 FB_SubSub 被实例化。通过在线更改，在功能块中添加 1 个附加变量。

假设 - 情况 1：

- 无需重新初始化基本实现。这意味着，在方法 FB_SubSub.FB_reinit 中不存在 SUPER^.FB_reinit() 调用。

调用顺序：

功能块实例 fbSubSub 的方法 FB_exit、FB_init 和 FB_reinit 的调用顺序如下：

1. Impliziter Aufruf von FB_SubSub.FB_exit(TRUE) für fbSubSub
2. Impliziter Aufruf von FB_Sub.FB_exit(TRUE) für fbSubSub
3. Impliziter Aufruf von FB_Base.FB_exit(TRUE) für fbSubSub
4. Impliziter Aufruf von FB_Base.FB_init(FALSE, TRUE) für fbSubSub
5. Impliziter Aufruf von FB_Sub.FB_init(FALSE, TRUE) für fbSubSub
6. Impliziter Aufruf von FB_SubSub.FB_init(FALSE, TRUE) für fbSubSub
7. Impliziter Aufruf von FB_SubSub.FB_reinit() für fbSubSub

假设 - 情况 2：

- 需要重新初始化基本实现。这意味着，方法 FB_SubSub.FB_reinit 和 FB_Sub.FB_reinit 包含 SUPER^.FB_reinit() 调用。

调用顺序：

功能块实例 fbSubSub 的方法 FB_exit、FB_init 和 FB_reinit 的调用顺序如下：

1. Impliziter Aufruf von FB_SubSub.FB_exit(TRUE) für fbSubSub
2. Impliziter Aufruf von FB_Sub.FB_exit(TRUE) für fbSubSub
3. Impliziter Aufruf von FB_Base.FB_exit(TRUE) für fbSubSub
4. Impliziter Aufruf von FB_Base.FB_init(FALSE, TRUE) für fbSubSub
5. Impliziter Aufruf von FB_Sub.FB_init(FALSE, TRUE) für fbSubSub
6. Impliziter Aufruf von FB_SubSub.FB_init(FALSE, TRUE) für fbSubSub

7. Impliziter Aufruf von FB_SubSub.FB_reinit() für fbSubSub
8. Expliziter Aufruf von FB_Sub.FB_reinit() für fbSubSub
9. Expliziter Aufruf von FB_Base.FB_reinit() für fbSubSub

17 参照用户界面

TwinCAT 用户界面默认显示主命令。如要自定义菜单配置，从 **Tools（工具）** 菜单中选择 **Customize（自定义）**。

另请参见：

- --- FEHLENDER LINK ---

17.1 文件

17.1.1 存档选项

TwinCAT 开发环境提供了可用于存储 TwinCAT 项目的 3 种不同存档文件类型，例如，用于存档或传递给同事：tnzip、tszip 和 tpzip。

您应该使用的存档文件的类型取决于您想要在存档文件夹中存储哪些项目。

存档文件类型	焦点	内容	命令
tnzip	解决方案 [▶ 798]	*.tnzip 存档文件夹包含在解决方案中所包含的所有 TwinCAT 项目类型。 这些项目可以是 TwinCAT、TwinCAT HMI、Scope 或 Connectivity 项目。	构建 [▶ 798] 打开 [▶ 799]
tszip	TwinCAT 项目 [▶ 799]	*.tszip 存档文件夹包含要存档的 TwinCAT 项目。	构建 [▶ 988] 打开 [▶ 799]
tpzip	PLC 项目 [▶ 800]	*.tpzip 存档文件夹包含要存档的 PLC 项目。	构建 [▶ 800] 打开 [▶ 800]

17.1.1.1 解决方案

- 创建 *.tnzip TwinCAT 解决方案存档：命令将 <solution name> 另存为存档…… [▶ 798]
- 打开 *.tnzip TwinCAT 解决方案存档：命令从存档中打开解决方案 [▶ 799]

17.1.1.1.1 命令将 <solution name> 另存为存档……

功能：此命令可打开用于将文件另存为存档的标准对话框。解决方案可作为 ***.tnzip** 存档保存在目标路径下。

调用：File（文件）菜单，上下文菜单

要求：在 Solution Explorer（解决方案资源管理器）中选择解决方案。

***.tnzip 的内容**	*.tnzip 存档文件夹包含在解决方案中所包含的所有 TwinCAT 项目类型。 这些项目可以是 TwinCAT、TwinCAT HMI、Scope 或 Connectivity 项目。
打开命令	使用以下命令可以重新打开 tnzip 存档： 命令从存档中打开解决方案 [▶ 799]。
关于 PLC 项目的说明	如果解决方案包含 1 个或多个 PLC 项目，则在这些 PLC 项目的存档文件夹中存储的文件和文件夹将取决于相关项目的 PLC 项目设置。 设置选项卡 [▶ 859]

17.1.1.1.2 命令从存档中打开解决方案

功能：此命令可提取 TwinCAT 解决方案存档 ***.tnzip**。

调用：菜单 **File** > **Open**（文件 > 打开）

执行此命令打开 **Open**（打开）对话框。从文件系统中选择存档文件并确认对话框。**Select Folder for new Solution**（选择新解决方案的文件夹）对话框打开。选择用于存储所提取解决方案文件的文件夹。

***.tnzip 的内容**	*.tnzip 存档文件夹包含在解决方案中所包含的所有 TwinCAT 项目类型。 这些项目可以是 TwinCAT、TwinCAT HMI、Scope 或 Connectivity 项目。
创建命令	使用以下命令可以创建 tnzip 存档： 命令将 <solution name> 另存为存档…… [▶ 798]
关于 PLC 项目的说明	如果解决方案包含 1 个或多个 PLC 项目，则在这些 PLC 项目的存档文件夹中存储的文件和文件夹将取决于相关项目的 PLC 项目设置。 设置选项卡 [▶ 859]

17.1.1.2 TwinCAT 项目

- 创建 *.tszip TwinCAT 项目存档：命令将 <TwinCAT project name> 另存为存档…… [▶ 988]
- 打开 *.tszip TwinCAT 项目存档：命令项目/解决方案（打开项目/解决方案） [▶ 799]

17.1.1.2.1 命令项目/解决方案（打开项目/解决方案）

符号： 

热键：[Ctrl] + [Shift] + [O]

功能：此命令可打开用于打开文件的默认对话框。您可以在此处浏览 TwinCAT 项目文件的文件系统并在开发系统中打开它。

调用：菜单 **File** > **Open**（文件 > 打开）

打开项目对话框

文件类型	用于过滤文件类型的选择列表 • 可以打开所有支持格式的文件。
选项	• 添加（例如，该选项仅用于将测量项目添加至解决方案。不要使用该选项将多个 TwinCAT 项目添加至解决方案。） • 关闭解决方案
打开	TwinCAT 打开所选项目文件。必要时，首先进行转换。

TwinCAT *.tszip 项目存档

***.tszip 的内容**	*.tszip 存档文件夹包含要存档的 TwinCAT 项目。
创建命令	使用以下命令可以创建 tszip 存档： 命令将 <TwinCAT project name> 另存为存档…… [▶ 988]
关于 PLC 项目的说明	如果 TwinCAT 项目包含 1 个或多个 PLC 项目，则在这些 PLC 项目的存档文件夹中存储的文件和文件夹将取决于相关项目的 PLC 项目设置。 设置选项卡 [▶ 859]

另请参见：

- PLC 文档：您的第 1 个 TwinCAT 3 PLC 项目 [▶ 23]
- PLC 文档：创建标准项目 [▶ 48]

17.1.1.3 PLC 项目

- 创建 *.tpzip PLC 项目存档：命令将 <PLC project name> 另存为存档…… [▶ 800]
- 打开 *.tpzip PLC 项目存档：命令添加现有项目（项目） [▶ 800]

17.1.1.3.1 命令将 <PLC project name> 另存为存档……

功能：此命令可打开用于将文件另存为存档的标准对话框。PLC 项目可作为 ***.tpzip** 存档保存在目标路径下。

调用：File（文件）菜单，上下文菜单

要求：在 Solution Explorer（解决方案资源管理器）中选择 TwinCAT PLC 项目（<PLC project name>）。

存档文件夹中的文件和文件夹取决于 PLC 项目设置。

***.tpzip 的内容**	*.tpzip 存档文件夹包含要存档的 PLC 项目。
打开命令	使用以下命令可以重新打开 tpzip 存档： 命令添加现有项目（项目） [▶ 800]
关于 PLC 项目的说明	在 PLC 项目的存档文件夹中存储的文件和文件夹取决于该 PLC 项目的 PLC 项目设置。 设置选项卡 [▶ 859]

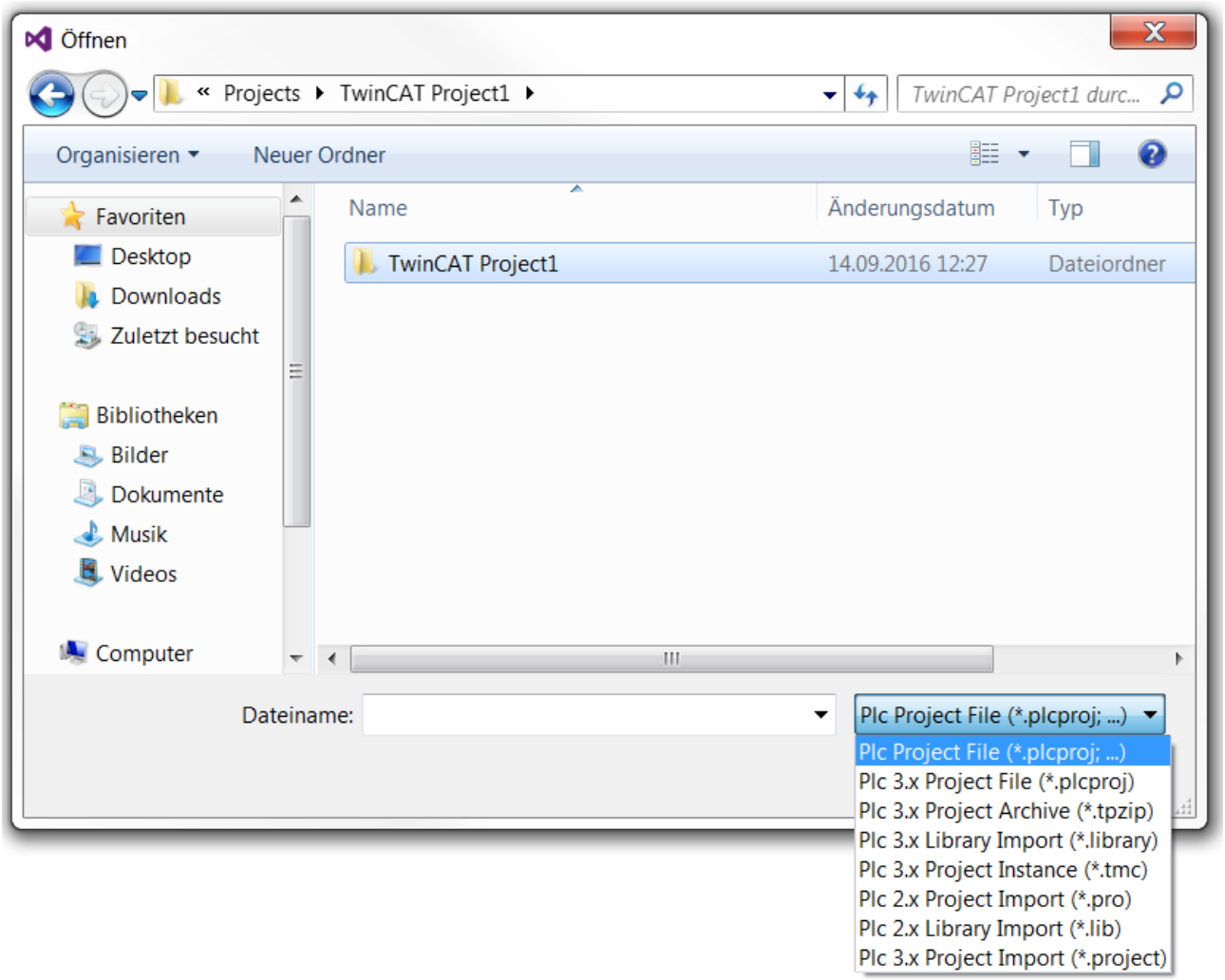
17.1.1.3.2 命令添加现有项目（项目）

符号： 

功能：此命令可打开标准浏览器对话框，该对话框可用来搜索 PLC 项目文件并在编程系统中打开它。如果安装了合适的转换器，则您可以打开不同格式的项目。

调用：Project（项目）菜单或 Solution Explorer（解决方案资源管理器）中的 PLC 对象上下文菜单

要求：在 TwinCAT 项目树中选择 PLC 节点。



文件类型	默认情况下，您可以将筛选器设置为以下 1 种文件类型： • PLC 3.x 项目文件（*.PLCproject）：TwinCAT 3 PLC 项目，带扩展名“.PLCproject” • PLC 3.x 项目存档（*.tpzip）：TwinCAT 3 PLC 项目存档，带扩展名“.tpzip” ◦ 另请参见：命令将 <PLC project name> 另存为存档…… [▶ 800] • PLC 3.x 库导入（*.library）：TwinCAT 3 PLC 库，带扩展名“.library” • PLC 2.x 项目文件（*.pro）：TwinCAT 2 PLC 项目，带扩展名“.pro” • PLC 2.x 导入库（*.lib）：TwinCAT 2 PLC 库，带扩展名“.lib” • PLC 3.x 项目导入（*.PLCproject）：PLC 项目，带扩展名“.project”
打开	所选项目打开或转换后打开。

*.tpzip PLC 项目存档

***.tpzip 的内容**	*.tpzip 存档文件夹包含要存档的 PLC 项目。
创建命令	使用以下命令可以创建 tpzip 存档： 命令将 <PLC project name> 另存为存档…… [▶ 800]
关于 PLC 项目的说明	在 PLC 项目的存档文件夹中存储的文件和文件夹取决于该 PLC 项目的 PLC 项目设置。 设置选项卡 [▶ 859]

打开 PLC 项目时可能出现的情况

当您打开项目时，可能会出现以下情况：

1. 另一个项目仍打开。 [▶ 802]
2. 项目以 TwinCAT 3 的旧版本保存。 [▶ 802]
3. 项目未以 TwinCAT 3 保存。 [▶ 802]
4. 项目未正确终止并启用“自动保存”。 [▶ 803]
5. 项目为只读。 [▶ 803]
6. 该库安装在库的存储库中并从中检索。 [▶ 803]

1. 另一个项目仍打开。

您会被询问是否保存并关闭其他项目。

2. 项目以 TwinCAT 3 的旧版本保存。

如果因为打开的项目以 TwinCAT 3 的旧版本保存而导致文件格式不同，则有两个选项：

- 如果无法以当前所用编程系统的格式保存项目，则必须对其进行更新以便继续处理项目。此时出现的表达式：**The changes you made... (您作出的更改...)** 指加载项目时各组件的内部任务。
- 如果仍将项目保存为之前的格式，则您可以决定是否更新或保留格式。如果您决定保留格式，则数据可能丢失。如果您决定更新格式，则无法再通过旧版本编程系统打开项目。

除了文件格式外，显式插入库的版本、可视化配置文件以及打开项目的编译器版本可能与当前编程系统安装的版本不同。

如果在当前编程系统上安装了更新的版本，则 **Project Environment (项目环境)** 对话框自动打开，您可在其中更新版本。如果此时未更新，则可在之后随时通过 **Options > Project Environment (选项 > 项目环境)** 对话框进行更新。

注意编译器版本

如果打开一个用旧版本编程系统创建的项目，并且在项目设置中设置了该项目的最新编译器版本，则当编译器版本的项目环境设置在新编程系统中设置为 **Do not update (不更新)** 时，将继续使用上一次在旧项目中使用的编译器版本 (即不是新环境中的“当前”版本)。

3. 项目未以 TwinCAT 3 保存。

案例 1)

如果您在选择要打开的项目时设置了文件过滤器，并且有合适的转换器可用，则会自动使用转换器并将项目转换为当前格式。该转换特定于转换器实现。通常，系统会提示您定义对引用库或设备引用的处理。

TwinCAT 3 转换器

在导入期间，只有在转换器能够编译该项目而且没有错误时才能成功地根据 TwinCAT 3 语法调整 TwinCAT PLC 控制项目。

如果在选择待打开的项目时已设置 **All Files (全部文件)** 选项，则不启用转换器并且 **Convert Project (转换项目)** 对话框打开。在对话框中，您需要通过选择一个选项显式触发项目的转换。

- **Convert to the current format (转换至当前格式)：** 从选择列表中选择您要使用的转换器 (转换应用)。转换后，无法以旧版本打开该项目。
- **Create a new project and add a specific device (创建一个新项目并添加特定设备)：** (未实现)

TwinCAT 2.x PLC 控制项目选项

在 TwinCAT 2.x PLC 控制项目选项中设置项目目录路径并且在 **Project Information (项目信息)** 对话框中采用项目信息。

案例 2)

如果库集成在“转换映射”尚未存储在库选项中的项目中，则会出现 **Converting a library reference (转换库引用)** 对话框，您可在该对话框中定义如何转换该参考：

- **Convert and install the library (转换并安装库)**：如果选择此选项，所引用的库将转换为新格式并在项目中保持引用。它自动安装在 **Other (其他)** 类别的库存储库中并继续使用。如果库中没有安装所需的项目信息（标题、版本），将提示您在 **Enter Project Information (输入项目信息)** 对话框中将其输入。
- **Use the following library, which is already installed (使用以下已安装的库)**：如果您选择了这些选项，则所引用的库将被已安装在本地系统上的另一个库所替代。使用 **Select (选择)** 按钮打开 **Select... (选择...)** 对话框。您可在此选择其中一个已安装库的所需版本。这对应于在 **Library Properties (库属性)** 对话框中处理的版本配置。通常，星号（“*”）表示该项目中使用系统上可用库的最新版本。可用库列表的结构与 **Library Repository (库存储库)** 对话框中相同。您可以按照公司和类别对列表进行排序。
- **Ignore the library (忽视库)。The reference will not appear in the converted project (引用将不出现在转换后的项目中)**：如果您启用该选项，则库引用将被删除。之后，转换后的项目不再包含库。
- **Use this mapping in future if this library is present (如果该库存在，则在未来使用该映射)**：如果您启用此选项，则当引用相应库时，在此对话框中所作出的设置将立刻应用于未来的项目转换。

在转换后的项目中，库引用在解决方案资源管理器的全局库管理器中定义。转换库引用后，如上文所述，可通过 **Open Project (打开项目)** 对话框继续项目转换。

关于库管理的一般信息，参见 PLC 文档中的“使用库”部分。

案例 3)

当您打开引用了一个未在 TwinCAT 2.x PLC 控制转换器选项中定义“转换映射”的设备的 TwinCAT 2. x PLC 控制项目时，**Device Conversion (设备转换)** 对话框打开，您可以在其中指定是否以及如何将旧设备引用替换为更新的设备引用。显示最初使用的设备。选择以下选项之一：

- **Use the following already installed device (使用以下已安装的设备)**：点击 **Select (选择)** 按钮打开 **Select target system (选择目标系统)** 对话框，您可在该对话框中选择当前安装在系统上的一个设备。之后，该设备被插入到转换项目（非旧项目）的 **Solution Explorer (解决方案资源管理器)** 中。选择 **Select a target system...(选择目标系统...)** 选项以选择所列出的一个设备。可用设备列表的结构与 **Device Repository (设备存储库)** 对话框中相同。您可以按照制造商或类别对列表进行排序。
- **Ignore the device (忽视设备)。No application-specific objects will be available (没有应用特定对象可用)**：如果启用该选项，在新项目的 **Solution Explorer (解决方案资源管理器)** 中不会创建设备条目，即设备在转换过程中被忽视，并且未提供任务配置等应用特定对象。
- **Save this assignment for future reference (保存此任务供未来引用)**：如果选择该选项，则该对话框的所有设置，即所显示的设备“转换映射”被保存在 TwinCAT 2.x PLC 控制转换器选项中并且应用于未来的转换。

4. 项目未正确终止并启用“自动保存”。

如果在 **Load and Save (加载并保存)** 选项中启用了 **Auto Save (自动保存)** 功能，并且 TwinCAT 3 PLC 在项目最后一次修改后在未保存的情况下未定期终止，**Auto Save Backup (自动保存备份)** 对话框打开以处理备份副本。

5. 项目为只读。

如果要打开的项目为只读状态，将询问您是以写保护模式打开项目还是要解锁项目。

6. 该库安装在库存储库中并从中检索。

如果尝试打开安装在库存储库中的库项目，将显示一条错误消息。您无法使用此路径打开库项目。通过 **OK (确认)** 关闭对话框后，项目名称出现在用户界面的标题栏中。名称后的星号（“*”）表示该项目自上次保存以来已被修改。

另请参见：

- PLC 文档：打开 TwinCAT 3 PLC 项目 [▶ 51]
- PLC 文档：打开 TwinCAT 2 PLC 项目 [▶ 51]

17.1.2 命令：项目...（创建新的 TwinCAT 项目）

符号：

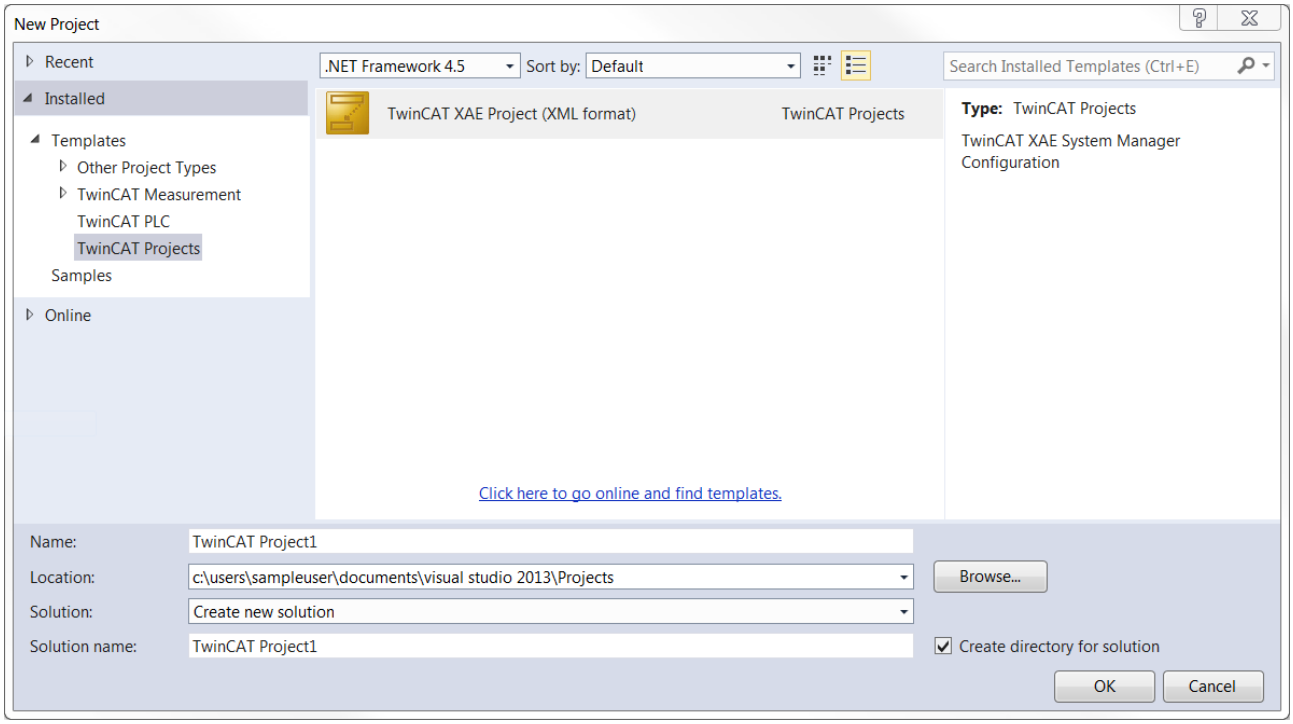
热键：[Ctrl] + [Shift] + [N]

功能：此命令打开用于创建新 TwinCAT 项目文件的 **New Project（新项目）**对话框。

调用：菜单 **File > New（文件 > 新建）**

新项目对话框

根据所选模板，您将获得自动具有一定数量对象的项目。



类别

最近	显示最近使用的项目模板。
已安装 > 模板	显示 TwinCAT 项目模板： • 其他项目类型 • TwinCAT Measurement • TwinCAT PLC • TwinCAT 项目
在线	不相关

模板

类别：其他项目类型	
Visual Studio 解决方案	空白 Visual Studio 解决方案
类别：TwinCAT Measurement	
波特图	波特图
Scope	Scope YT Project (Scope YT 项目) Scope YT NC Project (Scope YT NC 项目) Scope YT Project with Reporting (带报告的 Scope YT 项目) Scope XY Project (Scope XY 项目) Scope XY Project with Reporting (带报告的 Scope XY 项目)
类别：TwinCAT PLC	
	TwinCAT PLC 项目
类别：TwinCAT 项目	
	TwinCAT XAE 项目

Name	待创建项目的名称。根据模板显示默认名称。数字补充确保文件名在文件系统中的唯一性。 您可以根据操作系统的文件路径约定更改文件名。名称中不允许有点符号。 TwinCAT 自动添加与所选模板匹配的文件扩展名。
位置	新项目文件的位置。 **Browse (浏览)...** 按钮打开用于浏览文件系统的对话框。 组合框字段显示之前输入路径的历史。
解决方案	• 创建新的解决方案 • 添加 • 在新实例中创建
创建解决方案目录	☑ 解决方案目录已创建。
解决方案名称	解决方案的名称。默认情况下自动采用 TwinCAT 项目名。
确认	TwinCAT 打开新项目。

另请参见:

- PLC 文档: 您的第 1 个 TwinCAT 3 PLC 项目 [▶ 23]
- PLC 文档: 创建标准项目 [▶ 48]

17.1.3 命令项目/解决方案（打开项目/解决方案）

符号:

热键: [Ctrl] + [Shift] + [O]

功能: 此命令可打开用于打开文件的默认对话框。您可以在此处浏览 TwinCAT 项目文件的文件系统并在开发系统中打开它。

调用: 菜单 **File** > **Open**（文件 > 打开）

打开项目对话框

文件类型	用于过滤文件类型的选择列表 • 可以打开所有支持格式的文件。
选项	• 添加（例如，该选项仅用于将测量项目添加至解决方案。不要使用该选项将多个 TwinCAT 项目添加至解决方案。） • 关闭解决方案
打开	TwinCAT 打开所选项目文件。必要时，首先进行转换。

TwinCAT *.tszip 项目存档

***.tszip 的内容**	*.tszip 存档文件夹包含要存档的 TwinCAT 项目。
创建命令	使用以下命令可以创建 tszip 存档: 命令将 <TwinCAT project name> 另存为存档…… [▶ 988]
关于 PLC 项目的说明	如果 TwinCAT 项目包含 1 个或多个 PLC 项目，则在这些 PLC 项目的存档文件夹中存储的文件和文件夹将取决于相关项目的 PLC 项目设置。 设置选项卡 [▶ 859]

另请参见:

- PLC 文档: 您的第 1 个 TwinCAT 3 PLC 项目 [▶ 23]
- PLC 文档: 创建标准项目 [▶ 48]

17.1.4 命令：从目标打开项目

功能：此命令加载来自目标系统的项目。

调用：菜单 **File > Open（文件 > 打开）**

要求：必须配置目标系统的网络路径。

执行此命令将打开网络上的所有设备概览。从该概览中选择目标系统。**Select Folder for new Solution（选择新解决方案的文件夹）**对话框打开。

17.1.5 命令：新项目...（添加新的 TwinCAT 项目）

功能：此命令打开用于在解决方案中创建其他 TwinCAT 项目文件的 **New Project（新项目）**对话框。

调用：菜单 **File > Add（文件 > 添加）**

要求：打开一个 TwinCAT 项目。

例如，仅使用此命令将 Measurement 项目添加到解决方案中。不要使用此命令将多个 TwinCAT 项目添加至一个解决方案。TwinCAT 目前不支持此功能。

17.1.6 命令：现有项目...（添加现有 TwinCAT 项目）

Function（功能）:此命令打开用于将 TwinCAT 项目文件添加至解决方案的 **Add Existing Item（添加现有项）**对话框。

Call（调用）:菜单 **File > Add（文件 > 添加）**

要求：打开 TwinCAT 项目。

例如，仅使用此命令将 Measurement 项目添加到解决方案中。不要使用此命令将多个 TwinCAT 项目添加至一个解决方案。TwinCAT 目前不支持此功能。

17.1.7 命令：最近项目和解决方案

功能：此命令打开最近使用的项目列表，您可以从中选择一个项目打开。

调用：菜单 **File（文件）**

另请参见：

- PLC 文档：创建并配置项目 [▶ 48]

17.1.8 命令：保存全部

符号： 

功能：此命令保存所有 TwinCAT 项目的对象。

调用：File（文件）菜单，标准工具栏选项

17.1.9 命令：保存

符号： 

热键：[Ctrl] + [S]

功能：此命令保存解决方案、TwinCAT 项目、TwinCAT PLC 项目或当前名称下所选的 PLC 对象（主、GVL ...）。

调用：File（文件）菜单，标准工具栏选项

要求：在 Solution Explorer（解决方案资源管理器）中选择待保存的解决方案、TwinCAT 项目对象、PLC 项目对象（<PLC project name> 项目）或 PLC 对象。

保存对象

此命令以当前名称保存对象。如果对象自上次保存后发生变更，则对象的“磁盘”图标为红色，打开对象的编辑器标题栏中的名称用星号标记（"*"）。

另请参见：

- 命令：将 <TwinCAT project name> 另存为 [▶ 807]

17.1.10 命令将 <Solution name> 另存为

功能：此命令可打开用于保存文件的默认对话框。解决方案可保存在目标路径下。默认选择文件类型 UTF-8 解决方案文件（* .sln）。

调用：菜单 File（文件）

要求：在 Solution Explorer（解决方案资源管理器）中选择解决方案。

17.1.11 命令：将 <TwinCAT project name> 另存为

功能：此命令打开用于保存文件的默认对话框。项目可保存在目标路径和文件类型下。默认选择文件类型 TwinCAT XAE 项目 (*.tsproj)。

调用：菜单 File（文件）

要求：在 Solution Explorer（解决方案资源管理器）中选择 TwinCAT 项目对象。

注意：例如，在此保存操作过程中，仅在不同位置创建 *.tsproj 文件。所引用项目和其中包含的对象不会存储在这个新位置（例如集成在 TwinCAT 3 项目中的 PLC 项目及其对象）。

另请参见：

- 命令：保存 [▶ 806]
- PLC 文档：

17.1.12 命令将 <PLC project name> 另存为

功能：此命令可打开用于指定 PLC 项目文件的目标目录的对话框。PLC 项目对象和 .plcproj 文件保存在所选目录中。

调用：PLC 项目对象上下文菜单

要求：在 Solution Explorer（解决方案资源管理器）中选择 PLC 项目对象（<PLC project name>）。

17.1.13 命令创建反汇编文件

功能：该命令可从当前项目中创建 <project name>.asm 反汇编文件，并将它保存在项目文件夹中的文件目录中。

调用：PLC 菜单 > 创建反汇编文件

17.1.14 命令：通过电子邮件发送...

符号： 

功能：此命令启动系统中设置的电子邮件程序并打开一个新电子邮件，该邮件以所选项目的存档文件作为附件。

调用：File（文件）菜单，上下文菜单

17.1.15 命令：关闭解决方案

符号：

功能：此命令关闭当前打开的项目。TwinCAT 保持打开。

调用：File（文件）菜单或在项目仍打开的情况下打开一个新的/不同的项目时隐藏。

如果项目包含未保存的更改，则会出现是否应保存项目的问询。

如果尚未显示保存项目，则会出现确认是否要删除该项目文件的问询。

另请参见：

- PLC 文档：创建和配置 PLC 项目 [▶ 48]

17.1.16 命令：关闭

功能：此命令关闭打开的编辑器。

调用：菜单 File（文件）

要求：待关闭的编辑器处于活动状态，或在 PLC 项目树中选择对象。

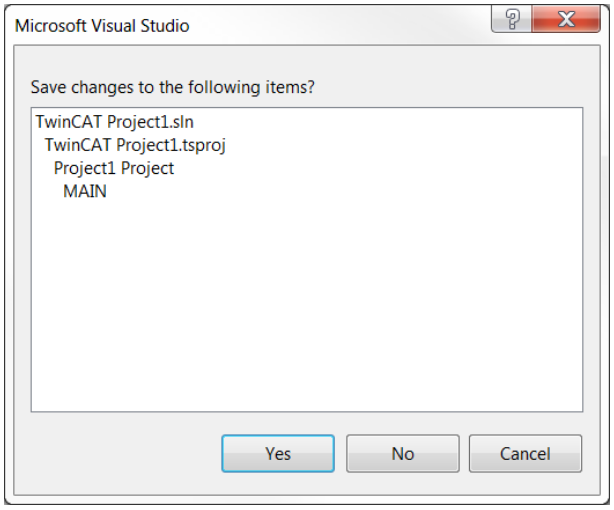
17.1.17 命令：退出

符号：

热键：[Alt] + [F4]

功能：此命令停止编程系统。如果当前打开一个自上次保存后已被更改的项目，则出现是否要保存该项目的问询对话框。

调用：菜单 File（文件）



17.1.18 命令：页面设置...

符号：

功能：此命令打开用于配置项目内容打印版本布局的 **Page settings（页面设置）**对话框。

调用：菜单 **File（文件）**

要求：编辑器窗口处于活动状态。

另请参见：

- 命令：打印 [▶ 809]

17.1.19 命令：打印

符号： 

功能：此命令打开用于打印文档的标准 Windows 对话框。

调用：菜单 **File（文件）**

要求：编辑器窗口处于活动状态。

17.2 编辑

17.2.1 标准命令

TwinCAT 提供以下标准命令：

- 撤销

   ，热键：[Ctrl] + [Z]
- 重做：

   ，热键：[Ctrl] + [Y]
- 剪切

   ，热键：[Ctrl] + [X]
- 复制：

   ，热键：[Ctrl] + [C]
- 粘贴：

   ，热键：[Ctrl] + [C]
- 删除：

   ，热键：[Del]
- 全选：热键：[Ctrl] + [A]

Call: Edit（调用：编辑）菜单，PLC 项目树上下文菜单，编辑器窗口上下文菜单

并非所有编辑器都支持 **Paste（粘贴）**命令，或者此命令的应用受到限制。在图形编辑器中，仅当粘贴操作创建正确的结构时才支持该命令。

在 PLC 项目树中，该命令指当前所选对象。可以作出多个选择。

17.2.2 命令删除

热键：[Del]

功能：此命令可从解决方案中删除所选的 PLC 对象。对象留在项目目录中。

调用：PLC 对象上下文菜单

17.2.3 命令全选

热键：[Ctrl + A]

功能：此命令可选择整个内容。

调用：Edit（编辑）菜单，编辑器窗口上下文菜单

17.2.4 命令：输入助手

符号： 

热键：[F2]

功能：此命令打开 Input Assistant（输入助手）对话框，根据当前光标位置为编程元素的选择提供支持。

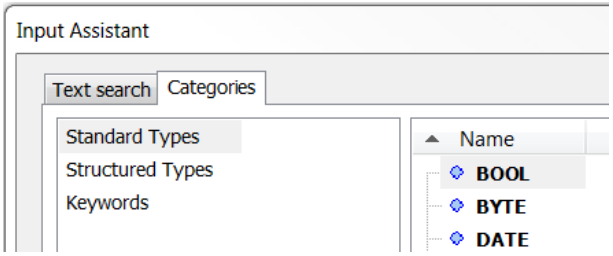
调用：Edit（编辑）菜单，编辑器窗口上下文菜单

要求：POU 在编辑器中打开，光标位于程序行中。

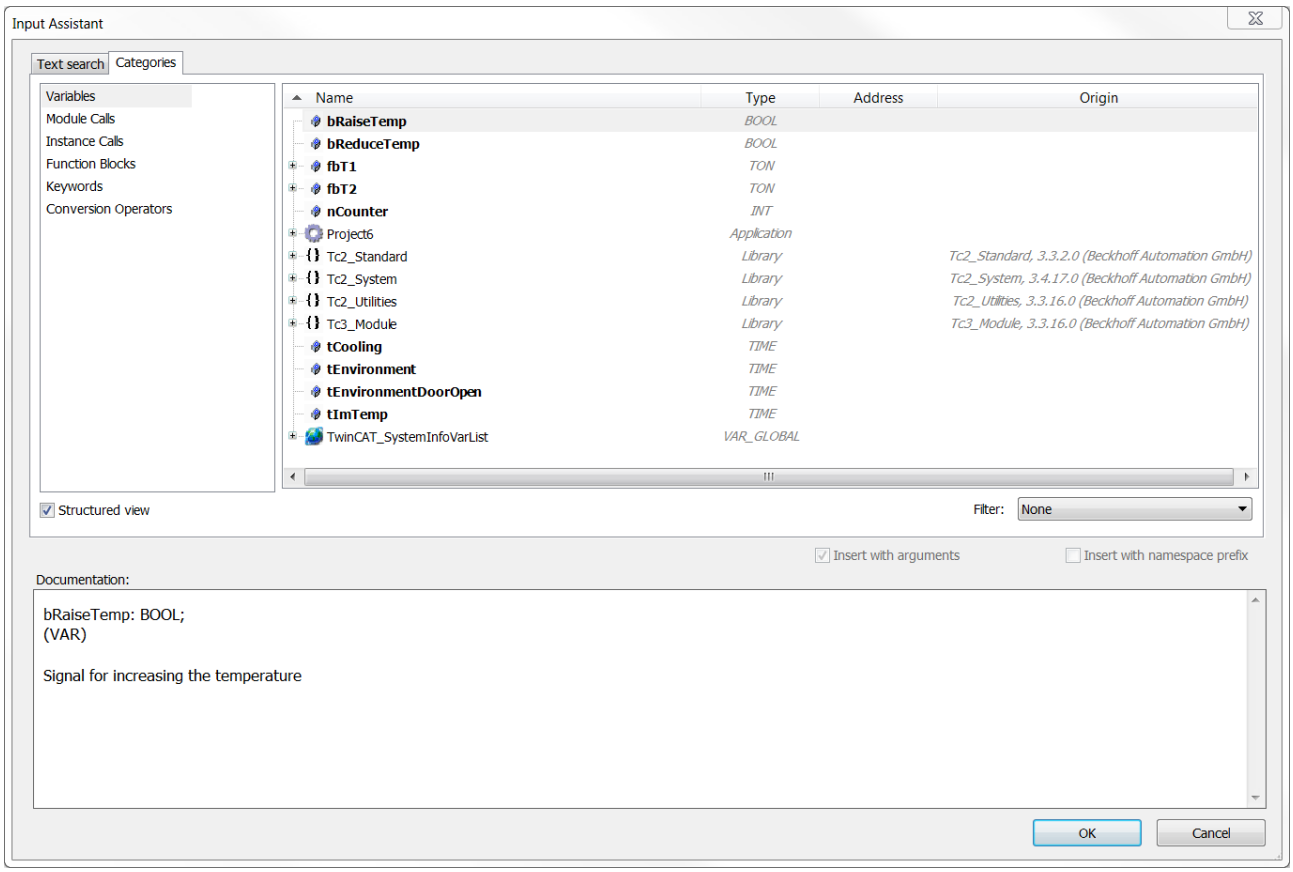
输入助手对话框 - 类别选项卡

对话框提供所有编程元素，您可以将其添加到编辑器中的当前光标位置。这些元素根据类别排序。此外，您还可以在 Variables（变量）类别中为范围设置筛选器，例如，局部变量、全局变量或常量。

编辑器的声明部分中的 Input Assistant（输入助手）对话框的详细信息：



编辑器的实现部分中的 Input Assistant（输入助手）对话框：



结构化视图	☑：元素以树状结构显示。通过右键点击子菜单中的列标题，您可以隐藏或显示类型列、地址列和原点列。 ☐：元素以扁平结构显示。
筛选器	您可以在下拉列表框中设置附加的变量类型筛选器。
显式文档	☑：显示所选项目的描述。
带参数插入	☑：TwinCAT 在光标位置插入带函数等参数的元素以及这些参数。 示例：如果您插入包含输入变量 fb1_in 和输出变量 fb1_out 的功能块 fb1，“带参数”，则在编辑器中显示如下：fb1(fb1_in:= , fb1_out=>)
带命名空间前缀插入	☑：TwinCAT 插入带命名空间前缀的所选元素。对于库功能块，如果库属性指定命名空间为必填项，您将无法使用复选框。

输入助手对话框 – 文本搜索选项卡

您可以使用该选项卡搜索特定对象。当您在搜索框中键入一个或多个字符时，结果窗口将列出名称包含此搜索字符串的所有对象的名称。双击所需对象以将其插入到当前光标位置的编辑器。

过滤器	将搜索限制为特定的变量类别。

另请参见：

- PLC 文档：使用输入向导 [▶ 123]

17.2.5 命令：自动声明

功能：此命令可打开支持变量声明的 Auto Declare（自动声明）对话框。

调用：Edit（编辑）菜单；编辑器窗口上下文菜单

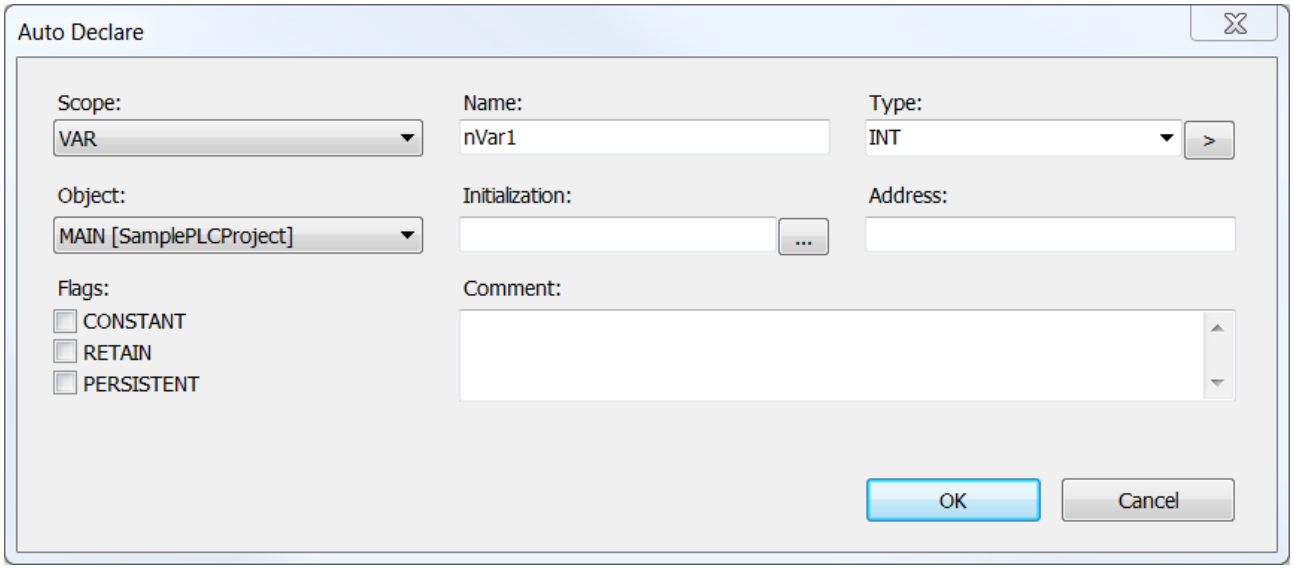
要求： POU 在编辑器中打开，光标位于程序行中。

即使光标位于 POU 实现部分中包含未声明变量名称的行，自动声明功能也将打开 **Auto Declare**（自动声明）对话框。为此，您必须在 TwinCAT 选项（**Extras** > **Options** > **TwinCAT** > **PLC Programming Environment** > **Smart Coding**）（附加功能 > 选项 > TwinCAT > PLC 编程环境 > 智能编码）中激活选项 **Declare unknown variables automatically (AutoDeclare)**（自动声明未知变量（AutoDeclare））。

由于存在智能标签功能，因此，如果您将光标放在 ST 编辑器的实现部分中尚未声明的变量上，然后点击

，也会出现命令 **Auto Declare**（自动声明）（Build 4026 及以上版本）。

自动声明对话框



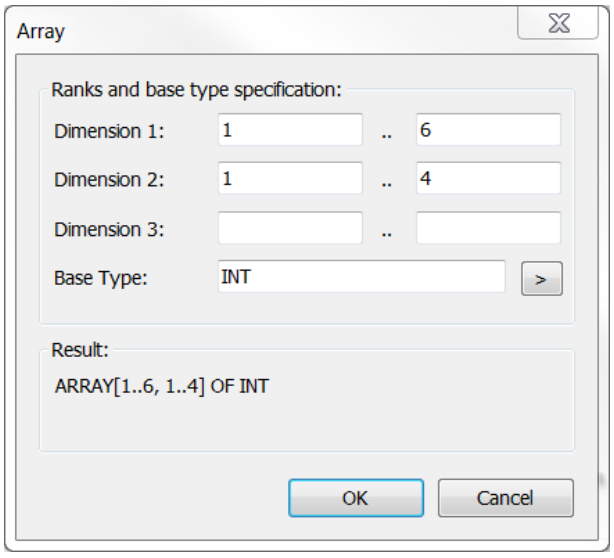
Scope	尚未声明的变量的有效范围。 示例：VAR（本地变量的默认设置）
Name	尚未声明的变量名 示例：bVar
数据类型	：列出标准数据类型。 ： • 输入助手：打开 **Input Assistant（输入助手）** 对话框 • 阵列辅助：打开 **Array（阵列）** 对话框 示例：BOOL
对象	声明新变量的对象。默认为您正在编辑的对象。 ：列出可以声明变量的对象。 如果所选范围没有可用对象，则显示条目 <Create object>。如果选择 <Create object> 条目，则用于创建合适对象的对话框 **Add object（添加对象）** 打开。
初始值	如果未输入初始化值，则变量会自动初始化。 ：打开 **Initialization（初始化）** 对话框。此方法有助于初始化结构变量。 示例：FALSE
地址	存储器地址（参见 PLC 文档：地址 [▶ 695]） 示例：%IX1.0
标记	属性关键词 • CONSTANT：常量关键词 • RETAIN：保留型变量关键词 • PERSISTENT：持久型变量关键词（比 RETAIN 更严格） 所选属性关键词被添加至变量声明。
注释	表格声明编辑器显示输入到注释列中的注释。在文本声明编辑器中，它显示在变量声明上方。 示例：New variable
通过重构应用更改	此选项用于以下有效性范围： • 输入变量（VAR_INPUT） • 输出变量（VAR_OUTPUT） • 输入和输出变量（VAR_IN_OUT） 如果在 TwinCAT 选项（**Tools > Options > TwinCAT > PLC Environment > Refactoring（工具 > 选项 > TwinCAT > PLC 环境 > 重构）** 中启用自动声明选项 **On renaming variables（重命名变量）** 和 **On adding or removing variables, or on changing the scope（添加或删除变量或更改范围）**，此选项自动启用（参见对话框选项 - 重构 [▶ 909]）。 如果启用此选项，则在关闭对话框时尚未声明变量。相反，**Refactoring（重构）** 对话框将打开，可在其中进一步编辑更改。

确认	变量被声明并出现在声明中。
	示例：
	`VAR`
	`    // New variable`
	`    bVar: BOOL := FALSE;`
	`END_VAR`

另请参见：

- PLC 文档：声明变量 [▶ 60]

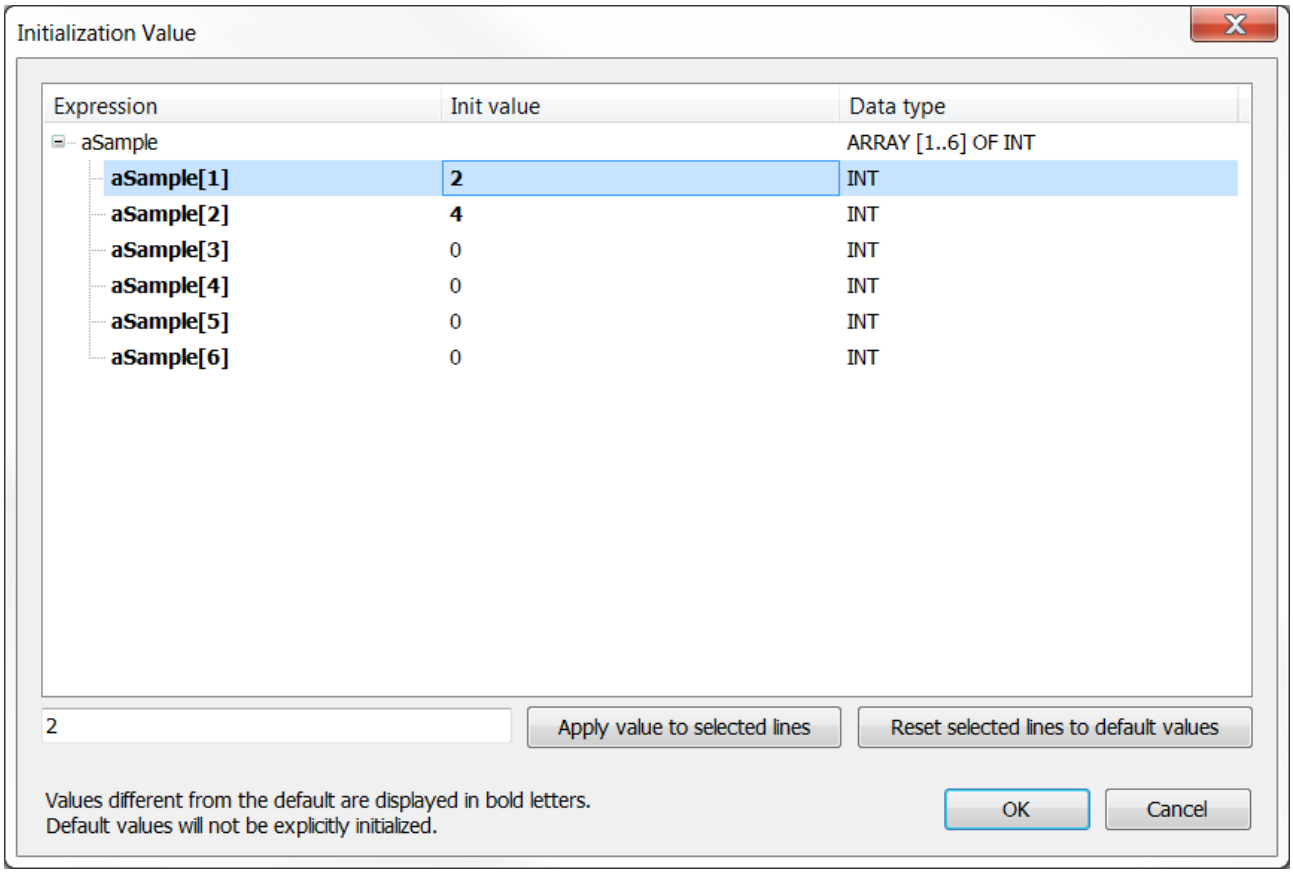
阵列对话框



等级和基本类型规格	通过输入阵列的上下边界和基本类型定义字段大小（维度）。如果点击 > 按钮，您可以直接输入或者使用 **Input Assistant（输入助手）** 或 **Array（阵列）** 对话框输入基本类型。
结果	显示定义的阵列。

TwinCAT 只在更改变量的初始值时才重新初始化变量。

初始化值对话框



变量列表，包括名称（表达式）、初始化值和数据类型。 已更改的初始化值以粗体显示。	
列表下的输入字段	输入所选变量的初始化值。
将值应用于所选行	根据输入字段的值更改所选行的初始化值。
将所选行重置为默认值	设置默认的初始化值。
确定	TwinCAT 采用 **Auto Declare**（自动声明）对话框中的初始化值。

如果要通过此对话框初始化的变量是带有扩展 FB_Init 方法的功能块实例，则在初始化值表格上方会显示另一个表格（请参见 PLC 文档：方法 FB_init、FB_reinit 和 FB_exit [▶ 786]）。本表格列出了附加的 FB_Init 参数。含义和操作与下方表格基本相同，区别如下：

- 必须为所有变量分配初始化值。否则无法选择 OK（确定）。
- 对于复杂数据类型（结构、数组），不会显示其中包含的组件（类型无法扩展）。在这种情况下，必须使用相应的变量对复杂类型进行初始化。

对于以这种方式配置的 FB_Init 参数，在 **Auto Declare**（自动声明）对话框中的初始化值后面会显示 1 个相应的符号。

另请参见：

- PLC 文档：使用输入向导 [▶ 123]

17.2.6 命令：添加至监视

符号：

功能：此命令将光标当前所在的变量添加到监视列表以进行在线监控。

调用：上下文菜单

要求：PLC 项目处于在线模式，且光标位于编辑器中的变量上。

该命令将变量插入当前打开的监视列表中。如果当前未打开监视列表，则该命令将变量插入监视列表 1 并打开其视图。

另请参见：

- PLC 文档：使用监视列表 [▶ 209]
- PLC 文档：监控值 [▶ 205]

17.2.7 命令：浏览调用树

符号：

功能：此命令打开 **Call Tree（调用树）**视图，其中显示功能块调用及其调用端。

调用：编辑器窗口上下文菜单

要求：在编辑器中打开功能块，且光标位于变量中。

17.2.8 命令：转至

功能：此命令将光标移动至指定代码行。

调用：菜单 **Edit（编辑）**

要求：POU 在编辑器中打开，光标位于程序行中。

命令打开对话框和一个 **Line number（行号）**输入字段。

17.2.9 命令转至定义

符号： 

热键：[F12]

功能：此命令可显示变量或功能的定义点。

PLC 编辑器

调用：编辑器窗口上下文菜单

要求：POU 在编辑器中打开，光标位于变量或功能上。

PLC 过程映像

调用：解决方案资源管理器上下文菜单

要求：扩展过程映像（项目实例），光标位于过程映像中已分配的变量处。

17.2.10 命令转至实例

符号： 

功能：此命令可在新窗口中打开功能块的实例。

调用：编辑器窗口上下文菜单

要求：PLC 项目处于在线模式。POU 在编辑器中打开，光标位于功能块的实例上。

该命令不适用于临时实例或编译库中的实例。

17.2.11 命令转至实现

符号： 

功能：此命令可在新窗口中打开方法的在线视图。

调用：编辑器窗口上下文菜单

要求：PLC 项目处于在线模式。POU 在编辑器中打开，光标位于方法调用上。

TC3.1 Build 4026 及以上可用

17.2.12 命令转至引用

符号： 

功能：此命令可在在线模式中打开当前聚焦指针所引用变量的声明位置。

调用：编辑器窗口上下文菜单

要求：PLC 项目处于在线模式。POU 在编辑器中打开，光标位于指针上。引用的变量位于静态内存中。

如果指针没有精确地指向变量的开头，则在切换到变量声明时会输出相应的消息。

TC3.1 Build 4026 及以上可用

17.2.13 命令查找所有引用

热键：[Shift+F12]

功能：此命令可显示 Cross Reference List（交叉引用列表）视图中所用变量的所有位置。

调用：上下文菜单

要求：POU 在编辑器中打开，并且光标位于变量上，或者 Cross Reference List（交叉引用列表）视图打开，并且在 Name（名称）字段中指定 1 个变量。

另请参见：

- PLC 文档：查找使用交叉引用列表的位置 [▶ 145]

17.2.14 命令：导航至

功能：此命令打开用于选择待打开特定元素的对话框。

调用：菜单 Edit（编辑）

17.2.15 命令：大写

功能：此命令将所选代码中的所有小写字母转换为大写字母。

调用：菜单 Edit > Advanced（编辑 > 高级）

要求：POU 在编辑器中打开，并选择代码。

17.2.16 命令：小写

功能：此命令将所选代码中的所有大写字母转换为小写字母。

调用：菜单 Edit > Advanced（编辑 > 高级）

要求：POU 在编辑器中打开，并选择代码。

17.2.17 命令：查看空白处

符号： 

功能：此命令可显示空格和选项卡的控制字符。

调用：菜单 Edit > Advanced（编辑 > 高级）

要求：POU 在编辑器中打开。

TwinCAT 用点显示空间，用箭头显示制表符。

17.2.18 命令注释选择

热键：[Ctrl+K] + [Ctrl+C]

功能：此命令可注释掉所选的代码段。该代码段会被排除在编译之外，对程序的执行没有任何影响。

调用：菜单 Edit > Advanced（编辑 > 高级）

要求：功能块在编辑器中打开，并选择代码。

该命令可用于为程序或程序段的文档创建注释，或从编译中临时排除代码段。命令 命令取消注释选择 [▶ 818] 可用于取消注释，以便将注释掉的代码段重新集成到程序执行中。

17.2.19 命令取消注释选择

热键：[Ctrl+K] + [Ctrl+U]

功能：此命令可取消注释，并将注释掉的代码段重新集成回到程序执行中。

调用：菜单 **Edit** > **Advanced**（编辑 > 高级）

要求：功能块在编辑器中打开，并选择先前使用命令 命令注释选择 [▶ 817] 注释掉的代码。

17.2.20 命令：快速查找

符号： 

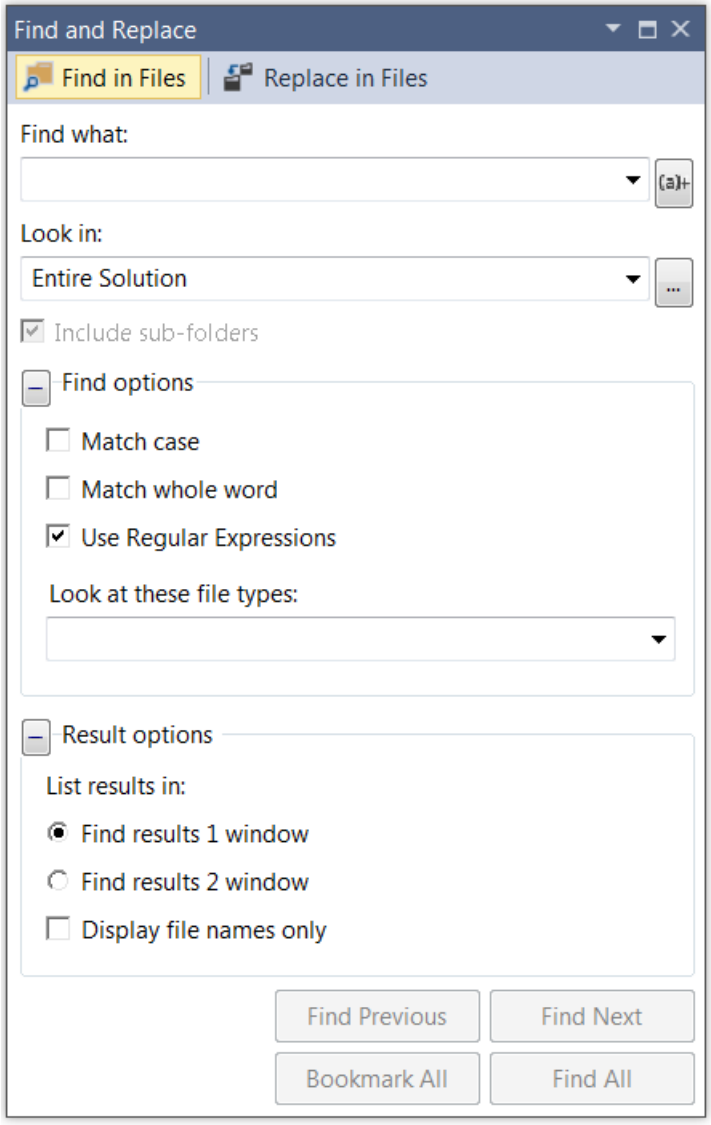
热键：[Ctrl] + [F]

功能：此命令扫描项目或项目的一部分以搜索特定字符串。

调用：菜单 **Edit** > **Find and Replace（编辑 > 查找和替换）**

命令打开 **Find and Replace（查找和替换）**对话框（**Find in Files（在文件中查找）**按钮启用），可在此对话框中输入搜索字符串和搜索选项。

查找并替换对话框


Find and Replace
Find in Files
Replace in Files
Find what:
Look in:
Entire Solution
Include sub-folders
Find options
Match case
Match whole word
Use Regular Expressions
Look at these file types:
Result options
List results in:
Find results 1 window
Find results 2 window
Display file names only
Find Previous
Find Next
Bookmark All
Find All

在文件中替换	切换至 Find and Replace（查找和替换）对话框（Replace in Files（在文件中替换）按钮启用）
查找内容	搜索字符串。
查看	 ：带被搜索对象的选择列表： 整个解决方案：扫描所有项目对象中的所有可编辑位置。 当前项目： 所有打开的文件：扫描窗口中当前打开的所有编辑器。 当前文档：仅搜索光标当前所在的编辑器。  ：打开可以更精确定义待搜索对象的对话框。
区分大小写	 ：该搜索区分大小写。
全字匹配	 ：仅查找与搜索字符串完全匹配的字符串。
查看这些文件类型	用于选择文件类型的下拉列表
使用正则表达式	启用  按钮，以帮助您输入正则表达式。 该功能不支持 PLC 编辑器！
仅显示文件名	 ：仅显示文件名。
查找下一个	开始搜索。下一个搜索结果显示在相应编辑器中的对应位置。
查找全部	在消息窗口中显示所有搜索结果。显示对象和搜索结果的具体位置。 • (Decl)：对象的声明部分 • (Impl)：对象的实现部分 双击列表条目以显示编辑器中的搜索结果。

另请参见：

- 命令：快速替换 [▶ 820]
- PLC 文档：在整个项目中查找和替换 [▶ 148]

17.2.21 命令：快速替换

符号：

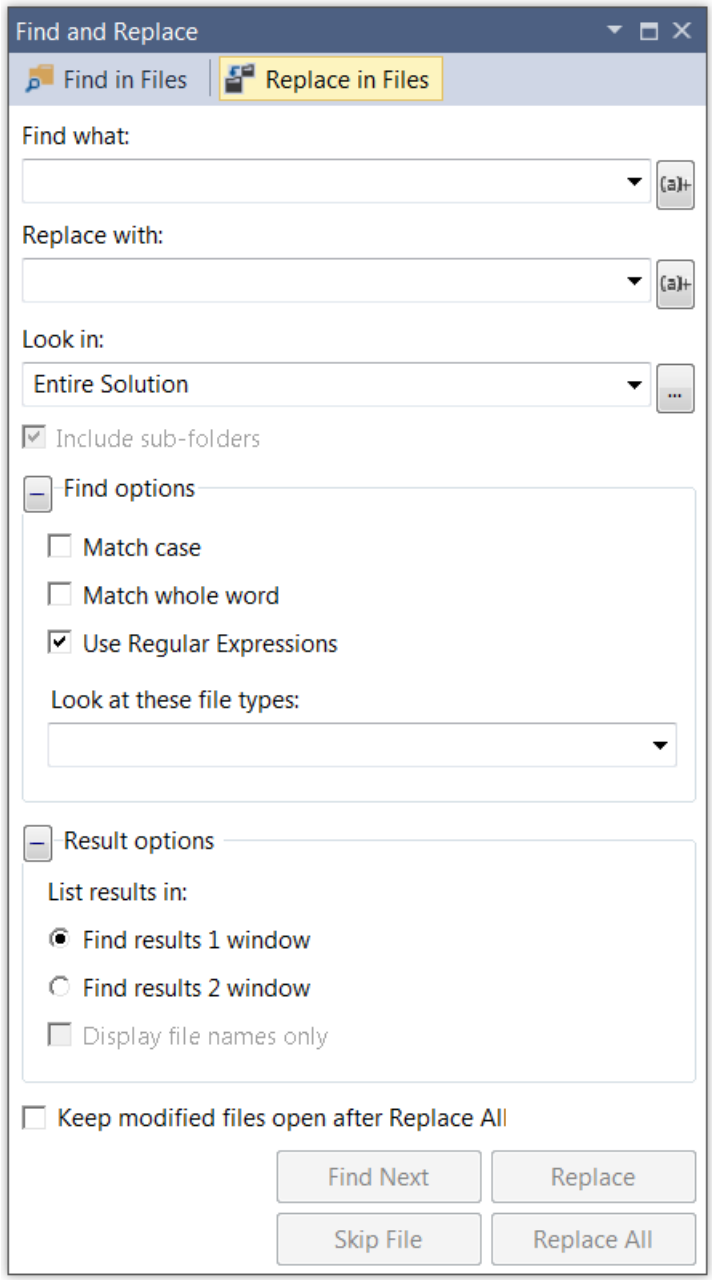
热键：[Ctrl] + [H]

功能：此命令扫描项目或项目的一部分以搜索特定字符串并将其替换。

调用：菜单 Edit > Find and Replace（编辑 > 查找和替换）

命令打开 Find and Replace（查找和替换）对话框（Replace in Files（在文件中替换）按钮启用），可在此对话框中输入待替换的字符串和新字符串以及搜索选项。

查找并替换对话框



除了“Find”（查找）对话框中的选项之外，还可以进行以下设置：

替换为	新字符串的输入字段。
替换	查找到的下一个字符串在编辑器中高亮显示并替换（逐步替换）。
全部替换	立即替换查找到的所有字符串并且不在编辑器中显示。
在全部替换后将修改后的文件保持在打开状态	查找到的对象的编辑器保持打开状态。

另请参见：

- 命令：快速查找 [▶ 818]
- PLC 文档：在整个项目中查找和替换 [▶ 148]

17.2.22 命令：切换写入模式

热键：[Insert]

功能：此命令启动覆盖模式或插入模式。

调用：双击状态和信息栏中的 [INS] 或 [OVR]

要求：编辑器窗口处于活动状态。

如果覆盖模式激活，则在输入新字符时会覆盖光标后的字符。如果插入模式激活，则插入字符时保留光标后的现有字符。

17.2.23 命令：重命名

功能：此命令可对 Solution Explorer（解决方案资源管理器）中的 PLC 对象进行重命名。

调用：PLC 对象上下文菜单

17.2.24 命令：编辑对象（脱机）

功能：此命令在其编辑器中脱机打开对象。

调用：Project（项目）菜单，上下文菜单

要求：PLC 项目处于在线模式。在 PLC 项目树中选择对象。

这意味着您也可以在在线模式中编辑对象。之后，通过 Online Change（在线更改）或 Download（下载）将更改发送到控制器。

另请参见：

- 命令在线更改 [▶ 887]
- 命令：下载 [▶ 887]

17.2.25 命令：重命名 '<variable>'

功能：此命令打开 Rename（重命名）对话框以对对象或变量进行重命名。

调用：上下文菜单 PLC object (PLC 对象)，上下文菜单 Editor window > Refactoring (编辑器窗口 > 重构)

要求：在 PLC 项目树中选择对象，或者光标位于编程对象声明部分变量识别符前或之上。

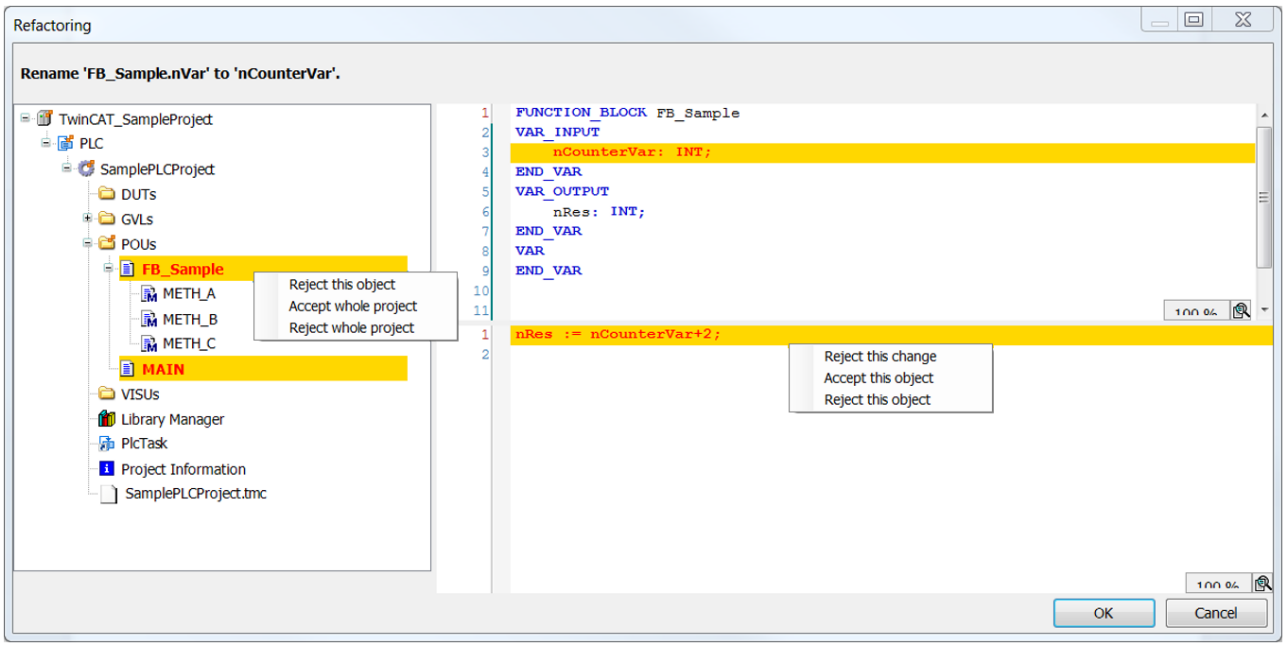
您可以重命名：

- 变量
- POU
- GVL
- 方法
- 属性

重命名对话框

当前名称	对象或变量名
新名称	新名称的输入字段。 如果输入的名称已存在，TwinCAT 将在该输入字段正下方报告这一情况。
确认	如果已在 New Name（新名称）中输入有效名称，则可激活。 打开 Refactoring（重构）对话框。 两个窗口中的各对象和位置均以颜色标记。 在两个窗口中可以为每个位置指定一个操作。可在上下文菜单中使用不同命令。

重构对话框



该对话框显示项目内的所有使用位置。各对象和位置以颜色高亮显示。	
对话框左部	带各自对象的项目导航树。
对话框右部	显示对象中出现当前名称的位置。
在两个窗口中可以为每个位置指定一个操作。可在上下文菜单中使用以下命令。	
拒绝此更改	放弃对话框右侧的各个更改。
接受此对象	接受受影响对象中的所有更改
拒绝此对象	放弃受影响对象中的所有更改
接受整个项目	接受项目中的所有更改
拒绝整个项目	放弃项目中的所有更改
TwinCAT 通过黄色背景显示接受的更改，通过灰色背景显示放弃的更改。	

另请参见：

- PLC 文档：重构 [▶ 148]

17.2.26 命令：添加 '<variable>'

符号：

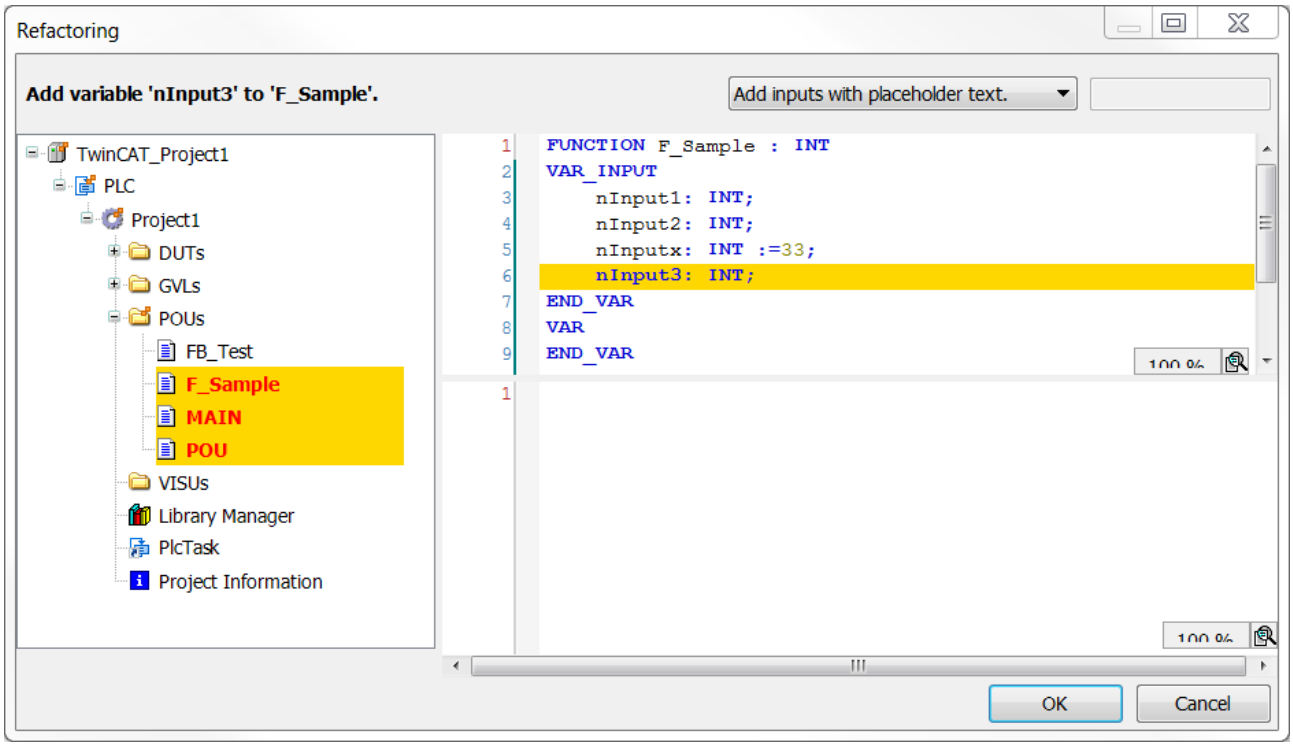
功能：此命令在 POU 中声明一个新变量并且在 POU 使用点自动更新。

调用：上下文菜单 Editor window > Refactoring（编辑器窗口 > 重构）

要求：重点在于 POU 的声明部分。

此命令打开用于声明变量的默认对话框。使用 **OK（确认）**关闭声明对话框后，将出现由两部分组成的 **Refactoring（重构）**对话框。

重构对话框



对话框左部	项目导航树。 使用 POU 的功能块颜色：红色字体，黄色背景。 点击 POU 对象后，详细视图在对话框右部打开。
对话框右部	声明部分和在声明中添加变量的 POU 实现部分。 更改点的颜色：新添加的声明使用蓝色字体，黄色背景。

在决定要在哪些点应用哪些更改之前，请从右上角的下拉列表中选择所需选项：

添加带占位符文本的输入	默认占位符文本：_REFACTOR_；可编辑 此处定义的占位符文本出现在实现代码中新添加变量的使用点。用于查找受影响的位置。
添加带以下值的输入	新变量的初始化值

可在对话框左部和右部的变更点上下文菜单使用接受或拒绝更改的命令。另请参见 命令：重命名 '<variable>' [▶ 822] 的描述。

示例：

1. 通过重构，功能 F_Sample 被分配一个带初始值“1”的新输入变量“nInput3”。此更改具有以下效果：

之前：

```
F_Sample(nVarA + nVarB, 3, TRUE);
F_Sample(nInput1:= nVarA + nVarB, nInput2 :=3 , nInputx := TRUE);
```

之后：

```
F_Sample(nVarA + nVarB, 3, TRUE, nInput3 := 1);
F_Sample(nInput1:= nVarA + nVarB, nInput2 :=3 , nInputx := TRUE, nInput3 := 1);
```

2. 通过重构，功能 F_Sample 被分配一个带占位符文本“_REFACTOR_”的新输入变量“nInput3”。

之前：

```
F_Sample(nInput1 := nVarA + nVarB, nInput2 := 3, nInputx := TRUE);
F_Sample(nVarA + nVarB, 3, TRUE);
```

之后：

```
F_Sample(nInput1 := nVarA + nVarB, nInput2 := 3, nInputx := TRUE, nInput3 := _REFACTOR_);
F_Sample(nVarA + nVarB, 3, TRUE, nInput3 := _REFACTOR_);
```

另请参见：

- PLC 文档：重构 [▶ 148]
- PLC 文档：自动声明对话框 [▶ 812]

17.2.27 命令：删除 '<variable>'

符号： 

功能：此命令删除 POU 和所有 POU 使用点中的输入或输出变量。

调用：上下文菜单 Editor window > Refactoring（编辑器窗口 > 重构）

要求：光标位于 POU 声明部分中待删除的变量识别符上。

此命令首先打开一个对话框，显示所要删除的信息。确认后，**Refactoring（重构）**对话框出现。关于对话框的描述，可参见“命令：添加 '<variable>' [▶ 823]”部分。

如果接受 **Refactoring（重构）**对话框中的更改，则删除受影响 POU 的使用点处的相应输入或输出参数。

在 CFC 中，仅断开被删除的输入或输出与功能块的连接。图中保留了输入或输出本身。

ST 中的样本：

使用 **Refactoring（重构）**删除 POU 中的输入变量“nInput4”。在各使用点进行自动调节：

删除前：

```
F_Sample(nInput1 := nVarA + nVarB, nInput2 := 3, nInput4 := 1, nInput5 := TRUE);
F_Sample(nVarA + nVarB, 3, 1, TRUE);
```

删除后：

```
F_Sample(nInput1 := nVarA + nVarB, nInput2 := 3, nInput5 := TRUE);
F_Sample(nVarA + nVarB, 3, TRUE);
```

另请参见：

- PLC 文档：重构 [▶ 148]

17.2.28 命令：重新排序变量

符号： 

功能：在声明编辑器中，您可使用此命令更改当前重点范围 VAR_INPUT、VAR_OUTPUT 或 VAR_IN_OUT 中的变量序列。

调用：声明编辑器中当前重点范围的上下文菜单

要求：重点是声明上述范围之一，并在其中声明多个变量。

此命令打开 **Rearrange（重新排列）**对话框，其中带有当前重点范围的所有声明列表。您可以使用鼠标将所选声明向上或向下拖动到其他位置。

另请参见：

- PLC 文档：在声明中重新排列变量 [▶ 150]

17.3 视图

17.3.1 命令：打开对象

符号：

功能：此命令打开编辑器中的对象。

调用：View（视图）菜单，上下文菜单 PLC 对象，双击 PLC 对象

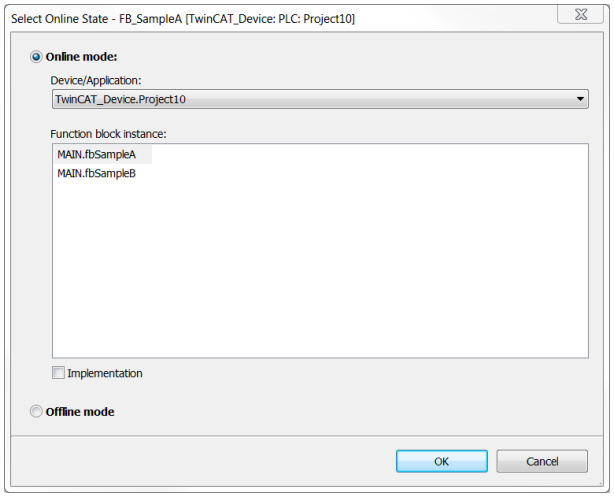
要求：在 PLC 项目树中选择一个对象。

在在线模式中，**Select Online State（选择在线状态）**对话框打开，您可在其中选择查看应打开对象的视图。明确选择对象时，对话框不会打开。在此情况下，直接在在线模式中打开。

选择在线状态对话框

功能：此对话框确认如何在在线模式中打开未在脱机模式中打开的对象（功能块等）。您可以选择打开一个实例或对象本身的基本实现（可在在线或脱机模式中选择）。

要求：PLC 项目包含所选对象的多个实例。



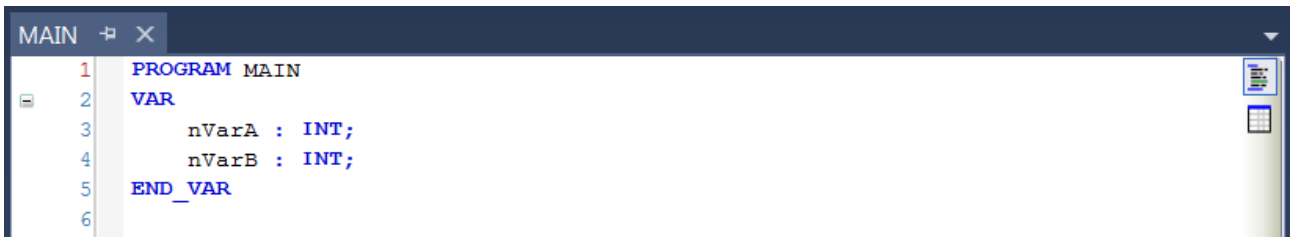
在线模式	启用该选项以在在线模式中获得视图。
设备/应用	显示被分配该对象的应用（项目）。
功能块实例	如果对象是功能块，则显示应用中当前使用的所有实例列表。
实现	选择该选项将显示功能块的基本实现，无论所选实例如何。该选项没有用于非实例化对象的功能。
脱机模式	启用该选项以在脱机模式中获得视图。

17.3.2 命令：文本视图

符号：

功能：此命令在文本视图中打开声明编辑器。

调用：按钮位于编辑器右侧边缘



另请参见：

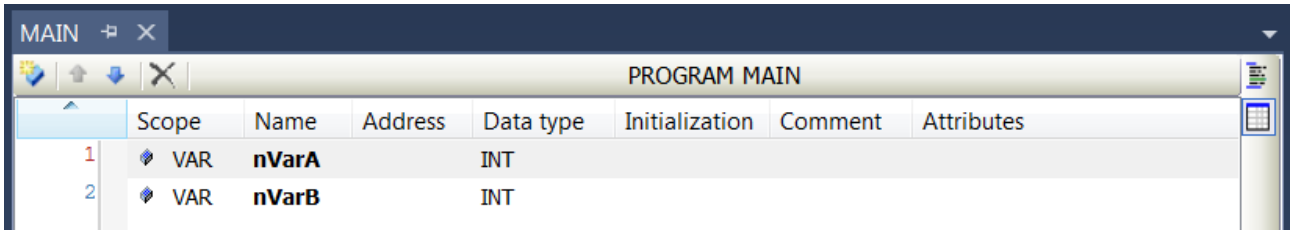
- PLC 文档：使用声明编辑器 [▶ 64]

17.3.3 命令：表格视图

符号：

功能：此命令在表格视图中打开声明编辑器。

调用：按钮位于编辑器右侧边缘



另请参见：

- PLC 文档：使用声明编辑器 [▶ 64]

17.3.4 命令：全屏

符号：

热键：[Ctrl] + [Shift] + [F12]

功能：此命令将 TwinCAT 显示切换为全屏模式。

调用：View（视图）菜单

启用该命令时，TwinCAT 用户界面的主窗口将以全屏模式显示。您可以通过再次禁用该命令恢复到预设大小。

17.3.5 命令：工具栏

功能：此命令打开一个用于选择所显示工具栏的菜单。

调用：View（视图）菜单，工具栏区域上下文菜单

在打开的菜单中，选择要显示或隐藏的工具栏。该命令是一个选项，即，当工具栏显示时，它会出现在前面带有一个勾选标记的菜单中。

另请参见：

- TC3 用户界面文档：自定义工具栏

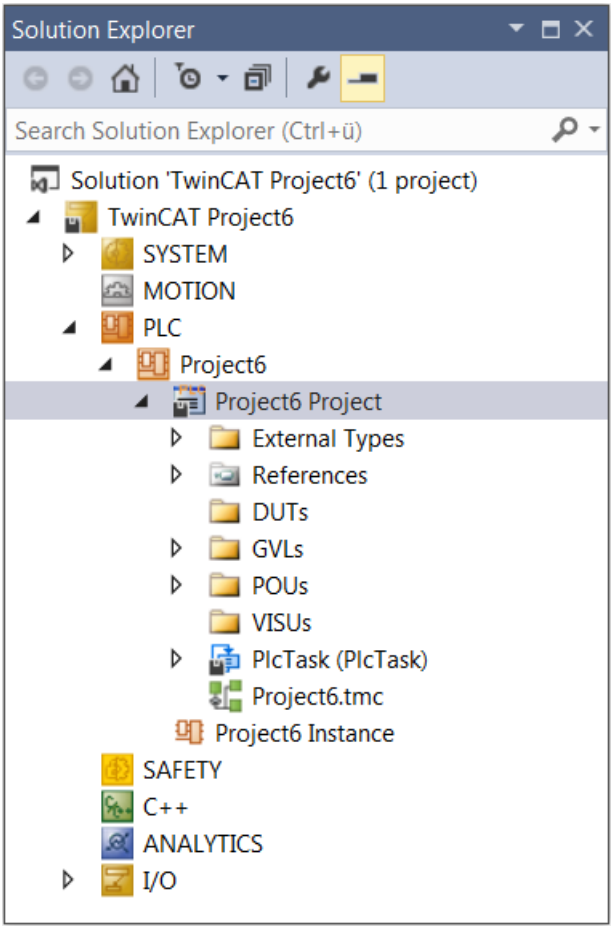
17.3.6 命令：解决方案资源管理器

符号：

功能：此命令打开 Solution Explorer（解决方案资源管理器）视图。

解决方案资源管理器视图

Solution Explorer（解决方案资源管理器）视图以结构化形式显示 TwinCAT 3 项目与相应的项目元素。在此视图中，您可以打开用于编辑和配置的对象。



17.3.7 命令：属性窗口

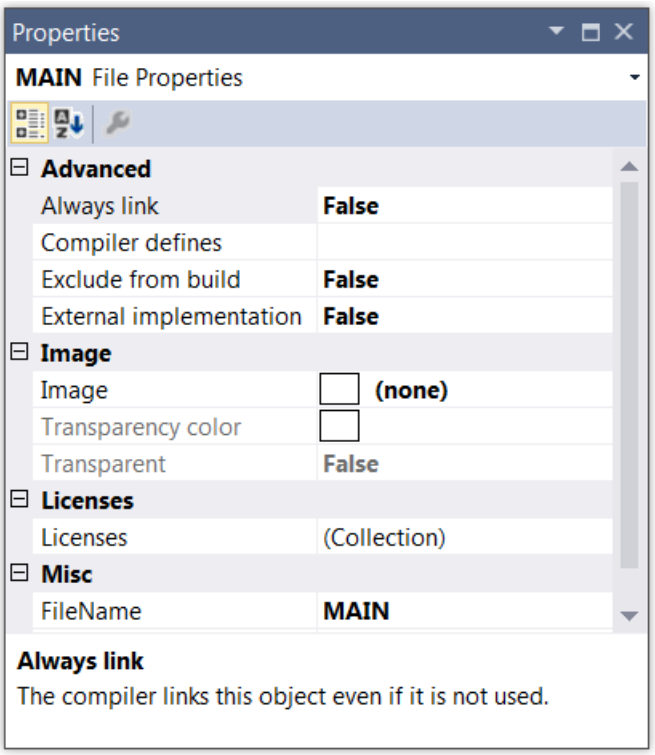
符号：

功能：此命令打开 Properties（属性）视图。

调用：View（视图）菜单

属性视图

Properties（属性）视图显示解决方案资源管理器中当前所选对象的属性。元素属性默认为在表中按类别排序。点击类别前的加号或减号以显示或隐藏相关参数。用鼠标点击参数的值字段将启用输入模式，可在该模式中编辑值或属性。您可以对属性视图进行筛选或排序。



您还可以从 PLC 项目树中一个对象的上下文菜单中调出属性窗口。关于命令以及各种对象属性的描述，请参见命令属性（对象） [▶ 836] 部分。

17.3.8 命令：工具箱

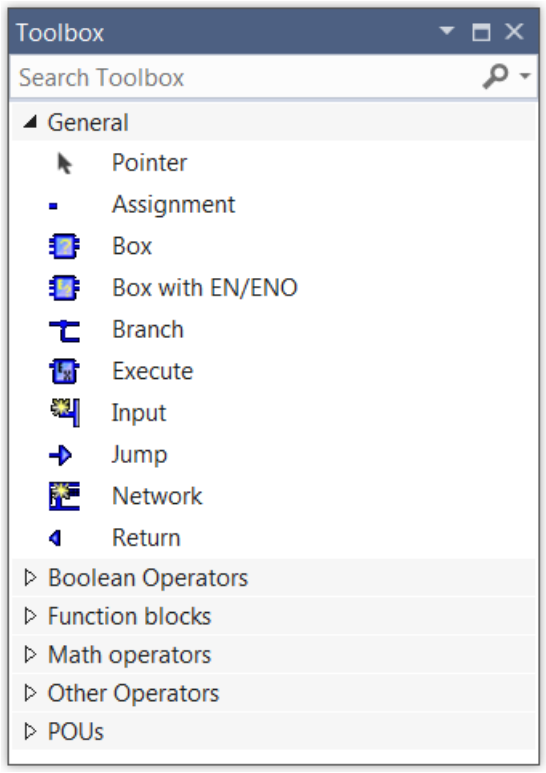
符号：

功能：此命令打开 Toolbox（工具箱）视图。

调用：View（视图）菜单

工具箱视图

Toolbox（工具箱）视图显示当前处于活动状态的编辑器的现有工具。该视图默认可使用图形编辑器或可视化工具。它包含图形编辑元素，您可将这些元素拖动到编辑器窗口。



17.3.9 命令：错误列表

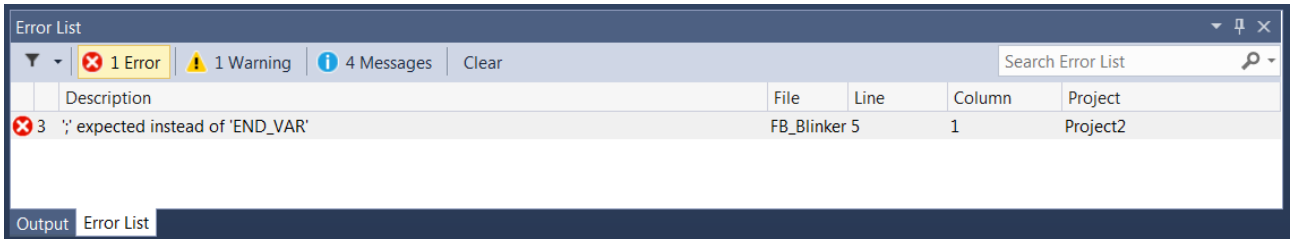
符号：

功能：此命令打开 Error List（**错误列表**）视图。

调用：View（**视图**）菜单

错误列表视图

Error List（**错误列表**）视图显示与语法检查、编译过程（编译错误、代码大小）、导入过程或库管理器相关的错误、警告和消息。消息以表格形式显示。



过滤器	使用用于选择代码文件组的下拉菜单： 打开文档 当前项目 当前文档
消息类别 •  ：错误 •  ：警告 •  ：信息	单击消息类别符号以显示或隐藏消息。在每个符号旁边，TwinCAT 显示已出现的消息数量。
清除	清除消息显示
	清除错误列表的过滤器设置
严重性级别（消息类别）	消息文本与原因对象以及项目内的位置。 双击表中的消息条目以转至源文本位置。
代码	
描述	
项目	
文件	
Line（线条）	

上下文菜单中的命令

清除	清除消息显示
显示列	添加附加列，以更详细地描述错误
 复制	复制所选错误消息
下一个错误	选择下一条消息。显示下一个错误的源文本位置。
上一个错误	选择上一条消息。显示上一个错误的源文本位置。

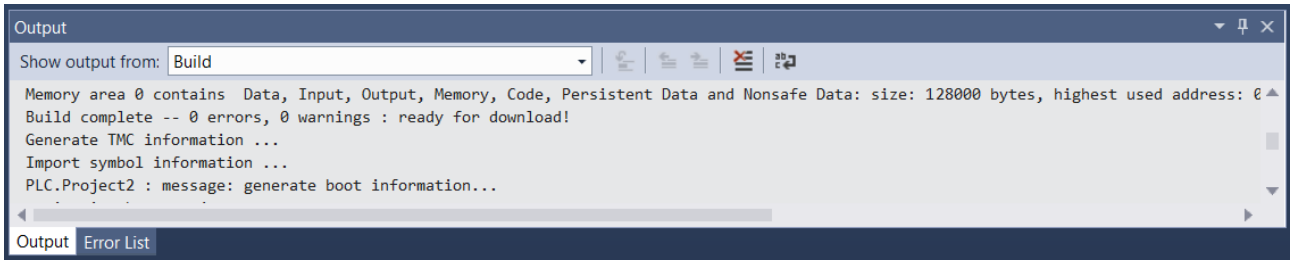
17.3.10 命令：输出

符号： 

功能：此命令打开 **Output（输出）**视图。

调用：View（视图）菜单

输出视图



消息类别	消息按组件或功能分类，并且可以在选择对话框中使用。可以通过选择一个类别过滤消息显示。
 在代码中查找消息	显示消息的源文本位置。 要求：高亮显示消息。
 转至上一条消息	已选择上一条消息。
 转至下一条消息	已选择下一条消息。
 删除全部	删除全部消息
 切换换行符	启用或禁用换行符

上下文菜单中的命令

 复制	复制消息文本
 删除全部	删除全部消息
 转至位置	显示消息的源文本位置。 要求：高亮显示消息。
 转至下一个位置	已选择下一条消息。
 转至上一个位置	已选择上一条消息。

17.4 项目

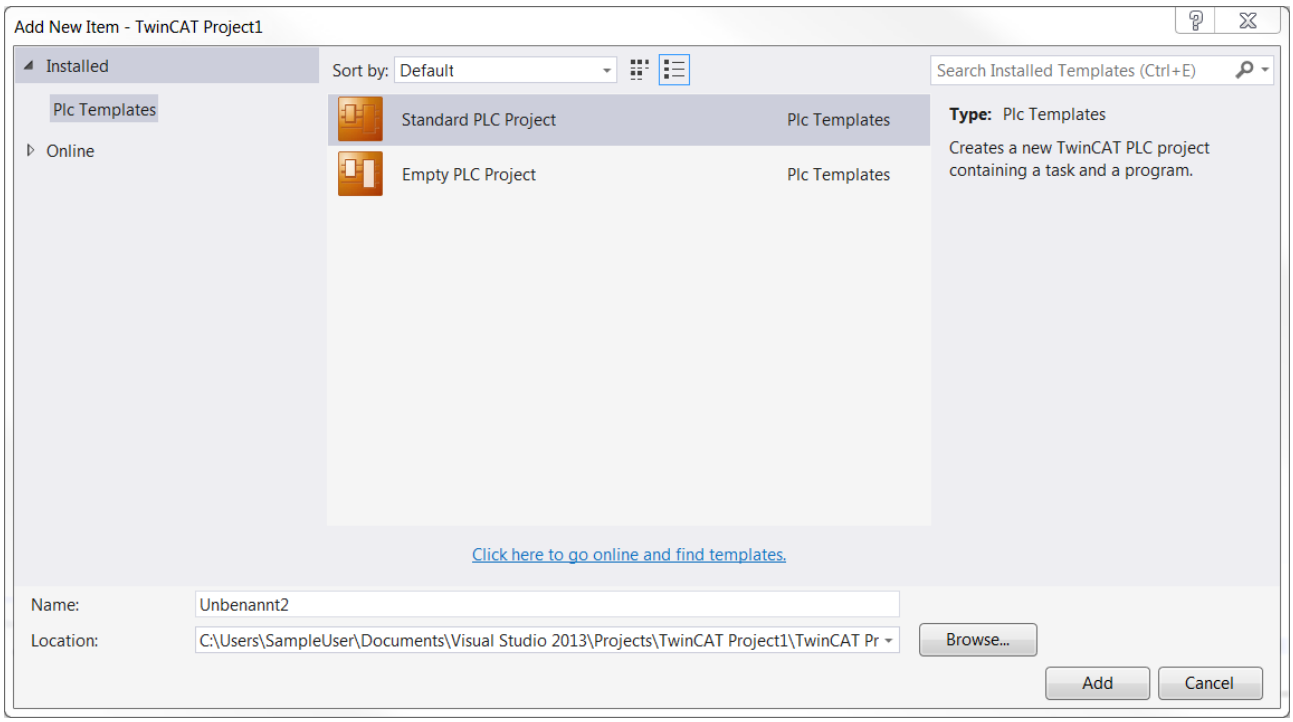
17.4.1 命令添加新项目（项目）

符号： 

功能：此命令可打开 **Add New Item** （添加新项目）对话框，您可以通过该对话框创建新的 PLC 项目文件。（此命令仅在选择 PLC 节点时可用。）

调用：Project（项目）菜单或 **Solution Explorer**（解决方案资源管理器）中的 PLC 对象上下文菜单

要求：在 TwinCAT 项目树中选择 PLC 节点。



Plc 模板	选择所列模板之一。该模板确定 PLC 项目文件的基本配置。默认可用以下模板： • 标准 PLC 项目：创建 1 个新的 TwinCAT PLC 项目（*.project）。该项目由向导支持，包含 1 个库管理器、1 个 POU“MAIN”程序和 1 个引用的任务。 • 空 PLC 项目：创建库项目的“空白”TwinCAT PLC 项目（*.library）。该项目未包含对象或设备。
名称	在此处定义新项目的名称。默认名称取决于所选模板（通常是“Unnamed<n>”）并且包含序号，以确保项目名称在文件系统中的唯一性。您可以根据本地操作系统的文件路径约定来更改默认名称。可添加文件扩展名（例如，.project）。所选模板默认自动添加相应的扩展名。
位置	指定新项目文件的位置。默认路径取决于所选模板。您可以使用 **Browse...**（浏览……）按钮打开默认浏览器并指定路径，或者您可以使用相应的下拉列表选择先前输入的路径。
添加	点击 **Add**（添加）创建基于设置的新项目。如果光标位于错误符号上，则工具提示会提供关于如何继续的信息。如果另一个 PLC 项目已打开，则会打开 1 个对话框询问您是否要在打开新项目之前保存并关闭该项目。新项目名称显示在 TwinCAT XAE 边框窗口的标题栏中。名称后的星号（“*”）表示该项目自上次保存以来已被修改。

另请参见：

- PLC 文档：您的第 1 个 TwinCAT 3 PLC 项目 [▶ 23]
- PLC 文档：创建和配置 PLC 项目 [▶ 48]

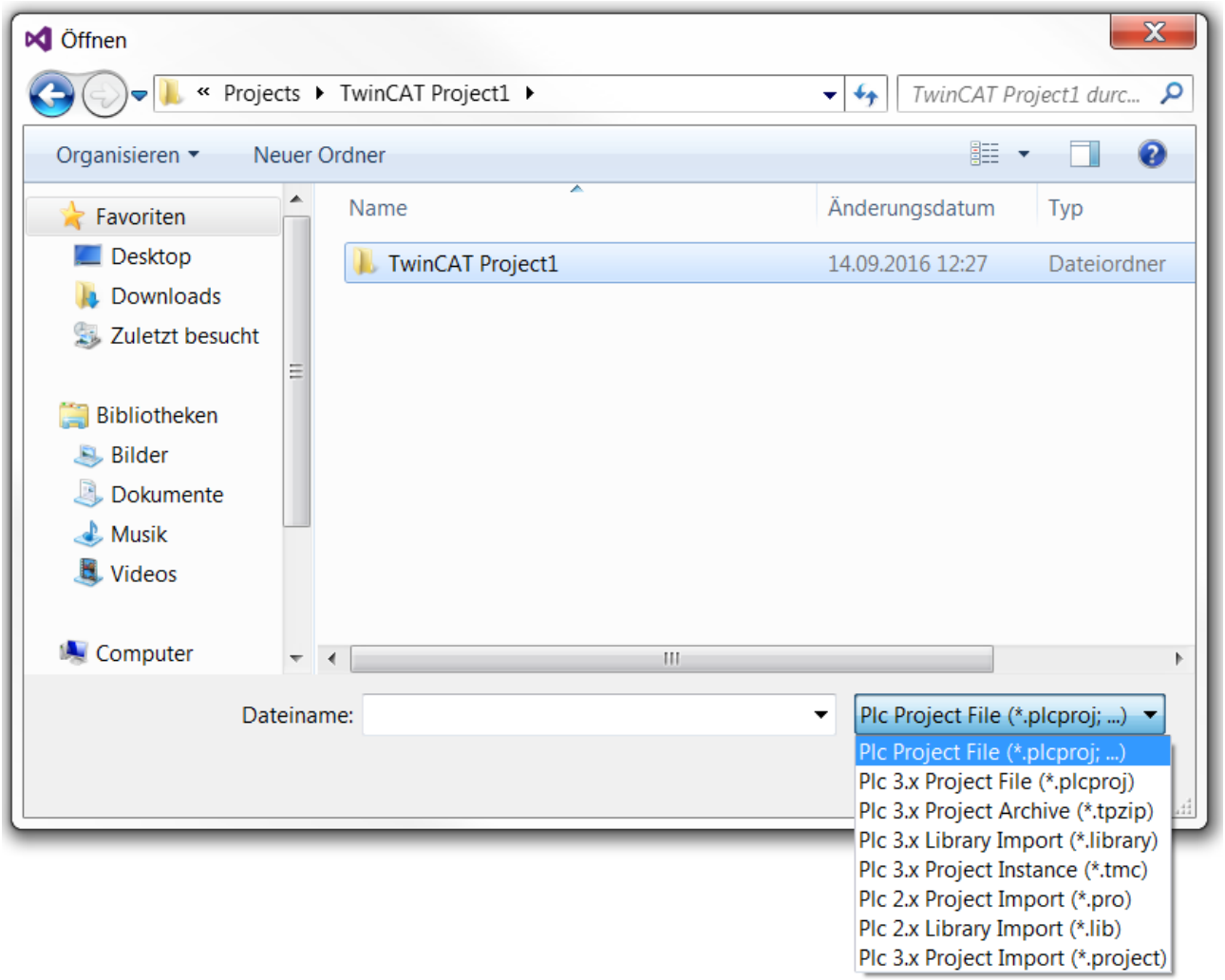
17.4.2 命令添加现有项目（项目）

符号： 

功能：此命令可打开标准浏览器对话框，该对话框可用来搜索 PLC 项目文件并在编程系统中打开它。如果安装了合适的转换器，则您可以打开不同格式的项目。

调用：Project（项目）菜单或 **Solution Explorer**（解决方案资源管理器）中的 PLC 对象上下文菜单

要求：在 TwinCAT 项目树中选择 PLC 节点。



文件类型	默认情况下，您可以将筛选器设置为以下 1 种文件类型： • PLC 3.x 项目文件（*.PLCproject）：TwinCAT 3 PLC 项目，带扩展名“.PLCproject” • PLC 3.x 项目存档（*.tpzip）：TwinCAT 3 PLC 项目存档，带扩展名“.tpzip” ◦ 另请参见：命令将 <PLC project name> 另存为存档…… [▶ 800] • PLC 3.x 库导入（*.library）：TwinCAT 3 PLC 库，带扩展名“.library” • PLC 2.x 项目文件（*.pro）：TwinCAT 2 PLC 项目，带扩展名“.pro” • PLC 2.x 导入库（*.lib）：TwinCAT 2 PLC 库，带扩展名“.lib” • PLC 3.x 项目导入（*.PLCproject）：PLC 项目，带扩展名“.project”
打开	所选项目打开或转换后打开。

*.tpzip PLC 项目存档

***.tpzip 的内容**	*.tpzip 存档文件夹包含要存档的 PLC 项目。
创建命令	使用以下命令可以创建 tpzip 存档： 命令将 <PLC project name> 另存为存档…… [▶ 800]
关于 PLC 项目的说明	在 PLC 项目的存档文件夹中存储的文件和文件夹取决于该 PLC 项目的 PLC 项目设置。 设置选项卡 [▶ 859]

打开 PLC 项目时可能出现的情况

当您打开项目时，可能会出现以下情况：

1. 另一个项目仍打开。 [▶ 835]
2. 项目以 TwinCAT 3 的旧版本保存。 [▶ 835]
3. 项目未以 TwinCAT 3 保存。 [▶ 835]
4. 项目未正确终止并启用“自动保存”。 [▶ 836]
5. 项目为只读。 [▶ 836]
6. 该库安装在库的存储库中并从中检索。 [▶ 836]

1. 另一个项目仍打开。

您会被询问是否保存并关闭其他项目。

2. 项目以 TwinCAT 3 的旧版本保存。

如果因为打开的项目以 TwinCAT 3 的旧版本保存而导致文件格式不同，则有两个选项：

- 如果无法以当前所用编程系统的格式保存项目，则必须对其进行更新以便继续处理项目。此时出现的表达式：**The changes you made... (您作出的更改...)** 指加载项目时各组件的内部任务。
- 如果仍将项目保存为之前的格式，则您可以决定是否更新或保留格式。如果您决定保留格式，则数据可能丢失。如果您决定更新格式，则无法再通过旧版本编程系统打开项目。

除了文件格式外，显式插入库的版本、可视化配置文件以及打开项目的编译器版本可能与当前编程系统安装的版本不同。

如果在当前编程系统上安装了更新的版本，则 **Project Environment (项目环境)** 对话框自动打开，您可在其中更新版本。如果此时未更新，则可在之后随时通过 **Options > Project Environment (选项 > 项目环境)** 对话框进行更新。

注意编译器版本

如果打开一个用旧版本编程系统创建的项目，并且在项目设置中设置了该项目的最新编译器版本，则当编译器版本的项目环境设置在新编程系统中设置为 **Do not update (不更新)** 时，将继续使用上一次在旧项目中使用的编译器版本（即不是新环境中的“当前”版本）。

3. 项目未以 TwinCAT 3 保存。

案例 1)

如果您在选择要打开的项目时设置了文件过滤器，并且有合适的转换器可用，则会自动使用转换器并将项目转换为当前格式。该转换特定于转换器实现。通常，系统会提示您定义对引用库或设备引用的处理。

TwinCAT 3 转换器

在导入期间，只有在转换器能够编译该项目而且没有错误时才能成功地根据 TwinCAT 3 语法调整 TwinCAT PLC 控制项目。

如果在选择待打开的项目时已设置 **All Files (全部文件)** 选项，则不启用转换器并且 **Convert Project (转换项目)** 对话框打开。在对话框中，您需要通过选择一个选项显式触发项目的转换。

- **Convert to the current format (转换至当前格式)：** 从选择列表中选择您要使用的转换器（转换应用）。转换后，无法以旧版本打开该项目。
- **Create a new project and add a specific device (创建一个新项目并添加特定设备)：**（未实现）

TwinCAT 2.x PLC 控制项目选项

在 TwinCAT 2.x PLC 控制项目选项中设置项目目录路径并且在 **Project Information (项目信息)** 对话框中采用项目信息。

案例 2)

如果库集成在“转换映射”尚未存储在库选项中的项目中，则会出现 **Converting a library reference (转换库引用)** 对话框，您可在该对话框中定义如何转换该参考：

- **Convert and install the library (转换并安装库)**：如果选择此选项，所引用的库将转换为新格式并在项目中保持引用。它自动安装在 **Other (其他)** 类别的库存储库中并继续使用。如果库中没有安装所需的项目信息（标题、版本），将提示您在 **Enter Project Information (输入项目信息)** 对话框中将其输入。
- **Use the following library, which is already installed (使用以下已安装的库)**：如果您选择了这些选项，则所引用的库将被已安装在本地系统上的另一个库所替代。使用 **Select (选择)** 按钮打开 **Select... (选择...)** 对话框。您可在此选择其中一个已安装库的所需版本。这对应于在 **Library Properties (库属性)** 对话框中处理的版本配置。通常，星号（“*”）表示该项目中使用系统上可用库的最新版本。可用库列表的结构与 **Library Repository (库存储库)** 对话框中相同。您可以按照公司和类别对列表进行排序。
- **Ignore the library (忽视库)。The reference will not appear in the converted project (引用将不出现在转换后的项目中)**：如果您启用该选项，则库引用将被删除。之后，转换后的项目不再包含库。
- **Use this mapping in future if this library is present (如果该库存在，则在未来使用该映射)**：如果您启用此选项，则当引用相应库时，在此对话框中所作出的设置将立刻应用于未来的项目转换。

在转换后的项目中，库引用在解决方案资源管理器的全局库管理器中定义。转换库引用后，如上文所述，可通过 **Open Project (打开项目)** 对话框继续项目转换。

关于库管理的一般信息，参见 PLC 文档中的“使用库”部分。

案例 3)

当您打开引用了一个未在 TwinCAT 2.x PLC 控制转换器选项中定义“转换映射”的设备的 TwinCAT 2. x PLC 控制项目时，**Device Conversion (设备转换)** 对话框打开，您可以在其中指定是否以及如何将旧设备引用替换为更新的设备引用。显示最初使用的设备。选择以下选项之一：

- **Use the following already installed device (使用以下已安装的设备)**：点击 **Select (选择)** 按钮打开 **Select target system (选择目标系统)** 对话框，您可在该对话框中选择当前安装在系统上的一个设备。之后，该设备被插入到转换项目（非旧项目）的 **Solution Explorer (解决方案资源管理器)** 中。选择 **Select a target system...(选择目标系统...)** 选项以选择所列出的一个设备。可用设备列表的结构与 **Device Repository (设备存储库)** 对话框中相同。您可以按照制造商或类别对列表进行排序。
- **Ignore the device (忽视设备)。No application-specific objects will be available (没有应用特定对象可用)**：如果启用该选项，在新项目的 **Solution Explorer (解决方案资源管理器)** 中不会创建设备条目，即设备在转换过程中被忽视，并且未提供任务配置等应用特定对象。
- **Save this assignment for future reference (保存此任务供未来引用)**：如果选择该选项，则该对话框的所有设置，即所显示的设备“转换映射”被保存在 TwinCAT 2.x PLC 控制转换器选项中并且应用于未来的转换。

4. 项目未正确终止并启用“自动保存”。

如果在 **Load and Save (加载并保存)** 选项中启用了 **Auto Save (自动保存)** 功能，并且 TwinCAT 3 PLC 在项目最后一次修改后在未保存的情况下未定期终止，**Auto Save Backup (自动保存备份)** 对话框打开以处理备份副本。

5. 项目为只读。

如果要打开的项目为只读状态，将询问您是以写保护模式打开项目还是要解锁项目。

6. 该库安装在库存储库中并从中检索。

如果尝试打开安装在库存储库中的库项目，将显示一条错误消息。您无法使用此路径打开库项目。通过 **OK (确认)** 关闭对话框后，项目名称出现在用户界面的标题栏中。名称后的星号（“*”）表示该项目自上次保存以来已被修改。

另请参见：

- PLC 文档：打开 TwinCAT 3 PLC 项目 [▶ 51]
- PLC 文档：打开 TwinCAT 2 PLC 项目 [▶ 51]

17.4.3 命令属性（对象）

功能：此命令可启用 **Properties**（属性）视图，该视图显示关于当前所选对象的一般信息。

调用： PLC 对象上下文菜单

要求： 在 PLC 项目树中选择 1 个对象。

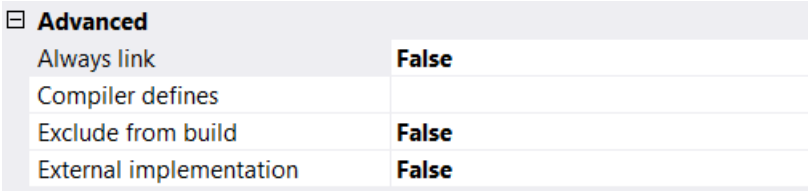
根据当前所选对象，显示以下属性领域：

- 高级 [▶ 837]（编译设置）
- 图像 [▶ 837]
- 授权 [▶ 838]
- 一般信息 [▶ 838]（对象名称、对象路径）
- SFC 设置 [▶ 839]（顺序功能图标志）
- CFC 设置 [▶ 840]（执行顺序模式）

在 可视化对象 [▶ 367] 下记录了特殊可视化属性，在 命令属性 [▶ 333] 下记录了特殊库和占位符属性。

高级

该区域显示对象编译设置。



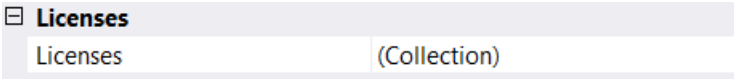
始终绑定	真：在编译器中选择对象，因此对象始终包含在编译信息中。因此，它始终会在 PLC 上进行编译和加载。如果对象位于应用程序下或通过同样位于应用程序下的库引用，则此选项将具有关联性。编译信息还用作可选符号配置变量的基础。 另外，也可以使用 {attribute 'linkalways'} [▶ 748] 编译指示来指示编译器始终包含 1 个对象。
编译器定义	在此处输入的编译器定义不会被评估。 如果您想要使用编译器定义，则在 PLC 项目属性中输入它们。 请参见： • 命令属性（PLC 项目） > 类别编译 [▶ 844] • PLC 文档：引用编程 > 编译指示 > 条件编译指示 [▶ 775]
从编译中排除	真：对象未包含在下一次编译运行中。
外部实现	（运行时系统中的后期绑定） 只有在特殊的系列情况中才能使用该功能。通常情况下，您可以忽略该选项。 真：在编译项目时，不会为该对象生成代码。在将项目加载到目标系统上之前，该对象不会被链接，前提是该对象存在于目标系统上（在 PLC 运行时系统或其他实时模块中）。

图像

在该区域中，您可以为对象分配一个图像，该图像显示在库管理器的图形视图和 FBD/LD/IL 编辑器的工具箱中。可以通过选择一种颜色实现图像的透明度，该颜色之后会显示为透明。如果您选择 **Transparency Color（透明色）** 选项，则可以使用其右侧的矩形按钮打开用于选择颜色的标准对话框。

授权

该部分包含对象的授权列表。



一般信息

在本部分中，您将会了解有关所选对象的一般信息。

FileName	文件/对象名称
FullPath	对象的存储路径/位置 （此时不可编辑）
版本	文件版本，值： • 1.1.0.1：当对象以 XML 格式保存时，使用此文件版本。 • 1.2.0.0：当对象以 Base64 格式保存时，使用此文件版本。 请注意，不能将文件版本为 1.2.0.0（或更高）的对象加载到低于 TC3.1.4024 的开发环境版本中！ （此时不可编辑，可通过存储格式间接配置，请参见**格式**属性）

文件版本 1.2.0.0（或更高）与 TwinCAT 3.1 < Build 4024 在工程上不兼容

请注意，不能将以文件版本 1.2.0.0（或更高）保存的对象加载到低于 TwinCAT 3.1.4024 的开发环境版本中！

由于在使用可选的 Base64 格式时，对象会自动以文件版本 1.2.0.0 保存，因此无法使用低于 TwinCAT 3.1.4024 的开发环境版本加载 Base64 格式的对象。

如果 1 个 PLC 项目包含文件版本为 1.1.0.1 的对象和文件版本为 1.2.0.0 的对象，则 1.1.0.1 对象将以低于 TwinCAT 3.1.4024 的开发环境版本加载。未加载文件版本为 1.2.0.0 的对象。

使用 XAE 版本 TwinCAT 3.1.4024 或更高版本，可以将以文件版本 1.2.0.0 保存的文件的文件版本重置为 1.1.0.1。

选项

在本部分中，您将会了解可为 PLC 对象配置的一些选项。

格式	**存储格式的单独设置选项：** 此时，可针对下列对象类型单独配置对象的存储格式。 存储格式，值： • XML：对象以 XML 格式保存。 ◦ 使用此存储格式的对象以文件版本 1.1.0.1 存储，请参见**版本**属性。 • Base64：对象以 Base64 格式保存。 ◦ 使用此存储格式的对象以文件版本 1.2.0.0 存储，请参见**版本**属性。请注意，不能将文件版本为 1.2.0.0（或更高）的对象加载到低于 TC3.1.4024 的开发环境版本中！ **Base64 相对于 XML 的优势：** 与 XML 相比，Base64 可实现压缩存储。因此，通过对这些对象的文件访问可以提高性能，例如，在加载、移动或复制对象时可以使用文件访问。 **Base64 的可用性：** 对于 Build 4024 及以上版本，下列 PLC 对象可选择使用 Base64 存储格式： • 以图形化实现语言对 POU 主体进行编程的 POU ◦ 顺序功能图（SFC） ◦ FBD/LD/IL（功能块图/梯形图/指令表） ◦ CFC（连续功能图和面向页面的 CFC） ◦ UML 类图和状态图 • 带有以图形化实现语言编程的子元素（例如，动作、方法）的 POU（有关图形语言的信息，请参见第 1 个关键点） • 可视化 • 可视化管理器 • 文本列表 • 配方管理器 • 图像池 **标准存储格式的设置选项：** 对于 PLC 项目，PLC 项目属性（高级类别 [▶ 853]）中的设置“Write object content as”（将对象内容写为）可用于定义上述对象类型的标准存储格式。
单独的 LineIds	值：真或假 • 真：该 POU 的行 ID 存储在 1 个单独的文件（LineIDs.dbg）中。 • 假：该 POU 的行 ID 存储在 POU 本身中。 （此时不可编辑，可在写入选项 [▶ 922]中配置）
排序	值：名称或 GUID 指定子对象（例如方法）在父对象中的存储顺序：按名称或 GUID 排序。 （此时不可编辑，可在写入选项 [▶ 922]中配置）
写入 ProductVersion	值：真或假 （此时不可编辑，可通过 PLC 项目属性的 高级类别 [▶ 853] 中的设置“在文件中写入产品版本”进行配置）。

SFC 设置

该区域显示用于编译和处理当前所选 SFC 对象隐式变量的当前设置。

SFC	
Use default SFC settings	True
SFC Build	
CalculateActiveTransitionOnly	False
SFC Flags	
SFCCurrentStep	Declare
SFCEnableLimit	Declare
SFCError	Declare
SFCErrorAnalyzation	Declare
SFCErrorAnalyzationTable	Declare
SFCErrorPOU	Declare
SFCErrorStep	Declare
SFCInit	Use
SFCPause	Declare
SFCQuitError	Declare
SFCReset	UseDeclare
SFCTip	Declare
SFCTipMode	Declare
SFCTrans	Declare

使用默认的 SFC 设置	真（默认）：通过此选项，在 PLC 项目属性中定义的默认值可被应用于当前所选对象并在对象的 **Properties**（属性）视图中显示。 假：此选项允许您配置特定于此 SFC 对象的有效 SFC 设置。
编译（“构建”）和变量（“标志”）	这些项目的含义对应于 PLC 项目属性中配置的 SFC 对象的默认设置。

CFC 设置

TC3.1 Build 4026 及以上可用

该区域设置所选 CFC 对象的执行顺序模式。在 CFC 编辑器中，您可以自由定位元素，从而定位网络。为了避免 CFC 编程块的执行顺序不确定，有 2 种模式可供选择。

CFC	
Explicit Execution Order	False

显式执行顺序	假（默认）：自动数据流模式 真：显式执行顺序模式

自动数据流模式

在这种模式下，根据数据流自动设置执行顺序，如果存在不明确的情况，则根据网络拓扑进行设置。编程块和输出端在内部编号。上层网络先于下层网络执行，左侧网络先于右侧网络执行。

优势：自动定义的执行顺序可以优化时间和周期。在开发过程中，您不需要任何有关内部管理的执行顺序的信息。

CFC 编辑器中的元素不带标记和编号。无法手动更改执行顺序。对于带有反馈的网络，您可以额外设置 1 个起点。

在菜单 **CFC > Execution order**（CFC > 执行顺序）中，以下命令在此模式下可用：

- 命令显示执行顺序 [▶ 941]
- 命令设置反馈开始 [▶ 941]

显式执行顺序模式

在这种模式下，您可以显式设置执行顺序。为此，在 CFC 编辑器中显示的元素带有标记和编号，并提供可令您确定顺序的菜单命令。

下列命令在菜单 **CFC > Execution order**（CFC > 执行顺序）中可用：

- 命令：移动到开始 [▶ 942]
- 命令：移动到结束 [▶ 942]
- 命令：向前移动一位 [▶ 943]
- 命令：向后移动一位 [▶ 943]
- 命令：设置执行次序 [▶ 943]
- 命令：按数据流排序 [▶ 944]
- 命令：按拓扑结构排序 [▶ 944]

对于 Build 4026 之前的版本，这是 CFC 编程块的通常行为。请注意，您必须自行调整执行顺序并自行判断后果和影响。为此，系统会不断显示执行顺序。

17.4.4 命令属性（PLC 项目）

符号：

功能：此命令可打开 1 个编辑器窗口，其中显示和定义项目的属性以及其他与项目相关的信息。

调用：PLC 项目对象（<PLC project name> 项目）的上下文菜单或 **Project**（项目）菜单

要求：打开 1 个项目。

TwinCAT 将 PLC 项目属性直接存储在 PLC 项目中。

PLC 项目属性范围

注意：不同项目属性之间的范围不同！
一些属性仅影响您当前配置属性的 PLC 项目。
另一方面，其他属性影响开发环境中的所有 PLC 项目。您可在 PLC 项目的项目属性中更改这些属性，但这些属性也会影响所有其它 PLC 项目，并且标题为 **Solution options (解决方案选项)**。

另请参见：

- PLC 文档：配置 PLC 项目 [▶ 54]

17.4.4.1 类别：通用

类别 **Common**（通用）包含项目文件的一般信息和元信息。TwinCAT 使用此信息创建 **Properties**（属性）选项卡中的密钥。例如，如果 **Company**（公司）文本字段包含名称“Company_A”，**Properties**（属性）选项卡包含值为“Company_A”的 **Company**（公司）密钥。

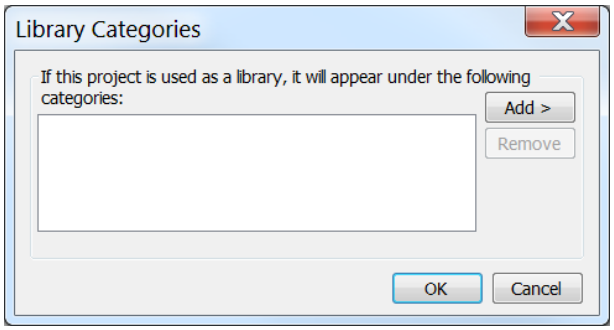
项目信息

对于库项目，必须输入公司、标题和版本才能安装库。	
公司	创建该项目（应用程序或库）的公司的名称。除了库类别之外，它还被用于库存储库的排序
题目	项目标题
版本	项目版本，例如“0.0.0.1”
已发布	☑：启用修改保护。 后果：当您现在编辑项目时，将出现 1 个提示询问您是否想要更改项目。如果您的回答是 Yes（是），则当再次编辑项目时，提示将不再出现。
库类别	您可在库存储库对话框中进行排序的库项目的类别。如果未指定类别，则库会被分配到类别“Other”（其他）。如要将库分配到其他类别，则必须定义该类别。 在 1 个或多个 XML 格式的外部描述文件中定义库类别。如要分配库，您可以调用此类文件或调用另一个已经从描述文件中获取类别信息的库文件。 要求：该项目是 1 个库项目。
…	**Library Categories**（库类别）对话框打开，您可在该对话框中添加库类别。
默认命名空间	库的命名空间的默认设置为库标题。或者，可显式定义不同命名空间，通常在库生成期间在项目信息中为库定义，或者在库引用的 **Properties**（属性）对话框中进行定义，以便在项目中本地使用库。 库的命名空间必须用作标识符的前缀，以便能够明确访问在项目中多次存在的模块，或者库属性强制使用此前缀：LanguageModelAttribute“qualified-access-only”（“对库模块或变量的明确访问”）。 如果您在此处没有定义标准命名空间，则库文件的名称将自动用作命名空间。
占位符	此时，可以指定占位符的默认名称，该名称代表或引用此库。如果此时未显式指定占位符，则库的占位符名称的默认设置对应于库标题。
作者	项目作者
描述	项目的简要描述（例如，内容、功能、一般信息，例如“仅供内部使用”等）

库特性

创建全局版本结构	在 PLC 项目中创建全局变量列表，其中包含版本信息。
自动生成库信息 POU	**Add**（添加）按钮：在项目树中自动创建“函数”类型的 POU 对象，可用于访问应用程序中的项目属性。在这种情况下，将为属性 **Company**（公司）、**Title**（标题）和 **Version**（版本）生成特殊函数（F_GetCompany、F_GetTitle、F_GetVersion）。如果已通过点击 **Add**（添加）将这些函数添加到项目中，则可通过点击 **Remove**（删除）将其从项目中删除。
文档格式	选项： • TC3.1 Build 4024 及以下：**reStructuredText**： • TC3.1 Build 4026 及以上：**TcDocGen** 在库创建期间，对应于特殊格式的注释在此自定义视图中重新构建，并在库管理器的 **Documentation**（文档）选项卡中显示。这会打开库文档的更多选项。
允许对已编译的库进行隐式检查	TC3.1 Build 4026 及以上可用 ☑：TwinCAT 也会对受保护库（*.compiled-libraries）中的功能块执行隐式检查。 要求：在 **PLC project properties**（PLC 项目属性）中的**Compiler definitions**（编译器定义）字段（类别 **Compile**（编译））中输入编译器定义“checks_in_libs”。 另请参见：使用功能块进行隐式检查 [▶ 151]
强制对库访问进行 Qualified_only	TC3.1 Build 4026 及以上可用 ☑：该库中的对象只能与该库的命名空间规范一起使用。 另请参见：属性“qualified only” [▶ 761]
允许作为库引用	TC3.1 Build 4026 及以上可用 ☑：您可以在另一个 PLC 项目中引用 PLC 项目作为库。 另请参见：使用 PLC 项目作为引用库

库类别对话框



类别列表	分配至库项目的类别列表。类别列表可以源于多个来源。输入所有需要的类别后，通过 **OK（确定）** 确认对话框。
添加	显示命令 **From Description File...（从描述文件...）** 和 **From Other Library...（从其他库...）**。
删除	TwinCAT 删除所选类别。
从描述文件...	**Select Description File（选择描述文件）** 对话框出现，可在其中选择扩展名为 *.libcat.xml 的描述文件。文件包含命令类别。通过 **Open（打开）** 退出对话框后，TwinCAT 将应用这些类别。
从其他库...	**Select Library（选择库）** 对话框出现，可在其中选择待采用命令类别的库（*.library）。通过 **Open（打开）** 退出对话框后，TwinCAT 将应用这些类别。
确认	TwinCAT 提供类别作为项目信息，并在 **Library Categories（库类别）** 字段中显示。
取消	关闭对话框。过程中止。

另请参见：

- PLC 文档：配置 PLC 项目 [▶ 54]

- PLC 文档：使用库 [▶ 246]

17.4.4.2 类别编译

PLC 项目属性范围

注意：不同项目属性之间的范围不同！
一些属性仅影响您当前配置属性的 PLC 项目。
另一方面，其他属性影响开发环境中的所有 PLC 项目。您可在 PLC 项目的项目属性中更改这些属性，但这些属性也会影响所有其它 PLC 项目，并且标题为 Solution options（解决方案选项）。

类别“Compile”（编译）用于配置编译器选项。

设置

编译器定义	您可以在此处输入编译器定义/“定义”（请参见 {define} 语句）和编译应用程序的条件（条件编译）。 有关可用条件编译指示的说明，请参见“条件编译指示 [▶ 775]”部分。在此处也可以输入此类编译指示中所用的表达式 expr。可以通过逗号分隔列表的形式输入若干条目。
系统编译器定义	TwinCAT 3.1 Build 4024 及以上版本可用 在PLC项目中编译器定义 [▶ 858]已经在系统管理器的设置中进行了配置。这些配置会自动应用到当前的项目中，无需再次手动设置。
下载应用程序信息	TwinCAT 3.1 Build 4024 及以上版本可用 情景：您正在向控制器加载 1 个 PLC 项目，该项目与控制器中已存在的项目不同。在这种情况下，会出现 1 个包含“Details”（详细信息）按钮的消息窗口。使用此按钮可打开“Application information”（应用程序信息）窗口，您可以通过该窗口查看当前 PLC 项目与控制器上的 PLC 项目之间的区别。其中包括比较功能块的数量、数据和存储位置。 “Application information”（应用程序信息）窗口包含对各项区别的简要描述，例如： • MAIN 声明已更改 • 在 MAIN 中已插入变量 fbMyNewInstance • FB_Sample 的方法/动作的数量已更改 ☑（默认）：如果启用此设置，则会将 PLC 项目内容的相关信息加载到 PLC 上。这使得可以对当前PLC项目和控制器上的PLC项目进行扩展检查。扩展检查选项的不同之处在于，“Application information”（应用程序信息）窗口包含 1 个附加的“Online comparison”（在线比较）选项卡，该选项卡会显示树形比较视图。该视图会告知您哪些 POU 已更改、删除或添加。当您执行“Application information”（应用程序信息）窗口下部区域中的蓝色下划线命令时，就会显示该附加选项卡（“应用程序并非最新版本。立即生成代码以显示在线比较？”）。
生成 tpy 文件	TwinCAT 3.1 Build 4024 及以上版本可用 tpy 文件包含项目、路由、编译器和目标系统信息等。它是对 TwinCAT 2 PLC 项目进行说明时所使用的格式。为了确保与现有应用程序兼容，如有需要，可为 TwinCAT 3 PLC 项目创建此文件。 ☐（默认）：在创建 PLC 项目时，不会创建与该项目相关的 tpy 文件。 ☑：在创建 PLC 项目时，会创建与该项目相关的 tpy 文件，并将其存储在项目文件夹中。 请注意，该选项的值和配置可用性取决于是否将 TPY 文件配置为目标文件（请参见“设置选项卡 [▶ 859]”）。 • 如果启用 TPY 文件作为目标文件，则会发生以下情况： ◦ TwinCAT 会记住“Generate tpy file”（生成 tpy 文件）选项的当前状态（= “原始值”，见下文。）。 ◦ 如果未发生这种情况，则会在下一次创建项目时自动激活“Generate tpy file”（生成 tpy 文件）选项。 ◦ 此外，“Generate tpy file”（生成 tpy 文件）选项还会显示为灰色，因此，只要将 TPY 文件配置为目标文件，用户就无法禁用该选项。 • 如果随后禁用 TPY 文件作为目标文件，则会发生以下情况： ◦ 下次创建项目时，将为“Generate tpy file”（生成 tpy 文件）选项分配其“原始值”（见上文）。 ◦ 此外，该选项不再显示为灰色，用户可以再次进行配置。

在 TMC 文件中添加声明的初始化值	TwinCAT 3.1 Build 4026.12 及以上版本可用
	
	☑（默认）：将声明的初始化值保存在 TMC 文件中。例如，如果其他应用程序需要初始化值，可使用此选项。
	
	☐：声明的初始化值不保存在 TMC 文件中。这样可以减少 TMC 文件的大小并缩短编译时间，尤其是当 PLC 项目中的声明包含大量初始化值时。
	示例：
	`VAR`
	`  aSample : ARRAY[1..10] OF INT := [2, 38, 5, 9, 74, 62, 87, 3, 16, 4];`
	`END_VAR`

在 TMC 文件中添加 POU 和 DUT 声明的相关注释	TwinCAT 3.1 Build 4026 及以上版本可用
	
	☑（默认）：将 POU 和 DUT 声明上方的注释与数据类型一起存储在 TMC 文件中。例如，如果其他应用程序需要注释，可使用此选项。
	
	☐：POU 和 DUT 声明上方的注释不存储在 TMC 文件中。这样可以减少 TMC 文件的大小并缩短编译时间，尤其是当注释中包含对 POU 和 DUT 的详细描述时。
	Sample：
	`// This function block represents an axis`
	`FUNCTION_BLOCK FB_Axis`
	…

解决方案选项

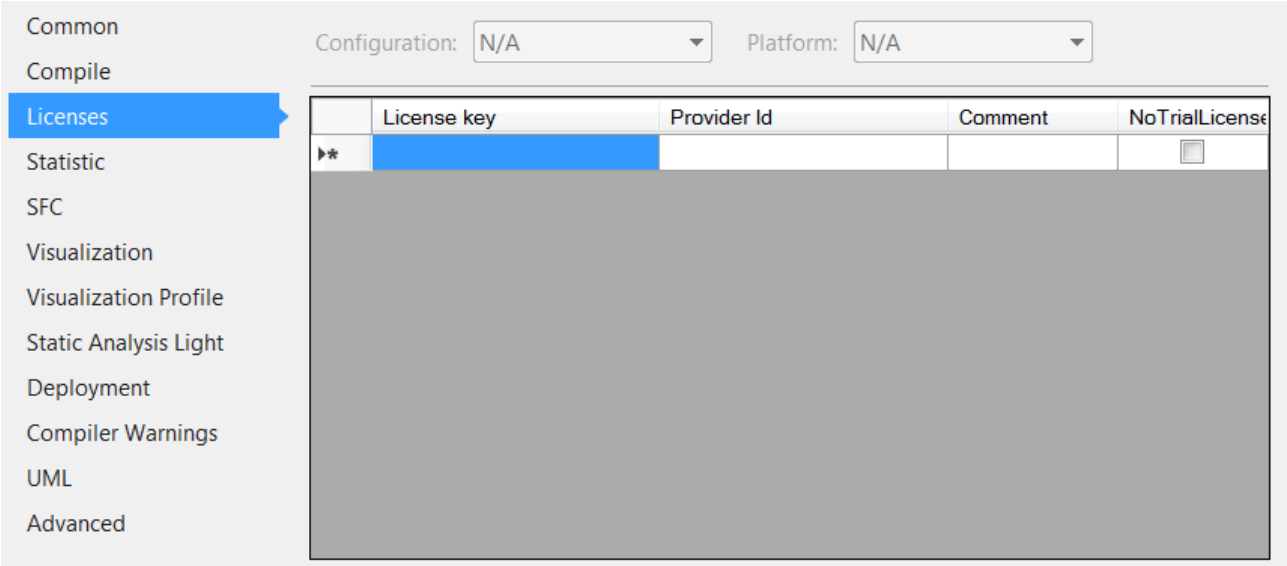
编译器版本	定义了 TwinCAT 在编译期间和编译加载期间使用的编译器版本。 请注意，该设置不会替代远程管理器。 如果 PLC 项目是应用程序项目，在处理不同版本时，应始终使用远程管理器。 在使用远程管理工具的情况下，编译器的版本应该设置为最新的版本。 编译器版本设置只有当你的PLC项目是一个库项目并且需要管理不同版本时才适用。 建议将库保存为最终要使用的最旧版本。为此，必须将编译器版本设置为相应的固定版本（例如，“3.1.4018.0”）。
最大警告数	指的是 TwinCAT 在“**Error List**”（错误列表）视图中发布的警告的最大数量。 在“**Project Settings**”（项目设置）对话框的类别“**Compiler warnings**”（编译器警告）中定义所显示编译器警告的选择。
替换常量	☑：TwinCAT 会直接加载每个标量类型常量的值，也就是说，不包括 STRING、ARRAY 或结构的值。在在线模式下，TwinCAT 可在声明编辑器或监控窗口中用一个符号标识这些常量的值。在这种情况下，无法通过 ADR 运算符进行访问（例如，强制或写入这些常量的值）。 ☐（默认）：可以访问常量。计算时间略有增加。

另请参见：

- 类别：编译器警告 [▶ 851]

17.4.4.3 授权类别

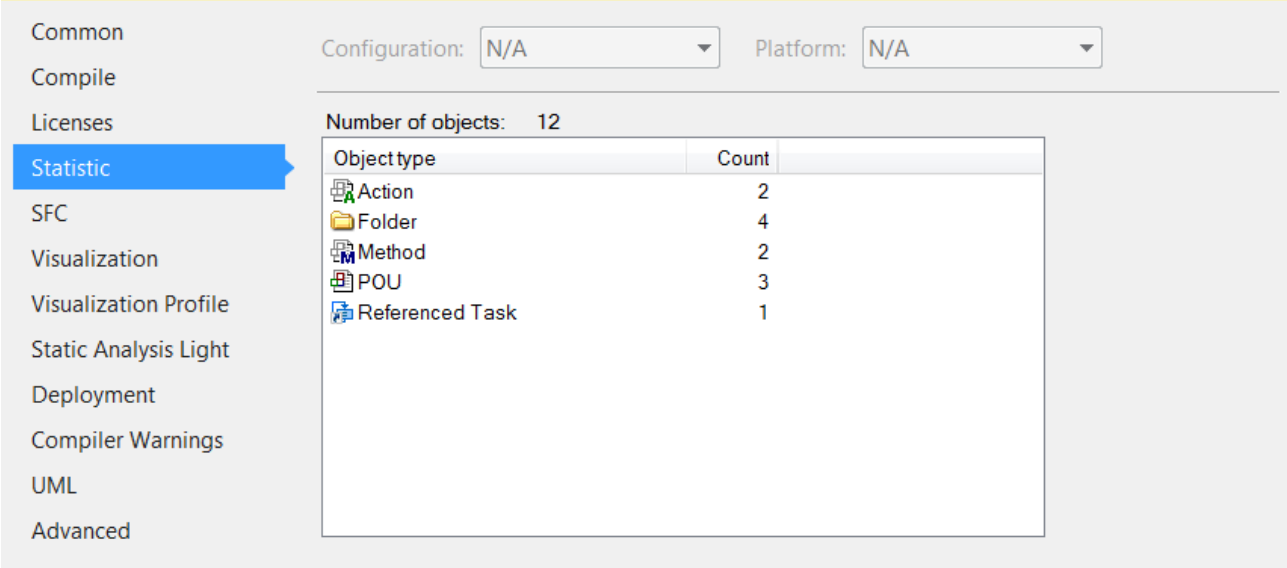
保留以供将来使用！



注意：用户目前必须始终在库代码中查询自己库的 OEM 授权。请参见在 PLC 应用程序中的查询 OEM 授权。

17.4.4.4 类别：统计

Statistic（统计）类别提供关于项目中存在多少不同类型对象的统计信息。



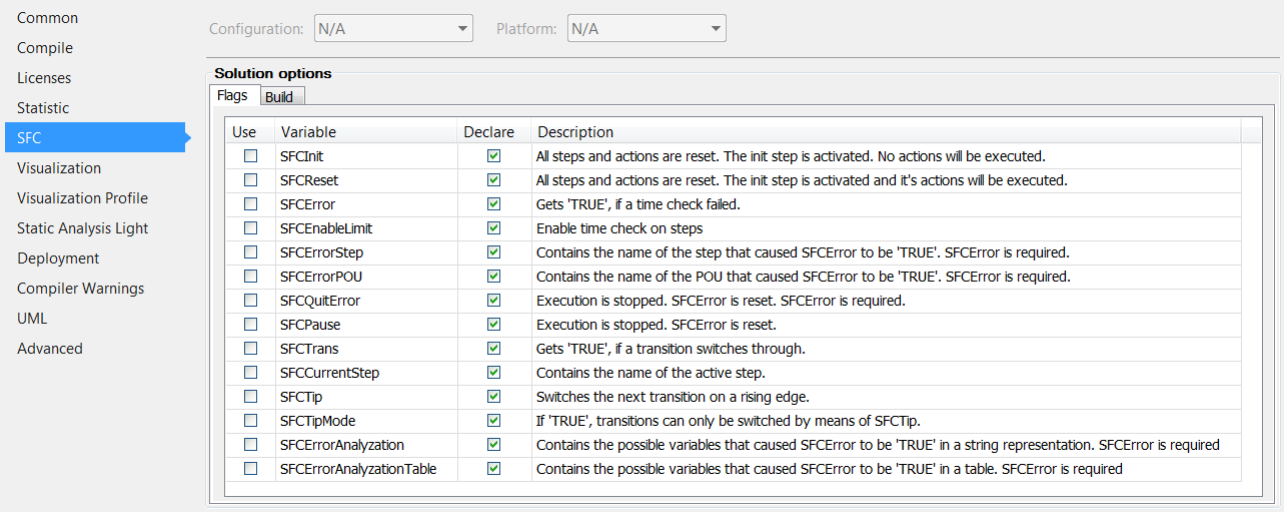
17.4.4.5 类别：SFC

PLC 项目属性范围

注意：不同项目属性之间的范围不同！
一些属性仅影响您当前配置属性的 PLC 项目。
另一方面，其他属性影响开发环境中的所有 PLC 项目。您可在 PLC 项目的项目属性中更改这些属性，但这些属性也会影响所有其它 PLC 项目，并且标题为 **Solution options（解决方案选项）**。

SFC 类别用于配置 SFC 对象的设置。每个新 SFC 对象在其属性中自动具有已配置的设置。

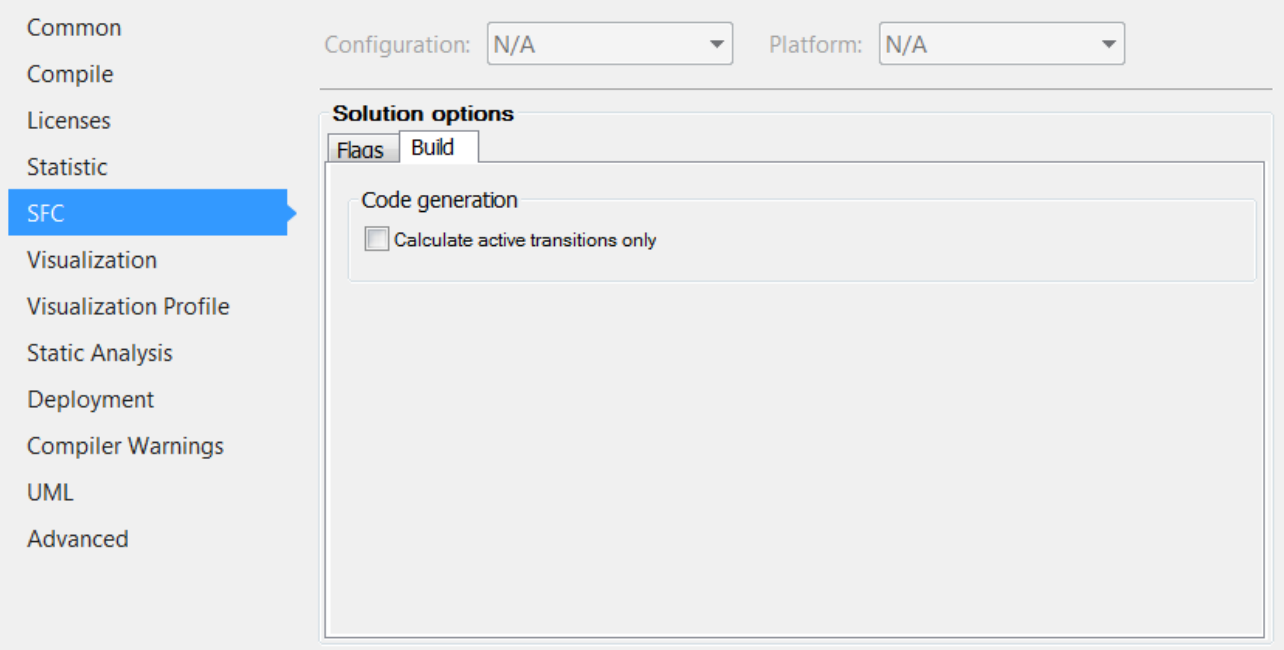
标签选项卡



隐式生成变量（标志），用于控制和监控 SFC 图中的流程。	
使用	：使用相应变量。
声明	：自动创建相应变量。否则，如果计划使用（设置使用），则用户必须声明变量。

自动声明的标志变量出现在 SFC 编辑器的声明部分中，但仅在在线模式下适用。

构建选项卡



代码生成

仅计算活动转换	：TwinCAT 仅为当前活动的转换生成代码。

另请参见：

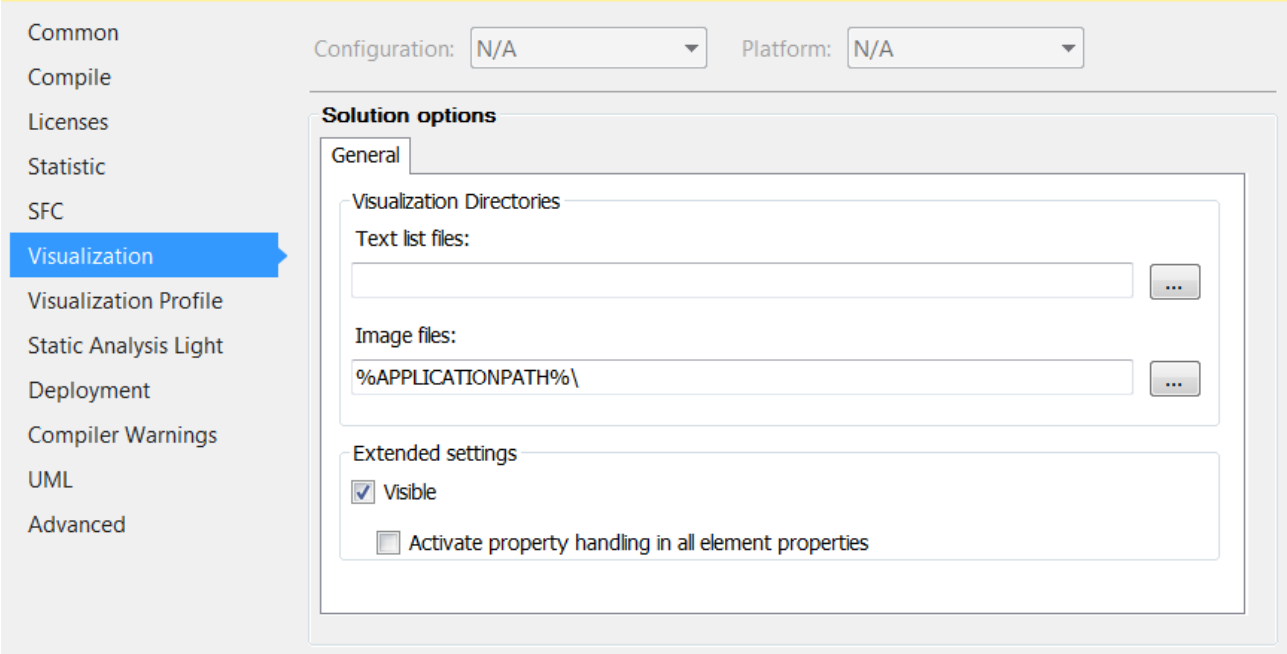
- SFC 标志 [▶ 584]

17.4.4.6 类别：可视化

PLC 项目属性范围

注意：不同项目属性之间的范围不同！
一些属性仅影响您当前配置属性的 PLC 项目。
另一方面，其他属性影响开发环境中的所有 PLC 项目。您可在 PLC 项目的项目属性中更改这些属性，但这些属性也会影响所有其它 PLC 项目，并且标题为 Solution options（解决方案选项）。

Visualization（可视化）类别用于为可视化类型的对象配置项目范围的设置。



常规选项卡

可视化目录

文本列表文件	目录，包含项目中可用于配置不同语言文本的文本列表。例如，在导出或导入文本列表时，TwinCAT 使用此目录。 点击 ... 打开 **Find Folder（查找文件夹）**对话框，可通过此对话框选择文件系统中的目录。
图像文件	目录，包含项目中可用的图像文件。多个文件夹以分号分隔。例如，TwinCAT 在导出或导入图像文件时使用此目录。 点击 ... 打开 **Find Folder（查找文件夹）**对话框，可通过此对话框选择文件系统中的目录。

扩展设置

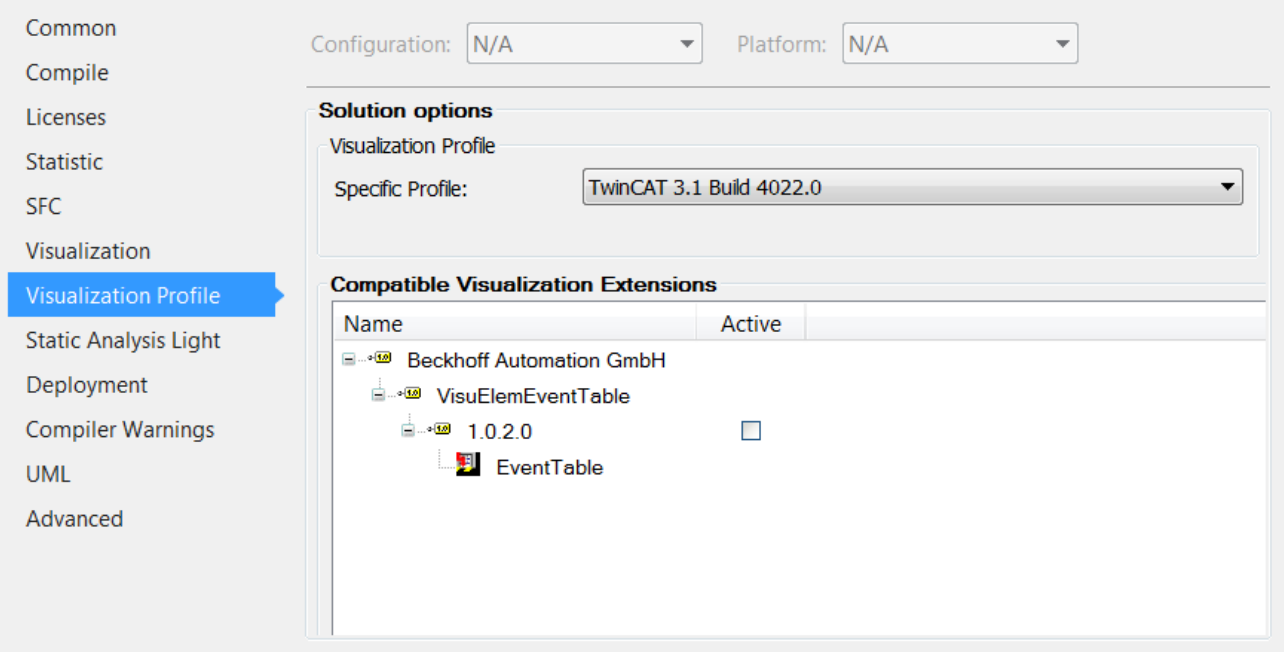
在所有元素属性中启用属性处理	☑：还可以在其属性中配置可视化元素，其中可以选择具有 属性的 IEC 变量。然后，TwinCAT 在编译可视化时生成用于属性处理的附加代码。 要求：IEC 代码至少包含一个类型接口属性的对象，即属性 。 MAIN (PRG) Property_A Get Set 要求：**Visible（可见）**启用。

17.4.4.7 类别：可视化配置文件

PLC 项目属性范围

注意：不同项目属性之间的范围不同！
一些属性仅影响您当前配置属性的 PLC 项目。
另一方面，其他属性影响开发环境中的所有 PLC 项目。您可在 PLC 项目的项目属性中更改这些属性，但这些属性也会影响所有其它 PLC 项目，并且标题为 Solution options（解决方案选项）。

通过 Visualization Profile（可视化配置文件）类别，可以设置可视化配置文件。



可视化配置文件

特定配置文件	TwinCAT 在项目中使用的配置文件，确定项目中可用的可视化元素。 选择列表包含所有之前安装的配置文件。

17.4.4.8 类别静态分析

PLC 项目属性范围

注意：不同项目属性之间的范围不同！
一些属性仅影响您当前配置属性的 PLC 项目。
另一方面，其他属性影响开发环境中的所有 PLC 项目。您可在 PLC 项目的项目属性中更改这些属性，但这些属性也会影响所有其它 PLC 项目，并且标题为 Solution options（解决方案选项）。

类别 Static analysis（静态分析）定义了在静态代码分析中考虑的检查。

轻量静态分析：

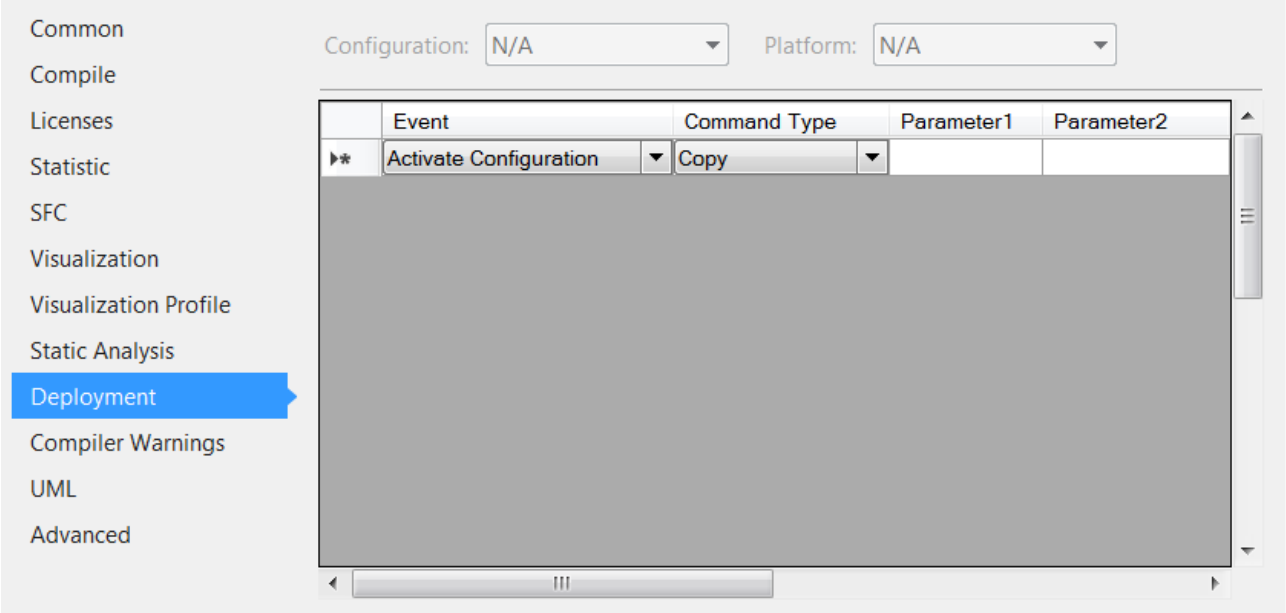
- 如果您尚未激活附加的 TE1200 工程授权，则您可以使用免授权版本的静态分析（轻量静态分析），其中包含一些编码规则。免费的轻量版本使您能够熟悉产品的基本操作，例如，基于 1 套精简的功能。
- 有关轻量静态分析的更多信息，请参见：
  PLC 文档：对 PLC 项目进行编程 > 检查语法和分析代码 > 代码分析（静态分析） [▶ 136]

全量静态分析：

- 如果启用了额外的 TE1200 工程授权，则可使用全量静态分析功能（保存和加载设置、100 多条编码规则、命名规范、度量、禁止符号）。
- 有关全量静态分析的更多信息，请参见：
  TE1200 静态分析。

17.4.4.9 类别：部署

Deployment（部署）类别用于设置在安装和启动应用时执行的命令。



以下事件可用，之后可以调用列表中列出的命令：

启用配置	启用配置后，将调用所需命令。
Plc 下载	将 PLC 应用下载到目标系统后，将调用所需命令。
Plc 在线更改	在线更改成功后，将调用所需命令。
Plc 编译后	编译 PLC 应用后，将调用所需命令。

可以执行以下命令：

复制	将参数 1（源路径）中的文件复制到参数 2（目标路径）中指定的位置。
执行	执行参数 1 下列出的应用或脚本。

源路径和目标路径可能包含虚拟环境变量，而 TwinCAT 会相应地进行解析。

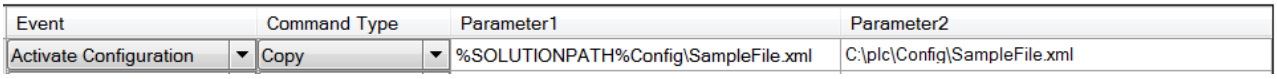
支持以下环境变量：

虚拟环境变量	注册表值	默认值
%TC_INSTALLPATH%	InstallDir	C:\TwinCAT\3.x \
%TC_TARGETPATH%	TargetDir	C:\TwinCAT\3.x \Target\
%TC_BOOTPRJPATH%	BootDir	C:\TwinCAT\3.x \Boot\
%TC_RESOURCEPATH%	ResourceDir	C:\TwinCAT\3.x \Target\Resource\
%SOLUTIONPATH%	-	解决方案文件的位置

注册表值存储在注册表中的以下项下：*\HKLM\Software\Beckhoff\TwinCAT3*。

示例：

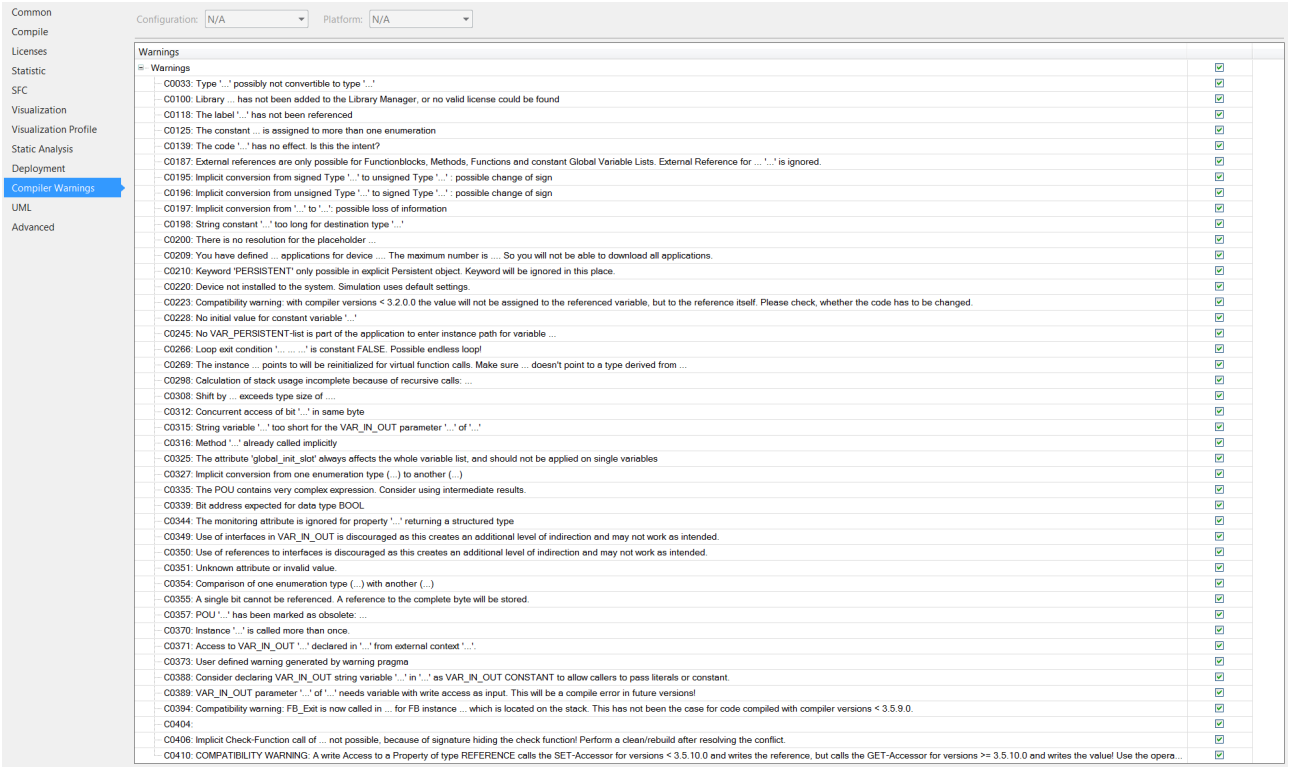
在以下示例中，文件 *SampleFile.xml* 从解决方案的配置项目子文件夹复制到目标系统的文件夹 *C:\plc\config* 中。



17.4.4.10 类别：编译器警告

Compiler Warnings（编译器警告）类别用于选择编译运行期间 TwinCAT 在消息窗口中显示的编译器警告。

可以在 **Compile（编译）**类别中指定所列出的最大警告数。



另请参见：

- 命令构建 PLC 项目 [▶ 863]
- 类别编译 [▶ 844]

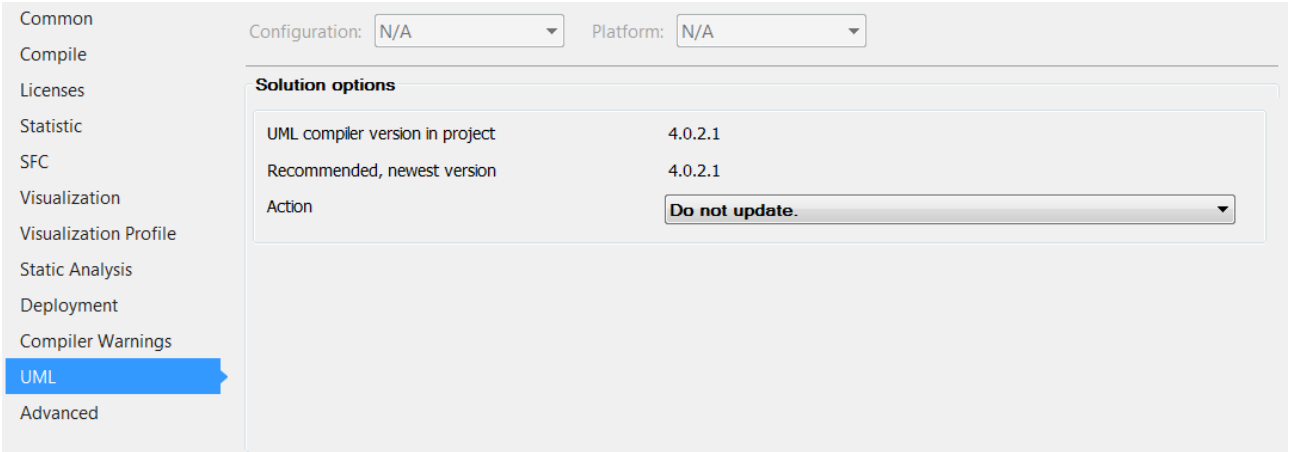
17.4.4.11 类别 UML

PLC 项目属性范围

注意：不同项目属性之间的范围不同！
一些属性仅影响您当前配置属性的 PLC 项目。
另一方面，其他属性影响开发环境中的所有 PLC 项目。您可在 PLC 项目的项目属性中更改这些属性，但这些属性也会影响所有其它 PLC 项目，并且标题为 **Solution options（解决方案选项）**。

在类别 **UML** 中，您可以更改 UML 编译器版本。此设置仅在使用 UML 状态图时相关。

有关配置选项的更多信息，请参见 TF1910 TC3 UML 文档的“UML 编译器版本”部分。



17.4.4.12 高级类别

PLC 项目属性范围

注意：不同项目属性之间的范围不同！
一些属性仅影响您当前配置属性的 PLC 项目。
另一方面，其他属性影响开发环境中的所有 PLC 项目。您可在 PLC 项目的项目属性中更改这些属性，但这些属性也会影响所有其它 PLC 项目，并且标题为 Solution options（解决方案选项）。

类别“Advanced”（高级）用于配置高级属性。

写入选项

文件版本 1.2.0.0（或更高）与 TwinCAT 3.1 < Build 4024 在工程上不兼容

请注意，不能将以文件版本 1.2.0.0（或更高）保存的对象加载到低于 TwinCAT 3.1.4024 的开发环境版本中！

由于在使用可选的 Base64 格式时，对象会自动以文件版本 1.2.0.0 保存，因此无法使用低于 TwinCAT 3.1.4024 的开发环境版本加载 Base64 格式的对象。

如果 1 个 PLC 项目包含文件版本为 1.1.0.1 的对象和文件版本为 1.2.0.0 的对象，则 1.1.0.1 对象将以低于 TwinCAT 3.1.4024 的开发环境版本加载。未加载文件版本为 1.2.0.0 的对象。

使用 XAE 版本 TwinCAT 3.1.4024 或更高版本，可以将以文件版本 1.2.0.0 保存的文件的文件版本重置为 1.1.0.1。

将对象内容写为 （“Write object content as”）	TwinCAT 3.1 Build 4024 及以上版本可用 **背景信息：** 从 Build 4024 版本开始，**Base64** 引入了 1 种新的存储格式，下列 PLC 对象可选择使用该存储格式： • POU（程序组织单元）的主体是用图形化编程语言编写的。 ◦ 顺序功能图 (SFC) ◦ FBD/LD/IL（功能块图/梯形图/指令表） ◦ CFC（连续功能图和面向页面的 CFC） ◦ UML 类图和状态图 • 带有以图形化实现语言编程的子元素（例如，动作、方法）的 POU（有关图形语言的信息，请参见第 1 个关键点） • 可视化 • 可视化管理器 • 文本列表 • 配方管理器 • 图像池 到目前为止，这些对象默认以 XML 格式保存。 从 Build 4024 版本开始，您可以配置应该将这些对象类型保存为 XML 或 Base64。 **Base64 相对于 XML 的优点：** 与 XML 相比，Base64 可实现压缩存储。因此，通过文件访问这些对象可以提高性能，例如，这种方法在加载项目或移动、复制对象时特别有用。 **标准存储格式的设置选项：** 对于 PLC 项目，PLC 项目属性中的设置“Write object content as”（将对象内容写为）可用于定义上述对象类型的默认存储格式。 选择的标准存储格式只会用于新添加的对象（例外：不适用于新添加的 POU 子对象。示例：将 POU 保存为 XML，并将标准存储格式配置为 Base64。如果随后在 POU 中添加图形子对象，那么POU的存储格式以及这个子对象的存储格式仍然保持为XML格式）。 在更改和保存非标准存储格式的现有对象时，其存储格式不会自动更改。可通过“Properties”（属性）窗口单独更改现有对象的存储格式（见下文）。或者，在更改标准存储格式时，还可以选择对所有现有对象采用新选定的存储格式。如果在此时更改存储格式，则会出现相应的问询窗口。 设置“Write object content as”（将对象内容写为）可提供以下选项： • **XML**（默认）：上述 PLC 对象默认以 XML 格式保存。 ◦ 使用此存储格式的对象按文件版本 1.1.0.1 进行存储。 • **Base64**：上述 PLC 对象默认以 Base64 格式保存。 ◦ 使用此存储格式的对象按文件版本 1.2.0.0 进行存储。 请注意，不能使用低于 TwinCAT 3.1.4024 版本的开发环境加载文件版本为 1.2.0.0（或更高）的对象！ **存储格式的单独设置选项：** 对于上述对象类型，可在对象的“**Properties**”（属性）窗口中单独配置该对象的存储格式。有关更多信息，请参见关于属性 [▶ 836]的描述（**格式**属性）。

在文件中写入产品版本 （“Write product version in files”）	TwinCAT 3.1 Build 4024 及以上版本可用 产品版本表示使用哪个插件版本来保存 PLC 文件（例如，功能块）。该复选框的设置对整个项目有效，并且是该 PLC 项目中所有已修改或新添加的 PLC 对象的默认设置。 ☐（默认）：不会将产品或插件版本写入文件。 • 如果您将该设置从启用改为禁用，则会出现 1 个问询窗口，您可以在其中选择是否从所有现有文件中移除产品版本信息。 • 禁用选项的用例：如果您对产品版本不感兴趣，则可以使用该设置。这样可以最大限度地减少源代码管理系统对文件的更改。 ☑ 将产品或插件版本写入文件（在XAE中看不到这个版本信息；只有在文件级别进行分析时才会显示出来）。 • 如果您将该设置从禁用改为启用，则会出现 1 个问询窗口，您可以在其中选择是否将产品版本添加到所有现有文件中。 • 启用选项的使用场景：例如，可以使用该设置将文件版本信息包含在文件中，以便进行调试或跟踪。 • 请注意以下几点：如果以不同的产品版本保存文件，会导致该文件发生更改，在使用源代码管理系统时会显示文件存在差异。
使用配置文件写入对象内容	配置文件定义了对象的保存格式。例如，在 Build 4024 中，为 PLC HMI 添加了新功能。因此，用 Build 4024 保存的可视化文件无法直接用旧版本打开。如果您在此处设置了 4022 配置文件，可视化文件将以相应的格式保存，并可使用 Build 4022 打开。 **要求：**例如，若要在下拉菜单中显示 4022 配置文件，则必须安装 4022 远程管理器，或者通过先前的 4022 XAE 安装程序安装当前最新的 4024 XAE。
将书签写入文件	TwinCAT 3.1 Build 4026 及以上版本可用 ☐（默认）：仅将书签保存在 Visual Studio 的 .suo 文件中。从项目目录中删除已创建的 .bookmarks 文件。 ☑：同时将 Visual Studio 项目用户选项文件（.suo）中存储的书签写入单独的文件。该文件以 .bookmarks 结尾，位于项目目录中。因此，它也是已知归档选项的一部分。 新建 PLC 项目时，是否将书签存储在单独文件中的全局默认设置请参见“对话框选项 - 写入选项 [▶ 922]”。在创建新 PLC 项目时，这个设置的值会被传递到本地项目设置中。

多用户选项

使用 Multiuser （“Use Multiuser”）	TwinCAT 3.1 Build 4024 及以上版本可用 ☐（默认）：不启用 PLC 项目的多用户功能。 ☑：启用 PLC 项目的多用户功能。 另请参见多用户文档中的更多信息。
多用户 URL	TwinCAT 3.1 Build 4026 及以上版本可用 相关描述见后文。
更新父节点	TwinCAT 3.1 Build 4026 及以上版本可用 相关描述见后文。

解决方案选项

安全在线模式 （“Secure Online Mode”）	☐：（默认）：出于安全考虑，始终会提示用户在调用以下命令时确认执行这些命令。 • 激活配置 • 在配置/运行模式下重启 TwinCAT 系统 • 冷复位 • 初始值复位 ☑：除了上述命令之外（默认出现确认提示），还将提示您确认以下命令。 • 登录 • 开始 • 停止 • 单循环 • 写入值 • 强制值 • 释放值
自动更新可视化配置文件	可通过此选项配置可视化配置文件的自动更新操作。 当您打开使用过时可视化配置文件的 PLC 项目时，会在消息窗口中出现警告（“New Version found for Visualization profile”）（发现可视化配置文件的新版本）。 ☐（默认）：如果禁用了“**Autoupdate Visu Profile**”（自动更新可视化配置文件）选项，可视化配置文件版本不会自动更改。您可以双击警告“New Version found for Visualization profile”（发现可视化配置文件的新版本），打开 ProfileUpdate 对话框，在该对话框中，您可以手动更改可视化配置文件版本。 ☑：在这种情况下，如果启用了“**Autoupdate Visu Profile**”（自动更新可视化配置文件）选项，会将可视化配置文件版本自动设置为最新版本。在此类可视化配置文件版本自动更新后，消息窗口中将显示相应的警告（例如，“Visualization profile set from ‘TwinCAT 3.1 Build 4020.10’ to ‘TwinCAT 3.1 Build 4022.0’”（已将可视化配置文件从“TwinCAT 3.1 Build 4020.10”设置为“TwinCAT 3.1 Build 4022.0”））。
自动更新 UML 配置文件	可通过此选项配置 UML 编译器版本的自动更新操作。 如果您打开使用过时 UML 编译器版本的 PLC 项目，会在消息窗口中显示相应的警告（“new version for UML found”（发现 UML 的新版本））。 ☐（默认）：如果禁用了“**Autoupdate Uml Profile**”（自动更新 UML 配置文件）选项，UML 编译器版本不会自动更改。双击警告“new version for UML found”（发现 UML 的新版本），打开 ProfileUpdate 对话框，在该对话框中，您可以手动更改 UML 编译器版本。 ☑：在这种情况下，如果启用了“**Autoupdate Uml Profile**”（自动更新 UML 配置文件）选项，会将 UML 编译器版本自动设置为最新版本。如果 UML 编译器版本自动更新，消息窗口中将显示相应的警告（例如，“UML set from ‘4.0.2.0’ to ‘4.0.2.1’”（已将 UML 从“4.0.2.0”设置为“4.0.2.1”））。 有关更多信息，请参见“UML 编译器版本”。
写入行 ID	TwinCAT 3.1 Build 4026 及以上版本可用 ☐（默认）：不生成单独的行 ID。在这种情况下，机器代码指令的分配和断点处理会使用行号。因此，对于空格或注释等更改，需要进行在线更改。 ☑：会为项目的 POU 生成并存储行 ID（TwinCAT 3.1 Build 4024 及以下版本的默认操作）。行 ID 可用于将代码行分配给机器代码指令，在进行断点处理等操作时需要使用行 ID。 新建PLC项目时，全局默认设置“写入行ID”，请参见“对话框选项 - 写入选项 [▶ 922]”。在创建新 PLC 项目时，这个设置的值会被传递到本地项目设置中。

兼容性

将 PLC 项目转换为先前的 TwinCAT 版本	TwinCAT 3.1 Build 4026 及以上版本可用 在打开的对话框中，您可以选择 PLC 项目应该转换的 TwinCAT 版本（TwinCAT 3.1 Build 4022 或 4024）。使用“Convert”（转换）确认转换之后，项目将会关闭并保存为与所选版本兼容的版本。 请注意，在转换过程中，项目数据将会更改，较新版本的设置和属性将会丢失。因此，该转换不适合在不同版本之间进行多次切换。

将事件写入事件日志

您可以使用以下 2 个选项来配置是否要记录用户事件，以及记录何种类型的用户事件。以下事件属于用户事件：

- 登录
- 退出
- 下载
- 在线更改
- 启动 PLC 项目
- 停止 PLC 项目
- 单循环
- 冷复位
- 初始值复位
- 设置断点
- 删除断点

运行时会将事件发送至 EventLogger。此外，如果将 EventLogger 选项“Output as Task Item”（作为任务项目输出）设置为 True，将会在消息窗口（错误列表）中输出事件。

如果您更改了下述选项，下次登录时需要进行在线更改。设置随即生效。

信息	TwinCAT 3.1 Build 4026 及以上版本可用 ☐（默认）：在发生成功的用户事件时，不记录任何消息。 ☑：如果上述用户事件之一成功发生，那么与此同时将会记录此过程的相应信息。
错误	TwinCAT 3.1 Build 4026 及以上版本可用 ☐（默认）：当用户事件失败时，不记录任何消息。 ☑：如果在上述用户事件之一发生时出现错误，那么与此同时将会记录此失败过程的相应信息。例如，如果因在运行时出错而导致登录失败，就会执行此操作。

常规

允许在 1 个解决方案中多次出现具有相同 ID 的（相同）对象 （或“Allow (identical) objects with the same ID multiple times in one solution”）	☐（默认）：PLC 对象（例如，POU）通过其 GUID 进行识别，该 GUID 在整个解决方案中只分配 1 次。在通常使用情况下，该选项可以且应该保持禁用状态。 ☑：该激活的选项仅适用于在 1 个解决方案中多次存在具有相同 ID/GUID 的对象的情况。在这种情况下，必须更改对象的标识类型，以便将 PLC 对象的 GUID 链接至 PLC 项目的 GUID（使用 XOR）。这样可以避免在不同项目中多次使用同一个对象时更改其GUID。 **信息：** • 如果您更改了选项的状态，则必须重新加载项目。下次登录时也需要进行**下载**。 • 如果**激活**该选项，该 PLC 项目中所有对象的标识类型都会发生更改。 • 如果**激活**该选项，则**无法使用或无法充分使用**以下 PLC 功能： ◦ 可视化 ◦ UML SC ◦ 安全管理 该选项已重命名为 TwinCAT 3.1 Build 4026.x。先前的名称为：“Minimize ID changes in TwinCAT files”（最小化 TwinCAT 文件中的 ID 更改）或“Minimize ID changes in TwinCAT files”（最小化 TwinCAT 文件中的 ID 更改）。该选项的操作保持不变。

17.4.5 PLC 项目设置

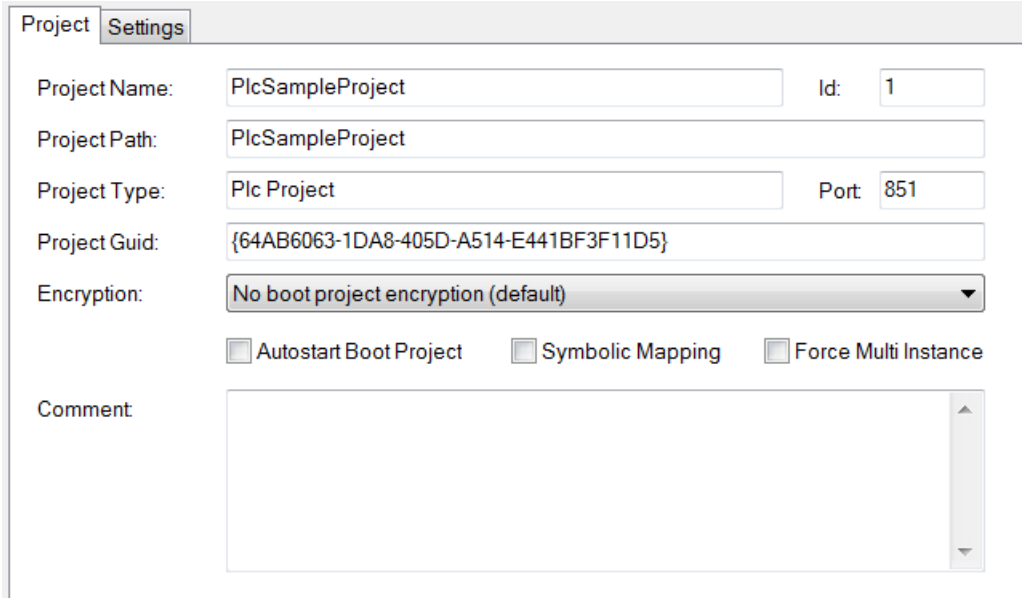
功能：此命令可打开编辑器，可在其中定义项目设置。

调用：在 Solution Explorer（解决方案资源管理器）中双击 PLC 项目

另请参见：

- PLC 文档：配置 PLC 项目 [▶ 54]

17.4.5.1 项目选项卡

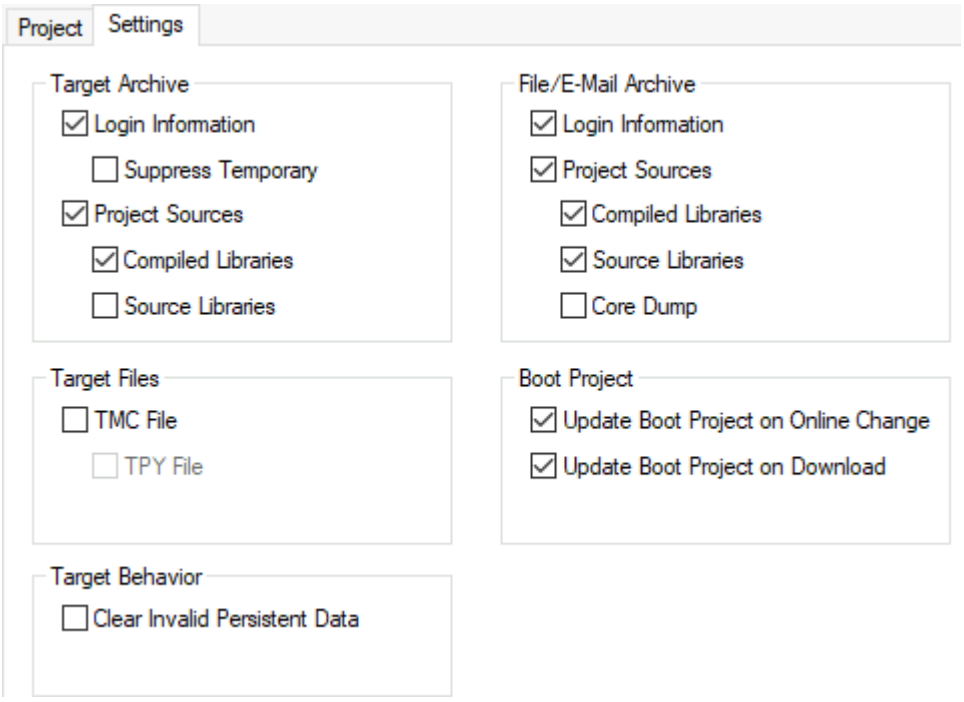


项目名称	PLC 项目的名称
Id	PLC 项目的 ID
项目路径	PLC 项目存储位置路径
项目类型	项目类型
端口	运行时系统的 AMS 端口编号
项目 Guid	PLC 项目的 GUID
加密	启动项目的加密 • 无启动项目加密（默认） • 加密启动项目
自动运行启动项目	☑ 在启动 TwinCAT 运行时环境后，将自动加载并运行 PLC 启动项目。 该设置直接传输到当前选定的目标系统。设置未保存在 TwinCAT 项目中。 在解决方案资源管理器中，此选项对应于 PLC 项目节点的上下文菜单中的 **Autostart Boot Project**（自动运行启动项目）命令。
符号映射	☑ 符号映射已启用。
强制多实例	☑ 用于登录 PLC 项目多实例的选项已启用。
注释	注释框
编译器定义（TC3.1 Build 4024 及以上可用）	
手动	在此处，您可以在系统管理器级别定义自己的编译器定义，并将其传递给 PLC 项目。在 PLC 项目属性中的编译类别下，可将这些定义作为系统编译器定义 [▶ 844]输入。
隐式	☑ 所选项目变体的名称以及项目变体所属的所有组都会被自动设置为编译器定义，并传递给 PLC 项目。在 PLC 项目属性中的编译类别下，可将这些定义作为系统编译器定义 [▶ 844]输入。 **注意：**为了激活该复选框，必须启用变体管理的定义。

另请参见：

- 变体管理：概念：集成到 PLC 项目中
- 命令属性（PLC 项目）：类别编译：系统编译器定义 [▶ 844]

17.4.5.2 设置选项卡



目标存档

在“Target Archive”（目标存档）分组框中，您可以指定在创建启动项目时将哪些信息与其他数据一起传输到目标系统。

登录信息	包含 PLC 项目编译器信息的 COMPILEINFO 文件。
临时抑制	如果用户在登录目标系统时还取消选择了“Update boot project”（更新启动项目）选项，暂时不会将 COMPILEINFO 文件加载到目标系统中。这样可加快下载/在线更改的速度。
项目源	PLC 项目源代码文件的可读源代码形式。
编译的库	在 PLC 项目中以编译形式使用的库。
源库	在 PLC 项目中以可读源代码形式存在的库。

文件/电子邮件存档

在“File/E-Mail Archive”（文件/电子邮件存档）分组框中，您可以指定在存档 PLC 项目 [▶ 800]、TwinCAT 项目 [▶ 988]或解决方案 [▶ 798]时存储哪些信息。如果勾选激活相应复选框，项目存档中将存储下表中描述的文件。

登录信息	包含 PLC 项目编译器信息的 COMPILEINFO 文件。
项目源	PLC 项目源代码文件的可读源代码形式。
编译的库	在 PLC 项目中以编译形式使用的库。
源库	在 PLC 项目中以可读源代码形式存在的库。
核心转储	核心转储文件位于PLC项目目录中，而编译信息文件位于项目目录中的“_CompileInfo”文件夹中。 注意：如果激活“核心转储 [▶ 883]”设置，编译信息文件也会被保存到存档中，因为需要具备这些文件才能使用核心转储。

源代码传输

如果您已将目标或文件/电子邮件存档设置配置为包含项目源代码和/或源库，请注意，在传递/交付目标系统时或在传递文件/电子邮件存档时，项目源代码和/或项目中使用的源库（*.library）将以可读的源代码形式包含在ZIP存档中。

在配置上述设置以及存储和引用库（*.library 与 *.compiled-library）时，请牢记这一点。

有关库管理的更多信息，请参见“使用库 [▶ 246]”部分。

有关源代码加密的信息，请参见“安全管理”文档。

目标文件

在“Target Files（目标文件）分组框中，您可以设置在目标系统上创建启动项目时将哪些信息传输到 *\Boot\Plc 文件夹*。

TMC 文件	PLC 项目的 TMC 文件（TwinCAT 模块类）
TPY 文件	TPY 文件（包含项目信息、路由信息、编译器信息、目标系统信息等信息）

启动项目

在“Boot Project”（启动项目）分组框中，您可以设置在下载或在线更改时是否应默认更新目标系统上的启动项目。

在线更改时更新启动项目	在进行在线更改时默认更新目标系统上的启动项目。
下载时更新启动项目	在进行下载时默认更新目标系统上的启动项目。

目标行为

在“Target Behavior”（目标行为）分组框中，您可以设置目标系统如何处理断电保持数据。

清除无效的断电保持数据	存储的断电保持数据备份将被忽略。这样可确保不接受任何无效数据，而是将其丢弃。 请参见：断电保持数据备份 [▶ 861]

断电保持数据备份

在 TwinCAT 系统停止/关闭期间，断电保持数据会定期存储在 *TwinCAT\Boot 文件夹*的 .bootdata 文件中。在下一次系统启动时（TwinCAT 运行模式），将读取此文件，并使用该文件中的值初始化运行时系统中的断电保持变量。系统将 .bootdata 文件重命名为 .bootdata-old。

如果包含断电保持数据的文件 (.bootdata) 不存在，会在系统启动时读取断电保持数据的备份文件 (.bootdata-old)。这是一种例外情况，但也有可能会发生，例如当未配备不间断电源的 IPC 遇到断电且 TwinCAT 无法正常关闭时。

- 如果可以预见备份文件的内容在新系统启动时无法使用，可以启用选项“**Clear Invalid Persistent Data**”（清除无效的断电保持数据），忽略备份文件。例如，如果批处理信息或工具数据已存储在生产设施中并且必须处于最新状态，便可能出现这种情况。
- 如果因在线更改导致断电保持数据结构（其程序代码中的数据类型或符号路径）发生改变，后续加载过时持久数据文件的操作则毫无意义。在这种情况下，应该提前启用“**Clear Invalid Persistent Data**”（清除无效的断电保持数据）选项。

在这两种情况下，还应确保当前断电保持数据文件处于可用状态。为此，可以使用 FB_WritePersistentData (PLC Lib Tc2_Utilities) 等功能块和防止突然断电的不间断电源保护。

在使用断电保持数据时，务必对全局结构 PlcAppSystemInfo 中的相应标志（BootDataLoaded 和 OldBootData）进行评估（参见“系统 > 全局数据类型”文档）。

如果常规文件和备份文件均无法加载，或者二者均不存在，标记为 PERSISTENT 的变量将使用显式指定的初始化值或通过标准初始化以与其他“普通”变量相同的方式重新初始化。

另请参见：

- PLC 文档：剩余变量 - PERSISTENT, RETAIN [▶ 630]

17.5 构建

17.5.1 命令构建解决方案

符号： 

功能：此命令可启动解决方案中包含的所有项目的编译过程或代码生成。

调用：Build（构建）菜单或解决方案的上下文菜单

要求：选择解决方案。

解决方案中包含的所有项目依次进行编译。这也涉及到在 TwinCAT 项目下集成的项目（PLC、C++ 等）。在命令构建 PLC 项目 [▶ 863] 部分中描述了针对 PLC 项目执行的步骤。

17.5.2 命令重建解决方案

功能：此命令可启动解决方案中包含的所有项目的编译过程，即使之前编译时没有错误也不例外。

调用：Build（构建）命令或解决方案的上下文菜单

要求：选择解决方案。

在重建解决方案时，首先要对其进行清除（另请参见：命令清除解决方案 [▶ 862]），然后再进行构建（另请参见：命令构建解决方案 [▶ 861]）。

另请参见：

- 命令重建 PLC 项目 [▶ 863]

17.5.3 命令清除解决方案

功能：此命令可开始清除解决方案中包含的所有项目。

调用：Build（构建）菜单或解决方案的上下文菜单

要求：选择解决方案。

依次清除解决方案中包含的所有项目。这也涉及到在 TwinCAT 项目下集成的项目（PLC、C++ 等）。在 命令清除 PLC 项目 [▶ 863] 部分中描述了针对 PLC 项目执行的步骤。

17.5.4 命令检查所有对象

功能：此命令可启动编译运行，即对 PLC 项目的项目树中的所有对象进行语法检查。这在创建库或处理库项目时非常有用。

调用：Solution Explorer（解决方案资源管理器）中的 PLC 项目对象（<PLC project name> 项目）的上下文菜单

与只检查所用对象的 命令构建 PLC 项目 [▶ 863] 不同，在执行该命令时，将会检查 PLC 项目中所有对象的语法。

该命令不会导致代码生成。在项目目录中未创建包含编译运行信息的文件。

17.5.5 命令构建 TwinCAT 项目

符号： 

功能：此命令可启动当前活动的 TwinCAT 项目的编译过程或代码生成。

调用：如果当前选择的是 TwinCAT 项目，则调用 Build（构建）菜单，或 TwinCAT 项目的上下文菜单

要求：选择 TwinCAT 项目。

TwinCAT 项目中包含的所有项目（PLC、C++ 等）依次进行编译。在 命令构建 PLC 项目 [▶ 863] 部分中描述了针对 PLC 项目执行的步骤。

另请参见：

- 命令重建 TwinCAT 项目 [▶ 862]

17.5.6 命令重建 TwinCAT 项目

功能：此命令可启动当前活动的 TwinCAT 项目的编译过程或代码生成，即使上次编译时没有错误也不例外。

调用：如果当前选择的是 TwinCAT 项目，则调用 Build（构建）菜单，或 TwinCAT 项目的上下文菜单

要求：选择 TwinCAT 项目。

在重建项目时，首先要对 TwinCAT 项目进行清除（另请参见：命令清除 TwinCAT 项目 [▶ 862]），然后再进行构建（另请参见：命令构建 TwinCAT 项目 [▶ 862]）。

17.5.7 命令清除 TwinCAT 项目

功能：此命令可删除当前活动的 PLC 项目的本地编译信息，并更新所有对象的语言模型。

调用：如果当前选择的是 TwinCAT 项目，则调用 Build（构建）菜单，或 TwinCAT 项目的上下文菜单

要求：选择 TwinCAT 项目。

TwinCAT 项目中包含的所有项目（PLC、C++ 等）依次进行清除。在 命令清除 PLC 项目 [▶ 863] 部分中描述了针对 PLC 项目执行的步骤。

另请参见：

- 命令重建 TwinCAT 项目 [▶ 862]

17.5.8 命令构建 PLC 项目

符号： 

功能：此命令可启动当前活动的 PLC 项目的编译过程或代码生成。

调用：如果当前选择的是 PLC 项目，则调用 **Build**（构建）菜单，或者 **Solution Explorer**（解决方案资源管理器）中的 PLC 项目对象（<PLC project name> 项目）的上下文菜单

要求：选择 PLC 项目。

在编译期间，TwinCAT 对 PLC 项目中所用的所有对象执行语法测试。当您想要用更改后的程序记录项目时，始终自动执行编译程序。检查完成后，TwinCAT 会在 Error List [▶ 830]（错误列表）视图中显示任何错误消息或警告。

此外，在构建项目时会创建 PLC 项目的编译信息，并将其保存在解决方案中的本地文件（*.compileinfo）中。

如果自上次无错编译过程后程序未更改，则不会重新编译。如果无论如何都要执行语法测试，请使用 命令重建 PLC 项目 [▶ 863]。

17.5.9 命令重建 PLC 项目

功能：此命令可启动当前活动的 PLC 项目的编译过程或代码生成，即使上次编译时没有错误也不例外。

调用：如果当前选择的是 PLC 项目，则调用 **Build**（构建）菜单，或者 **Solution Explorer**（解决方案资源管理器）中的 PLC 项目对象（<PLC project name> 项目）的上下文菜单

要求：选择 PLC 项目。

在重建项目时，首先要对项目进行清除（另请参见：命令清除 PLC 项目 [▶ 863]），然后再进行构建（另请参见：命令构建 PLC 项目 [▶ 863]）。

17.5.10 命令清除 PLC 项目

功能：此命令可更新当前活动的 PLC 项目中的所有对象的语言模型。

调用：如果当前选择的是 PLC 项目，则调用 **Build**（构建）菜单，或者 **Solution Explorer**（解决方案资源管理器）中的 PLC 项目对象（<PLC project name> 项目）的上下文菜单

前提条件：选择 PLC 项目。

如果清除 PLC 项目，则只更新 PLC 项目中的所有对象的语言模型。目标系统上的编译信息将会保留。

另请参见：

- 命令重建 PLC 项目 [▶ 863]

17.6 调试

17.6.1 命令：新断点

符号： 

功能：此命令打开 **Breakpoint Properties（断点属性）**对话框。

调用：Debug（调试）菜单，Breakpoint（断点）视图中的按钮 New（新）（PLC > Window（窗口） > Breakpoints（断点））。

要求：PLC 项目处于在线模式。

在线模式下，命令 **Toggle Breakpoint（切换断点）**可用于直接在当前光标位置设置新断点。

另请参见：

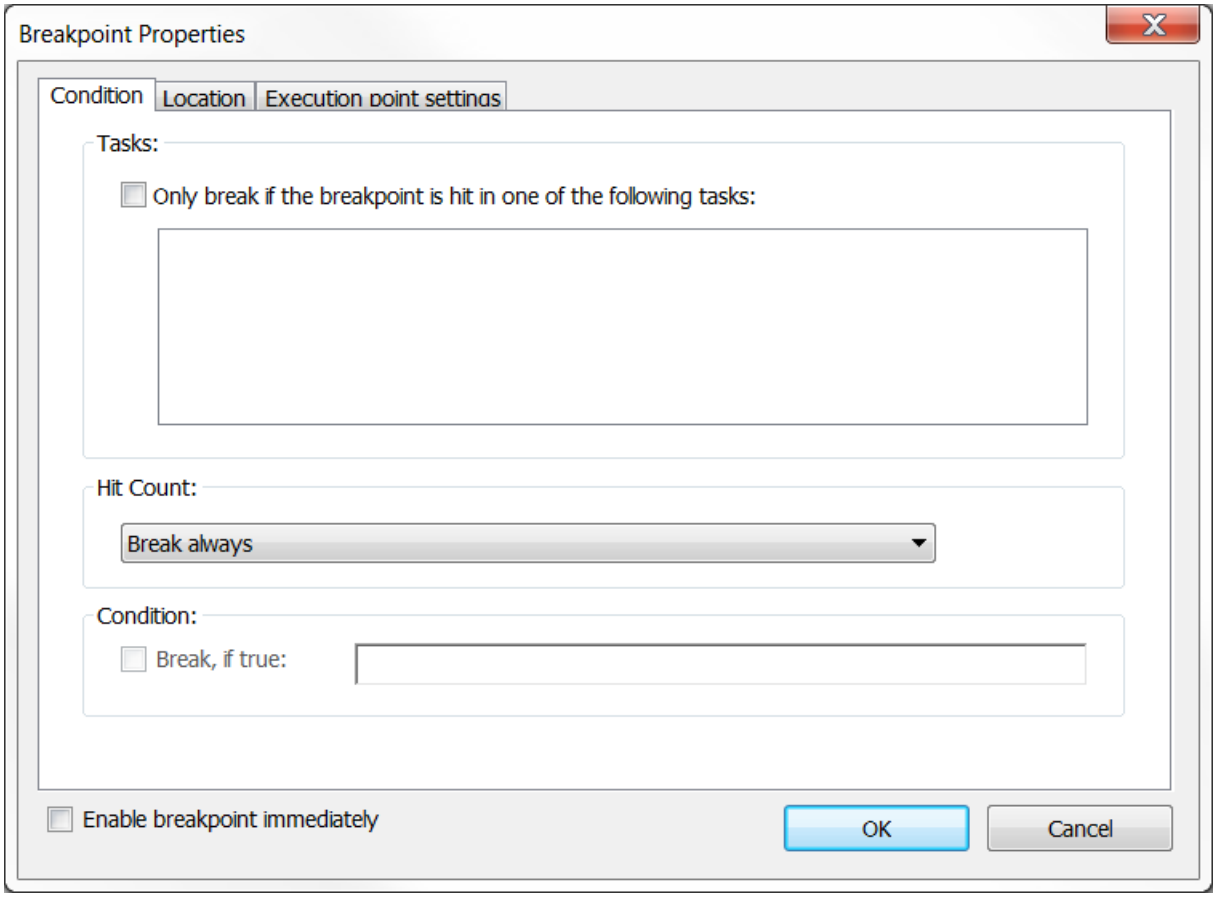
- 命令：切换断点 [▶ 867]
- PLC 菜单：命令：断点 [▶ 878]
- PLC 文档：使用断点 [▶ 195]

断点属性对话框

立即启用断点	断点已激活。 断点未激活。如要稍后激活，可点击 **Breakpoints**（断点）视图中的 按钮。

条件选项卡

该对话框定义程序执行应在断点处停止的条件。



任务

仅在以下任务之一中遇到断点时中断	☑：TwinCAT 仅在某些任务达到时评估断点。所需任务必须激活。 例如，您可以定义单个“调试任务”，从而防止在调试期间也使用该功能块的其他任务受到影响。

命中次数

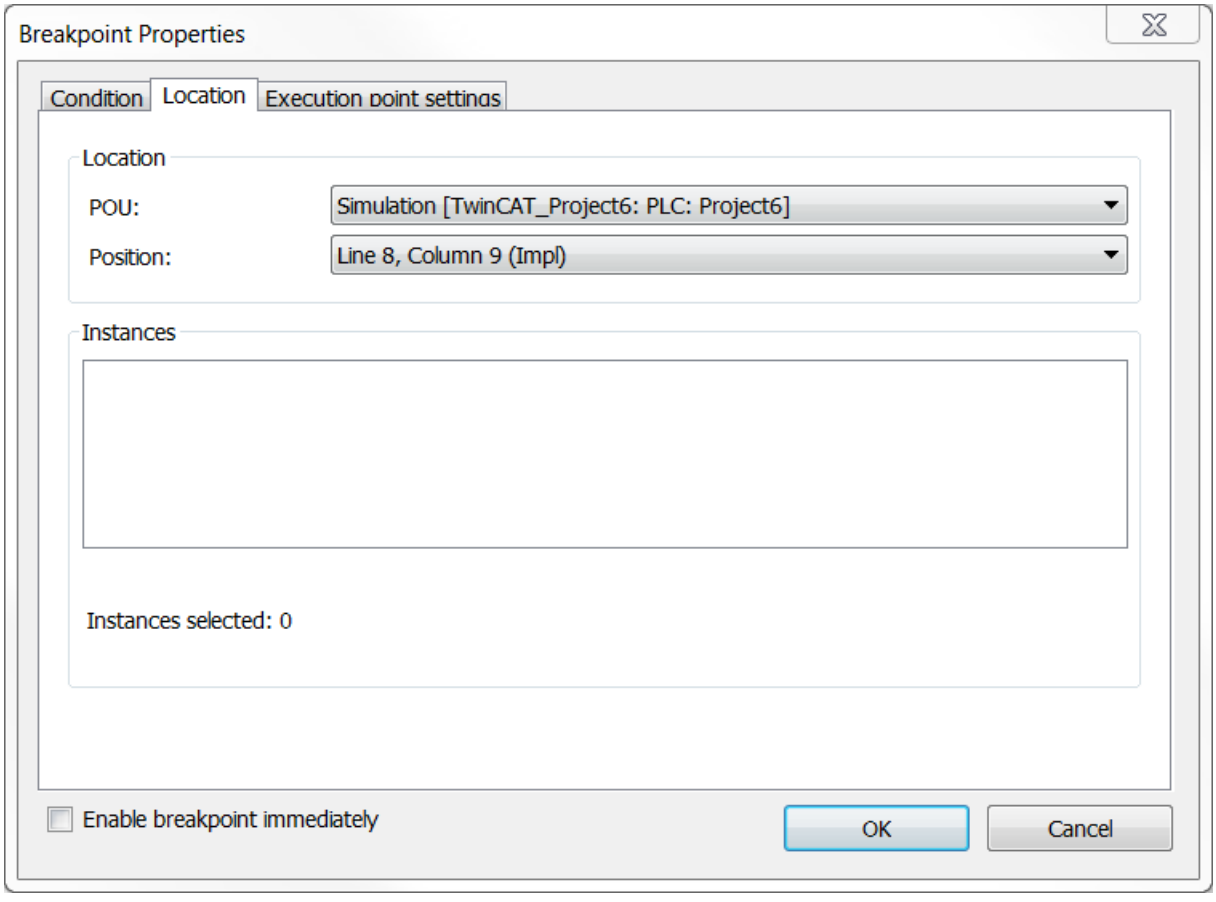
命中次数	始终断开：程序始终在此断点处停止。 替代方案：当断点如以下定义的频率命中时，程序在断点处停止（输入所需命中次数或从次数列表中选择）： • 在命中次数达到……时断开 • 在命中次数达到……倍数时断开 • 在命中次数大于或等于……时断开

条件

如果为 TRUE，断开	条件断点的定义。仅可在在线模式下输入条件。 ☑：TwinCAT 评估指定条件，并在结果为 TRUE 时在此断点处停止程序。可输入有效的 Boolean 表达式作为条件。示例：x>100、x[y]=z、a AND b、boolVar。

即使条件不为 TRUE，使用条件断点也会减慢代码执行速度。

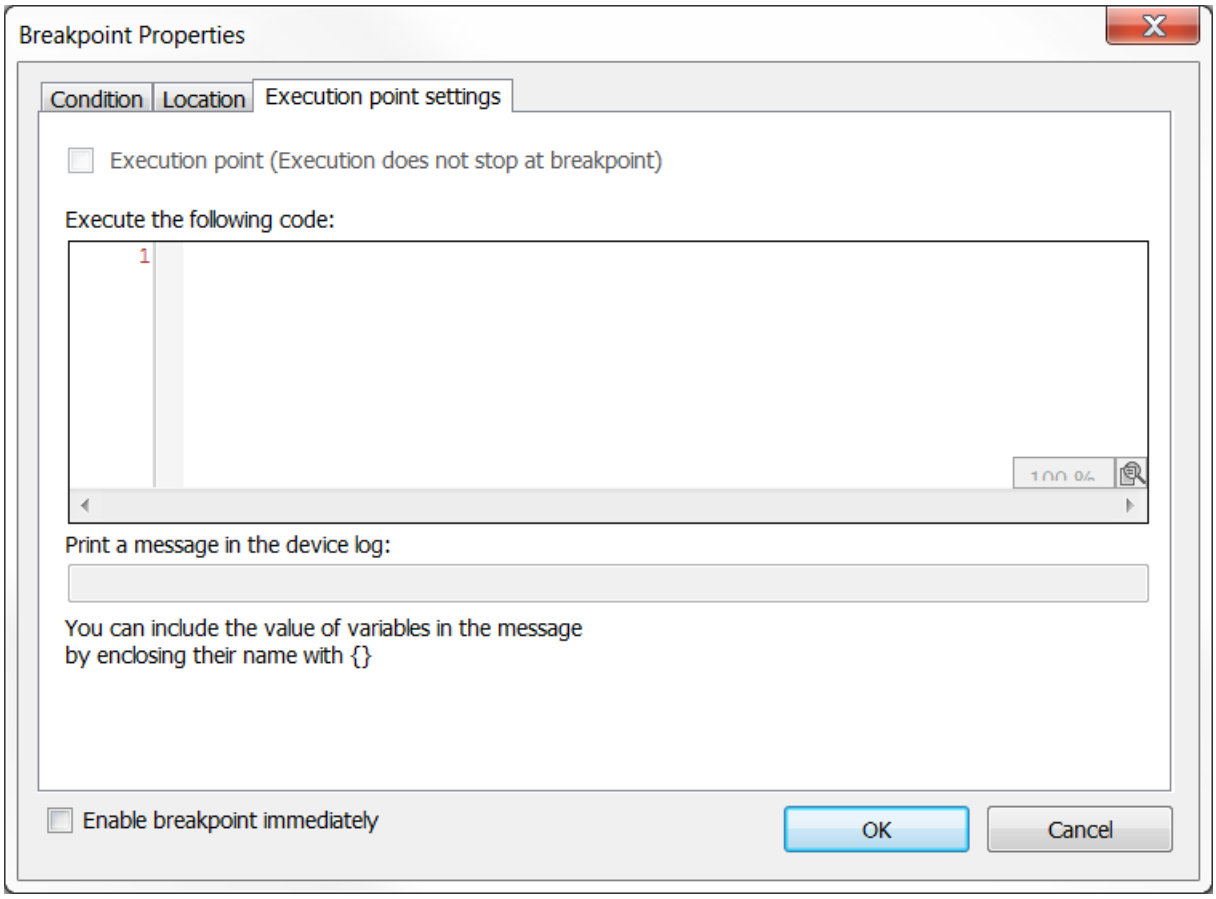
位置选项卡



POU	要在其中定位断点的活动 PLC 项目的功能块。
位置	POU 中的断点位置。以行号和列号（文本编辑器）或网络或项目编号的形式指定。
实例	对于功能块，您必须指定在实现中还是在实例中设置断点 ☑ TwinCAT 在实例中设置断点。通过此选项，您可以选择实例路径。 ☐ TwinCAT 在实现中设置断点。

执行点设置选项卡

在这里，现有的断点可以转换为执行点。



执行点（执行不会在断点处停止）	☑：断点成为执行点。此时执行不会停止，但会执行指定代码。 已启用：●，已禁用：○
执行以下代码	达到执行点时执行的代码。 不可使用循环结构（For，While）以及 IF 或 CASE 表达式。
打印设备日志中的消息	此选项不可用。

另请参见：

- PLC 文档：使用断点 [▶ 195]

17.6.2 命令：编辑断点

符号：

功能：此命令打开 **Breakpoint Properties（断点属性）**对话框。

调用：Debug（调试）菜单，**Breakpoints（断点）**视图中的按钮  （**PLC > Window（窗口） > Breakpoints（断点）**）

要求：PLC 项目处于在线模式。光标位于断点上。

另请参见：

- 命令：新断点 > 断点属性对话框 [▶ 864]
- PLC 文档：使用断点 [▶ 195]

17.6.3 命令：启用断点

符号： 

功能：此命令启用禁用的断点。

调用：Debug（调试）菜单，**Breakpoints（断点）**视图中的按钮  （**PLC > Window（窗口） > Breakpoints（断点）**）

要求：PLC 项目处于在线模式。光标位于禁用的断点上。

另请参见：

- PLC 文档：使用断点 [▶ 195]

17.6.4 命令：禁用断点

符号： 

功能：此命令禁用已启用的断点。

调用：Debug（调试）菜单，**Breakpoints（断点）**视图中的按钮  （**PLC > Window（窗口） > Breakpoints（断点）**）

要求：PLC 项目处于在线模式。光标位于已启用的断点上。

另请参见：

- PLC 文档：使用断点 [▶ 195]

17.6.5 命令：切换断点

热键：**[F9]**

功能：此命令设置断点或删除现有断点。

调用：Debug（调试）菜单

要求：PLC 项目处于在线模式。光标位于断点上。

另请参见：

- PLC 文档：使用断点 [▶ 195]

17.6.6 命令：单步跳过

符号： 

快捷键：[F10]

功能：此命令可执行程序当前所处的指令，并在编程块中的下一条指令之前停止。

调用：Debug（调试）菜单、TwinCAT PLC toolbar options（TwinCAT PLC 工具栏选项）

要求：PLC 项目处于在线模式。程序处于停止位置（调试模式）。

如果要执行的指令包含调用（来自程序、功能块实例、函数、方法或动作），则在 1 个步骤中可以完全遍历下级编程块，然后程序返回到调用位置。在下一语句（下一行代码）之前停止。

选择命令 Single step（单步），跳转到下级功能块并逐步执行。

另请参见：

- 命令：单步跳入 [▶ 868]
- PLC 文档：逐步处理程序（步进） [▶ 197]

17.6.7 命令：单步跳入

符号： 

快捷键：[F11]

功能：此命令可执行程序当前所处的指令，并在下一条指令之前停止。

调用：Debug（调试）菜单、TwinCAT PLC toolbar options（TwinCAT PLC 工具栏选项）

要求：PLC 项目处于在线模式。程序处于停止位置（调试模式）。

如果要执行的语句包含调用（来自程序、功能块实例、函数、方法或动作），则程序会跳转到该下级编程块。其代码在 1 个单独的编辑器中显示。在执行第 1 条指令后，程序会在下一条指令之前停止。然后，新的当前停止位置就会出现在被调用的编程块中。

选择命令 Step over （单步跳过），留在当前活动的编程块中并一步完成调用。

另请参见：

- 命令：单步跳过 [▶ 868]
- PLC 文档：逐步处理程序（步进） [▶ 197]

17.6.8 命令：单步跳出

符号： 

快捷键：[Shift] + [F11]

功能：此命令可执行程序，直到下一次返回，然后停止。

调用：Debug（调试）菜单、TwinCAT PLC toolbar options（TwinCAT PLC 工具栏选项）

要求：PLC 项目处于在线模式。程序处于停止位置（调试模式）。

如果当前停止位置在下级编程块中，则程序会运行至结束。然后，程序跳回到调用编程块中的调用位置，并停在那里（在调用行中）。

如果当前停止位置在主程序中，则编程块将运行至结束。然后，程序跳回至起点（编程块中第 1 行代码处的程序起点）并停止。

另请参见：

- PLC 文档：逐步处理程序（步进） [▶ 197]

17.6.9 命令运行至光标处

符号：

功能：此命令可将程序执行到标有光标的位置。

调用：上下文菜单

要求：PLC 项目处于在线模式。程序处于停止位置（调试模式）。您已经用光标标记任何编程块中的任何 1 行代码。

在 1 个步骤中执行当前停止位置和光标位置之间的指令。然后，在光标位置停止执行，光标位置成为下一个停止位置。请注意，您放置光标的代码行已到达，但并未执行。

另请参见：

- PLC 文档：逐步处理程序（步进） [▶ 197]

17.6.10 命令：显示下一语句

符号：

功能：此命令可显示将在下一步执行的程序语句。

调用：Debug（调试）菜单

要求：PLC 项目处于在线模式。程序处于停止位置。保持位置在您看不到的代码行中。

该命令会使当前停止位置的窗口（在代码中为黄色，并用符号 标记）进入活动状态，并显示停止位置。如果您打开多个编辑器，而保持位置隐藏在处于非活动状态的编辑器中，这将非常有用。

另请参见：

- PLC 文档：逐步处理程序（步进） [▶ 197]

17.6.11 命令设置下一语句

符号：

功能：此命令可决定下一步执行哪个语句。

调用：上下文菜单

要求：PLC 项目处于在线模式。程序处于停止位置（调试模式）。您已经用光标标记任何编程块中的任何 1 行代码

标有光标的代码行将成为当前停止位置，而不执行跳转到的语句或中间语句。

另请参见：

- PLC 文档：逐步处理程序（步进） [▶ 197]

17.7 TwinCAT

17.7.1 命令激活配置

符号：

功能：此命令可启用新配置。先前的旧配置将被覆盖。

调用：菜单 **TwinCAT、TwinCAT XAE Base toolbar options**（TwinCAT XAE 基础工具栏选项）

在执行该命令后出现的确认窗口中，您可以设置是否应该为 TwinCAT 项目中的所有 PLC 项目激活 **Autostart boot project**（自动运行启动项目）设置。

另请参见：

- 命令：激活启动项目
-

17.7.2 命令：重启 TwinCAT 系统

符号： 

功能：此命令在运行模式下启动 TwinCAT。

调用：菜单 **TwinCAT、TwinCAT XAE Base Toolbar Options（TwinCAT XAE 基础工具栏选项）**

17.7.3 命令：重启 TwinCAT（配置模式）

符号： 

功能：此命令在配置模式下启动 TwinCAT（配置模式）。

调用：菜单 **TwinCAT、TwinCAT XAE Base Toolbar Options（TwinCAT XAE 基础工具栏选项）**

17.7.4 命令：重新载入设备

符号： 

功能：此命令加载创建的 I/O 设备。

调用：菜单 **TwinCAT、TwinCAT XAE Base Toolbar Options（TwinCAT XAE 基础工具栏选项）**

17.7.5 命令：扫描

符号： 

功能：此命令启动设备扫描。系统搜索可用的 I/O 设备、连接“盒”以及总线模块和 IP-Link 扩展模块（如果适用）。

调用：菜单 **TwinCAT、TwinCAT XAE Base Toolbar Options（TwinCAT XAE 基础工具栏选项）**

要求：在 **Solution Explorer（解决方案资源管理器）**中的 TwinCAT 项目结构中选择“I/O”对象。

17.7.6 命令：切换自由运行状态

符号： 

功能：此命令将查找到的 I/O 设备设置为自由运行模式。这意味着，例如，总线端子可将 I/O 通道设置（写入）为特定状态，而无需处于活动状态的 PLC 项目或其他触发任务。

调用：菜单 **TwinCAT、TwinCAT XAE Base Toolbar Options（TwinCAT XAE 基础工具栏选项）**

要求：系统当前处于配置模式。

如果目标系统之前处于运行模式，则必须先执行一次命令 Reload Devices（重新加载设备），然后才能将设备 I/O 驱动程序设置为自由运行状态。

17.7.7 命令：显示在线数据

符号：

功能：此命令连接所选目标系统，并在相应视图中显示目标系统上启用的参数值和设置。

调用：菜单 TwinCAT、TwinCAT XAE Base Toolbar Options（TwinCAT XAE 基础工具栏选项）

17.7.8 命令：选择目标系统

功能：下拉列表，选择控制应用的目标设备。

调用：TwinCAT XAE Base Toolbar Options（TwinCAT XAE 基础工具栏选项）

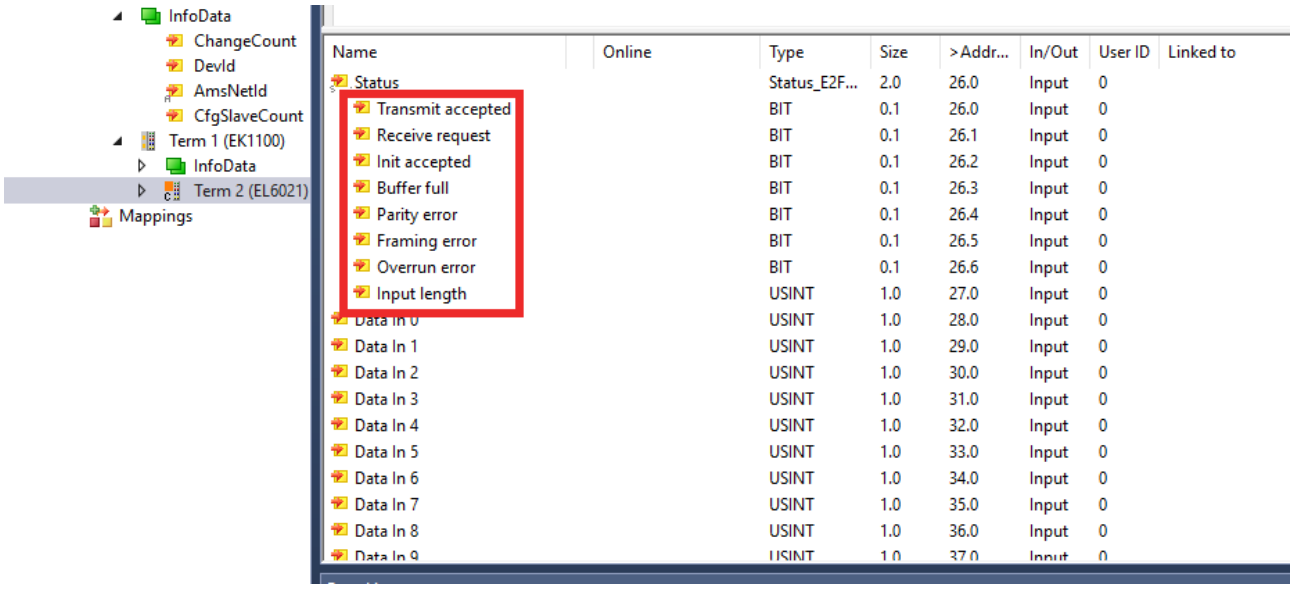
选择 <Local> 可将控制代码直接加载至编程设备的本地运行时。如果希望选择其他目标设备，请从下拉列表中选择 Choose Target System（选择目标系统）。

17.7.9 命令：显示子项

符号：

功能：此命令在设备概览视图中显示元素的子元素及其属性和值。命令可以启用或禁用。该命令不涉及 TwinCAT 项目树中元素的表示。

调用：菜单 TwinCAT, TwinCAT Base XAE toolbar options（TwinCAT, TwinCAT Base XAE 工具栏选项）



17.7.10 命令软件保护

符号：

功能：此命令可打开 Software Protection（软件保护）对话框。

调用：TwinCAT 菜单

在 Software Protection（软件保护）对话框中，您可以定义 TwinCAT 项目的安全性和用户设置。

有关安全性和用户设置的更多信息，请参见软件保护文档。

17.7.11 命令隐藏禁用项目

符号： 

功能：此命令可以使禁用的对象在整个项目树中不可见并再次可见。通过这种方式，可以将显示限制为活动的对象，从而提高项目树内的清晰度。

调用：菜单 **TwinCAT、TwinCAT XAE Base toolbar options**（TwinCAT XAE 基础工具栏选项）

17.8 PLC

17.8.1 窗口

17.8.1.1 显示内存视图命令

符号： 

功能：此命令可打开**内存视图**。在这里，您可以以在线模式显示控制器的内存转储。在该视图中，您可以配置应显示哪个应用程序和哪个区域的内存。

调用：菜单“**PLC > Window**”（PLC > 窗口）

要求：原则上，控制器可支持该功能。至少加载 1 个应用程序，且该应用程序处于在线模式。

应用程序因不小心的更改而崩溃

您可以覆盖内存视图中显示的字节，并将更改传输到控制器。TwinCAT 不会检查是否允许进行更改。不小心的更改会导致应用程序崩溃！

您可以指定内存转储的绝对起始地址，例如通过输入指针变量的地址值。或者，也可以通过指定区域和偏移量来指定相对于控制器内存区域的起始地址。在内存视图中，可以导航到相邻的内存段，并将内存转储保存在文件中。可以通过十六进制格式覆盖数据，并将更改传输到控制器。工具栏上的命令可以帮助您完成这些操作。

例如，如果在进行故障排除时逐步执行应用程序，TwinCAT 会不断更新内存转储视图。

窗口的一般结构：

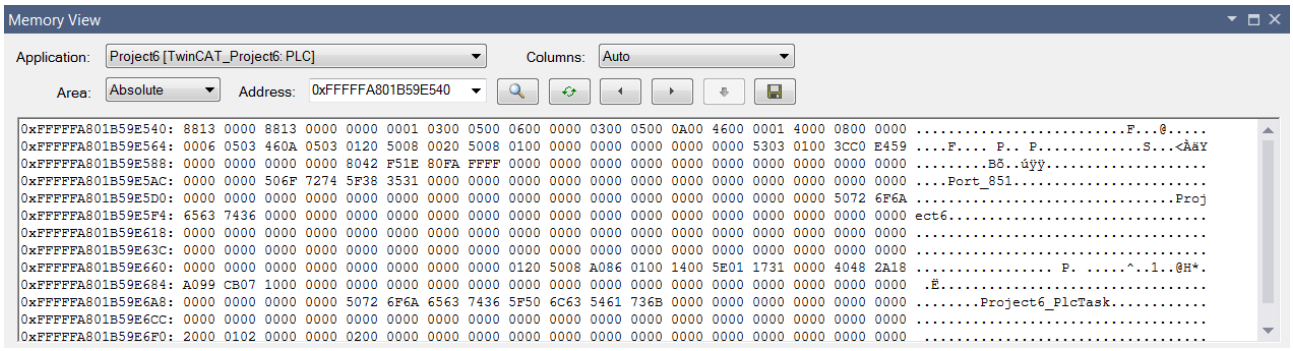
- 内存转储会在左侧显示内存的十六进制地址。
- 中间以十六进制格式显示数据，每列显示 2 个字节。
- 右侧以文本形式显示数据。不属于可显示字符的数据显示为“.” （点）。

提示：如果将光标放在十六进制数据中，可以右键单击（**复制地址**命令）这些数据，将这些数据的地址复制到剪贴板中。

对**内存视图**的解释说明如下。接下来将介绍以下用例：

- 通过变量来设置内存转储的起始地址
- 通过指定内存区域来设置内存转储的起始地址
- 编辑数据
- 对活跃应用程序进行内存检查

内存视图



Application（应用程序）	选择要显示内存视图的 PLC 项目。必须通过此项目登录控制器。不一定必须是“活跃的 PLC 项目”。
Area（区域）	• Absolute（绝对值）：内存直接完全寻址。地址位于其旁边的输入字段中。 • Area <i>（区域 <i>）：控制器的内存区域，从区域 0 开始。不显示专用于代码的内存区域。
Address（地址）	核心转储的绝对起始地址 要求：在“Area”（区域）中选择“Absolute”（绝对值）。
Offset（偏移）	所选内存区域的地址偏移量，以字节为单位，例如 0x0200、16#0200 或十进制数 512 要求：在“Area”（区域）中选择内存区域，例如区域 0。TwinCAT 会提供所有当前所用内存区域以供选择。不显示专用于代码的内存区域。
	查找变量地址：显示用于选择 IEC 变量的输入助手。如果已选择变量，TwinCAT 将通过变量地址预设起始地址。
	加载/更新内存视图
	显示上一内存段：导航至上一内存段
	显示下一内存段：导航至下一内存段
	**注意 TwinCAT 不会检查是否允许进行更改。如果不小心进行更改，可能会导致应用程序崩溃** 加载对 PLC 的更改：TwinCAT 将新数据传输至控制器。 要求：已在内存视图中覆盖了 1 个或多个字节。
	保存存储内容至文件：显示“Save memory content as binary file”（以二进制文件保存存储内容）对话框。选择 1 个位置。
列	内存转储十六进制格式的列数，位于窗口中间。选择“Auto”（自动）后，列数会根据窗口大小进行调整。

通过变量来设置内存转储的起始地址

✓ 您已将应用程序加载到控制器，并且已登录该应用程序。您想将某一特定变量的地址指定为内存转储的起始地址。

1. 选择命令“View → Show Memory View”（视图 → 显示内存视图）。
   ⇨ 内存视图打开。
2. 在“Area”（区域）中输入变量地址。可以直接输入地址，例如从指针变量的值中输入，也可以使用输入助手  选择变量。

⇨ “Absolute ”（绝对值）选项会自动设置。

⇨ TwinCAT 以变量地址周围的绝对区域显示内存转储。

通过指定内存区域来设置内存转储的起始地址

✓ 您已将应用程序加载到控制器，并且已登录该应用程序。您想将某一特定内存区域的起始地址和偏移量指定为内存转储的起始地址。

1. 选择命令“**View → Show Memory View**”（视图 → 显示内存视图）。
   ⇨ 内存视图打开。
2. 例如，在“**Area**”（区域）中选择“**Area 0**”（区域 0）选项。可选区域的列表将视设备而定。
3. 在“Offset”（偏移）中输入所需的起始地址偏移量。例如：“512”。
   提示：必须填写该字段。如果不需要偏移，请输入 0。

⇨ TwinCAT 显示内存区域 0 + 512 的内存转储。

编辑数据

您可以覆盖内存视图中显示的字节，并将更改传输到控制器。TwinCAT 不会检查是否允许进行更改。如果不小心进行更改，可能会导致应用程序崩溃！

✓ 您已将应用程序加载到控制器，并且已登录该应用程序。

1. 执行上文“通过指定内存区域来指定起始地址”的说明中的第 1 - 3 步。
2. 在内存视图中，点击数据十六进制格式的 1 个字节并输入新值。
   ⇨ 覆盖先前值的新值以红色字体显示。
3. 点击  按钮。

⇨ TwinCAT 将该数据传输至控制器。

对活跃应用程序进行内存检查

请参见：“检查活跃应用程序内存”命令 [▶ 874]

17.8.1.2 “检查活跃应用程序内存”命令

功能：该命令可对活跃应用程序的内存区域进行检查。需对内存进行各种检查。结果会显示在信息窗口中。

调用：PLC > Window（PLC > 窗口）菜单

要求：至少加载 1 个应用程序，且该应用程序处于在线模式。

对活跃应用程序的内存区域进行以下检查：

- BOOL 变量是否有定义值。
- STRING 和 WSTRING 变量是否已终止。
- 枚举变量是否具有有效值。
- 子区域类型的变量是否具有有效值。
- 常量块类型（STRUCT、ARRAY 等）的变量与其初始化值相比，是否保持不变。
- 函数指针地址是否具有内部一致性。
- 项目中的函数指针是否指向由 TwinCAT 3 PLC 管理的内存。是否已排除指向库块或外部块的函数指针；可将这些函数指针存储在外部内存中。
- 指向某种类型的指针会指向这种类型的内存位置。
- 指针是否指向地址或根据类型分级进行对齐、或为 0 或 -1 的内存位置。
- 接口变量和函数表格指针是否具有内部一致性。
- 编译块的代码是否保持不变。
  注意：带有断点的编译块不在检查范围内，因为会在运行时对这些块的代码进行操作。

检查结果将显示在消息窗口中。若违反检查标准，将会显示警告。

如果双击某个警告，则会显示**内存**视图，相关位置会以橙色突出显示。也可以通过“显示内存视图命令 [▶ 872]”打开**内存**视图。

可要求使用“memory_check”属性进行检查。

17.8.1.3 命令：监视列表 <n>

符号： 

功能：此命令打开视图 **Watch List <n>（监视列表 <n>）**。您可以使用项目中的变量填充监视列表，以便能够在单个视图中以在线模式监控、强制或写入这些变量的值。n 可以为 1、2、3、4，表示最多可以配置四个监视列表。

调用：菜单 **PLC > Window（PLC > 窗口）**

另请参见：

- PLC 文档：使用监视列表 [▶ 209]

17.8.1.4 命令：监视所有强制执行

符号： 

功能：此命令可打开 **Watch all forces**（监视所有强制执行）视图，这是特殊形式的监视列表。

调用：PLC > Window（PLC > 窗口）菜单

要求：在离线或在线模式下打开 PLC 项目。

该视图包含列表中当前准备强制执行的 PLC 项目的所有变量以及 PLC 项目的所有强制执行变量。在列表中，您可以执行在其他监视列表中可能执行的动作。

显示所有强制执行

以表格形式显示已经强制执行或准备强制执行的应用程序的所有变量

表达式	变量名
数据类型	变量的数据类型
值	变量的当前强制值
准备值	为强制执行准备的值
在周期开始时被覆盖的值	对于输入，在执行用户代码之前，实际值已被强制值覆盖。因此，这就是实际值。 对于输出，这是强制值。
在周期结束时被覆盖的值	对于输出，这是在周期中计算出的值。但是，在周期结束时，该值会被强制值覆盖。 对于输入，这是强制值。

此外，**Unforce**（取消强制）选择菜单包含以下命令：

- **Unforce and keep all selected values**（取消强制并保留所有选定值）：对于列表中的所有选定条目，变量将被设置为强制值，并取消强制。
- **Unforce and restore all selected values**（取消强制并保存所有选定值）：对于列表中的所有选定条目，变量将被重置为强制之前的值，并取消强制。

另请参见：

- PLC 文档：强制和写入变量 [▶ 198]
- PLC 文档：使用监视列表 [▶ 209]

17.8.1.5 命令：交叉引用表

符号： 

功能：此命令打开 Cross Reference List（交叉引用列表）视图。

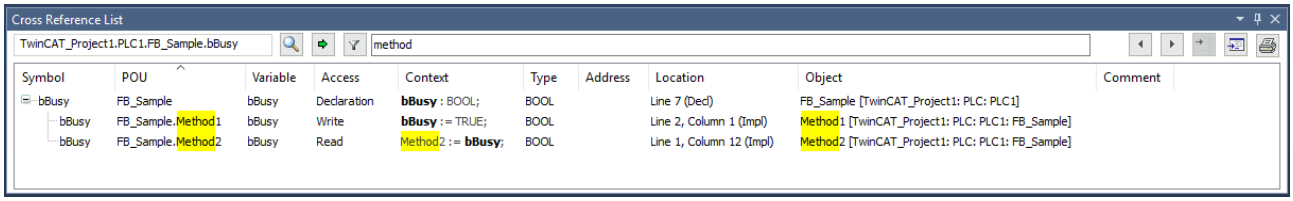
调用：菜单 PLC > Window（PLC > 窗口）

交叉引用列表视图

此视图显示项目中符号的交叉引用列表。符号可以是变量、POU（程序、功能块、函数）或用户特定的数据类型（DUT）。

交叉引用列表基本上提供 2 种类型的搜索：

- 文本搜索：输入符号名称可显示项目中与该名称相关的所有符号的交叉引用。如果找到多个具有相同名称的符号，则可通过上下文菜单将显示范围限制为单个声明。
- 声明搜索：通过输入助手或输入限定路径可以选择符号，例如 MAIN.nVar。之后，即使存在具有相同名称的其他符号，也只会显示该符号的使用位置。



工具栏

名称 （输入字段）	符号名称（变量名、功能块名称、DUT 名称） 可能的条目： • 通过输入助手（ 按钮）选择声明的符号 • 手动输入符号名称。 通过 按钮或 [Enter] 键触发搜索。 您可以使用占位符“*”（任意数量的字符）或“?” （任意字符）与用于文本搜索的变量标识符的子字符串的组合。如果您想要搜索 IEC 地址，可使用“%”。示例：“%MW8”、“%M*” 视图 **Cross Reference List**（交叉引用列表）之外的其他选项： • 如果在编辑器中选择声明符号的名称，或者光标位于名称中，则使用上下文菜单命令 **Find All References**（查找所有引用）。 • 如果在编辑器中选择声明符号的名称，或者光标位于名称中，则自动进行。如果在项目树中选择对象，也可以进行自动搜索。 要求：**Cross Reference List**（交叉引用列表）视图已打开，TwinCAT 选项 **Automatically list selection in cross reference view**（在交叉引用视图中自动选择列表）（**Smart Coding**（智能编码）类别）已激活。 以下条目有效： • 变量名，简单或限定：例如，“nVar”、“MAIN.nVar” • 功能块名称：例如，“MAIN”、“FB_MyFB” • DUT 名称：例如，“ST_MySTRUCT” • 字符串与占位符“*”（任意字符）或“?”的组合 （任意字符）： 示例： “nVar*”匹配 nVar1、nVarGlob2、nVar45 等…… “nVar?” 匹配 nVar1、nVar2、nVarX 等，但不匹配 nVarGlob2、nVar45 等 • “%<IEC address>”：TwinCAT 搜索已分配给该地址的变量并直接访问内存。 示例：“%QB0”、“%Q0 := 2” 输入字符串开头和结尾的大写/小写和空格将被忽略。
	打开输入助手，以选择符号。
	查找交叉引用：执行搜索。
	定义搜索字符串的列
筛选器 （输入字段）	要在所选列中搜索的字符串 找到的位置被标记为黄色。没有该字符串的交叉引用将被隐藏。
	显示上一个交叉引用的源位置
	显示下一个交叉引用的源位置
	将结果限制为当前声明 当发现 1 个符号有多个声明时可用。限制显示为您刚刚在列表中选择的声明。
	显示所选交叉引用的源位置：焦点跳转到符号的使用位置。
	打印交叉引用列表：显示用于设置打印作业的标准对话框。

找到交叉引用表

符号	符号（变量、POU、DUT）的位置按其声明分组。声明位置构成根节点，项目中的使用点在其下方缩进。系统会显示该符号在使用时的准确表达式。 示例：如果项目中有 1 个全局变量“nVar”，且功能块中有 1 个局部声明的变量“nVar”，则在交叉引用的文本搜索后，在列表中会出现 2 个根节点条目，变量“nVar”的相应使用点在下方显示。
功能块	例如，功能块名称、DUT 名称；在任务配置中功能块调用的情况下还有任务名称。
变量	纯变量名。示例：“nVar”。
访问	在使用时访问变量的类型： 声明/读取/写入/调用。 指针特例：在搜索“nVar1”时，类型赋值 pSample := ADR(nVar1) 显示为 Write | Address（写入|地址）。原因：在搜索“nVar1”时，不显示对“pSample”的任何写入访问。通过指针变量也可以进行写入访问。
上下文	使用变量的上下文。 示例：“nVar := 1”
类型	变量的数据类型。
地址	IEC 地址，如果分配至变量。 示例：“AT%QB0”
位置	相关 POU 编辑器内的使用位置：例如，行号、网络编号、声明部分或实现部分。 示例：“第 1 行，第 1 列（Impl）”或“第 9 行（Decl）”。
对象	POU 名称 + 方括号内使用点的完整路径。 示例：“MAIN [TwinCAT_SampleProject: PLC: SamplePLCProject]”
注释	注释，如果存在于变量声明中。

搜索将返回项目中以及已插入未编译库中的所有位置。

交叉引用列表的上下文菜单中的命令

显示源代码位置	打开受影响的功能块并标记使用位置：对于根条目，标记声明；对于其下面的子条目，标记各自的使用位置。或者，您也可以双击 1 行。
将结果限制为当前声明	如果找到多个声明，则将结果显示限制为所选符号声明。
全部展开	列表显示所有单个位置。
全部折叠	列表只显示所有位置的根节点。

另请参见：

-
-
-
- PLC 文档：使用交叉引用列表查找出现 [▶ 145]

17.8.1.6 命令：断点

符号： 

功能：此命令打开 **Breakpoints（断点）**视图。

调用：菜单 **PLC > Window（PLC > 窗口）**

断点视图

该视图提供了应用的所有已定义断点的概述。视图中提供了所有断点命令。

当前断点表

应用	从列表中选择所需的 PLC 项目。
POU	包含断点的功能块名称。
位置	POU 中的断点位置 • 文本编辑器：行号和列号 • 图形编辑器：网络或元素编号 如果是功能块，则“(Impl)”表示断点在功能块的实现中，而不是在实例中。
实例路径	断点位置的完整对象路径。
任务	断点有效执行的任务名称。如果没有限制，则显示“(all)”。
条件	• 始终中断：未定义其他启用条件。断点始终处于活动状态。 • 布尔表达式。表达式必须返回 TRUE 才能使断点处于活动状态。
计数次数条件	指示断点应生效的时间（其中依赖于计数次数）。
当前计数次数	指定在执行期间断点已经传递（“计数”）的频率。

工具栏

	新断点 (对应于 **Debug（调试）** 菜单中的命令 命令：新断点 [▶ 863])	打开 **Breakpoint Properties（断点属性）** 对话框
	清除断点	删除断点。 不要将此命令与禁用命令混淆。
	启用/禁用断点 (对应于 **Debug（调试）** 菜单中的命令 命令：启用断点 [▶ 867]和命令：禁用断点 [▶ 867])	在“启用”和“禁用”状态之间切换断点或执行点。 • 断点已启用 • 断点已禁用 • 执行点已启用 • 执行点已禁用 与 **Clear breakpoint（清除断点）** 相反，禁用的断点仍保留在列表中，而且可以再次启用。
	属性	打开 **Breakpoint Properties（断点属性）** 对话框以编辑断点参数。在在线模式下，可以在此处将断点转换为执行点。
	转到源代码位置	打开相应功能块的在线视图。光标位于断点位置。
	删除所有断点	删除应用的所有断点和执行点。列表已清空。不要与禁用混淆！
	启用所有断点	启用所有当前禁用的断点和执行点。
	禁用所有断点	禁用所有当前启用的断点和执行点。这些点保留在列表中，且可以重新启用。

另请参见：

- 命令：新断点 > 断点属性对话框 [▶ 864]
- 命令：切换断点 [▶ 867]
- PLC 文档：使用断点 [▶ 195]

17.8.1.7 命令：调用栈

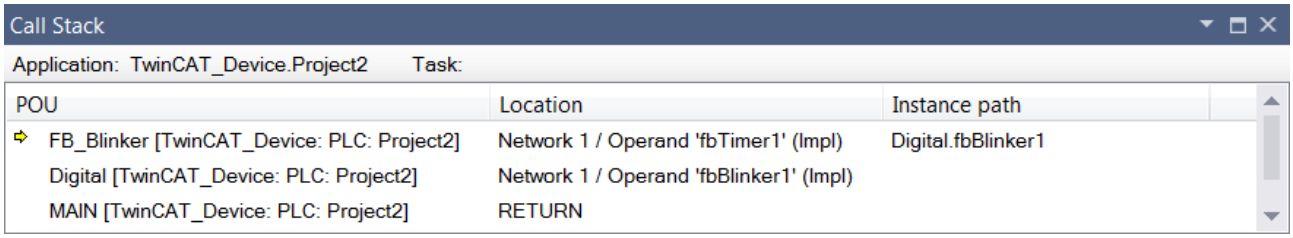
符号： 

功能：此命令打开 **Call Stack（调用堆栈）**视图。

调用：菜单 **PLC** > **Window（PLC > 窗口）**

调用堆栈视图

如果要逐步运行程序，该视图十分有用。视图显示当前到达的位置以及完整的调用路径。



应用	控制当前达到的程序块的活动 PLC 项目名称。
任务	控制当前达到的程序块的任务名称。

POU	程序执行所在程序块的名称。 列表中第一行描述了当前的执行位置。通过黄色箭头进行标记。如果此位置在另一功能块调用的功能块中，则在第二行中描述调用位置。如果调用端被另一功能块调用，则在第三行中描述此调用位置，依此类推。
位置	位于程序执行所在程序块内的位置 • 文本编辑器的行号和列号 • 图形编辑器的网络或元素编号
实例路径	执行程序的实例。

如果未使用任何调试功能，则调用堆栈也可在脱机模式和正常在线模式下使用。在这种情况下，其包含在逐步执行期间显示的最后位置，但是以“灰色”字体显示。

与 **Call Stack（调用堆栈）**视图相反，**Call Tree（调用树）**视图随时提供关于功能块的调用信息。

另请参见：

- PLC 文档：使用断点 [▶ 195]

17.8.1.8 命令：调用树

符号：

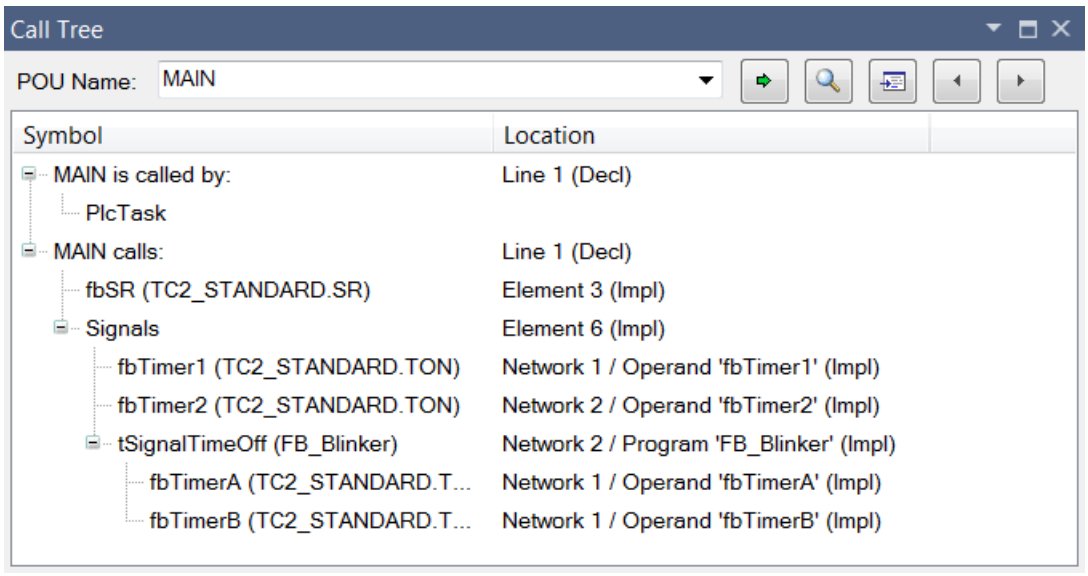
功能：此命令打开 **Call Tree（调用树）**视图。

调用：菜单 **PLC** > **Window（PLC > 窗口）**

调用树视图

编译应用前，可以随时使用调用树。它是功能块的调用端和调用的静态表示，可以显式指定。这意味着调用树始终包含两个根节点，在这些根节点下，相应的调用序列显示为连续缩进条目。在此树视图中可以快速识别递归调用。

功能块 (2) MAIN 的调用树 (1) 示例：



- (3) 节点“<function block name> 被调用：”
- (4) 节点“<function block name> 调用：”

POU 名	通过从其他视图拖动或使用 按钮手动输入程序块的名称。 选择列表包含最后输入的功能块名称。

工具栏和键盘操作

< 查找功能块	TwinCAT 搜索“功能块名称”中指定的功能块，并显示其调用端和调用。
从输入助手获取功能块	**Input Assistant（输入助手）**对话框出现，用于选择功能块调用或实例调用。选择后，调用树将自动更新。
显示所选功能块的源代码位置	TwinCAT 跳转到程序源代码中所用功能块的位置。
显示下一功能块的源代码位置 显示上一功能块的源代码位置	调用树中的选择跳转到调用结构中的下一或上一功能块。同时，在相关编辑器中打开相应的源代码位置。 双击“调用树”中的条目也可打开相应的源代码位置。

显示调用树

位置	对于调用树中的根节点：功能块声明（“Decl”）的行号。 对于根节点下的调用端和调用：取决于实现语言、行号、列号、位置网络号。

当前在树中所选条目的上下文菜单

全部折叠	除两个根节点外，调用树中的扩展条目将被折叠。
显示源代码位置	TwinCAT 跳转到程序源代码中所用功能块的位置。
设置为新根节点	调用树中的所选条目以“功能块名称”显示。树将自动适应新的根节点。

相比始终为功能块提供调用信息的静态调用树，**Call Stack（调用堆栈）**视图用于在逐步处理程序期间的即时信息。调用堆栈始终显示当前到达位置的完整调用路径。

17.8.1.9 命令在线更改内存保留设置

功能：此命令可打开 Online Change Memory Reserve（在线更改内存保留）视图。

调用：菜单 **PLC > Window**（PLC > 窗口）。

该视图用于为功能块的在线更改配置内存保留。

浏览应用程序	• 搜索所选 PLC 项目中的功能块，并在 **Function blocks**（功能块）区域中显示它们 • 在再次编译 PLC 项目之后，更新 **Function blocks**（功能块）区域 • 在在线更改之后，更新 **Function blocks**（功能块）区域
打开的 TwinCAT 项目的 PLC 项目选择列表	选择要在该视图中显示和/或编辑其功能块的 PLC 项目

功能块：

全部	显示所选 PLC 项目的所有功能块。
无内存保留	显示内存保留为 0 字节的所有功能块。
<Memory reserve> 字节	显示具有在 **memory reserve**（内存保留）中定义的字节数的所有功能块。
关于功能块的信息 在选择用于配置内存保留的功能块时，也可以进行多项选择。	
功能块	功能块的名称
大小	功能块的大小 功能块实例的大小 以字节为单位的规格
实例数	项目中的功能块的实例数
内存保留	显示功能块的每个实例的内存保留量
所有实例的额外内存	**实例数**与**内存保留**的乘积
剩余内存保留	每个功能块实例可用作保留的字节数

设置：

内存保留（以字节为单位）	所选功能块的内存保留的输入字段。 以字节为单位的规格 要求：控制器上尚没有 PLC 项目，或者您已通过点击 **Allow editing**（允许编辑）区域中的 **Allow**（允许）按钮允许更改内存保留。
适用于选择	**memory reserve (in bytes)**（内存保留（以字节为单位））将被分配给功能块，**Memory reserve**（内存保留）表格列也将更新。 如果选择多个功能块，则会将输入的值分配给每个功能块。 如要更新 **Size**（大小）、**Number of instances**（实例数）、**Additional memory for all instances**（所有实例的额外内存）和 **Remaining memory reserve size**（剩余内存保留大小）列，首先要选择 **Create > Create**（创建 > 创建），然后点击 **Browse application**（浏览应用程序）按钮。

启用编辑：

启用	输入字段 **Memory reserve (in bytes)**（内存保留（以字节为单位））变为可编辑。 该按钮变为 **Editable**（可编辑）。

信息：

FB 的数量	PLC 项目中功能块的总数
所有实例的额外内存	PLC 项目的所有功能块实例的内存保留总和 以字节为单位的规格

另请参见：

- PLC 文档：对 PLC 项目进行编程 > 为在线更改配置内存保留 [▶ 122]

17.8.1.10 命令 PLC 书签

符号：

功能：此命令可打开视图 **Bookmarks**（书签）。

调用：PLC > Window（PLC > 窗口）菜单

上一个书签	跳转到所选行上一行表格中显示的书签，并在编辑器中打开相应的 POU。
下一个书签	跳转到所选行下一行表格中显示的书签，并在编辑器中打开相应的 POU。
	从表格和相应的 POU 中删除所选书签。

项目书签列表，包含书签、对象和位置信息：

书签	TwinCAT 按照编号升序分配的书签名称：Bookmark_0、Bookmark_2 等。 当您选中书签并点击进入该字段时，它会变成可编辑状态，您可以更改书签名称。
对象	设置书签的 POU 的名称和项目路径
位置	书签在 POU 中的位置 例如：第 3 行，第 1 列（Impl） （Impl）：在 POU 的实现部分中 （Decl）：在 POU 的声明部分中

您可以通过拖放来更改书签的顺序。

如果您双击 1 行，则 TwinCAT 会在编辑器中打开相应的对象并跳转到此书签。

另请参见

- 命令上一个书签 [▶ 886]
- 命令下一个书签 [▶ 885]
- 设置和使用书签 [▶ 146]（“PLC”文档）

17.8.2 核心转储

17.8.2.1 命令生成核心转储

功能：此命令可令 TwinCAT 首先检查目标系统上是否已有核心转储文件。

- 如果目标系统上有核心转储文件，则 TwinCAT 可以为您将该文件加载到项目目录中。对于是否从目标系统加载核心转储文件的询问，有 3 种不同的回应方式。

- 是：如果目标系统的核心转储文件与当前登录的 PLC 项目相匹配，则 TwinCAT 会从目标系统将核心转储文件加载到项目目录中。随后退出 PLC 项目并使用 命令加载核心转储 [▶ 884]，您可以打开该文件。
- 否：在项目目录中将会生成 1 个新的核心转储文件。这样做的前提条件是，PLC 项目当前处于断点或出现异常错误。
- 取消：中止生成核心转储文件。

- 如果目标系统上没有可用的核心转储文件，则 TwinCAT 将使用项目目录中的当前 PLC 项目数据启动新文件的生成。这样做的前提条件是，PLC 项目当前处于断点或出现异常错误。

生成的核心转储文件会直接保存在 PLC 项目目录下：<PLC_project_name>.<PLC_project_GUID>.core

调用： 菜单 PLC > 核心转储

要求： PLC 项目处于在线模式。

在目标系统上自动生成核心转储

如果在目标系统上运行的 PLC 项目当前未登录开发环境，则在出现异常错误时，运行时系统会在目标系统上自动生成核心转储。该文件默认位于目标系统的启动文件夹中（默认情况下，对于 TC3.1.4026.0 以下版本：*C:\TwinCAT\3.1\Boot\Plc*；对于 TC3.1.4026.0 及以上版本：*C:\ProgramData\Beckhoff\TwinCAT\3.1\Boot\Plc*）。如果需要，您可以修改自动创建核心转储的存储路径，有关更多信息，请参见利用核心转储进行错误分析 [▶ 227]。

借助**命令 Generate core dump**（生成核心转储），可将该核心转储文件从目标系统自动加载到本地项目目录中。还可以将核心转储文件从目标系统复制到开发计算机中。

因此，使用命令加载核心转储 [▶ 884]显示转储，可用于（后续的）错误分析。

核心转储只能与相关的编译信息文件一起使用

如果您存档或保存核心转储文件，请注意，您必须有相关的项目和相关的编译信息文件（*.compileinfo 文件，例如，在创建项目时存储在“_CompileInfo”文件夹中），才能加载核心转储。否则，TwinCAT 以后将无法使用转储。

另请注意此处 设置选项卡 [▶ 859] 上的设置选项。借助 **Core Dump**（核心转储）设置，您可以配置是否将可能位于项目目录下的核心转储文件与可用的编译信息文件一起保存在 TwinCAT 文件存档中。

另请参见：

- PLC 文档：运行时的 PLC 项目 > 利用核心转储进行错误分析 [▶ 227]
- 命令加载核心转储 [▶ 884]

17.8.2.2 命令加载核心转储

功能： TwinCAT 在项目目录中搜索核心转储文件。

- 如果 TwinCAT 在项目目录中找到核心转储文件，则系统会询问您是想要加载该核心转储还是想要浏览转储文件。
- 如果 TwinCAT 在项目目录中找不到核心转储文件，则您可以浏览其他转储文件。

加载到项目中会导致显示 PLC 项目的在线视图，其中包含在创建核心转储时的 PLC 项目的状态。您可以查看其中包含的变量值。调用树也可供使用。

调用： 菜单 PLC > 核心转储

要求： 应用程序处于离线模式。

核心转储视图只能通过 命令关闭核心转储 [▶ 885] 命令再次关闭。“Logout”（退出）命令在此视图中无效。

核心转储只能与相关的编译信息文件一起使用

如果您存档或保存核心转储文件，请注意，您必须有相关的项目和相关的编译信息文件（*.compileinfo 文件，例如，在创建项目时存储在“_CompileInfo”文件夹中），才能加载核心转储。否则，TwinCAT 以后将无法使用转储。

另请注意此处 设置选项卡 [▶ 859] 上的设置选项。借助 **Core Dump**（核心转储）设置，您可以配置是否将可能位于项目目录下的核心转储文件与可用的编译信息文件一起保存在 TwinCAT 文件存档中。

另请参见：

- PLC 文档：运行时的 PLC 项目 > 利用核心转储进行错误分析 [▶ 227]
- 命令生成核心转储 [▶ 883]
- 命令关闭核心转储 [▶ 885]

17.8.2.3 命令关闭核心转储

功能：此命令可关闭当前在开发环境中打开的 PLC 项目的核心转储视图。

调用：菜单 PLC > 核心转储

要求：PLC 项目处于离线模式，您已将核心转储文件加载到项目中。

另请参见：

- PLC 文档：运行时的 PLC 项目 > 利用核心转储进行错误分析 [▶ 227]

17.8.3 PLC 书签

17.8.3.1 命令启用/禁用书签

符号： 

功能：此命令可设置或删除当前位置的书签。

调用：**PLC** > **PLC Bookmarks**（PLC > PLC 书签）菜单

要求：POU 在编辑器中打开，光标位于程序行中。

另请参见

- 书签 [▶ 146]（文档“PLC”）

17.8.3.2 命令下一个书签

符号： 

功能：此命令可在 **Bookmarks**（书签）视图和项目中跳转到下一个书签，并打开相应的 POU。书签跳转的顺序与 **Bookmarks**（书签）视图表格中的书签顺序相对应。

调用：

- 菜单 **PLC** > **PLC Bookmarks**（PLC > PLC 书签）
- **Bookmarks**（书签）视图中的按钮  **Next bookmark**（下一个书签）

要求：

- 1 个项目已打开
- **Bookmarks**（书签）视图已打开

另请参见：

- 命令 PLC 书签 [▶ 883]
- 命令下一个书签（活动编辑器） [▶ 886]
- 书签 [▶ 146]（文档“PLC”）

17.8.3.3 命令上一个书签

符号： 

功能：此命令可在 **Bookmarks**（书签）视图和项目中跳转到上一个书签，并打开相应的 POU。书签跳转的顺序与 **Bookmarks**（书签）视图表格中的书签顺序相对应。

调用：

- 菜单 **PLC** > **PLC Bookmarks**（PLC > PLC 书签）
- Bookmarks（书签）视图中的按钮  Previous bookmark（上一个书签）

要求：

- 1 个项目已打开
- **Bookmarks**（书签）视图已打开

另请参见：

- 命令 PLC 书签 [▶ 883]
- 命令上一个书签（活动编辑器） [▶ 887]
- 书签 [▶ 146]（文档“PLC”）

17.8.3.4 命令清除所有书签

符号： 

功能：此命令可删除已打开项目的所有书签。

调用：PLC > **PLC Bookmarks**（PLC > PLC 书签）菜单

要求：POU 在编辑器中打开，光标位于 POU 中。

另请参见：

- 命令清除所有书签（活动编辑器） [▶ 887]
- 书签 [▶ 146]（文档“PLC”）

17.8.3.5 命令下一个书签（活动编辑器）

符号： 

功能：此命令可跳转到活动编辑器中的下一个书签。

调用：PLC > **PLC Bookmarks**（PLC > PLC 书签）菜单

要求：POU 在编辑器中打开，光标位于 POU 中

另请参见：

- 命令下一个书签 [▶ 885]
- 书签 [▶ 146]（文档“PLC”）

17.8.3.6 命令上一个书签（活动编辑器）

符号： 

功能：此命令可跳转到活动编辑器中的上一个书签。

调用：PLC > PLC Bookmarks（PLC > PLC 书签）菜单

要求：POU 在编辑器中打开，光标位于 POU 中

另请参见：

- 命令上一个书签 [▶ 886]
- 书签 [▶ 146]（文档“PLC”）

17.8.3.7 命令清除所有书签（活动编辑器）

符号： 

功能：此命令可删除活动编辑器中的所有书签

调用：PLC > PLC Bookmarks（PLC > PLC 书签）菜单

要求：POU 在编辑器中打开，光标位于 POU 中。

另请参见：

- 命令清除所有书签 [▶ 886]
- 书签 [▶ 146]（文档“PLC”）

17.8.4 命令：下载

功能：此命令编译活动的 PLC 项目，然后将其下载到控制器。

调用：PLC 菜单

要求：PLC 项目处于在线模式。

通过此命令，TwinCAT 执行语法检查并生成程序代码。此代码加载到控制器上。另外，TwinCAT 还在项目目录中生成编译日志 <projectname>.<devicename>. <application ID>. compileinfo。

在下载期间，除了持久型变量，所有变量都将重新初始化。

Login（登录）命令说明解释了登录和加载时可能出现的情况。

如果在试图加载 PLC 项目的同时，已有该项目的相同版本位于控制器上，则会显示以下消息：“The program is unchanged. Application was not loaded（程序未更改。未加载应用）”。TwinCAT 不会将项目加载到 PLC 上。

加载时，将在 **Output（输出）**视图中显示已执行操作的日志（创建代码、执行初始化等）。此外，还显示关于存储区域、代码大小、全局数据和分配存储器的信息。为清晰起见，与在线更改相比，不再列出已更改的功能块。

另请参见：

- 命令登录 [▶ 889]

17.8.5 命令在线更改

功能：此命令可用于启动当前活动的 PLC 项目的在线更改。TwinCAT 仅将已在控制器上运行的 PLC 项目的修改部分重新加载到控制器中。

调用： PLC 菜单

要求： PLC 项目处于在线模式。

在 Clean All（全部清除）和 Clean（清除）命令后无法进行在线更改。清除过程将删除每次生成代码时自动保存并构成在线更改基础的编译信息（编译日志）。

⚠ 警告

由于设备或系统的意外行为而造成的财产和人身损害

在线更改会修改正在运行的应用程序，并且不会导致重新启动。根据不同的受控设备，可能会损坏设备或工件，或者可能会危及人员的健康和生命。

- 确保新程序代码实现受控系统的所需行为。

项目特定的初始化

在执行在线更改时，由于设备保持其状态，因此不会执行项目特定的初始化（归位等）。为此，新程序代码可能没有所需的效果。

下载代码中的重大更改

如果在线更改导致下载代码发生重大变化（例如，需要切换变量），则会出现 1 个对话框，提供有关效果的信息，并允许您取消在线更改。

快速在线更改

对于微小更改（例如，在实现部分中，不需要切换变量），可执行“快速在线更改”。在这种情况下，仅会编译和重新加载修改后的功能块。特别是，在这种情况下不会生成初始化代码。这也意味着不会生成带“init_on_onlchange”属性的初始化变量代码。通常，这不会产生任何影响，因为该属性一般用于初始化带地址的变量，但变量在快速在线更改时无法更改其地址。

如要确保将 init_on_onlchange 属性应用于整个应用程序代码，可使用 no_fast_online_change 编译器定义停用 PLC 项目的快速在线更改。为此，可在 PLC 项目属性的 Compile [▶ 844]（编译）类别中插入定义。

“init_on_onlchange”属性对单个 FB 变量没有影响

属性“init_on_onlchange” [▶ 746] 仅适用于功能块的全局变量、程序变量和局部静态变量。

如要在在线更改期间重新初始化功能块，务必使用属性来声明功能块实例。对于功能块中的单个变量，不会对属性进行评估。

指针变量

指针保留了上一个周期的值。如果指针指向已通过在线更改调整大小的变量，并因此在内存中移动，则指针变量不再返回变量的正确位置。确保在每个周期中重新分配指针。

在可能产生意外后果的在线更改过程中，TwinCAT 会在 Details（详细信息）对话框中列出已更改的接口、受影响的变量以及已生成新代码的所有功能块。如果存储位置发生变化，则会显示对话框，指明指针可能存在问题。

另请参见：

- PLC 文档：对 PLC 项目进行编程 [▶ 60]

什么会阻止在线更改？

在某些 TwinCAT 动作之后，将无法再在控制器上进行在线更改。之后，始终需要下载 [▶ 233]项目。1 个典型的例子是动作 Clean（清除）和 Clean all（全部清除），它们会删除在上次下载期间存储的编译信息。不过，也有一些“正常”的编辑动作会导致您在下次登录时无法进行在线更改。以下动作可能会阻止在线更改：

检查函数	激活或删除检查函数 [▶ 151]（CheckBounds、CheckRange、CheckDiv 等）。 更改检查函数的接口（包括插入或删除局部变量）。
任务配置	更改配置设置。
项目设置	类别编译 [▶ 844]：更改 **Settings**（设置）部分中的编译器定义（替换常量） 类别：通用 [▶ 841]：最小化 TwinCAT 文件中的 ID 更改。
功能块属性	更改选项 External implementation（外部实现）
功能块	更改功能块的基本块（EXTENDS [▶ 177]FB_Base），或者插入或删除这样的基本块。 更改接口列表（IMPLEMENTS [▶ 183]I_Sample）。例外：在列表末尾添加新接口。
数据类型	将变量的数据类型从 1 种用户定义的数据类型改为另一种用户定义的数据类型（例如，从 TON 改为 TOF）。 将数据类型从用户定义的数据类型改为基本数据类型（例如，从 TON 改为 TIME）。 注意：作为 1 种变通方法，可始终在更改数据类型的同时更改变量的名称。结果，变量被初始化为新变量，旧变量被删除。然后可以进行在线更改。

另请参见：

- 执行在线更改 [▶ 231]
- 命令登录 [▶ 889]

17.8.6 命令登录

符号： 

功能：此命令可将编程系统（所选 PLC 项目）与目标系统（控制器）连接，从而建立在线操作。在目标系统上创建 PLC 项目的实例并加载。

调用：PLC 菜单或 **TwinCAT PLC toolbar options**（TwinCAT PLC 工具栏选项）或 **Solution Explorer**（解决方案资源管理器）中的 PLC 项目对象（<PLC project name> 项目）的上下文菜单

要求：PLC 项目无错误，目标系统处于运行模式。

可能的登录情况：

- 控制器尚不存在 PLC 项目：系统将提示您确认下载。
- PLC 项目已在控制器上，且自上次下载后未更改。无需进一步互动即可登录。
- PLC 项目已在控制器上，但自上次下载后已更改。
  系统将提示您选择以下选项之一：
  - 在线更改后登录（请参见“命令在线更改 [▶ 887]”部分中的说明）
  - 下载后登录
  - 无更改登录

  此时，您还可以更新控制器上的启动项目。
- 控制器上已存在未知版本的 PLC 项目。系统将询问您 TwinCAT 是否应该替换它。
- PLC 项目的 1 个版本已在控制器上运行。系统将询问您 TwinCAT 是否应该登录并覆盖当前正在运行的 PLC 程序。
- 控制器上的 PLC 程序当前停止在断点处。您已退出并更改程序：TwinCAT 警告您，如果在线更改或下载，PLC 将完全停止。即使有多个任务并且只有其中 1 个受断点影响，也会发生这种情况。

登录前编译项目

如果 PLC 项目自上次更改后尚未编译，则 TwinCAT 会在登录前编译项目。此操作对应于命令 **Compile in logged-out state（在注销状态下编译）**。

如果在编译时发生错误，则会显示一个消息对话框。错误显示在 **Error List（错误列表）** 视图中。然后，您可以决定是否希望在不将程序加载到控制器的情况下登录。

另请参见：

- 命令构建 PLC 项目 [▶ 863]

登录时出错

如果在登录控制器时发生错误，TwinCAT 将中断加载过程并显示错误消息。错误对话框可以显示错误详情。如果发生异常错误并且日志消息中包含文本 *SOURCEPOSITION*，则可以使用命令“在编辑器中显示”以在编辑器中显示相关功能。光标跳转至导致错误的行。

输出关于加载过程的信息

如果 TwinCAT 在登录时将项目加载到控制器上，则消息窗口中将显示以下信息：

- 生成代码的大小
- 全局数据的大小
- 控制器上的存储器要求
- 受影响的功能块列表（用于在线更改）

在在线模式下，无法更改设备或模块的设置。如要更改设备参数，必须退出 PLC 项目。但是，根据总线系统，可能存在一些可以在在线模式下更改的特殊参数。

TwinCAT 分别针对在线和脱机模式存储视图配置。此外，关闭无法在操作模式下使用的视图。为此，登录时视图可能会自动更改。

17.8.7 命令：开始

符号：

键盘快捷键：[F5]

功能：此命令开始执行程序。

Call（调用）：菜单 **PLC**，**TwinCAT PLC Toolbar Options（PLC，TwinCAT PLC 工具栏选项）**

要求：PLC 项目处于在线模式。

通过 **PLC** 菜单调用命令时，会影响当前启用的 PLC 项目。

17.8.8 命令：停止

符号：

键盘快捷键：[Shift] + [F5]

功能：此命令停止执行程序。

Call（调用）：菜单 **PLC**，**TwinCAT PLC Toolbar Options（PLC，TwinCAT PLC 工具栏选项）**

要求：PLC 项目处于在线模式。

通过 **PLC** 菜单调用命令时，会影响当前启用的 PLC 项目。

17.8.9 命令：登出

符号：

功能：此命令终止开发系统与目标系统（控制器或模拟设备）之间的连接，从而切换到脱机模式。

Call（调用）：菜单 **PLC，TwinCAT PLC Toolbar Options（PLC，TwinCAT PLC 工具栏选项）**

17.8.10 命令冷重置

符号：

功能：此命令可将活动的 PLC 项目的所有变量重置为其初始化值，PERSISTENT 和 RETAIN 变量除外。

调用：菜单 **PLC、TwinCAT PLC toolbar options**（TwinCAT PLC 工具栏选项）

要求：PLC 项目处于在线模式。

活动的 PLC 项目的所有变量都将被重置，PERSISTENT 变量和 RETAIN 变量除外。情况与控制器上已加载应用程序（“冷启动”）启动时的情况相同。

执行此命令后，仍会启用在重置活动的 PLC 项目之前启用的 PLC 程序断点。执行此命令后，之前禁用的断点仍会禁用。

如果您在程序停在断点处时选择此命令，则系统将询问您是否终止当前周期。或者，TwinCAT 立即执行重置。但是，并非所有运行时系统都能够在不终止当前周期的情况下执行重置。

重置后，您必须通过 **Start**（启动）命令启动 PLC 程序。

另请参见：

- PLC 文档：剩余变量 - PERSISTENT，RETAIN [▶ 630]
- PLC 文档：重置 PLC 项目 [▶ 201]
- 命令：初始值复位 [▶ 891]
- 命令：开始 [▶ 890]

17.8.11 命令：初始值复位

符号：

功能：此命令将活动 PLC 项目的所有变量（包括掉电保持变量（RETAIN、PERSISTENT 变量））重置为其初始化值，并从控制器中删除应用程序。

Call（调用）：菜单 **PLC，TwinCAT PLC Toolbar Options（PLC，TwinCAT PLC 工具栏选项）**

要求：PLC 项目处于在线模式。

重置会停用程序中当前设置的断点。如果在程序停在断点处时选择此命令，系统将询问是否终止当前循环。或者，TwinCAT 立即执行重置。但是，并非所有运行时系统都能够在不终止当前循环的情况下执行重置。

另请参见：

- 命令冷重置 [▶ 891]

还请参阅有关此

剩余变量 - PERSISTENT，RETAIN [▶ 630]

重置 PLC 项目 [▶ 201]

17.8.12 命令：单循环

符号：

功能：此命令执行一个周期的活动 PLC 程序。

Call（调用）：菜单 **PLC，TwinCAT PLC Toolbar Options（PLC，TwinCAT PLC 工具栏选项）**

要求： PLC 项目处于在线模式，且程序处于程序步骤。

17.8.13 命令：流控制

符号：

功能： 此命令启用或停用流量控制。

Call（调用）： 菜单 **PLC, TwinCAT PLC Toolbar Options (PLC，TwinCAT PLC 工具栏选项)**

要求： PLC 项目处于在线模式。

主动流量控制扩展了 PLC 项目的运行时！

另请参见：

- PLC 文档：流量控制 [▶ 202]

17.8.14 命令：强制值

符号：

功能： 此命令将控制器上的变量值永久设置为预定义值。

Call（调用）： 菜单 **PLC, TwinCAT PLC Toolbar Options (PLC，TwinCAT PLC 工具栏选项)**

要求： PLC 项目处于在线模式。

⚠ 谨慎

受控系统异常行为导致的物料损坏和人身伤害

更改控制器上运行的 PLC 程序中的变量值可能会导致受控系统的异常行为。根据受控系统，可能会损坏设备和工件，或者危及人员健康和生命。

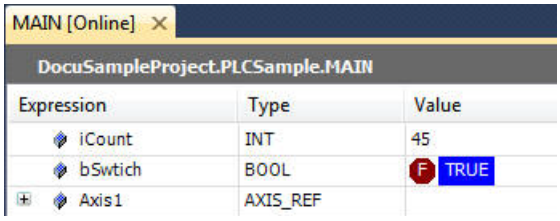
- 在强制变量值前评估可能的风险并采取适当的安全预防措施。

通过此命令，TwinCAT 将控制器上活动应用的一个或多个变量永久设置为定义值。此设置在每个处理循环的开始和结束时执行。执行顺序：1. 读取输入、2. 强制值、3. 执行代码、4. 强制值、5. 写入输出。

您可以通过以下方式准备数值

- 点击声明部分中的 **prepared value（准备值）** 字段并输入新值。对于布尔变量，通过单击字段更改值。
- 点击 FBD/LD/IL 编辑器实现部分中的内联监控字段并输入新值。
- 点击监控窗口中的 **prepared value（准备值）** 字段并输入新值。

“强制”值以 F 标记。



TwinCAT 执行强制，直到用户通过

- 命令 **Unforce values（取消强制值）** 显式取消
- 通过 **Prepare Value（准备值）** 对话框取消强制
- 退出应用

默认情况下，菜单中不包括影响 TwinCAT 项目中所有 PLC 项目的命令 **Force values for all applications（强制所有应用的值）**。

另请参见：

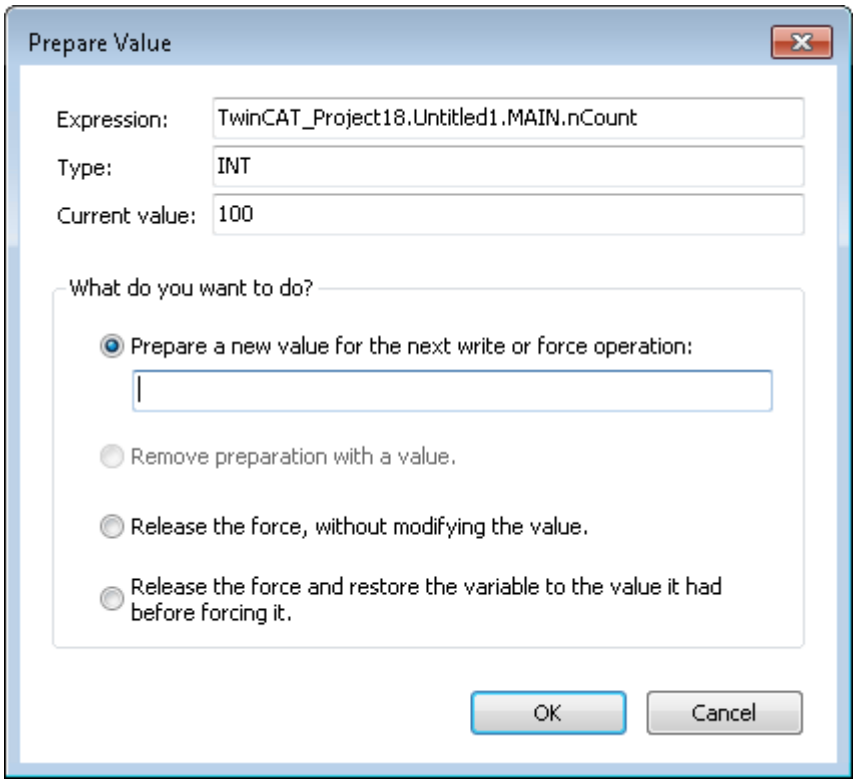
- 命令：取消强制值 [▶ 893]
- 对话框准备值 [▶ 893]
- PLC 文档：强制和写入变量值 [▶ 198]

17.8.14.1 对话框准备值

功能：此对话框用于为已强制变量准备值。TwinCAT 使用下一个强制值执行准备的操作。

调用：TwinCAT 在以下情况下打开对话框：

- 在声明部分中点击强制变量的 **prepared value（准备值）**字段
- 在实现部分中点击强制变量的 inline monitoring（内联监控）字段
- 在监控窗口中点击强制变量的 **prepared value（准备值）**字段



为下一次写入或强制操作准备新值	TwinCAT 在下一次强制操作期间写入变量的值
删除带值的准备工作	TwinCAT 删除准备值。
释放强制，而不修改值。	TwinCAT 保留强制值并终止强制。TwinCAT 以 <Unforce> 标记变量。
释放强制，并将变量恢复为强制之前的值。	TwinCAT 重置强制值并终止强制。变量以 <Unforce and restore> 进行标记。

另请参见：

- 命令：强制值 [▶ 892]

17.8.15 命令：取消强制值

符号：

功能：此命令重置所有变量的强制。变量从控制器获取其当前值。

Call（调用）：菜单 **PLC，TwinCAT PLC Toolbar Options（PLC，TwinCAT PLC 工具栏选项）**

要求：PLC 项目处于在线模式。

⚠ 谨慎

受控系统异常行为导致的物料损坏和人身伤害

更改控制器上运行的 PLC 程序中的变量值可能会导致受控系统的异常行为。根据受控系统，可能会损坏设备和工件，或者危及人员健康和生命。

- 在重置强制变量值前评估可能的风险并采取适当的安全预防措施。

默认情况下，菜单中不包括影响 TwinCAT 项目中所有 PLC 项目的命令 **Unforce values for all applications（取消强制所有应用的值）**。

另请参见：

- 命令：强制值 [▶ 892]
- PLC 文档：强制和写入变量值 [▶ 198]

17.8.16 命令：写入值

符号：

功能：此命令将控制器上的变量值设置为一次预定义值。

Call（调用）：菜单 **PLC，TwinCAT PLC Toolbar Options（PLC，TwinCAT PLC 工具栏选项）**

要求：PLC 项目处于在线模式。

⚠ 谨慎

受控系统异常行为导致的物料损坏和人身伤害

更改控制器上运行的 PLC 程序中的变量值可能会导致受控系统的异常行为。根据受控系统，可能会损坏设备和工件，或者危及人员健康和生命。

- 在写入变量值前评估可能的风险并采取适当的安全预防措施。

通过此命令，可将控制器上活动 PLC 项目的一个或多个变量设置为一次定义值。在下一个循环开始时进行一次写入。

您可以通过以下方式准备数值

- 点击声明部分中的 **prepared value（准备值）**字段并输入新值。对于布尔变量，通过单击字段更改值。
- 点击 FBD/LD/IL 编辑器实现部分中的内联监控字段并输入新值。
- 点击监控窗口中的 **prepared value（准备值）**字段并输入新值。

默认情况下，菜单中不包括影响 TwinCAT 项目中所有 PLC 项目的命令 **Write values for all applications（写入所有应用的值）**。

另请参见：

- 命令：强制值 [▶ 892]
- PLC 文档：强制和写入变量值 [▶ 198]

17.8.17 命令：显示模式 - 二进制、十进制、十六进制

功能：Display（显示）子菜单的命令可用于在在线模式下进行监控时设置值的显示格式。

调用：PLC 菜单、上下文菜单

要求： PLC 项目处于离线或在线模式。

显示格式“二进制”和“十六进制”无符号，“十进制”有符号。

另请参见：

- PLC 文档：监控值 [▶ 205]
- PLC 文档：声明编辑器 [▶ 569]

17.8.18 命令继承的显示 - 简单、结构化

TC3.1 Build 4026 及以上可用

功能： 子菜单 **Presentation of inheritance**（继承的显示）的命令可用于在在线模式下进行监控时设置功能块和结构的继承层次结构的显示格式。

调用： **PLC** 菜单、**上下文菜单**

要求： PLC 项目处于离线或在线模式。

结构化	功能块和结构的继承层次结构以树状结构显示。变量作为声明它们的功能块或结构的子节点显示。
简单	以平面列表的形式显示。

另请参见：

- PLC 文档：监控值 [▶ 205]
- PLC 文档：声明编辑器 [▶ 569]

17.8.19 命令：创建本地化模板

功能： 此命令将打开 **Create Localization Template（创建本地化模板）** 对话框。可在此处定义将项目中的哪些文本信息导出到文件格式 pot 的翻译模板。

调用： 菜单 **PLC** > **Project Localization（项目本地化）**

要求： 打开一个项目。

创建本地化模板对话框

此对话框用于选择要包含在本地化模板中的文本信息。

包括以下信息

名称	PLC 项目树中的对话框标题、对象名称等文本
标识符	变量标识符，例如：“nCounter”
字符串	例如，以下声明中的“计数”：sVar : STRING := 'count'
注释	编程块中的注释文本

位置信息	选择以上所选项目中文本类别的哪些位置应包含在翻译文件中。位置信息始终位于翻译部分的第一行。
	示例：
	`#: D:\Proj1.project\Project_Settings:1`
	`msgid „Projekteinstellungen“`
	`msgstr „“`
	• “All (全部)”：列出所有找到的文本位置。
	• “First occurrence (首次出现)”：翻译文件包括项目中第一次出现待翻译文本的位置。
	• “None (无)”

创建	该按钮可打开用于保存文件的对话框。翻译模板在 POT 翻译模板 (*.pot) 类型的文本文件中创建。每次进一步的创建过程会生成完整的新模板文件。

17.8.20 命令：管理本地化

功能：此命令打开 **Manage Localization (管理本地化)** 对话框。在对话框中，选择所需的本地化语言或项目原始版本。另外，还可以将本地化文件 *.<Language>.po 添加至项目或从项目中删除。

调用：菜单 **PLC > Project Localization (项目本地化)**

要求：打开一个项目。

管理本地化对话框

可用本地化	项目中存在的本地化文件列表。
	示例：
	`proj1-de.po`
	`proj1-en.po`
	`<Originalversion>`
	原始版本始终可用。项目仅可在原始版本中进行编辑。

添加	此按钮可打开用于从文件系统中选择其他 po 文件的文件框。
删除	此按钮可从项目中删除在左侧选择的 po 文件。
标准本地化	 当前选择的本地化成为标准本地化。条目以粗体显示。
更改本地化	使用按钮切换至当前所选的本地化。
确认	项目以文件中选定文件所提供的国家语言显示。如果选择 <Original version>，项目将以可编辑的非本地化版本显示。

17.8.21 命令：切换本地化

符号： 

功能：此命令在当前设置的项目本地化和 <Original version> 之间切换。

调用：

- 菜单 **Project (项目) > Localization (本地化)**
- **Manage Localizations (管理本地化)** 对话框中的按钮
- 工具栏中的按钮

要求：打开一个项目。在 **Manage Localizations (管理本地化)** 对话框中定义了项目的标准本地化。

另请参见：

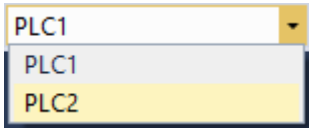
- 命令：管理本地化 [▶ 896]

17.8.22 命令：激活 PLC 项目

功能：用于选择活动的 PLC 项目的下拉列表。当前聚焦的项目会自动被设置为活动的 PLC 项目。

调用：TwinCAT PLC toolbar options（TwinCAT PLC 工具栏选项）

要求：TwinCAT 项目包含多个 PLC 项目。



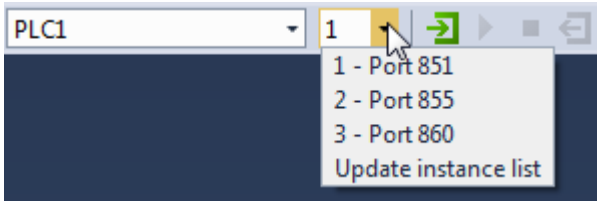
另请参见：

- 命令：激活 PLC 实例 [▶ 897]
- 命令登录 [▶ 889]

17.8.23 命令：激活 PLC 实例

功能：用于选择相应 PLC 项目的活动 PLC 实例的下拉列表。

调用：TwinCAT PLC toolbar options（TwinCAT PLC 工具栏选项）



另请参见：

- 命令：激活 PLC 项目 [▶ 897]
- 命令登录 [▶ 889]

17.9 工具

17.9.1 命令：选项

功能：此命令打开用于配置 TwinCAT 选项的 Options（选项）对话框。这些选项定义了 TwinCAT 用户界面的行为和外观。TwinCAT 将本地系统上的当前设置保存为默认设置。

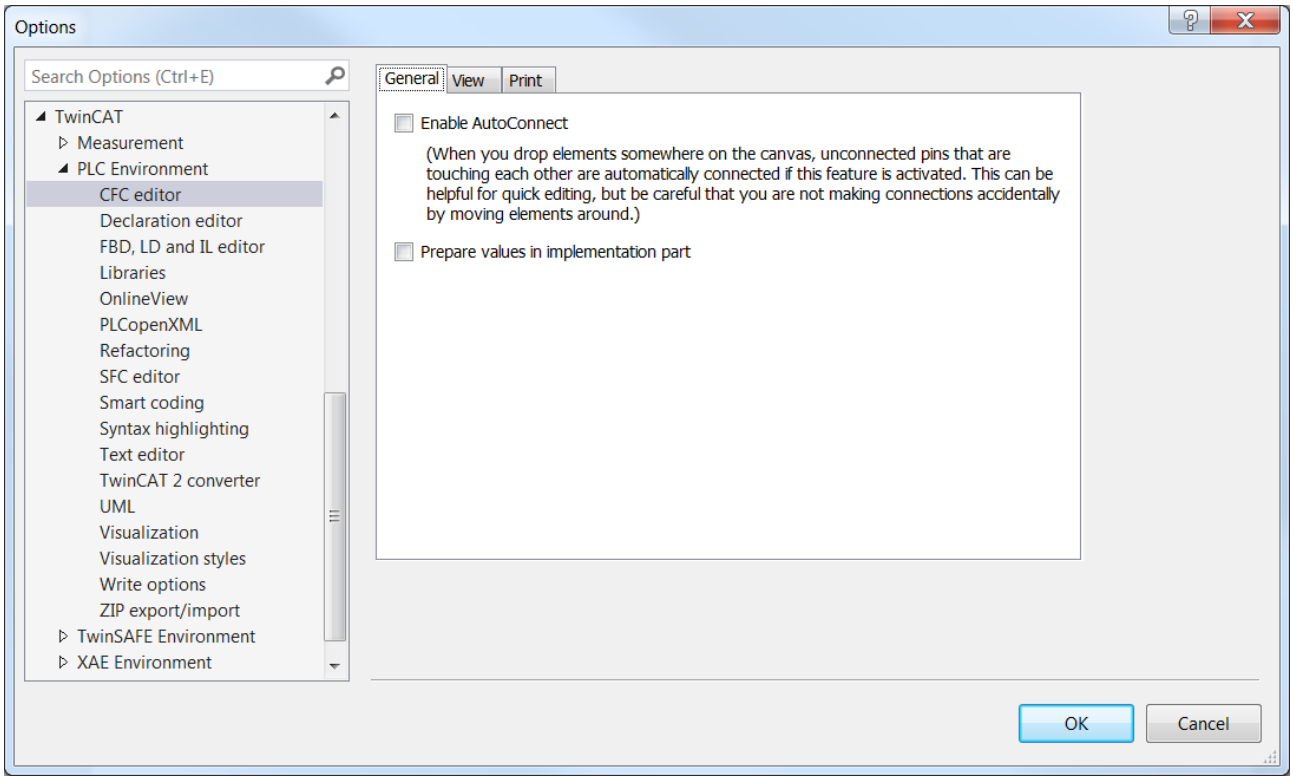
调用：菜单 Tools（工具）

17.9.1.1 对话框选项 - CFC 编辑器

功能：此对话框用于配置 CFC 编辑器中的编辑和打印设置。

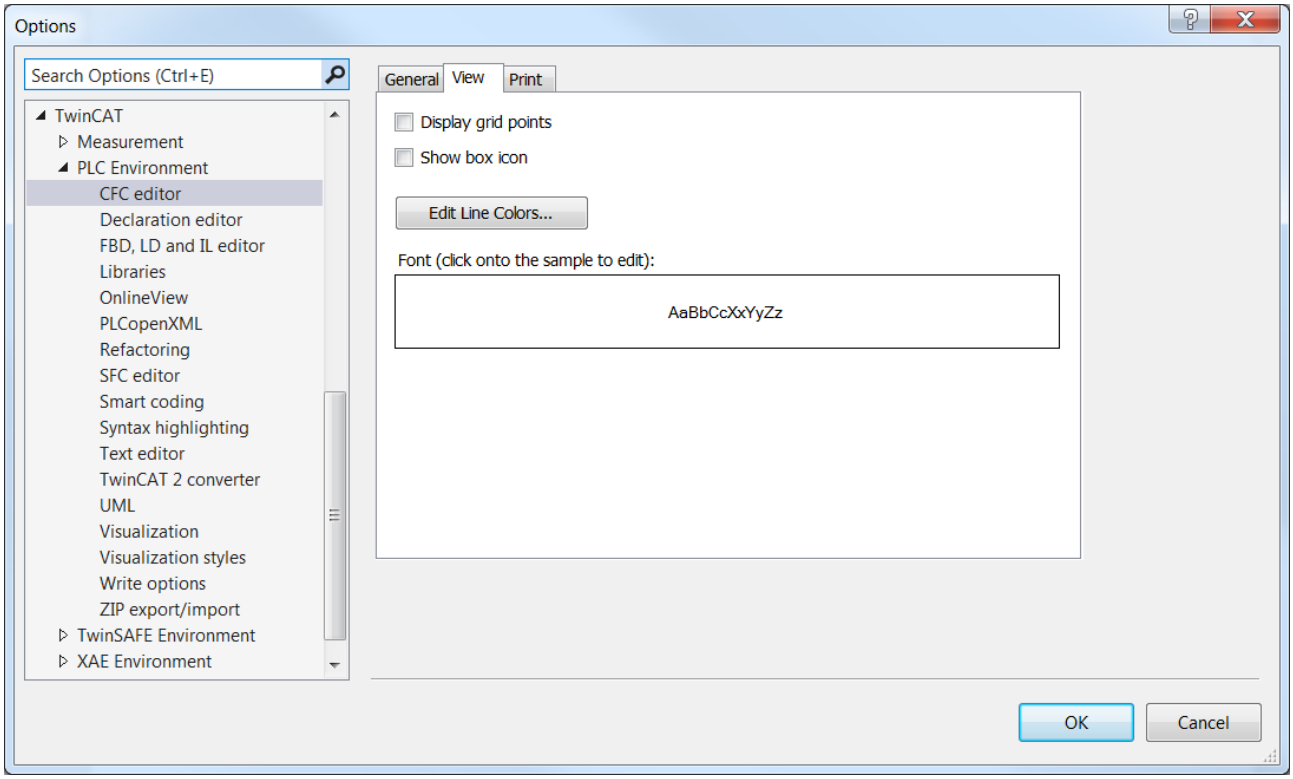
调用：TwinCAT > PLC Environment（PLC 环境）> CFC editor（CFC 编辑器）

常规选项卡



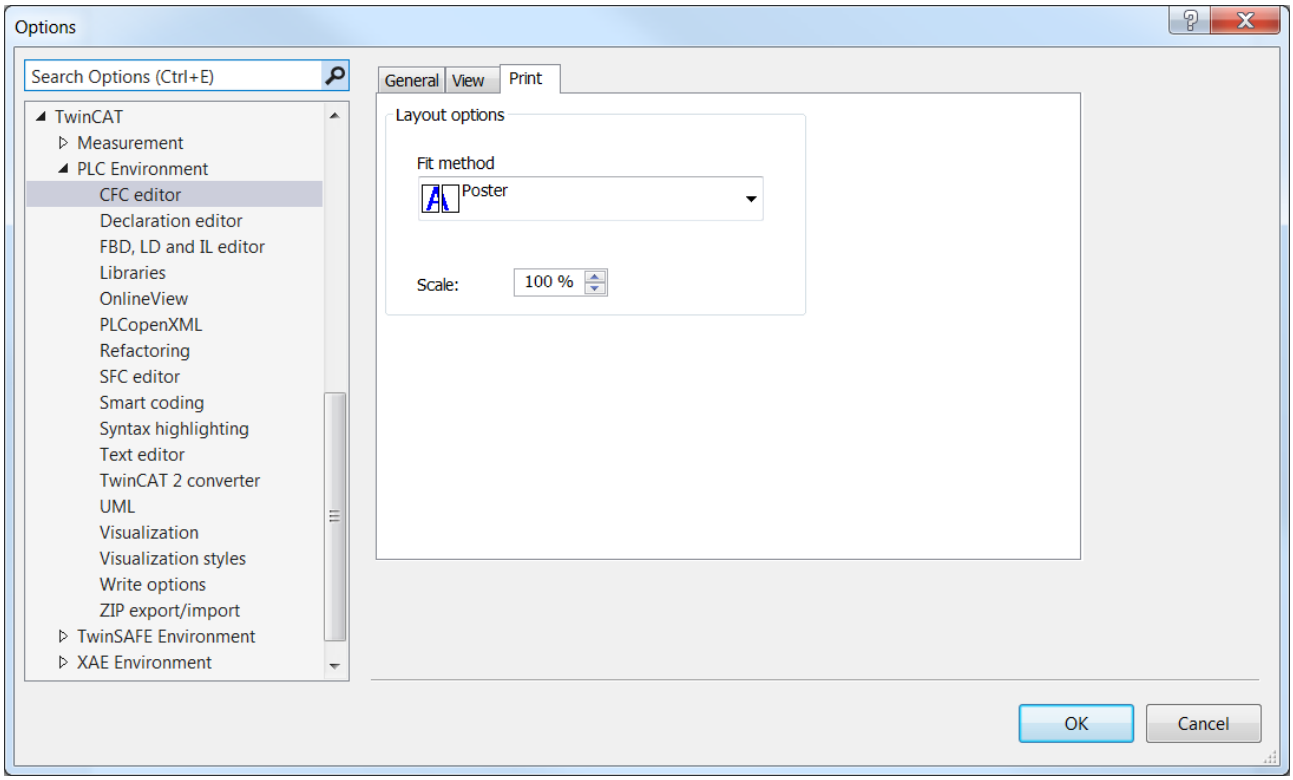
启用 AutoConnect	☑：将 CFC 元素拖动并粘贴到编辑器的工作区时，TwinCAT 会自动连接相互“接触”的未链接引脚。移动元素时，确保不要创建任何不需要的连接！
准备实现部分中的值	☑：在在线模式下，还可以在 CFC 功能块的实现部分中准备为写入和强制准备变量值。此外，TwinCAT 在尖括号中显示变量内联监控框中准备的值。

视图选项卡



显示网格点	☑：在编辑器中，网格点是可以定位元素的有效区域。
显示框图标	☑：TwinCAT 显示以符号链接至位图的 CFC 编辑器中的功能块。 要求：已经为对象属性中的功能块或功能创建链接，或者使用库进行加载。
编辑行颜色	打开 **Edit Line Colors（编辑行颜色）** 对话框，以根据当前数据类型定义连接行的颜色。在脱机和在线模式下，行以这些颜色显示，但 TwinCAT 用粗体黑色和蓝色行覆盖这些颜色以指示布尔数据流。 • 添加类型：将数据类型添加到列表中。 • 删除类型
字体	显示用于更改字体的字体和按钮。

打印选项卡



布局选项

拟合方式	页面或海报
Scale（标度）	可能值：20% - 200%

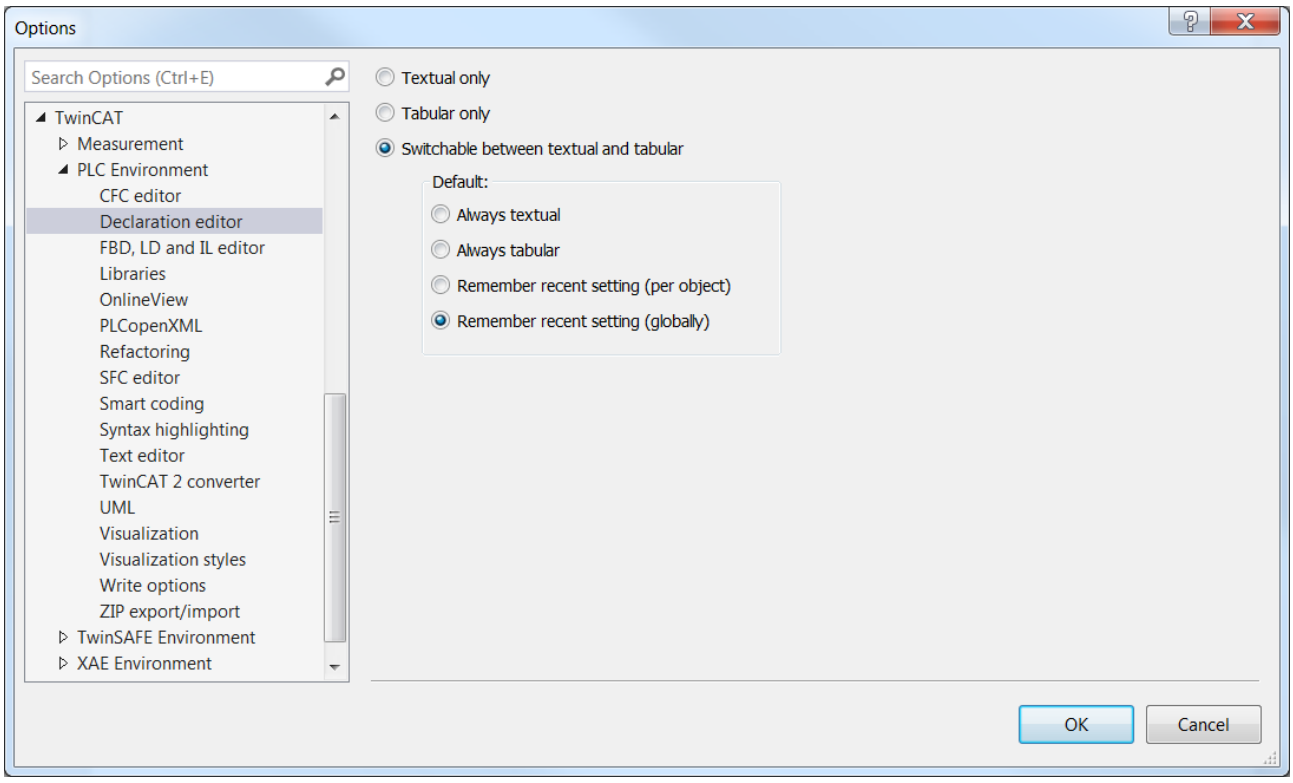
另请参见：

- PLC 文档：CFC 语言编程 [▶ 104]
- PLC 文档：编程语言及其编辑器 [▶ 569]

17.9.1.2 对话框选项 - 声明编辑器

功能：此对话框用于配置声明编辑器的显示设置。

调用：TwinCAT > PLC Environment（PLC 环境）> Declaration editor（声明编辑器）



仅文本	声明编辑器的文本视图
仅表格	声明编辑器的表格视图
可在文本与表格之间切换	声明编辑器提供两个按钮，用于在文本视图与表格视图之间切换： ：文本视图 ：表格视图 以下选项定义了打开编程对象时默认显示的视图： • 始终文本 • 始终表格 • 记住最近的设置（每个对象） • 记住最近的设置（全局）

另请参见：

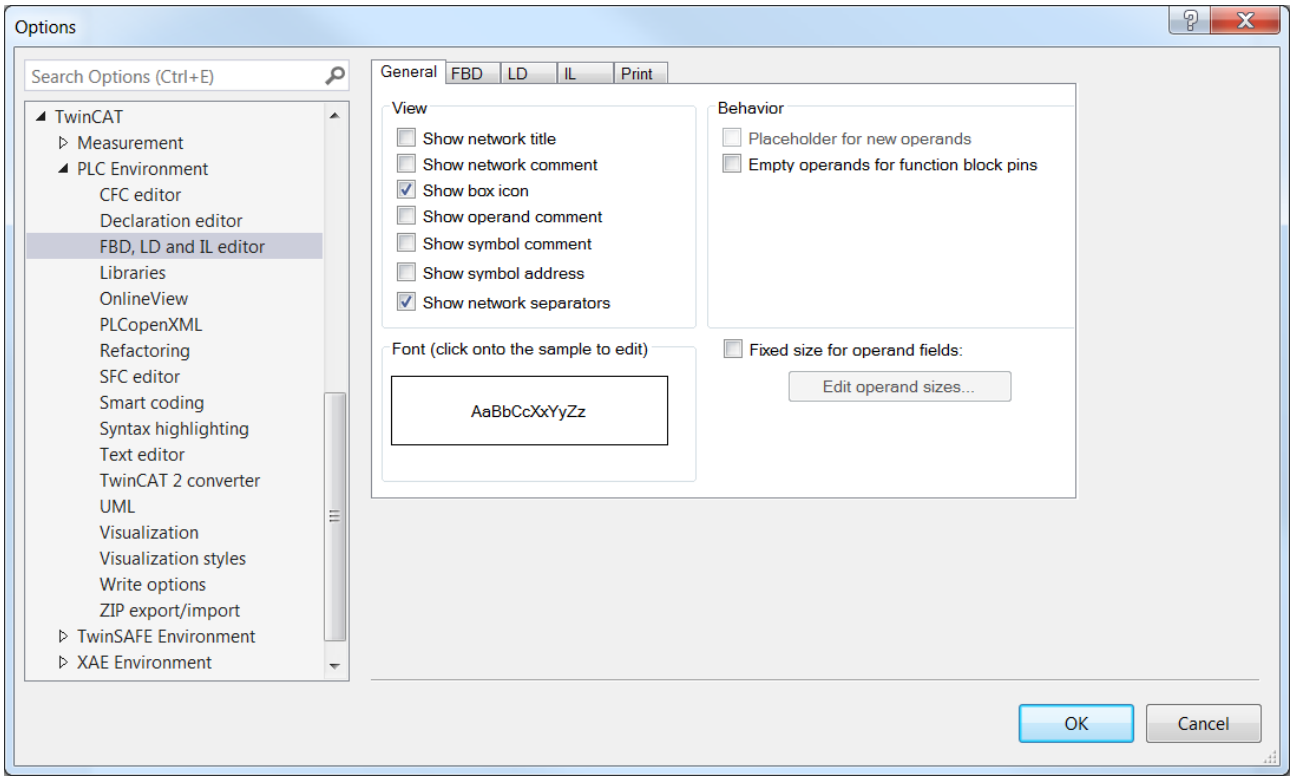
- PLC 文档：使用声明编辑器 [▶ 64]

17.9.1.3 对话框选项 - FBD、LD 和 IL

功能：此对话框用于配置 FBD/LD/IL 编辑器的演示选项。

调用：TwinCAT > PLC Environment (PLC 环境) > FBD, LD and IL (FBD、LD 和 IL)

常规选项卡



视图

显示网络标题	网络标题显示在网络左上角。
显示网络注释	网络注释显示在网络左上角。如果 TwinCAT 也显示网络标题，则注释将显示在下方的行中。
显示框图标	功能块符号显示在 FBD 和 LD 编辑器的块元素中。标准运算符也带有符号。
显示操作数注释	TwinCAT 显示已分配给实现部分中的变量的注释。相比“符号注释”，操作数注释仅表示变量的本地使用点。
显示符号注释	TwinCAT 显示在变量名上方声明中分配给变量或符号的注释。除符号注释之外或代替符号注释，还可以分配本地“操作数注释”。
显示符号地址	如果将地址分配给符号（变量），则该地址显示在变量名上方。
显示网络分隔符	在各个网络之间显示分界线。

Behavior（行为）

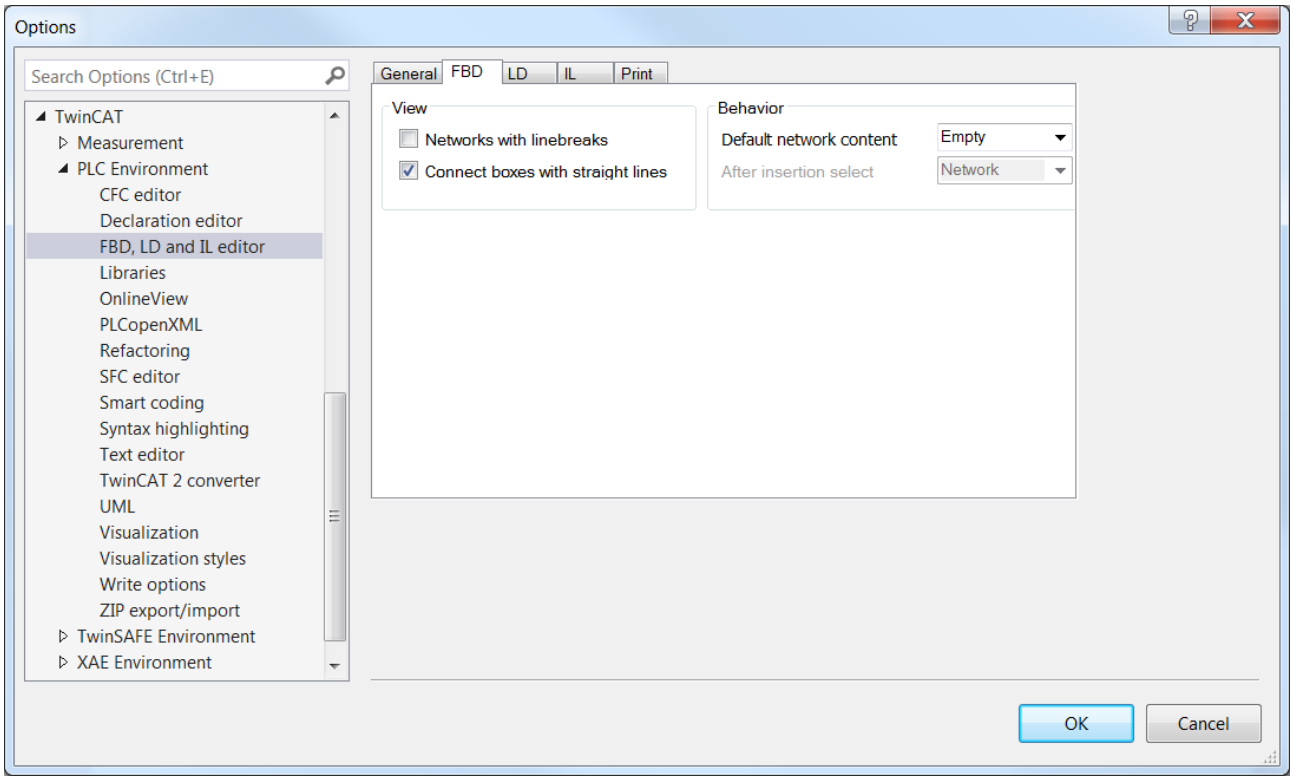
新操作数占位符	新功能块引脚的操作数字段留空（而不是“???”）。
清空功能块引脚的操作数	插入空操作数，而不是 ???。

字体

点击输入字段，打开 **Font（字体）** 对话框。

操作数字段的固定大小	☑：可以启用编辑操作数大小。
编辑操作数大小	显示 **Operand Size（操作数大小）** 对话框，以设置字符数和行数。

FBD 选项卡



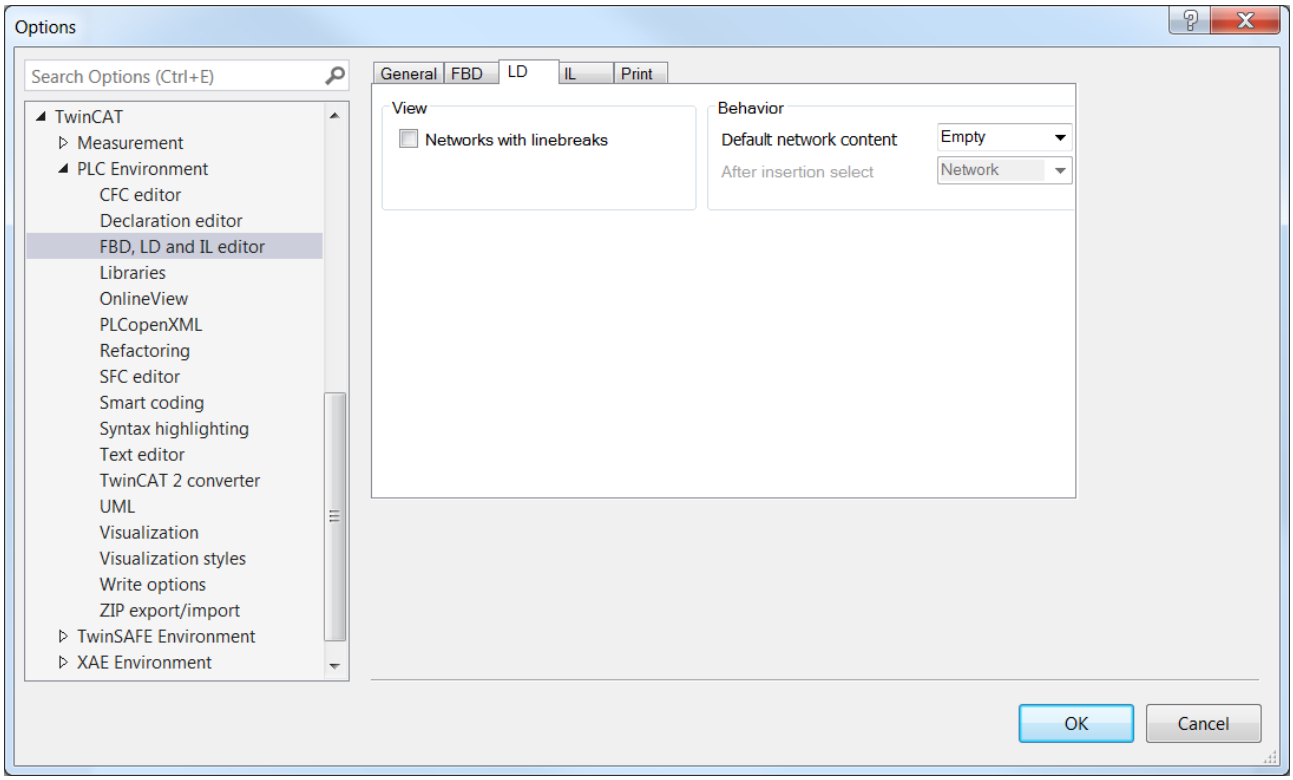
视图

带换行符的网络	☑：使用换行符表示网络，以便 TwinCAT 可以在当前窗口宽度内显示尽可能多的功能块。
使用直线连接框	☑：元素之间的行有固定的短长度。

Behavior（行为）

默认网络内容	选择列表：新网络的内容。
插入后选择	选择列表：插入新网络后 TwinCAT 选择的元素。

LD 选项卡



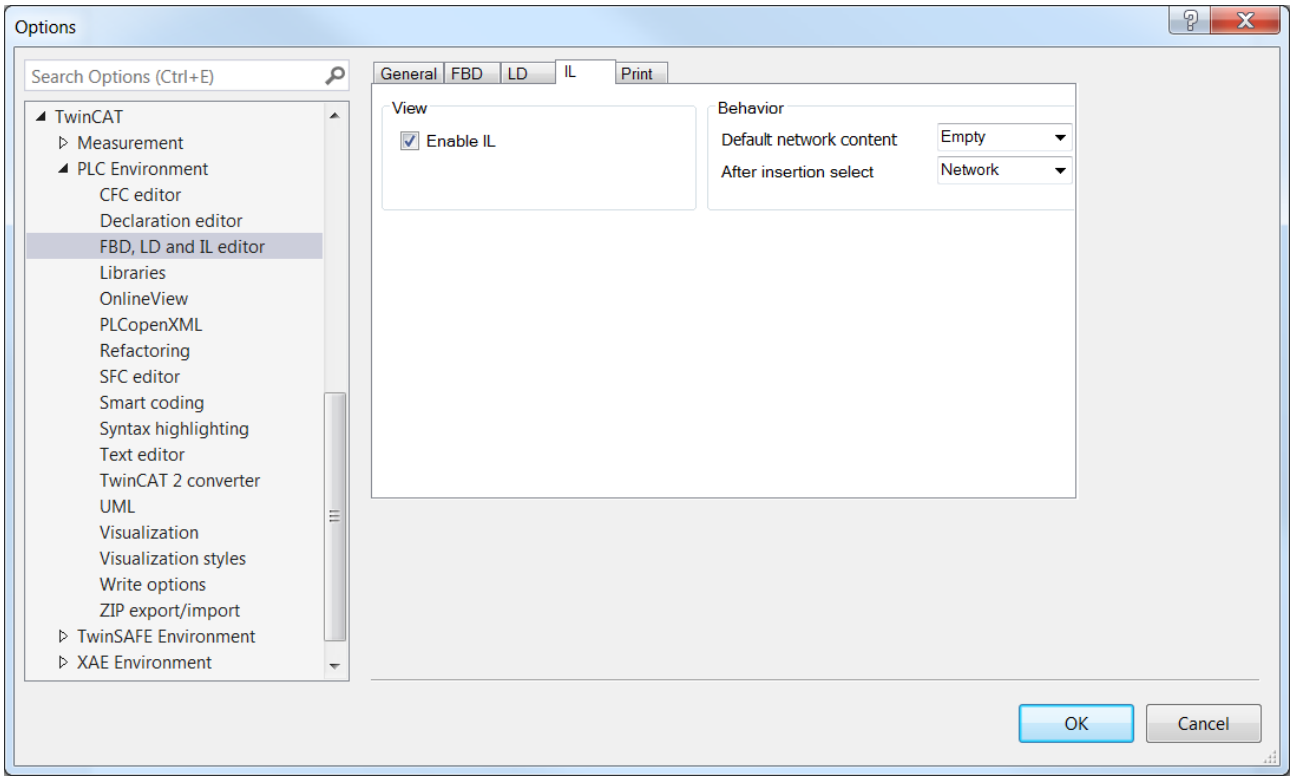
视图

带换行符的网络	☑：使用换行符表示网络，以便 TwinCAT 可以在当前窗口宽度内显示尽可能多的功能块。

Behavior（行为）

默认网络内容	选择列表：新网络的内容。
插入后选择	选择列表：插入新网络后 TwinCAT 选择的元素。

IL 选项卡



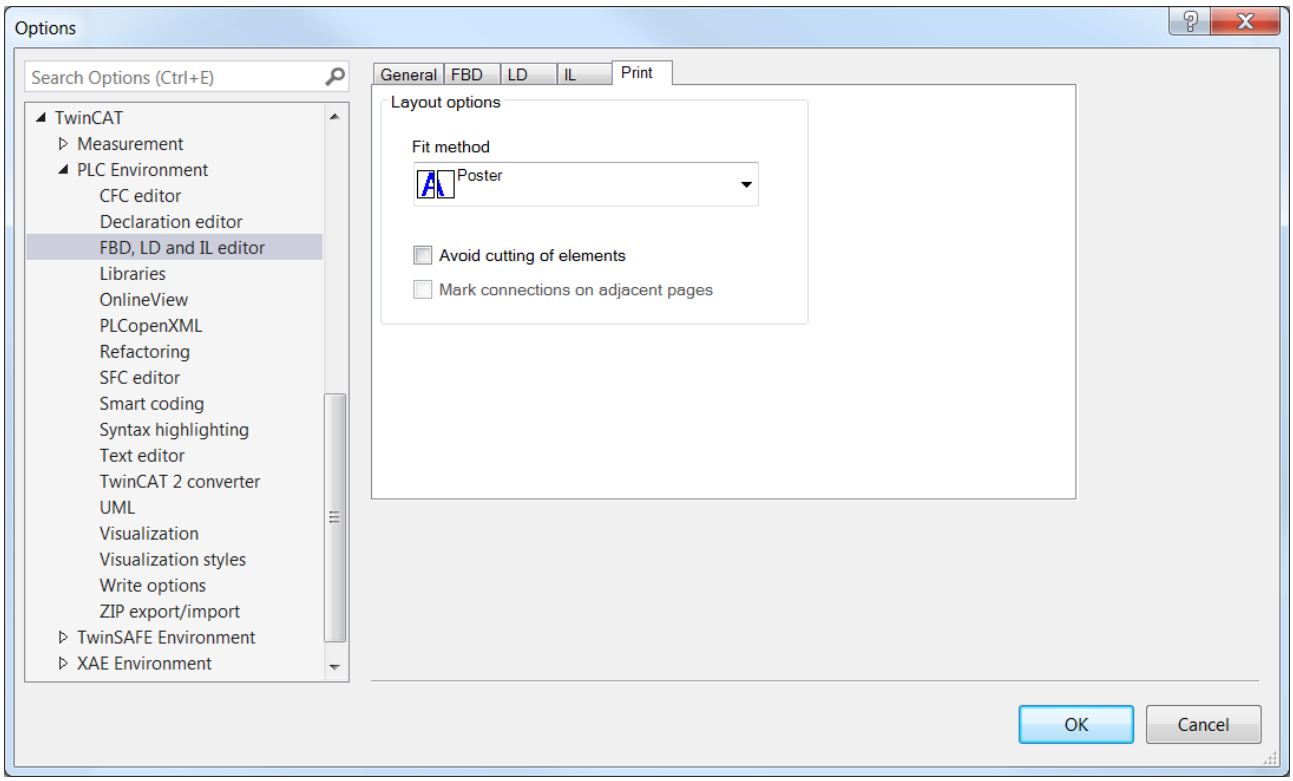
视图

启用 IL	实现语言 IL 在开发系统中可用。

Behavior（行为）

默认网络内容	选择列表：新网络的内容。
插入后选择	选择列表：插入新网络后 TwinCAT 选择的元素。

打印选项卡



布局选项

拟合方式	尺寸调整的选择列表。
避免切割元素	不适合该页面的项目将打印在下一页上。
标记相邻页面上的连接	如果启用了 **Avoid cutting of elements (避免切割元素)**，则可以启用。

另请参见：

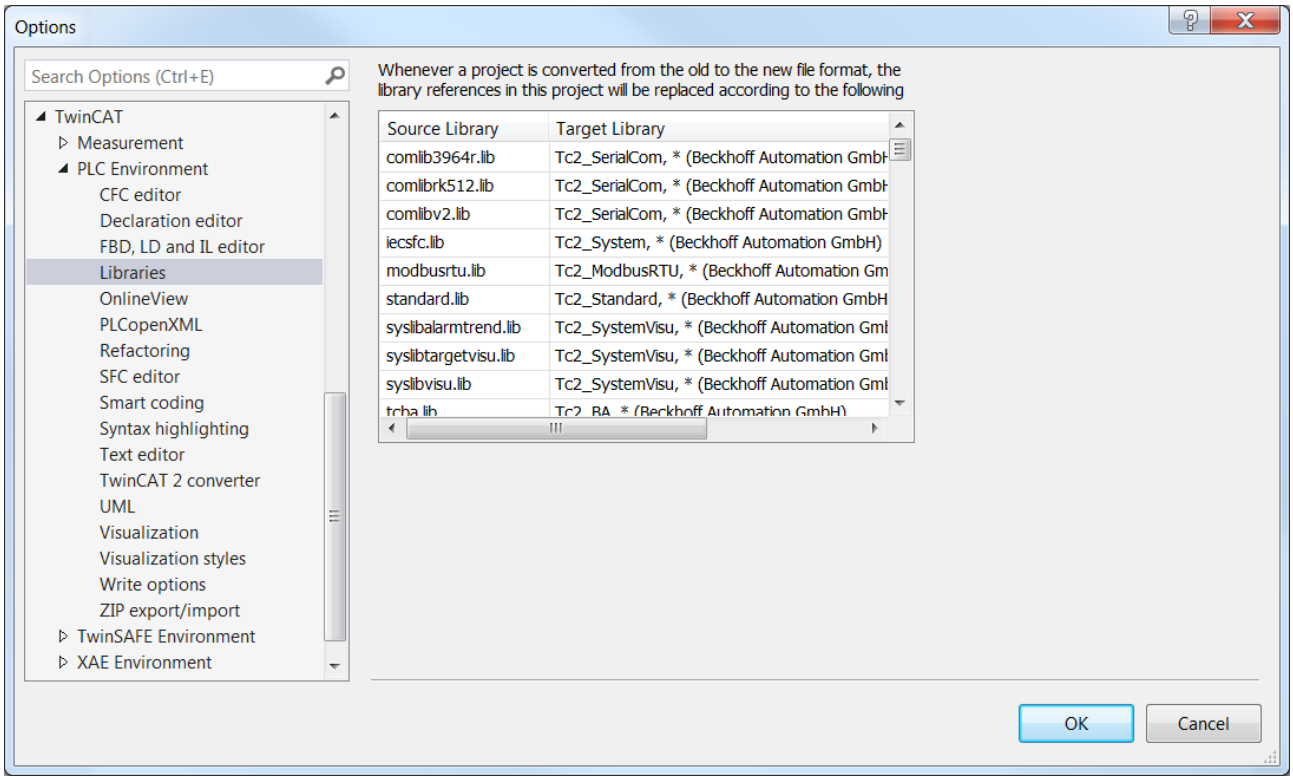
- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：编程语言及其编辑器 [▶ 569]

17.9.1.4 对话框选项 - 库

功能：该对话框包含库的设置。

调用：TwinCAT > PLC programming environment > Libraries（TwinCAT > PLC 编程环境 > 库）

转换选项卡



此选项卡用于管理 TwinCAT 在转换旧项目时所用库引用的映射。如果您尚未保存特定库的映射，则您在每次打开包含此库的旧项目时都必须重新定义映射。

映射定义了在将项目转换为当前格式后库引用的外观。提供 3 个选项：

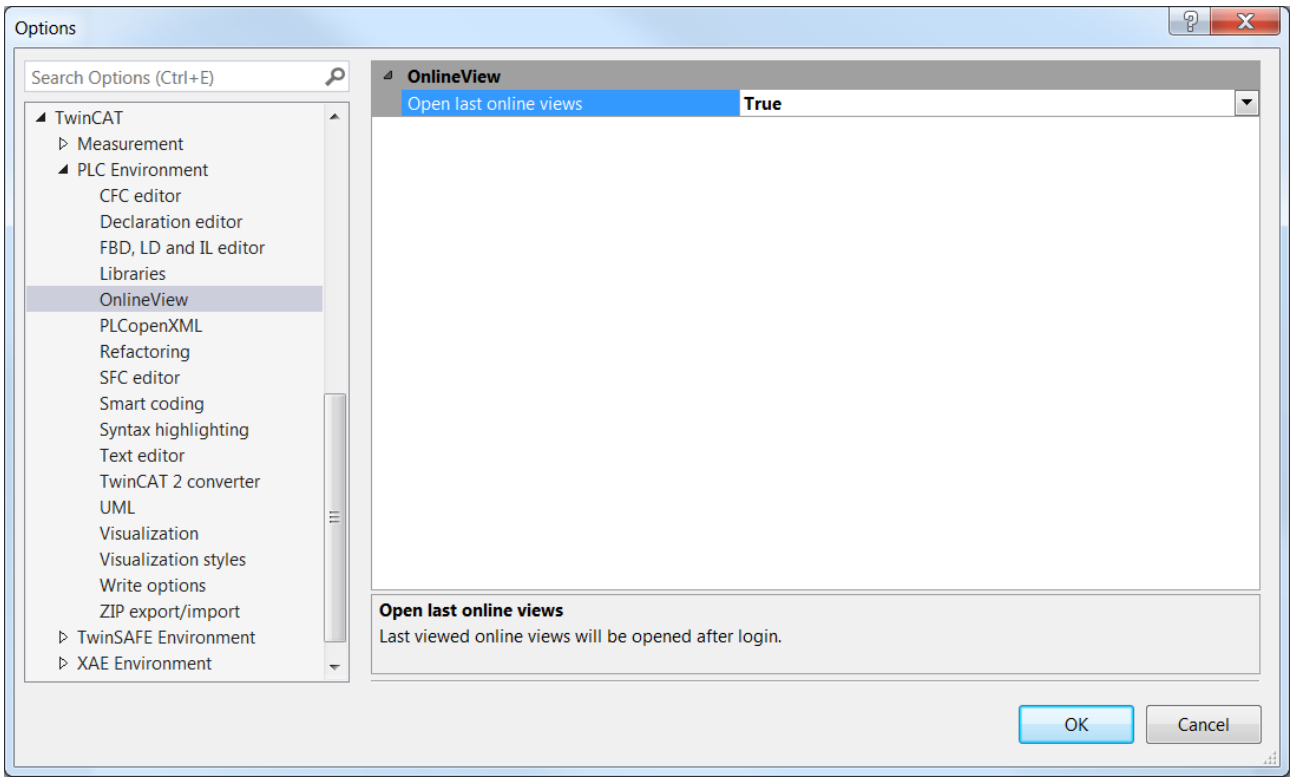
- 您保留引用。这意味着，TwinCAT 还会将库转换为当前格式（*.library）并将其安装在本地库存储库中。
- 您将引用替换为另一个引用。这意味着其中 1 个已安装的库将替换之前引用的库。
- 您删除引用。这意味着转换后的项目不再集成库。

在下一次旧项目转换期间，TwinCAT 将所有列出的映射应用于其库引用。这意味着如果待转换项目中再次包含相同的库，则您必须重复映射定义。您可以在最后一行输入新映射。	
源库	转换前项目中所包含库的路径。 双击条目，使字段可编辑，并显示 Input Assistant（输入助手）按钮。
目标库	转换后项目中待包含库的名称和位置。 双击条目将打开“Set target system library”（设置目标系统库）对话框。

另请参见：

- PLC 文档：使用库 [▶ 246]

17.9.1.5 对话框选项 - 在线视图



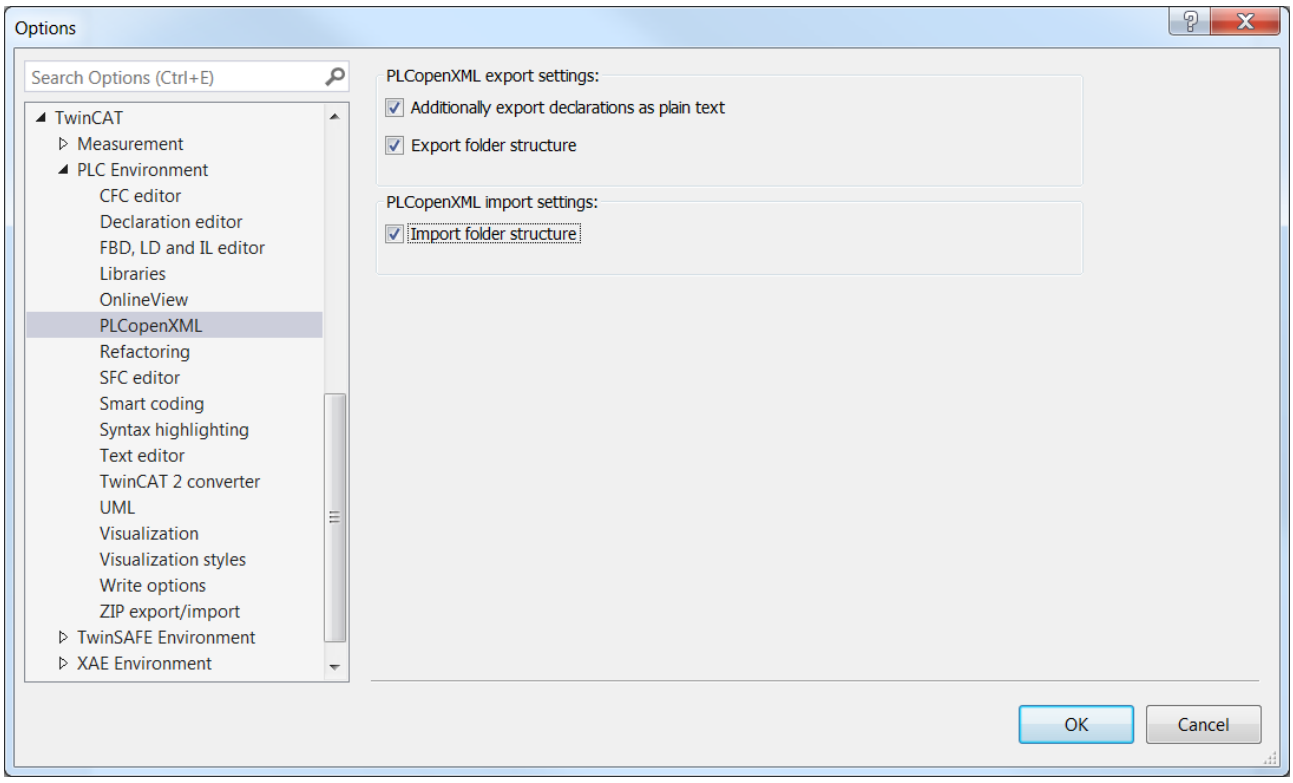
在线视图

打开上一次在线视图	• TRUE（真）（默认设置）：登录时，上一条在线会话的编辑器窗口打开。当前脱机视图保持打开状态。 • FALSE（假）：登录时，脱机视图保持打开状态。上一条在线会话的编辑器窗口将被丢弃，并且不会重新打开。

17.9.1.6 对话框选项 - PLCopenXML

功能：此对话框包含在 PLCopenXML 导出或导入时的 TwinCAT 行为设置。

调用：TwinCAT > PLC Environment (PLC 环境) > PLCopenXML



PLCopenXML 导出设置

另外，以纯文本形式导出声明	默认情况下，TwinCAT 会根据 PLCopenXML 模式将声明部分分割为单个变量，从而导致格式和一些注释信息丢失。 ☑：保留格式和注释。另外，TwinCAT 还将已导出声明部分的纯文本写入 PLCopenXML 文件，从而扩展 PLCopenXML 模式。
导出文件夹结构	☑：如果包含所选对象之一，TwinCAT 还会导出文件夹。这是 PLCopenXML 模式的 TwinCAT 特定扩展。

PLCopenXML 导入设置

导入文件夹结构	☑：如果导入文件包含关于对象文件夹结构的信息，TwinCAT 将导入此结构。 ☐：TwinCAT 导入不带结构的对象。

另请参见：

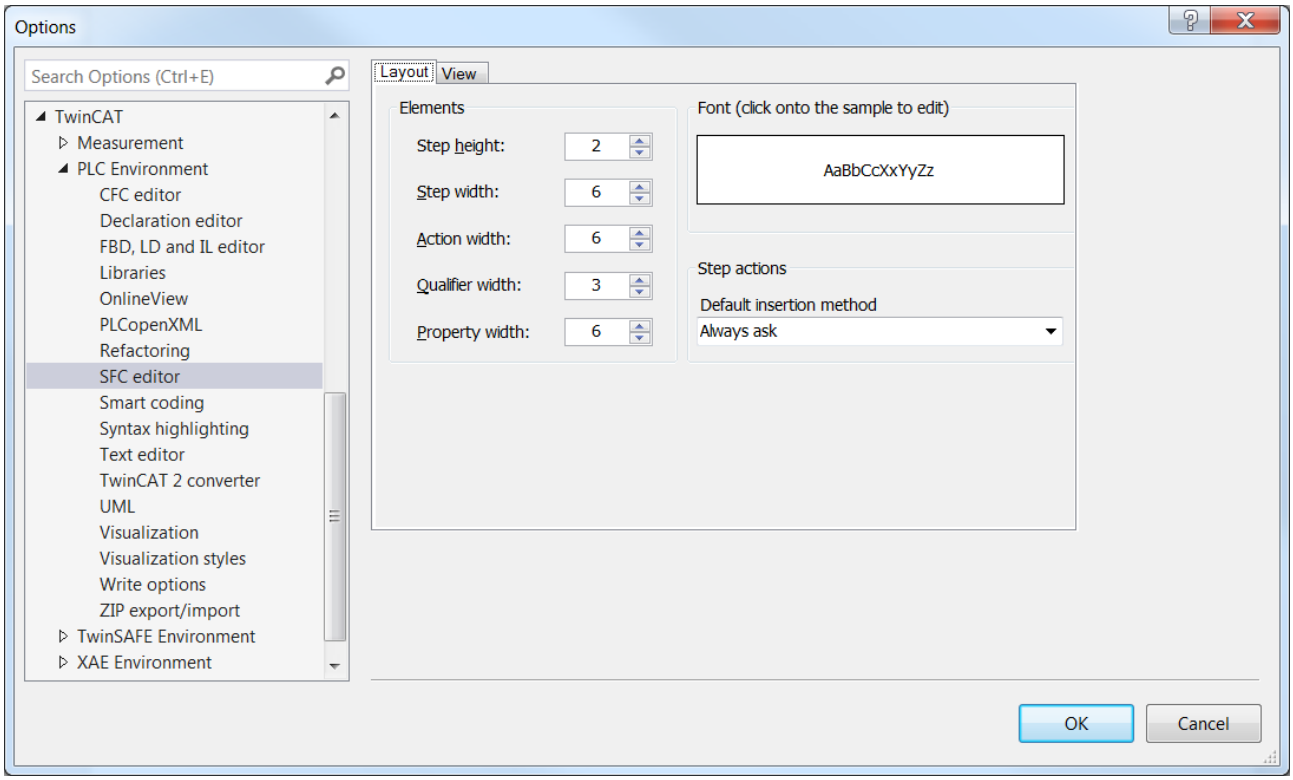
- 命令：导出 PLCopenXML
- 命令：导入 PLCopenXML
- PLC 文档：导出和导入 PLC 项目 [▶ 56]

17.9.1.7 对话框选项 - SFC 编辑器

功能：此对话框用于配置 SFC 编辑器的设置。

调用：TwinCAT > PLC Environment (PLC 环境) > SFC editor (SFC 编辑器)

布局选项卡



元素

定义 SFC 元素步骤、操作、限定符和属性的规格。以网格单位指定值。1 网格单位 = 当前在文本编辑器选项（文本区域/字体）中设置的字体大小。设置始终在所有当前打开的 SFC 编辑器窗口中立即生效。	
步高	可能值：1-100
步宽	可能值：2-100
操作宽度	可能值：2-100
限定符宽度	可能值：2-100
属性宽度	可能值：2-100

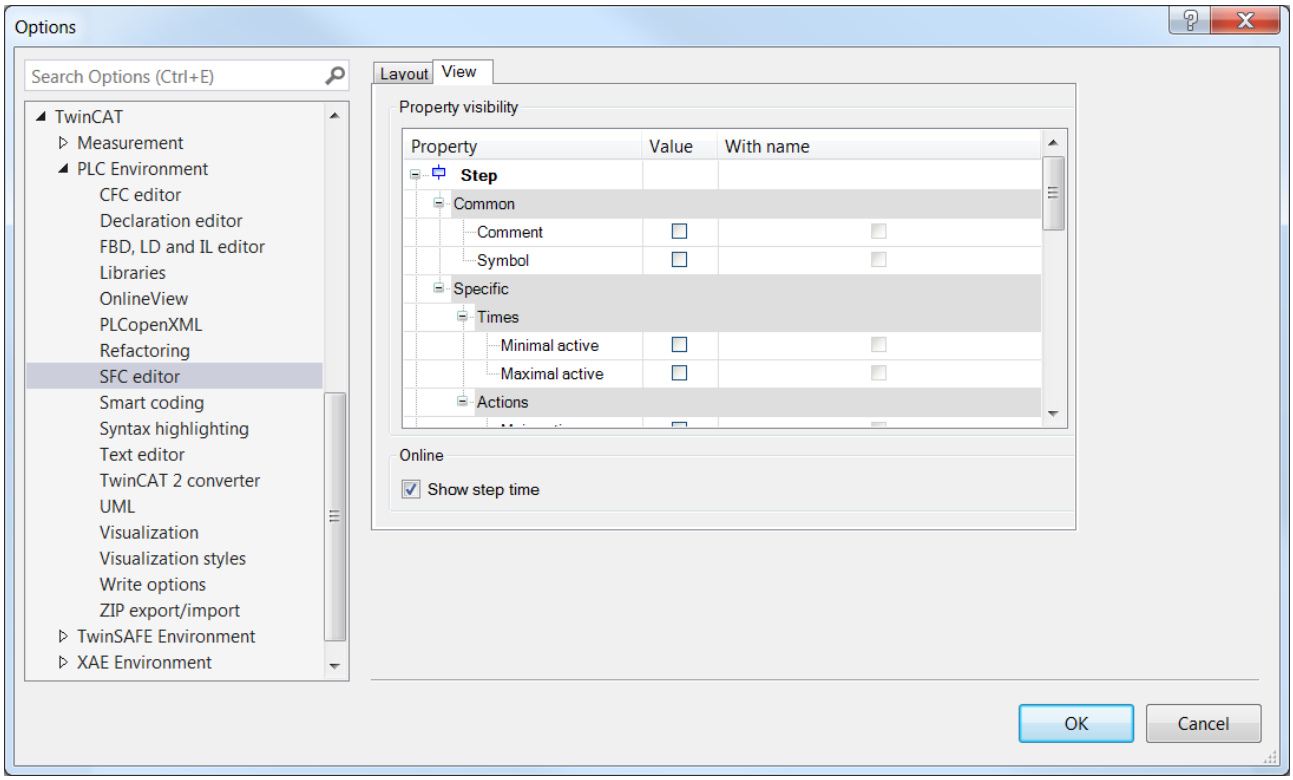
字体

示例文本显示当前设置的字体。点击可更改字体。

步骤动作

默认插入方法	• 始终询问 • 重复实现 • 复制引用

视图选项卡



属性可见性

列出普通和特殊类别的项目属性并定义显示选项。	
属性	定义 SFC 图表中项目旁边显示的项目属性。
Value	☑：显示属性值。
带名称	☑：显示带名称的属性值。

在线

显示步骤时间	☑：TwinCAT 在步骤右侧显示在线模式下的步骤时间。

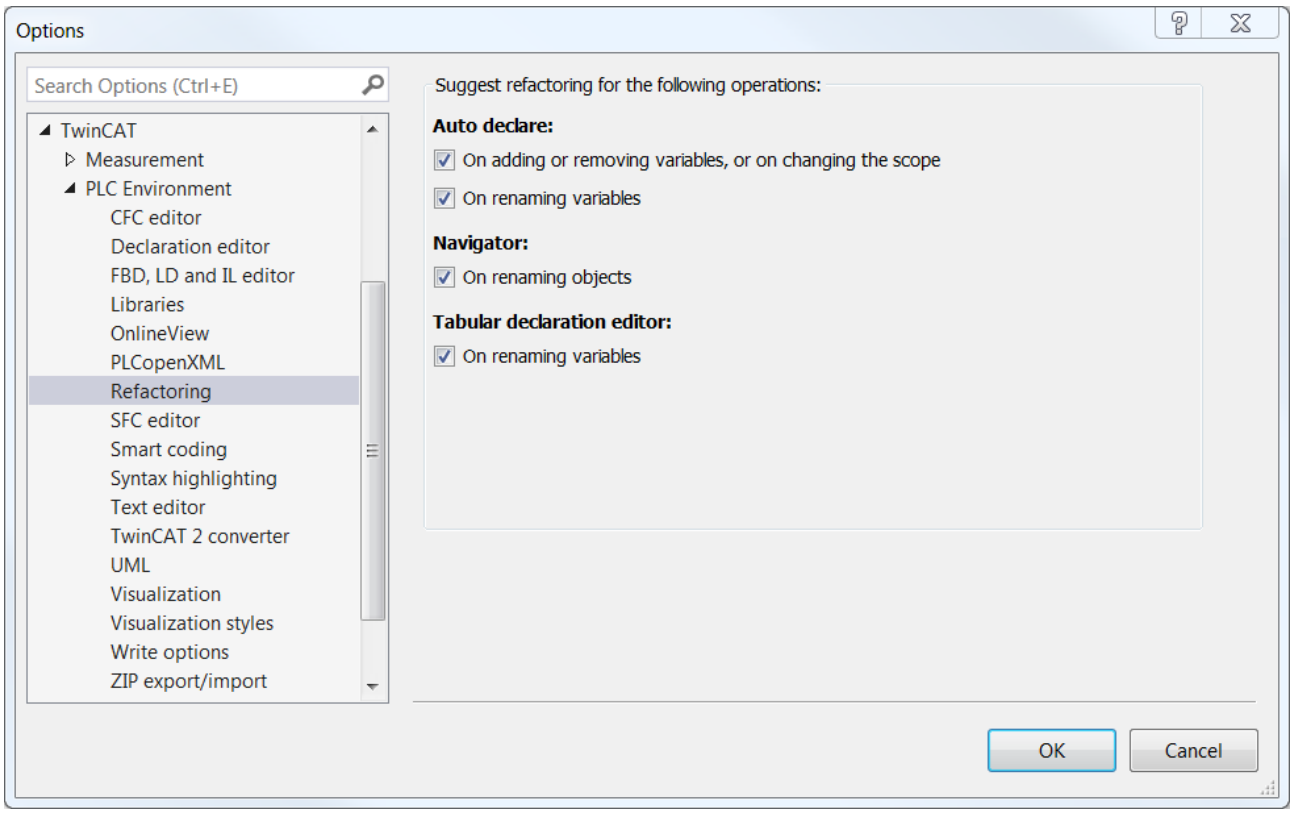
另请参见：

- PLC 文档：顺序功能图（SFC） [▶ 114]
- PLC 文档：编程语言及其编辑器 [▶ 569]

17.9.1.8 对话框选项 - 重构

功能：此对话框用于定义自动提出重构的项目中的操作。重构功能支持您的增强工作。

调用：TwinCAT > PLC programming environment > Refactoring（TwinCAT > PLC 编程环境 > 重构）



自动声明

如果您通过调用自动声明（**Auto Declare**（自动声明）对话框）更改变量名或者添加输入或输出变量，则会自动启用选项 **Apply changes using refactoring**（通过重构应用更改）。在确认对话框之后，**Refactoring**（重构）对话框将打开，并且您可以在项目范围内更改变量。

在添加或删除变量时，或者用于更改范围

☑：通过自动声明（**Auto Declare**（自动声明）对话框）添加新的输入或输出变量，或在自动声明中删除变量名，然后通过 **OK**（确定）关闭对话框。这将打开 **Refactoring**（重构）对话框，以在项目范围内添加或删除变量

在重命名变量时

☑：在自动声明（**Auto Declare**（自动声明）对话框）中重命名该名称，然后通过 **OK**（确定）关闭对话框。这将打开 **Refactoring**（重构）对话框，以在项目范围内重命名变量。

导航器

在重命名对象时

☑：如果您更改 PLC 项目树中对象的名称，则会出现提示，询问 TwinCAT 是否应该执行“Automatic refactoring”（自动重构）。

表格声明编辑器

在重命名变量时

☑：如果您在表格声明编辑器中更改变量名，则会出现提示，询问 TwinCAT 是否应该执行“Automatic refactoring”（自动重构），以进行重命名。

UML 类图	为类图编辑器中的更改提供重构支持的选项。 **在添加或删除变量时** ☑：当您在类图中的 VAR_INPUT、VAR_OUTPUT 和 VAR_IN_OUT 部分添加或删除变量时，支持重构。 **在重命名功能块时** ☑：如果您更改类图中的功能块名称，则支持重构。 **在重命名变量或属性时** ☑：如果您重命名类图中的变量或属性，则支持重构。 如果在 UML 选项中启用 **Skip Refactoring Preview**（跳过重构预览）选项，则可能会在项目中所有受影响的位置上执行重构，而不会首先显示在 **Refactoring**（重构）对话框中，具体视情况而定。（请参见 对话框选项 - UML [▶ 918]）

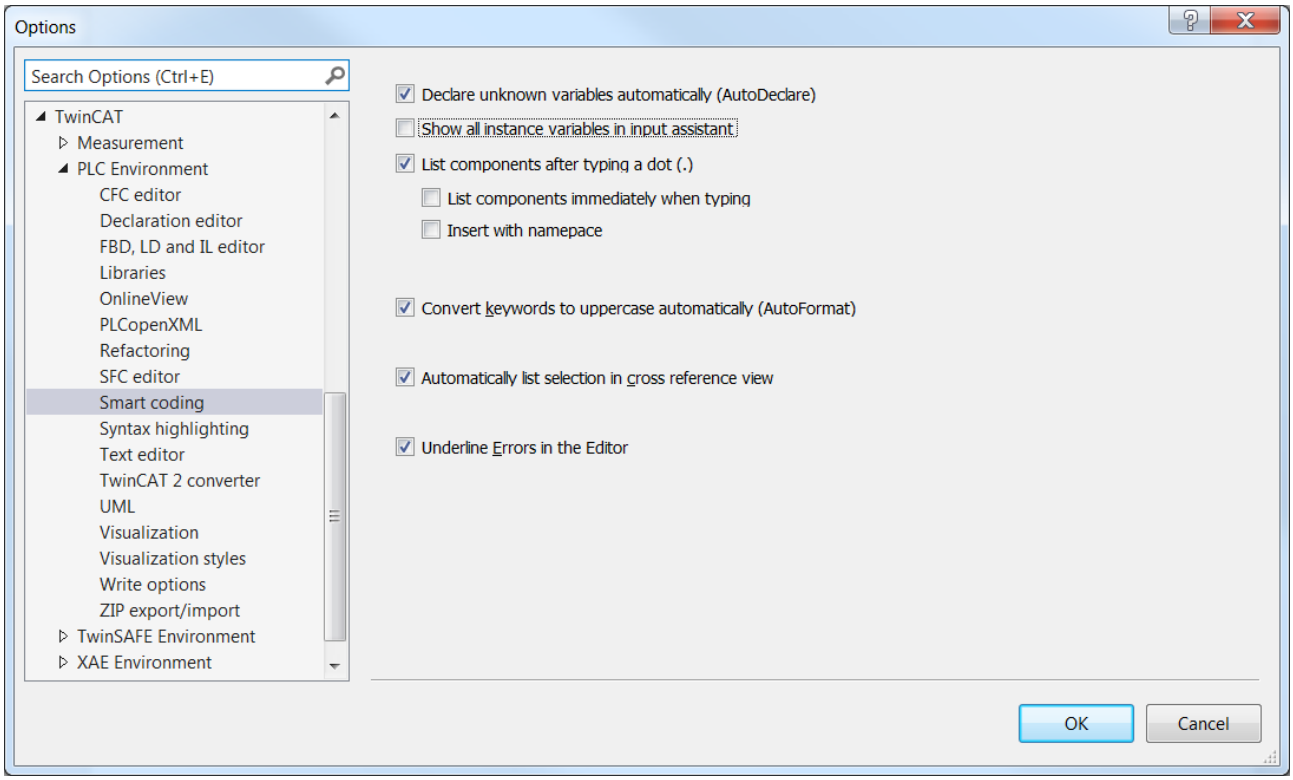
另请参见：

- 命令：自动声明 [▶ 811]
- 命令：重命名 '<variable>' [▶ 822]
- 命令：添加 '<variable>' [▶ 823]
- 命令：删除 '<variable>' [▶ 825]
- PLC 文档：重构 [▶ 148]

17.9.1.9 对话框选项 - 智能编码

功能：此对话框用于配置更易于输入代码的设置。

调用：TwinCAT > PLC programming environment > Smart Coding（TwinCAT > PLC 编程环境 > 智能编码）



自动声明未知变量（AutoDeclare）	☑：当您在编程语言编辑器中输入未声明的标识符并退出输入行后，**Auto Declare**（自动声明）对话框将打开。 对于 Build 4026 及以上版本，为了使 AutoDeclare 功能在 ST 编辑器中也可用，还必须启用选项 **Enable for the ST editor**（为 ST 编辑器启用）。
为 ST 编辑器启用	前提条件：选项 **Declare unknown variables automatically (AutoDeclare)**（自动声明未知变量（AutoDeclare））已启用。 ☑：AutoDeclare 功能在 ST 编辑器中也可用。 ☐：AutoDeclare 功能在 ST 编辑器中不可用。 （Build 4026 及以上可用）
在输入助手中显示所有实例变量	☑：**List components**（列出组件）功能还提供功能块实例的局部变量以供选择。 ☐：**List components**（列出组件）功能仅提供 FB 实例的输入和输出变量以供选择。
键入点（.）后列出组件	☑：启用 **List components**（列出组件）功能。 这意味着：如果您在 TwinCAT 需要标识符的位置输入 1 个点，则会出现 1 个带输入选项的选择列表。
键入时立即列出组件	要求：选项 **List components after typing a dot (.)**（键入点（.）后列出组件）已启用。 ☑：在输入任何字符串后，将显示可用标识符和运算符的选择列表
插入命名空间	☑：TwinCAT 自动在标识符前插入命名空间。
自动将关键字转换为大写（AutoFormat）	☑：TwinCAT 自动以大写字母写入所有关键字。
在交叉引用视图中自动列出选择	☑：交叉引用列表自动显示当前选定或光标所在位置的变量/POU/DUT 的引用。
编辑器中的下划线错误	☑：错误或未知的程序代码加下划线。 （Build 4026 及以上可用）
突出显示符号	☑：在编辑器中，光标所在的符号的所有使用位置都会以彩色突出显示。这样，您可以在编辑器中快速识别交叉引用。 （Build 4026 及以上可用）

另请参见：

- PLC 文档：编程语言及其编辑器 [▶ 569]
- PLC 文档：使用输入助手 [▶ 123]
- PLC 文档：查找使用交叉引用列表的位置 [▶ 145]

17.9.1.10 对话框：选项 - 梯形图编辑器

符号：

功能：此对话框用于配置梯形图编辑器的显示选项。

调用：TwinCAT > PLC Environment > Ladder editor（TwinCAT > PLC 环境 > 梯形图编辑器）

常规选项卡

视图

显示网络标题	网络标题显示在网络的左上角。
显示网络注释	网络注释显示在网络的左上角。如果 TwinCAT 也显示网络标题，则注释将显示在下方的行中。
显示方框图标	方框图标显示在梯形图编辑器的方框元素中。标准运算符也带有符号。
显示操作数注释	TwinCAT 显示已分配给实现部分中的变量的注释。与“symbol comment”（符号注释）相比，操作数注释仅表示变量的本地使用点。 注释会根据可用空间自动截断。
显示符号注释	TwinCAT 显示在变量名上方声明中分配给变量或符号的注释。除符号注释之外或代替符号注释，您还可以分配本地“操作数注释”。
显示符号地址	如果将地址分配给符号（变量），则该地址显示在变量名上方。

操作数大小

最大显示行数	显示操作数名称的最大行数。
每行最大平均字符数	每行显示操作数名称的最大字符数

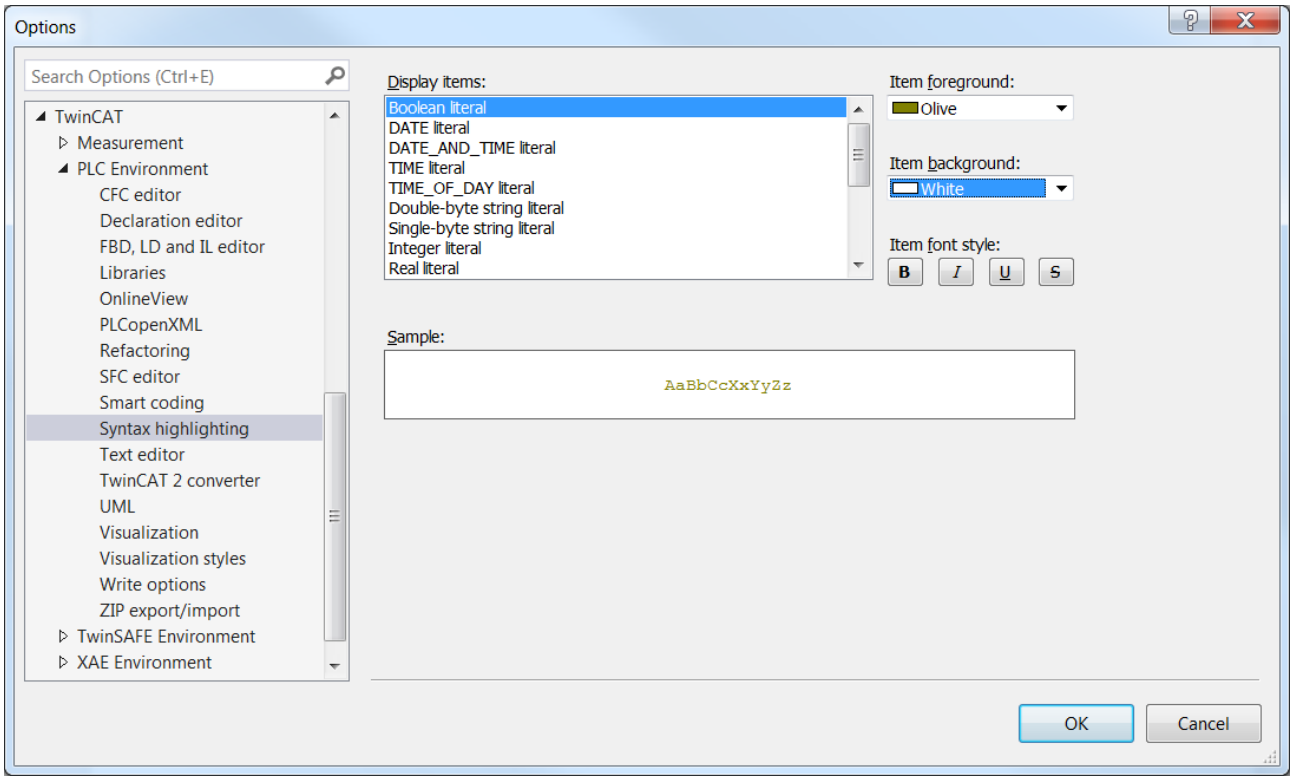
监控选项卡

浮点数的显示位数	位数的选择列表 示例：设置的 *5* 位数的值 *750.15* 会变为设置的 *2* 位数的值 *7.5e+02*。
显示字符串长度	字符数的选择列表

17.9.1.11 对话框选项 - 语法突出显示

功能：此对话框用于配置编辑器文本元素的颜色和字体设置（例如操作数、编译指示）。

调用：TwinCAT > PLC Environment （PLC 环境）> Syntax highlighting（语法突出显示）



显示项	文本项的选择列表
项目前景	文本项的前景颜色
项目背景	文本项的背景颜色
项字体样式	文本项字体样式（粗体、斜体、下划线、删除线）
示例	示例文本在预览中显示当前设置

17.9.1.12 对话框选项 - 文本编辑器

功能：此对话框包含文本编辑器中的显示和工作设置。

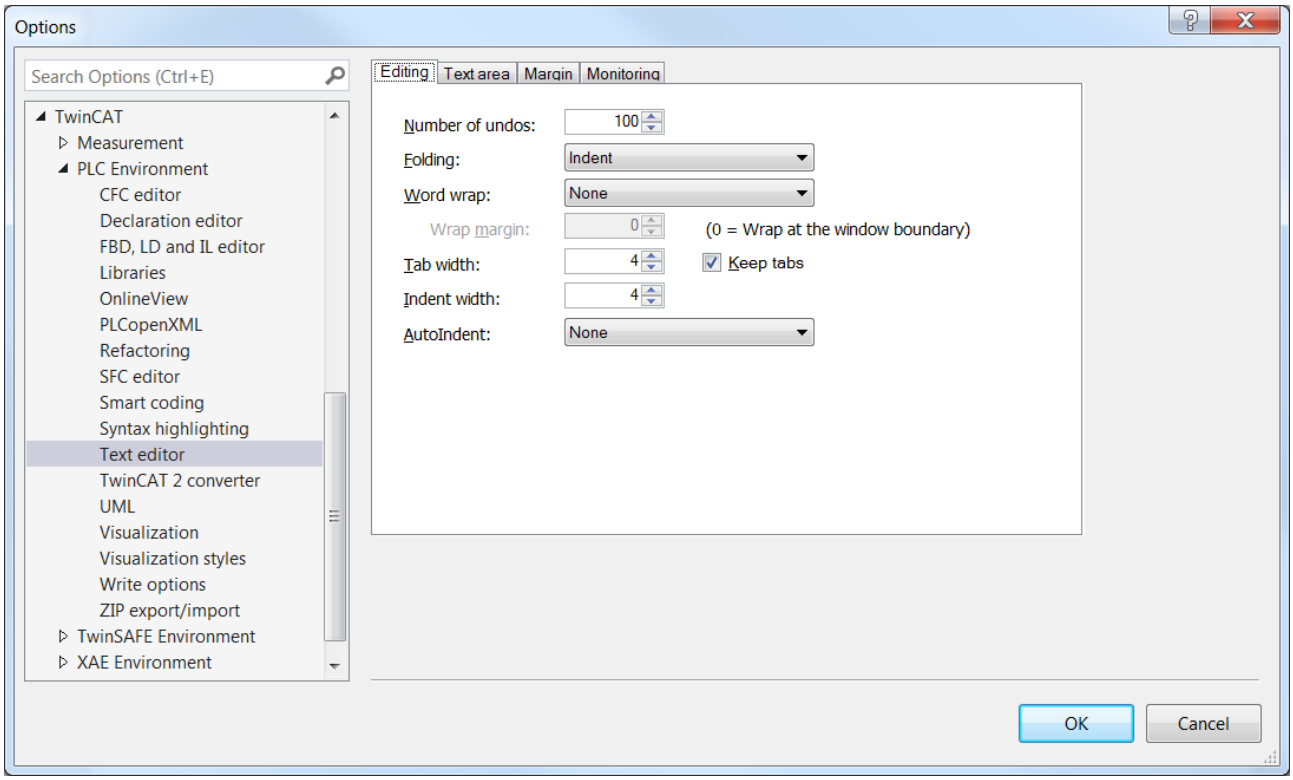
调用：TwinCAT > PLC programming environment > Text editor（TwinCAT > PLC 编程环境 > 文本编辑器）

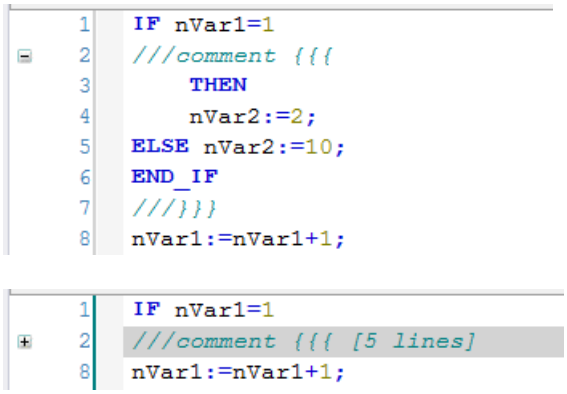
主题选项卡

在此选项卡中，您可以针对 ST 编辑器界面设计设置所需的主题。（Build 4026 及以上可用）

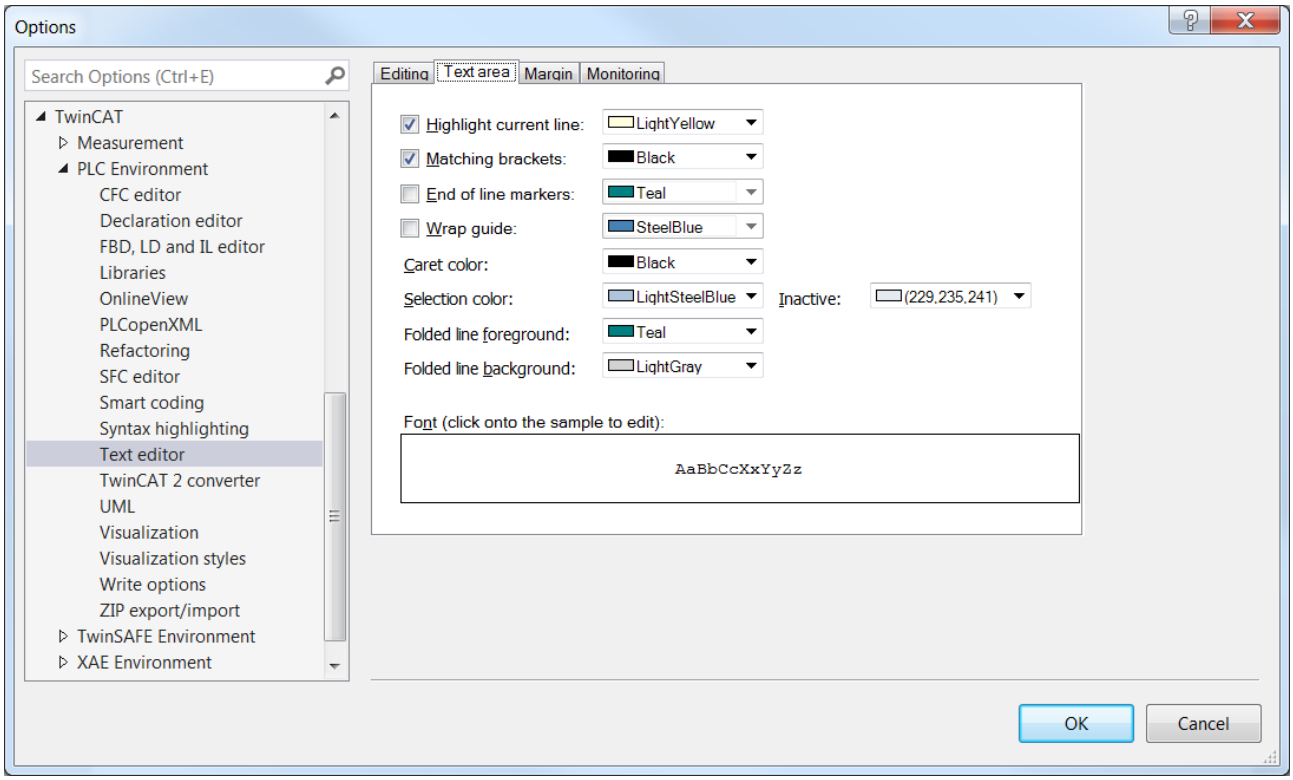
主题	文本编辑器的配色方案。所选主题将显示在预览窗口中。

编辑选项卡



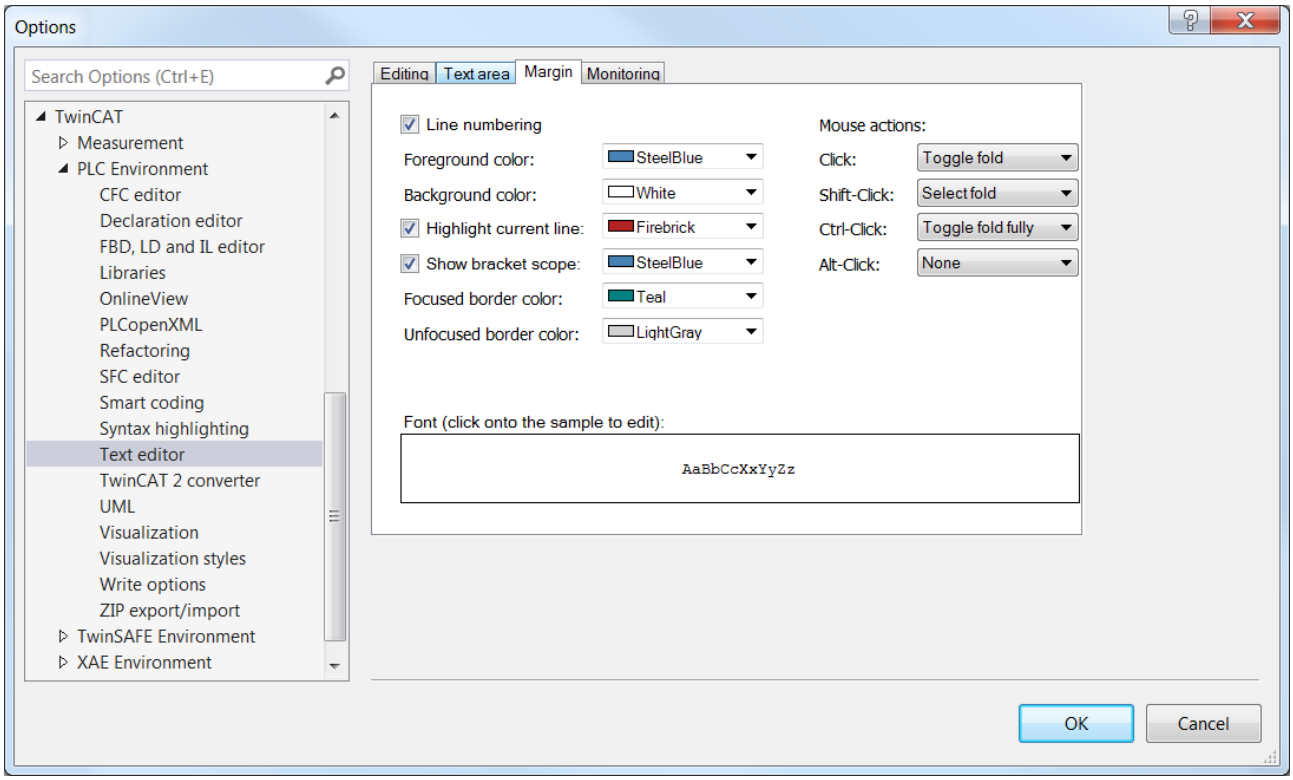
撤消数	命令 **Edit** > **Undo**（编辑 > 撤销）可用的最大步骤数。
折叠	通过缩进定义代码结构。 在选择缩进时，您可以使用各部分第 1 行左侧的加号或减号展开或折叠缩进部分。 • **缩进**：TwinCAT 将所有相对于前一行缩进的行组合在 1 个缩进单元中。 • **显式**：您显式使用注释来标识要在缩进单元中分组的代码部分：该部分必须以包含 3 个前花括号“{{{”的注释开头，并且以包含 3 个后花括号“}}}”的注释结束。注释可以包含附加文本。示例： 
自动换行	• **软**：如果为**换行边距**输入 0，则换行发生在编辑器窗口边缘。 • **硬**：换行发生在换行边距指定的字符数之后。
制表符宽度	字符数
保留制表符	☑：您使用 [Tab] 键插入的空格之后不会被 TwinCAT 转换为单独的空格。
缩进宽度	如果您已为 **Auto Indent**（自动缩进）选项激活智能或智能代码补全，则 TwinCAT 将在行起始处插入一定数量的空格。
自动缩进	• **不要自动缩进** • **块**：新行自动采用前一行的缩进。 • **智能**：包含关键字（例如，VAR）的行后面的行会自动缩进指定的缩进宽度。 • **智能代码补全**：缩进与 **Smart**（智能）选项相同，此外 TwinCAT 插入最终关键字（例如，END_VAR）。

文本区域选项卡



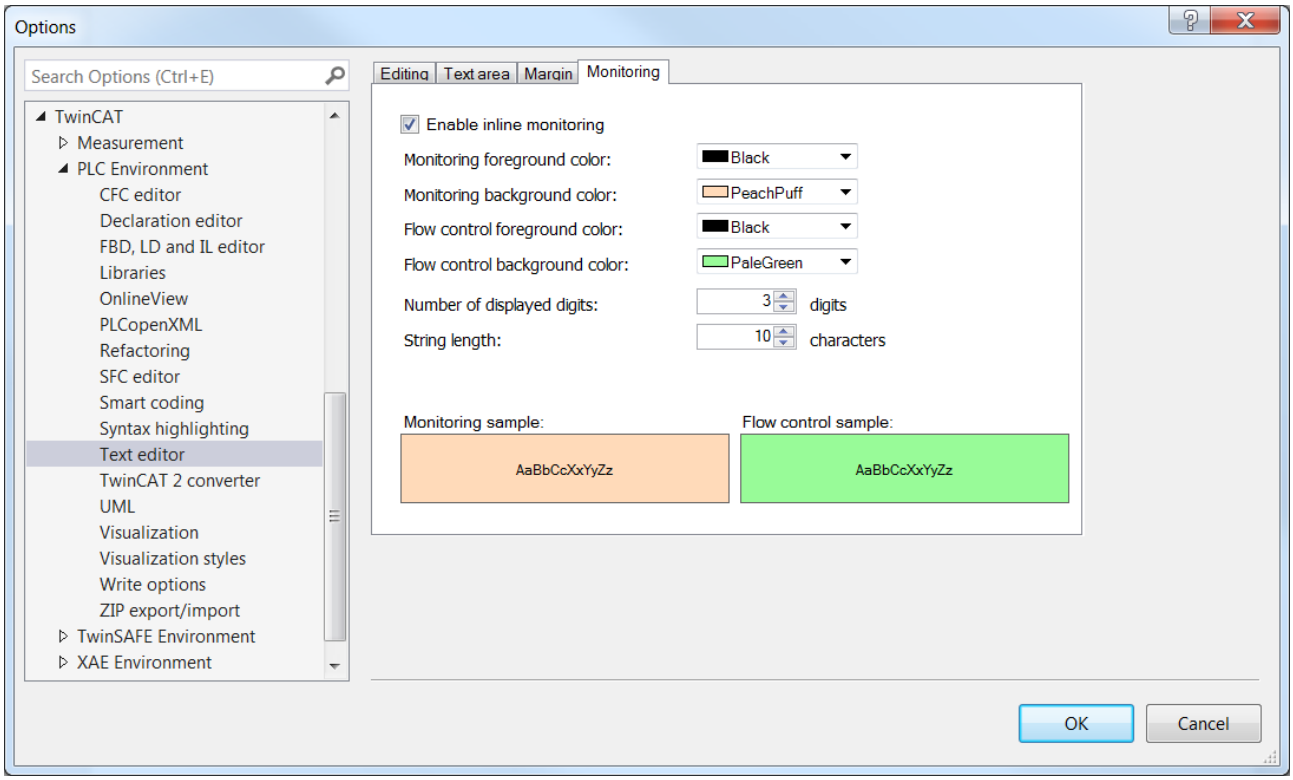
突出显示当前行	☑：突出显示光标所在的行。
匹配括号	☑：如果光标位于代码行中括号之前或之后，则 TwinCAT 会使用边框标记相应的右括号或左括号。
行标记	☑：在行最后一个字符（包括空格）之后，TwinCAT 使用小水平线标记各编辑器行的末端。
换行指南：	☑：如果启用软换行符或硬换行符，则定义的换行指南将通过垂直线显示。
插入符颜色	光标字符的颜色
选择颜色	所选文本区域的颜色
未激活	在相应窗口未激活情况下的选择颜色（焦点在另一窗口上）。
折叠行前景	代码中闭合缩进部分的标题行颜色
折叠行背景	代码中闭合缩进部分的标题行以彩色突出显示。
字体	点击字段将打开用于配置字体的标准对话框。

页边选项卡



文本编辑器窗口左侧页边的设置（由输入区域的垂直线分隔）：	
行号	☑：在声明和实现部分中显示行号，每个行号以 1 开头
前景颜色	行号颜色
背景颜色	页边颜色
突出显示当前行	☑：突出显示光标所在行的行号。
显示括号范围	☑：括号包括打开和关闭结构的关键字之间的行，例如 IF 和 END_IF。 如果该选项已启用并且光标位于结构的某个关键字之前、之后或之中，则括号区域通过页边处的方括号指示。
聚焦的边框颜色	页边和输入区域之间分界线的颜色
未聚焦的边框颜色	窗口当前未激活部分的页边和输入区域之间分界线的颜色
鼠标操作	可以为每个规定的鼠标动作或鼠标按钮快捷方式分配以下操作之一。在括号区域标题行前面的加号或减号处执行鼠标动作时，TwinCAT 执行以下动作： • **无**：鼠标动作不会触发任何动作。 • **选择折叠**：TwinCAT 选择括号区域的所有行。 • **切换折叠**：TwinCAT 打开或关闭括号区域，或者在嵌套括号的情况下打开或关闭括号区域的第 1 级。 • **完全切换折叠**：TwinCAT 打开或关闭嵌套括号区域的所有级。

监控选项卡



用于显示监控字段的设置	
启用内联监控	☑：在在线模式下，监控字段在变量后显示
监控前景颜色	展示监控字段中的值
监控背景颜色	展示监控字段中的背景
流程控制前景颜色	展示流程控制项处监控字段中的值
流程控制背景颜色	展示流程控制位置处监控字段的背景
显示位数	监控字段中的小数位数
字符串长度	监控字段中字符串变量值的最大长度

另请参见：

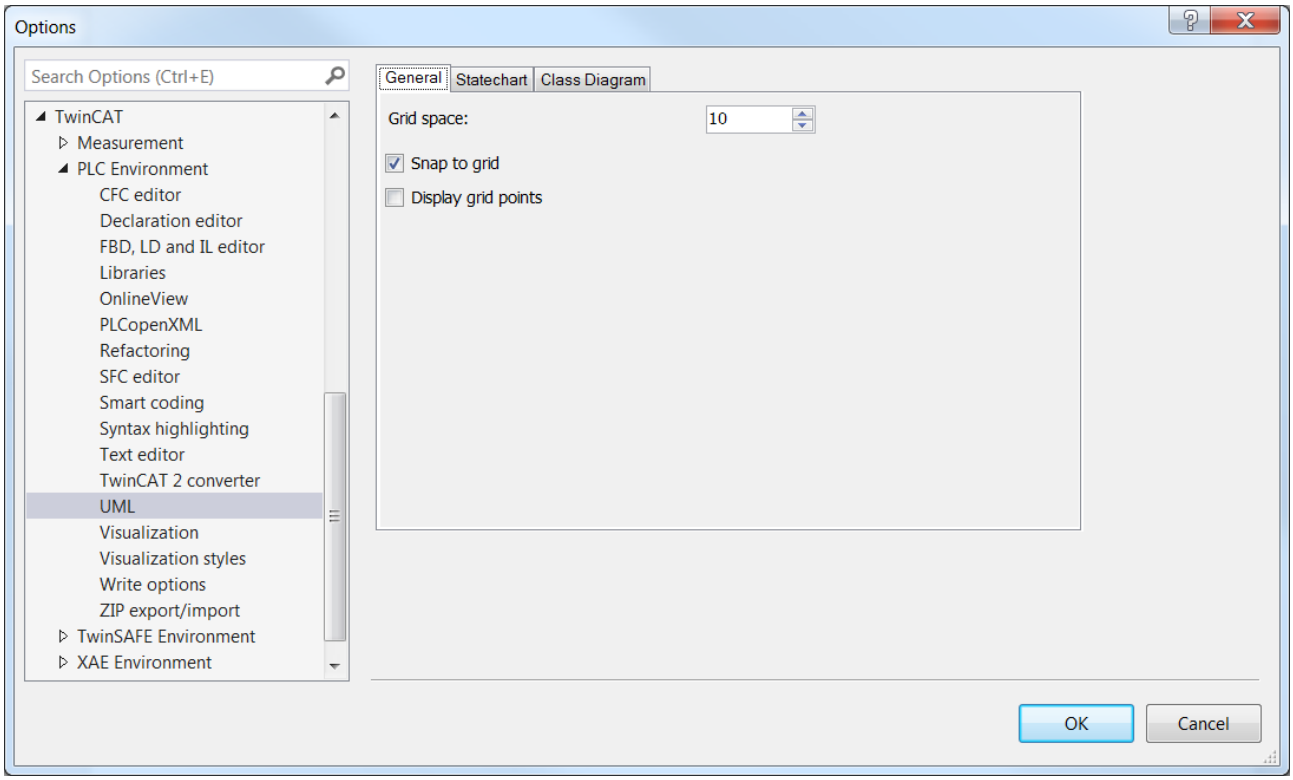
- PLC 文档：编程语言及其编辑器 [▶ 569]

17.9.1.13 对话框选项 - UML

功能：此对话框用于配置 UML 编辑器。

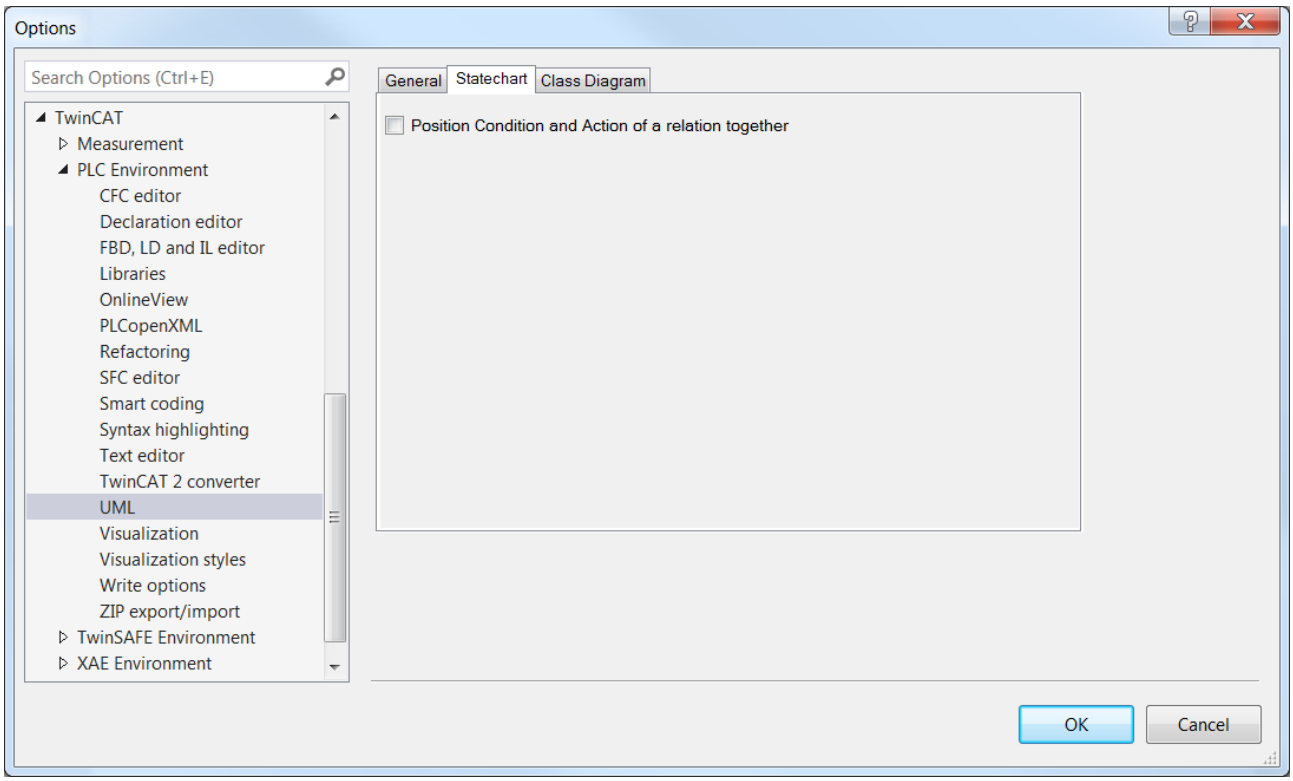
调用：TwinCAT > PLC programming environment > UML（TwinCAT > PLC 编程环境 > UML）

常规选项卡



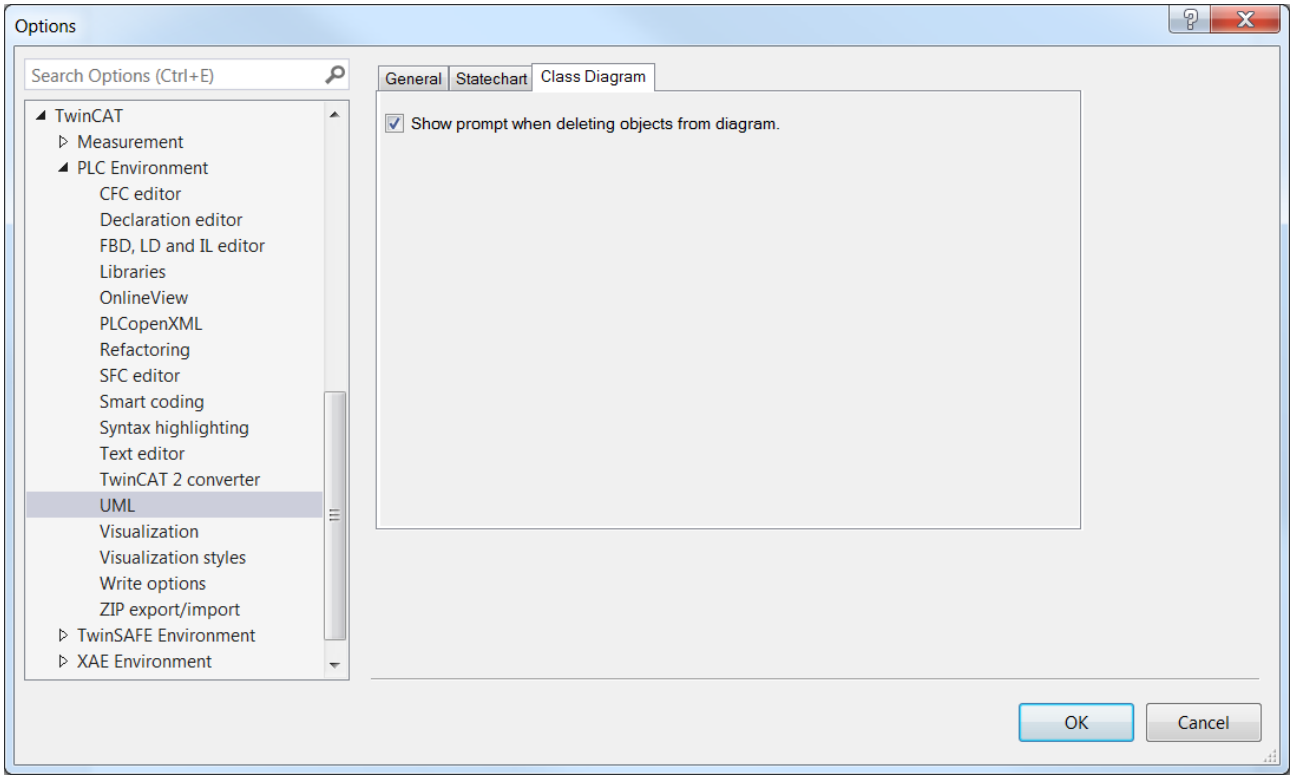
触点间距	网格线间距（单位：像素）。 默认值：10
连接网格	☑：UML 编辑器中的所有元素均与网格对齐。
显示网格点	☑：网格点在 UML 编辑器中显示。

状态图选项卡



关系的位置条件和动作	☑：在状态图中，同步移动属于同一转换的保护条件和动作。

类图选项卡



从图中删除对象时显示提示	对象仅可从图表或图表和项目中删除。 ☐：默认情况下，仅从图中删除对象。 ☑：删除对象时，将出现选择窗口，以配置仅从图中还是从项目中删除对象。
跳过重构预览	☑：如果在图中启动重构，则会执行项目范围内的更改，而无需首先打开带有所有更改点的预览的 **Refactoring**（重构）对话框。 另请参见： • PLC 文档：重构 [▶ 148]

另请参见：

- UML 文档：

17.9.1.14 对话框选项 - 可视化

功能：此对话框用于配置可视化编辑器。

调用：TwinCAT > PLC Environment (PLC 环境) > Visualization (可视化)

常规选项卡

这些设置仅用于集成可视化，即不用于 TwinCAT PLC HMI (TargetVisu) 和 TwinCAT PLC HMI Web 显示变体。

显示选项

固定	可视化保持其原始大小。
各向同性	可视化保持其比例。
各向异性	可视化根据开发系统中的窗口大小进行调整
抗锯齿图	可视化的特点是抗锯齿方法，在编辑时以及在运行时集成可视化时均不例外。 提示：如果在特定的可视化平台上画了一条焦点不准的水平线或垂直线，则可以通过在线厚度方向上偏移 0.5 px 进行校正；参见项属性 **Absolute motion (绝对运动)**，选项 **Use REAL values (使用 REAL 值)**。要求：平台支持使用 REAL 坐标

编辑选项

链接到移位/关键变量	可视化元素属性中的 < 占位符 <Shift/Key variable> 启用。 如果将属性颜色变量、颜色更改的项目拖动至可视化编辑器，则将通过占位符 <Shift/Key variable> 配置此属性。 以下项将受到影响：按钮、框架、图像、线条、扇形、多边形、矩形、文本字段、滚动条

网格选项卡

Grid (网格)

可见	网格线在可视化编辑器中按距离大小可见
启用	网格线在可视化编辑器中按距离大小启用。项与网格对齐，所有位置值都在一条网格线上。启用网格时已在可视化中的项目不会自动对齐。为此，必须先将其拖动到其他位置。 网格线可能启用且不可见
大小	网格线距离以像素为单位

文件选项选项卡

文本“列出组件”的文本文件。	.csv 类型文件的文件名和位置。其包含一个表格，而表格包含文本列表格式的文本。 如果 **List components (列出组件)** 功能用作输入助手，则提供文件中的条目。 通过“命令：导入/导出文本列表”将此文件创建为全局文本列表的导出文件。

可视化目录

文本列表文件	文本列表的存储位置。 只要在项目设置、类别可视化、文本列表文件中未配置其他位置，此设置即可适用于项目。
图像文件	图像文件的存储位置。用分号分隔多个存储位置。 例如，TwinCAT 在导出或导入图像文件时使用此目录。 只要在项目属性、类别可视化、图像文件中未配置其他位置，此设置即在项目中有效。

另请参见：

- PLC 文档：创建可视化 [▶ 344]

17.9.1.15 对话框选项 - 可视化样式

功能：此对话框用于配置可视化样式。

调用：TwinCAT > PLC Environment > visualization styles（TwinCAT > PLC 环境 > 可视化样式）

样式选择

显示所有版本（仅限专家）	☐ 除了当前所选样式之外，存储库的所有其他样式均可供选择，但仅在最新版本中可用。如果为所选样式安装了较新版本，则也会列出这些新版本。 ☑ 所有已安装版本中的所有已安装样式均可供选择。

新可视化管理器的样式

最近使用：<style, version, manufacturer>	在您添加新的可视化应用时自动设置为已选择的样式。 尽管已进行此设置，根据设备的不同，显示变体会以不同方式显示。
默认：<style, version, manufacturer>	设置制造商的默认样式。
<Style, Version, Manufacturer>	为样式、版本和制造商设置显示变体。

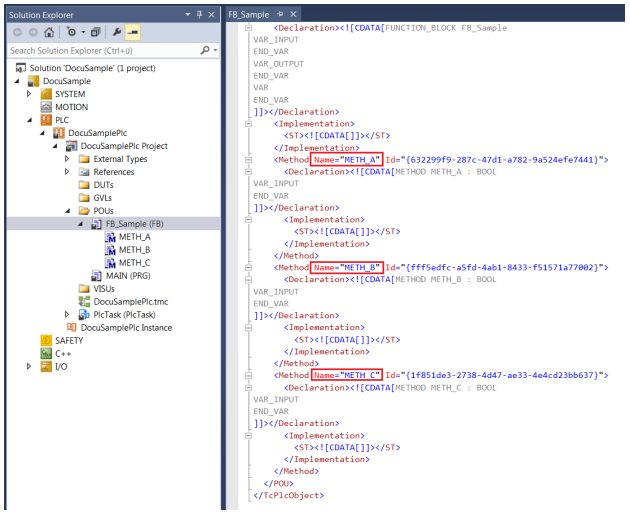
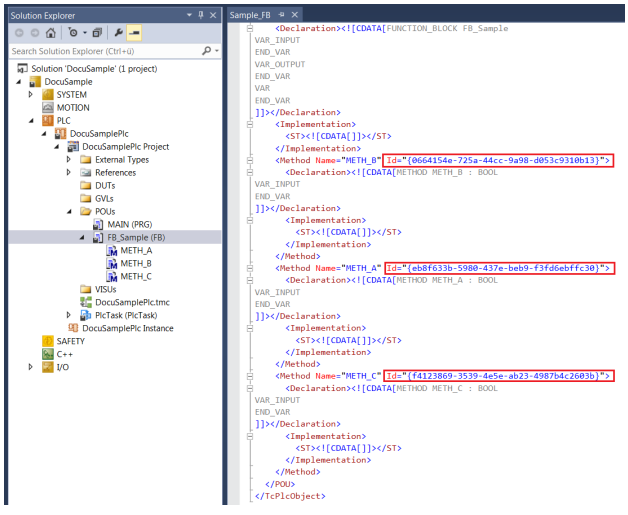
17.9.1.16 对话框选项 - 写入选项

写入选项

单独的行 ID	• TRUE：POU 的行 ID 存储在 1 个单独的文件（LineIDs.dbg）中，因此行 ID 的更改不会导致 POU 的更改，而 POU 的更改会在源码控制系统中被误解为内容更改。对于 TC3.1 Build 4026 及以上版本，对此的要求“Write Line IDs”（写入行 ID）的值为 TRUE。（默认设置：FALSE） 对于 TC3.1 Build 4024 及以下版本，例如，需要行 ID 进行断点处理，并确保可以将代码行分配给机器代码指令。
按名称排序	• TRUE（默认设置）：POU 的子元素（动作、方法、属性）按名称而非内部 ID 排序（请参见示例 [▶ 922]）。
写入行 ID	TC3.1 Build 4026 及以上可用 • TRUE：在新项目中为 POU 创建并存储行 ID。（默认设置：FALSE） 此设置为全局默认设置。在创建新的 PLC 项目时，该设置的值将被传输到本地项目设置中 1 次。这可以在 PLC 项目属性（高级类别 [▶ 853]）中找到，并可以在那里针对每个项目进行调整。
将 PLC 书签写入文件	TC3.1 Build 4026 及以上可用 • TRUE：在新项目中，书签存储在项目目录下的单独 .bookmarks 文件中。（默认设置：FALSE） 此设置为全局默认设置。在创建新的 PLC 项目时，该设置的值将被传输到本地项目设置中 1 次。这可以在 PLC 项目属性（高级类别 [▶ 853]）中找到，并可以在那里针对每个项目进行调整。

示例

示例说明了 METH_A、METH_B 和 METH_C 方法的不同存储顺序，具体取决于 **Sort by name**（按名称排序）选项是启用还是禁用。如果禁用该选项（FALSE），则 METH_B 方法不会根据其名称排在第 2 位，而是根据其内部 ID 排在第 1 位。

按名称排序 = TRUE	按名称排序 = FALSE
	

17.9.1.17 对话框选项 - ZIP 导出/导入

功能：此对话框用于配置 ZIP 导出和导入设置。

调用：TwinCAT > PLC Environment > ZIP export/import（TwinCAT > PLC 环境 > ZIP 导出/导入）

另请参见：

- PLC 文档：导出和导入 PLC 项目 [▶ 56]

17.9.2 命令：自定义

功能：此命令打开 **Customize（自定义）**对话框。对话框包含用于配置用户界面的选项卡。您可在此自定义符合您要求的菜单、工具栏和键盘映射。

调用：菜单 **Tools（工具）**

您可以随时通过 **Reset（重置）**按钮恢复 TwinCAT 标准设置。

另请参见：

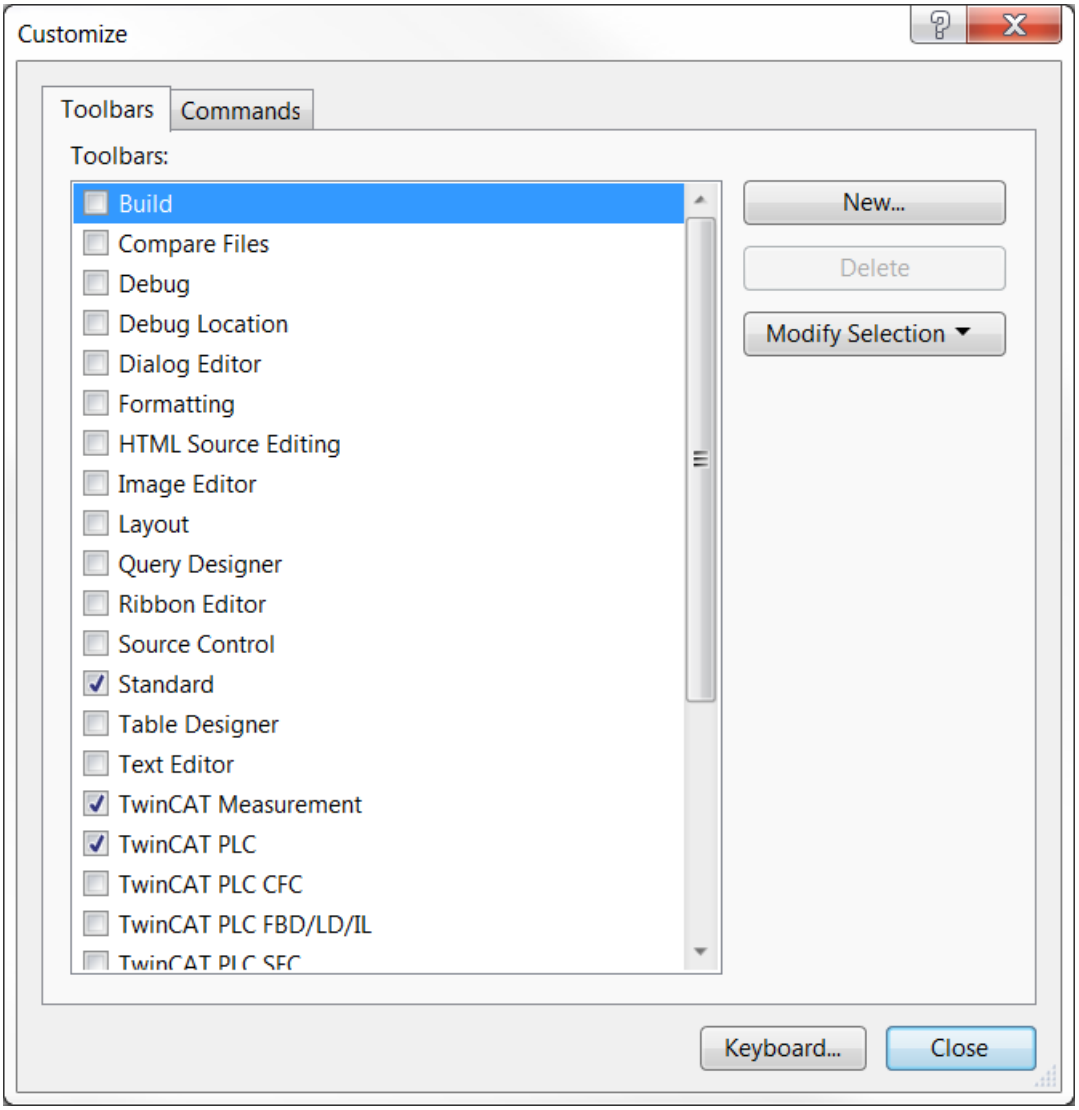
- TC3 用户界面文档 > 自定义菜单
- TC3 用户界面文档 > 自定义工具栏
- TC3 用户界面文档 > 自定义键盘快捷方式

17.9.2.1 对话框自定义 - 工具栏选项卡

功能：您可使用此对话框创建新的工具栏或调整现有工具栏。

调用：菜单 **Tools > Customize（工具 > 自定义）**

使用 **Close（关闭）**关闭对话框后，这些更改将在 TwinCAT 用户界面的菜单栏中显示。



新建... (添加工具栏)	TwinCAT 在所选工具栏上添加一个工具栏。对话框打开，可在其中输入名称。
Delete (删除) (删除工具栏)	TwinCAT 删除所选工具栏。 您只能删除自己创建的工具栏。
修改选择 (位置工具栏)	TwinCAT 将所选工具栏置于主窗口的上、下、左或右框
键盘....	打开 **Options (选项)** 对话框，您可以在其中定义键盘快捷方式。

工具栏

显示当前定义的工具栏。	
(隐藏)	隐藏用户界面上所选的工具栏。
(显示)	显示 TwinCAT 用户界面中所选的隐藏工具栏。

另请参见：

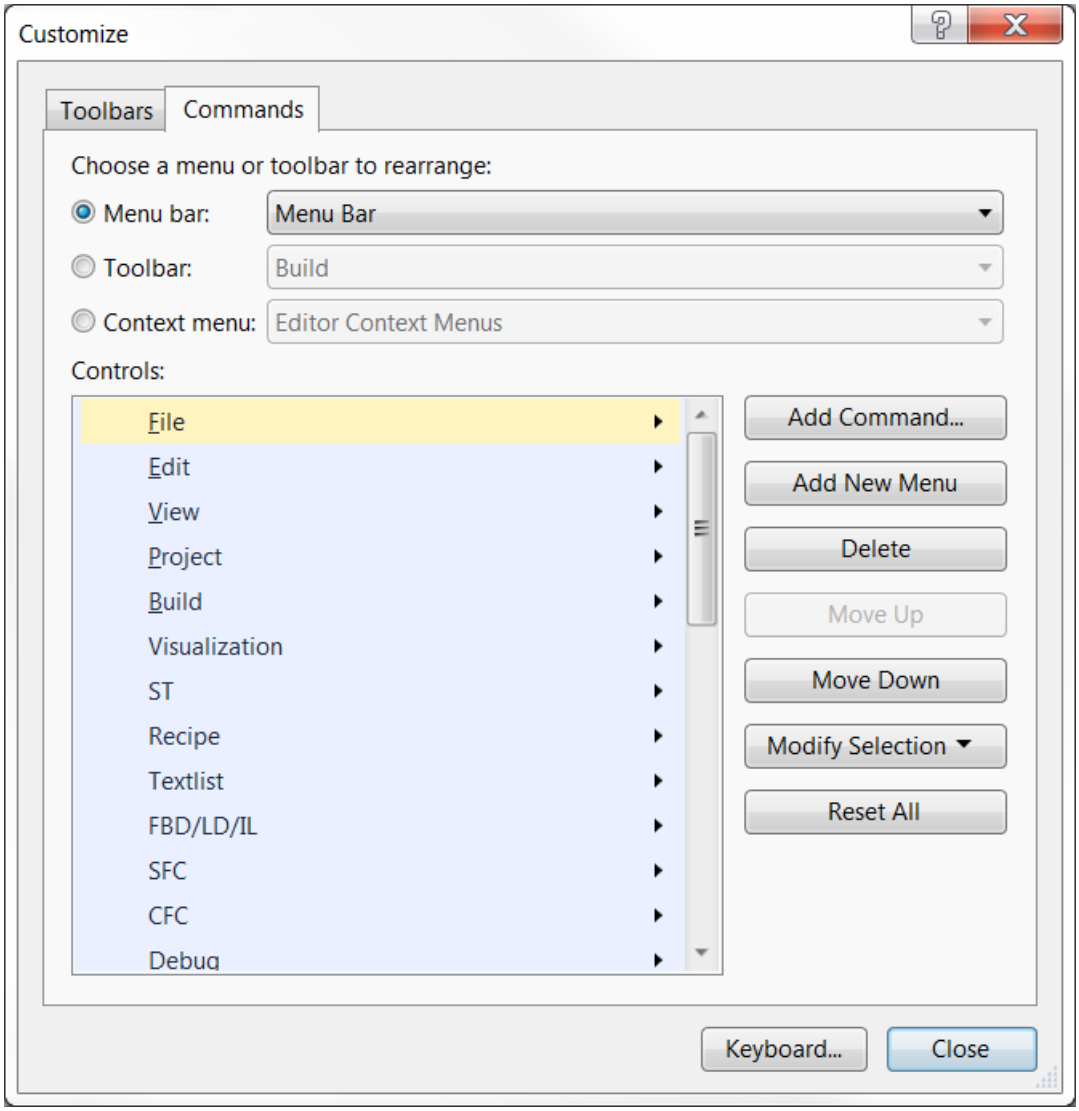
- TC3 用户界面文档 > 自定义工具栏

17.9.2.2 对话框自定义 - 命令选项卡

功能：您可通过此对话框定义命令、菜单的结构和内容以及用户界面的工具栏。

调用：菜单 **Tools** > **Customize（工具 > 自定义）**

使用 **Close（关闭）**关闭对话框后，这些更改将在 TwinCAT 用户界面的菜单栏中显示。



菜单和工具栏

显示当前定义的工具栏、菜单、子菜单以及其中所包含的命令。	
菜单栏	菜单和子菜单列表
工具栏	工具栏列表
上下文菜单	上下文菜单列表

控制

控制	所选菜单或工具栏中包含的命令或子菜单列表。从上到下的排列对应于之后在 TwinCAT 菜单或工具栏中显示的排列方式。

添加命令	打开 **Add Command（添加命令）** 对话框。**Add Command（添加命令）** 对话框用于选择一项或多项命令。左部：类别列表。右部：所选类别的命令列表。 在所选命令上添加命令。
添加新菜单	在所选菜单上添加新菜单。
修改选择	打开可确定新添加菜单名称的菜单。
Delete（删除）	删除所选菜单或命令。
向上移动	按命令或菜单顺序向上移动所选命令或菜单。
向下移动	按命令或菜单顺序向下移动所选命令或菜单。
全部重置	将整个菜单重置为默认设置。
键盘	打开 **Options（选项）** 对话框，您可以在其中定义键盘快捷方式。

17.10 窗口

17.10.1 命令：浮动

功能： 此命令将停靠（固定）至用户界面边框的视图或窗口从边框分离，并将其作为独立窗口置于屏幕上。

调用： **Window（窗口）** 菜单，视图标题或选项卡（窗口）中的上下文菜单或按钮

之后，可以将该视图放在用户界面之外。使用 **Dock（程序坞）** 命令将分离的视图重新附加到用户界面边框。

另请参见：

- 命令：程序坞 [▶ 926]
- TC3 用户界面文档：排列视图和窗口

17.10.2 命令：程序坞

功能： 此命令将之前通过 **Float（浮动）** 命令分离并且目前作为单独视图位于屏幕上的视图与用户界面边框重新“对接”。

调用： **Window（窗口）** 菜单，视图标题或选项卡（窗口）中的上下文菜单或按钮

另请参见：

- 命令：浮动 [▶ 926]
- TC3 用户界面文档：排列视图和窗口

17.10.3 命令：隐藏

符号：

功能： 此命令隐藏视图。

调用： **Window（窗口）** 菜单，视图标题中的上下文菜单或按钮

要求： 视图已启用。

隐藏指视图被关闭。因此，该命令相当于通过视图标题栏中的 按钮关闭视图。在 **View（视图）** 菜单中使用此命令可取消隐藏或打开隐藏的视图。

另请参见：

- 视图 [▶ 826]
- TC3 用户界面文档：显示/隐藏视图

17.10.4 命令：自动隐藏全部

功能：此命令隐藏所有视图。

调用：Window（窗口）菜单，视图标题中的上下文菜单或按钮

隐藏指 TwinCAT 在用户界面内仅以选项卡形式显示所有视图，并且仅当您点击选项卡时才可见。如果您在之后点击视图标题栏中的  按钮或选择 **Dock（程序坞）**命令，则该视图将被重新固定在用户界面。

另请参见：

- 命令：程序坞 [▶ 926]
- 视图 [▶ 826]
- TC3 用户界面文档：显示/隐藏视图

17.10.5 命令：自动隐藏

符号： 

功能：此命令将视图置于后台。

调用：Window（窗口）菜单，视图标题中的上下文菜单或按钮

要求：视图已启用。

置于后台表示 TwinCAT 在用户界面内仅以选项卡形式显示视图，并且仅当您点击选项卡时才可见。如果您在之后再次点击标题栏中的  按钮或选择 **Dock（程序坞）**命令，则该视图将被重新固定在用户界面。

另请参见：

- 命令：程序坞 [▶ 926]
- TC3 用户界面文档：显示/隐藏视图

17.10.6 命令：固定选项卡

符号： 

功能：此命令将当前活动的选项卡固定在主窗口左边缘。

调用：Window（窗口）菜单，选项卡（窗口）标题栏中的上下文菜单或按钮

要求：选项卡（窗口）已启用。

另请参见：

- TC3 用户界面文档：显示/隐藏视图

17.10.7 命令：新水平选项卡组

符号： 

功能：此命令将活动窗口移动到现有选项卡下的一个单独的新选项卡组。

调用：Window（窗口）菜单，选项卡（窗口）标题栏的上下文菜单

要求：若干编辑器窗口作为选项卡依次排列。

如果在编辑器中打开另一个对象，则会被自动放置在重点关注的选项卡组中。

另请参见：

- TC3 用户界面文档：排列视图和窗口
- 命令：新垂直选项卡组 [▶ 928]

17.10.8 命令：新垂直选项卡组

符号： 

功能：此命令将活动窗口移动到现有选项卡右侧的一个单独的新选项卡组。

调用：Window（窗口）菜单，选项卡（窗口）标题栏的上下文菜单

要求：若干编辑器窗口作为选项卡依次排列。

如果在编辑器中打开另一个对象，则会被自动放置在重点关注的选项卡组中。

另请参见：

- TC3 用户界面文档：排列视图和窗口
- 命令：新水平选项卡组 [▶ 927]

17.10.9 命令：重置窗口布局

功能：此命令将所有当前打开的窗口和视图重置到默认位置。您需要在执行前确认命令。

调用：Window（窗口）菜单

17.10.10 命令：关闭所有文件

符号： 

功能：此命令关闭所有当前打开的编辑器窗口。

调用：Window（窗口）菜单

要求：至少打开一个编辑器窗口。

另请参见：

- TC3 用户界面文档：显示/隐藏视图

17.10.11 命令：窗口

功能：此命令打开 Window（窗口）对话框，其中显示所有打开的对象。您可以启动或关闭其中的窗口。

调用：Window（窗口）菜单

17.10.12 窗口子菜单命令

功能：此命令启用所选窗口。

调用：Window（窗口）菜单

在每个打开的编辑器窗口中，Window（窗口）菜单包含 <n><Object name> 命令，您可通过此命令启用该窗口，即重点关注该窗口。在脱机模式中，TwinCAT 在该命令后添加扩展名（脱机）。为功能块添加扩展名（impl）或 <instance path> 以区分实现和实例。

另请参见：

- 命令：窗口 [▶ 928]

17.11 SFC

17.11.1 命令：初始化步骤

符号： 

功能：此命令将当前所选步骤转换为初始化步骤。

调用：菜单 **SFC**，上下文菜单

执行该命令可以将步骤元素的边框更改为双行。该步骤（之前是初始化步骤）自动成为“普通”步骤并通过单边框表示。

也可在步骤元素的 **Properties（属性）** 视图中启用或禁用属性 **Init step（初始化步骤）**，尽管在此情况下 TwinCAT 不会自动调整其他步骤的设置。

如果您要更改图表，则此命令十分有用。创建新的 SFC 对象时将自动包含一个初始化步骤，然后转换（TRUE）并跳转回初始化步骤。

注意：可以在在线模式下使用 SFC 标记 SFCInit 和 SFCReset 将图表重置为初始化步骤。

另请参见：

- PLC 文档：顺序功能图（SFC） [▶ 114]
- PLC 文档：SFC 编辑器 [▶ 580]
- PLC 文档：SFC 元素属性 [▶ 593]

17.11.2 命令：插入步骤转换

符号： 

功能：此命令在当前所选位置前添加一个步骤和转换。

调用：菜单 **SFC**，上下文菜单

如果您已选择一个步骤，TwinCAT 将插入新的步骤转换组合。如果您已选择一个转换，则新的转换步骤组合已插入。

新步骤的默认名称为步骤 <n>。n 是一个序号，第一步从 0 开始，并且在初始化步骤之外添加。因此，新转换的默认名称为 Trans<n>。您可以通过单击名称直接编辑默认名称。

另请参见：

- 命令：在...后插入步骤-转换 [▶ 929]
- PLC 文档：顺序功能图（SFC） [▶ 114]
- PLC 文档：SFC 编辑器 [▶ 580]
- PLC 文档：SFC 元素步骤和转换 [▶ 588]

17.11.3 命令：在...后插入步骤-转换

符号：

功能：此命令在当前所选位置后添加一个步骤和转换。

调用：菜单 **SFC**，上下文菜单

如果您已选择一个步骤，TwinCAT 将插入新的转换步骤组合。如果您已选择一个转换，则新的步骤转换组合已插入。

新步骤的默认名称为步骤 <n>。n 是一个序号，第一步从 0 开始，并且在初始化步骤之外添加。因此，新转换的默认名称为 Trans<n>。您可以通过单击名称直接编辑默认名称。

另请参见：

- 命令：插入步骤转换 [▶ 929]
- PLC 文档：顺序功能图（SFC） [▶ 114]
- PLC 文档：SFC 编辑器 [▶ 580]
- PLC 文档：

17.11.4 命令：平行

符号： 

功能：此命令将所选替代分支转换为平行分支。

调用：菜单 SFC，上下文菜单

要求：选择分支的平行连接行。

注意：转换分支后，您必须检查并调整分支前后的步骤和转换顺序。

另请参见：

- 命令：替代 [▶ 930]
- PLC 文档：顺序功能图（SFC） [▶ 114]
- PLC 文档：SFC 编辑器 [▶ 580]

17.11.5 命令：替代

符号： 

功能：此命令将所选平行分支转换为替代分支。

调用：菜单 SFC，上下文菜单

要求：选择分支的平行连接行。

注意：转换分支后，您必须检查并调整分支前后的步骤和转换顺序。

另请参见：

- 命令：平行 [▶ 930]
- PLC 文档：顺序功能图（SFC） [▶ 114]
- PLC 文档：SFC 编辑器 [▶ 580]

17.11.6 命令：插入分支

符号： 

功能：此命令在当前所选位置左侧添加分支。

调用：菜单 SFC，上下文菜单

该命令的效果对应于命令 **Insert branch right（在右侧插入分支）**。

另请参见：

- 命令：在右侧插入分支 [▶ 931]
- PLC 文档：顺序功能图（SFC） [▶ 114]
- PLC 文档：SFC 编辑器 [▶ 580]

- PLC 文档：SFC 元素分支 [▶ 591]

17.11.7 命令：在右侧插入分支

符号：

功能：此命令在当前所选位置右侧添加分支。

调用：菜单 SFC，上下文菜单

所插入的分支类型取决于所选元素：

- 如果当前所选元素最上方的元素是一个转换或一个替代分支，TwinCAT 将插入一个替代分支。
- 如果当前所选元素最上放的元素是一个步骤、一个宏、一个跳转或一个平行分支，TwinCAT 将添加一个带标签的平行分支：分支 <x>，其中 x 是一个序号。您可以编辑此默认标签名称。您可以将标签指定为跳转目标。
- 如果当前选择了现有分支的公共元素（水平线），TwinCAT 将最右边的新分支添加为附加分支。如果当前选择了现有分支的整个分支，TwinCAT 将新分支作为新分支直接添加到其右侧。

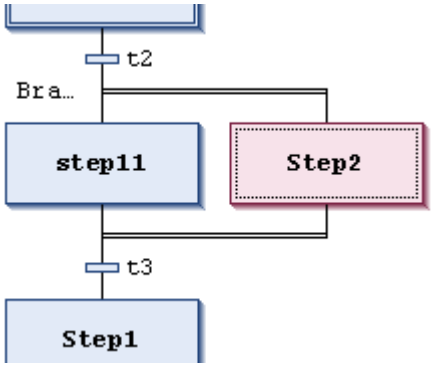
注意：您可以使用命令 **Alternative（替代）**或 **Parallel（平行）**以将分支转换为相应的其他类型。

示例：平行分支

下图显示了一个新插入的平行分支，通过“命令：在右侧插入分支”生成，同时已选步骤 11。TwinCAT 自动插入步骤（示例中的步骤 2）。

在在线模式下处理：如果 t2 返回 TRUE，TwinCAT 将在步骤 11 之后立即执行步骤 2，然后再评估 t3。

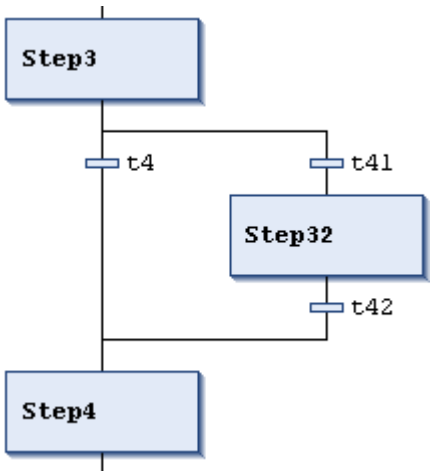
与替代分支相反，TwinCAT 执行两个分支。



示例：替代分支

下图显示了一个新插入的替代分支，通过“命令：在右侧插入分支”生成，同时已选转换 t4。TwinCAT 自动插入一个步骤（示例中的步骤 32）、一个之前的转换和一个之后的转换（t41、t42）。

在在线模式下处理：如果步骤 3 处于活动状态，TwinCAT 将从左到右评估以下转换（t4、t41）。执行第一次转换返回 TRUE 的分支的第一个连接点。与平行分支相反，仅执行一个连接点。



另请参见：

- 命令：替代 [▶ 930]
- 命令：平行 [▶ 930]
- 命令：插入分支 [▶ 930]
- PLC 文档：顺序功能图（SFC） [▶ 114]
- PLC 文档：SFC 编辑器 [▶ 580]
- PLC 文档：SFC 元素分支 [▶ 591]

17.11.8 命令：插入动作关联

符号：

功能：此命令将 IEC 动作分配到一个步骤。

调用：菜单 SFC，上下文菜单

要求：选择一个步骤。

TwinCAT 在当前所选步骤元素右侧添加动作元素。

如果您已将一个或多个动作分配至该步骤，它们将从上至下依次显示在“操作列表”中。之后，新动作排列如下：

- 如果您选择了步骤元素，则作为该步骤的第一个操作，即位于操作列表顶部。
- 如果您选择了步骤操作列表中的一个当前操作，则位于该操作之前，即在该操作上方。

操作元素左部包含限定符，默认为 N，您可在右部输入操作名称。为此，点击方框以编辑边框。您必须创建此操作作为项目中的 POU。

您还可以编辑限定符。关于有效限定符列表的描述，请参见“SFC 中的操作限定符”一节。

另请参见：

- 命令：在...后插入动作关联 [▶ 932]
- PLC 文档：顺序功能图（SFC） [▶ 114]
- PLC 文档：SFC 编辑器 [▶ 580]
- PLC 文档：SFC 中的操作限定符 [▶ 583]

17.11.9 命令：在...后插入动作关联

符号：

功能：此命令将 IEC 动作分配到一个步骤。

调用：菜单 SFC，上下文菜单

要求：选择一个步骤。

该命令对应于插入操作关联的描述。不同之处在于，TwinCAT 未将新操作放在操作列表的第一个位置，而是放在操作列表的最后一个位置。如果您在操作列表中选择了一个操作，TwinCAT 不会将新操作放在其上方，而是放在下方。

另请参见：

- 命令：插入动作关联 [▶ 932]
- PLC 文档：顺序功能图（SFC） [▶ 114]
- PLC 文档：SFC 编辑器 [▶ 580]
- PLC 文档：SFC 中的操作限定符 [▶ 583]

17.11.10 命令：插入跳转

符号：

功能：此命令在当前所选元素前插入一个跳转元素。

调用：菜单 SFC，上下文菜单

要求：选择一个步骤。

TwinCAT 通过跳转目标步骤自动插入跳转。然后必须用实际跳转目标替换此跳转目标。您可以选择带输入助手的目标。

另请参见：

- 命令：在...后插入跳转 [▶ 933]
- PLC 文档：顺序功能图（SFC） [▶ 114]
- PLC 文档：SFC 编辑器 [▶ 580]
- PLC 文档：SFC 元素跳转 [▶ 592]

17.11.11 命令：在...后插入跳转

符号：

功能：此命令在当前所选元素后插入一个跳转元素。

调用：SFC 菜单

TwinCAT 通过跳转目标步骤自动插入跳转。然后必须用实际跳转目标替换此跳转目标。您可以选择带输入助手的目标。

另请参见：

- 命令：插入跳转 [▶ 933]
- PLC 文档：顺序功能图（SFC） [▶ 114]
- PLC 文档：SFC 编辑器 [▶ 580]
- PLC 文档：SFC 元素跳转 [▶ 592]

17.11.12 命令：插入宏

符号：

功能：此命令在当前所选元素前插入一个宏元素。

调用：菜单 SFC，上下文菜单

新宏的默认名称为宏 <x>。x 是一个序号，第一个宏从 0 开始。您可以通过单击名称直接编辑默认名称。

如要编辑宏，使用宏编辑器中的 **Show macro（显示宏）** 命令打开它。

另请参见：

- 命令：显示宏 [▶ 934]
- 命令：在...后插入宏 [▶ 934]
- PLC 文档：顺序功能图（SFC） [▶ 114]
- SFC 编辑器 [▶ 580]

17.11.13 命令：在...后插入宏

符号：

功能：此命令在当前所选元素后插入一个宏元素。

调用：菜单 **SFC**，上下文菜单

该命令对应于命令 **Insert macro（插入宏）** 的描述。

另请参见：

- 命令：插入宏 [▶ 933]
- 命令：显示宏 [▶ 934]
- PLC 文档：顺序功能图（SFC） [▶ 114]
- PLC 文档：SFC 编辑器 [▶ 580]

17.11.14 命令：显示宏

符号：

功能：此命令在宏编辑器中打开一个宏以进行编辑。

调用：SFC 菜单

要求：已选择一个宏。

该命令使 TwinCAT 关闭 SFC 编辑器的主视图，转而打开宏编辑器。这也是一个 SFC 编辑器，您现在可在其中编辑 SFC 示意图的一部分，该部分在主视图中显示为宏框。

使用命令 **Exit macro（编辑宏）** 返回至主视图。

另请参见：

- 命令：退出宏 [▶ 934]
- PLC 文档：顺序功能图（SFC） [▶ 114]
- PLC 文档：SFC 编辑器 [▶ 580]

17.11.15 命令：退出宏

符号：

功能：此命令关闭宏编辑器并且返回 SFC 编辑器的主视图。

调用：SFC 菜单

要求：在宏编辑器中打开一个宏。

另请参见：

- 命令：显示宏 [▶ 934]

- PLC 文档：顺序功能图（SFC） [▶ 114]
- PLC 文档：SFC 编辑器 [▶ 580]

17.11.16 命令：在...后插入

符号：

功能：此命令在当前所选位置后插入剪贴板中的元素。

调用：SFC 菜单

17.11.17 命令：添加输入操作

符号：

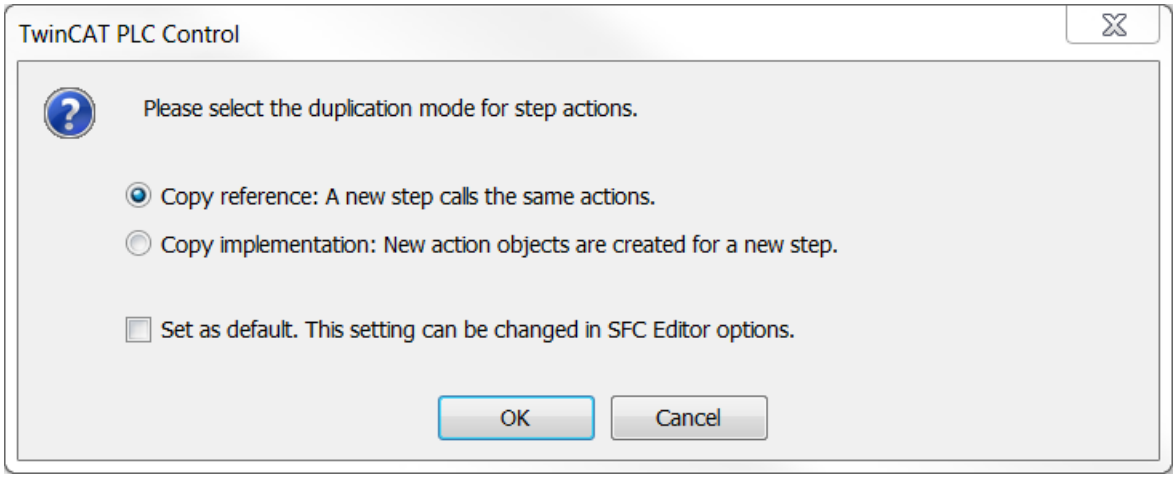
功能：此命令引导至 **Add entry action（添加输入操作）**对话框，您可在其中定义类型为“输入操作”的新步骤操作。根据 SFC 选项，可能会预先出现选择新步骤操作复制模式的提示。

调用：SFC 菜单，所选步骤元素的上下文菜单

要求：选择一个步骤元素。

在 ST 编辑器中自动打开输入操作。在左下角为该步骤元素分配一个“E”。

用于选择复制模式的查询对话框



复制引用。新步骤将调用相同的操作	如在 SFC 内复制该步骤，则也会复制该步骤操作的链接。因此，彼此复制的步骤将调用所有相同的操作。
复制实现。为新步骤创建新的操作对象	这意味着为复制的步骤“嵌入”步骤操作。默认设置下，新创建的操作对象出现在解决方案资源管理器 PLC 项目树中的 SFC 功能块下方。最初，这些对象包含各自操作的原始实现代码副本。
设置为默认。可在 SFC 编辑器选项中更改此设置。	对话框中的设置作为默认设置接受。您可以在类别 **SFC editor（SFC 编辑器）**的 TwinCAT 选项中更改默认设置。为此，在下拉列表 **Standard insert method（标准插入方法）**的组字段 **Step actions（步骤操作）**中，选择 **Always ask（始终询问）**、**Copy reference（复制引用）**或 **Duplicate implementation（复制实现）**条目。

另请参见：

- 命令：选项 > 对话框选项 - SFC 编辑器 [▶ 907]
- PLC 文档：命令：添加退出操作 [▶ 936]

- PLC 文档：顺序功能图（SFC） [▶ 114]
- PLC 文档：SFC 编辑器 [▶ 580]
- PLC 文档：用顺序功能图（SFC）编程 [▶ 114]
- PLC 文档：SFC 元素动作 [▶ 588]

17.11.18 命令：添加退出操作

符号：

功能：此命令引导至 **Add exit action（添加退出操作）**对话框，您可在其中定义类型为“输入操作”的新步骤操作。根据 SFC 选项，可能会预先出现选择新步骤操作复制模式的提示。请参见命令 **Add entry action（添加输入操作）**的帮助页面。

调用：SFC 菜单，所选步骤元素的上下文菜单

要求：选择 SFC 中的步骤元素。

另请参见：

- 命令：添加输入操作 [▶ 935]
- 命令：选项 > 对话框选项 - SFC 编辑器 [▶ 907]
- PLC 文档：顺序功能图（SFC） [▶ 114]
- PLC 文档：用顺序功能图（SFC）编程 [▶ 114]
- PLC 文档：SFC 编辑器 [▶ 580]
- PLC 文档：SFC 元素动作 [▶ 588]

17.11.19 命令：更改复制 - 设置

功能：此命令将 SFC 功能块中的步骤或转换调用的各步骤操作或转换永久链接到调用端。操作或转换对象只能由该调用端调用（伪嵌入）。因此，复制调用操作或转换的步骤和转换元素会自动创建新的操作或转换对象。在每种情况下都会复制实现代码。

调用：SFC 菜单

有关复制模式的详细信息，请参见帮助页面以了解 SFC 元素属性和添加步骤操作的指令。

另请参见：

- PLC 文档：SFC 元素属性 [▶ 593]
- PLC 文档：用顺序功能图（SFC）编程 [▶ 114]

17.11.20 命令：更改复制 - 删除

功能：此命令通过调用整个 SFC 功能块的步骤或转换来删除动作或转换对象的固定链接。这取消了操作或转换对象的伪嵌入。如果之后复制调用操作或转换的步骤和转换元素，则副本将调用与源相同的操作和转换。

调用：SFC 菜单

有关复制模式的详细信息，请参见帮助页面以了解 SFC 元素属性和添加步骤操作的指令。

另请参见：

- PLC 文档：SFC 元素属性 [▶ 593]
- PLC 文档：用顺序功能图（SFC）编程 [▶ 114]

17.11.21 命令：插入步骤

此命令不作为标准包含在 SFC 菜单中。

符号：

功能：此命令在当前所选位置前添加一个步骤。

调用：SFC 菜单，SFC 编辑器中的上下文菜单

新步骤的默认名称为步骤 <n>。n 是一个序号，第一步从 0 开始，并且在初始化步骤之外添加。可以通过点击名称进行编辑。

插入不带转换的步骤或不带步骤的转换将导致编译错误。

另请参见：

- 命令：在...后插入步骤-转换 [▶ 929]
- 命令：初始化步骤 [▶ 929]
- PLC 文档：SFC 元素步骤和转换 [▶ 588]

17.11.22 命令：在...后插入步骤

此命令不作为标准包含在 SFC 菜单中。

符号：

功能：此命令在当前所选位置后插入一个步骤。

调用：SFC 菜单，SFC 编辑器中的上下文菜单

新步骤的默认名称为步骤 <n>。n 是一个序号，第一步从 0 开始，并且在初始化步骤之外添加。可以通过点击名称进行编辑。

插入不带转换的步骤或不带步骤的转换将导致编译错误。

另请参见：

- 命令：初始化步骤 [▶ 929]
- 命令：在...后插入步骤-转换 [▶ 929]
- PLC 文档：SFC 元素跳转 [▶ 592]

17.11.23 命令：插入转换

此命令不作为标准包含在 SFC 菜单中。

符号：

功能：此命令在当前所选位置前添加一个转换。

调用：SFC 菜单，SFC 编辑器中的上下文菜单

新转换的默认名称为转换 <n>。n 是一个序号，第一个转换从 0 开始。可以通过点击名称进行编辑。

插入不带转换的步骤或不带步骤的转换将导致编译错误。

另请参见：

- 命令：在...后插入步骤-转换 [▶ 929]
- PLC 文档：SFC 元素步骤和转换 [▶ 588]

17.11.24 命令：在...后插入转换

此命令不作为标准包含在 SFC 菜单中。

符号： +↓

功能：此命令在当前所选位置后插入一个转换。

调用： SFC 菜单，SFC 编辑器中的上下文菜单

新转换的默认名称为转换 <n>。n 是一个序号，第一个转换从 0 开始。可以通过点击名称进行编辑。

插入不带转换的步骤或不带步骤的转换将导致编译错误。

另请参见：

- 命令：在...后插入步骤 [▶ 937]
- PLC 文档：SFC 元素步骤和转换 [▶ 588]

17.12 CFC

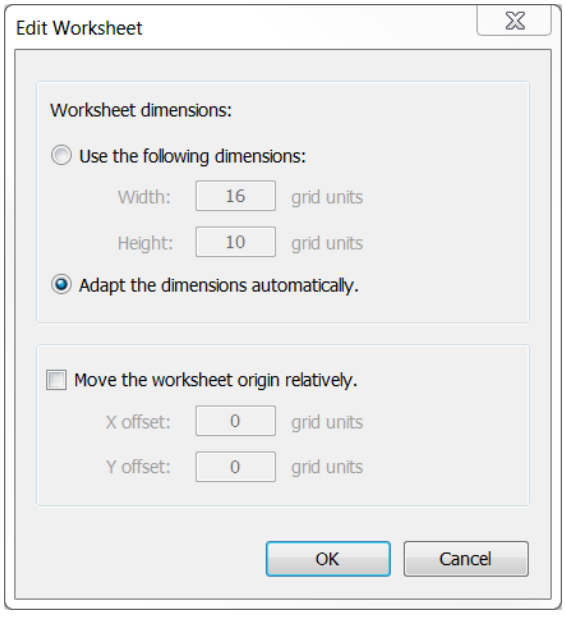
17.12.1 命令：编辑工作表

功能： 此命令打开 Edit Worksheet（编辑工作表）对话框，您可在其中指定工作表的大小。

调用： CFC 菜单

要求： CFC 编辑器处于活动状态。

编辑工作表对话框



使用以下尺寸	您可在此设置工作表的大小。只有尺寸大小满足现有程序时，才接受您的更改。
自动调整尺寸	根据您程序的大小自动调整工作表尺寸。
相对移动工作表来源	在 x 或 y 轴上移动工作表。允许输入负值。

另请参见：

- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]

17.12.2 命令：编辑页面大小

功能：此命令打开编辑页面大小对话框。可使用该对话框更改以页面为导向的 CFC 编辑器的大小。

调用：CFC 菜单

要求：以页面为导向的 CFC 编辑器处于活动状态。

对话框：编辑页面大小

宽度	页面宽度（最小 24，最大 1024）。工作区域外的元素以红色高亮显示。
高度	页面高度（最小 24，最大 1024）。工作区域外的元素以红色高亮显示。
边框宽度	边距宽度（最小 6，最大 25% 或页面宽度）。
设置为新 CFC 对象的默认设置	 ：当前设置被设置为新 CFC 对象的默认设置。

另请参见：

- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]
- PLC 文档：CFC 元素页面 [▶ 617]

17.12.3 命令：否定

符号： 

功能：此命令否定功能块的输入或输出。

调用：CFC 菜单，上下文菜单

要求：CFC 编辑器处于活动状态。选择一个功能块输入或输出。

另请参见：

- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]

17.12.4 命令：EN/ENO

符号： 

功能：此命令向所选功能块添加布尔输入 EN（启用）和布尔输出 ENO（启用输出）。

调用：CFC 菜单，上下文菜单

要求：CFC 编辑器处于活动状态。选择一个功能块。

添加的输入“EN”启用功能块。仅当它的值为 TRUE 时执行功能块。在输出 ENO 处输出该信号的值。

另请参见：

- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]

17.12.5 命令：无

符号： 

功能：此命令删除来自“输出”元素输入的重置（R）、设置（S）或 REF。

调用：CFC > Set/Reset（CFC > 设置/重置）菜单；上下文菜单 > 设置/重置

要求：CFC 编辑器处于活动状态。选择 1 个 **Output**（输出）元素的输入。

另请参见：

- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]

17.12.6 命令 R（重置）

符号： 

功能：此命令将重置添加到布尔元素**输出**的输入。

调用：菜单 **CFC > Set/Reset (CFC > 设置/重置)**，上下文菜单 > 设置/重置

要求：CFC 编辑器处于活动状态。选择一个“输出”元素的输入。

如果**输出**元素有一个重置输入，则当输入值为 TRUE 时，布尔输出值被设置为 FALSE。保留输出端的值 FALSE，即使输入值再次被更改也不例外。

另请参见：

- 命令 S（设置） [▶ 940]
- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]

17.12.7 命令 S（设置）

符号： 

功能：此命令将设置 (S) 添加到布尔元素“输出”的输入。

调用：菜单 **CFC > Set/Reset (CFC > 设置/重置)**，上下文菜单 > 设置/重置

要求：CFC 编辑器处于活动状态。选择一个**输出**元素的输入。

如果**输出**元素有一个设置输入，则当输入值为 TRUE 时，布尔输出值被设置为 TRUE。保留输出端的值 TRUE，即使输入值再次被更改也不例外。

另请参见：

- 命令 R（重置） [▶ 940]
- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]

17.12.8 命令：REF = (引用分配)

符号： 

功能：此命令将引用分配到“输出”元素。

调用：菜单 **CFC > Set/Reset (CFC > 设置/重置)**，上下文菜单 > 设置/重置

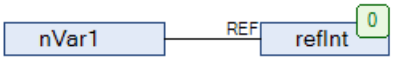
要求：CFC 编辑器处于活动状态。选择一个**输出**元素的输入。

示例：

声明：

```
refInt : REFERENCE TO INT;
nVar1 : INT;
```

CFC：



这对应于 ST 代码

```
refInt REF= nVar1;
```

更多信息，请参见对数据类型的引用中的描述。

另请参见：

- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]
- PLC 文档：引用 [▶ 712]

17.12.9 命令显示执行顺序

符号： 

功能：此命令简要显示编程对象的所有 CFC 元素的编号标记。

调用：

- **CFC > Execution order**（CFC > 执行顺序）菜单
- CFC 编辑器中的上下文菜单 > **Execution order**（执行顺序）

要求：CFC 编辑器处于活动状态，自动数据流模式已启用。

数字反映的是自动确定的执行顺序。执行顺序根据数据流确定，而且，在多个网络的情况下，则根据它们在编辑器中的拓扑位置确定。

只要您点击进入 CFC 编辑器，标记就会隐藏。

另请参见：

- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]
- PLC 文档：按数据流自动执行顺序 [▶ 110]
- PLC 文档：CFC 设置 [▶ 840]

17.12.10 命令设置反馈开始

符号： 

功能：此命令将所选元素定义为反馈循环的起点。

调用：

- **CFC > Execution order**（CFC > 执行顺序）菜单
- CFC 编辑器中的上下文菜单 > **Execution order**（执行顺序）

要求：CFC 编辑器处于活动状态，自动数据流模式已启用。CFC 编程块的网络维持 1 个反馈循环，选择该反馈循环中的 1 个元素。

在 CFC 编辑器中，反馈中的起点用符号  进行装饰。该元素在反馈中具有最低的执行顺序号。在运行时，反馈的处理从该元素开始。

另请参见：

- PLC 文档：命令显示执行顺序 [▶ 941]
- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]
- PLC 文档：按数据流自动执行顺序 [▶ 110]
- PLC 文档：CFC 设置 [▶ 840]

17.12.11 命令：移动到开始

符号： 

功能：此命令对元素进行编号，以便所选元素位于执行顺序的开头。

调用：CFC > Execution order（CFC > 执行顺序）菜单、上下文菜单 > 执行顺序

要求：CFC 编辑器处于活动状态，显式执行顺序模式已启用。选择至少 1 个元素。

选中的元素会被分配到从 0 开始的最小数字，同时保持先前的顺序。对其余元素进行编号，使其执行顺序保持不变。元素的拓扑位置未改变。

另请参见：

- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]
- PLC 文档：按数据流自动执行顺序 [▶ 110]
- PLC 文档：CFC 设置 [▶ 840]

17.12.12 命令：移动到结束

符号： 

功能：此命令对元素进行编号，以便所选元素位于执行顺序的最后。

调用：CFC > Execution order（CFC > 执行顺序）菜单、上下文菜单 > 执行顺序

要求：CFC 编辑器处于活动状态，显式执行顺序模式已启用。选择至少 1 个元素。

选中的元素会被分配到最大数字，同时保持先前的顺序。对其余元素进行编号，使其执行顺序保持不变。元素的拓扑位置未改变。

另请参见：

- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]
- PLC 文档：按数据流自动执行顺序 [▶ 110]
- PLC 文档：CFC 设置 [▶ 840]

17.12.13 命令：向前移动一位

符号： 

功能：此命令对元素进行编号，以便所选元素位于前一位。

调用：CFC > Execution order（CFC > 执行顺序）菜单、上下文菜单 > 执行顺序

要求：CFC 编辑器处于活动状态，显式执行顺序模式已启用。选择至少 1 个元素。

选中元素的编号会减少 1 个，同时保留先前的顺序，以便提前 1 个执行位置处理它们。对其余元素进行编号，使其执行顺序保持不变。元素的拓扑位置未改变。

另请参见：

- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]
- PLC 文档：按数据流自动执行顺序 [▶ 110]
- PLC 文档：CFC 设置 [▶ 840]

17.12.14 命令：向后移动一位

符号： 

功能：此命令对元素进行编号，以便所选元素位于后一位。

调用：CFC > Execution order（CFC > 执行顺序）菜单、上下文菜单 > 执行顺序

要求：CFC 编辑器处于活动状态，显式执行顺序模式已启用。选择至少 1 个元素。

选中元素的编号会增加 1 个，同时保留先前的顺序，以便推迟 1 个执行位置处理它们。对其余元素进行编号，使其执行顺序保持不变。元素的拓扑位置未改变。

另请参见：

- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]
- PLC 文档：按数据流自动执行顺序 [▶ 110]
- PLC 文档：CFC 设置 [▶ 840]

17.12.15 命令：设置执行次序

功能：此命令打开用于将所选元素的编号设置为任何值的对话框。

调用：CFC > Execution order（CFC > 执行顺序）菜单、上下文菜单 > 执行顺序

要求：CFC 编辑器处于活动状态，显式执行顺序模式已启用。只选择 1 个元素。

所选元素将获得在对话框中指定的编号。对其余元素进行编号，使其执行顺序保持不变。元素的拓扑位置未改变。

另请参见：

- PLC 文档：命令按数据流排序 [▶ 944]
- PLC 文档：命令按拓扑结构排序 [▶ 944]
- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]
- PLC 文档：按数据流自动执行顺序 [▶ 110]
- PLC 文档：CFC 设置 [▶ 840]

17.12.16 命令：按数据流排序

功能：此命令根据数据流对程序中的元素进行编号，而且，在多个网络的情况下，则根据它们在编辑器中的拓扑位置进行编号。

调用：CFC > Execution order（CFC > 执行顺序）菜单、上下文菜单 > 执行顺序

要求：CFC 编辑器处于活动状态，显式执行顺序模式已启用。

执行顺序根据数据流和（在多个网络的情况下）网络的拓扑位置进行排列。编程块的所有编号元素都会进行相应地设置。这样，执行顺序与自动数据流模式下的执行顺序相同。元素的拓扑位置未改变。

另请参见：

- PLC 文档：命令按拓扑结构排序 [▶ 944]
- PLC 文档：命令设置执行顺序 [▶ 943]
- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]
- PLC 文档：按数据流自动执行顺序 [▶ 110]
- PLC 文档：CFC 设置 [▶ 840]

17.12.17 命令：按拓扑结构排序

功能：此命令根据元素的拓扑位置从右到左以及从上到下排列元素的执行顺序。

调用：CFC > Execution order（CFC > 执行顺序）菜单、上下文菜单 > 执行顺序

要求：CFC 编辑器处于活动状态，显式执行顺序模式已启用。选择至少 1 个元素。

该命令影响程序中的所有元素，即使在执行该命令时未选择所有元素也不例外。元素的拓扑位置未改变。

另请参见：

- PLC 文档：命令按数据流排序 [▶ 944]
- PLC 文档：命令设置执行顺序 [▶ 943]
- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]
- PLC 文档：按数据流自动执行顺序 [▶ 110]
- PLC 文档：CFC 设置 [▶ 840]

17.12.18 命令：连接所选引脚

符号： 

功能：此命令建立所选引脚之间的链接。

调用：CFC 菜单，上下文菜单

要求：CFC 编辑器处于活动状态。仅选择一个输出和多个输入。

如要选择引脚，点击引脚时长按 **[CTRL]**。然后执行该命令。

另请参见：

- 命令：选择连接引脚 [▶ 945]
- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]

17.12.19 命令：解锁连接

符号： 

功能：此命令解除锁定的连接。

调用：菜单 **CFC > Routing (CFC > 路由选择)**，上下文菜单 > 路由选择

要求：CFC 编辑器处于活动状态。选择一个连接或连接标记。

为自动路由选择更改连接将会导致锁定连接。如要重新执行自动路由选择，您必须首先解除锁定连接。

您还可通过点击锁定连接图标解除该连接。

另请参见：

- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]
- PLC 文档：CFC 元素连接标记源端/目标端 [▶ 620]

17.12.20 命令：显示下一个冲突

功能：此命令指示编辑器中的下一个冲突并标记受影响的位置。

调用：编辑器右上角的  按钮

要求：CFC 编辑器处于活动状态，并且至少与冲突有一个连接。

如果您使用大型网络并且只有一个子集可见，则该功能非常实用。冲突也用编辑器右上角带红框的符号表示。

另请参见：

- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]

17.12.21 命令：选择连接引脚

符号： 

功能：此命令选择连接到当前所选行或面向页面的 CFC 中当前所选连接标记的所有引脚。

调用：上下文菜单

要求：CFC 编辑器或面向页面的 CFC 编辑器处于活动状态。选择 1 个行，并仅选择 1 个连接或 1 个连接标记。

另请参见：

- 命令：连接所选引脚 [▶ 944]
- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]

17.12.22 命令使用属性组件作为输入端

符号： 

功能：此命令允许连接结构组件与标量类型的输入端。

调用：CFC > Pins（CFC > 引脚）菜单、上下文菜单 > 引脚

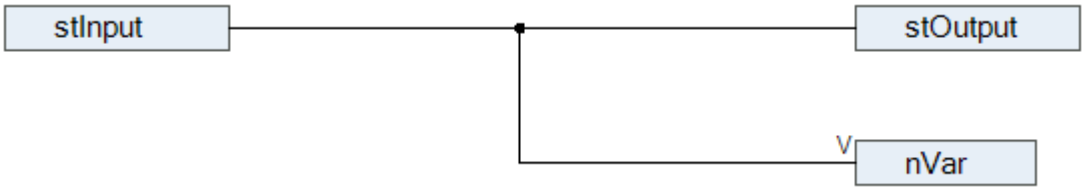
要求：CFC 编辑器处于活动状态，并选择 1 个功能块输入端。

将与后续功能块的输入端连接的结构组件必须具有 {attribute 'ProcessValue'} 属性。结构组件的数据类型必须与后续输入端的数据类型兼容。以这种方式连接的输入端标有 V 符号。

示例

```
TYPE ST_Sample :
STRUCT
    {attribute 'ProcessValue'}
    nVar1 : INT;
    nVar2 : INT;
END_STRUCT
END_TYPE
```

```
PROGRAM MAIN
VAR
    stInput   : ST_Sample;
    stOutput  : ST_Sample;
    nVar      : INT;
END_VAR
```



如果您没有为此链接执行命令 **Use Attributed Component as Input**（使用属性组件作为输入端），则会生成编译器错误。

另请参见：

- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]

17.12.23 命令：重置引脚

符号：

功能：此命令恢复删除的功能块引脚。

调用：菜单 **CFC > Pins (CFC > 引脚)**，上下文菜单 > 引脚

要求：CFC 编辑器处于活动状态，并选择一个功能块。

根据实现中的定义，该命令恢复功能块的所有输入和输出。

另请参见：

- 命令：删除未使用的引脚 [▶ 946]
- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]

17.12.24 命令：删除未使用的引脚

符号：

功能：此命令删除所选元素的所有未使用引脚。

调用：菜单 **CFC > Pins (CFC > 引脚)**，上下文菜单 > 引脚

要求：CFC 编辑器处于活动状态。选择一个元素。

另请参见：

- 命令：重置引脚 [▶ 946]
- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]

17.12.25 命令：添加输入引脚

符号： 

功能：此命令添加其他输入至所选功能块。

调用：菜单 **CFC > Pins (CFC > 引脚)**，上下文菜单 > 引脚

要求：CFC 编辑器处于活动状态。选择一个功能块。

另请参见：

- 命令：添加输出引脚 [▶ 947]
- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]

17.12.26 命令：添加输出引脚

符号： 

功能：此命令添加其他输出至所选功能块。

调用：菜单 **CFC > Pins (CFC > 引脚)**，上下文菜单 > 引脚

要求：CFC 编辑器处于活动状态。选择一个合适的功能块。

另请参见：

- 命令：添加输入引脚 [▶ 947]
- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]

17.12.27 命令：引导所有连接

符号： 

功能：此命令撤消对程序中的连接所作的所有手动更改并恢复原始状态。

调用：菜单 **CFC > Routing (CFC > 路由选择)**，上下文菜单 > 路由选择

要求：CFC 编辑器处于活动状态。

TwinCAT 无法自动引导通过控制点固定的连接。您必须在执行该命令前删除控制点。为此，使用“命令：删除控制点”。此外，您必须断开手动更改的连接（以图标  标记）。为此，使用“命令：解锁连接”。

另请参见：

- 命令：删除控制点 [▶ 948]
- 命令：解锁连接 [▶ 945]
- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]

17.12.28 命令：创建控制点

符号： 

功能：此命令在连接器上创建控制点。

调用：上下文菜单 > 路由选择

要求：CFC 编辑器处于活动状态。将光标移动到连接上。

在调用命令时光标所在的连接点处创建控制点。该命令对应于 **Toolbox（工具箱）**窗口中的 **Control point（控制点）**元素。

另请参见：

- 命令：删除控制点 [▶ 948]
- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]
- PLC 文档：CFC 元素控制点 [▶ 617]

17.12.29 命令：删除控制点

功能：此命令删除控制点。

调用：上下文菜单 > 路由选择

要求：CFC 编辑器处于活动状态。您已选择了一个连接行。

如果将鼠标指针移动到所选连接行上，将以黄色圆圈符号显示现有的控制点。将光标移动到待删除的控制点并执行上下文菜单中的命令。

另请参见：

- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]
- PLC 文档：CFC 元素控制点 [▶ 617]
- 命令：创建控制点 [▶ 948]

17.12.30 命令：连接标记

符号： 

功能：此命令切换连接行与连接标记之间的两个元素间连接的显示。

调用：CFC 菜单，上下文菜单

要求：CFC 编辑器处于活动状态。选择一个连接或连接标记。

如果您选择了连接行，该命令将删除该行并将连接标记源添加到其中一个元素的输出中，并将连接标记目标添加到另一个元素的输入中。默认情况下，两者都分配相同的名称“C-<n>”；n 是一个序号。

如果选择了一个连接标记对，则该命令将这些标记转换为连接行。

另请参见：

- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]
- PLC 文档：CFC 元素连接标记源端/目标端 [▶ 620]

17.12.31 命令：创建组

符号： 

功能：此命令对所选元素进行分组。

调用：菜单 **CFC > Group (CFC > 组)**，上下文菜单 > 组

要求：CFC 编辑器处于活动状态。选择多个元素。

同组元素一起移动。元素位置不受分组影响。

另请参见：

- 命令：取消组 [▶ 949]
- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]

17.12.32 命令：取消组

符号： 

功能：此命令取消之前创建的分组。

调用：菜单 **CFC > Group (CFC > 组)**，上下文菜单 > 组

要求：CFC 编辑器处于活动状态。选择一个分组。

另请参见：

- 命令：创建组 [▶ 949]
- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]

17.12.33 命令：编辑参数

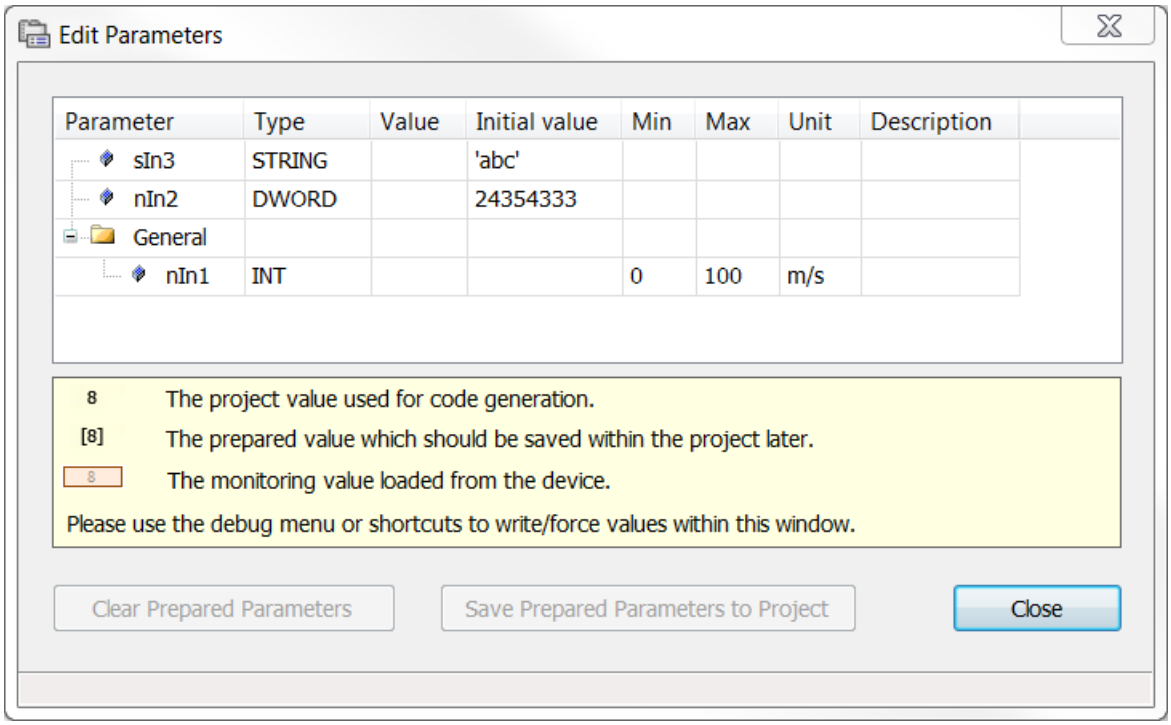
功能：此命令打开“Edit Parameters（编辑参数）”对话框，您可在该对话中更改功能块的常量输入参数。

调用：菜单 **CFC > Edit parameters (CFC > 编辑参数)**，上下文菜单 > 编辑参数，点击功能块字段 **Parameter (参数)**。

要求：CFC 编辑器处于活动状态。功能块实例化，其声明中带有 VAR_INPUT CONSTANT 变量。

TwinCAT 通过功能块左下角的“Parameter（参数）”一词指示带 VAR_INPUT CONSTANT 变量的功能块。

对话框：编辑参数



参数	变量名
类型	变量的数据类型
Value	点击字段以输入值。
初始值	初始值
分类	有关参数的附加信息。这些值根据属性定义并且无法在该对话框中更改 • parameterCategory • parameterUnit • parameterMinValue • parameterMaxValue
单位	
最小	
最大	
删除预备参数	如果您已写入一个预备值，则命令激活（命令：调试 > 写入值）

使用 **OK（确认）** 退出该字段并退出对话框后，值的更改将应用于项目。

带常量输入的功能块示例：

```
FUNCTION_BLOCK FB_Sample
VAR_INPUT CONSTANT
    {attribute 'parameterCategory':='General'}
    {attribute 'parameterUnit':= 'm/s'}
    {attribute 'parameterMinValue':= '0'}
    {attribute 'parameterMaxValue':= '100'}
    nIn1 : INT;
    nIn2 : DWORD:=24354333;
    sIn3 : STRING:='abc';
END_VAR
```

该功能以及带关键词 VAR_INPUT CONSTANT 的变量声明只适用于 CFC 编辑器。在 FBD 编辑器中，TwinCAT 始终显示功能块上的所有输入参数，无论它们是被声明为 VAR_INPUT 还是 VAR_INPUT CONSTANT 均不例外。TwinCAT 也不会区分各文本编辑器。

另请参见：

- 命令：将预备参数保存到项目中 [▶ 951]
- PLC 文档：在线模式下的 CFC 编辑器 [▶ 614]

17.12.34 命令强制 FB 输入端

TC3.1 Build 4026 及以上可用

这种类型的强制在内部使用断点，因此有别于通过命令 **Force values**（强制值）进行的强制：通过命令 **Force FB input**（强制 FB 输入端）强制输入的值不会响应命令 **Watch all forces**（监视所有强制执行）或 **Cancel forcing for all values**（取消对所有值的强制）。

功能：此命令打开用于强制功能块的所选输入端的 **Force value**（强制值）对话框。使用相同的命令和对话框可以取消强制。

调用：CFC 菜单、上下文菜单

要求：CFC 编辑器处于在线模式，并选择功能块的输入端。

在对话框 **Force value**（强制值）中，您可以输入 1 个强制功能块的输入端的值，或者再次删除当前强制的值。

在强制后，输入端将显示为绿色背景。Boolean 输入端也会获得 1 个带有强制值的小型监控窗口。强制值在监控视图的 **Value**（值）列中显示，即程序块的声明部分或监视列表中。

强制值对话框

表达式	功能块输入端的名称
类型	输入数据类型

您想做什么？

设置 1 个新的强制值	您可以在输入字段中输入所需的新值。格式必须与数据类型相对应。
删除值	取消输入端的强制。

另请参见：

- PLC 文档：连续功能图（CFC） [▶ 104]
- PLC 文档：CFC 编辑器 [▶ 607]

17.12.35 命令：将预备参数保存到项目中

功能：此命令在项目中使用预备参数。

调用：CFC 菜单

要求：CFC 编辑器处于活动状态。在在线模式中更改功能块实例的参数值。该应用处于脱机模式。

如果控制器上的常量值与应用中的值不同，则由参数字段右侧的红色星号标志表示。使用命令 **Apply prepared parameter values（应用预备参数值）**将控制值应用于您的应用。

另请参见：

- 命令：编辑参数 [▶ 949]
- PLC 文档：更改功能块实例的常量输入参数 [▶ 614]

17.13 FBD/LD/IL

17.13.1 命令：插入接触点（右）

符号：

功能：此命令将一个接触点插入到所选元素的右侧。

调用：菜单 **FBD/LD/IL**，上下文菜单

要求：LD 编辑器处于活动状态。选择行、接触点或框。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]
- PLC 文档：LD 元素触点 [▶ 605]

17.13.2 命令：插入网络

符号： 

功能：此命令在 FBD/LD/IL 编辑器中插入另一个网络。

调用：菜单 **FBD/LD/IL**，上下文菜单

要求：FBD、LD 或 IL 编辑器处于活动状态。未选择框。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]
- PLC 文档：FBD/LD/IL 元素网络 [▶ 602]

17.13.3 命令：插入网络（下）

符号： 

功能：此命令在所选网络下的 FBD/LD/IL 编辑器中插入另一个网络。

调用：菜单 **FBD/LD/IL**，上下文菜单

要求：FBD、LD 或 IL 编辑器处于活动状态。选择网络，但不选择框。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]
- PLC 文档：FBD/LD/IL 元素网络 [▶ 602]

17.13.4 命令：切换注释状态

符号： 

功能：此命令切换所选网络的注释状态。

调用：菜单 **FBD/LD/IL**，上下文菜单

要求：FBD、LD 或 IL 编辑器处于活动状态。选择网络，但不选择框。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]

17.13.5 命令：插入赋值

符号：

功能：此命令在 FBD 或 LD 编辑器中插入一个赋值。

调用：菜单 **FBD/LD/IL**，上下文菜单

要求：FBD、LD 或 IL 编辑器处于活动状态。选择网络，但不选择框。

在 IL 中，使用操作符 LD 和 ST 对赋值进行编程。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]
- PLC 文档：FBD/LD/IL 元素赋值 [▶ 602]

17.13.6 命令：插入框

符号： 

功能：此命令将项目中可用的框插入到所选网络的末尾。

调用：菜单 **FBD/LD/IL**，上下文菜单

要求：FBD、LD 或 IL 编辑器处于活动状态。选择网络，但不选择框。

当您选择该命令时，输入助手打开。您可在此选择目标框。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]
- PLC 文档：FBD/LD/IL 元素功能块 [▶ 602]

17.13.7 命令：插入带 EN/ENO 的框

符号： 

功能：此命令将带有布尔输入“启用”和布尔输出“启用输出”的框插入到所选网络的末尾。

调用：菜单 **FBD/LD/IL**，上下文菜单

要求：FBD、LD 或 IL 编辑器处于活动状态。选择网络，但不选择框。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]
- PLC 文档：带 EN/ENO 的 FBD/LD/IL 元素功能块 [▶ 603]

17.13.8 命令：插入空框

符号： 

功能：此命令将一个空框插入当前所选网络的末尾。

调用：菜单 **FBD/LD/IL**，上下文菜单

要求：FBD、LD 或 IL 编辑器处于活动状态。选择网络，但不选择框。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]
- PLC 文档：FBD/LD/IL 元素功能块 [▶ 602]

17.13.9 命令：插入带 EN/ENO 的框

符号： 

功能：此命令将带有布尔输入“启用”和布尔输出“启用输出”的空框插入到所选网络的末尾。

调用：菜单 **FBD/LD/IL**，上下文菜单

要求：FBD 编辑器、LD 编辑器或 IL 编辑器处于活动状态。必须选择一个网络。不可选择其他框。

如果在调用框时“启用”的值为 FALSE，则不执行框中定义的操作。如果“启用”为 TRUE，则执行这些操作。ENO 输出可作为 EN 输入的中继器。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]
- PLC 文档：带 EN/ENO 的 FBD/LD/IL 元素功能块 [▶ 603]

17.13.10 命令：插入跳转

符号： 

功能：此命令在当前所选元素前插入一个跳转元素。

调用：菜单 **FBD/LD/IL**，上下文菜单

要求：FBD、LD 或 IL 编辑器处于活动状态。选择连接器。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]
- PLC 文档：FBD/LD/IL 元素跳转 [▶ 603]

17.13.11 命令：插入标签

符号： 

功能：此命令将标签插入到当前所选的网络中。

调用：菜单 **FBD/LD/IL**，上下文菜单

要求：FBD、LD 或 IL 编辑器处于活动状态。选择网络。未选择标签。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]
- PLC 文档：FBD/LD/IL 元素标签 [▶ 603]

17.13.12 命令：插入返回

符号： 

功能：此命令在所选位置插入“返回”元素。

调用：菜单 **FBD/LD/IL**，上下文菜单

要求：FBD、LD 或 IL 编辑器处于活动状态。选择框输出。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]
- PLC 文档：FBD/LD/IL 元素返回 [▶ 603]

17.13.13 命令：插入输入

符号： 

功能：此命令将附加输入添加到所选输入上方的可扩展框 (ADD、OR、ADD、MUL、SEL)。

调用：菜单 **FBD/LD/IL**

要求：FBD/LD 编辑器处于活动状态。选择框输入。

如果选择了一个框，可在上下文菜单中使用命令 **Append name input (附加名称输入)**。将该输入插入到框的底部。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]

17.13.14 命令：插入平行框（下）

符号： 

功能：此命令插入一个与所选框平行的空框。

调用：菜单 **FBD/IL/LD**，上下文菜单

要求：在 LD 编辑器中选择一个框。

另请参见：

- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]

17.13.15 命令：插入线圈

符号： 

功能：此命令将线圈插入到网络中。

调用：菜单 **FBD/LD/IL**，上下文菜单

要求：LD 编辑器处于活动状态。选择网络、线圈或连接器，但不选择框。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]

- PLC 文档：LD 元素线圈 [▶ 605]

17.13.16 命令：插入置位线圈

符号： 

功能：此命令将置位线圈插入网络中。

调用：菜单 **FBD/LD/IL**，上下文菜单

要求：LD 编辑器处于活动状态。选择网络、线圈或行，但不选择框。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]
- PLC 文档：LD 元素线圈 [▶ 605]

17.13.17 命令：插入复位线圈

符号： 

功能：此命令将复位线圈插入网络中。

调用：菜单 **FBD/LD/IL**，上下文菜单

要求：LD 编辑器处于活动状态。选择网络、线圈或行，但不选择框。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]
- PLC 文档：LD 元素线圈 [▶ 605]

17.13.18 命令：插入接触点

符号： 

功能：此命令将一个接触点插入到所选元素的左侧。

调用：菜单 **FBD/LD/IL**，上下文菜单

要求：LD 编辑器处于活动状态。选择行或接触点。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]
- PLC 文档：LD 元素触点 [▶ 605]

17.13.19 命令：插入平行接触点（下）

符号： 

功能：此命令将带有行的平行接触点插入到所选元素下方。

调用：菜单 **FBD/LD/IL**，上下文菜单

要求： LD 编辑器处于活动状态。选择行、接触点或框。

您可以在 LD 网络中将关闭的平行分支编程为短路评估 (SCE) 或 OR 结构。SCE 分支由双垂直行、带单行的 OR 分支表示。参见“关闭的行分支 [▶ 606]”的帮助页面。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]
- PLC 文档：LD 元素触点 [▶ 605]

17.13.20 命令：插入平行接触点（上）

符号：

功能： 此命令将带有行的平行接触点插入到所选元素上方。

调用： 菜单 FBD/LD/IL，上下文菜单

要求： LD 编辑器处于活动状态。选择行、接触点或框。

您可以在 LD 网络中将关闭的平行分支编程为短路评估 (SCE) 或 OR 结构。SCE 分支由双垂直行、带单行的 OR 分支表示。参见“关闭的行分支 [▶ 606]”的帮助页面。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]
- PLC 文档：LD 元素触点 [▶ 605]

17.13.21 命令：插入否定的接触点

符号：

功能： 此命令将一个否定的接触点插入到所选元素左侧。

调用： 菜单 FBD/LD/IL，上下文菜单

要求： LD 编辑器处于活动状态。选择行或接触点。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]
- PLC 文档：LD 元素触点 [▶ 605]

17.13.22 命令：插入否定的平行接触点（下）

符号：

功能： 此命令将带有行的否定平行接触点插入到所选元素下方。

调用： 菜单 FBD/LD/IL，上下文菜单

要求： LD 编辑器处于活动状态。选择行、接触点或框。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]
- PLC 文档：LD 元素触点 [▶ 605]

17.13.23 命令：粘贴接触点：粘贴在下方

功能：此命令将带有行的之前复制的接触点粘贴到所选元素下方。

调用：菜单 FBD/LD/IL，上下文菜单

要求：LD 编辑器处于活动状态。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]
- PLC 文档：LD 元素触点 [▶ 605]

17.13.24 命令：粘贴接触点：粘贴在上方

功能：此命令将带有行的之前复制的接触点粘贴到所选元素上方。

调用：菜单 FBD/LD/IL，上下文菜单

要求：LD 编辑器处于活动状态。选择行或接触点。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]
- PLC 文档：LD 元素触点 [▶ 605]

17.13.25 命令：粘贴接触点：粘贴在右侧（之后）

功能：此命令将之前复制的接触点粘贴到所选元素右侧。

调用：菜单 FBD/LD/IL，上下文菜单

要求：LD 编辑器处于活动状态。选择行或接触点。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]
- PLC 文档：LD 元素触点 [▶ 605]

17.13.26 命令：在下方插入 IL 行

符号： 

功能：此命令将一个指令行插入到所选行的下方。

调用：菜单 FBD/LD/IL，上下文菜单

要求：IL 编辑器处于活动状态。选择行。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]

17.13.27 命令：删除 IL 行

符号： 

功能：此命令删除所选的指令行。

调用：菜单 **FBD/LD/IL**，上下文菜单

要求：IL 编辑器处于活动状态。选择行。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]

17.13.28 命令：否定

符号： 

功能：此命令否定以下元素：

- 框的输入/输出
- 跳转
- 返回
- 线圈

调用：菜单 **FBD/LD/IL**，上下文菜单

要求：FBD 或 LD 编辑器处于活动状态。选择对应的元素。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]

17.13.29 命令边缘检测

符号 FBD： 

符号 LD： 

功能：此命令将 1 个边缘检测插入到所选方框输入端或方框输出端之前。通过执行 1 次、2 次或 3 次命令可以区分以下功能：

- 在执行 1 次命令时插入的边缘检测可用于检测上升沿。符号的箭头指向右侧。
- 如果再次执行命令，则边缘检测将反转，从而检测到下降沿。符号的箭头指向左侧。
- 如果再一次执行命令，则边缘检测和符号将被删除。

调用：菜单 **FBD/LD/IL**，上下文菜单

要求：FBD 或 LD 编辑器处于活动状态。选择方框输入端或输出端。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]

17.13.30 命令：设置/重置

符号： 

功能：对于带布尔输出的元素，此命令在“重置”、“设置”和“无”标签之间切换。

调用：菜单 **FBD/LD/IL**，上下文菜单

要求：FBD 或 LD 编辑器处于活动状态。选择带布尔输出的元素。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]

17.13.31 命令：设置输出连接

符号： 

功能：此命令将所选框输出分配为互联框输出。

调用：菜单 **FBD/LD/IL**，上下文菜单

要求：FBD 或 LD 编辑器处于活动状态。选择若干框输出之一。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]

17.13.32 命令：插入分支

符号： 

功能：此命令在所选行上创建一个打开的分支。

调用：菜单 **FBD/LD/IL**，上下文菜单

要求：FBD 或 LD 编辑器处于活动状态。选择框输入或输出。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]
- PLC 文档：FBD/LD/IL 元素线路分支 [▶ 604]

17.13.33 命令：在上方插入分支

符号： 

功能：此命令在所选的打开分支上方创建一个分支。

调用：菜单 **FBD/LD/IL**，上下文菜单

要求：FBD 或 LD 编辑器处于活动状态。选择一个打开的分支。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]
- PLC 文档：FBD/LD/IL 元素线路分支 [▶ 604]

17.13.34 命令：在下方插入分支

符号： 

功能：此命令在所选的打开分支下方创建一个分支。

调用：菜单 **FBD/LD/IL**，上下文菜单

要求：FBD 或 LD 编辑器处于活动状态。选择一个打开的分支。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]
- PLC 文档：FBD/LD/IL 元素线路分支 [▶ 604]

17.13.35 命令：设置分支开始点

符号： 

功能：此命令在所选行上设置分支的开始点。

调用：菜单 **FBD/LD/IL**，上下文菜单

要求：LD 编辑器处于活动状态。选择行。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]
- PLC 文档：关闭的行分支 [▶ 606]

17.13.36 命令：设置分支结束点

符号： 

功能：此命令在所选行上设置分支的结束点。

调用：菜单 **FBD/LD/IL**，上下文菜单

要求：LD 编辑器处于活动状态。选择行。分支的开始点已设置。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]
- PLC 文档：关闭的行分支 [▶ 606]

17.13.37 命令切换平行模式

功能：此命令在 OR 结构和短路求值（SCE）之间切换平行分支。

调用：FBD/LD/IL 菜单、上下文菜单

要求：FBD/LD/IL 编辑器处于活动状态。选择 1 条平行分支的垂直线。

您可以在 LD 网络中将关闭的平行分支编程为短路求值（SCE）或 OR 结构。SCE 拓扑扩展模块用双垂直线表示，OR 拓扑扩展模块用单行表示。请参见“封闭分支 [▶ 606]”的帮助页面。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]
- PLC 文档：封闭分支 [▶ 606]

17.13.38 命令：更新参数

功能：此命令将所选元素声明中的更改应用于图形。

调用：菜单 **FBD/LD/IL**，上下文菜单

要求：FBD、LD 或 CFC 编辑器处于活动状态。选择框。对声明进行了扩展更改。

该命令在声明编辑器中检查框及其声明是否匹配。仅当声明被扩展时才能将更改应用于框。删除和覆盖不更新。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]

17.13.39 命令：删除未使用的 FB 调用参数

符号： 

功能：此命令删除所选功能块的输入和输出，该功能块未被赋予变量和值。但始终保留该功能块的默认输入和输出。

调用：菜单 **FBD/LD/IL**，上下文菜单

要求：FBD 或 LD 编辑器处于活动状态。选择一个功能块。该功能块具有未赋值的界面。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]

17.13.40 命令：修复 POU

符号： 

功能：此命令修复所选功能块的内部不一致。

调用：菜单 **FBD/LD/IL**，上下文菜单

要求：FBD 或 LD 编辑器处于活动状态。选择一个故障功能块。编辑器检测到编程块中的内部不一致，该问题可以自动解决。TwinCAT 在 **Error list（错误列表）** 视图中报告不一致现象。

当您编辑使用较旧版本编程系统（该系统未将这种不一致视为错误）创建的项目时，可能会发生这种情况。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]

17.13.41 命令：以功能块示意图的形式查看

注意

数据丢失

零错误转换需要语法正确的代码。否则，部分实现可能会丢失。

功能：此命令将活动的指令列表或活动的梯形图转换为功能块示意图。

调用：菜单 **FBD/LD/IL > View (FBD/LD/IL > 视图)**

要求：LD 或 IL 编辑器处于活动状态。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]

17.13.42 命令：以梯形图的形式查看

注意

数据丢失

零错误转换需要语法正确的代码。否则，部分实现可能会丢失。

功能：此命令将当前功能块代码或活动的指令列表转换为梯形图。

调用：菜单 **FBD/LD/IL > View (FBD/LD/IL > 视图)**

要求：FBD 或 IL 编辑器处于活动状态。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]

17.13.43 命令：以指令列表的形式查看

若需要，可通过 TwinCAT 选项启用 IL。

注意

数据丢失

零错误转换需要语法正确的代码。否则，部分实现可能会丢失。

功能：此命令将活动的功能块代码或活动的梯形图转换为指令列表。

调用：菜单 **FBD/LD/IL > View (FBD/LD/IL > 视图)**

要求：FBD 或 LD 编辑器处于活动状态。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]

17.13.44 命令：转至

符号：

功能：可通过此命令跳转到任何网络。

调用：菜单 **FBD/LD/IL**

要求：LD、FBD 编辑器或 IL 编辑器处于活动状态。选择网络。

该命令打开带输入字段的对话框。在输入字段中输入所需网络的编号。

另请参见：

- PLC 文档：FBD/LD/IL [▶ 99]
- PLC 文档：FBD/LD/IL 编辑器 [▶ 594]

17.14 梯形图编辑器

在梯形图编辑器中工作的命令可在**梯形图**菜单和相应的上下文菜单中找到。

17.14.1 命令注释掉

符号：

功能：此命令可在注释掉和未注释掉状态之间切换网络 [▶ 966]。

调用：Ladder（梯形图）菜单；上下文菜单

网络内容以灰色显示，文本以斜体显示。在处理过程中，不会考虑网络。

17.14.2 命令否定

符号：

功能：此命令否定以下元素：

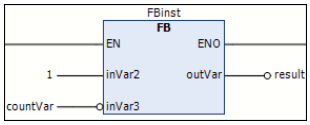
- 方框的输入端/输出端
- 跳转
- 返回
- 线圈
- 触点
- 变量

调用：Ladder（梯形图）菜单；上下文菜单

要求：选择要否定的元素。

示例：

1 个功能块上的否定输入端和否定输出端：



否定触点：

否定线圈：

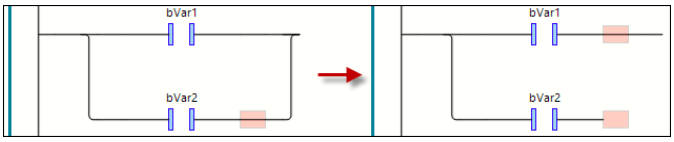
17.14.3 命令打开平行分支

符号：

功能：此命令可打开 1 个封闭的平行分支。

调用：Ladder（梯形图）菜单；上下文菜单

要求：必须从要重新打开的分支的 2 条线路中选择 1 条。



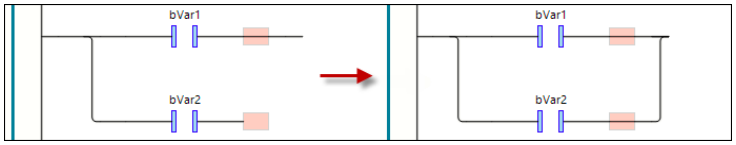
17.14.4 命令关闭平行分支

符号：

功能：此命令可将打开的平行分支关闭。

调用：Ladder（梯形图）菜单；上下文菜单

要求：必须选择要关闭的分支的 2 条线路：



或者，也可以通过将 1 个分支的选择标记拖到另一个分支的选择标记处来关闭 1 个打开的分支。

17.14.5 命令设置/重置 - 设置，设置/重置 - 重置

符号：设置 、重置

功能：设置命令为编辑器中所选的元素提供 1 个设置修饰符。重置命令为元素提供 1 个重置修饰符。

例如，您可以用它将 1 个线圈 [▶ 967]和 1 个非主方框制成 1 个“设置线圈”。此命令仅在适当的位置可用。

调用：Ladder（梯形图）菜单、Set/Reset（设置/重置）子菜单；上下文菜单

17.14.6 命令边缘检测 - 上升沿

符号：

功能：此命令将 1 个上升沿检测插入到所选方框输入端或触点之前。

调用：Ladder（梯形图）菜单；上下文菜单

要求：选择 1 个方框输入端或触点。

17.14.7 命令边缘检测 - 下降沿

符号：

功能：此命令将 1 个下降沿检测插入到所选方框输入端或触点之前。

调用：Ladder（梯形图）菜单；上下文菜单

要求：选择 1 个方框输入端或触点。

17.14.8 命令 EN/ENO：EN

符号：

功能：此命令用于添加或删除所选方框中的 Boolean 输入端“启用”。

调用：Ladder（梯形图）菜单、**EN/ENO** 子菜单；上下文菜单

Boolean EN 输入端控制方框的执行。如果在方框调用期间 EN 值为 FALSE，则不会执行在方框中定义的操作。如果 EN 为 TRUE，则执行这些操作。

如果 1 个方框有 EN 输入端，也可以为其添加 ENO 输出端 [▶ 966]：ENO 的值与 EN 的值相同。

17.14.9 命令 EN/ENO：ENO

符号：

功能：此命令为带有 EN 输入端的方框添加 ENO 输出端。

调用：Ladder（梯形图）菜单、**EN/ENO** 子菜单；上下文菜单

要求：选择 1 个带有 EN 输入端的方框。

ENO 输出端的值与 EN 输入端的值相同。

17.14.10 命令插入网络

符号：

功能：此命令在梯形图编辑器中插入 1 个附加网络。当您将元素拖到实现部分上时，您可以通过光标 上的加号识别可能的插入位置。

调用：Ladder（梯形图）菜单；上下文菜单

要求：未选择方框。

17.14.10.1 元素：网络

网络是 LD 程序的基本单元。在梯形图编辑器中，网络以列表形式排列在彼此下面。每个网络左侧都有 1 个递增的网络编号，可能包括以下内容：逻辑和算术表达式、程序/函数/功能块调用、跳转或返回语句。

您可以为每个网络提供标题、注释或跳转标签。

网络标题：双击网络顶部的第 1 行，指定它。

网络注释：双击网络顶部的第 2 行，指定它。

跳转标签：双击网络顶部的第 3 行，指定它。然后，在跳转 [▶ 968]元素中可以将跳转标签指定为目标。

在 TwinCAT 选项的 梯形图 [▶ 912]类别中，您可以定义是否在编辑器中显示网络标题、网络注释以及各个网络之间的分隔线。

17.14.11 命令插入触点

符号： ，在编辑器中为

功能：此命令将 1 个触点插入到网络中所选元素的左侧。当您将元素从工具箱拖到实现部分时，光标 上的加号会帮助您识别可能的插入位置。

调用：Ladder（梯形图）菜单；上下文菜单

要求：选择行或触点。

17.14.11.1 元素：触点

触点从左到右传递 TRUE（ON）或 FALSE（OFF）信号，直到信号到达网络右侧的线圈。为此，可为触点分配 1 个包含信号的 Boolean 变量。为此，可将触点上方的 ??? 占位符替换为 Boolean 变量的名称。

您可以将多个触点串联或并联起来。如果使用 2 个并联触点，则仅限在其中 1 个触点的值为 TRUE 时，才能向右传递 ON。如果将触点串联起来，则必须将所有触点设置为 TRUE，才能从串联的最后一个触点向右传递 ON。因此，您可以使用 LD 对并联和串联电路进行编程。

当变量值为 FALSE 时，否定触点 会传递 TRUE 信号。您可以使用 **Ladder → Negate**（梯形图 → 否定）命令启用或禁用已插入触点的否定功能。

17.14.12 命令插入线圈

符号：

编辑器中的符号：

功能：此命令将线圈插入到网络中。当您将元素从工具箱拖到实现部分时，光标 上的加号会帮助您识别可能的插入位置。

调用：Ladder（梯形图）菜单；上下文菜单

要求：选择网络、线圈或连接器。未选择方框。

17.14.12.1 元素：线圈

线圈可获取从左侧传递过来的值，并将其存储到分配给它的 Boolean 变量中。其输入端可以有值 TRUE（ON）或 FALSE（OFF）。

在否定 [▶ 964]线圈 中，输入信号的否定值存储在分配给线圈的 Boolean 变量中。

通过设置/重置 [▶ 965]子菜单的命令，您可以定义“设置线圈”或“重置线圈”：

设置线圈：如果 TRUE 值到达设置线圈，则该线圈会保留 TRUE 值。只要应用程序处于运行状态，在此处就不再覆盖该值。

重置线圈：如果 TRUE 值到达重置线圈，则该线圈会保留 FALSE 值。只要应用程序处于运行状态，在此处就不再覆盖该值。

17.14.13 命令插入方框

符号：

功能：此命令可插入在所选网络的末尾可用的方框。当您将元素从工具箱拖到实现部分时，光标 上的加号会帮助您识别可能的插入位置。

调用：Ladder（梯形图）菜单；上下文菜单

该命令默认插入 1 个 EN/ENO 方框。**Ladder**（梯形图）EN/ENO 菜单命令可用于删除和添加 EN [▶ 966] 和 ENO [▶ 966] 修饰符。

当前未实现：如果是在项目中定义的方框，则在更改该方框后必须更新梯形图中的用法。

17.14.13.1 元素：方框

方框元素可代表其他函数：例如，IEC 功能块、IEC 函数、库功能块或运算符。

如果方框还提供图像文件，则可在方框内显示方框图标。前提条件：在 TwinCAT 选项的 **Ladder**（梯形图）类别中已启用 **Show box icon**（显示方框图标）选项。

17.14.14 命令插入跳转

符号：

功能：此命令在所选的元素前插入 1 个跳转元素。当您将元素从工具箱拖到实现部分时，光标 上的加号会帮助您识别可能的插入位置。

调用：Ladder（梯形图）菜单；上下文菜单

要求：选择 1 个连接器。

17.14.14.1 元素：跳转

在 LD 中，根据当前光标位置，您可以直接在输入端前、输出端后或网络末端插入跳转。

您可以直接在跳转元素后输入网络的跳转标签作为跳转目标。

17.14.15 命令插入返回

符号：

功能：此命令在所选位置插入 Return（返回）元素。当您将元素从工具箱拖到实现部分时，光标 上的加号会帮助您识别可能的插入位置。

调用：Ladder（梯形图）菜单；上下文菜单

17.14.15.1 元素：返回

当 Return（返回）元素的输入端变为 TRUE 时，该元素可立即中断梯形图方框的执行。您可以将 Return（返回）语句置于与前面元素平行或在其之后的位置。

17.14.16 命令插入输入端

符号：

功能：此命令在所选位置插入输入端。当您将元素从工具箱拖到实现部分时，光标 上的加号会帮助您识别可能的插入位置。您可以用变量名或常量替换最初显示的问号 *???* 。

调用：Ladder（梯形图）菜单；上下文菜单

17.14.16.1 元素：输入

该元素用于将变量值分配给信号流中的另一个元素。它有 1 条向右延伸的线。

17.14.17 命令插入输出端

符号：

功能：此命令在所选位置插入输出端。当您将元素从工具箱拖到实现部分时，光标 上的加号会帮助您识别可能的插入位置。您可以用变量名或常量替换最初显示的问号 **???** 。

调用：Ladder（梯形图）菜单；上下文菜单

17.14.17.1 元素：输出

该元素用于将来自左侧的信号分配给与该元素相关的变量。它有 1 条来自左侧的输入行。

17.14.18 命令转换为新梯形图

在 FBD/LD 编辑器中可用。

17.15 声明

17.15.1 命令粘贴

符号：

功能：此命令在表格声明编辑器中为变量声明插入新行，并打开变量名的输入字段。

调用：表格声明编辑器的声明标题中的按钮、表格声明编辑器中的上下文菜单

如要编辑声明行的其他字段，可双击这些字段，然后从选择列表或相应对话框中选择信息。

另请参见：

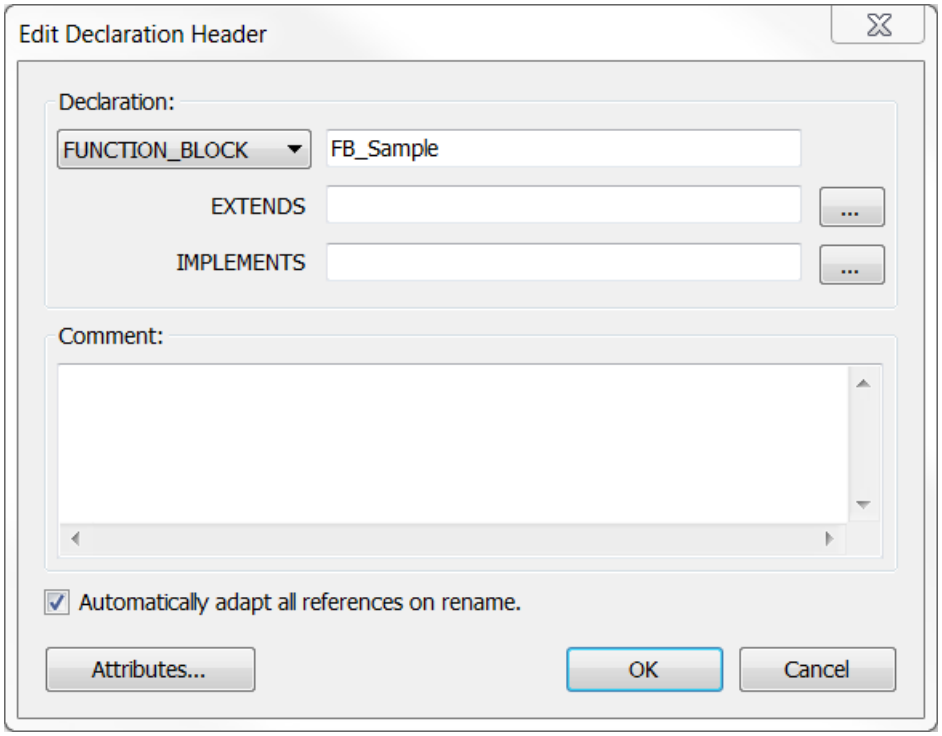
- PLC 文档：使用声明编辑器 [▶ 64]

17.15.2 命令编辑声明标题

功能：此命令可打开 **Edit the declaration header**（标记声明标题）对话框，在表格声明编辑器中使用该对话框配置 POU 的标题。

调用：鼠标点击表格声明编辑器的标题栏、表格声明编辑器中的上下文菜单

对话框编辑声明标题



声明	更改 POU 类型的选择列表 • PROGRAM • FUNCTION_BLOCK ◦ EXTENDS：基本功能块的输入字段 ◦ IMPLEMENTS：接口的输入字段 • FUNCTION ◦ 返回类型 带有当前 POU 名称的输入字段：您可以更改 POU 的名称。
在重命名时自动调整所有引用	 ：**Refactoring**（重构）对话框会打开。  ：重命名仅在 POU 的声明标题中生效。
属性	**Attributes**（属性）对话框将打开，用于输入属性和编译指示。

另请参见：

- PLC 文档：使用声明编辑器 [▶ 64]
- PLC 文档：编译指示 [▶ 733]
- PLC 文档：重构 [▶ 148]

17.15.3 命令向下移动

符号： 

功能：此命令将变量声明向下移动 1 行。

调用：表格声明编辑器的声明标题中的按钮、表格声明编辑器中的上下文菜单

要求：在表格声明编辑器中选择带有变量声明的行。

另请参见：

- PLC 文档：使用声明编辑器 [▶ 64]

17.15.4 命令向上移动

符号： 

功能：此命令将变量声明向上移动 1 行。

调用：表格声明编辑器的声明标题中的按钮、表格声明编辑器中的上下文菜单

要求：在表格声明编辑器中选择带有变量声明的行。

另请参见：

- PLC 文档：使用声明编辑器 [▶ 64]

17.16 文本列表

17.16.1 命令：添加语言

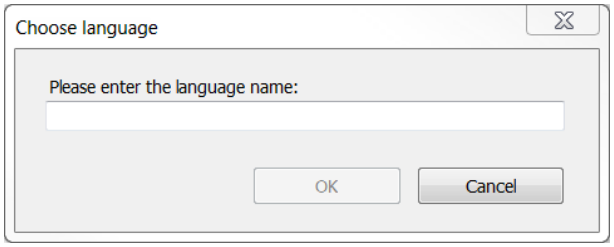
符号： 

功能：此命令在文本列表中添加一个附加的语言列。

调用：菜单 **Textlist（文本列表）**，上下文菜单

要求：文本列表或全局文本列表打开并处于活动状态。

在 **Choose language（选择语言）**对话框中输入新语言的缩写，例如“en-US”。TwinCAT 将缩写作为列标题插入。



另请参见：

- PLC 文档：管理文本列表中的文本 [▶ 127]

17.16.2 命令：删除语言

符号： 

功能：此命令删除文本列表中的所选语言列。

调用：菜单 **Textlist（文本列表）**，上下文菜单

要求：文本列表或全局文本列表打开并处于活动状态。选择您要删除的语言列中的字段。

另请参见：

- PLC 文档：管理文本列表中的文本 [▶ 127]

17.16.3 命令：插入文本

符号： 

功能：此命令在文本列表中的所选行上方插入一个新行。在 Standard（标准）下打开一个输入字段，您可在该字段中输入源文本。

调用：菜单 **Textlist（文本列表）**

要求：文本列表（不是 GlobalTextList）打开并处于活动状态。在表中选择一个字段。

另请参见：

- PLC 文档：管理文本列表中的文本 [▶ 127]

17.16.4 命令：导入/导出文本列表

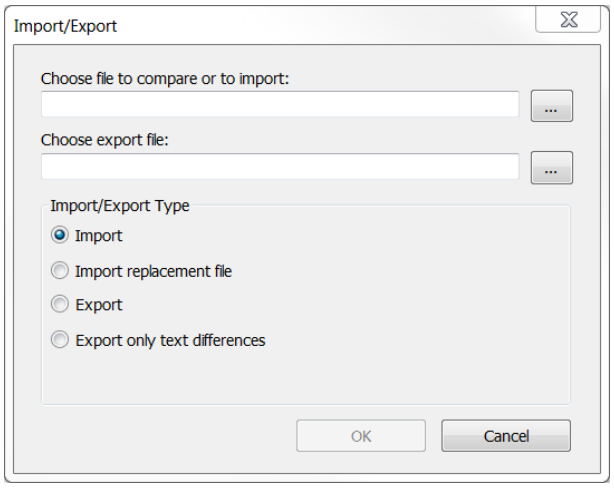
符号： 

功能：此命令导出一个活动文本列表、导入一个文件或将文本列表与文件同步。该文件是 CSV 格式。**Import/Export（导入/导出）**对话框提供了用于此目的的选项。

调用：菜单 **Textlist（文本列表）**，上下文菜单

要求：文本列表或全局文本列表打开并处于活动状态。

导入/导出对话框



选择要比较或导入的文件	TwinCAT 读取的文件。 ... 打开 **Choose textlist file**（选择文本列表文件）对话框，您可在其中选择 1 个文件。
选择导出文件	TwinCAT 写入的文件。 ... 打开 **Choose textlist file** （选择文本列表文件）对话框，您可在其中选择 1 个文件和 1 个目录。

导入/导出类型：

导入	要求：在 **Choose file to compare or to import**（选择要比较或导入的文件）中选择 1 个文件。 该文件可能包含全局文本列表或文本列表的文本列表记录。 全局文本列表： • TwinCAT 读取文件、比较相同源文本的文本列表记录并将差异应用于翻译中。必要时，TwinCAT 将覆盖项目中的翻译。 文本列表： • TwinCAT 读取文件、比较相同 ID 的文本列表记录并将差异应用于项目的源文本和翻译中。必要时，TwinCAT 将覆盖项目中的文本列表记录。 • 如果文件包含 1 个新 ID，文本列表条目将被导入到项目的文本列表中，并且该文本列表将被修改。
导入替换文件	要求：在 **Choose file to compare or to import**（选择要比较或导入的文件）中选择 1 个替换文件。 替换文件包含全局文本列表的替换。 TwinCAT 逐行处理替换文件并在全局文本列表中实现指定的替换。有关替换文件的结构，请参见 管理全局文本列表中的静态文本 [▶ 130] 部分。
导出	要求：在“Choose export file”（选择导出文件）中选择 TwinCAT 写入的文件。 TwinCAT 导出当前项目的所有文本列表中的所有文本。项目中可用的所有语言均作为列插入到导出文件中。该文件可用于从外部翻译与语言相关的文本。
仅导出文本差异	要求：在 **Choose file to compare or to import**（选择要比较或导入的文件）中选择 1 个用于比较的导入文件。在 **Choose export file**（选择导出文件）中选择 TwinCAT 写入的导出文件。 TwinCAT 读取导入文件并将活动文本列表的行与之进行比较。TwinCAT 忽视任何匹配的行。如果行不同，TwinCAT 将该行写入导出文件并应用文本列表中的翻译。TwinCAT 应用导入文件中的翻译并覆盖（如适用）。

另请参见：

- PLC 文档：管理文本列表中的文本 [▶ 127]

17.16.5 命令：删除未使用的文本列表记录

符号： 

功能：此命令检查项目中的文本列表记录是否用作静态文本。如果不是，TwinCAT 将从文本列表中删除该文本。

调用：菜单 **Textlist（文本列表）**，上下文菜单

要求：全局文本列表打开并处于活动状态。在表中选择一个字段。

另请参见：

- PLC 文档：管理文本列表中的文本 [▶ 127]

17.16.6 命令：检查图形文本 ID

符号： 

功能：此命令检查项目中的文本列表记录 ID 是否正确并报告结果。

调用：菜单 **Textlist（文本列表）**，上下文菜单

要求：全局文本列表打开并处于活动状态。在表中选择一个字段。

如果 TwinCAT 检测到可视化的全局文本列表与静态文本不匹配，可能是因为全局文本列表为只读状态。需要在项目中设置用户管理。

另请参见：

- PLC 文档：管理文本列表中的文本 [▶ 127]

17.16.7 命令：更新可视化文本 ID

符号： 

功能：此命令更新静态文本列表中的所有不一致 ID。

调用：菜单 **Textlist（文本列表）**，上下文菜单

要求：全局文本列表打开并处于活动状态。在表中选择一个字段。该对象为只读状态。

如果 TwinCAT 检测到可视化的全局文本列表与静态文本不匹配，可能是因为全局文本列表为只读状态。需要在项目中设置用户管理。

另请参见：

- PLC 文档：管理文本列表中的文本 [▶ 127]

17.16.8 命令导出全部

符号： 

功能：此命令导出项目的所有文本列表。

调用：菜单 **Textlist**（文本列表）、上下文菜单

要求：

- 文本列表或全局文本列表打开并处于活动状态。
- 可视化未通过 Unicode 对文本字符进行编码。

TwinCAT 为每个文本列表创建 1 个 .txt 格式的普通文本。文本列表名称成为文件的名称。该文件保存在 TwinCAT 项目的目录中。

控制器可读取并使用此格式。例如，您可以将文件复制到控制器，并通过可视化管理器中的设置配置该文件，以便在加载 PLC 项目时不会再次传输文本列表。

另请参见：

- PLC 文档：管理文本列表中的文本 [▶ 127]

17.16.9 命令导出所有 Unicode

符号： 

功能：此命令导出项目的所有文本列表。

调用：菜单 **Textlist**（文本列表）、上下文菜单

要求：

- 文本列表或全局文本列表打开并处于活动状态。
- 可视化通过 Unicode 对文本字符进行编码。
  - 在可视化管理器中启用选项 **Use Unicode strings** （使用 Unicode 字符串）。

TwinCAT 为每个文本列表创建 1 个 .txt 格式的普通文本。文本列表名称成为文件的名称。该文件保存在 TwinCAT 项目的目录中。

控制器可读取并使用此格式。例如，您可以将文件复制到控制器，并通过可视化管理器中的设置配置该文件，以便在加载 PLC 项目时不会再次传输文本列表。

另请参见：

- PLC 文档：管理文本列表中的文本 [▶ 127]

17.16.10 命令：添加文本列表支持

符号： 

功能：此命令将文本列表支持添加到 **Enumeration（枚举）** 类型的所选 DUT 对象中。

调用：Enumeration（枚举） 类型标准 DUT 对象的上下文菜单 （  ）

文本列表支持实现了枚举组件标识符的本地化以及可视化文本输出中符号组件值的表示。

另请参见：

- PLC 文档：对象 DUT [▶ 68]
- 命令：删除文本列表支持 [▶ 974]

17.16.11 命令：删除文本列表支持

符号： 

功能：此命令删除所选枚举对象中的文本列表支持。

调用：带文本列表支持的枚举对象的上下文菜单 （  ）。

文本列表支持实现了枚举组件标识符的本地化以及可视化文本输出中符号组件值的表示。

17.17 配方

17.17.1 命令：添加新配方

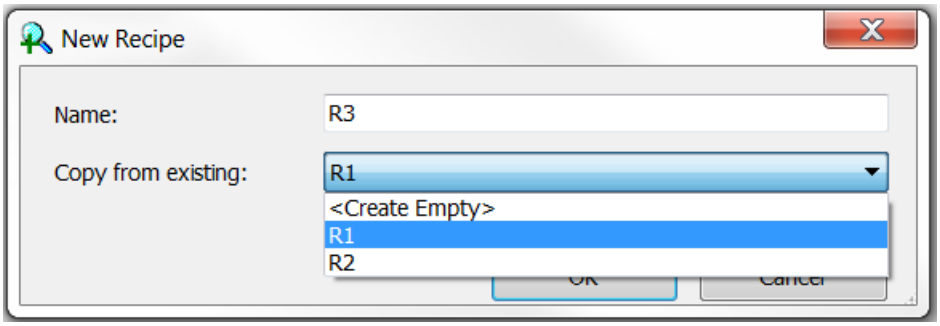
符号： 

功能：此命令打开一个用于在配方定义中插入新配方（新列）的对话框。

调用：菜单 **Recipes（配方）**，上下文菜单

要求：配方定义在编辑器中打开。

当您运行该命令时，将打开一个对话框，您可以在其中设置新配方的名称。对话框还提供用于将现有配方复制到新配方的选项。



另请参见：

- PLC 文档：更改带配方的值 [▶ 210]

17.17.2 命令：删除配方

符号： 

功能：此命令从当前打开的配方定义中删除配方。

调用：菜单 **Recipes（配方）**，上下文菜单

要求：在配方定义的配方列中选择一个字段。

另请参见：

- PLC 文档：更改带配方的值 [▶ 210]

17.17.3 命令：加载配方

符号： 

功能：此命令加载文件中的配方。

调用：Recipes（配方）菜单；上下文菜单

要求：在配方定义的配方列中选择 1 个字段。

当您运行该命令时，将打开用于选择文件的标准对话框。筛选器自动设置为文件扩展名 *.txtrecipe。加载后，将覆盖配方定义中所选配方的值并更新显示。

如果您只想用新值覆盖单个配方变量，请在将配方加载到配方文件中之前删除其他变量的值。不读取没有值规范的条目，这意味着这些变量不受控制器上和项目中更新的影响。以下是配方文件中的条目示例。加载时，仅写入带新值（6）的 MAIN.nVar：

```
MAIN.nVar1:=
MAIN.nVar2:=6
MAIN.nVar3:=
MAIN.sVar4:=
MAIN.wsVar5:=
```

对于 REAL/LREAL 数据类型的值，在某些情况下也会将十六进制值写入配方文件。这样做很有必要，以便在反向转换过程中恢复完全相同的值。在这种情况下，更改十进制值并删除十六进制值。

另请参见：

- PLC 文档：更改带配方的值 [▶ 210]

17.17.4 命令：保存配方

符号： 

功能：此命令将配方变量的值保存在文件中。

调用：菜单 **Recipes（配方）**，上下文菜单

要求：在配方定义中选择配方值。

执行该命令时，TwinCAT 将所选配方的值保存在扩展名为 *.txtrecipe 的文件中，必须定义该文件的名称。用于保存文件的标准对话框打开。格式通过 **Storage（存储）** 选项卡中的配方管理器进行设置。

覆盖隐式配方文件

不得覆盖作为读取和写入配方的剪贴板的隐式配方文件。这意味着该文件名称必须与 <Recipe name>.<Recipe definition name>.txtrecipe 不同。

另请参见：

- PLC 文档：更改带配方的值 [▶ 210]

17.17.5 命令：读取配方

符号： 

功能：此命令读取来自控制器的配方变量值。

调用：菜单 **Recipes（配方）**，上下文菜单

要求：PLC 项目处于在线模式，并在配方定义中选择配方值。

执行该命令时，TwinCAT 用从控制器读取的值覆盖所选配方的值。在此过程中，值被隐式存储在控制器的文件中，并同时显示在配方定义表中。

另请参见：

- PLC 文档：更改带配方的值 [▶ 210]

17.17.6 命令：写入配方

符号： 

功能：此命令将配方值写入控制器中的变量。

调用：菜单 **Recipes（配方）**，上下文菜单

要求：PLC 项目处于在线模式，并在配方定义中选择配方值。

执行该命令时，TwinCAT 用所选配方的值覆盖控制器中的值。

另请参见：

- PLC 文档：更改带配方的值 [▶ 210]

17.17.7 命令：加载和写入配方

符号： 

功能：此命令加载文件中的配方并将值写入控制器中的变量。

调用：Recipes（配方）菜单；上下文菜单

要求：PLC 项目处于在线模式，并在配方定义中选择配方值。

在运行命令之后，系统将询问您文件中的值应该写入项目中的配方还是只写入 PLC。需要在下一次登录时进行在线更改才能更新配方中的值。

当您执行命令时，TwinCAT 将覆盖配方定义中所选配方的值。此外，控制器中的变量值被这些配方值覆盖。

如果您只想用新值覆盖单个配方变量，请在将配方加载到配方文件中之前删除其他变量的值。不读取没有值规范的条目，这意味着这些变量不受控制器上和项目中更新的影响。以下是配方文件中的条目示例。加载时，仅写入带新值（6）的 MAIN.nVar1。

```
MAIN.nVar1:=
MAIN.nVar2:=6
MAIN.nVar3:=
MAIN.sVar4:=
MAIN.wstVar5:=
```

对于 REAL/LREAL 数据类型的值，在某些情况下也会将十六进制值写入配方文件。这样做很有必要，以便在反向转换过程中恢复完全相同的值。在这种情况下，更改十进制值并删除十六进制值。

另请参见：

- PLC 文档：更改带配方的值 [▶ 210]

17.17.8 命令：读取和保存配方

符号： 

功能：此命令读取控制器中的配方变量值并将该值保存在文件中。

调用：菜单 **Recipes（配方）**，上下文菜单

要求：PLC 项目处于在线模式，并在配方定义中选择配方值。

执行命令后，将询问您是否将变量值读入项目中的配方，还是只保存它们。需要在下一次登录时进行在线更改才能更新配方中的值。

根据配方管理器的设置，这些值以配方文件的默认名称保存（**Storage（存储）**选项卡）。

另请参见：

- PLC 文档：更改带配方的值 [▶ 210]

17.17.9 命令：插入变量

符号： 

功能：此命令在所选位置前添加一个变量到当前打开的配方定义。

调用：菜单 **Recipes（配方）**，上下文菜单

要求：在编辑器中打开配方定义，并选择简单视图。

TwinCAT 在 **Variable（变量）** 列中添加默认文本“NewVariable”。您必须用相应的有效变量名替换该名称。为此，通过 [...] 按钮打开输入助手或直接在表字段中输入变量名。

另请参见：

- PLC 文档：更改带配方的值 [▶ 210]

17.17.10 命令：删除变量

符号 

功能：此命令删除配方定义中的所选变量。

调用：[Del] 键，上下文菜单

要求：已选择一个变量。

另请参见：

- PLC 文档：更改带配方的值 [▶ 210]

17.17.11 命令：更新结构化变量

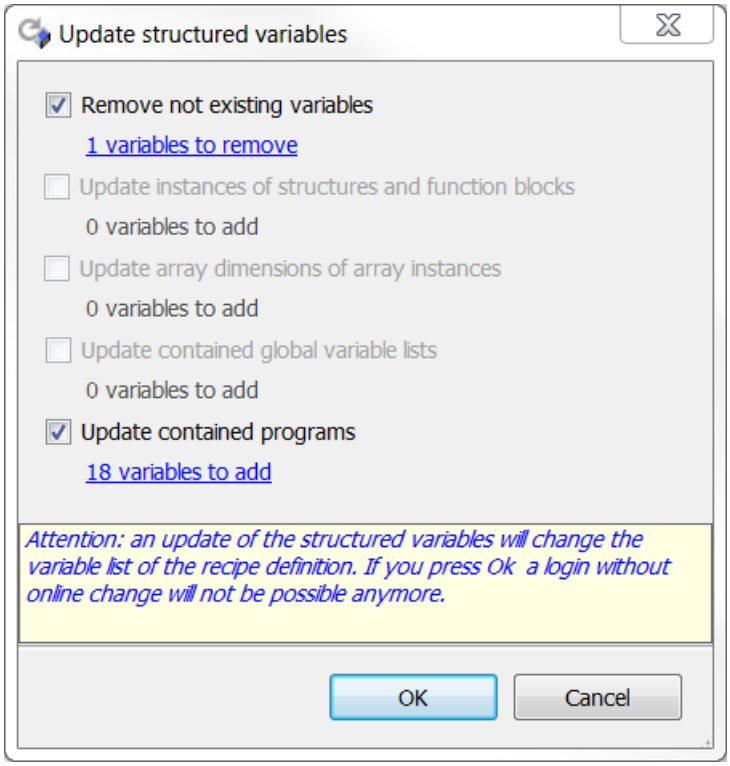
符号： 

功能：此命令打开对话框 **Update structured variables（更新结构化变量）**。

调用：菜单 **Recipes（配方）**

在对话框中，如果结构化变量或功能块的声明已更改，则可以更新配方定义。例如，如果阵列尺寸已更改，则可以自动删除或添加配方定义中的条目。

更新结构化变量对话框



删除不存在的变量	：从配方定义中删除由于结构化元素的更改而不再存在于项目中的变量。
更新结构和功能块的实例	：如果扩展了由配方定义中的实例表示的结构或功能块声明，则相应变量将被添加到配方定义中。
更新阵列实例的阵列尺寸	：如果扩展了由配方定义中的实例表示的阵列尺寸，则相应变量将被添加到配方定义中。
更新所包含的全局变量列表	：如果扩展了由配方定义中的实例表示的全局变量列表声明，则相应变量将被添加到配方定义中。
更新所包含的程序	：如果扩展了在配方定义中实例化的程序声明，则相应变量将被添加到配方定义中。

另请参见：

- PLC 文档：更改带配方的值 [▶ 210]

17.17.12 命令从设备下载配方

符号： 

功能：此命令启动项目中当前打开的配方定义中的配方与设备上以配方文件形式存在的配方的同步。

调用：菜单 **Recipes**（配方）

要求：PLC 项目处于在线模式，您已在编辑器中打开配方定义。具体来说，同步具有以下效果：

- 项目中存在的配方变量的当前值将被控制器上的配方值覆盖。这可能会在下次登录时触发在线更改。
- 如果在控制器上的配方文件中定义了配方变量，而项目中的配方定义中缺少这些变量，则在加载时会忽略这些变量。对于每个配方文件，都会出现 1 条包含受影响变量的消息。
- 如果控制器上的配方文件中缺少配方变量（这些变量包含在项目的配方定义中），则会为每个配方文件显示 1 条显示受影响变量的消息。
- 如果在控制器上为这些变量创建了附加配方，这些配方将被添加到项目中的配方定义中。

17.18 库

命令	PLC 文档的更多信息
库创建	
命令另存为库……	命令另存为库 [▶ 249]
命令另存为库并安装……	命令另存为库并安装 [▶ 250]
库安装	
命令库存储库	库存储库 [▶ 251]
库管理	
库管理器对话框	库管理器 [▶ 254]
其他命令和对话框	
命令添加库	命令添加库 [▶ 329]
命令添加无解析的库	添加无占位符解析的库命令 [▶ 330]
命令尝试重新加载库	命令尝试重新加载库 [▶ 331]
命令删除库	命令删除库 [▶ 331]
命令详细信息	命令详细信息 [▶ 332]
命令相关性	命令相关性 [▶ 332]
命令属性	命令属性 [▶ 333]
命令占位符	占位符 [▶ 258]
命令设置为有效版本	命令设置为有效版本 [▶ 334]
命令设置始终最新版本	命令设置为始终最新版本 [▶ 334]

另请参见：

- PLC 文档：使用库 > 更多命令和对话框

17.19 可视化

可视化命令由可视化命令菜单类别中的 **Visual Editor（可视化编辑器）** 插件提供，位于对话框 **Tools > Customize（工具 > 自定义）** 中。您可通过其编辑可视化编辑器中的可视化对象。

大多数命令作为标准命令包含在 **Visualization（可视化）** 菜单中，因此也可在可视化编辑器的上下文菜单中使用它们。必要时，打开对话框 **Tools > Customize（工具 > 自定义）** 以查看或修改类别 **Visualization（可视化）** 的菜单配置。

17.19.1 命令：界面编辑器

符号：

功能： 此命令打开用于在可视化中定义边框参数的界面编辑器，这些参数将在其他可视化的 **Frame（边框）** 元素中引用。其显示在可视化编辑器上部的选项卡视图中。

调用： 菜单 **Visualization（可视化）**

17.19.2 命令：热键配置

符号：

功能： 此命令打开当前可视化的键盘配置编辑器。其显示在可视化编辑器上部的选项卡视图中。

调用： 菜单 **Visualization（可视化）**

17.19.3 命令：元素列表

符号： 

功能：此命令打开当前可视化的元素列表编辑器。其显示在可视化编辑器上部的选项卡视图中。

调用：菜单 **Visualization（可视化）**

17.19.4 命令：左对齐

符号： 

功能：此命令将所有选中的可视化元素与其最左侧元素的左边缘对齐。

调用：菜单 **Visualization（可视化）**，上下文菜单

17.19.5 命令：顶部对齐

符号： 

功能：此命令将所有选中的可视化元素与其最顶部元素的上边缘对齐。

调用：菜单 **Visualization（可视化）**，上下文菜单

17.19.6 命令：右对齐

符号： 

功能：此命令将所有选中的可视化元素与其最右侧元素的右边缘对齐。

调用：菜单 **Visualization（可视化）**，上下文菜单

17.19.7 命令：底部对齐

符号： 

功能：此命令将所有选中的可视化元素与其最底部元素的下边缘对齐。

调用：菜单 **Visualization（可视化）**，上下文菜单

17.19.8 命令：垂直居中对齐

符号： 

功能：此命令将所有选中的可视化元素与其共同的垂直中心对齐。

调用：菜单 **Visualization（可视化）**，上下文菜单

17.19.9 命令：水平居中对齐

符号： 

功能：此命令将所有选中的可视化元素与其共同的水平中心对齐。

调用：菜单 **Visualization（可视化）**，上下文菜单

17.19.10 命令：使水平间距相等

符号： 

如果选中三个或更多元素，则此命令激活。

1. 选择所有要具有相同水平间距的元素。
   ⇨ 第一个元素以蓝色高亮显示，其他元素显示为灰色。
2. 运行命令 **Make horizontal spacing equal (使水平间距相等)**。

⇨ 最左和最右的元素保持其位置，位于它们之间的元素以相等的水平间距对齐。

17.19.11 命令：增加水平间距

符号： 

如果选中两个或更多元素，则此命令激活。

1. 选择所有要增加水平间距的元素。
   ⇨ 第一个元素以蓝色高亮显示，其他元素显示为灰色。
2. 运行命令 **Increase horizontal spacing (增加水平间距)**。

⇨ 蓝色元素保持其位置，其他元素水平对齐，各元素之间的间距增加。间距增加 1 个像素。

17.19.12 命令：缩小水平间距

符号： 

如果选中两个或更多元素，则此命令激活。

1. 选择所有要缩小水平间距的元素。
2. 运行命令 **Decrease horizontal spacing (缩小水平间距)**。

⇨ 蓝色元素保持其位置，其他元素水平对齐，各元素之间的间距缩小。间距减小 1 个像素。

17.19.13 命令：删除水平间距

符号： 

如果选中两个或更多元素，则此命令激活。

1. 选择所有不设置水平间距的元素。
   ⇨ 第一个元素以蓝色高亮显示，其他元素显示为灰色。
2. 运行命令 **Remove horizontal spacing (删除水平间距)**。

⇨ 蓝色元素保持其位置，其他元素水平对齐，彼此之间无间距。

17.19.14 命令：使垂直间距相等

符号： 

如果选中两个或更多元素，则此命令激活。

1. 选择所有要具有相同垂直间距的元素。
   ⇨ 第一个元素以蓝色高亮显示，其他元素显示为灰色。
2. 运行命令 **Make vertical spacing equal (使垂直间距相等)**。

⇨ 最顶部和最底部的元素保持其位置，位于它们之间的元素以相等的垂直间距对齐。

17.19.15 命令：增加垂直间距

符号： 

如果选中两个或更多元素，则此命令激活。

1. 选择所有要增加垂直间距的元素。
   ⇨ 第一个元素以蓝色高亮显示，其他元素显示为灰色。
2. 运行命令 **Increase vertical spacing (增加垂直间距)**。

⇨ 蓝色元素保持其位置，其他元素垂直对齐，各元素之间的间距增加。间距增加 1 个像素。

17.19.16 命令：缩小垂直间距

符号： 

如果选中两个或更多元素，则此命令激活。

1. 选择所有要缩小垂直间距的元素。
   ⇨ 第一个元素以蓝色高亮显示，其他元素显示为灰色。
2. 运行命令 **Decrease vertical spacing (缩小垂直间距)**。

⇨ 蓝色元素保持其位置，其他元素垂直对齐，各元素之间的间距缩小。间距减小 1 个像素。

17.19.17 命令：删除垂直间距

符号： 

如果选中两个或更多元素，则此命令激活。

1. 选择所有不设置垂直间距的元素。
   ⇨ 第一个元素以蓝色高亮显示，其他元素显示为灰色。
2. 运行命令 **Remove vertical spacing (删除垂直间距)**。

⇨ 蓝色元素保持其位置，其他元素垂直对齐，彼此之间无间距。

17.19.18 命令：使宽度相等

符号： 

如果选中超过一个元素，则此命令激活，但选择一个行或多边形元素除外。

1. 选择应具有相同宽度的元素。
   ⇨ 第一个元素以蓝色高亮显示，其他元素显示为灰色。
2. 运行命令 **Make same width (使宽度相同)**。

⇨ 所有元素被设置为以蓝色标记的元素的宽度。

17.19.19 命令：使高度相等

符号： 

如果选中超过一个元素，则此命令激活，但选择一个行或多边形元素除外。

1. 选择应具有相同高度的元素。
   ⇨ 第一个元素以蓝色高亮显示，其他元素显示为灰色。

2. 运行命令 **Make same width (使高度相同)**。
⇨ 所有元素被设置为以蓝色标记的元素的高度。

17.19.20 命令：使大小相等

符号： 

如果选中超过一个元素，则此命令激活，但选择一个行或多边形元素除外。

1. 选择应具有相同大小的元素。
   ⇨ 第一个元素以蓝色高亮显示，其他元素显示为灰色。
2. 运行命令 **Make same width (使大小相同)**。
⇨ 所有元素被设置为以蓝色标记的元素的大小。

17.19.21 命令：对齐网格

符号： 

如果选中超过一个元素，则此命令激活，但选择一个行或多边形元素除外。

1. 选择位置和大小应与网格对齐的所有元素。
   ⇨ 第一个元素以蓝色高亮显示，其他元素显示为灰色。
2. 运行命令 **Size to Grid (对齐网格)**。
⇨ 所有元素均基于其大小和位置对齐到网格。

17.19.22 命令：前置

符号： 

功能：此命令将所选元素向上移动一级，即可视化前景中的更高一层。较低层次的元素被较高层次的元素遮盖。

调用：菜单 **Visualization (可视化)**，上下文菜单

17.19.23 命令：置于顶层

符号： 

功能：此命令将所选元素移动到可视化的前景中，即最高层。较低层次的元素被较高层次的元素遮盖。

调用：菜单 **Visualization (可视化)**，上下文菜单

17.19.24 命令：后置

符号： 

功能：此命令将所选元素向下移动一级，即可视化背景中的更低一层。较低层次的元素被较高层次的元素遮盖。

调用：菜单 **Visualization (可视化)**，上下文菜单

17.19.25 命令：置于底层

符号：

功能：此命令将所选元素移动到可视化的背景中，即最低层。较低层次的元素被较高层次的元素遮盖。

调用：菜单 **Visualization（可视化）**，上下文菜单

17.19.26 命令：分组

符号：

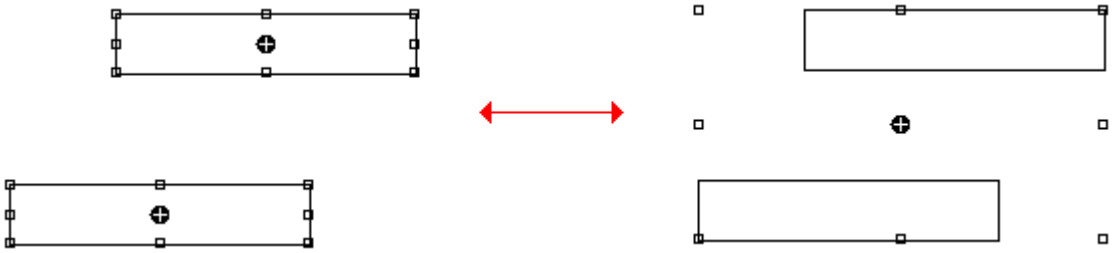
功能：此命令对当前所选可视化元素进行分组并将该组显示为一个单一对象。

调用：菜单 **Visualization（可视化）**，上下文菜单

要求：选择多个可视化元素。如要选择多个对象，按住 [<Shift>] 键并点击需要的项。或者可以点击元素外的编辑器窗口，并按住鼠标键围绕目标元素绘制一个矩形。

使用命令 **Ungroup（取消组）** 解除该组。

下图显示的是两个矩形元素的分组（从左到右）或取消组（从右到左）：



- 命令：取消组 [▶ 985]

17.19.27 命令：取消组

符号：

功能：此命令解除可视化元素组。单独的元素再次被单独选中。

调用：菜单 **Visualization（可视化）**，上下文菜单

另请参见：

- 组 [▶ 985]

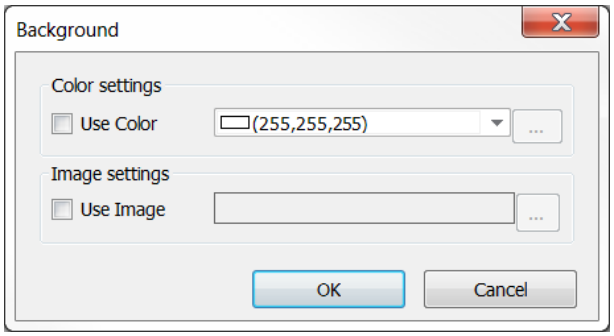
17.19.28 命令：背景

符号：

功能：此命令打开对话框 **Frame Configuration（边框配置）**，可在该对话框中选择可视化背景的颜色或图像。

调用：菜单 **Visualization（可视化）**，上下文菜单

背景对话框



启用所需选项：

- 位图：要定义背景图像，请输入项目图像库中可用的图像文件路径。输入图像库的名称和图像文件 ID，用点号“.”分隔：<image pool>.<ID>（例如“Images_1.drive_icon>”，“Images_1.43”）。
- 颜色：要定义可视化的背景颜色，从颜色列表中选择所需颜色。

17.19.29 命令：全选

符号： /

功能：此命令选择当前在编辑器中打开的所有可视化元素。

调用：菜单 **Visualization（可视化）**，上下文菜单

另请参见：

- 取消全选 [▶ 986]

17.19.30 命令：取消全选

符号：

功能：此命令清除当前选择的可视化元素。

调用：菜单 **Visualization（可视化）**

另请参见：

- 命令：全选 [▶ 986]

17.19.31 命令：倍增可视化元素

符号：

功能：此命令打开对话框 **Multiply visu element（倍增可视化元素）**，您可使用该对话框配置倍增步骤。

调用：菜单 **Visualization（可视化）**，上下文菜单

要求：可视化处于活动状态并选择模板元素。

倍增可视化元素对话框

TwinCAT 添加更多与模板元素类似的元素。您可在此对话框中设置索引的数量和排列以及它们的替换。

基本设置选项卡

元素总数：

TwinCAT 将新元素作为表格插入。行数以**水平**方式设置，列数以**垂直**方式设置。两者的乘积决定了要插入元素的实际总数。	
水平	每行的元素数 预设：对应于 $FIRSTDIM$ 中的组件数
垂直	每列的元素数 预设：对应于 $SECONDDIM$ 中的组件数

元素之间的偏移量：

TwinCAT 将可视化中的元素排列为表格。如果您指定了一个偏移量，则在元素之间插入间距。 • 0：元素边框重叠一个像素 • 1：元素相互接触 • <n>：元素之间插入可见的 n-1 像素间距	
水平	元素行间距的像素数
垂直	元素列间距的像素数

高级设置选项卡

第一层：

起始指数	$FIRSTDIM$ 的起始指数 预设：1 表示 $FIRSTDIM$ 的特定指数，从 1 开始。示例：阵列 [1, <$SECONDDIM$>]
增量	指数的递增数 预设：1

第二层：

起始指数	$SECONDDIM$ 的起始指数 预设：1 表示 $SECONDDIM$ 的特定指数，从 1 开始。示例：阵列 [<$FIRSTDIM$>, 1] 具有相关性
增量	指数的递增数 预设：1

17.19.32 命令：激活键盘

符号：

功能：此命令可在菜单栏中用于集成可视化（诊断可视化）。该命令启用或禁用可视化在线模式中的键盘使用。

调用：菜单 **Visualization（可视化）**

如果启用键盘，可以通过特定的快捷方式输入和选择元素。在此情况下，只要可视化编辑器激活并在线，就不会执行其他键盘命令。

17.20 其他

17.20.1 命令：实现界面

功能：此命令通过添加功能块当前未包含的界面元素来更新功能块的实现界面。

调用：在 PLC 项目树中选择功能块时的上下文菜单。

要求：该功能块使用您已更改的界面。例如您已将另一个方法添加到界面。

应用

执行此命令时，自动创建的方法或属性将被赋予一个编译指示属性，这将引发编译错误或警告。这提供了支持，因为自动创建的元素不会意外保持为空。根据应用使用错误或警告属性。

案例 1：

情况：执行命令 **Implement interfaces（实现界面）**的功能块**并非**源于其他功能块。

结果：执行命令时，界面元素在功能块中创建而无需实现（“存根”）并被赋予警告属性（在方法/属性声明的第一行）。编译期间生成的警告提醒您，这些元素是自动生成的，您需要添加所需的实现代码。

```
{warning 'add method/property implementation'}
```

步骤：将所需的实现代码添加到相应的界面元素（方法或属性）。然后删除方法或属性声明中的警告属性。

案例 2：

情况：执行命令“Implement interfaces（实现界面）”的功能块源于其他功能块。在派生功能块中执行命令时创建的元素（方法或属性）**并非**由基础功能块的继承提供（也就是说，元素不在基础功能块或更高父类别之下）。

结果/步骤：见案例 1。

案例 3：

情况：执行命令 **Implement interfaces（实现界面）**的功能块源于其他功能块。在派生功能块中执行命令时创建的元素（方法或属性）已由基础功能块的继承提供（也就是说，元素位于基础功能块或更高父类别之下）。

结果：执行命令时，界面元素在派生功能块中创建而无需实现（“存根”）并被赋予错误属性（在方法/属性声明的第一行）。编译期间生成的警告提醒您，这些元素是自动生成的，并且该方法或属性覆盖基本功能块的对应元素。

```
{error 'add method/property implementation or delete method/property to use base implementation'}
```

步骤：如果您希望覆盖或增强基本功能块的方法或属性，请将所需的实现代码添加到派生功能块下的元素中。然后删除方法或属性声明中的错误属性。但是，如果您**不**想覆盖基本功能块的方法或属性，请删除派生功能块下的方法或属性。在此情况下，使用基本功能块的方法或属性。

17.21 上下文菜单 TwinCAT 项目

17.21.1 命令将 <TwinCAT project name> 另存为存档……

符号： 

功能：此命令可打开用于将文件另存为存档的标准对话框。项目可作为 ***.tszip** 存档保存在目标路径下。

调用：File（文件）菜单，上下文菜单

要求：在 Solution Explorer（解决方案资源管理器）中选择 TwinCAT 项目。

***.tszip 的内容**	*.tszip 存档文件夹包含要存档的 TwinCAT 项目。
打开命令	使用以下命令可以重新打开 tszip 存档： 命令项目/解决方案（打开项目/解决方案）[▶ 799]
关于 PLC 项目的说明	如果 TwinCAT 项目包含 1 个或多个 PLC 项目，则在这些 PLC 项目的存档文件夹中存储的文件和文件夹将取决于相关项目的 PLC 项目设置。 设置选项卡 [▶ 859]

17.21.2 命令：通过电子邮件发送...

符号： 

功能：此命令启动系统中设置的电子邮件程序并打开一个新电子邮件，该邮件以所选项目的存档文件作为附件。

调用：File（文件）菜单，上下文菜单

17.21.3 命令自动将 <TwinCAT project names> 备份到目标系统

功能：此命令可用于启用或禁用在激活过程中将 TwinCAT 项目以 .tszip 格式自动存储到目标系统中。如果您想要稍后使用命令 命令：从目标打开项目 [▶ 806] 从目标系统加载并打开项目，则必须这样做。

调用：TwinCAT 项目的上下文菜单

要求：在 Solution Explorer（解决方案资源管理器）中选择 TwinCAT 项目。

17.21.4 命令将 <TwinCAT project name> 与目标系统进行比较……

功能：此命令可以使用 TwinCAT 项目比较工具将所选的 TwinCAT 项目与目标系统上的项目状态进行比较。

调用：TwinCAT 项目的上下文菜单

要求：在 Solution Explorer（解决方案资源管理器）中选择 TwinCAT 项目，并选择已下载 TwinCAT 项目的目标系统。

另请参见：

- 命令使用目标系统更新项目…… [▶ 989]

17.21.5 命令使用目标系统更新项目……

功能：此命令可以将所选的 TwinCAT 项目更新为连接的目标系统的项目状态。为此，系统会打开 1 个弹出对话框，显示已更改的文件列表，您必须确认该对话框才能成功执行动作。此处不提供详细的比较。

调用：TwinCAT 项目的上下文菜单

要求：在 Solution Explorer（解决方案资源管理器）中选择 TwinCAT 项目。

另请参见：

- 命令将 <TwinCAT project name> 与目标系统进行比较…… [▶ 989]

17.21.6 命令使用 TwinCAT 2.xx 版本加载项目……

功能：此命令可打开标准浏览器对话框，通过该对话框可以选择并导入 TwinCAT 2 系统管理器文件。通过这种方式，您可以将现有的 TwinCAT 2 项目（包括系统管理器设置和 PLC 项目）转换为 TwinCAT 3。

调用：TwinCAT 项目的上下文菜单

要求：TwinCAT 项目已新建且未进行任何更改，并在 Solution Explorer（解决方案资源管理器）中选中。

另请参见：

- 打开 TwinCAT 2 PLC 项目 [▶ 51]

17.21.7 命令显示隐藏的配置

功能：此命令可打开 1 个详细菜单，其中列出了隐藏的配置，您可以再次显示它们。

调用：TwinCAT 项目的上下文菜单

要求：在 Solution Explorer（解决方案资源管理器）中选择 TwinCAT 项目。

17.21.8 命令从解决方案中删除

符号： 

功能：此命令可以从解决方案中删除 TwinCAT 项目。

调用：TwinCAT 项目的上下文菜单

要求：在 Solution Explorer（解决方案资源管理器）中选择 TwinCAT 项目。

17.21.9 命令重命名

符号： 

功能：此命令可以在 **Solution Explorer**（解决方案资源管理器）中重命名 TwinCAT 项目。

调用：TwinCAT 项目的上下文菜单

要求：在 Solution Explorer（解决方案资源管理器）中选择 TwinCAT 项目。

17.21.10 命令构建 TwinCAT 项目

符号： 

功能：此命令可启动当前活动的 TwinCAT 项目的编译过程或代码生成。

调用：如果当前选择的是 TwinCAT 项目，则调用 **Build**（构建）菜单，或 TwinCAT 项目的上下文菜单

要求：选择 TwinCAT 项目。

TwinCAT 项目中包含的所有项目（PLC、C++ 等）依次进行编译。在 命令构建 PLC 项目 [▶ 863] 部分中描述了针对 PLC 项目执行的步骤。

另请参见：

- 命令重建 TwinCAT 项目 [▶ 990]

17.21.11 命令重建 TwinCAT 项目

功能：此命令可启动当前活动的 TwinCAT 项目的编译过程或代码生成，即使上次编译时没有错误也不例外。

调用：如果当前选择的是 TwinCAT 项目，则调用 **Build**（构建）菜单，或 TwinCAT 项目的上下文菜单

要求：选择 TwinCAT 项目。

在重建项目时，首先要对 TwinCAT 项目进行清除（另请参见：命令清除 TwinCAT 项目 [▶ 990]），然后再进行构建（另请参见：命令构建 TwinCAT 项目 [▶ 990]）。

17.21.12 命令清除 TwinCAT 项目

功能：此命令可删除当前活动的 PLC 项目的本地编译信息，并更新所有对象的语言模型。

调用：如果当前选择的是 TwinCAT 项目，则调用 **Build**（构建）菜单，或 TwinCAT 项目的上下文菜单

要求：选择 TwinCAT 项目。

TwinCAT 项目中包含的所有项目（PLC、C++ 等）依次进行清除。在 命令清除 PLC 项目 [▶ 863] 部分中描述了针对 PLC 项目执行的步骤。

另请参见：

- 命令重建 TwinCAT 项目 [▶ 990]

17.21.13 命令卸载项目

功能：此命令可卸载 TwinCAT 项目，以便释放 TwinCAT 项目的所有文件。

调用：项目菜单或 TwinCAT 项目的上下文菜单

要求：在 Solution Explorer（解决方案资源管理器）中选择 TwinCAT 项目。

17.21.14 通过 AML DataExchange 导入 AutomationML……

功能：此命令可打开标准浏览器对话框，通过该对话框您可以搜索并导入 AutomationML 格式的文件。

调用：从 TwinCAT 项目的上下文菜单中的 **Import AutomationML**（导入 AutomationML）下或者从菜单栏中的 **TwinCAT** 项目的 **AutomationML** 和 **Import AutomationML**（导入 AutomationML）下可以调用此命令。

要求：在 Solution Explorer（解决方案资源管理器）中选择 TwinCAT 项目。

另请参见

- 命令：

17.21.15 导出 AutomationML……

功能：此命令可打开标准浏览器对话框，以 AutomationML 格式保存文件，用于导出 I/O 节点下的拓扑结构。

调用：从 TwinCAT 项目的上下文菜单或者从菜单栏中的 **TwinCAT** 项目的 **AutomationML** 下可以调用此命令。

要求：在 Solution Explorer（解决方案资源管理器）中选择 TwinCAT 项目。

18 PLC 编程规范

在编程语言领域中，根据编程语言的工具、重点和来源，以及应用程序所处的行业，有一些需要遵守的规范。

根据公认的编程指南（例如，MISRA C++ 2008 指南 [...]、Java 编码规范、C# 编程指南或 PLCopen），以下部分包含更多编程信息，有助于创建经过优化且实用的 PLC 程序代码。

遵循 PLC 编程规范，可实现以下优势：

- 确保 PLC 程序的统一结构
- 对象、变量和实例保持一致的命名
- 代码（特别是功能块、方法和函数的接口）可轻松识读并直观理解
- 简化程序的开发、使用和维护
- 通过系统且尽早地避免错误源，提高代码质量

全新倍福 TwinCAT 3 PLC 库、程序示例和应用程序均遵循这些“TwinCAT 3 PLC 编程规范”。

例外：

- Tc2_MC2_xxx 库遵循 PLCopen 编程规范。
- Tc2_xxx 库遵循原有的 TwinCAT 2 库。
- 行业特定库基于相关行业标准。
- IEC61131 标准的功能块，是 Tc2_Standard 库的一部分。

分类

PLC 编程规范分为以下几个章节：

- 编程风格 [▶ 993]
- 命名规范 [▶ 1003]
- 编程 [▶ 1012]

每个章节下设若干子章节，子章节中又列出了各个主题。

单个主题被分类为

- 标有 [++] 的规定，或
- 标有 [+] 的建议。

您必须执行被归类为规定的主题。如果可能，建议应该始终得到执行，因为出于多种原因，这些建议具有合理性。

静态代码分析

为降低运行时出错的风险，至少应该使用免费的静态代码分析变体（轻量静态分析 [▶ 139]）对代码进行检查。TwinCAT 3.1 Build 4018 及以上版本均包含此功能，其中包含代码审查的最小范围。借助 TE1200 PLC 静态分析的全部功能，还可以对编程规范进行更详细的审查（请参见以下部分）。

借助 TE1200 PLC 静态分析验证编程规范

TE1200 PLC 静态分析可对 PLC 程序代码进行全面检查。借助这些功能，可以检查 PLC 编程规范是否符合各项规定、建议和命名规范。

子章节编程

在可用的情况下，每条规定/建议都附有相应的静态分析规则，可用于验证是否符合该规定/建议。

另请参见：

- 子章节 编程 [▶ 1012]
- TE1200 PLC 静态分析中的功能规则

子章节命名规范

在可用的情况下，可为每项命名规范提供静态分析配置字段的 ID，其中必须输入相应的推荐名称前缀。这样，静态分析可以检查 POU、DUT、变量和实例的大部分推荐前缀。

另请参见：

- 子章节 命名规范 [▶ 1003]
- TE1200 PLC 静态分析中的功能命名规范

编译指示和属性

另请注意，对于不应再报告特定规则/命名规范的已检查位置，通过编译指示或属性可以在本地禁用静态分析规则和命名规范（另请参见：TE1200 PLC 静态分析中的编译指示和属性章节）。理想情况下，您应该对本地停用情况进行注释，并做出适当解释。

下载静态分析配置文件

1) 规定和建议

通过以下链接，您可以下载 1 个预先构建的静态分析配置文件，其中包含作为规则启用的编程规范的规定 [++] 和建议 [+]。在执行静态分析之后，违反规定的行为会被输出为错误，而违反建议的行为会被列为警告。

https://infosys.beckhoff.com/content/1033/tc3_plc_intro/Resources/12663400331.zip

2) 规定、建议和命名规范

通过以下链接，您可以下载 1 个预先构建的静态分析配置文件，其中包含作为规则启用的编程规范的规定 [++]、建议 [+] 和命名规范。在执行静态分析之后，违反规定或命名规范的行为会被输出为错误，而违反建议的行为会被列为警告。

https://infosys.beckhoff.com/content/1033/tc3_plc_intro/Resources/12663402507.zip

更多规则

除了在以下编程规范中明确提及的规则之外，TE1200 还提供了许多其他有用的规则，它们可用于在早期阶段检测疏失错误或一般编程错误。其中包括但不限于以下规则。

- SA0008：检查子范围类型
- SA0041：可能的循环不变量代码
- SA0044：引用接口的声明
- SA0056：常量超出有效范围
- SA0059：比较操作总是返回 TRUE 或 FALSE
- SA0060：零被用作无效操作数
- SA0062：在表达式中使用 TRUE 和 FALSE
- SA0140：注释掉的语句

18.1 编程风格

本部分 PLC 编程规范涵盖以下主题。

字体和编辑器设置

1. 使用相同的字体设置 [▶ 994] [+]

语言

1. 使用英语 [▶ 994] [+]
2. 观察有效字符 [▶ 994] [+]

项目结构

1. 模块化项目结构 [▶ 995] [+]
2. 文件夹结构 [▶ 995] [+]

3. 项目树中的排序模式 [▶ 995] [+]
4. 没有未使用的声明/对象或无用代码 [▶ 996] [+]

程序结构

1. 面向对象的编程 [▶ 996] [+]
2. 程序元素的结构 [▶ 996] [+]
3. 文本块的结构 [▶ 996] [+]
4. 注释 [▶ 996] [+]

18.1.1 字体和编辑器设置

主题：

1. 使用相同的字体设置 [▶ 994] [+]

使用相同的字体设置

您应该使用相同的字体设置，以确保在不同程序中实现字体统一。当 TwinCAT 3 集成到 Visual Studio 中时，字体和制表符宽度在默认情况下是预先设置的。

设置：

工具 > 选项 > 环境 > 字体和颜色	
字体	Consolas
字体大小	10
工具 > 选项 > TwinCAT > PLC 环境 > 文本编辑器	
轮廓	缩进
自动换行	不中断
制表符宽度	4
保留制表符	是
缩进宽度	4
自动缩进	智能代码补全

18.1.2 语言

主题：

1. 使用英语 [▶ 994] [+]
2. 观察有效字符 [▶ 994] [+]

使用英语

PLC 程序（程序代码、变量名、注释等）完全用英语编写。此处建议使用美式英语。

观察有效字符

程序元素和变量的名称只能包含以下字符、数字和特殊字符：

- 0……9
- A……Z
- a……z
- 下划线字符“_”是唯一有效的分隔符（更多信息见下文）

一般来说，您应避免使用双下划线（“__”），因为它们用于内部变量。

使用下划线字符“_”：

下划线字符“_”作为唯一有效的分隔符，适用于以下名称：

- 对于介于前缀和对象名称之间的对象。示例：
  ```
  FB_GetData, ST_BufferEntry, I_CylinderControl, E_StandardColor
  ```
- 可选择对项目树中的对象实施排序方案 [▶ 995]

18.1.3 项目结构

主题：

1. 模块化项目结构 [▶ 995] [+]
2. 文件夹结构 [▶ 995] [+]
3. 项目树中的排序模式 [▶ 995] [+]
4. 没有未使用的声明/对象或无用代码 [▶ 996] [+]

模块化项目结构

您以模块化的方式构建 TwinCAT 3 PLC 项目。文件夹结构应以 PLC 项目的各种功能和对象为指导。如有必要，您可以根据固定方案对程序元素进行排序。

文件夹结构

TwinCAT 3 PLC 项目的文件夹结构应该是模块化的，并以 PLC 项目的不同功能/对象为基础。在第 1 级别，您可以将 PLC 项目划分为多个模块文件夹。在第 2 级别，您可以根据模块的复杂程度，按照元素类型（DUT、ITF、POU 等）实施更精细的模块化或排序。

- 第 1 级的文件夹名称：功能/模块的描述
- 第 2 级的文件夹名称：程序元素的进一步模块化或描述

示例：

```
Tc3_Plc_Project
                    + [Topic X]
                                        + DUTs
                                        + ITFs
                                        + POUs
                    + [Topic Y]
                                        + [Topic Y, Part a]
                                                            + DUTs
                                                            + POUs
                                        + [Topic Y, Part b]
                                                            + ITFs
                                                            + POUs
```

项目树中的排序模式

项目树中的对象按 TwinCAT 3 的字母顺序排序。因此，项目树中对象排序的可选方案源自于对象的名称。

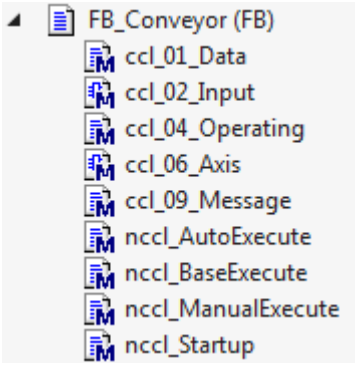
排序方案可由数字、关键字或其缩写的组合而成。它位于前缀和对象的实际名称之间，并用下划线将前缀和对象名称隔开。如果排序方案由多个元素（例如，带有关键字或其缩写的数字）组成，则各子元素之间用下划线字符隔开。

在程序元素中的适当位置应该标注排序方案。

另外，POU 子文件夹可用于按主题对 POU 的子项目进行排序。

排序方案的示例

以下功能块的方法使用缩写和数字进行排序。缩写根据调用间隔排列方法（“ccl” = 周期性；“nccl” = 非周期性，例如，事件驱动调用）；数字根据预期调用顺序对周期性方法进行排序。以这些方法为例进行说明。您可自由选择用于排序的关键字；在关键字和数字之间可以选择下划线（例如，“ccl01_Data”或“ccl_01_Data”）。



没有未使用的声明/对象或无用代码

项目中未使用的程序元素很快就会导致程序树/代码结构混乱。在维护和扩展的过程中，如果项目只包含实际使用的程序元素，则可大大提高代码的识读率。

另请参见 编程 [▶ 1012] 部分中的没有未使用的声明/对象或无用代码 [▶ 1019]主题。

18.1.4 程序结构

主题：

1. 面向对象的编程 [▶ 996] [+]
2. 程序元素的结构 [▶ 996] [+]
3. 文本块的结构 [▶ 996] [+]
4. 注释 [▶ 996] [+]

面向对象的编程

如要利用面向对象的编程的优势，您应该将 PLC 程序设计为类结构。使用的不是大量的功能块，而是精选的带有适当方法的类。

您应该根据具体情况评估面向对象的实现的好处和用户友好性。

程序元素的结构

请参见：程序元素的结构 [▶ 997]

文本块的结构

请参见：文本块的结构 [▶ 999]

注释

尽可能避免不必要的注释。相反，尽可能使命名和程序代码一目了然，这样无需进一步注释。如果因为对理解（例如，单元）有很大帮助而需要注释，请注意以下几点：

注释旨在描述程序元素及其组件的用途、好处和内容，从而使人们更快、更容易理解程序。它们会在查看程序元素时显示，也会以工具提示的形式显示。

注释位置

如果提供注释，则注释可位于以下位置：

- 程序元素的注释：在元素声明的第 1 行上方（例如，在 FUNCTION_BLOCK 关键字上方）

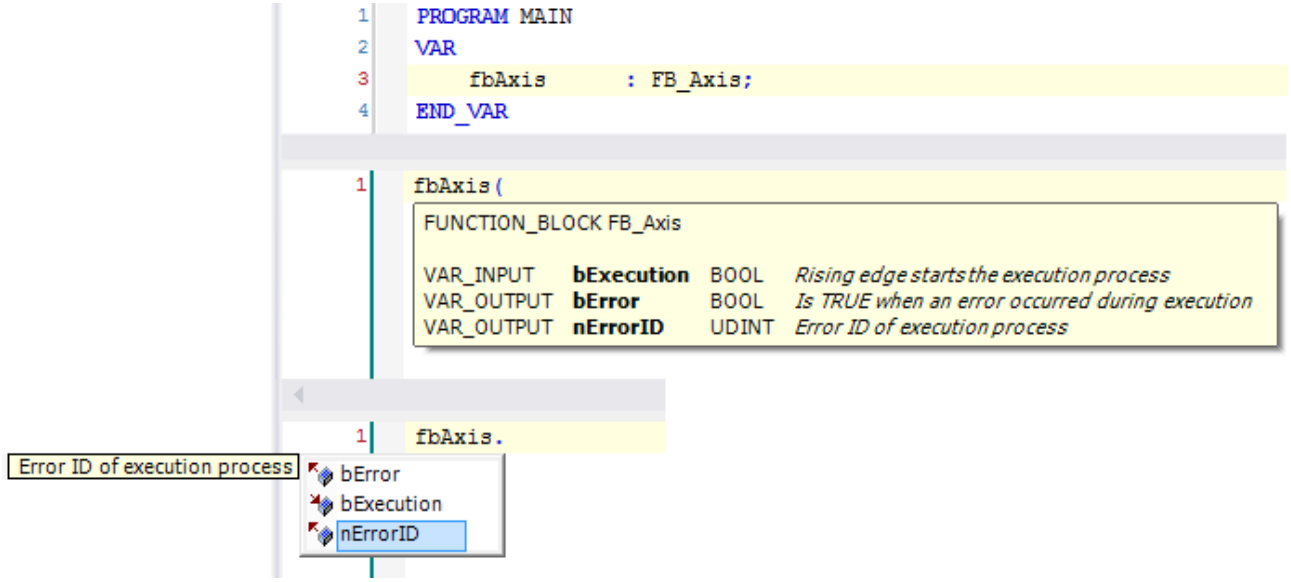
- 变量的注释：在声明后的同一行中
- 声明/程序块：在程序块上方一行

注释运算符

- 对于单行注释，可使用注释运算符“//”。
- 对于影响多行的注释，可使用注释运算符“//”或运算符“(* ... *)”。

示例

```
// This function block represents an axis
FUNCTION_BLOCK FB_Axis
VAR_INPUT
     bExecution     : BOOL;     // Rising edge starts the execution process
END_VAR
VAR_OUTPUT
     bError         : BOOL;     // Is TRUE when an error occurred during execution
     nErrorID       : UDINT;    // Error ID of execution process
END_VAR
```



18.1.4.1 程序元素的结构

每个新的语句/声明都应该以新行开始。

以下部分将解释如何使用文本块、文本段和文本区域来构建程序元素（包括 POU、DUT 和 GVL）。

文本块

- 在文本块中，主题相关的声明、赋值或者只有 1 个参数或没有参数的调用均被分组到一起。
- 带有多个参数的调用、语句（例如，IF 或 CASE）和循环用单独的文本块表示。
- 每个文本块均以空行开始和结束。
- 如果您想要利用可以折叠区域注释下方代码的优势，可以使用制表符键在文本区域内将文本块缩进 1 级。如果您使用缩进，则应在 POU 内一致地实现。

文本段

- 与主题相关的文本块被捆绑在文本段中。
- 每个文本段均以空行开始和结束。
- 文本段的第 1 行包含与后续文本块相关的注释。
- 如果您想要利用可以折叠区域注释下方代码的优势，可以使用制表符键在文本区域内将文本段缩进 1 级。如果您使用缩进，则应在 POU 内一致地实现。

文本区域

- 与主题相关的文本段被捆绑在文本区域中。

- 每个文本区域均以注释中的分界线开始和结束。
- 分隔符下面的第 1 行包含与后续文本段以及文本区域相关的注释。该区域注释与上面的注释字符处于同一级别。
- 区域注释不缩进。如果区域中包含的文本块和文本段是缩进的，则除区域注释外，文本区域的所有行都可以展开或折叠。

示例

有 2 个文本区域（“钻孔设置”和“传送带设置”）和 2 个文本段（“位置”和“速度”）。在文本区域“钻孔设置”中，文本段“速度”由 2 个文本块（与主题相关的关联和 1 个 IF 语句）组成。

程序代码：

```
//=========================================================
// Drill settings

    // Positions
    fbDrill.nPositionLower  := 100;
    fbDrill.nPositionTop    := 500;

    // Velocities
    fbDrill.fVelocityRated  := 40.0;
    fbDrill.fVelocityMax    := 100.0;

    IF fbDrill.fVelocityAverage > fbDrill.fVelocityRated THEN
        bWarning := TRUE;
        sWarning := 'Drill velocity: Average exceeded rated';
    END_IF

//=========================================================
// Conveyor settings

    // Positions
    fbConveyor.nPositionFilling    := 50;
    fbConveyor.nPositionDrill      := 200;
    fbConveyor.nPositionDischarge  := 300;

    // Velocities
    fbConveyor.fVelocityRated      := 75.5;

//=========================================================
```

展开文本区域：

```
1  //=========================================================
2  // Drill settings
3
4      // Positions
5      fbDrill.nPositionLower  := 100;
6      fbDrill.nPositionTop    := 500;
7
8      // Velocities
9      fbDrill.fVelocityRated  := 40.0;
10     fbDrill.fVelocityMax    := 100.0;
11
12     IF fbDrill.fVelocityAverage > fbDrill.fVelocityRated THEN
13         bWarning    := TRUE;
14         sWarning    := 'Drill velocity: Average exceeded rated';
15     END_IF
16
17 //=========================================================
18 // Conveyor settings
19
20     // Positions
21     fbConveyor.nPositionFilling     := 50;
22     fbConveyor.nPositionDrill       := 200;
23     fbConveyor.nPositionDischarge   := 300;
24
25     // Velocities
26     fbConveyor.fVelocityRated       := 75.5;
27
28 //=========================================================
```

折叠文本区域：

```
1  //=========================================================
2  // Drill settings
3  [13 lines]
17 //=========================================================
18 // Conveyor settings
19 [8 lines]
28 //=========================================================
```

另请参见：

- 实现折叠功能的另一种可能性：区域编译指示 [▶ 781]

18.1.4.2 文本块的结构

DUT、GLV 和 POU 中的文本块由声明、表达式、赋值、调用、语句或循环组成。

声明

- 声明由 3 至 5 列组成：
  - 变量名
  - [Allocation (e.g. AT %Q*)]
  - 数据类型
  - [Initialization]
  - [Comment]

- 是否存在分配、初始化和注释的列取决于相关程序。
- 程序元素中的每个声明列都从同一行级别开始（例如：所有数据类型都从同一行级别开始）。
- 这些变量声明（如果存在）的创建顺序是一致的：

```
VAR_INPUT
END_VAR
VAR_IN_OUT CONSTANT
END_VAR
VAR_IN_OUT
END_VAR
VAR_OUTPUT
END_VAR

VAR
END_VAR
VAR PERSISTENT
END_VAR
VAR CONSTANT
END_VAR
```

- 每个声明的变量都有 1 个标识符，尽可能使标识符一目了然。如果需要更多信息来说明变量的目的或功能，则可在声明之后（在同一行中）通过注释进行描述。
- 与主题相关的声明
  - 被捆绑在 1 个声明块中，
  - 使用注释进行描述
  - 并使用空行与其他声明块隔开。

示例：

```
VAR
    // Movement control
    bExecuteMovements     AT%I*   : BOOL;             (* Controls the movement
                                                        execution of car and bike *)
    bMovementsDone        AT%Q*   : BOOL;             (* Indicates if the car and
                                                        bike finished their movement *)

    // General distance variables
    nMovementTime                 : INT;              // Elapsed movement time in seconds
    nDistanceTotal                : INT;              // Total distance being covered by car and bike
    nStartPosition                : INT       := 100; // General start position where car and bike be
gin to move

    // Car
    fbCar                         : FB_Car;           // Car-FB whose movement is controlled
    nDistanceCar                  : INT;              // Distance being covered by car

    // Bike
    fbBike                        : FB_Bike;          // Bike-FB whose movement is controlled
    nDistanceBike                 : INT;              // Distance being covered by bike
END_VAR
```

表达式

- 对于表达式，每个运算符之间用空格隔开。
- 根据子表达式的数量和复杂程度，可以使用括号来直观地说明处理顺序并提高识读率。

示例：

```
nDistanceCar   := fbCar.nStartPosition + (fbCar.nVelocity * nMovementTime);
nDistanceBike  := fbBike.nStartPosition + (fbBike.nVelocity * nMovementTime);
nDistanceTotal := nDistanceCar + nDistanceBike;
```

赋值

- 赋值块的分配运算符从同一行级别开始。
- 通过制表符键可以实现相同的行级别。
- 分配运算符后跟 1 个空格。
- 超过屏幕宽度的长赋值将通过换行进行分割。在新行结束的文本部分将缩进（使用制表符键）到分配运算符之后的 1 级，例如：

```
bExpressionResult := bCondition1 AND (bCondition2 OR bCondition3)
                     AND bCondition4 AND (bCondition5 OR bCondition6);
```

- 与主题相关的赋值
  - 被捆绑在 1 个赋值块中
  - 使用空行与其他块隔开。

示例：

```
//=======================================================
// Submodule enable

    // Sensors
    fbSensorEStop.bEnable    := bGeneralEnable AND bSensorEnable;
    fbSensorStartPos.bEnable := bGeneralEnable AND bSensorEnable;

    // Cylinder
    fbCylinderSystem1Main.bEnable := bGeneralEnable AND bCylinderEnable;
    fbCylinderSystem1Sub.bEnable  := bGeneralEnable AND bCylinderEnable;
    fbCylinderSystem2Main.bEnable := bGeneralEnable AND bCylinderEnable;
    fbCylinderSystem2Sub.bEnable  := bGeneralEnable AND bCylinderEnable;

    // Axes
    fbAxisSubfoil.bEnable := bGeneralEnable AND bAxisEnable AND NOT bStop;
    fbAxisMain.bEnable    := bGeneralEnable AND bAxisEnable AND NOT bStop;

//=======================================================
```

调用方法/函数/功能块

- 方法/函数/功能块的调用，最多 3 个参数可以单行编写。
- 多个单行调用可以被捆绑在 1 个文本块中。
- 如果程序元素有 3 个以上的参数，则它们代表单独的文本块，并使用空行与其他文本块隔开。在这里，第 1 个参数位于调用行或下一行。其他参数均以新行开始。
- 使用制表符键可以将参数开头缩进到同一级别。
- 分配运算符也从同一行级别开始。
- 分配运算符后跟 1 个空格。
- 如果在程序中调用实例的同一时刻将参数传递到功能块或读取参数，则在调用功能块时发生，而不是在实例调用之前或之后立即发生。
- 为了提高程序代码的识读率，建议对方法/函数/功能块进行限定调用。在这里，通过在调用时对参数名称进行显式命名可以实现赋值。

反面示例：

```
fbTimerStart.IN := TRUE;
fbTimerStart.PT := T#10S;
fbTimerStart();
bStartExecution := fbTimerStart.Q;
```

正面示例：

```
fbTimerStart( IN := TRUE,
              PT := T#10S,
              Q  => bStartExecution);
```

详细示例：

```
//=======================================================
// Stop

    // Cylinder
    fbCylinderPos1.Stop(bExecute := TRUE);
    fbCylinderPos2.Stop(bExecute := TRUE);
    fbCylinderPos3.Stop(bExecute := TRUE);

    // Axes
    fbAxis1.Stop(bExecute := TRUE, bDone => bAxis1Stopped);
    fbAxis2.Stop(bExecute := TRUE, bDone => bAxis2Stopped);

//=======================================================
```

语句

- RETURN、CONTINUE、EXIT、JMP、IF、CASE

- JMP 语句：可以且应该避免使用跳转语句，因为它们会降低代码的识读率。
- IF 语句：
  - IF 语句构成了 1 个单独的文本块，使用空行与其他块隔开。
  - 为了更好地理解并提高软件质量，应避免嵌套 IF 语句。相反，应该始终应用定义的状态，并使用 CASE 语句实现。
  - 所有 IF-ELSIF 语句均应包含 ELSE 分支。
    另请参见编程 [▶ 1012] 部分中的带有 ELSE 分支的 IF-ELSIF 语句 [▶ 1016]主题。
  - 语句缩进 1 级，因此，只有关键字 IF/ELSIF/ELSE/END_IF 在同一级别。关键字 THEN 没有自己的行。
  - 超过屏幕宽度的长条件将通过换行进行分割。在新行结束的文本部分将使用制表符键缩进到相对于第 1 行的适当级别 – 缩进到 IF 关键字的级别，或者对于嵌套条件，缩进到相应/等效子条件的级别，例如：

示例：

```
IF a > 10 THEN
   …
ELSIF a < 10 THEN
   …
ELSE
   …
END_IF

IF bCondition1 AND bCondition2 AND (bCondition3 OR bCondition4)
   AND bCondition5 AND bCondition6
   AND (bCondition7 OR bCondition8 OR bCondition9 OR bCondition10) THEN
   …
END_IF
```

- CASE 语句：
  - CASE 语句构成了 1 个单独的文本块，使用空行与其他块隔开。
  - 枚举值之间用空行隔开，并且相对于 CASE 语句缩进 1 级，因此，只有关键字 CASE/ELSE/END_CASE 在同一级别。
  - 此外，关于每个值块的注释也十分有用。
  - 枚举值的语句从该值下方 1-2 行开始（不直接位于值旁边），并且相对于该值缩进 1 级。

示例：

```
//=============================================================
// Cylinder sorting process

CASE eSortingState OF
    E_SortingState.DetectionOfBox:
        fbCylinder.MoveToBase();
        sVisuMessage := 'System in detection mode.';

        IF fbSensorDelay.bOut THEN
            eSortingState := eSorting_MoveCylToWorkPos;
        END_IF

    E_SortingState.MoveCylToWorkPos:
        fbCylinder.MoveToWork();

        IF fbCylinder.bAtWorkPos THEN
            eSortingState := eSorting_MoveCylToBasePos;
            sVisuMessage  := '';
        ELSE
            sVisuMessage  := 'Waiting for cylinder in work pos.';
        END_IF

    E_SortingState.MoveCylToBasePos:
        fbCylinder.MoveToBase();

        IF fbCylinder.bAtBasePos THEN
            eSortingState := eSorting_DetectionOfBox;
            sVisuMessage  := '';
        ELSE
            sVisuMessage  := 'Waiting for cylinder in base pos.';
        END_IF

ELSE
```

```
    sVisuMessage := 'System in non-existent state. Please check application.';
END_CASE

//=========================================================
```

循环（FOR、WHILE、REPEAT）

- 循环参数各在 1 行中（FOR ... DO、WHILE ... DO、REPEAT）。
- 语句缩进 1 级，因此，只有关键字 FOR/END_FOR、WHILE/END_WHILE 或 REPEAT/UNTIL/END_REPEAT 在同一级别。

示例：

```
//=========================================================
// Initialize storage areas with FOR loop

FOR nAreaCounter := cStorageStart TO cStorageEnd BY 1 DO
    aStorageAreas[nAreaCounter] := nAreaCounter;
END_FOR

//=========================================================
// Initialize delivery areas with WHILE loop

nAreaCounter := cDeliveryStart;

WHILE nAreaCounter <= cDeliveryEnd DO
    aDeliveryAreas[nAreaCounter] := nAreaCounter;
    nAreaCounter                 := nAreaCounter + 1;
END_WHILE

//=========================================================
```

18.2 命名规范

本部分 PLC 编程规范涵盖以下主题。

一般信息

1. 一般名称规范 [▶ 1003] [+]
2. 常见缩写 [▶ 1004] [+]

标识符

1. POU 和 DUT 的标识符 [▶ 1006] [+]
2. 变量和实例的标识符 [▶ 1007] [+]

全局变量列表和参数列表

1. 全局变量列表 [▶ 1010] [+]
2. 参数列表 [▶ 1011] [+]
3. 对 GVL 使用属性“qualified_only” [▶ 1011] [+]

示例

请参见：示例 [▶ 1011]

18.2.1 一般信息

主题：

1. 一般名称规范 [▶ 1003] [+]
2. 常见缩写 [▶ 1004] [+]

一般名称规范

- 名称有助于理解对象的目的。

- 避免多次使用相同的名称。另请参见 编程 [▶ 1012] 部分中的不得多次使用相同的名称 [▶ 1020]主题。
- 如果 1 个标识符由多个单词组成，则每个单词的第 1 个字母要大写（驼峰格式）。单词之间通常不使用分隔符，例如，“_”。如果在特殊情况下使用驼峰格式无法实现良好的识读率，建议使用下划线字符作为分隔符。有关在对象名称中使用下划线字符的更多例外情况，请参见观察有效字符 [▶ 994]主题。
- 标识符具有一致的前缀，这样便于识别对象类型。如果标识符需要前缀，请注意前缀区分大小写。另请参见：标识符 [▶ 1006]
- 标识符总是以字母开头。
- 缩写也采用驼峰格式书写。（独立的术语除外，例如，ID、CRLF、PC、PLC、…… 它们可以采用大写格式书写）

反面示例：

```
tADSTimeout     : TIME;
eCMDType        : E_CommandType;
```

正面示例：

```
nAddr           : UDINT;
nMsgCtrlMask    : DWORD;
cMaxCharacters  : UDINT;
tAdsTimeout     : TIME;
eCmdType        : E_CommandType;
stRemotePCInfo  : ST_RemotePCInfo;
```

常见缩写

您可以使用以下常见缩写。相应术语不得使用其他缩写。

该列表还包含不应缩写的术语。一般情况下，避免使用使术语缩短少于 3 个字母的缩写，因为这不会提高识读率。（缩写 Cnt 和 Idx 例外。）

缩写	术语
Abs	绝对
Ack	确认
Act	实际/活动（当前）
Addr	地址
/	报警
Auto	自动
Avg	平均
Bwd	向后
/	缓冲
Calc	计算方式/计算
Char	字符
Cmd	命令
Com	通信
Config	配置
Cnt	计数（数量）
Dst	目标
Diag	诊断
Diff	差异
Dim	尺寸
/	错误
/	执行
Fwd	前进
Hdl	处理
ID	标识符
Idx	索引
In	输入
Info	信息
Init	初始化
Itf	接口
Len	长度
Lib	库
max	最大
Mem	内存
min	最小
Msg	消息
Obj	对象
Op	操作
Out	输出
Ptr	指针
Pos	位置
Prev	上一个
Rcv	接收/已接收
Ref	引用
Src	来源
Srv	服务器
sync	同步
Temp	温度
Tmp	临时
/	超时

缩写	术语
Velo	速度
可视化	可视化
/	警告

18.2.2 标识符

主题：

1. POU 和 DUT 的标识符 [▶ 1006] [+]
2. 变量和实例的标识符 [▶ 1007] [+]

POU 和 DUT 的标识符

请参见：POU 和 DUT 的标识符 [▶ 1006]

变量和实例的标识符

请参见：变量和实例的标识符 [▶ 1007]

18.2.2.1 POU 和 DUT 的标识符

在命名 POU 和用户定义的数据单元类型（DUT）时，您应该考虑以下几点。

- 对象的前缀始终大写。
- 下划线字符“_”可用作前缀和实际对象名称之间的分隔符。
- 实际对象名称始终以大写字母开头。
- 如果直接嵌入 TcCOM 对象的接口，则只使用前缀“I”，例如，ITcUnknown。

静态分析：

另请注意使用 TE1200 PLC 静态分析检查编程规范 [▶ 992]选项。

前缀：

对象	前缀	描述	示例	静态分析 命名规范 ID
FUNCTION_BLOCK	FB_	功能块	FB_WritePersistentData	NC0103
ACTION		（功能块或程序的）动作	MoveAbsolute	NC0106
METHOD		（功能块或接口的）方法	Reset	NC0105
PROPERTY	根据返回类型（请参见变量和实例的标识符 [▶ 1007]）	（功能块、程序或接口的）属性	nErrorID fMotorTempC	NC0107 另请参见：
PROGRAM		程序	ModuleControl	NC0102
FUNCTION	F_	功能	F_MeterToInch	NC0104
STRUCT	ST_	结构	ST_BufferEntry	NC0151
ENUM	E_	枚举类型	E_MachineState E_Quality	NC0152
类型	T_	别名类型	T_Nibble	NC0154
UNION	U_	联合	U_Control	NC0153
INTERFACE	I_	接口	I_Cylinder	NC0108
GVL	GVL 作为名称或 GVL_ 作为前缀	全局变量列表	GVL GVL_Axis GVL_Subsystem	
	GCL	全局常量列表	GCL	
Param	Param 作为名称或 Param_ 作为前缀	全局参数列表	Param Param_Subsystem	

如果属性 <name> 直接由功能块的变量表示，则该变量被称为 _<name>。

枚举：
在定义枚举时，可使用属性 {attribute 'qualified_only'}，这样可以简化枚举的使用，同时也不必缩写枚举类型。另请参见 编程 [▶ 1012] 部分中的对枚举使用属性“qualified_only”和“strict” [▶ 1024]主题。

```
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE E_SignalStates :
(
     Red    := 0,
     Yellow,
     Green
);
END_TYPE
```

18.2.2.2 变量和实例的标识符

在命名变量和实例时，请考虑以下几点：

- 变量名和实例名的前缀用小写字母书写。
- 前缀后的第 1 个字符大写（从静态分析中启用命名规范的选项**前缀后的第 1 个字符应为大写字母**，请参见以下关于静态分析的说明）。

静态分析：

另请注意使用 TE1200 PLC 静态分析检查编程规范 [▶ 992]选项。

前缀：

1）基本数据类型和动态数据类型的变量：

类型	前缀	描述	声明示例	调用示例	静态分析 命名规范 ID
SINT、USINT、 INT、UINT、 DINT、UDINT、 LINT、ULINT、 BYTE、WORD、 DWORD、LWORD、 XINT、UXINT、 XWORD	n	整数、 位数	nErrorID : UDINT; nSize : UINT; nMask : WORD; nMessages : UINT;	nErrorID := 16#745; nSize := SIZEOF(ST_BufferEntry); nMask := nMask & 16#0FF0; nMessages := 3;	NC0005- NC0016、 NC0035、 NC0037、 NC0038、 NC0160
BOOL、BIT	b	boolean（位）	bConfigured bReset		NC0003、 NC0004
REAL、LREAL	f	浮点数	fPosition		NC0017、 NC0018
STRING	s	字符串	sNetID		NC0019
WSTRING	ws	宽字符串 （unicode）	wsRouteName		NC0020
TIME、LTIME	t	时间	tDelay		NC0021、 NC0022
TIME_OF_DAY	td	当日时间	tdClockTime		NC0025
DATE	d	日期	dProductionStart		NC0023
DATE_AND_TIME	dt	日期和时间	dtProductionStart		NC0024
ARRAY[...] OF ...	a	数组	aMessages		NC0030
POINTER TO ...	p	指针	pData pNextProduct	pData := ADR(aBuffer);	NC0026
POINTER TO POINTER TO ...	pp	嵌套指针	ppData		
POINTER TO INTERFACE	pip	指向接口的指针	pipCylinder		
REFERENCE TO ... 1）作为方法输入 2）在其他情况下	1）根据引用的基本数据类型 2）引用	引用	1）nData（带有 REFERENCE TO INT） 2）refSize	1）SampleMethod(nData := <…>) 2）refSize REF= nSize;	NC0027

- **指针：**
  - 出于安全考虑，类型指针优于非类型指针。
  - 指针应声明为 POINTER TO <data type>，其中 <data type> 与指针所指向的变量的数据类型一致。
  - 指向数组的指针应声明为 POINTER TO ARRAY [...] OF <data type>，其中 <data type> 与指针所指向的数组的数据类型一致。
  - 如果指针所指向的变量的数据类型未知，因此无法类型化，则应声明为 POINTER TO BYTE（或 PVOID）。由于 64 位兼容性的原因，数据类型 DWORD 不能用作指针。
  - 内容运算符“^”用于解除引用指针。
  - 应特别注意指针所指向地址的有效性。

示例 1：

```
nSample       : INT;
pSampleToInt  : POINTER TO INT;
```

```
pSampleToInt  := ADR(nSample)
pSampleToInt^ := 123;              // Result: nSample = 123
```

示例 2：

```
aSample                    : ARRAY[1..3] OF LREAL;
pSampleToArrayOfLreal      : POINTER TO ARRAY[1..3] OF LREAL;
```

```
pSampleToArrayOfLreal     := ADR(aSample);
pSampleToArrayOfLreal^[2] := 4.56; // Result: aSample[2] = 4.56
```

- **指针运算/指针比较：**
  - 如果可能的话，出于安全考虑，应避免在指针上进行指针运算和使用比较运算符（例如，[pSampleToLreal: = pSampleToLreal + SIZEOF(LREAL)] 等递增运算或使用 >、>=、<、<= 进行指针比较）。
  - 如果您离不开这些操作，则最多只能在数组中使用它们。
  - 尤其重要的是，要确保指针指向同一个数组的元素，并且遵守数组的地址范围。

2）对象实例和用户定义的数据类型：

在 IEC61131 标准中定义且具有基本重要性的功能块实例（R_TRIG、TON.....）没有其他前缀。校正命名的示例：

```
fbAdsTimer : TON;
```

类型	前缀	描述	声明示例	调用示例	静态分析 命名规范 ID
FUNCTION_BLOCK	fb	功能块的实例	fbWritePersData : FB_WritePersistentData;	fbWritePersData();	NC0031
STRUCT	st	结构的实例	stBufferEntry : ST_BufferEntry;	stBufferEntry.nCounter := 5;	NC0032
ENUM	e	枚举的 实例	eMachineState : E_MachineState; eQuality : E_Quality;	eMachineState := E_MachineState.Stop; eQuality := E_Quality.Good;	NC0029
类型（别名）	根据内部数据类型	别名类型的实例	nNibble : T_Nibble; sNetId : T_AmsNetId;	nNibble := 16#1; sNetId := '1.2.3.4.5.6';	NC0033 另请参见：
INTERFACE	ip	接口的实例	ipCylinder : I_Cylinder;	ipCylinder := fbCylinderBase;	NC0036
UNION	u	联合的实例	uCtrl : U_Control;	uCtrl.aRegister[1] := 16#02;	NC0034

3）其他：

类型	前缀	描述	声明示例	静态分析 说明
嵌套类型	仅根据外部类型 例外 1：“pp”表示 POINTER TO POINTER TO 例外 2：“pip”表示 POINTER TO INTERFACE		pTelegramData : POINTER TO ARRAY[0..7] OF BYTE;	禁用选项**可组合数据类型的递归前缀** 另请参见：

局部和全局常量	c		VAR CONSTANT
			cSyncID : UINT := 16#80;
			END_VAR
			VAR_GLOBAL CONSTANT
			cLowerThreshold : UDINT := 9;
			cOptionMove : DWORD := 16#04;
			END_VAR
				NC0062、NC0070

全局变量		同样的命名规则也适用于全局变量。（不要为变量名添加第 2 个前缀。）	VAR_GLOBAL nProducedUnits : UINT; END_VAR	禁用选项**将范围前缀与数据类型前缀组合在一起** 另请参见：
HRESULT	hr		hrErrorCode : HRESULT;	NC0160
PVOID	p	指针	pData : PVOID;	NC0160
GUID		没有为 GUID 分配前缀。		
AT%I*	数据类型前缀前可选：I	分配的输入	fActPos AT%I*: LREAL; oder: IfActPos AT%I*: LREAL;	
AT%Q*	数据类型前缀前可选：O	分配的输出	fSetPos AT%Q*: LREAL; oder: OfSetPos AT%Q*: LREAL;	

分配的输入或输出可以在数据类型前缀前附加可选的前缀“I”或“O”。在实例中，它代表唯一大写的前缀。在决定是否使用该可选前缀时，应保持一致性，以便在整个项目中统一所分配变量的名称。

18.2.3 全局变量列表和参数列表

主题：

1. 全局变量列表 [▶ 1010] [+]
2. 参数列表 [▶ 1011] [+]
3. 对 GVL 使用属性“qualified_only” [▶ 1011] [+]

全局变量列表

为全局变量或常量列表分配名称“GVL”或前缀“GVL_”，然后在名称中加上对 GVL 的描述。

示例：

- GVL
- GVL_Axis
- GVL_Subsystems

参数列表

为全局参数列表分配名称“Param”或前缀“Param_”，然后在名称中加上对参数列表的描述。

示例：

- Param
- Param_Subsystems

对 GVL 使用属性“qualified_only”

在定义全局变量列表 [▶ 66]或参数列表 [▶ 67]时使用属性 {attribute 'qualified_only'} [▶ 761]，在使用变量时强制使用 GVL 命名空间。通过使用命名空间（例如：GVL_Ctrl.bPaintingActive），变量的全局范围变得一目了然。

正面示例：

全局变量列表“GVL_Ctrl”：

```
{attribute 'qualified_only'}
VAR_GLOBAL
    bPaintingActive  : BOOL;
END_VAR
```

另请参见 编程 [▶ 1012] 部分中的对 GVL 使用属性“qualified_only” [▶ 1011]主题。

18.2.4 示例

```
VAR_GLOBAL
    nSocketPort     : INT;
    sNetID          : T_AmsNetId;
END_VAR
```

```
VAR_GLOBAL CONSTANT
    cSocketAddr     : INT           := 1;
    cLocalNetID     : T_AmsNetId    := '';
END_VAR
```

```
FUNCTION F_CompareVelocity : UINT
```

```
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE E_StandardColor :
(
    Blue    := 0,
    Green   := 1
);
END_TYPE
```

```
FUNCTION_BLOCK FB_WirelessCom
VAR CONSTANT
    cFrequency      : REAL := 868.0;
END_VAR
VAR
    eColor          : E_StandardColor;
    aColors         : ARRAY[1..cMaxColors] OF E_StandardColor;
END_VAR
----------------------------------------------------------------------
IF (sNetId = cLocalNetID) THEN
    eColor      := E_StandardColor.Green;
    aColors[1] := eColor;
END_IF
```

```
FUNCTION_BLOCK FB_ModuleControl
VAR_IN_OUT
    aTerminalData   : ARRAY[1..cOversamples] OF INT;
END_VAR
VAR
    fbCalc          : FB_Calculate;
END_VAR
----------------------------------------------------------------------
fbCalc.Reset();
fbCalc( pDataIn     := ADR(aTerminalData),
        nDataInSize := SIZEOF(aTerminalData) );
```

18.3 编程

本部分 PLC 编程规范涵盖以下主题。

另请注意使用 TE1200 PLC 静态分析检查编程规范 [▶ 992]选项。

一般信息

循环和条件

1. 仅带有枚举实例的 CASE 语句 [▶ 1014] [++]
2. 将循环的限值声明为常量 [▶ 1015] [+]
3. 处理 CASE 语句中枚举变量的所有枚举值 [▶ 1016] [+]
4. 带有 ELSE 分支的 IF-ELSIF 语句 [▶ 1016] [+]
5. IF 条件的顺序 [▶ 1017] [+]

错误代码

1. 错误代码的数据类型 [▶ 1017] [++]
2. 函数和方法的错误信息 [▶ 1018] [+]
3. 功能块的错误信息 [▶ 1018] [+]

识读率、可维护性

1. 没有未使用的声明/对象或无用代码 [▶ 1019] [++]
2. 没有“魔术数字” [▶ 1020] [+]
3. 不得多次使用相同的名称 [▶ 1020] [+]
4. 声明和实现中的相同符号 [▶ 1020] [+]
5. 避免空白语句（;） [▶ 1021] [+]
6. 每个赋值在单独的代码行中 [▶ 1021] [+]

库

库开发

1. 库名称 [▶ 1022] [++]
2. 版本控制 [▶ 1022] [++]
3. 内部类型定义的封装 [▶ 1022] [+]
4. 库中的标识符 [▶ 1023] [+]

库的使用

1. 库的使用 [▶ 1023] [++]
2. 版本检查 [▶ 1023] [++]
3. 固定库版本 [▶ 1023] [+]

DUT

DUT 的实现

1. 对枚举使用属性“qualified_only”和“strict” [▶ 1024] [++]
2. 不初始化任何值，只初始化枚举的第 1 个值或所有值 [▶ 1024] [+]

DUT 的使用

1. 仅将枚举变量分配给相关的枚举符 [▶ 1025] [++]

POU

函数、方法和动作的实现

1. 函数/方法中没有大参数的“按值调用” [▶ 1026] [++]

2. 不要在函数/方法中声明大变量 [▶ 1027] [++]
3. 不要使用动作 [▶ 1027] [+]
4. 在内部使用函数/方法的所有参数 [▶ 1027] [+]
5. 仅在 1 处分配函数/方法的返回值 [▶ 1028] [+]
6. 尽可能限制对方法的访问 [▶ 1029] [+]
7. 将参数分组为结构 [▶ 1029] [+]

函数、方法的使用

1. 评估 POU 返回的错误信息 [▶ 1029] [++]
2. 使用函数/方法的返回值 [▶ 1030] [+]
3. 不要在自身内部调用函数/方法 [▶ 1031] [+]

功能块的实现

1. 确保在线更改能力 [▶ 1032] [++]
2. 具有一次性异步处理功能的统一接口 [▶ 1032] [+]
3. 具有连续异步处理功能的统一接口 [▶ 1032] [+]
4. 在内部使用功能块的所有参数 [▶ 1033] [+]
5. 将参数分组为结构 [▶ 1033] [+]

功能块的使用

1. 不要将功能块实例声明为 VAR PERSISTENT [▶ 1033] [++]
2. 评估 POU 返回的错误信息 [▶ 1029] [++]
3. 不要从外部访问 POU 的局部变量 [▶ 1034] [++]
4. 不得直接分配对象 [▶ 1035] [++]
5. 没有临时功能块实例 [▶ 1035] [+]

变量

一般信息

1. 不要测试浮点数是否相等或不等 [▶ 1036] [++]
2. 使用显式转换避免不必要的结果 [▶ 1037] [+]

变量封装

1. 将不变的变量声明为 VAR CONSTANT [▶ 1037] [+]
2. 不要遮蔽全局标识符 [▶ 1038] [+]
3. 限制 ADS 访问 [▶ 1038] [+]

数组

1. 通过常量定义数组限值 [▶ 1039] [++]
2. 数组下限值为 1 [▶ 1039] [+]

指针、引用、接口

1. 指向临时存在的对象的指针/引用/接口的临时存在 [▶ 1040] [++]
2. 每个周期重置指针/引用 [▶ 1040] [++]
3. 在每次使用前检查指针/引用/接口 [▶ 1041] [++]
4. 优先选择引用，而不是指针 [▶ 1042] [++]

分配的变量

1. 不要使用直接寻址 [▶ 1042] [++]
2. 避免对输出进行多次写入访问 [▶ 1043] [++]
3. 避免地址变量的内存区重叠 [▶ 1043] [+]

全局变量

1. 对 GVL 使用属性“qualified only” [▶ 1044] [+]
2. 明智地使用全局变量 [▶ 1044] [+]

字符串

1. 大小和长度规格的标识符 [▶ 1045] [++]
2. 建议的默认大小 [▶ 1045] [+]
3. 大字符串的传输 [▶ 1045] [+]
4. 大字符串的处理 [▶ 1046] [+]

运行时行为

一般信息

1. 截距除以零。 [▶ 1046][++]

动态内存

1. 仔细地执行动态内存分配 [▶ 1047] [++]

多个任务

1. 避免对内存区进行并发访问 [▶ 1048] [++]
2. 避免对功能块进行并发访问 [▶ 1049] [++]
3. 明智地使用全局变量 [▶ 1044] [+]

18.3.1 一般信息

18.3.1.1 循环和条件

主题：

1. 仅带有枚举实例的 CASE 语句 [▶ 1014] [++]
2. 将循环的限值声明为常量 [▶ 1015] [+]
3. 处理 CASE 语句中枚举变量的所有枚举值 [▶ 1016] [+]
4. 带有 ELSE 分支的 IF-ELSIF 语句 [▶ 1016] [+]
5. IF 条件的顺序 [▶ 1017] [+]

仅带有枚举实例的 CASE 语句

在 CASE 语句中，应使用包含相应枚举符的枚举实例来定义状态机，而不是使用“魔术数字”。这样，状态机的值可以自我说明，而不会使用“魔术数字”。

另请注意以下有关编程规范的主题：

- 对枚举使用属性“qualified only”和“strict” [▶ 1024]
- 没有“魔术数字” [▶ 1020]

反面示例：

```
PROGRAM Sample_neg
VAR
    nState  : INT; // Used to operate the sample state machine
END_VAR
```

```
// NON COMPLIANT: Integer variable and "magic values" are used to define state machine
CASE nState OF
    0: F_DoSomethingUsefulHere();
    1: F_DoSomethingUsefulHere();
    2: F_DoSomethingUsefulHere();
ELSE
    F_DoSomethingUsefulHere();
END_CASE
```

正面示例：

```
// Enumeration for the positive sample
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE E_SampleState :
(
    Entry,  // Entrance part of a state
    Work,   // Action part of a state
    Finish  // Exit part of a state
);
END_TYPE
```

```
PROGRAM Sample_pos
VAR
    eState  : E_SampleState;  // Used to operate the sample state machine
END_VAR
```

```
// COMPLIANT: Enum instance and enum values are used to define state machine
CASE eState OF
    E_SampleState.Entry:
        F_DoSomethingUsefulHere();
    E_SampleState.Work:
        F_DoSomethingUsefulHere();
    E_SampleState.Finish:
        F_DoSomethingUsefulHere();
END_CASE
```

将循环的限值声明为常量

常量变量应该用作循环的限值。如果在循环内访问数组，则应使用相同的常量变量声明数组。

- 数组的下限值 = 循环的下限值 = 常量变量 1
- 数组的上限值 = 循环的上限值 = 常量变量 2

另请注意以下建议：

- 通过常量定义数组限值 [▶ 1039]
- 数组下限值为 1 [▶ 1039]

静态分析：

按主题建议的静态分析规则：

- SA0072：计数器变量的无效使用
- SA0080：数组索引的循环索引变量超出数组范围
- SA0081：上限值不是常量

以下示例的一般程序元素：

```
TYPE ST_Object :
STRUCT
    sName        : STRING;
    nID          : UINT;
END_STRUCT
END_TYPE
```

```
PROGRAM Sample
VAR CONSTANT
    cMin         : UINT := 1;
    cMax         : UINT := 10;
END_VAR
VAR
    aObjects     : ARRAY[cMin..cMax] OF ST_Object;
    bInitDone    : BOOL;
    nForIdx      : UINT;
END_VAR
```

反面示例：

```
VAR
    nUpperBorder : UINT;
END_VAR
```

```
// Initialize objects
IF NOT bInitDone THEN
    nUpperBorder := cMin;

    FOR nForIdx := nUpperBorder TO 10 BY 1 DO
        aObjects[nForIdx].sName := CONCAT('Object_', UDINT_TO_STRING(nForIdx));
```

```
        aObjects[nForIdx].nID   := nForIdx;
    END_FOR

    bInitDone := TRUE;
END_IF
```

正面示例：

```
// Initialize objects
IF NOT bInitDone THEN
    FOR nForIdx := cMin TO cMax BY 1 DO
        aObjects[nForIdx].sName := CONCAT('Object_', UDINT_TO_STRING(nForIdx));
        aObjects[nForIdx].nID   := nForIdx;
    END_FOR

    bInitDone := TRUE;
END_IF
```

处理 CASE 语句中枚举变量的所有枚举值

在 CASE 语句中，您应该处理枚举变量的所有枚举值。如果这对于大量相同或未使用的枚举值没有意义，则可以对这些情况使用 ELSE 分支。

静态分析：

使用以下静态分析规则进行验证：

- SA0075：缺少 ELSE
- SA0076：缺少枚举常量

以下示例的一般程序元素：

```
// Enumeration for all samples in this rule
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE E_ColorTrafficLight :
(
    Red      := 0,
    Yellow,
    Green
);
END_TYPE
```

```
PROGRAM Sample
VAR
    eColorTrafficLight : E_ColorTrafficLight; // Used to handle the state of the traffic light
END_VAR
```

反面示例：

```
CASE eColorTrafficLight OF      // NON COMPLIANT: enum value Green is not handled
    E_ColorTrafficLight.Red:
        SetLightRed();
    E_ColorTrafficLight.Yellow:
        SetLightYellow();
END_CASE
```

正面示例：

```
CASE eColorTrafficLight OF      // COMPLIANT: all enum values are handled
    E_ColorTrafficLight.Red:
        SetLightRed();
    E_ColorTrafficLight.Yellow:
        SetLightYellow();
    E_ColorTrafficLight.Green:
        SetLightGreen();
END_CASE
```

带有 ELSE 分支的 IF-ELSIF 语句

出于安全考虑，如果 ELSIF 可用，建议在 IF 语句中添加 ELSE。在 ELSIF 语句后应加上 ELSE，以表明已考虑到所有可能的情况。如果没有为 ELSE 分支规划任何语句，则应在 ELSE 之后加上分号。在程序的验收/调试期间，可以使用注释来解释已考虑过 ELSE 情况，以及为什么没有针对它的语句。

以下示例的一般程序元素：

```
PROGRAM Sample
VAR
    nTest   : INT:= 10;         // Test value for sample
END_VAR
```

反面示例：

```
IF nTest < 10 THEN              // NON COMPLIANT: the ELSE should be used in any way
    nTest := 20;
ELSIF nTest > 20 THEN
    nTest := 10;
END_IF
```

正面示例 1：

```
IF nTest < 10 THEN              // COMPLIANT with any action in ELSE
    nTest := 20;
ELSIF nTest > 20 THEN
    nTest := 10;
ELSE
    F_DoSomethingUsefulHere(); // it's just a sample
END_IF
```

正面示例 2：

```
IF nTest < 10 THEN              // COMPLIANT with a comment in ELSE
    nTest := 20;
ELSIF nTest > 20 THEN
    nTest := 10;
ELSE
    ;(* No action needed for the case that nTest is >= 10 and <= 20 *)
END_IF
```

IF 条件的顺序

根据发生概率排列 IF/ELSIF/ELSE 条件的顺序。因此，应首先处理最有可能发生的情况。这不仅可以提高识读率，而且，在某些情况下还可以提高性能，因为降低了非匹配查询的处理频率。

正面示例：

```
IF SUCCEEDED(hrErrorCode) THEN
    IF nReqIdx < cIdxMax THEN    // requested index in range
        ; // handle request
    ELSIF nReqIdx = cIdxMax THEN // requested index in range but at upper limit
        ; // handle request
    ELSE
        ; // add error output (error caused by invalid index)
    END_IF
ELSE
    ;     // add error handling (error in previous step)
END_IF
```

18.3.1.2 错误代码

主题：

1. 错误代码的数据类型 [▶ 1017] [++]
2. 函数和方法的错误信息 [▶ 1018] [+]
3. 功能块的错误信息 [▶ 1018] [+]

错误代码的数据类型

错误代码应进行指定，并以 hrErrorCode（HRESULT 类型）的形式传输。HRESULT 类型的特点是以负值表示错误。通过正值可以输出警告或信息。

或者，您也可以指定 nErrorID（UDINT 类型）。

声明	错误范围	无错误	消息/信息	检查函数

hrErrorCode : HRESULT;	<0	>=0	>0	IF SUCCEEDED(hrErrorCode) THEN
				...
				END_IF
				IF FAILED(hrErrorCode) THEN
				...
				END_IF

nErrorID : UDINT;	>0	0		IF nErrorID = 0 THEN
				...
				END_IF
				IF nErrorID <> 0 THEN
				...
				END_IF

函数和方法的错误信息

函数和方法不需要错误信息，前提是没有发生错误。示例：c := AddTwoValues(a, b);

在特殊情况下，函数和方法可以仅使用 Boolean 值输出错误情况，前提是错误原因只能有 1 个。

在大多数情况下，函数和方法通过返回值中的错误代码（HRESULT）来指示错误。功能块通常会输出最后调用的方法的错误。另请参见主题功能块的错误信息 [▶ 1018]。

功能块的错误信息

功能块不需要错误信息，前提是没有发生错误。

建议按以下方式提供功能块的错误信息。

对于出错原因较少的小型功能块，以下输出已足够：

```
VAR_OUTPUT
    bError          : BOOL;         // TRUE if an error occurred.
    hrErrorCode     : HRESULT;      // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

对于可能有许多不同的错误原因和不同的错误源的更复杂的功能块，建议进行进一步输出：

```
VAR_OUTPUT
    bError          : BOOL;         // TRUE if an error occurred.
    hrErrorCode     : HRESULT;      // '< 0' = error; '> 0' = info; '0' = no error/info
    ipErrorMessage  : I_TcMessage :=
fbErrorMessage; // shows detailed information about occurred errors, warnings and more
END_VAR
VAR
    {attribute 'conditionalshow'}
    fbErrorMessage  : FB_TcMessage;
END_VAR
```

bError 能够以尽可能简单的方式查询是否存在错误情况。此外，如果错误在多个任务周期内没有解决，您可以在 PLC 编辑器的在线视图中轻松查看该错误。

hrErrorCode 指定了错误代码，可以很容易地将其传递出去。在 PLC 编辑器的在线视图中会显示十六进制值。

ipErrorMessage（或 ipResultMessage）会显示所发生错误的详细信息。在 PLC 编辑器的在线视图中还会显示与语言相关的纯文本错误描述等内容。此外，还可以将该消息传输到 TwinCAT 3 EventLogger。使用事件列表中的自定义事件类可以定义自定义错误代码的自定义文本。有关更多信息，请参见 PLC 库 Tc3_EventLogger 的文档和相关示例。在功能块的输出端，信息可作为接口提供，以限制对相关内容的访问。不过，建议在内部确保接口指针始终有效。

有关错误代码的主题，另请参见以下关于编程规范的主题：

- 评估 POU 返回的错误信息 [▶ 1029]

18.3.1.3 识读率、可维护性

主题：

1. 没有未使用的声明/对象或无用代码 [▶ 1019] [++]
2. 没有“魔术数字” [▶ 1020] [+]
3. 不得多次使用相同的名称 [▶ 1020] [+]
4. 声明和实现中的相同符号 [▶ 1020] [+]
5. 避免空白语句（;） [▶ 1021] [+]
6. 每个赋值在单独的代码行中 [▶ 1021] [+]

没有未使用的声明/对象或无用代码

1 个项目不应包含不可执行的路径、“死”（不必要）代码、未使用的类型声明或变量。

项目中未使用的程序元素很快就会导致代码结构混乱。在维护和扩展的过程中，如果项目只包含实际使用的程序元素，则可大大提高代码的识读率。

静态分析：

使用以下静态分析规则进行验证：

- SA0001：不可达代码
- SA0002：空对象
- SA0031：未使用的签名
- SA0032：未使用的枚举常量
- SA0033：未使用的变量
- SA0035：未使用的输入变量
- SA0036：未使用的输出变量

免授权版本轻量静态分析 [▶ 139]也包含规则 SA0033。

另请注意，对于不应再报告特定规则/命名规范的已检查位置，通过编译指示或属性可以在本地禁用静态分析规则和命名规范（另请参见：TE1200 PLC 静态分析中的编译指示和属性章节）。理想情况下，您应该对本地停用情况进行注释，并做出适当解释。

例如，您可以通过 {analysis -2} 在本地为该 FB 主体禁用功能块的预期空主体的规则 SA0002。

没有不可执行的路径

反面示例：

```
IF FALSE THEN                    // NON COMPLIANT
    F_DoSomethingUsefulHere();   // will never be executed
END_IF
```

正面示例：

```
IF bTest THEN                    // COMPLIANT
    F_DoSomethingUsefulHere();   // it's just a sample
END_IF
```

没有“死”（不必要）代码

反面示例：

```
FUNCTION F_Sample_neg : INT
VAR_INPUT
    nA       : INT;              // Variable a in term y = a*a + b
    nB       : INT;              // Variable b in term y = a*a + b
END_VAR
VAR
    nTemp    : INT;              // Used for temporary calculation of a*a
END_VAR
```

```
nTemp        := nA * nA;         // NON COMPLIANT: nTemp will not be used later
F_Sample_neg := nA * nA + nB;
```

正面示例：

```
FUNCTION F_Sample_pos : INT
VAR_INPUT
    nA       : INT;             // Variable a in term y = a*a + b
    nB       : INT;             // Variable b in term y = a*a + b
END_VAR
```

```
F_Sample_pos := nA * nA + nB;   // COMPLIANT: no dead code in this sample
```

没有“魔术数字”

不要使用“魔术数字”。

另外，常量或其他固定值也可用于比较或赋值。例如，诸如输出或比较文本等文本应该存储在常量中，而不是直接在源代码中进行定义。

另请注意以下有关编程规范的主题：

- 仅带有枚举实例的 CASE 语句 [▶ 1014]

反面示例：

```
PROGRAM Sample_neg
VAR
    nValue   : INT;   // Sample INT-variable that is used for comparison
    bValueOk : BOOL;  // Indicates if the value of sample INT-variable is OK
END_VAR
```

```
// NON COMPLIANT: A "magic value" is used for comparison
bValueOk := (nValue = 125);
```

正面示例：

```
PROGRAM Sample_pos
VAR
    nValue    : INT;            // Sample INT-variable that is used for comparison
    bValueOk  : BOOL;           // Indicates if the value of sample INT-variable is OK
END_VAR
VAR CONSTANT
    cValueOk  : INT   := 125;   // Used to validate the sample INT-variable
END_VAR
```

```
// COMPLIANT: A constant variable is used for comparison
bValueOk := (nValue = cValueOk);
```

不得多次使用相同的名称

避免多次使用相同的名称。这包括以下情况：

- 枚举常量的名称与变量的名称的比较
- 对象之间的名称
- 对象的名称与变量的名称的比较

静态分析：

使用以下静态分析规则进行验证：

- SA0013：具有相同变量名的声明
- SA0027：多次使用名称

免授权版本 轻量静态分析 [▶ 139] 也包含规则 SA0027。

声明和实现中的相同符号

出于统一性、识读率和可维护性的考虑，您应该在声明和实现中使用相同的程序元素和变量的符号。

静态分析：

借助静态分析规则进行检查：

- SA0029：实现中的符号与声明中的符号不同

以下示例的一般程序元素：

```
FUNCTION F_Sample
```

反面示例：

```
PROGRAM Sample
VAR
    bTest  : BOOL;
END_VAR
-------------------------
IF btest THEN
    f_Sample();
END_IF
```

正面示例：

```
PROGRAM Sample
VAR
    bTest  : BOOL;
END_VAR
-------------------------
IF bTest THEN
    F_Sample();
END_IF
```

避免空白语句（;）

避免空白语句（;），以免不必要地扩展代码。如果空白语句对于澄清特定情况而言绝对必要，则空白语句应在单独 1 行中。此外，您还应注释说明该程序分支为何为空。

静态分析：

借助静态分析规则进行检查：

- SA0003：空白语句

以下示例的一般程序元素：

```
PROGRAM Sample
VAR
    nTest : INT := 10;  // Test value used in sample
END_VAR
```

反面示例：

```
IF nTest = 10 THEN
    ;                   // NON COMPLIANT without a useful comment
ELSE
    nTest := 10;
END_IF
```

正面示例：

```
IF nTest = 10 THEN
    ;                   // COMPLIANT with a comment: In case that nTest is equal to 10, no action is
 needed. This status is intended.
ELSE
    nTest := 10;
END_IF
```

每个赋值在单独的代码行中

每个赋值都应该在单独的代码行中。因此，赋值不应包含在条件中，分配运算符也不应包含子表达式。

静态分析：

借助静态分析规则进行检查：

- SA0095：条件分配

以下示例的一般程序元素：

```
PROGRAM Sample
VAR
    bVar1 : BOOL;
    bVar2 : BOOL;
    nA    : INT;
    nB    : INT;
    nC    : INT;
END_VAR
```

反面示例：

```
// NON COMPLIANT: assignment in condition
IF bVar1 := bVar2 THEN
    DoSomething();
END_IF

// NON COMPLIANT: more than one assignment in a single line
nA := nC * (nB := nC + nC);
```

正面示例：

```
IF bVar1 = bVar2 THEN
    DoSomething();
END_IF

nB := nC + nC;
nA := nC * nB;
```

18.3.2 库

18.3.2.1 库开发

主题：

1. 库名称 [▶ 1022] [++]
2. 版本控制 [▶ 1022] [++]
3. 内部类型定义的封装 [▶ 1022] [+]
4. 库中的标识符 [▶ 1023] [+]

库名称

倍福 PLC 库以前缀 **Tc** 开头（例如：**Tc3_EventLogger**），文件名扩展名为 ***.compiled-library**，并包含在库存储库中。

选择简短而明确的库名称。

倍福 PLC 库的命名空间与库的名称相匹配。

倍福 PLC 库的默认占位符与库的名称相匹配。

版本控制

每个 PLC 库都有 1 个 4 位数的版本号（Major.Minor.Build.Revision）。

主要	如果新版本与旧版本不兼容，则第 1 个数字会递增。 因此，次要版本和构建版本的数字会被重置为 1。
次要	如果新版本包含功能扩展，则第 2 个数字会递增。 在这种情况下，构建版本的数字会被重置为 1。
构建（补丁）	如果新版本只包含错误修复，则第 3 个数字会递增。
修订	如果是发布版本，则第 4 个数字总是 0。

如果 PLC 库仍然处于测试状态，则主要版本为 0 或版本号为 0.x.y.z。发布版本的版本号至少为 1.1.1.0。

如果其他程序部分提供了 1 个 3 位数版本（Major.Minor.Patch），则可相应地使用前 3 位数（构建 = 补丁）。

内部类型定义的封装

为了防止使用库外的内部类型定义，并为用户突出显示与其相关的 POU 和 DUT，在 PLC 库中使用了封装。为此，建议使用访问说明符 INTERNAL。

PRIVATE 访问说明符也可用于功能块的个别方法。如要隐藏功能块的局部变量，可使用 属性 “conditionalshow all locals” [▶ 738]。

库中的标识符

为了防止编译器因标识符重复而出错，并允许在没有强制命名空间的情况下使用库，建议在标识符中使用额外的缩写。该库名称缩写位于前缀和相应类型的实际名称之间。

同样，库的全局变量列表也应加上缩写，以避免出现多个名为“GVL”的列表。

示例（来自 PLC 库 Tc3_Filter）：

```
STRUCT ST_FTR_PT1
ENUM E_FTR_Type
FUNCTION_BLOCK FB_FTR_PT1
VAR_GLOBAL GVL_FTR
```

另请参见：

- 库的使用

18.3.2.2 使用库

主题：

1. 库的使用 [▶ 1023] [++]
2. 版本检查 [▶ 1023] [++]
3. 固定库版本 [▶ 1023] [+]

库的使用

PLC 库作为占位符被引用。

有关详细信息，请参见 TwinCAT 3 PLC 文档中的库占位符 [▶ 257]章节。

版本检查

PLC 库包含 1 个 ST_LibVersion [▶ 730] 类型的全局常量，您可以在其中找到有关库版本的信息。在运行时可以读取该常量，通过函数 F_CmpLibVersion 可以将其与所需的库版本进行比较（请参见 Tc2_System 库）。

固定库版本

在附加引用作为库占位符时，库将附带版本“* / 最新版本”。

不过，一旦在目标系统上创建了存档（例如，PLC 存档或 TwinCAT 存档）或激活了 TwinCAT 配置，就应立即固定库版本。最简单的方法是使用引用节点的上下文菜单中的“Set to Effective Version”（设置为有效版本）命令。

另请注意有关主题 项目交付 [▶ 36] 的更多信息。

另请参见：

- 库的使用

18.3.3 DUT

18.3.3.1 DUT 的实现

主题：

1. 对枚举使用属性“qualified only”和“strict” [▶ 1024] [++]
2. 不初始化任何值，只初始化枚举的第 1 个值或所有值 [▶ 1024] [+]

对枚举使用属性“qualified_only”和“strict”

在定义枚举时，可使用属性 {attribute 'qualified_only'} [▶ 761] 和 {attribute 'strict'} [▶ 762]。

静态分析：

使用以下静态分析规则进行验证：

- SA0025：非限定枚举常量
- SA0171：枚举应具有“strict”属性

反面示例：

```
TYPE E_SignalColor :
(
    Red,
    Yellow,
    Green
);
END_TYPE
```

正面示例：

```
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE E_SignalColor :
(
    Red,
    Yellow,
    Green
);
END_TYPE
```

不初始化任何值，只初始化枚举的第 1 个值或所有值

在枚举中，您不会初始化任何值，只使用分配运算符“:=”初始化第 1 个值或所有值。

反面示例：

```
// NON COMPLIANT: init none, the first or all enumerators
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE E_SignalColor :
(
    Red    := 0,
    Yellow := 1,
    Green
);
END_TYPE
```

正面示例 1：

```
// COMPLIANT: no enumerator is initialized
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE E_SignalColor :
(
    Red,
    Yellow,
    Green
);
END_TYPE
```

正面示例 2：

```
// COMPLIANT: first enumerator is initialized
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE E_SignalColor :
(
    Red  := 0,
    Yellow,
    Green
);
END_TYPE
```

正面示例 3：

```
// COMPLIANT: all enumerators are initialized
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE E_SignalColor :
(
    Red    := 0,
    Yellow := 1,
    Green  := 2
);
END_TYPE
```

另请参见：

- DUT 对象

18.3.3.2 DUT 的使用

主题：

1. 仅将枚举变量分配给相关的枚举符 [▶ 1025] [++]

仅将枚举变量分配给相关的枚举符

仅应将相关枚举的枚举符分配给枚举的实例。如要强制执行该规则，建议在定义枚举的类型时设置属性 {attribute 'strict'} [▶ 762]。另请参见：对枚举使用属性“qualified_only”和“strict” [▶ 1024]

静态分析：

使用以下静态分析规则进行验证：

- SA0034：赋值不正确的枚举变量
- SA0171：枚举应具有“strict”属性

反面示例：

```
{attribute 'qualified_only'}
TYPE E_ColorTrafficLight :
(
    Red    := 0,
    Yellow := 1,
    Green  := 2
);
END_TYPE
```

```
PROGRAM Sample_neg
VAR
    eColorTrafficLight : E_ColorTrafficLight;  // Used to handle the state of the traffic light
END_VAR
```

```
CASE eColorTrafficLight OF
    E_ColorTrafficLight.Green:
        eColorTrafficLight  := E_ColorTrafficLight.Yellow;
    E_ColorTrafficLight.Yellow:
        eColorTrafficLight  := E_ColorTrafficLight.Red;
    E_ColorTrafficLight.Red:
        eColorTrafficLight  := 2;              // NON COMPLIANT: Only values defines in E_StandardCo
lor should be assigned to enum instance eColorTrafficLight
ELSE
    eColorTrafficLight := E_ColorTrafficLight.Red;
END_CASE
```

正面示例：

```
{attribute 'strict'}
{attribute 'qualified_only'}
TYPE E_ColorTrafficLight :
(
    Red    := 0,
    Yellow := 1,
    Green  := 2
);
END_TYPE
```

```
PROGRAM Sample_pos
VAR
    eColorTrafficLight : E_ColorTrafficLight;             // Used to handle the state of the traffic
 light
END_VAR
```

```
CASE eColorTrafficLight OF
    E_ColorTrafficLight.Green:
        eColorTrafficLight  := E_ColorTrafficLight.Yellow;
    E_ColorTrafficLight.Yellow:
        eColorTrafficLight  := E_ColorTrafficLight.Red;
    E_ColorTrafficLight.Red:
        eColorTrafficLight  := E_ColorTrafficLight.Green; // COMPLIANT
ELSE
    eColorTrafficLight := E_ColorTrafficLight.Red;
END_CASE
```

另请参见：

- 对象 DUT [▶ 68]

18.3.4 POU

18.3.4.1 函数、方法和动作的实现

主题：

1. 函数/方法中没有大参数的“按值调用” [▶ 1026] [++]
2. 不要在函数/方法中声明大变量 [▶ 1027] [++]
3. 不要使用动作 [▶ 1027] [+]
4. 在内部使用函数/方法的所有参数 [▶ 1027] [+]
5. 仅在 1 处分配函数/方法的返回值 [▶ 1028] [+]
6. 尽可能限制对方法的访问 [▶ 1029] [+]
7. 将参数分组为结构 [▶ 1029] [+]

函数/方法中没有大参数的“按值调用”

在大多数情况下，在调用函数/方法时会复制方法的输入和输出参数以及返回值，这就是“按值调用”。对于占用大量内存的参数，该复制步骤需要更多时间。如要编写高效的程序代码，应使用“按引用调用”来传递大参数。因此，仅设置 1 个指针。这有几种不同的可能性，在示例中都有所涉及。请注意，某些选项还允许对输入参数进行写入访问。

另请参见“传输大字符串 [▶ 1045]”。

反面示例：

```
METHOD SampleMethod_neg : ST_DataCollection    // return value with call by value
VAR_INPUT
    aSensor1Data     : ARRAY[1..100] OF LREAL; // input with call by value
    aSensor2Data     : ARRAY[1..200] OF LREAL; // input with call by value
    aSensor3Data     : ARRAY[1..300] OF LREAL; // input with call by value
    aSensor4Data     : ARRAY[1..400] OF LREAL; // input with call by value
END_VAR
VAR_OUTPUT
    aOutputData      : ARRAY[1..500] OF LREAL; // output with call by value
END_VAR
```

正面示例：

```
METHOD SampleMethod_pos : HRESULT
VAR_INPUT
    aSensor1Data     : REFERENCE TO ARRAY[1..100] OF LREAL; // input with call by reference (write
access possible)
    pSensor2Data     : POINTER TO ARRAY[1..200] OF LREAL;   // input buffer, similar to call by
reference (write access possible)
    nSensor2DataSize : UDINT;                               // size in bytes of buffer to ensure the
correct buffer size
    aOutputData      : REFERENCE TO ARRAY[1..500] OF LREAL; // output with call by reference
    stDataCollection : REFERENCE TO ST_DataCollection;      // output with call by reference
```

```
END_VAR
VAR_IN_OUT CONSTANT
    aSensor3Data     : ARRAY[1..300] OF LREAL;              // input with call by reference
END_VAR
VAR_IN_OUT
    aSensor4Data     : ARRAY[1..400] OF LREAL;              // input with call by reference (write
access possible)
END_VAR
```

不要在函数/方法中声明大变量

对于函数/方法中的变量声明，您应避免较大的数字和较大的数据大小。此类变量声明取自栈内存。如果函数/方法被嵌套调用，则所需的内存量将被添加到堆栈中。可用的栈内存不是无限量的，应该事先避免“堆栈溢出”的情况。

在 TwinCAT 项目中的节点系统下方的 Real-Time（实时）节点中规定了栈内存的最大大小。这通常是 64 KB。因此，建议尽可能减少在函数/方法中声明的数据总量，例如，低于 1 KB。因此，特别是在使用 STRING 类型和任意类型的数组时，应确保长度尽可能短。

或者，也可以在功能块实例中声明变量。请注意，这些变量是永久可用的，并不是每次调用方法时都要初始化。

这同样适用于函数/方法的参数。请参见“函数/方法中没有大参数的‘按值调用’ [▶ 1026]”。

反面示例：

```
FUNCTION_BLOCK FB_Sample_neg
VAR
END_VAR
```

```
METHOD SampleMethod_neg : LREAL
VAR
    fSum     : LREAL;
    nCnt     : UINT;
    aLogList : ARRAY[1..50] OF STRING(1023); // locally declared in method leads to stack allocation
END_VAR
```

正面示例：

```
FUNCTION_BLOCK FB_Sample_pos
VAR
    aLogList : ARRAY[1..50] OF STRING(1023); // declared as member of the FB instance
END_VAR
```

```
METHOD SampleMethod_pos : LREAL
VAR
    fSum     : LREAL;
    nCnt     : UINT;
END_VAR
```

正面示例（备选）：

```
FUNCTION_BLOCK FB_Sample2_pos
VAR
END_VAR
```

```
METHOD SampleMethod2_pos : LREAL
VAR
    fSum     : LREAL;
    nCnt     : UINT;
END_VAR
VAR_INST
    aLogList : ARRAY[1..50] OF STRING(1023); // declared as member of the FB instance
END_VAR
```

不要使用动作

程序或功能块不应包含动作。而应包含实现方法。根据它们的用途，可将它们定义为 PUBLIC、PRIVATE 或 PROTECTED。

在内部使用函数/方法的所有参数

函数或方法的所有参数也应在内部使用。

反面示例：

```
FUNCTION F_Sample_neg : INT
VAR_INPUT
    nA       : INT;          // Variable a in term y = a*a
    nB       : INT;          // NON COMPLIANT: nB will not be used
END_VAR
```

```
F_Sample_neg := nA * nA;
```

正面示例：

```
FUNCTION F_Sample_pos : INT  // COMPLIANT: no unused parameter
VAR_INPUT
    nA       : INT;          // Variable a in term y = a*a
END_VAR
```

```
F_Sample_pos := nA * nA;
```

仅在 1 处分配函数/方法的返回值

仅可在 1 处分配函数/方法的返回值。在 IF 或 CASE 语句的不同分支中进行赋值时，这条规则有例外。

示例 1：

反面示例：

```
FUNCTION F_Sample_neg_1 : LREAL
VAR
    fOnePlusHalfEpsilon  : LREAL;          // Auxiliary variable to calculate and validate the machi
ne epsilon
END_VAR
```

```
// NON COMPLIANT: F_Sample_neg_1 is assigned or used more than once
F_Sample_neg_1            := 1.0;

REPEAT
    F_Sample_neg_1       := 0.5 * F_Sample_neg_1;        // NON COMPLIANT: see above
    fOnePlusHalfEpsilon  := 1.0 + 0.5 * F_Sample_neg_1;  // NON COMPLIANT: see above
UNTIL fOnePlusHalfEpsilon <= 1.0
END_REPEAT;
```

正面示例：

```
FUNCTION F_Sample_pos_1 : LREAL
VAR
    fOnePlusHalfEpsilon  : LREAL;          // Auxiliary variable to calculate and validate the machi
ne epsilon
    fResult              : LREAL := 1.0;   // Temporary variable to
END_VAR
```

```
REPEAT
    fResult              := 0.5 * fResult;
    fOnePlusHalfEpsilon  := 1.0 + 0.5 * fResult;
UNTIL fOnePlusHalfEpsilon <= 1.0
END_REPEAT;

F_Sample_pos_1 := fResult;                 // COMPLIANT: F_Sample_pos_1 is only assigned here
```

示例 2：

该示例的一般程序元素：

```
// Enumeration for sample 2
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE E_SampleStatus :
(
    Ok,                         // Used when the status is OK
    Warning,                    // Used when a warning occurs
    Error                       // Used when an error occurs
);
END_TYPE
```

```
// GVL_Sample - global variable list for sample 2
VAR_GLOBAL CONSTANT
    cLimitMin    : INT := 10;  // Minimal limit to validate a sample value
    cLimitMax    : INT := 20;  // Maximal limit to validate a sample value
    cTolerance   : INT := 2;   // Upper and lower tolerance to validate a sample value
END_VAR
```

```
// Returns a status variable (ok, warning, error) for a measure value, using a constant range of val
ues defined on a GVL
FUNCTION F_Sample : E_SampleStatus
```

```
VAR_INPUT
    nValue       : INT;        // Sample value that is validated in respect of a range of values
END_VAR
```

反面示例：

```
// NON COMPLIANT: F_Sample is assigned within different IF-
statements. Multiple write access on return value cannot be excluded.
IF (nValue >= cLimitMin) AND (nValue <= cLimitMax) THEN
    F_Sample := E_SampleStatus.Ok;
END_IF

IF (nValue >= (cLimitMin - cTolerance))
AND (nValue <= (cLimitMax + cTolerance)) THEN
    F_Sample := E_SampleStatus.Warning;
END_IF

IF (nValue < (cLimitMin - cTolerance))
OR (nValue > (cLimitMax + cTolerance))
    F_Sample := E_SampleStatus.Error;
END_IF
```

正面示例：

```
// COMPLIANT: F_Sample is only assigned within one IF-
statement. No multiple write access on return value occurs.
IF (nValue >= cLimitMin) AND (nValue <= cLimitMax) THEN
    F_Sample := E_SampleStatus.Ok;
ELSIF (nValue >= (cLimitMin - cTolerance))
  AND (nValue <= (cLimitMax + cTolerance)) THEN
    F_Sample := E_SampleStatus.Warning;
ELSE
    F_Sample := E_SampleStatus.Error;
END_IF
```

尽可能限制对方法的访问

使用 PRIVATE 或 PROTECTED 访问修饰符，应尽可能限制对方法的访问。因此，不能在外部调用仅供内部调用的方法。这有助于确保更有可能以适合功能块的预期用途的方式使用它。因此，方法的封装会使代码更加安全。此外，如果用户没有从外部调用内部方法，那么随后对内部方法的更改会更加容易。

将参数分组为结构

如果您在调用时需要指定很多参数，最好将它们分组到 1 个或多个结构中。

这样做的好处是，在结构定义中可以存储默认值。结构数据类型的一些常量又可以为不同的使用情况提供不同的默认值。

通过使用“按引用调用”来传递结构，这可以有效地实现调用。这能够避免许多单独的“按值调用”传输。另请参见主题函数/方法中没有大参数的“按值调用” [▶ 1026]。

另请参见：

- 对象方法 [▶ 82]
- 对象函数 [▶ 73]

18.3.4.2 函数和方法的使用

主题：

1. 评估 POU 返回的错误信息 [▶ 1029] [++]
2. 使用函数/方法的返回值 [▶ 1030] [+]
3. 不要在自身内部调用函数/方法 [▶ 1031] [+]

评估 POU 返回的错误信息

如果函数、方法或功能块返回错误信息，一定要对其进行评估。

静态分析：

按主题建议的静态分析规则：

- SA0009：未使用的返回值
- SA0169：忽略的输出

反面示例：

```
PROGRAM Sample_neg
VAR
    fbFileOpen    : FB_FileOpen;      // FileOpen-FB for logger
    bFileOpenExec : BOOL;             // Execute FileOpen-FB
END_VAR
```

```
// NON COMPLIANT: error information of fbFileOpen will not be used
fbFileOpen(
    sPathName := 'C:\TestFile.txt',
    nMode     := FOPEN_MODEWRITE OR FOPEN_MODETEXT,
    ePath     := PATH_GENERIC,
    bExecute  := bFileOpenExec,
    tTimeout  := T#3S);
```

正面示例：

```
PROGRAM Sample_pos
VAR
    fbFileOpen        : FB_FileOpen;  // FileOpen-FB for logger
    bFileOpenExec     : BOOL;         // Execute FileOpen-FB
    bFileOpenError    : BOOL;         // Error flag of FileOpen-FB
    nFileOpenErrorID  : UDINT;        // Error code of FileOpen-FB
END_VAR
```

```
// COMPLIANT: error information will be handled
fbFileOpen(
    sPathName := 'C:\TestFile.txt',
    nMode     := FOPEN_MODEWRITE OR FOPEN_MODETEXT,
    ePath     := PATH_GENERIC,
    bExecute  := bFileOpenExec,
    tTimeout  := T#3S,
    bError    => bFileOpenError,
    nErrId    => nFileOpenErrorID);

IF bFileOpenError THEN
    F_DoSomethingUsefulHere();        // Handle error here
END_IF
```

使用函数/方法的返回值

在函数/方法的调用点应该使用函数/方法的返回值，即查询和评估。如果以这种方式返回错误值，则这会特别有用。不过，也有可能会出现例外情况，即每次调用函数/方法时都不必使用返回值。

静态分析：

借助静态分析规则进行检查：

- SA0009：未使用的返回值

该规则的一般程序元素：

```
(* Function for all samples in this rule: Adds a message to logger system.
   Returns TRUE if successful, FALSE on Error. *)
FUNCTION F_AddLogMessage : BOOL
VAR_INPUT
    sMessage        : WSTRING;        // Message to be logged
    dtTimestamp     : DATE_AND_TIME;  // Timestamp of the message
END_VAR
```

反面示例：

```
PROGRAM Sample_neg
VAR
    dtNow           : DATE_AND_TIME;  // Actual system time
END_VAR
```

```
// NON COMPLIANT: return value of function will not be used
F_AddLogMessage(sMessage := "Test Message", dtTimestamp := dtNow);
```

正面示例：

```
PROGRAM Sample_pos
VAR
    dtNow           : DATE_AND_TIME;  // Actual system time
    bSendMessageOk  : BOOL;           // Used to check if sending of message was successful
END_VAR
```

```
// COMPLIANT: return value of function will be used
bSendMessageOk := F_AddLogMessage(sMessage := "Test Message", dtTimestamp := dtNow);

IF NOT bSendMessageOk THEN
    F_DoSomethingUsefulHere();        // Handle error here
END_IF
```

不要在自身内部调用函数/方法

函数/方法不应直接或间接调用自身，以避免递归。在 IEC61131 之外的编程语言中，也应该谨慎地使用递归。

静态分析：

借助静态分析规则进行检查：

- SA0160：递归调用

反面示例：

```
FUNCTION F_Sample_neg : DWORD
VAR_INPUT
    nFac : DWORD;     // The faculty of this value will be calculated
END_VAR
------------------------------------------------------------------------
IF nFac = 0 THEN
    F_Sample_neg := 1;
ELSE
    // NON COMPLIANT: implicit recursion. F_Sample_neg_IndirectFac calls F_Sample_neg
    F_Sample_neg := nFac * F_Sample_neg_IndirectFac(nFac := (nFac - 1));
END_IF
------------------------------------------------------------------------
------------------------------------------------------------------------
FUNCTION F_Sample_neg_IndirectFac : DWORD
VAR_INPUT
    nFac : DWORD;     // The faculty of this value will be calculated
END_VAR
------------------------------------------------------------------------
F_Sample_neg_IndirectFac := F_Sample_neg(nFac := nFac);
```

正面示例：

```
FUNCTION F_Sample_pos : DWORD
VAR_INPUT
    nFac : DWORD;     // The faculty of this value will be calculated
END_VAR
VAR
    nTemp   : DWORD;  // Temporary variable used to calculate faculty
    nCount  : DWORD;  // Counter variable used to calculate faculty
END_VAR
------------------------------------------------------------------------
nTemp := 1;

IF nFac > 0 THEN
    FOR nCount := 1 TO nFac DO
        nTemp := nTemp * nCount;
    END_FOR
END_IF

F_Sample_pos := nTemp;
```

另请参见：

- 对象方法 [▶ 82]
- 对象函数 [▶ 73]

18.3.4.3 功能块的实现

主题：

1. 确保在线更改能力 [▶ 1032] [++]
2. 具有一次性异步处理功能的统一接口 [▶ 1032] [+]
3. 具有连续异步处理功能的统一接口 [▶ 1032] [+]
4. 在内部使用功能块的所有参数 [▶ 1033] [+]
5. 将参数分组为结构 [▶ 1033] [+]

确保在线更改能力

在线更改能力是 PLC 应用程序的 1 个重要基本特征。通过在线更改的方式，PLC 程序可以进行各种更改，而无需停止周期序列。通常情况下，程序代码会在实现部分进行更改，为此也会在声明部分添加个别变量。

在实现功能块时，如果由于在线更改而在内存中移动实例，则实例的功能不会受到影响。

有关详细信息，请参见 TwinCAT 3 PLC 文档中的在线更改的执行 [▶ 231]章节。

具有一次性异步处理功能的统一接口

除了变量的命名规范 [▶ 1007]和变量声明的文本块顺序 [▶ 999]之外，统一接口还有助于功能块的使用。

对于主体调用处理异步和一次性 ADS 处理的功能块，您应该使用以下接口。如果不是 ADS 通信，则可省略参数 sNetId。

处理从输入 bExecute 的上升沿开始。只有当功能块不是 bBusy 时才有可能实现这一点。在处理过程中，用户不应该更改输入参数。这也可以确保处理结果（输出）与指定参数（输入）相匹配。

在处理完成后会设置 bBusy := FALSE。最迟现在您必须设置所有输出参数。因此，用户有可能对 bBusy = FALSE 做出反应，并评估作为结果的输出。

```
FUNCTION_BLOCK FB_Sample
VAR_INPUT
    bExecute      : BOOL;
    ...           : ...;                          // optional inputs
    ...           : ...;                          // optional inputs
    tTimeout      : TIME := DEFAULT_ADS_TIMEOUT; // ADS communication timeout
    sNetId        : T_AmsNetId := '';             // keep empty '' for the local device
END_VAR
VAR_OUTPUT
    bBusy         : BOOL;
    bError        : BOOL;
    hrErrorCode   : HRESULT;
    ...           : ...;                          // optional outputs
    ...           : ...;                          // optional outputs
END_VAR
```

具有连续异步处理功能的统一接口

除了变量的命名规范 [▶ 1007]和变量声明的文本块顺序 [▶ 999]之外，统一接口还有助于功能块的使用。

对于主体调用处理异步和连续 ADS 处理的功能块，可使用以下接口。如果不是 ADS 通信，则可省略参数 sNetId。

处理从 bEnable 的上升沿开始。只要设置 bEnable，功能块就会运行。由于是异步处理，因此，重置 bEnable 不一定会导致立即终止所有处理。这可以通过 bBusy = FALSE 显示。在设置 bEnable 时，可以处理更改后的输入参数。

输出 bValid 表示处理成功和可选输出参数的有效性。

输出 bError 表示已发生错误。输出 hrErrorCode 的错误代码可提供更加详细的信息。由于在连续处理过程中的任何时间均可设置 bError，而且它通常只存在 1 个周期，因此用户应该永久查询该输出。

```
FUNCTION_BLOCK FB_Sample
VAR_INPUT
    bEnable       : BOOL;
    ...           : ...;                          // optional inputs
    ...           : ...;                          // optional inputs
    tTimeout      : TIME := DEFAULT_ADS_TIMEOUT; // ADS communication timeout
```

```
    sNetId        : T_AmsNetId := '';           // keep empty '' for the local device
END_VAR
VAR_OUTPUT
    bValid        : BOOL;
    bBusy         : BOOL;
    bError        : BOOL;
    hrErrorCode   : HRESULT;
    ...           : ...;                        // optional outputs
    ...           : ...;                        // optional outputs
END_VAR
```

在内部使用功能块的所有参数

功能块的所有参数也应在内部使用。

反面示例：

```
FUNCTION FB_Sample_neg
VAR_INPUT
    nA       : INT;          // Variable a in term y = a*a
    nB       : INT;          // NON COMPLIANT: nB will not be used
END_VAR
VAR_OUTPUT
    nY       : INT;
END_VAR
```

```
nY := nA * nA;
```

正面示例：

```
FUNCTION FB_Sample_pos       // COMPLIANT: no unused parameter
VAR_INPUT
    nA       : INT;          // Variable a in term y = a*a
END_VAR
VAR_OUTPUT
    nY       : INT;
END_VAR
```

```
nY := nA * nA;
```

将参数分组为结构

如果您在功能块中需要指定很多参数，最好将它们分组到 1 个或多个结构中。例如，在视觉上可以很好地分开配置参数与控制参数。

另请参见：

- 对象功能块 [▶ 76]

18.3.4.4 功能块的使用

主题：

1. 不要将功能块实例声明为 VAR PERSISTENT [▶ 1033] [++]
2. 评估 POU 返回的错误信息 [▶ 1033] [++]
3. 不要从外部访问 POU 的局部变量 [▶ 1034] [++]
4. 不得直接分配对象 [▶ 1035] [++]
5. 没有临时功能块实例 [▶ 1035] [+]

不要将功能块实例声明为 VAR PERSISTENT

不要将功能块实例声明为 VAR PERSISTENT [▶ 630]，因为这会导致将整个 FB 实例存储在持久变量的文件中。例如，这可能会给在内部使用 ADS 块或指针变量的功能块带来问题。

评估 POU 返回的错误信息

如果函数、方法或功能块返回错误信息，一定要对其进行评估。

静态分析：

按主题建议的静态分析规则：

- SA0009：未使用的返回值
- SA0169：忽略的输出

反面示例：

```
PROGRAM Sample_neg
VAR
    fbFileOpen    : FB_FileOpen;      // FileOpen-FB for logger
    bFileOpenExec : BOOL;             // Execute FileOpen-FB
END_VAR
```

```
// NON COMPLIANT: error information of fbFileOpen will not be used
fbFileOpen(
    sPathName := 'C:\TestFile.txt',
    nMode     := FOPEN_MODEWRITE OR FOPEN_MODETEXT,
    ePath     := PATH_GENERIC,
    bExecute  := bFileOpenExec,
    tTimeout  := T#3S);
```

正面示例：

```
PROGRAM Sample_pos
VAR
    fbFileOpen        : FB_FileOpen;  // FileOpen-FB for logger
    bFileOpenExec     : BOOL;         // Execute FileOpen-FB
    bFileOpenError    : BOOL;         // Error flag of FileOpen-FB
    nFileOpenErrorID  : UDINT;        // Error code of FileOpen-FB
END_VAR
```

```
// COMPLIANT: error information will be handled
fbFileOpen(
    sPathName := 'C:\TestFile.txt',
    nMode     := FOPEN_MODEWRITE OR FOPEN_MODETEXT,
    ePath     := PATH_GENERIC,
    bExecute  := bFileOpenExec,
    tTimeout  := T#3S,
    bError    => bFileOpenError,
    nErrId    => nFileOpenErrorID);

IF bFileOpenError THEN
    F_DoSomethingUsefulHere();        // Handle error here
END_IF
```

不要从外部访问 POU 的局部变量

不应从 POU 外部访问 POU 的局部变量。

在 TwinCAT 3 中，无法从编程对象外部对局部变量进行写访问，因为这会导致编译器错误。另一方面，读取访问则不会被编译器拦截。

为了保持预期的数据封装，强烈建议不要从 POU 外部访问 POU 的局部变量，无论是在读取模式下还是在写入模式下均是如此。此外，使用库功能块时，无法保证功能块的局部变量在 PLC 库的后续更新过程中保持不变。这意味着在库更新之后，可能无法再对应用程序项目进行正确编译。

静态分析：

借助静态分析规则进行检查：

- SA0102：从外部访问程序/fb 变量

反面示例：

```
FUNCTION_BLOCK FB_Sample
VAR
    nLocal          : INT;
END_VAR
```

```
PROGRAM MAIN
VAR
    fbSample        : FB_Sample;
    nToBeProcessed  : INT;
END_VAR
------------------------------------------------------------------------
nToBeProcessed := fbSample.nLocal;  // NON COMPLIANT: Do not access local variables from external co
ntext.
```

正面示例：

```
FUNCTION_BLOCK FB_Sample
VAR_OUTPUT
    nOutput         : INT;
END_VAR
VAR
    nLocal          : INT;
END_VAR
```

```
PROGRAM MAIN
VAR
    fbSample        : FB_Sample;
    nToBeProcessed  : INT;
END_VAR
-------------------------------------------------------------------------
nToBeProcessed := fbSample.nOutput;  // COMPLIANT: Output variable is accessed from external context
.
```

不得直接分配对象

简单数据类型（INT、BYTE、STRING.....）甚至结构通常都是直接分配的。

您不应使用分配运算符“:=”来分配功能块的实例。在进行此类分配时，会复制 1 个对象。但是，功能块内部通常包含地址，如果分配它们，这些地址就会被破坏。

如要在原则上禁止此类分配，您可以在功能块的定义中设置属性“no_assign” [▶ 750]。

静态分析：

借助静态分析规则进行检查：

- SA0014：实例的分配

没有临时功能块实例

您不应在堆栈上声明功能块实例，即您不应将其声明为临时变量。临时实例是在方法中、在函数中或作为 VAR_TEMP 声明的实例，因此它们在每个处理周期或每个块调用中都要重新初始化。

功能块的状态通常会保留几个 PLC 周期。堆栈上的实例仅在函数调用期间存在。因此，只有在极少数情况下才需要创建 1 个实例作为临时变量。其次，功能块实例通常很大，需要在堆栈上占用大量空间，而控制器上的空间通常有限。第三，功能块的初始化和调度往往也会占用大量时间。

静态分析：

借助静态分析规则进行检查：

- SA0167：临时功能块实例

免授权版本 轻量静态分析 [▶ 139] 也包含规则 SA0167。

反面示例：

```
FUNCTION F_Sample
VAR
    fbCtrl : FB_Control;
END_VAR
```

正面示例：

```
FUNCTION F_Sample
VAR_INPUT
    fbCtrl : REFERENCE TO FB_Control;
END_VAR
```

另请参见：

- 对象功能块 [▶ 76]

18.3.5 变量

18.3.5.1 一般信息

主题：

1. 不要测试浮点数是否相等或不等 [▶ 1036] [++]
2. 使用显式转换避免不必要的结果 [▶ 1037] [+]

不要测试浮点数是否相等或不等

不应直接或间接测试浮点数是否相等或不等。而应使用运算符 <、>、<=、>=。

在计算机中，浮点数不可能总是在没有舍入误差的情况下表示出来。特别是在计算多个浮点数时，微小的舍入误差很快就会累加起来。因此，浮点数绝对不能直接与“=”运算符进行比较。最小可表示数的度量是所谓的机器精度（ε）。在下面的示例之后，介绍了 1 个可用于计算机器精度的函数。

唯一的例外是检查是否等于（“=”）或不等于（“<>”）零。

静态分析：

借助静态分析规则进行检查：

- SA0054：比较 REAL/LREAL 是否相等/不等

反面示例 1：

```
PROGRAM Sample_neg_1
VAR
    fTest           : REAL   := 1E-20;  // Test number
END_VAR
```

```
// The IF-condition is NON COMPLIANT: The comparison is not reliable due rounding errors
IF fTest = 0 THEN
    F_DoSomethingUsefulHere();          // it's just a sample
END_IF
```

正面示例 1：

```
PROGRAM Sample_pos_1
VAR
    fTest           : REAL   := 1E-20;  // Test number
END_VAR
VAR CONSTANT
    cFloatEpsilon   : REAL   := 1E-12;  // The deviation epsilon
END_VAR
```

```
// The IF-condition is COMPLIANT: Test with the deviation (± epsilon) around 0
IF (fTest > (0 - cFloatEpsilon)) AND (fTest < (0 + cFloatEpsilon)) THEN
    F_DoSomethingUsefulHere();          // it's just a sample
END_IF
```

反面示例 2：

```
PROGRAM Sample_neg_2
VAR
    fCounter        : REAL;             // Test counter
END_VAR
```

```
// The WHILE-
condition is NON COMPLIANT: The comparison is not reliable due rounding errors. Attention: This loop
 will be repeated endlessly!
WHILE fCounter <> 1 DO
    fCounter := fCounter + 0.001;
END_WHILE
```

正面示例 2：

```
PROGRAM Sample_pos_2
VAR
    fCounter        : REAL;             // Test counter
END_VAR
```

```
// The WHILE-
condition is COMPLIANT because of comparison operator '<'. The loop ends when fCounter is not smalle
r than 1 anymore.
```

```
WHILE fCounter < 1 DO
    fCounter := fCounter + 0.001;
END_WHILE
```

以下函数可用于计算机器的精度。

```
FUNCTION F_CalculateMachineEpsilon : LREAL
VAR
    fOnePlusHalfEpsilon  : LREAL;               // Auxiliary variable to calculate and validate the
machine epsilon
    fResult              : LREAL := 1.0;        // Temporary variable to calculate the result
END_VAR
```

```
REPEAT
    fResult   := 0.5 * fResult;                 // Devide fResult by 2.
    fOnePlusHalfEpsilon := 1.0 + 0.5 * fResult; // If new result devided by 2 plus 1.0 is smaller th
an or equal to 1.0, the Epsilon is calculated.
UNTIL fOnePlusHalfEpsilon <= 1.0
END_REPEAT;

F_CalculateMachineEpsilon := fResult;           // Set return value to fResult
```

使用显式转换避免不必要的结果

由于隐式转换或缺少转换，可能会出现不想要的结果。一方面，这可能是由于编译器的隐式转换太迟造成的（请参见示例和 SA0130）。另一方面，如果转换变量太迟，则可能会出现不想要的结果，因为该变量的操作会在处理器的平台相关寄存器宽度内执行，而不是在变量类型的大小内执行（请参见 SA0066）。因此，您在编程时应该注意变量类型（的大小），在必要时应该使用显式转换进行转换。

静态分析：

使用以下静态分析规则进行验证：

- SA0020：可能将截断值分配给 REAL 变量
- SA0066：临时结果的使用
- SA0130：隐式扩展转换

以下示例的一般程序元素：

```
PROGRAM Sample
VAR
    nLINT  : LINT;
    nDINT  : DINT;
END_VAR
```

反面示例：

```
// Possibly wrong result due to variable overflow caused by late converting of compiler
nLINT := nDINT * nDINT;
```

正面示例：

```
// Correct result due to explicit variable cast
nLINT := TO_LINT(nDINT) * TO_LINT(nDINT);
```

18.3.5.2 变量封装

主题：

1. 将不变的变量声明为 VAR CONSTANT [▶ 1037] [+]
2. 不要遮蔽全局标识符 [▶ 1038] [+]
3. 限制 ADS 访问 [▶ 1038] [+]

将不变的变量声明为 VAR CONSTANT

未修改的变量应声明为 VAR CONSTANT。

静态分析：

借助静态分析规则进行检查：

- SA0012：可声明为常量的变量

反面示例：

```
PROGRAM Sample_neg
VAR
    fTest         : REAL := 1E-20;   // Test value
    cFloatEpsilon : REAL := 1E-12;   // NON COMPLIANT: cFloatEpsilon will not be changed, but is not
 declared as VAR CONSTANT
END_VAR
-------------------------------------------------------------------
IF (fTest > (0 - cFloatEpsilon)) AND (fTest < (0 + cFloatEpsilon)) THEN
    F_DoSomethingUsefulHere();       // it's just a sample
END_IF
```

正面示例：

```
PROGRAM Sample_pos
VAR
    fTest         : REAL := 1E-20;   // Test value
END_VAR
VAR CONSTANT
    cFloatEpsilon : REAL := 1E-12;   // COMPLIANT: cFloatEpsilon will not be changed and is declared
 as VAR CONSTANT
END_VAR
-------------------------------------------------------------------
IF (fTest > (0 - cFloatEpsilon)) AND (fTest < (0 + cFloatEpsilon)) THEN
    F_DoSomethingUsefulHere();       // it's just a sample
END_IF
```

不要遮蔽全局标识符

内部关系中的标识符不应“遮蔽”更多的全局标识符。内部关系中的标识符包括在方法中实例化的变量。例如，在这种情况下，更多的全局标识符是相应功能块的变量。

静态分析：

使用以下静态分析规则进行验证：

- SA0013：具有相同变量名的声明
- SA0027：多次使用名称

免授权版本 轻量静态分析 [▶ 139] 也包含规则 SA0027。

以下示例的一般程序元素：

```
FUNCTION_BLOCK FB_Sample
VAR
    sDescription : STRING;  // STRING for POU description
END_VAR
-------------------------------------------------------------------
sDescription := 'This is a function block';
```

反面示例：

```
METHOD PUBLIC nccl_Test : STRING
VAR
    sDescription : STRING;  // NON COMPLIANT: sDescription is already defined in method’s outer scop
e FB_Sample
END_VAR
-------------------------------------------------------------------
sDescription  := 'This is a method';
nccl_Test     := sDescription;
```

正面示例：

```
METHOD PUBLIC nccl_Test : STRING
VAR
    sMethodDescription  : STRING;  // COMPLIANT
END_VAR
-------------------------------------------------------------------
sMethodDescription  := 'This is a method';
nccl_Test           := sMethodDescription;
```

限制 ADS 访问

如果需要，可以限制对变量的访问，以便无法通过 ADS 看到变量。这样，这些变量就无法用于 HMI。为此，可以使用属性“TcNoSymbol”。

18.3.5.3 数组

主题：

1. 通过常量定义数组限值 [▶ 1039] [++]
2. 数组下限值为 1 [▶ 1039] [+]

通过常量定义数组限值

您应该使用常量变量定义数组的限值。在循环中访问数组时，应该使用相同的常量变量来定义循环限值。

- 数组的下限值 = 循环的下限值 = 常量变量 1
- 数组的上限值 = 循环的上限值 = 常量变量 2

另请注意以下有关编程规范的主题：

- 将循环的限值声明为常量 [▶ 1015]。

以下示例的一般程序元素：

```
TYPE ST_Object :
STRUCT
    sName     : STRING;
    nID       : UINT;
END_STRUCT
END_TYPE
```

反面示例：

```
VAR
    aObjects  : ARRAY[1..10] OF ST_Object;
END_VAR
```

正面示例：

```
VAR CONSTANT
    cMin      : UINT := 1;
    cMax      : UINT := 10;
END_VAR
VAR
    aObjects  : ARRAY[cMin..cMax] OF ST_Object;
END_VAR
```

数组下限值为 1

对于 IEC 61131 语言中的数组，可以显式定义上限值和下限值。1 个值为 1 的常量变量应该用作下限值，以便将数组中的元素数量作为上限值。上限值也应由 1 个常量变量来定义。

这与高级语言中通常隐含的以 0 为下限值的情况相反。尽管如此，还是建议使用 1 作为下限值，以便对数组进行一致且轻松的处理。

根据特定的应用，其他限值也可能有用。不过，在 1 个功能块内，使用方法的一致性至关重要。

正面示例：

```
VAR CONSTANT
    cMin          : UDINT := 1;
    cMax          : UDINT := 10;
    cMaxObjects   : UDINT := 10;
END_VAR
VAR
    nIndex        : UDINT;
    aSample       : ARRAY[cMin..cMax] OF REAL;
    aObjects      : ARRAY[cMin..cMaxObjects] OF ST_Object;
END_VAR
```

```
FOR nIndex := cMin TO cMax DO
    aSample[nIndex] := 123.456;
END_FOR
```

18.3.5.4 指针、引用、接口

主题：

1. 指向临时存在的对象的指针/引用/接口的临时存在 [▶ 1040] [++]
2. 每个周期重置指针/引用 [▶ 1040] [++]
3. 在每次使用前检查指针/引用/接口 [▶ 1041] [++]
4. 优先选择引用，而不是指针 [▶ 1042] [++]

指向临时存在的对象的指针/引用/接口的临时存在

指向临时存在的对象的指针、引用或接口的存在时间应与对象本身的存在时间相同。
例如，在方法 [▶ 82]中、在函数 [▶ 73]中或作为 VAR TEMP [▶ 627] 进行实例化时，对象是临时存在的。

静态分析：

借助静态分析规则进行检查：

- SA0021：传输临时变量的地址

反面示例：

```
FUNCTION_BLOCK FB_Sample_neg
VAR
    pPointerToSample : POINTER TO ST_Sample;  // Sample pointer to a structure
END_VAR
```

```
METHOD PUBLIC nccl_TestMethod
VAR
    stSample         : ST_Sample;
END_VAR
-------------------------------------------------------------------------
pPointerToSample := ADR(stSample);
// NON COMPLIANT: stSample is declared in an inner scope. So pPointerToSample points to an invalid o
bject outside of the method.
```

正面示例：

```
FUNCTION_BLOCK FB_Sample_pos
VAR
END_VAR
```

```
METHOD PUBLIC nccl_TestMethod
VAR
    stSample         : ST_Sample;
    pPointerToSample : POINTER TO ST_Sample;  // COMPLIANT
END_VAR
-------------------------------------------------------------------------
pPointerToSample := ADR(stSample);
```

每个周期重置指针/引用

您应该每个周期都重置指针和引用。

在线更改可以改变静态声明的变量和对象的地址。因此，每个周期都应该对指向此类变量的指针和引用进行重置。为此，使用 ADR() 运算符 [▶ 639]可以再次查询变量的地址。

指向动态创建对象（__NEW [▶ 674]）的指针可能不会在每个周期进行重置，但在显式发布（__DELETE [▶ 676]）对象之前必须保留它们。

如果在方法输入时要求使用指针/引用，则所有方法输入参数的强制分配均会导致指针/引用被调用方重置。

反面示例：

```
FUNCTION_BLOCK FB_Sample_neg
VAR
    aBuffer : ARRAY[cMin..cMax] OF BYTE;         // Sample buffer
    pBuffer : POINTER TO BYTE := ADR(aBuffer);  // Sample pointer to buffer
// NON COMPLIANT: the memory address of aBuffer could change during an online change. So pBuffer cou
ld become invalid.
END_VAR
```

正面示例：

```
FUNCTION_BLOCK FB_Sample_pos
VAR
    aBuffer : ARRAY[cMin..cMax] OF BYTE;        // Sample buffer
    pBuffer : POINTER TO BYTE;                  // Sample pointer to buffer
END_VAR
-------------------------------------------------
pBuffer := ADR(aBuffer);    // COMPLIANT: pointer is updated before usage in implementation
```

在每次使用前检查指针/引用/接口

在每次使用前都应该检查指针、引用和接口是否有效。对于指针和接口，通过查询“不等于 0”（<> 0）可以完成此类检查；对于引用，则可以使用运算符 __ISVALIDREF [▶ 677]() 来完成此类检查。

对于指针的特殊检查，Tc2_System 库的函数 F_CheckMemoryArea 可用于特殊情况，该函数可用于查询指针所引用的内存区。

在检查指针有效性的条件中，不应同时使用指针。如果您想要实现这一点，您必须使用运算符 AND THEN。请参见示例 2。

静态分析：

使用以下静态分析规则进行验证：

- SA0039：可能的空指针解除引用
- SA0046：可能使用未初始化的接口
- SA0145：可能使用未初始化的引用

按主题建议的静态分析规则：

- SA0124：初始化中的解除引用访问
- SA0125：初始化中的引用

以下示例的一般程序元素：

功能块 FB_Sample 实现接口 I_Sample：

```
FUNCTION_BLOCK FB_Sample IMPLEMENTS I_Sample
```

示例程序：

```
PROGRAM Sample
VAR
    pSample   : POINTER TO FB_Sample;
    refSample : REFERENCE TO FB_Sample;
    ipSample   : I_Sample;
END_VAR
```

反面示例：

```
pSample^.DoSomething();
refSample.DoSomething();
ipSample.DoSomething();
```

正面示例 1：

```
IF pSample <> 0 THEN
    pSample^.DoSomething();
END_IF

IF __ISVALIDREF(refSample) THEN
    refSample.DoSomething();
END_IF

IF ipSample <> 0 THEN
    ipSample.DoSomething();
END_IF
```

正面示例 2：

```
IF ipBuffer <> 0 THEN
    IF ipBuffer.bAvailable THEN
        ipBuffer.Clear();
    END_IF
END_IF
```

```
IF ipBuffer <> 0 AND_THEN ipBuffer.bAvailable THEN
    ipBuffer.Clear();
END_IF
```

优先选择引用，而不是指针

如果可能的话，优先选择引用，而不是指针，因为引用比指针的类型安全性更高。

不过，在特殊情况下，例如，在需要进行指针运算时，则需要使用指针。同样，任意大小的缓冲区也可以随时传递给函数。指针也可以用于此目的。

静态分析：

可能需要的实用的指针使用规则：

- SA0017：非定期分配
- SA0019：隐式指针转换
- SA0061：指针上的异常操作
- SA0064：指针的添加
- SA0065：向基础大小添加不正确的指针

另请参见：

- 指针 [▶ 708]
- 引用 [▶ 712]
- 对象接口 [▶ 94]

18.3.5.5 分配的变量

主题：

1. 不要使用直接寻址 [▶ 1042] [++]
2. 避免对输出进行多次写入访问 [▶ 1043] [++]
3. 避免地址变量的内存区重叠 [▶ 1043] [+]

不要使用直接寻址

对于分配的变量，不应使用直接寻址。建议使用 * 占位符进行自动寻址，而不是直接寻址。如果使用占位符 *（%I*、%Q* 或 %M*），TwinCAT 将自动执行灵活的优化寻址。

在实现部分中也不应有直接地址访问。

静态分析：

使用以下静态分析规则进行验证：

- SA0047：访问直接地址
- SA0048：关于直接地址的 AT 声明

按主题建议的静态分析规则：

- SA0005：无效地址和数据类型

反面示例：

```
VAR
    bInputSignal  AT%IX4.0 : BOOL;
    bVar                   : BOOL;
END_VAR
```

```
bVar := %IX0.0;
```

正面示例：

```
VAR
    bInputSignal  AT%I*    : BOOL;
END_VAR
```

避免对输出进行多次写入访问

您应避免对输出进行多次写入访问。如果在程序中的多个点上写入输出，则会对输出进行多次写入访问。

此规则的例外情况：在 IF 或 CASE 语句的不同分支中进行赋值。

静态分析：

借助静态分析规则进行检查：

- SA0004：输出上的多次写入访问

免授权版本 轻量静态分析 [▶ 139] 也包含规则 SA0004。

反面示例：

```
PROGRAM Sample_neg
VAR
    bErrorLed      AT%Q*  : BOOL;             // Machine's error LED
    bErrorDrill           : BOOL;             // Drill error status
    bErrorTransport       : BOOL;             // Transport error status
    nTestCounter          : WORD;             // Simple counter
END_VAR
-------------------------------------------------------------------------
bErrorLed := bErrorDrill OR bErrorTransport;  // NON COMPLIANT: bErrorLed will be written more than
once: See action CounterInc
CounterInc();
-------------------------------------------------------------------------
-------------------------------------------------------------------------
(* --- Method: CounterInc ---
   This method increments the counter and checks if nCounterValue is equal to 0 *)
bErrorLed   := (nTestCounter = 0);            // NON COMPLIANT: bErrorLed will be written more than
once: See program MAIN
nTestCounter := nTestCounter + 1;
```

正面示例：

```
PROGRAM Sample_pos
VAR
    bErrorLed      AT%Q* : BOOL;     // Machine's error LED
    bErrorDrill          : BOOL;     // Drill error status
    bErrorTransport      : BOOL;     // Transport error status
    bErrorCounter        : BOOL;     // Test counter error status
    nTestCounter         : WORD;     // Simple counter
END_VAR
-------------------------------------------------------------------------
CounterInc();
    bErrorLed := bErrorDrill         // COMPLIANT
    OR bErrorTransport
    OR bErrorCounter;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
(* --- Method: CounterInc ---
This method increments the counter and checks if nCounterValue is equal to 0 *)
bErrorCounter  := (nTestCounter = 0); // COMPLIANT
nTestCounter   := nTestCounter + 1;
```

避免地址变量的内存区重叠

您应避免地址变量的内存区重叠。如果多个变量使用相同的存储空间，则会出现内存区重叠的情况。

如果由于编程原因需要重叠内存区，则可以出于风格方面的考虑而使用 UNION [▶ 725]。您应避免其他变量（例如，地址变量）中的内存区重叠。

静态分析：

借助静态分析规则进行检查：

- SA0028：内存区重叠

免授权版本 轻量静态分析 [▶ 139] 也包含规则 SA0028。

反面示例：

```
PROGRAM Sample_neg
VAR
    nTestWord       AT%MW2  : WORD;  // Test WORD
    nTestLowByte    AT%MB4  : BYTE;  // NON COMPLIANT: overlaps with nTestWord
    nTestHighByte   AT%MB5  : BYTE;  // NON COMPLIANT: overlaps with nTestWord
END_VAR
-----------------------------------------------------------------------
nTestLowByte   := 0;                // NON COMPLIANT: writes nTestWord = 16#xx00
nTestHighByte  := 0;                // NON COMPLIANT: writes nTestWord = 16#00xx
nTestWord      := 16#C0FF;          // Writes nTestLowByte and nTestHighByte too
```

正面示例：

```
// Structure for the positive sample
TYPE ST_2_BYTES :
STRUCT
    nLow        : BYTE;         // Low byte
    nHigh       : BYTE;         // High byte
END_STRUCT
END_TYPE
```

```
// Union for the positive sample
TYPE U_BYTE_WORD :
UNION
    stLowHigh   : ST_2_BYTES;  // Struct with two bytes
    nValue      : WORD;         // Test WORD to be united with the two-byte-struct
END_UNION
END_TYPE
```

```
PROGRAM Sample_pos
VAR
    uTestWordToBytes   AT%MW2  : U_BYTE_WORD;                // Test union
    nTestLowByte               : BYTE;                       // COMPLIANT: not addressed
    nTestHighByte              : BYTE;                       // COMPLIANT: not addressed
END_VAR
-----------------------------------------------------------------------
uTestWordToBytes.nValue := 16#C0FF;                          // A test value
nTestLowByte            := uTestWordToBytes.stLowHigh.nLow;  // Will be 16#FF
nTestHighByte           := uTestWordToBytes.stLowHigh.nHigh; // Will be 16#C0
```

另请参见：

- AT 声明 [▶ 63]
- 地址 [▶ 695]

18.3.5.6 全局变量

主题：

1. 对 GVL 使用属性“qualified_only” [▶ 1044] [+]
2. 明智地使用全局变量 [▶ 1044] [+]

对 GVL 使用属性“qualified_only”

在定义全局变量列表 [▶ 66]或参数列表 [▶ 67]时使用属性 {attribute 'qualified_only'} [▶ 761]，在使用变量时强制使用 GVL 命名空间。通过使用命名空间（例如：GVL_Ctrl.bPaintingActive），变量的全局范围变得一目了然。

正面示例：

全局变量列表“GVL_Ctrl”：

```
{attribute 'qualified_only'}
VAR_GLOBAL
    bPaintingActive  : BOOL;
END_VAR
```

明智地使用全局变量

为了防止并发访问并支持数据封装，您应尽可能避免全局实例化。

同样，您应尽可能避免在功能块中使用现有的全局变量。您可以通过输入参数分配必要的数据。

另请参见：

- 多个任务 [▶ 1048]

18.3.5.7 字符串

主题：

1. 大小和长度规格的标识符 [▶ 1045] [++]
2. 建议的默认大小 [▶ 1045] [+]
3. 大字符串的传输 [▶ 1045] [+]
4. 大字符串的处理 [▶ 1046] [+]

大小和长度规格的标识符

将字符串变量的大小（以字节为单位）称为“Size”。

将字符串的当前长度（以字符为单位）称为“Len”。

正面示例：

```
VAR
    sText     : T_MaxString;
    nTextSize : UDINT;
    nTextLen  : UDINT;
END_VAR
-----------------------------------------
nTextSize := SIZEOF(sText);
nTextLen  := LEN(sText);
```

建议的默认大小

字符串变量的默认建议大小为 T_MaxString（STRING(255) 的别名）。这不仅适用于字符串的局部声明，也适用于函数/方法中的参数的声明。

如果针对特定的使用情况固定了其他大小，则应将其作为大小限值（例如：T_AmsNetId）。

T_MaxString 的大小被定义为诸如 LEN() 和 CONCAT() 等函数的最大值，这会导致常规参数赋值的可接受性能为“按值调用”。

静态分析：

按主题建议的静态分析规则：

- SA0026：可能的截断字符串

正面示例：

```
FUNCTION F_Sample : BOOL
VAR_INPUT
    sParam1   : T_MaxString;
    sParam2   : T_AmsNetId;
END_VAR
```

```
PROGRAM MAIN
VAR
    sLocal1   : T_MaxString;
    sLocal2   : T_AmsNetId;
    bReturn   : BOOL;
END_VAR
```

```
bReturn := F_Sample(sParam1 := sLocal1,
                    sParam2 := sLocal2);
```

大字符串的传输

如果需要比 T_MaxString 更大的字符串，则应将传输用作“按引用调用”。这可以通过使用 VAR_IN_OUT CONSTANT [▶ 624]（用于只读访问）或 POINTER TO STRING 来实现。

输入参数（例如，REFERENCE TO STRING(1023)）只适用于有限的范围，因为这总是要求分配的变量保持固定的大小。如果可以接受或需要这种限制，则优先选择该变体。

正面示例：

```
FUNCTION F_Sample : BOOL
VAR_IN_OUT CONSTANT
    sLonger1     : STRING;
END_VAR
VAR_INPUT
    pLonger2     : POINTER TO STRING;
    nLonger2Size : UDINT;
END_VAR
```

```
PROGRAM MAIN
VAR
    sLocal       : STRING(1023);
    bReturn      : BOOL;
END_VAR
```

```
bReturn := F_Sample(sLonger1     := sLocal,
                    pLonger2     := ADR(sLocal),
                    nLonger2Size := SIZEOF(sLocal));
```

大字符串的处理

在使用 T_MaxString 或较小的字符串变量时，您可以使用诸如 CONCAT() 或 LEN() 等字符串函数（Tc2_Standard）。

对于较大的字符串变量，您必须使用诸如 CONCAT2() 或 LEN2() 等函数（Tc2_Utilities）。

18.3.6 运行时行为

18.3.6.1 一般信息

主题：

1. 截距除以零。 [▶ 1046][++]

截距除以零

为了防止运行时错误，始终应该在除法之前进行非零查询。

静态分析：

借助静态分析规则进行检查：

- SA0040：可能除以零

以下示例的一般程序元素：

```
FUNCTION F_Sample
VAR_INPUT
    fDividend : LREAL;
    fDivisor  : LREAL;
END_VAR
VAR
    fResult   : LREAL;
END_VAR
```

反面示例：

```
fResult := fDividend / fDivisor;
```

正面示例：

```
IF fDivisor <> 0 THEN
    fResult := fDividend / fDivisor;
END_IF
```

18.3.6.2 动态内存

主题：

1. 仔细地执行动态内存分配 [▶ 1047] [++]

> **PLC_Library Tc3_DynamicMemory**
> 借助 PLC 库 Tc3_DynamicMemory，可以简化动态内存的使用。

仔细地执行动态内存分配

使用 __NEW [▶ 674]() 进行动态内存分配时应小心谨慎。例如，原因是它们会在运行时改变内存要求和数组限值。为了防止未经授权的内存访问导致程序崩溃，始终应该考虑到这些改变。

在动态内存分配中，观察哪些内存被分配以及分配的频率尤为重要。由于动态内存分配可能会影响运行时，也许还会影响内存，因此只应在特定的时间进行 1 次此类分配，例如，在启动时或在系统改造后。

另请注意，随后必须使用 __DELETE [▶ 676]() 重新发布动态分配的内存。

反面示例：

```
FUNCTION_BLOCK FB_Sample_neg
VAR
    pDynamicLRealArray : POINTER TO LREAL;  // Pointer to dynamic array
    nArrayCounter      : INT;               // Counter variable
    bInit              : BOOL;              // Initialize array only once
END_VAR
```

```
(* NON COMPLIANT:
1: If the amount of new LREALs is constant (in this case 10 elements), use a static ARRAY [0..10] OF
 LREAL.
2: If the amount of new LREALs is dynamic, do not use magic numbers. If the amount of elements (10)
is changed here, there would be no way to ensure that this number is also changed in any other code
parts.
3: If an array is created dynamically and assigned to a pointer, do not use this original pointer to
 work on the array. The start address of the array would be overwritten. *)
IF NOT bInit THEN
    pDynamicLRealArray := __NEW(LREAL, 10);                   // NON COMPLIANT: see above (1, 2)
    FillDynamicArray();
    bInit              := TRUE;
END_IF
```

方法 FillDynamicArray：

```
FOR nArrayCounter := 1 TO 10 DO                               // NON COMPLIANT: see above (2)
    pDynamicLRealArray^ := 1.25;
    pDynamicLRealArray  := pDynamicLRealArray + SIZEOF(LREAL); // NON COMPLIANT: see above (3)
END_FOR
```

正面示例：

```
FUNCTION_BLOCK FB_Sample_pos
VAR
    pDynamicLRealArray  : POINTER TO LREAL; // Pointer to dynamic array
    nDynamicArrayLen    : INT;              // Dynamic array length
    nArrayCounter       : INT;              // Counter variable
    pDynamicLRealTemp   : POINTER TO LREAL; // Temp. pointer to dynamic array
    bInit               : BOOL;             // Initialize array only once
END_VAR
```

```
IF NOT bInit THEN
    nDynamicArrayLen    := F_CalculateBufferLen();
    pDynamicLRealArray  := __NEW(LREAL, nDynamicArrayLen);     // COMPLIANT
    FillDynamicArray();
    bInit               := TRUE;
END_IF
```

方法 FillDynamicArray：

```
// Do not work on the array with the pointer that points to the start address of the dynamic array.
Use a temporary pointer for working on the array to keep the start address of the array.
pDynamicLRealTemp := pDynamicLRealArray;

// The following FOR-statement is COMPLIANT if nDynamicArrayLen is the length of array in any case.
FOR nArrayCounter := 1 TO nDynamicArrayLen DO
    pDynamicLRealTemp^ := 1.25;
```

```
    pDynamicLRealTemp  := pDynamicLRealTemp + SIZEOF(LREAL);   // COMPLIANT: Temporary pointer used
to work on the array
END_FOR
```

替代方法 FillDynamicArray：

```
// The following FOR-statement is COMPLIANT if nDynamicArrayLen is the length of array in any case.
FOR nArrayCounter := 1 TO nDynamicArrayLen DO
    pDynamicLRealArray[nArrayCounter-1] := 1.25;              // COMPLIANT: Pointer is not changed
END_FOR
```

18.3.6.3 多个任务

主题：

1. 避免对内存区进行并发访问 [▶ 1048] [++]
2. 避免对功能块进行并发访问 [▶ 1049] [++]
3. 明智地使用全局变量 [▶ 1049] [+]

有关防止并发访问的详细信息，请参见 PLC 中的多任务数据访问同步 [▶ 336] 章节。

避免对内存区进行并发访问

您应避免对内存区进行并发访问。例如，当多个任务读取和/或写入变量且至少有 1 次写入访问时，就会对内存区进行并发访问。

注意

变量值不一致

例如，如果从多个任务上下文访问全局变量，并且至少有 1 次访问也是写入访问，变量值很有可能会变得不一致。变量值不一致通常会对程序流程造成严重的后果，还可能会导致 PLC 停止运行。

静态分析：

使用以下静态分析规则进行验证：

- SA0006：多个任务中的写入访问
- SA0103：对非原子数据的并发访问

免授权版本 轻量静态分析 [▶ 139] 也包含规则 SA0006。

反面示例：

GVL_Sample：

```
VAR_GLOBAL
    nTestOutput    AT%Q*  : WORD;  // Test output to be allocated
    nGlobalCounter        : UDINT;
    nLastErrorID          : UDINT;
END_VAR
```

在任务 M 中执行的程序 Sample_neg_task_M：

```
PROGRAM Sample_neg_task_M
VAR
    nLocalTaskM  : WORD;
    fbLocalTaskM : FB_Test;
END_VAR
```

```
GVL_Sample.nTestOutput     := nLocalTaskM;                  // NON COMPLIANT: Task N reads variable
 being written by Task M whereas the accesses are not synchronized.
GVL_Sample.nGlobalCounter  := GVL_Sample.nGlobalCounter + 1; // NON COMPLIANT: Task M & Task N write
 to GVL_Sample.nGlobalCounter
GVL_Sample.nLastErrorID    := fbLocalTaskM.nErrorID;        // NON COMPLIANT: Task M & Task N write
 to GVL_Sample.nLastErrorID
```

在任务 N 中执行的程序 Sample_neg_task_N：

```
PROGRAM Sample_neg_task_N
VAR
    nLocalTaskN  : DWORD;
    fbLocalTaskN : FB_Test;
END_VAR
```

```
nLocalTaskN                := GVL_Sample.nTestOutput;        // NON COMPLIANT: Task N reads variable
 being written by Task M whereas the accesses are not synchronized.
GVL_Sample.nGlobalCounter  := 0;                             // NON COMPLIANT: Task M & Task N write
 to GVL_Sample.nGlobalCounter
GVL_Sample.nLastErrorID    := fbLocalTaskN.nErrorID;         // NON COMPLIANT: Task M & Task N write
 to GVL_Sample.nLastErrorID
```

正面示例：

GVL_Sample：

```
VAR_GLOBAL
    nTestOutput    AT%Q* : WORD;  // Test output to be allocated
    nGlobalCounter       : UDINT;
    nLastErrorID_TaskM   : UDINT; // only used in Task M
    nLastErrorID_TaskN   : UDINT; // only used in Task N
END_VAR
```

在任务 M 中执行的程序 Sample_pos_task_M：

```
PROGRAM Sample_pos_task_M
VAR
    nLocalTaskM  : WORD;
    fbLocalTaskM : FB_Test;
END_VAR
```

```
GVL_Sample.nLastErrorID_TaskM  := fbLocalTaskN.nErrorID; // writes data that is declared for task M
and only used in task M
IF (* special condition to synchronize access - see InfoSys chapter 'multitask data access' *) THEN
    GVL_Sample.nTestOutput     := nLocalTaskM;
    GVL_Sample.nGlobalCounter  := GVL_Sample.nGlobalCounter + 1;
END_IF
```

在任务 N 中执行的程序 Sample_pos_task_N：

```
PROGRAM Sample_pos_task_N
VAR
    nLocalTaskN  : DWORD;
    fbLocalTaskN : FB_Test;
END_VAR
```

```
GVL_Sample.nLastErrorID_TaskN  := fbLocalTaskN.nErrorID; // writes data that is declared for task N
and only used in task N
IF (* special condition to synchronize access - see InfoSys chapter 'multitask data access' *) THEN
    nLocalTaskN                := GVL_Sample.nTestOutput;
    GVL_Sample.nGlobalCounter  := 0;
END_IF
```

避免对功能块进行并发访问

您应避免对功能块进行并发访问。

不允许从多个任务上下文访问 1 个功能块。这会涉及到主体的调用以及方法或属性的调用。内部流程很有可能会导致变量值不一致。其后果同样是严重影响程序序列或导致 PLC 立即停止运行。

在少数例外情况下，功能块的定义允许从多个任务上下文进行访问。如果针对此类使用情况设计和开发功能块，则会进行显式记录。在所有其他情况下，不允许从多个任务上下文进行全局实例化和使用。

明智地使用全局变量

为了防止并发访问并支持数据封装，您应尽可能避免全局实例化。

同样，您应尽可能避免在功能块中使用现有的全局变量。您可以通过输入参数分配必要的数据。

19 示例

在以下章节中，您将看到一些在使用 TwinCAT 3 PLC 时可能有用的示例。

19.1 基本示例

19.1.1 第 1 步 - 状态机、计时器、触发器

描述	本 PLC 示例显示了基本语法以及一些基本指令和功能。该项目包含 1 个基于枚举的状态机，通过以下机制可以控制状态变化： • 用于时间监控的计时器（TON） • 用于检测上升沿的触发器（R_TRIG）
示例项目	https://infosys.beckhoff.com/content/1033/tc3_plc_intro/Resources/7644037771.zip
更多信息	在 PLC 文档中：您的第 1 个 TwinCAT 3 PLC 项目 在 PLC 文档中：对 PLC 项目进行编程

19.1.2 第 1 步 - 基本 PLC 元素

描述	本 PLC 示例向您展示了如何使用一些可用于构建 PLC 项目的基本 PLC 元素。在本项目中使用的元素如下： • 程序 [▶ 77] • 功能块（FB） [▶ 76] • 函数 [▶ 73] • 方法 [▶ 82] • 结构 [▶ 719] • 枚举 [▶ 721] • 全局变量列表（GVL） [▶ 66]
示例项目	https://infosys.beckhoff.com/content/1033/tc3_plc_intro/Resources/7643973259.zip
更多信息	在 PLC 文档中：对 PLC 项目进行编程

19.1.3 STRING 函数

描述	本 PLC 示例包含一系列基本 STRING 函数的示例。介绍的函数如下： • CONCAT • DELETE • FIND • INSERT • LEFT • LEN • MID • REPLACE • RIGHT
示例项目	https://infosys.beckhoff.com/content/1033/tc3_plc_intro/Resources/7644039435.zip
更多信息	在 Tc2_Standard 库文档中：STRING 函数

19.1.4 OOP 基本示例

描述	本 PLC 示例说明了面向对象的编程（OOP）的一些基本功能。它具有以下元素/功能： • 功能块（FBs） • 方法 • 属性 • FB 继承/扩展 • 接口（ITF）实现和使用
示例项目	https://infosys.beckhoff.com/content/1033/tc3_plc_intro/Resources/7644034443.zip
更多信息	见 PLC 文档：面向对象的编程

19.2 扩展示例

19.2.1 OOP 扩展示例

描述	本 PLC 示例包含控制分拣系统的面向对象的程序。该应用程序可通过集成的可视化系统进行控制。
示例项目	https://infosys.beckhoff.com/content/1033/tc3_plc_intro/Resources/7644036107.zip
更多信息	见 PLC 文档：控制分拣设备的面向对象的程序

19.2.2 字节对齐

描述	本 PLC 示例包含一些采用隐式或显式字节对齐的结构。根据所使用的字节对齐方式和结构中变量的顺序，在结构内部或末尾会插入填充字节。
示例项目	https://infosys.beckhoff.com/content/1033/tc3_plc_intro/Resources/7643974923.zip
更多信息	在 PLC 文档中：对齐方式

19.2.3 多任务数据访问同步

描述	在 PLC 中有多种同步多任务数据访问的方法。下面的 PLC 示例显示了以下选项： • Mutex 程序（TestAndSet、FB_IecCriticalSection）用于确保临界区的安全 • 通过同步缓冲区进行数据交换 • 通过 PLC 过程映像进行数据交换
示例项目	https://infosys.beckhoff.com/content/1033/tc3_plc_intro/Resources/7643978251.zip https://infosys.beckhoff.com/content/1033/tc3_plc_intro/Resources/7644032779.zip https://infosys.beckhoff.com/content/1033/tc3_plc_intro/Resources/7643976587.zip https://infosys.beckhoff.com/content/1033/tc3_plc_intro/Resources/7643979915.zip
更多信息	在 PLC 文档中：PLC 中的多任务数据访问同步

19.2.4 库文档 reStructuredText

描述	文档格式 reStructuredText 可用于在库管理器中以更具吸引力的方式显示库对象的文档。本 PLC 示例介绍了 reStructuredText 文档选项中各种结构和布局元素的语法。
示例项目	https://infosys.beckhoff.com/content/1033/tc3_plc_intro/Resources/7644044171.zip
更多信息	在 PLC 文档中：扩展 - reStructuredText

更多信息:
www.beckhoff.com/te1000

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Germany
电话号码: +49 5246 9630
info@beckhoff.com
www.beckhoff.com