手册

Automation Interface

TwinCAT 3

版本: 1.3
日期: 2020-04-15

BECKHOFF

目录

1 前言 7
1.1 文档说明 7
1.2 安全说明 7
2 概述 9
2.1 产品描述 9
2.2 版本概览 11
2.3 常见问题 14
3 安装 15
3.1 系统要求 15
3.2 安装 15
4 组态 17
4.1 快速入门 17
4.2 基础 17
4.2.1 访问 TwinCAT 组态 17
4.2.2 浏览 TwinCAT 组态 20
4.2.3 自定义 TreeItem 参数 22
4.2.4 TwinCAT 项目模板 23
4.2.5 实现 COM 消息过滤器 23
4.2.6 处理不同的 Visual Studio 版本 26
4.2.7 免打扰模式 27
4.3 最佳方案 27
4.3.1 Visual Studio 27
4.3.2 系统 34
4.3.3 ADS 45
4.3.4 PLC 48
4.3.5 I/O 60
4.3.6 TcCOM 81
4.3.7 C++ 85
4.3.8 测量 88
4.3.9 Motion 91
4.3.10 安全 93
5 API 95
5.1 引用 95
5.2 ITcSysManager 96
5.2.1 ITcSysManager 96
5.2.2 ITcSysManager::NewConfiguration 97
5.2.3 ITcSysManager::OpenConfiguration 97
5.2.4 ITcSysManager::SaveConfiguration 98
5.2.5 ITcSysManager::ActivateConfiguration 98
5.2.6 ITcSysManager::IsTwinCATStarted 98
5.2.7 ITcSysManager::StartRestartTwinCAT 99
5.2.8 ITcSysManager::LinkVariables 99
5.2.9 ITcSysManager::UnlinkVariables 99

5.2.10 ITcSysManager2::GetTargetNetId 100
5.2.11 ITcSysManager2::SetTargetNetId 100
5.2.12 ITcSysManager2::GetLastErrorMessages 101
5.2.13 ITcSysManager::LookupTreeItem 101
5.2.14 ITcSysManager3::LookupTreeItemById 102
5.2.15 ITcSysManager3::ProduceMappingInfo 102
5.2.16 ITcSysManager3::ConsumeMappingInfo 103
5.3 ITcSmTreeItem 103
5.3.1 ITcSmTreeItem 103
5.3.2 ITcSmTreeItem 项目类型 105
5.3.3 树项子类型 108
5.3.4 ITcSmTreeItem::ProduceXml 134
5.3.5 ITcSmTreeItem::ConsumeXml 135
5.3.6 ITcSmTreeItem::CreateChild 136
5.3.7 ITcSmTreeItem::DeleteChild 138
5.3.8 ITcSmTreeItem::ImportChild 138
5.3.9 ITcSmTreeItem::ExportChild 139
5.3.10 ITcSmTreeItem::LookupChild 139
5.3.11 ITcSmTreeItem::GetLastXmlError 139
5.4 ITcPlcProject 140
5.4.1 ITcPlcProject 140
5.4.2 ITcPlcProject::GenerateBootProject 141
5.5 ITcPlcPou 141
5.5.1 ITcPlcPou 141
5.5.2 IECLanguageTypes 141
5.6 ITcPlcDeclaration 142
5.7 ITcPlcImplementation 142
5.8 ITcPlcIECProject 143
5.8.1 ITcPlcIECProject 143
5.8.2 PlcImportOptions 144
5.8.3 ITcPlcIECProject::PlcOpenExport 145
5.8.4 ITcPlcIECProject::PlcOpenImport 145
5.8.5 ITcPlcIECProject::SaveAsLibrary 146
5.9 ITcPlcLibraryManager 146
5.9.1 ITcPlcLibraryManager 146
5.9.2 ITcPlcLibraryManager::AddLibrary 148
5.9.3 ITcPlcLibraryManager::AddPlaceholder 149
5.9.4 ITcPlcLibraryManager::InsertRepository 149
5.9.5 ITcPlcLibraryManager::InstallLibrary 149
5.9.6 ITcPlcLibraryManager::MoveRepository 150
5.9.7 ITcPlcLibraryManager::RemoveReference 150
5.9.8 ITcPlcLibraryManager::RemoveRepository 150
5.9.9 ITcPlcLibraryManager::ScanLibraries 151
5.9.10 ITcPlcLibraryManager::SetEffectiveResolution 151
5.9.11 ITcPlcLibraryManager::UninstallLibrary 152
5.10 ITcPlcReferences 153

5.11 ITcPlcLibrary 153
5.12 ITcPlcLibraries 153
5.12.1 ITcPlcLibraries 153
5.12.2 ITcPlcLibraries::get_Item 154
5.13 ITcPlcLibRef 154
5.14 ITcPlcPlaceholderRef 154
5.15 ITcPlcLibRepository 155
5.16 ITcPlcLibRepositories 155
5.16.1 ITcPlcLibRepositories 155
5.16.2 ITcPlcLibRepositories::get_Item 156
5.17 ITcPlcTaskReference 156
6 实例 158
6.1 示例下载 158
6.2 脚本容器 158
6.2.1 脚本容器 158
6.2.2 项目 159
6.3 CodeGenerationDemo 162
6.4 Visual Studio 插件 - PlcVersionInfo 163
7 附录 165
7.1 各类错误代码 165

1 前言

1.1 文档说明

本说明仅供熟悉适用国家标准的控制和自动化工程专家使用。
在安装和调试元器件时，必须遵循本文档及以下注意事项和说明。
技术人员应负责在每次安装和调试时使用已发布的文档。

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

本文档经过精心准备。然而，所述产品正在不断开发中。
我们保留随时修改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®、EtherCAT®、EtherCAT G®、EtherCAT G10®、EtherCAT P®、Safety over EtherCAT®、TwinSAFE®、XFC®、XTS® 和 XPlanar® 均为倍福自动化有限公司的注册商标并由公司授权使用。
本出版物中使用的其他名称可能是商标，第三方出于自身目的使用它们可能侵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
包括在其他各国家的相应专利申请或注册。

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

1.2 安全说明

安全规范

请注意以下安全说明和阐述！
可在以下页面或安装、接线、调试等区域找到产品相关的安全说明。

责任免除

所有元器件在供货时都配有适合应用的特定硬件和软件配置。禁止未按文档所述修改硬件或软件配置，德国倍福自动化有限公司不对此承担责任。

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。

符号说明

在本文档中，下列符号随安全指示或说明一起使用。必须仔细阅读并严格遵守安全说明！

⚠ 危险

严重受伤的风险！

未遵守带有此符号的安全说明将直接危及人员生命和健康。

⚠ 警告

受伤的风险！

未遵守带有此符号的安全说明将危及人员生命和健康。

⚠ 谨慎

人身伤害！

未遵守带有此符号的安全说明可能导致人员受伤。

注意

危害环境或损坏设备

未遵守带有此符号的安全说明可能危害环境或损坏设备。

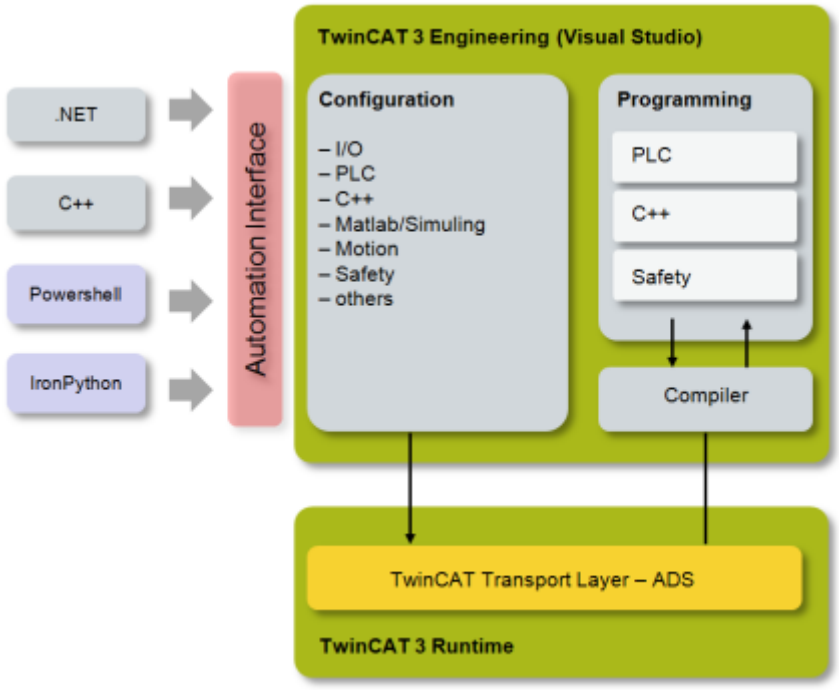
提示或指示

此符号表示该信息有助于更好地理解。

2 概述

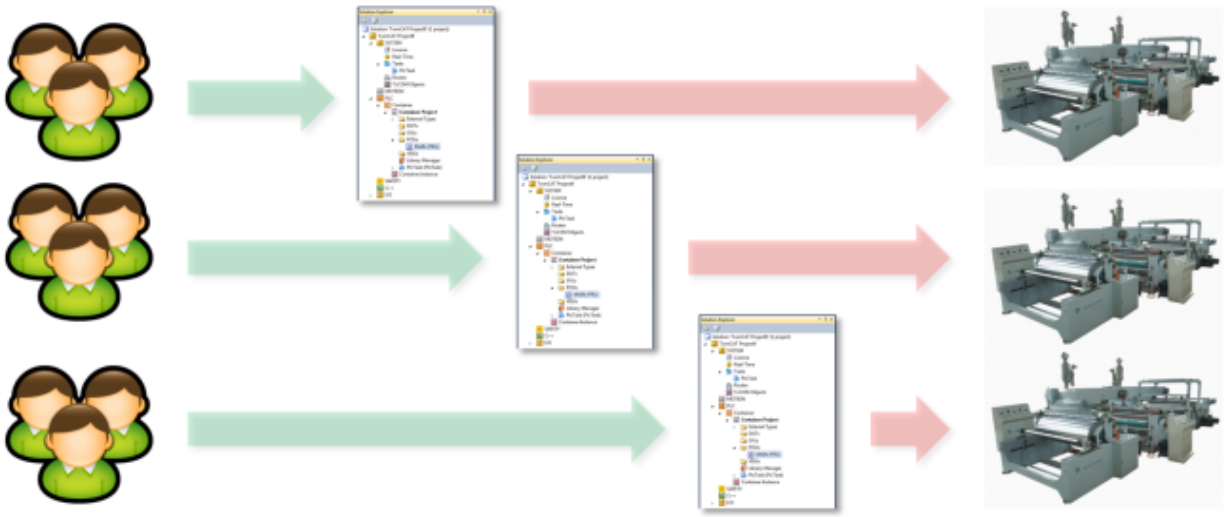
2.1 产品描述

TwinCAT 自动化接口可通过编程/脚本代码自动创建和操作 TwinCAT XAE 组态。TwinCAT 组态的自动化通过所谓的自动化接口来实现，这些接口可以通过所有 COM 支持的编程语言（例如 C++ 或 .NET）访问，也可以通过动态脚本语言访问 – 例如 Windows PowerShell、IronPython 或甚至传统的 VBscript。这些自动化接口绑定到 Visual Studio 自动化模型以及带 TwinCAT3 功能的扩展型 Visual Studio。

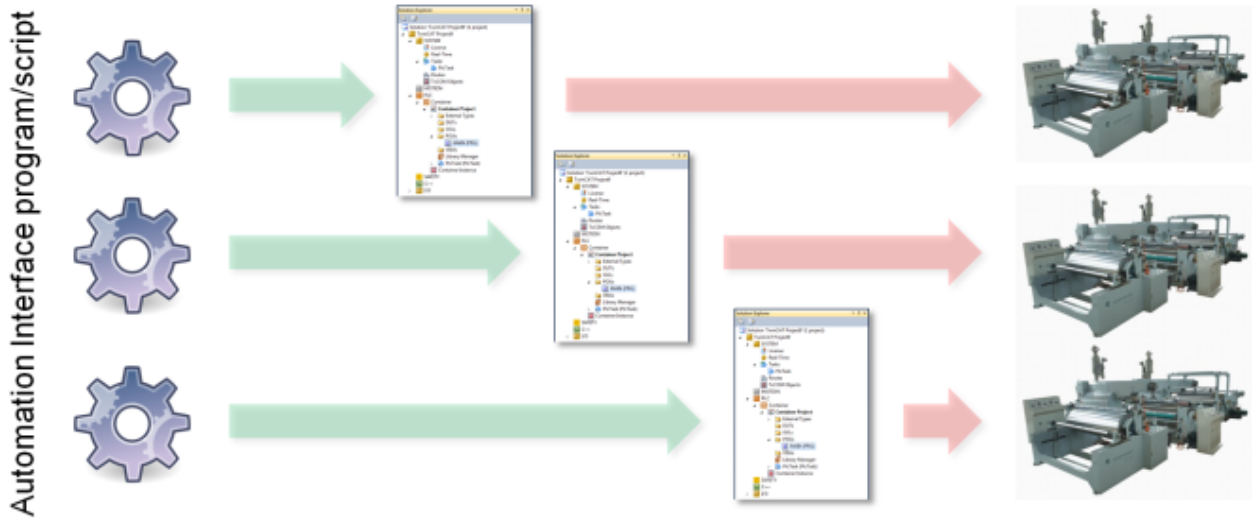


TwinCAT 自动化接口使客户能够自动组态完整的 TwinCAT 解决方案，实现高效的工程流程。

传统上，机器组态必须针对每个新项目进行手动调整，甚至必须从头开始创建，这不仅涉及大量的工程时间和成本，而且由于人为错误也很容易出错。



通过 TwinCAT 自动化接口，可以根据客户需求自动针对新环境调整 TwinCAT 组态，或甚至重新创建新的 TwinCAT 组态。



请继续阅读以下主题：

基础

主题	描述
创建/加载 TwinCAT XAE 组态 [▶ 17]	描述如何创建或打开 TwinCAT 组态
浏览 TwinCAT XAE [▶ 20]	描述如何浏览 TwinCAT 组态
自定义树项参数 [▶ 22]	描述如何访问项目的自定义参数。这对于访问 TwinCAT 树项的组态参数很重要。
实现 COM 消息筛选器 [▶ 23]	描述如何实现自身的 COM 消息筛器选来规避被拒绝的 COM 调用

最佳方案

主题	描述
创建和处理 PLC 项目 [▶ 48]	描述如何处理 PLC 项目
创建和处理 PLC POU [▶ 56]	描述如何处理 PLC 对象/代码
创建和处理 PLC 库 [▶ 52]	描述如何处理 PLC 库、存储库及占位符
创建和处理 MOTION 项目 [▶ 91]	描述如何创建 TwinCAT Motion 项目 (NC 任务、轴...)
创建和处理 EtherCAT 设备 [▶ 60]	描述如何创建 EtherCAT 设备并将其连接至 EtherCAT 拓扑结构
创建和处理 TwinCAT Measurement [▶ 88]	描述如何处理 TwinCAT Measurement 项目。
创建和处理 TcCOM 模块 [▶ 81]	描述如何处理 TcCOM 模块。
使用模板	描述生成模板和使用模板的过程。
创建和处理网络变量 [▶ 66]	描述如何创建网络变量 (发布服务器/订阅服务器变量)
创建和处理任务 [▶ 36]	描述如何创建任务并将其链接至其它对象 (PLC 项目...)
从离线组态到在线组态 [▶ 43]	一些 IO 设备需要物理地址信息才能激活组态。本文说明如何检索和设置此信息。
访问 Visual Studio 的 Error List (错误列表) 窗口 [▶ 33]	Error List (错误列表) 对于进行调试和诊断非常有用
访问 Visual Studio 中的窗口选项卡 [▶ 32]	描述如何访问 Visual Studio 窗口。
处理不同版本的 Visual Studio [▶ 26]	描述如何将不同版本的 Visual Studio 用于自动化接口
附加到正在运行的 Visual Studio 实例 [▶ 30]	显示如何附加到现有 (已运行的) Visual Studio 实例以使用自动化接口
设置 TwinCAT 目标平台 [▶ 29]	描述如何设置用于编译的 TwinCAT 目标平台。

此外，本文档还包括所有接口的完整 API 引用 [▶ 95]。如何操作与示例 [▶ 158]部分提供了脚本代码片段、组态步骤和演示项目的自由组合。还包含一个未分类且不断增加的“实际”示例列表。

还请参阅有关此

使用模板 [▶ 38]

2.2 版本概览

下表所示为 TwinCAT 2.11、TwinCAT 3.0 和 TwinCAT 3.1 相关自动化接口的可用功能概览和未来 TwinCAT 版本前瞻 (可能会有所变化)。

功能	TwinCAT 2.11	TwinCAT 3.0	TwinCAT 3.1	未来版本
常规设置				
导入组态模板	✓	✓	✓	✓
TwinCAT 系统服务处理（运行模式/组态模式）	✓	✓	✓	✓
加载/保存/创建/激活组态	✓	✓	✓	✓
支持远程 TwinCAT 目标	✓	✓	✓	✓
带过程映像的组态任务	✓	✓	✓	✓
无过程映像的组态任务	–	–	✓	✓
任务的多核支持	–	✓	✓	✓
处理 TwinCAT 授权	–	–	–	✓
路径管理				
添加/删除 ADS 路径	✓	✓	✓	✓
广播搜索	✓	✓	✓	✓
输入/输出设备				
扫描在线设备	✓	✓	✓	✓
添加/删除设备、接线盒和终端	✓	✓	✓	✓
设备、接线盒和终端参数化	✓	✓	✓	✓
EtherCAT 拓扑结构	✓	✓	✓	✓
网络变量	✓	✓	✓	✓
PLC				
变量映射，例如通过 I/O 或轴	✓	✓	✓	✓
添加/删除 PLC 项目	✓	✓	✓	✓
添加/删除 PLC POU、DUT、GVL	–	–	✓	✓
获取/设置 POU、DUT、GVL 的 PLC 代码	–	–	✓	✓
添加/删除 PLC 库	–	–	✓	✓
添加/删除 PLC 占位符	–	–	✓	✓
添加/删除 PLC 存储库	–	–	✓	✓
在存储库中添加/删除 PLC 库	–	–	✓	✓

功能	TwinCAT 2.11	TwinCAT 3.0	TwinCAT 3.1	未来版本
将 PLC 项目另存为 PLC 库	–	–	✓	✓
编译器和错误输出处理	–	–	✓	✓
PLCopen XML 导入/导出	–	–	✓	✓
编程语言：结构化文本 (ST)	–	–	✓ 2	✓ 2
编程语言：顺序功能图 (SFC)	–	–	✓ 1	✓ 1
C++				
添加/删除 C++ 项目模板	–	–	–	✓
编译器和错误输出处理	–	–	–	✓
Motion				
添加/删除 NC 任务	–	–	✓	✓
添加/删除轴	–	–	✓	✓
轴设置的参数化	–	–	✓ 3	✓ 3
变量映射，例如通过 PLC	–	–	✓	✓
TcCOM 模块				
添加/删除 TcCOM 模块	–	–	✓	✓
TcCOM 模块的参数化	–	–	✓	✓
测量				
添加/删除 TwinCAT Measurement 项目	–	–	✓	✓
添加/删除图表	–	–	✓	✓
添加/删除轴	–	–	✓	✓
添加/删除通道	–	–	✓	✓
图表、轴和通道的参数化	–	–	✓	✓
开始/停止记录	–	–	✓	✓

注意事项	
1	可通过 PLCopen XML 实现
2	可以明文或 PLCopen XML 实现源代码
3	受限。一些设置以二进制格式存储，无法编辑。

2.3 常见问题

- **什么是 TwinCAT 自动化接口？**

TwinCAT 自动化接口是从外部应用程序访问 TwinCAT 组态的接口。这使客户能够自动组态 TwinCAT。

- **我可以创建离线 TwinCAT 组态吗（不使用任何附加设备）？**

是。您可以通过手动附加（不使用“扫描设备”）所有设备后提供在线值（例如地址）来创建离线 TwinCAT 组态。有关详细信息，请参见示例 [▶ 158]页。另外，还通过如何操作示例 [▶ 79]说明如何为预先组态的 I/O 设备提供地址信息。

- **支持哪些编程和脚本语言？**

支持所有支持 COM 对象模型的编程或脚本语言。有关详细信息，请参见系统要求 [▶ 15]页。

- **通过自动化接口提供哪些 TwinCAT 功能？**

有关通过自动化接口提供哪些 TwinCAT 功能的详细信息，请参见版本概述 [▶ 11]页。

- **如果找不到适合特定设置的编程方法或属性怎么办？**

如果找不到适合特定设置的自动化接口方法或属性，可使用 TwinCAT 的导入/导出功能读/写该设置。有关详细信息，请参见有关自定义树项参数 [▶ 22]的章节。

- **我可以自动组态 TwinCAT PLC 吗？**

是。TwinCAT 3.1 将提供该功能。有关详细信息，请参见版本概述 [▶ 11]页。

- **我可以在 TwinCAT XAR（仅 Runtime）计算机上执行自动化接口代码吗？**

否。要执行自动化接口代码，需要使用 TwinCAT XAE (Engineering)，因为自动化接口直接访问 Visual Studio COM 对象以与 TwinCAT 组态通信。但是，您可以使用 TwinCAT XAE 计算机远程连接至 TwinCAT runtime 并对其进行组态。

- **我应该什么时候使用 ADS，什么时候使用自动化接口？**

这是一个很难回答的问题，因为这在很大程度上取决于您的使用目的。TwinCAT 自动化接口主要设计用于帮助客户自动组态 TwinCAT。如果您想循环读/写 IO 值或 PLC 变量，ADS API 可能更合适。

- **如果我更改 TwinCAT XAE 中的语言，例如从英语更改为德语，我需要修改自动化接口代码吗？**

所有依赖于语言的 TwinCAT XAE 项（设备、接线盒、轴、通道...）均可通过当前设置的 XAE 语言或其英文名称访问。例如，如果 XAE 语言从英语更改为德语，则术语“Channel”将在 XAE 中显示为“Kanal”，但仍可以通过自动化接口以名称“Channel”使用。为了完全兼容，建议基于英语术语构建您的自动化接口代码。

请注意：该功能仅随 TwinCAT 3.x 提供！基于 TwinCAT 2.x 的系统均依赖于语言！

- **我是机器制造商，对所有机型使用 TwinCAT 组态模板，并且只启用/禁用部分 I/O。我可以为此使用自动化接口吗？**

是。如何操作示例 [▶ 80]将为您说明如何操作。

- **我也可以创建 ADS 路径或执行广播搜索吗？**

是。有关详细信息，请参见示例 [▶ 158]与如何操作页。

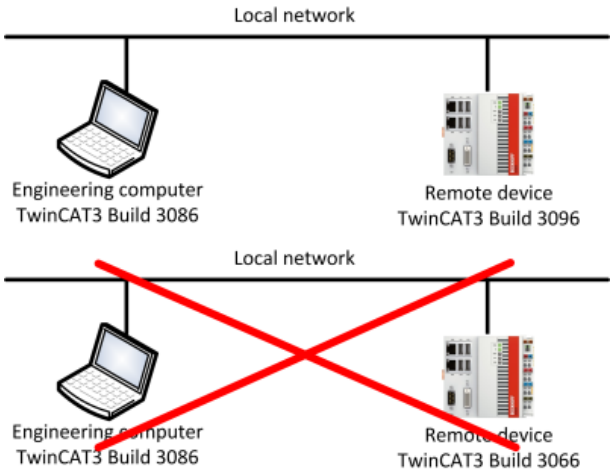
3 安装

3.1 系统要求

以下章节列出了 TwinCAT 自动化接口的所有硬件和软件要求，并提供了针对编程和脚本语言的一些建议。

硬件和软件

TwinCAT 自动化接口将通过 TwinCAT 设置自动安装。因此，需要满足与 TwinCAT 系统管理器/TwinCAT 3 XAE (Engineering) 相同的硬件和软件系统要求。使用自动化接口组态远程 TwinCAT 设备时，TwinCAT 的远程版本不得低于工程计算机上的版本。



请注意，您可以在 32 位和 64 位平台上执行自动化接口脚本，但需要确保您的程序/脚本已进行编译并以 32 位模式运行。

推荐的编程语言

对于 TwinCAT 自动化接口，推荐使用以下编程语言：

- .NET 语言，例如 C# 或 Visual Basic .NET

请注意：虽然也可以使用 C++ 实现，但强烈推荐使用上述语言之一，因为这些语言具有简单直接的 COM 相关编程概念。本文档中的大部分示例代码均基于 C#。

推荐的脚本语言

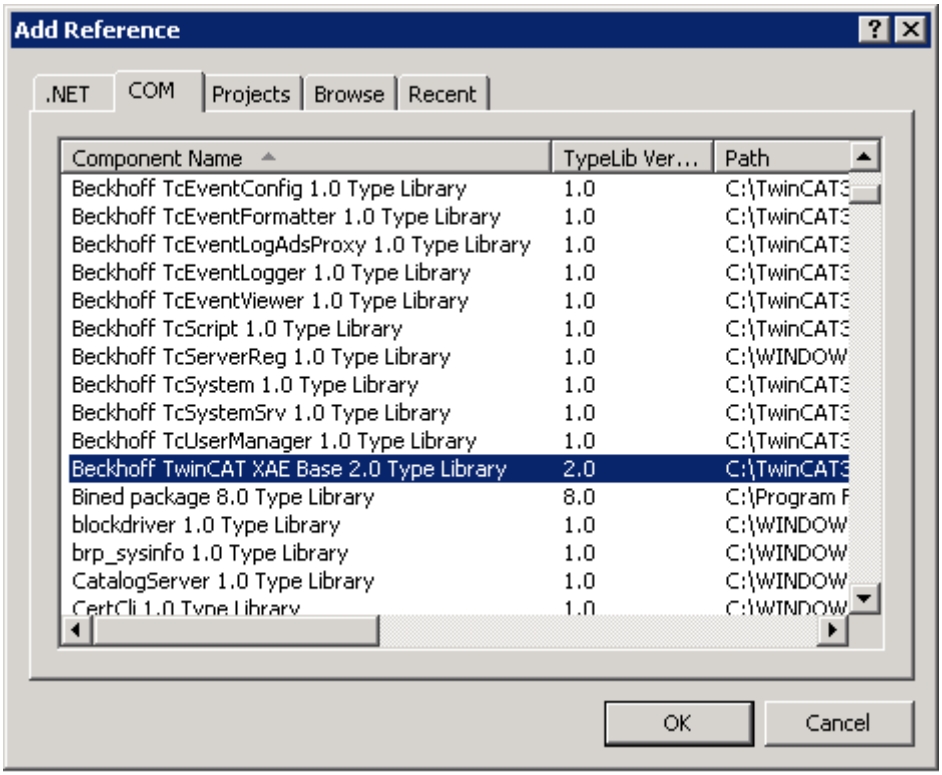
虽然 COM 支持的每种脚本语言均可用于访问 TwinCAT 自动化接口，但推荐使用 Windows Powershell，因为这种语言可以在操作系统和应用程序之间提供最佳的集成效果。请注意，至少需要 TwinCAT 3.1 Build 4020.0 才能使用动态语言，例如 Windows Powershell。

3.2 安装

TwinCAT 自动化接口需要的所有文件将在 TwinCAT 设置期间自动安装。如简介中所述，自动化接口通过 COM 与 TwinCAT 通信。所有需要的 COM 对象均自动组态，以使 COM 支持的编程和脚本语言可以访问这些对象。

在 .NET 应用程序 (C#、VB.NET...) 中使用自动化接口

要通过 .NET 应用程序访问自动化接口，需要在 Visual Studio 项目中为相应的 COM 对象 **Beckhoff TwinCAT XAE Base** 添加引用 (取决于 TwinCAT 版本，见下表)。



添加引用后，可以通过命名空间 **TCatSysManagerLib** 访问 COM 对象。

请继续阅读访问 TwinCAT 组态 [▶ 17]章节，其中更详细地说明了所有步骤。

在脚本语言中使用自动化接口

TwinCAT 自动化接口还可通过 COM 支持的脚本语言使用，例如 Windows PowerShell 或 IronPython。由于脚本语言在运行时被解释而不进行编译，因此可以随时访问操作系统中当前注册的所有 COM 对象。因此无需添加引用。

请继续阅读访问 TwinCAT 组态 [▶ 17]章节，其中更详细地说明了所有步骤。

类型库版本

在 TwinCAT 产品周期内，由于新增了功能和/或 TwinCAT 版本变化较大，可以提供上述类型库的不同版本。下表所示为所有不同类型库版本的概述。

类型库名称	类型库版本	TwinCAT 版本
Beckhoff TCatSysManager 1.1 Type Library	1.1	TwinCAT 2.11
Beckhoff TwinCAT XAE Base 2.0 Type Library	2.0	TwinCAT 3.0
Beckhoff TwinCAT XAE Base 2.1 Type Library	2.1	TwinCAT 3.1
Beckhoff TwinCAT XAE Base 3.1 Type Library	3.1	TwinCAT 3.1 Build 4020.0 及以上版本

4 组态

4.1 快速入门

由于 TwinCAT 自动化接口提供了很多可能性，因此有时可能难以了解如何开始。本快速入门提供了对 TwinCAT 自动化接口和本文档中不同文章的逐步介绍。

建议详细阅读下文：

步骤	章节	内容
1	Visual Studio ProgIDs [▶ 26]	描述如何通过 Visual Studio API 访问不同的 Visual Studio 版本
2	访问 TwinCAT 组态 [▶ 17]	描述如何使用自动化接口创建新的 TwinCAT 项目。另外介绍了 Visual Studio API 和 TwinCAT 自动化接口的共存及其如何相互关联。
3	浏览 TwinCAT 组态 [▶ 20]	描述如何使用不同的自动化接口功能浏览打开的 TwinCAT 组态。
4	COM MessageFilter 的必要性 [▶ 23]	描述为何每个自动化接口应用程序均应实现自定义 COM MessageFilter。
5	免打扰模式 [▶ 27]	描述如何将自动化接口转为“免打扰”

阅读这些基本章节后，可参阅最佳方案 [▶ 27]章节了解不同主题，例如通过自动化接口访问 PLC 或 I/O。

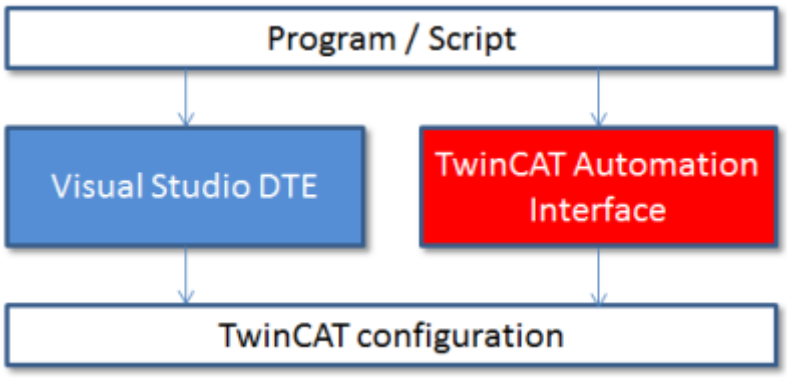
4.2 基础

4.2.1 访问 TwinCAT 组态

本章描述如何通过自动化接口创建和访问 TwinCAT XAE 组态项目。此创建流程旨在访问 TwinCAT XAE 项目(之前称为 TwinCAT 系统管理器组态)。新的 TwinCAT XAE 项目比 TwinCAT2 中的 TwinCAT 系统管理器组态更复杂，这意味着项目/组态处理概念略有改变。TwinCAT 3 支持将多个组态组合到一个 Visual Studio 解决方案容器中的额外层级结构。例如，这可用于将分布式资源的组态组织到解决方案中，或用于将 HMI 项目与系统组态打包在一起。该解决方案可将所有类型的 Visual Studio 和/或 TwinCAT XAE 项目绑定在一起。使用 TwinCAT XAE 自动化接口时，这意味着更多可能性。

基本信息

TwinCAT 3 已完全集成于 Visual Studio 中，可为用户提供标准化、最灵活的编辑器，以创建和管理 TwinCAT 项目。要创建并/或访问 TwinCAT 组态，可组合使用 Visual Studio 和 TwinCAT 自动化接口。例如：如果您想通过自动化接口创建新的 TwinCAT 组态，首先需要调用 Visual Studio API 的方法创建 Visual Studio 解决方案容器，然后使用 TwinCAT 自动化接口的方法添加 TwinCAT 项目。下面通过一些代码片段来说明这种情况。



此外，Visual Studio API（所谓的 **Visual Studio DTE** ）可为开发人员提供更多功能，例如访问错误输出窗口。 [▶ 33]有关 Visual Studio DTE 的详细信息，请访问 Microsoft MSDN 网站。

请注意：

- 在 Visual Studio 解决方案中创建新的 TwinCAT 项目时，需要指定 TwinCAT 项目模板的路径 [▶ 23]。请根据您的环境，在下面的代码片段中调整此路径。
- 以下代码片段使用了 Visual Studio DTE 对象的**动态链接**，这意味着将在应用程序运行期间确定对象的实际类型。如果您不想使用动态链接，而是事先指定数据类型，则需要在项目中包括命名空间 EnvDTE.DTE。

通过模板创建 TwinCAT 项目

请注意，需要为 COM 对象 **TcatSysManagerLib** 和 EnvDTE.DTE（Microsoft 开发）添加引用，以便使用 TwinCAT 自动化接口和 Visual Studio API。GetTypeFromProdID() 方法中使用的 ProgID 取决于应使用的 Visual Studio 版本。有关不同 ProgID 的详细信息，请参见本 [▶ 26]文中的内容。

代码片段（C#）：

```
Type t = System.Type.GetTypeFromProgID("VisualStudio.DTE.10.0");
EnvDTE.DTE dte = System.Activator.CreateInstance(t);

dte.SuppressUI = false;
dte.MainWindow.Visible = true;

if (Directory.Exists(@"C:\Temp\SolutionFolder"))
     Directory.Delete(@"C:\Temp\SolutionFolder", true);
Directory.CreateDirectory(@"C:\Temp\SolutionFolder");
Directory.CreateDirectory(@"C:\Temp\SolutionFolder\MySolution1");

dynamic solution = dte.Solution;
solution.Create(@"C:\Temp\SolutionFolder", "MySolution1");
solution.SaveAs(@"C:\Temp\SolutionFolder\MySolution1\MySolution1.sln");

string template = @"C:\TwinCAT\3.1\Components\Base\PrjTemplate\TwinCAT Project.tsproj"; //path to
project template
dynamic project = solution.AddFromTemplate(template, @"C:\Temp\SolutionFolder\MySolution1",
"MyProject");

ITcSysManager sysManager = project.Object;

sysManager.ActivateConfiguration();
sysManager.StartRestartTwinCAT();

project.Save();
solution.SaveAs(@"C:\Temp\SolutionFolder\MySolution1\MySolution1.sln");
```

代码片段（Powershell）：

您可以复制以下代码片段并粘贴到文本文件中，然后保存为“someName.ps1”。之后便可通过 Windows PowerShell 直接执行。

```
$targetDir = "C:\tmp\TestSolution"
$targetName = "TestSolution.tsp"
$template = "C:\TwinCAT\3.1\Components\Base\PrjTemplate\TwinCAT Project.tsproj"

$dte = new-object -com VisualStudio.DTE.10.0
$dte.SuppressUI = $false
$dte.MainWindow.Visible = $true

if(test-path $targetDir -pathtype container)
{
     Remove-Item $targetDir -Recurse -Force
}

New-Item $targetDir -type directory

$sln = $dte.Solution
$project = $sln.AddFromTemplate($template,$targetDir,$targetName)
$systemManager = $project.Object

$targetNetId = $systemManager.GetTargetNetId()
write-host $targetNetId

$systemManager.ActivateConfiguration()
```

```
$systemManager.StartRestartTwinCAT()

$project.Save()
$solutionPath = $targetDir + "\" + $targetName
$sln.SaveAs($solutionPath)
```

代码片段 (C++)：

在适当的头文件 (例如 stdafx.h) 中：

```
//the following #import imports EnvDTE based on its LIBID.
#import"libid:80cc9f66-e7d8-4ddd-85b6-d9e6cd0e93e2" version("10.0") lcid("0") raw_interfaces_only
named_guids
// Imports die "Beckhoff TCatSysManager 1.1 Type Library"
#import"libid:3C49D6C3-93DC-11D0-B162-00A0248C244B" version("1.1") lcid("0")
```

由于 VisualStudio 2010 (SP1) 中存在已知问题，因此生成的代理代码将不包含在 C++ 项目中。请使用 #import Known Issue 导入已知问题中所述的替代方法。

```
#include

using namespace std
using namespace TCatSysManagerLib;
using namespace EnvDTE;

int _tmain(int argc, _TCHAR* argv[])
{
    CoInitialize(NULL); // COM initialisieren
    cout << "Creating VisualSTudio.DTE.10.0 ...";

    // creating a new instance of Visual Studio
    CComPtr<_DTE> m_pDTE;
    HRESULT hr = m_pDTE.CoCreateInstance(L"VisualStudio.DTE.10.0", 0, CLSCTX_ALL);
    if (FAILED(hr)) { cout << " FAILED"; return 1; }
    cout << " created." << endl;

    // retrieves the EnvDTE.Solution-Objekt
    CComPtr<_Solution> pSolution;
    m_pDTE->get_Solution(&pSolution);
    CComBSTR strSolutionFolder(_T("C:\\SolutionFolder")); // Solution-main directory (has to exist)
    CComBSTR strSolutionName(_T("MySolution1"));
    CComBSTR strTemplatePath(_T("C:\\TwinCAT\\3.1\\Components\\Base\\PrjTemplate\\TwinCAT
Project.tsproj"));

    CComBSTR strSolutionPath; // Solution-Pfad (doesn't exist!)
    strSolutionPath=strSolutionFolder;
    strSolutionPath.Append(_T("\\"));
    strSolutionPath.Append(strSolutionName);
    strSolutionPath.Append(_T(".sln"));

    // create the solution
    hr = pSolution->Create(strSolutionFolder,strSolutionName);
    CComBSTR strProjectPath(strSolutionFolder); // project path
    strProjectPath.Append(_T("\\"));
    strProjectPath.Append(strSolutionName);
    CComBSTR strProjectName = "MyProject"; // project name // create projekt from a template
    CComPtr pProject;
    hr = pSolution-
>AddFromTemplate(strTemplatePath,strProjectPath,strProjectName,VARIANT_FALSE,&pProject);
    // Wenn z.B. Projekt bereits besteht >> error
    if (FAILED(hr)) { cout << " Project creation FAILED"; return 1; }
    cout << "Project created" << endl;

    // define project automation class (here the Coclass TcSysManager)
    CComPtr pDispatch;
    hr = pProject->get_Object(&pDispatch);

    // retrieve ITcSysManager interface
    CComPtr pSystemManager;
    hr = pDispatch.QueryInterface(&pSystemManager);

    // operate with SystemManager interface
    CComBSTR netId;
    netId = (pSystemManager->GetTargetNetId()).GetBSTR();
    cout << "TargetNetId: " << netId << endl;
    hr = pSystemManager->ActivateConfiguration();
    hr = pSystemManager->StartRestartTwinCAT();

    // save project and solution
```

```
    hr = pProject->Save(CComBSTR());
    hr = pSolution->SaveAs(strSolutionPath);
    cout << "Succeeded";
    return 0;
}
```

导入已知问题：

```
    CComPtr<_DTE> m_pDTE;
    CLSID clsid;
    CLSIDFromProgID(L"VisualStudio.DTE.10.0",&clsid);
    CComPtr punk;
    HRESULT hr = GetActiveObject(clsid,NULL,&punk); // retrieve actual instance of Visual
Studio .NET
    m_pDTE = punk;
```

请注意：

在这种情况下，ITcSysManager::NewConfiguration [▶ 97]、ITcSysManager::OpenConfiguraiton [▶ 97] 和 ITcSysManager::SaveConfiguration [▶ 98] 将创建错误消息，因为项目处理被委托给了 Visual Studio IDE（由 DTE 对象实现解决方案和项目实例）。

还请参阅有关此

- 访问 TwinCAT 组态 [▶ 18]
- 示例下载 [▶ 158]

4.2.2 浏览 TwinCAT 组态

我们已在专门的章节中介绍了如何通过 Visual Studio 自动化模型访问 TwinCAT [▶ 17]。对 TwinCAT 的引用以 ITcSysManager [▶ 96] 类型的对象表示。从现在开始，我们将讨论如何浏览 TwinCAT 组态。

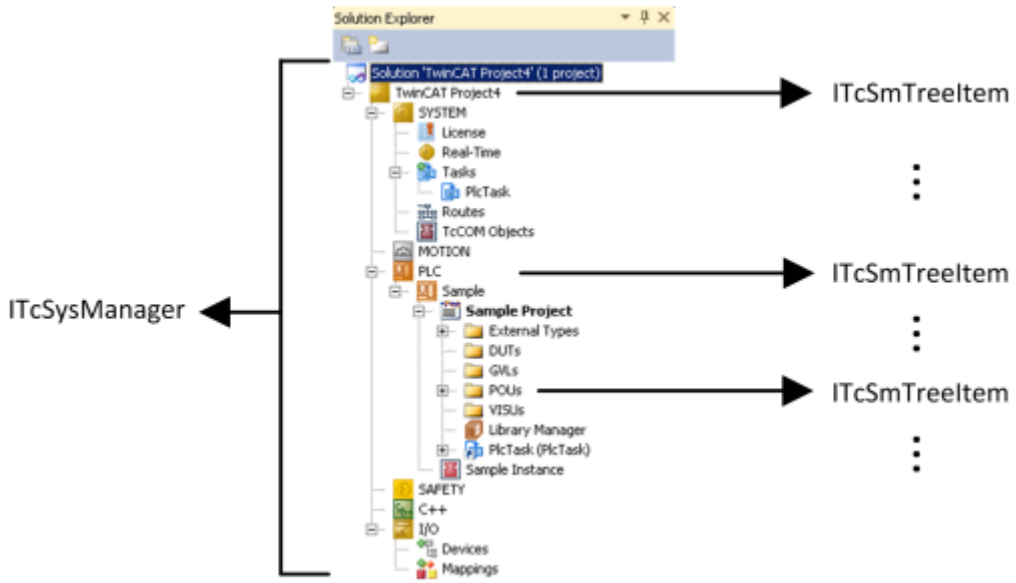
常规信息

TwinCAT 中的所有信息以树形结构排序，理解这一点非常重要。在自动化接口中，所有树节点以及 TwinCAT 组态的所有元素均由 ITcSmTreeItem [▶ 103] 接口表示。

根据应检索的信息分类，可通过不同方式浏览 TwinCAT 数据模型。

- **查找方法**通过指定的搜索条件搜索特定树项，例如树项路径
- **迭代程序**或浏览功能对检索到的树项组进行迭代

下文将对这两种方法进行讨论。



LookupMethods

查找方法始终用于整个数据模型（未过滤）。

- ITcSysManager::LookupTreeItem [▶ 101] 确定具有指定绝对路径名的树项。
- ITcSysManager3::LookupTreeItemById [▶ 102] 确定具有指定项类型和项 Id 的树项。
- ITcSmTreeItem::LookupChild [▶ 139] 确定由相对路径名指定的子树中的树项。

TwinCAT 中的各个树项均可由其唯一路径名标识。树项路径名基于其父项（其名称）的层级顺序及其自身的名称，并用抑扬符（“^”）分隔。为缩短路径名并避免语言依赖，顶级树项有专门的缩写，列于 ITcSysManager::LookupTreeItem [▶ 101] 中。

迭代程序

目前，支持三种不同类型的迭代功能：

- 浏览所有树项（未过滤）
- 浏览主树项
- 仅浏览变量/符号

浏览所有树项（未过滤）

要以未过滤的方式浏览所有树项，可使用属性 ITcSmTreeItem [▶ 103]:NewEnum。_NewEnum 对当前引用的 ITcSmTreeItem [▶ 103] 的所有子节点进行迭代。多种支持 COM 枚举器模型的编程和脚本语言（例如 .NET 语言、VB6 或脚本语言）均通过“foreach”语句使用该（COM-）属性。对于 C++ 等不支持的语言，则必须使用 _IEnumVariant 接口手动实现 foreach 循环。

建议采用这种方法迭代子节点。

示例（C#）：

```
ITcSmTreeItem parent = sysMan.LookupTreeItem("TIID^Device1^EK1100");
foreach(ITcSmTreeItem child in parent)
{
 Console.WriteLine(child.Name);
}
```

示例（C++）：

```
...
#import "C:\TwinCAT3\Components\Base\TCatSysManager.tlb" // imports the System Manager / XAE Base
type library
// uses automatically created auto-pointer (see MSDN documentation of #import command)
...

void CSysManDialog::IterateCollection(TCatSysManagerLib::ITcSmTreeItemPtr parentPtr)
{
     IEnumVARIANTPtr spEnum = parentPtr->_NewEnum;
     ULONG nReturned = 0;
     VARIANT variant[1] = {0};
     HRESULT hr = E_UNEXPECTED;

     do
     {
         hr = spEnum->Next(1, &variant[0], &nReturned);
         if(FAILED(hr))
             break;
         for(ULONG i = 0; i < nReturned; ++i)
         {
             IDispatchPtr dispatchPtr;
             IDispatch* pDispatch;
             TCatSysManagerLib::ITcSmTreeItemPtr childPtr;
             HRESULT hr;
             if(variant[0].vt == VT_DISPATCH)
             {
                 TCatSysManagerLib::ITcSmTreeItem* pChild = 0;
                 dispatchPtr.Attach((variant[0].pdispVal));
                 hr = dispatchPtr.QueryInterface(__uuidof(TCatSysManagerLib::ITcSmTreeItem),
reinterpret_cast(&pChild));
                 childPtr.Attach(pChild);
                 _bstr_t strName = pChild->GetName();
             }
         }
```

```
    }
    while(hr != S_FALSE); // S_FALSE zeigt Ende der Sammlung an
}
```

示例 (PowerShell):

```
$systemItem = $systemManager.LookupTreeItem("TIRC")
foreach($child in $systemItem)
{
write-host$child.Name
}
```

浏览主树项（已过滤）

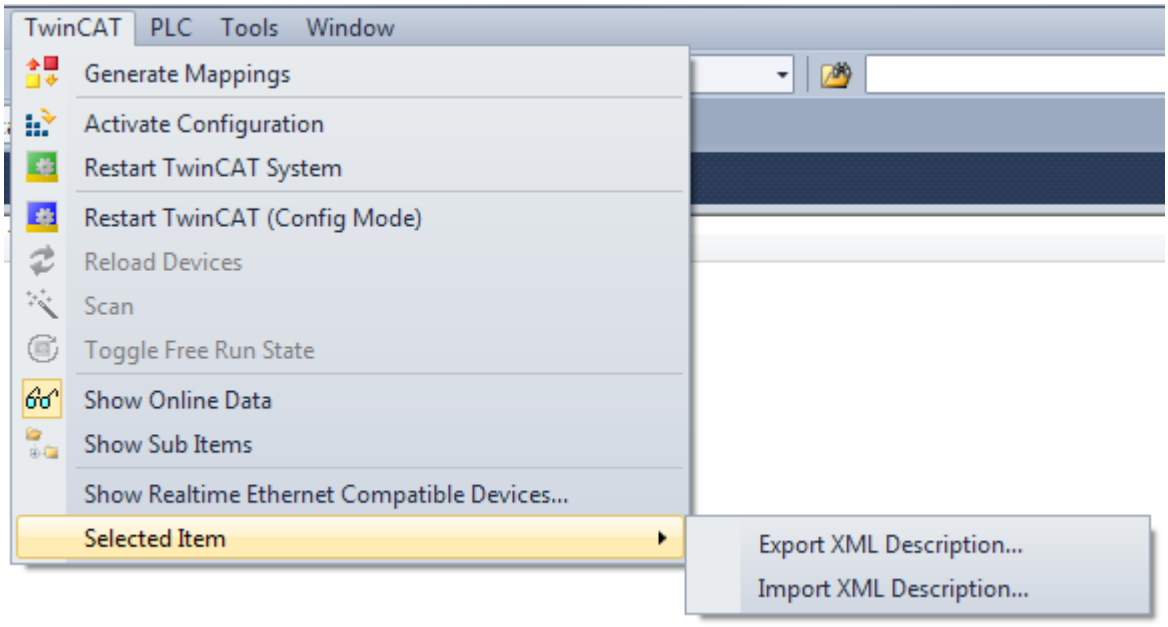
要仅浏览当前树项的主子项，使用 ITcSmTreeItem::ChildCount [▶ 103] 和 ITcSmTreeItem:Child(n) [▶ 96] 属性对。这些方法仅适用于直接子项（非递归）。

仅浏览变量/符号

要浏览变量/符号，使用 ITcSmTreeItem::VarCount(x) [▶ 103]、ITcSmTreeItem::Var(x,n) [▶ 103] 属性对。变量类型（输入变量或输出变量）的选择可以通过参数完成。

4.2.3 自定义 TreeItem 参数

ITcSmTreeItem [▶ 103] 接口由每个 TwinCAT 树项支持，并具有一个非常通用的字符。为支持所有设备、接线盒和终端及许多其他不同类型树项的规范，可以通过树项的 XML 描述访问树项的所有自定义参数。此 XML 字符串可以通过 ITcSmTreeItem::ProduceXml [▶ 134] 方法及其对应的 ITcSmTreeItem::ConsumeXml [▶ 135] 访问。此函数对的功能与 TwinCAT IDE 主菜单中的“Export XML Description...”（导出 XML 描述...）和“Import XML Description...”（导入 XML 描述...）命令相同（请参见下面的截屏）。



附图 1: TcSysMan_AutomationXmlParameters

使用此导入/导出函数集，脚本或自动化代码的许多部分可以在使用编码语言对其进行开发之前方便地进行测试和定制 - 只需导出树项数据、更改其内容并重新导入即可。

最佳方案是先导出 XML 内容、更改其内容、在 IDE 中导入，然后，如果所有操作成功进行，则将其打包至编程/脚本代码中，以便通过 ProduceXml 和 ConsumeXml 方法进行处理。

4.2.4 TwinCAT 项目模板

创建新的 TwinCAT 解决方案时，需要指定 TwinCAT 项目模板的路径 - 另请参见有关访问 TwinCAT XAE 组态 [▶ 17]的章节。

基础

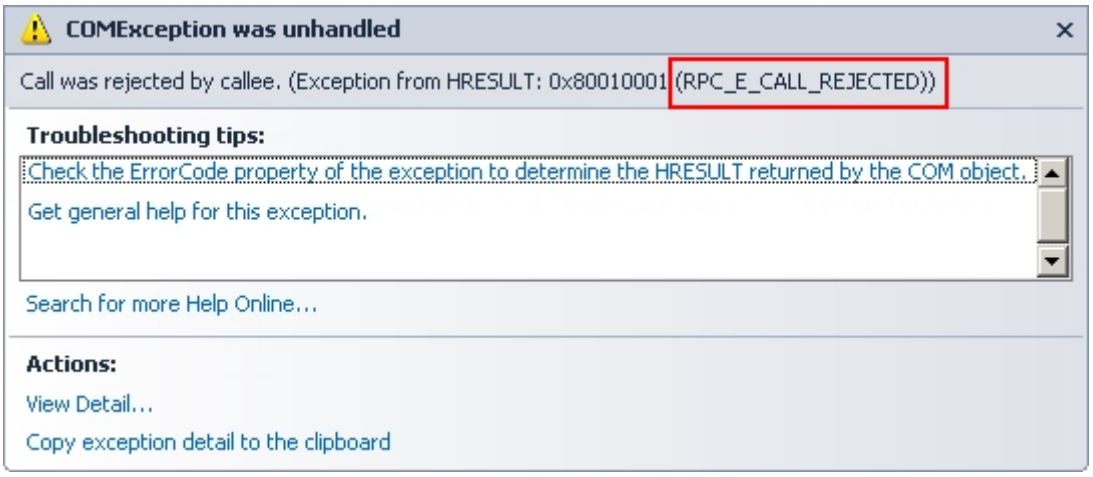
TwinCAT 版本	TwinCAT 项目模板的路径*
TwinCAT 3.0	C:\TwinCAT\3.0\Components\Base\PrjTemplate\TwinCAT Project.tsp
TwinCAT 3.1	C:\TwinCAT\3.1\Components\Base\PrjTemplate\TwinCAT Project.tsproj

* 请注意：上述路径基于默认的 TwinCAT 安装目录，如果在另一个文件夹中安装了 TwinCAT，路径可能会有所不同。

4.2.5 实现 COM 消息过滤器

消息过滤是一种机制，通过这种机制，服务器应用程序可以确定即将进行的方法调用是否以及何时可以安全地在其中一个对象上执行。COM 通常不知道应用程序的可重入性要求，并且默认情况下不过滤消息。尽管消息过滤不再像 16 位应用程序那样重要，因为消息队列的大小现在几近无限，但是您仍应实现**消息过滤**作为解决死锁的一种方法。COM 将调用接口 **IMessageFilter** 的实现来查明应用程序 (COM 服务器) 是否受阻，以便您可以对其作出反应并处理这种情况。例如，通过 COM 访问 TwinCAT XAE 时，Visual Studio 实例在仍执行先前发出的 COM 调用时拒绝其它 COM 调用。因此，客户端应用程序抛出一个 RPC_E_CALL_REJECTED 异常，并且在没有进一步干预的情况下，不会重复调用。如果客户端应用程序收到 COM 服务器拒绝 COM 调用的通知，则程序员可以通过编写自定义消息过滤器重复 COM 调用。

以下截屏显示了 Visual Studio COM 服务器在实例仍在执行先前发出的 COM 调用时抛出的典型异常。



为避免这种情况并实现对被拒绝的 COM 调用作出响应的消息过滤器，应用程序工程师需要实现 **IMessageFilter** 接口。此接口包括三种方法：

- **HandleIncomingCall()**：为即将进行的调用提供单一入口点
- **MessagePending()**：指示在 COM 等待响应以进行远程调用时收到一条消息。
- **RetryRejectedCall()**：可对被拒绝的 COM 调用作出响应。

请注意，消息过滤器仅可用于 STA 线程，每个线程仅可应用一个过滤器。多线程单元，例如控制台应用程序，不能有消息过滤器。这些应用程序需要在 STA 线程中运行才能应用消息过滤器。有关 COM 线程的详细信息，请查看本文档的附录。

以下代码片段显示如何在 C# 中使用 IMessageFilter 接口的示例。请注意，此代码也用于示例 [▶ 158]部分的许多示例中，也可以作为单独的示例下载。

```
[ComImport(), Guid("00000016-0000-0000-C000-000000000046"),
InterfaceTypeAttribute(ComInterfaceType.InterfaceIsIUnknown)]
interface IOleMessageFilter
```

```
{

[PreserveSig]
int HandleInComingCall(int dwCallType, IntPtr hTaskCaller, int dwTickCount, IntPtr lpInterfaceInfo);


[PreserveSig]
int RetryRejectedCall(IntPtr hTaskCallee, int dwTickCount, int dwRejectType);


[PreserveSig]
int MessagePending(IntPtr hTaskCallee, int dwTickCount, int dwPendingType);
}
```

下面类别实现此接口，并向其添加另外两个方法：Register() 和 Revoke()。

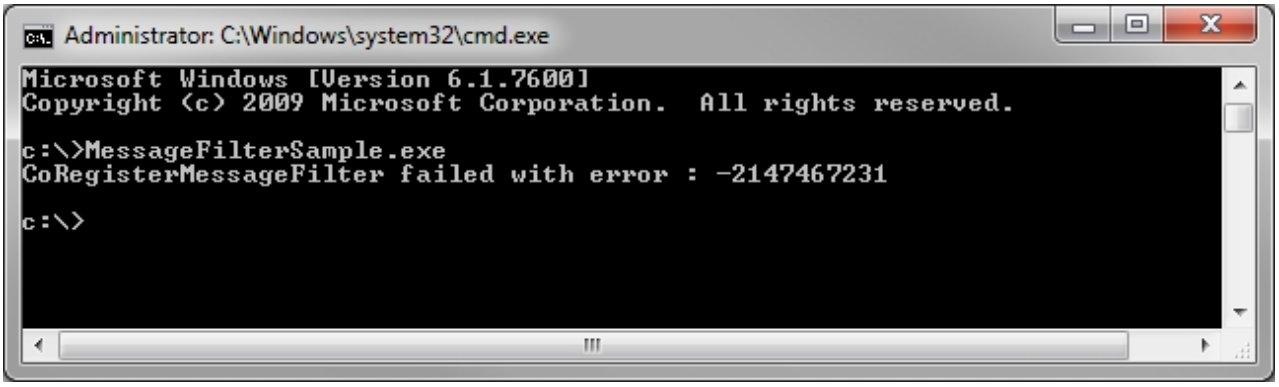
```
public class MessageFilter : IOleMessageFilter
{
public static void Register()
{
IOleMessageFilter newFilter = newMessageFilter();
IOleMessageFilter oldFilter = null;
int test = CoRegisterMessageFilter(newFilter, out oldFilter);

if (test != 0)
{
Console.WriteLine(string.Format("CoRegisterMessageFilter failed with error : {0}", test));
}
}


public static void Revoke()
{
IOleMessageFilter oldFilter = null;
int test = CoRegisterMessageFilter(null, out oldFilter);
}


int IOleMessageFilter.HandleInComingCall(int dwCallType, System.IntPtr hTaskCaller, int dwTickCount,
System.IntPtr lpInterfaceInfo)
{
//returns the flag SERVERCALL_ISHANDLED.
return 0;
}


int IOleMessageFilter.RetryRejectedCall(System.IntPtr hTaskCallee, int dwTickCount, int
dwRejectType)
{
// Thread call was refused, try again.
if (dwRejectType == 2)
// flag = SERVERCALL_RETRYLATER.
{
// retry thread call at once, if return value >=0 &
// <100.
return 99;
}
return -1;
}


int IOleMessageFilter.MessagePending(System.IntPtr hTaskCallee, int dwTickCount, int dwPendingType)
{
//return flag PENDINGMSG_WAITDEFPROCESS.
return 2;
}

// implement IOleMessageFilter interface.
[DllImport("Ole32.dll")]
private static extern int CoRegisterMessageFilter(IOleMessageFilter newFilter, outIOleMessageFilter
oldFilter);
```

应用程序工程师现在只需从另一个类调用 Register() 和 Revoke() 方法，以初始化和弃用 MessageFilter。因此，将按照 RetryRejectedCall() 方法中的定义重复被拒绝的 COM 调用。

以下代码片段显示如何在用 C# 编写的控制台应用程序中调用这些方法。如上所述，控制台应用程序默认在 MTA 线程中运行。因此，Main() 方法需要组态为在 STA 单元中运行，以便可以应用消息过滤器。

```
[STAThread]
static void Main(string[] args)
{
MessageFilter.Register();

/* ============================================================
* place COM calls for the Automation Interface here
* ...
* ...
* ============================================================ */

MessageFilter.Revoke();
}
```

如果试图将消息过滤器应用于在 MTA 单元中运行的应用程序，则在运行时尝试执行 CoRegisterMessageFilter() 方法时，将抛出以下异常：



有关不同 COM 线程模型的详细信息，请阅读有关了解和使用 COM 线程模型的 MSDN 章节。有关 IMessageFilter 接口的更多详细信息，请阅读有关 IMessageFilter 的 MSDN 文档。

以下代码片段显示如何为 Windows Powershell 实现 COM MessageFilter，也可以通过用 C# 编写自身的 Powershell Cmdlets 并使用上述代码实现 MessageFilter。

代码片段 (Powershell)：

```
function AddMessageFilterClass
{
$source = @"
namespace EnvDteUtils
{
using System;
using System.Runtime.InteropServices;

public class MessageFilter : IOleMessageFilter
{
public static void Register()
{
IOleMessageFilter newFilter = new MessageFilter();
IOleMessageFilter oldFilter = null;
CoRegisterMessageFilter(newFilter, out oldFilter);
}

public static void Revoke()
{
IOleMessageFilter oldFilter = null;
CoRegisterMessageFilter(null, out oldFilter);
}

int IOleMessageFilter.HandleInComingCall(int dwCallType, System.IntPtr hTaskCaller, int dwTickCount,
System.IntPtr lpInterfaceInfo)
{
return 0;
}

int IOleMessageFilter.RetryRejectedCall(System.IntPtr hTaskCallee, int dwTickCount, int
dwRejectType)
{
if (dwRejectType == 2)
{
return 99;
```

```
 }
return -1;
}

int IOleMessageFilter.MessagePending(System.IntPtr hTaskCallee, int dwTickCount, int dwPendingType)
{
return 2;
}

[DllImport("Ole32.dll")]
private static extern int CoRegisterMessageFilter(IOleMessageFilter newFilter, out IOleMessageFilter
oldFilter);
}

[ComImport(), Guid("00000016-0000-0000-C000-000000000046"),
InterfaceTypeAttribute(ComInterfaceType.InterfaceIsIUnknown)]
interface IOleMessageFilter
{
[PreserveSig]
int HandleInComingCall(int dwCallType, IntPtr hTaskCaller, int dwTickCount, IntPtr lpInterfaceInfo);

[PreserveSig]
int RetryRejectedCall(IntPtr hTaskCallee, int dwTickCount, int dwRejectType);

[PreserveSig]
int MessagePending(IntPtr hTaskCallee, int dwTickCount, int dwPendingType);
}
}
"@
Add-Type -TypeDefinition $source
}
```

4.2.6 处理不同的 Visual Studio 版本

本文说明如果您同时安装了两个或多个版本，如何在不同版本的 Visual Studio（例如 Visual Studio 2010 和 2012）中使用 TwinCAT 自动化接口。它包括以下主题：

- Visual Studio 程序编号
- 在自动化接口代码中指定 Visual Studio 版本

Visual Studio 程序编号

正如您在基本章节中所见，在实现自动化接口代码之前，您需要创建一个 Visual Studio DTE 对象，因为 TwinCAT 3 集成在 Visual Studio 环境中。如果您在 Engineering 计算机上安装了多个 Visual Studio 版本，并希望选择应与自动化接口代码一起使用的版本，则需要确定相应 Visual Studio 版本的程序编号 (ProgID)。ProgID 位于 Windows 注册表中的 HKEY_CLASSES_ROOT 下，格式为：VisualStudio.DTE.X.Y.

下表所示为当前支持的 Visual Studio 版本的 ProgID：

版本	ProgID
Visual Studio 2010	VisualStudio.DTE.10.0
Visual Studio 2012	VisualStudio.DTE.11.0
Visual Studio 2013	VisualStudio.DTE.12.0
Visual Studio 2015	VisualStudio.DTE.14.0
Visual Studio 2017	VisualStudio.DTE.15.0
TwinCAT XAE Shell	TcXaeShell.DTE.15.0

在自动化接口代码中指定 Visual Studio 版本

要在自动化接口代码中指定 Visual Studio 版本，需要在创建 DTE 对象时将 ProgID 用作 GetTypeDromProgID() 方法的参数。

代码片段 (C#)：

```
Type t = System.Type.GetTypeFromProgID("VisualStudio.DTE.15.0");
EnvDTE.DTE dte = (EnvDTE.DTE)System.Activator.CreateInstance(t);
```

代码片段 (Powershell)：

```
$dte = new-object-com VisualStudio.DTE.15.0
```

4.2.7 免打扰模式

有时，自动化接口脚本或程序应免打扰运行，这意味着没有任何消息框或其他可见的中断情况。尽管在大多数用例中，Visual Studio DTE 命令“dte.Visible = true/false”似乎已经足够，但 TwinCAT 自动化接口引入了一个新的免打扰模式开关，该开关自 TwinCAT 3.1 Build 4020.0 及以上版本提供。

这个新开关可按如下方式激活。

代码片段 (C#)：

```
var settings = dte.GetObject("TcAutomationSettings");
settings.SilentMode = true;
```

代码片段 (Powershell)：

```
$settings = $dte.GetObject("TcAutomationSettings")
$settings.SilentMode = $true
```

这将在使用 TwinCAT 自动化接口时禁止消息对话框。

4.3 最佳方案

4.3.1 Visual Studio

4.3.1.1 为构建过程选择项目

Visual Studio API 提供了为解决方案组态选择项目所需的所有机制。所需方法是 EnvDTE 命名空间的 SolutionBuild2 类的组成部分。以下示例演示了如何使用该类中的 BuildProject() 方法。

代码片段 (Powershell)：

```
$sln = $dte.Solution
$prj = $dte.Projects.Item(1) #SysMan Project
$sysManProjectName = $prj.FullName
$plcPrjProjectName = PathToPlcProjFile
$sln.SolutionBuild.BuildProject("Release|TwinCAT RT (x64)",$sysManProjectName,$true)
$sln.SolutionBuild.BuildProject("Release|TwinCAT RT (x64)",$plcPrjProjectName,$true)
```

占位符“pathToPlcProjFile”表示 *.plcproj 文件的完整路径，该文件代表 TwinCAT 3 PLC 项目。

4.3.1.2 访问 TeamFoundationServer 源管理

本章概述了如何通过使用 Visual Studio 提供的相应 DLL 以编程方式访问 Microsoft Team Foundation Server (TFS)。本文档旨在为您提供便利，并使您能够更容易地开始将 TwinCAT 自动化接口代码与 TFS DLL 组合使用。有关 TFS API 的详细信息，强烈建议查阅相应的 MSDN 章节。

本文档包含以下主题：

- 连接至 TFS 服务器
- 连接至 TeamProjects
- 使用工作区
- 创建工作文件夹
- 获取项目的最新版本
- 获得待定更改的列表
- 签入和签出
- 撤销操作

本文档中使用了以下 TFS API DLL：

- Microsoft.TeamFoundation.Client
- Microsoft.TeamFoundation.VersionControl.Client

连接至 TFS 服务器

Microsoft TFS API 可让您将包含 TwinCAT 自动化接口代码的应用程序连接至远程服务器上的开发存储库。连接需要一个字符串来描述远程服务器的地址、端口和集合，例如 http://your-tfs:8080/tfs/DefaultCollection。

连接前，请定义 Uri 类和 TfsConfigurationServer 类的实例。将远程服务器链接分配至组态服务器：

代码片段 (C#)：

```
string tfsServerUrl = "http://your-server:8080/tfs/DefaultCollection";
Uri configurationServerUri = new Uri(tfsServerUrl);
TfsTeamProjectCollection teamProjects =
TfsTeamProjectCollectionFactory.GetTeamProjectCollection(configurationServerUri);
VersionControlServer verControlServer = teamProjects.GetService<VersionControlServer>();
```

连接至 TeamProjects

通过执行以下代码片段，可以在服务器上获得团队项目列表或某个指定团队项目：

代码片段 (C#)：

```
// Get all Team projects
TeamProject[] allProjects = verControlServer.GetAllTeamProjects(true);

// Get specific Team Project
TeamProject project = verControlServer.TryGetTeamProject("TeamProjectName");
```

使用工作区

Team foundation 处理工作区，其中包括工作文件夹映射。这些映射将服务器端文件夹链接到硬盘上的本地文件夹。此外还将用户名和计算机名称存储为工作区名称的一部分。执行任何版本的控制任务前，都必须有一个工作区，因为要在这里存储有关文件、版本和待处理更改列表的信息。

首先，TFS Client API 提供一个 *Workspace* 类，其存储上述信息，并提供大量与文件和文件夹进行交互的方法。假设要为名为 *folderName* 的文件夹创建工作区，则创建一个包含计算机名称和文件夹的字符串：

代码片段 (C#)：

```
// Specify workspace name for later use
String workspaceName = String.Format("{0}-{1}", Environment.MachineName, "Test_TFSAPI");

// Create new workspace
Workspace newWorkspace = verControlServer.CreateWorkspace(workspaceName,
verControlServer.AuthorizedUser);
```

现在，服务器端文件夹与通过 *folderName* 在 *workspace* 下指定的本地文件夹之间可能存在映射。要获取现有工作区或删除工作区，仅需执行以下方法：

代码片段 (C#)：

```
// Delete workspace
bool workspaceDeleted = verControlServer.DeleteWorkspace(workspaceName,
verControlServer.AuthorizedUser);

// Get existing workspace
Workspace existingWorkspace = verControlServer.GetWorkspace(workspaceName,
verControlServer.AuthorizedUser);
```

创建工作文件夹

要连接至指定的项目文件夹，需要一个相对于根目录的项目路径。

例如，如果项目存储于：*your-server\Development\TcSampleProjectX*，则文件夹的相对路径是 *$\Development\TcSampleProjectX*。

要将此文件夹与服务器上的项目匹配，可以对服务器上的所有已注册项目进行迭代。已注册项目是在下面找到的项目集合的根。

代码片段 (C#)：

```
// Create mapping between server and local folder
string serverFolder = String.Format("$/{0}", teamProject.Name + "/Folder/SubFolder");
string localFolder = Path.Combine(@"C:\tfs", workspaceName);
WorkingFolder workingFolder = new WorkingFolder(serverFolder, localFolder);
existingWorkspace.CreateMapping(workingFolder);
```

获取项目的最新版本

如前所述，Workspace 类提供一组丰富的方法来执行代码中的 TFS 命令。但在演示之前，请参见以下创建工作文件夹中某个项目链接的示例。创建相对路径：

代码片段 (C#)：

```
String existingItemPath = "$/Developement/TcSampleProject/Examples/Samples00/TestPlc/POUs/
MAIN.TcPOU";
```

或创建绝对路径：

```
String existingItemPath = C:/tfs/Developement/TcSampleProject/Examples/TestPlc/POUs/MAIN.TcPOU";
```

要获取示例中项目的最新版本，添加以下代码行：

代码片段 (C#)：

```
String existingItemPath = "$/TeamProjectName/Folder/SubFolder";
GetRequest itemRequest = new GetRequest(new ItemSpec(existingItemPath, RecursionType.Full),
VersionSpec.Latest);
existingWorkspace.Get(itemRequest, GetOptions.GetAll);
```

在此，*RecursionType.Full* 对项目节点下的所有项目执行请求，但您可以根据应用程序的要求及现有项目层级进行选择。有关详细信息，请参见 MSDN 上的 API 文档。

获得待定更改的列表

还可以使用以下代码行获得针对项目或项目集合的待更改列表：

代码片段 (C#)：

```
PendingChange[] pendingChanges = existingWorkspace.GetPendingChanges(existingItemPath,
RecursionType.Full);
```

对于项目集合，请将带 Item[] 的重载方差视为参数。

签入和签出

您可以签出要编辑的项目或项目集合：

代码片段 (C#)：

```
int checkoutResult = existingWorkspace.PendEdit(existingItemPath, RecursionType.Full);
```

通过以下方式签入项目的待更改集合：

代码片段 (C#)：

```
int checkinResult = workspace.CheckIn(pendingChanges, usercomment);
```

撤销操作

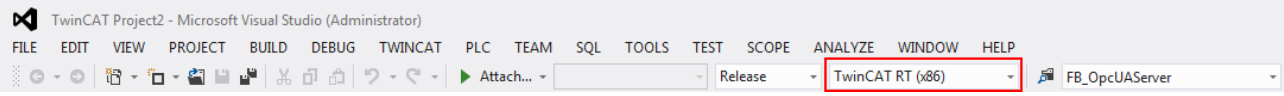
要撤销操作，仅需执行：

代码片段 (C#)：

```
int undoResult = existingWorkspace.Undo(existingItemPath, RecursionType.Full);
```

4.3.1.3 设置 TwinCAT 目标平台

本文描述如何通过自动化接口代码设置 TwinCAT 目标平台。目标平台确定应为哪个目标编译 TwinCAT 组态，例如 TwinCAT x86 或 TwinCAT x64，通常在 Visual Studio 的 TwinCAT XAE 工具栏中设置。



以下代码片段假设您已有链接至 TwinCAT 组态的 DTE 实例。它检查当前设置的目标平台并将其重新设置到另一个平台。

代码片段 (C#)：

```
ITcSysManager systemManager = pro.Object;
ITcSysManager7 sysManPlatform = (ITcSysManager7) systemManager;
ITcConfigManager configManager = (ITcConfigManager) sysManPlatform.ConfigurationManager;
if (configManager.ActiveTargetPlatform == "TwinCAT RT (x86)")
configManager.ActiveTargetPlatform = "TwinCAT RT (x64)";
else
configManager.ActiveTargetPlatform = "TwinCAT RT (x86)";
```

代码片段 (Powershell)：

```
$configManager = $systemManager.ConfigurationManager
if ($configManager.ActiveTargetPlatform -eq "TwinCAT RT (x86)"){
  $configManager.ActiveTargetPlatform = "TwinCAT RT
(x64)"} else {$configManager.ActiveTargetPlatform = "TwinCAT RT (x86)"}
```

注意 **本文描述如何更改目标硬件平台。如果您想更改要编写 TwinCAT 组态的实际远程目标，请使用 ITcSysManager::SetTargetNetId() 方法。**

以下代码片段显示如何获取访问 Visual Studio 组态管理器的权限，并启用或禁用用于构建过程的单个 TwinCAT 子项目 (PLC、C++…)。代码片段假设当前打开的 TwinCAT 项目命名为“TwinCAT Project1”且包含 PLC 项目“Untitled1”。然后，它禁用用于构建过程的 PLC 项目。

代码片段 (C#)：

```
EnvDTE.SolutionContexts solutionContexts =
solution.SolutionBuild.ActiveConfiguration.SolutionContexts;
foreach (EnvDTE.SolutionContext solutionContext in solutionContexts)
{
switch(solutionContext.ProjectName)
{
case "TwinCAT Project13\\Untitled1\\Untitled1.plcproj":
solutionContext.ShouldBuild = false;
break;
}
}
```

代码片段 (Powershell)：

```
$solutionContexts = $sln.SolutionBuild.ActiveConfiguration.SolutionContexts
foreach ($solutionContext in $solutionContexts)
{
  if($solutionContext.ProjectName -eq "TestSolution\\Untitled1\\Untitled1.plcproj")
  {
    $solutionContext.ShouldBuild = $false
  }
}
```

4.3.1.4 附加到现有的 Visual Studio 实例

以下代码片段演示如何附加到现有（已在运行）的 Visual Studio 实例。已在 C# 中编写代码片段。

示例包括三种相互依存的不同方法。当然，您可以相应地对其进行更改，使其更适合您的应用程序环境。下表更详细地说明了各种方法。

方法	描述
getRunningObjectTable()	查询运行对象表 (ROT)，以获取表中所有正在运行的进程的截屏。将找到的所有进程返回到散列表中，然后 getIdeInstances() 使用这些进程进一步过滤 DTE 实例。
getIdeInstances()	搜索 ROT 截屏中的 DTE 实例，并返回一个包含所有已找到实例的散列表。然后，可通过 attachToExistingDte() 方法使用此散列表选择单个 DTE 实例。 您可以根据更具体的 Visual Studio progId [▶ 26] 更改对 candidateName 的查询，例如用于查询 Visual Studio 2012 实例 的“VisualStudio.DTE.11.0”。
attachToExistingDte()	使用 getIdeInstances() 方法根据解决方案路径选择 DTE 实例，找到后将新的 DTE 对象附加至此实例。

getRunningObjectTable()

```
public static Hashtable GetRunningObjectTable()
{
    Hashtable result = new Hashtable();
    int numFetched;
    UCOMIRunningObjectTable runningObjectTable;
    UCOMIEnumMoniker monikerEnumerator;
    UCOMIMoniker[] monikers = new UCOMIMoniker[1];
    GetRunningObjectTable(0, out runningObjectTable);
    runningObjectTable.EnumRunning(out monikerEnumerator);
    monikerEnumerator.Reset();
    while (monikerEnumerator.Next(1, monikers, out numFetched) == 0)
    {
        UCOMIBindCtx ctx;
        CreateBindCtx(0, out ctx);
        string runningObjectName;
        monikers[0].GetDisplayName(ctx, null, out runningObjectName);
        object runningObjectVal;
        runningObjectTable.GetObject(monikers[0], out runningObjectVal);
        result[runningObjectName] = runningObjectVal;
    }
    return result;
}
```

请注意，您需要明确引用 CreateBindCtx() 方法作为 ole32.dll 的 DllImport，例如：

```
[DllImport("ole32.dll")]
private static extern int CreateBindCtx(uint reserved, out IBindCtx ppbc);
```

<b />

getIdeInstances()

```
public static Hashtable GetIDEInstances(bool openSolutionsOnly, string progId)
{
  Hashtable runningIDEInstances = new Hashtable();
  Hashtable runningObjects = GetRunningObjectTable();
  IDictionaryEnumerator rotEnumerator = runningObjects.GetEnumerator();
  while (rotEnumerator.MoveNext())
  {
    string candidateName = (string)rotEnumerator.Key;
    if (!candidateName.StartsWith("!" + progId))
      continue;
    EnvDTE.DTE ide = rotEnumerator.Value as EnvDTE.DTE;
    if (ide == null)
      continue;
    if (openSolutionsOnly)
    {
      try
      {
        string solutionFile = ide.Solution.FullName;
        if (solutionFile != String.Empty)
          runningIDEInstances[candidateName] = ide;
      }
      catch { }
    }
    else
      runningIDEInstances[candidateName] = ide;
```

```
  }
  return runningIDEInstances;
}
```

attachToExistingDte()

```
public EnvDTE.DTE attachToExistingDte(string solutionPath)
{
EnvDTE.DTE dte = null;
Hashtable dteInstances = GetIDEInstances(false, progId);
IDictionaryEnumerator hashtableEnumerator = dteInstances.GetEnumerator();

while (hashtableEnumerator.MoveNext())
{
EnvDTE.DTE dteTemp = hashtableEnumerator.Value as EnvDTE.DTE;
if (dteTemp.Solution.FullName == solutionPath)
{
Console.WriteLine("Found solution in list of all open DTE objects. " + dteTemp.Name); dte = dteTemp;
}
}
return dte;
}
```

4.3.1.5 访问 Visual Studio 中的窗口选项卡

Visual Studio 自动化接口提供访问 Visual Studio 中活动窗口的方法和属性。以下章节将显示一些示例代码。但是，我们还建议查看 Microsoft Developer Network (MSDN) 网页，了解有关 Visual Studio 对象模型的更多详细信息。本章提供以下任务的示例代码：

- 访问活动窗口
- 关闭活动窗口
- 通过 TwinCAT PLC 组态打开窗口

访问活动窗口

DTE 接口提供名为“ActiveWindow”的属性，该属性将 Visual Studio 中当前的活动窗口作为 EnvDTE.Window 类型的对象返回。

代码片段 (C#)：

```
EnvDTE.Window activeWin = dte.ActiveWindow;
```

代码片段 (Powershell)：

```
$activeWin = $dte.ActiveWindow
```

关闭活动窗口

根据客户应用程序，在通过自动化接口继续 TwinCAT 组态之前，可能需要关闭 Visual Studio 中的所有活动窗口。以下代码片段关闭除“Solution Explorer”（解决方案资源管理器）之外的所有活动窗口。

代码片段 (C#)：

```
try
{
while (!dte.ActiveWindow.Caption.Contains("Solution Explorer"))
dte.ActiveWindow.Close();
}
catch (InvalidOperationException ex)
{
// use DTE.Quit() to close main window
}
```

通过 TwinCAT PLC 组态打开窗口

还可通过 TwinCAT PLC 打开窗口，例如可视化窗口。以下代码片段可通过 PLC 项目“Untitled1”打开现有的 TwinCAT 可视化窗口，并使其成为当前的活动窗口。

代码片段 (C#)：

```
string fileName = @"C:\TwinCAT Project1\TwinCAT Project1\Untitled1\VISUs\Visu.TcVIS";
dte.ItemOperations.OpenFile(fileName);
```

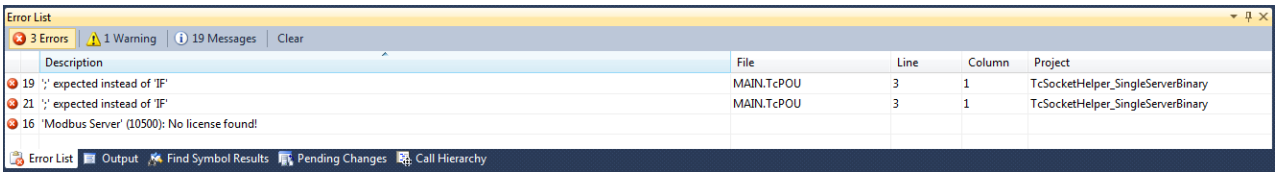
代码片段 (Powershell)：

```
$fileName = @"C:\TwinCAT Project1\TwinCAT Project1\Untitled1\VISUs\Visu.TcVIS"
dte.ItemOperations.OpenFile($fileName)
```

4.3.1.6 访问 Visual Studio 的 Error List（错误列表）窗口

本章描述如何访问 Visual Studio 的 Error List（错误列表）窗口。由于 TwinCAT 3 集成于 Visual Studio 框架中，阅读此窗口的内容可能对于进行调试和诊断至关重要。要阅读此窗口的内容，需要访问 Visual Studio COM 接口。由于访问 TwinCAT XAE（例如，如访问 TwinCAT 组态 [▶ 17]章节所述）时也需要此 COM 接口，因此无需向程序代码中添加任何其它引用。

以下文档描述如何访问 Error List（错误列表）窗口，此窗口可能非常有用，例如想要编译 PLC 项目并检查编译过程中是否有错误消息时。Error List（错误列表）窗口显示 Visual Studio（及 TwinCAT）开发环境中各种功能的状态消息。例如，其中包括编译项目或发布授权时出现的错误。



阅读此窗口内容至关重要，例如，在编译 PLC 项目时检查是否发生错误。以下代码片段显示如何使用 C# 应用程序阅读此窗口的内容。

代码片段 (C#)：

```
ErrorItems errors = dte.ToolWindows.ErrorList.ErrorItems;
for (int I = 0; i < errors.Count; i++)
{
  ErrorItem item = errors.Item(i);
}
```

请注意，“ToolWindows”属性在命名空间 EnvDTE80.DTE2 中可用。

代码片段 (Powershell)：

```
$errors = $dte.ToolWindows.ErrorList.ErrorItems
for ($i=0; $i -lt $errors.Count; $i++)
{
  $item = $errors.Item($i);
}
```

如您所见，ErrorItems 属性可将所有项目的集合返回到错误列表中，所有项目均为 ErrorItem 类型。例如，您可以使用 Count 属性对该集合进行迭代。所有 ErrorItem 对象都有以下属性，这些属性与 ErrorList 窗口中的列相同（与上面的截屏相比）：

属性	描述
描述	描述错误消息。
文件名	发生错误的文件名。
行	发生错误的行。
列	发生错误的列。
项目	发生错误的项目。

请注意，最后四项属性并非始终使用，也并非在所有情况下都有意义。例如，查看上面的截屏时，错误消息“No license found”（未找到授权）未绑定到指定文件、行或列，因为这是一条常规 TwinCAT 错误消息。

4.3.2 系统

4.3.2.1 打开和激活现有组态

要激活先前创建的组态，需要创建 TwinCAT XAE 实例，并加载和激活此组态。

步骤

ProgId “**VisualStudio.DTE.10.0**”用于创建 Visual Studio 实例。通过 Visual Studios DTE 对象，可对 Visual Studio 进行完全控制。创建 ITcSysManager [▶ 96] 接口（此处为“*sysMan*”实例）的步骤请参见访问 TwinCAT 组态 [▶ 17]章节。

示例 (CSharp)：

请注意，在本示例中，需要为项目中的“Microsoft Developer Environment 10.0”和“Beckhoff TwinCAT XAE Base”添加引用。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using EnvDTE100;
using System.IO;
using TCatSysManagerLib;

namespace ActivatePreviousConfiguration
{
class Program
{
static void Main(string[] args)
{
Type t = System.Type.GetTypeFromProgID("VisualStudio.DTE.10.0");
EnvDTE.DTE dte = (EnvDTE.DTE)System.Activator.CreateInstance(t);
dte.SuppressUI = false;
dte.MainWindow.Visible = true;

EnvDTE.Solution sol = dte.Solution;
sol.Open(@"C:\Temp\SolutionFolder\MySolution1\MySolution1.sln");

EnvDTE.Project pro = sol.Projects.Item(1);

ITcSysManager sysMan = pro.Object;

sysMan.ActivateConfiguration();
sysMan.StartRestartTwinCAT();
}
}
}
```

示例 (PowerShell)：

```
$prjDir = "C:\tmp\TestSolution\"
$prjName = "TestSolution.sln"
$prjPath = $prjDir += $prjName
$dte = new-object -com VisualStudio.DTE.10.0
$dte.SuppressUI = $false
$dte.MainWindow | %{$_.gettype().InvokeMember("Visible","SetProperty",$null,$_,$true)}

$sln = $dte.Solution
$sln.Open($prjPath)

$project = $sln.Projects.Item(1)
$systemManager = $project.Object

$systemManager.ActivateConfiguration()
$systemManager.StartRestartTwinCAT()
```

示例 (VBScript)：

```
dim dte,sln,proj,sysMan
set dte = CreateObject("VisualStudio.DTE.10.0")
set sln = dte.Solution
call sln.Open("C:\SolutionFolder\MySolution1.sln")
set proj = sln.Projects(1)
```

```
set sysMan = proj.Object
call sysMan.ActivateConfiguration
call sysMan.StartRestartTwinCAT
```

4.3.2.2 通过 TwinCAT 目标打开现有项目

本文描述如何通过已连接的 TwinCAT 目标打开现有的 TwinCAT 项目。需要提供目标，这意味着必须存在 ADS 路径才能通过目标检索项目。

以下代码片段显示如何通过已连接的 Target Runtime 检索 TwinCAT 项目。

代码片段 (C#)：

```
dte.ExecuteCommand("File.OpenProjectFromTarget", "CX-123456 C:\\ProjectDir ProjectName");
```

代码片段 (Powershell)：

```
$dte.ExecuteCommand("File.OpenProjectFromTarget", "CX-123456 C:\\ProjectDir ProjectName");
```

Visual Studio API 中的 ExecuteCommand() 方法允许触发 TwinCAT 命令 (File.OpenProjectFromTarget)，以通过已连接的 TwinCAT 目标打开项目。ExecuteCommand() 方法的第二个参数中将包括三个参数。这三个参数是：目标路径名、本地项目目录 (项目文件应存储的位置)、本地项目名。三个参数之间用空格分隔。

4.3.2.3 创建和处理变量映射

使用 TwinCAT 自动化接口的常见情况是自动创建变量映射，例如在 PLC 输入/输出变量与其对应的 I/O 对应项之间。自动化接口提供多种简化创建、删除或保存变量映射的任务的方法。下文简要介绍了这些方法，并提供了有关如何对其进行使用的示例。包括以下主题：

- 常规信息
- 链接变量
- 解除变量链接
- 获取/设置所有变量映射
- 删除所有变量映射

常规信息

变量映射可能发生在 TwinCAT 项目中的不同输入/输出树项之间，例如：

- PLC 输入/输出变量和 I/O (反之亦然) 之间
- PLC 输入/输出变量和 NC (反之亦然) 之间
- PLC 输入/输出变量和 TcCOM 对象 (反之亦然) 之间
- …

本文中的信息描述可用于所有这些用例的自动化接口机制。

链接变量

从自动化接口角度来看，变量映射始终在两个树项之间执行，例如在一个 PLC 输入/输出变量与其相应的 I/O 对应项之间。要链接两个树项，自动化接口提供方法 ITcSysManager::LinkVariables()，将给定的源树项与给定的目标树项链接起来。

代码片段 (C#)：

```
string source = "TIPC^PlcProj^PlcProj Instance^PlcTask Inputs^bIn";
string destination = "TIID^EtherCAT^EK1100^EL1004^Channel 1^Input";
systemManager.LinkVariables(source, destination);
```

代码片段 (Powershell)：

```
$source = "TIPC^PlcProj^PlcProj Instance^PlcTask Inputs^bIn"
$destination = "TIID^EtherCAT^EK1100^EL1004^Channel 1^Input"
$systemManager.LinkVariables($source, $destination)
```

解除变量链接

与链接变量相似，自动化接口提供方法 ITcSysManager::UnlinkVariables()，将解除给定源树项与给定目标树项之间的链接。

代码片段 (C#)：

```
string source = "TIPC^PlcProj^PlcProj Instance^PlcTask Inputs^bIn";
string destination = "TIID^EtherCAT^EK1100^EL1004^Channel 1^Input";
systemManager.UnlinkVariables(source, destination);
```

代码片段 (Powershell)：

```
$source = "TIPC^PlcProj^PlcProj Instance^PlcTask Inputs^bIn"
$destination = "TIID^EtherCAT^EK1100^EL1004^Channel 1^Input"
$systemManager.UnlinkVariables($source, $destination)
```

保存/还原所有变量映射

要保存或还原 TwinCAT 项目中的所有变量映射，可以使用方法 ITcSysManager2::ProduceMappingInfo() 和 ITcSysManager2::ConsumeMappingInfo()。前者读取 TwinCAT 项目中的所有变量映射，并将其以 XML 结构返回，之后可以使用后一种方法重新导入。

代码片段 (C#)：

```
ITcSysManager2 systemManager2 = (ITcSysManager2)systemManager;
string mappingInfo = systemManager2.ProduceMappingInfo();
systemManager2.ConsumeMappingInfo(mappingInfo);
```

代码片段 (Powershell)：

```
$mappingInfo = $systemManager.ProduceMappingInfo()
$systemManager.ConsumeMappingInfo($mappingInfo)
```

删除所有变量映射

要删除 TwinCAT 项目中的所有变量映射，可以使用方法 ITcSysManager2.ClearMappingInfo()。

代码片段 (C#)：

```
ITcSysManager2 systemManager2 = (ITcSysManager2)systemManager;
systemManager2.ClearMappingInfo();
```

代码片段 (Powershell)：

```
$systemManager.ClearMappingInfo()
```

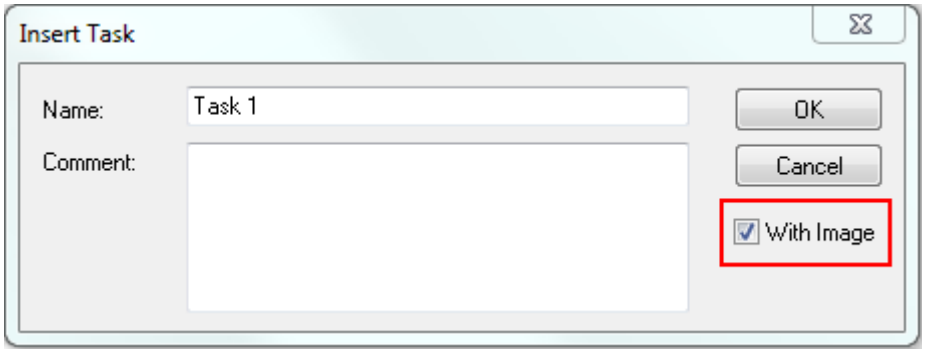
4.3.2.4 创建和处理任务

本文说明如何通过 TwinCAT 自动化接口创建和处理任务。它包括以下主题：

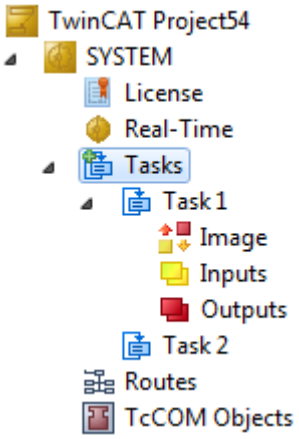
- 常规信息
- 插入任务
- 插入输入/输出变量

常规信息

有两种类型的任务可以在 TwinCAT XAE 中组态，因此也可以通过自动化接口组态：带和不带过程映像的任务。在 TwinCAT XAE 中插入新任务时，可以通过在“Insert Task”（插入任务）对话框中选择相应的复选框来决定是否要包含过程映像。



因此，插入的任务要么包含三个以上的子节点（映像、输入、输出），要么不包含 - 如以下示例所示（任务 1 = 带映像，任务2 = 无映像）。



插入任务

要通过自动化接口插入任务，可以使用 ITcSmTreeItem [▶ 103]::CreateChild() [▶ 136] 方法以及“带映像” (SubType = 0) 和“无映像” (SubType = 1) 的相应子类型。

代码片段 (C#)

```
ITcSmTreeItem tasks = systemManager.LookupTreeItem("TIRT");
tasks.CreateChild("Task 1 (With Image)", 0, null, null);
```

代码片段 (Powershell)：

```
$tasks = $systemManager.LookupTreeItem("TIRT")
$tasks.CreateChild("Task 1 (With Image)", 0, $null, $null)
```

代码片段 (C#)

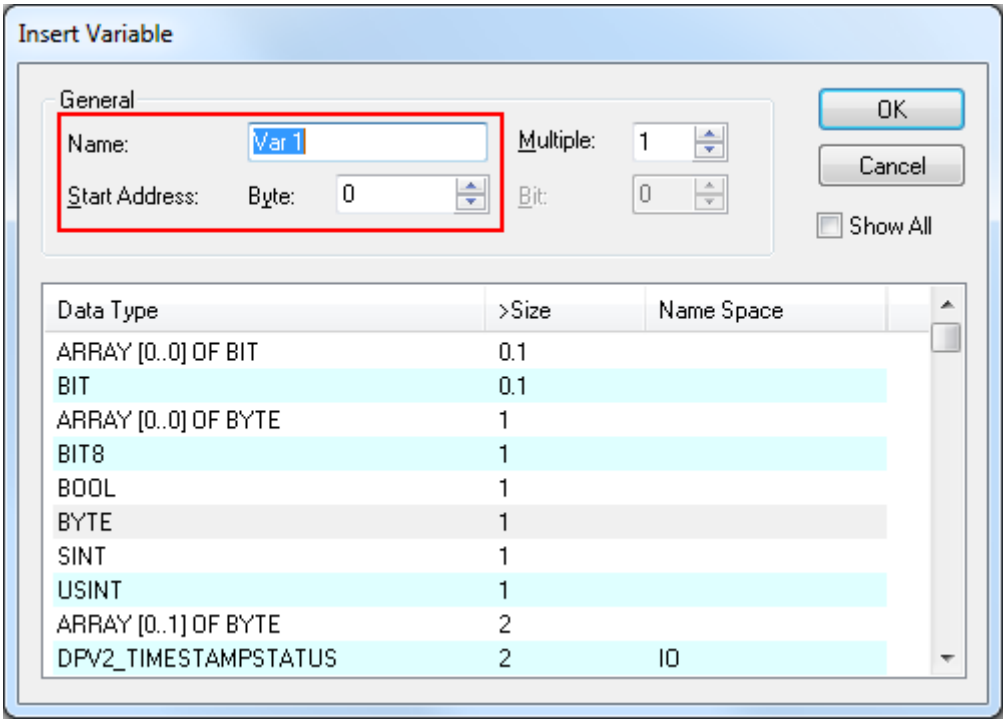
```
ITcSmTreeItem tasks = systemManager.LookupTreeItem("TIRT");
tasks.CreateChild("Task 2 (Without Image)", 1, null, null);
```

代码片段 (Powershell)：

```
$tasks = $systemManager.LookupTreeItem("TIRT")
$tasks.CreateChild("Task 1 (Without Image)", 1, $null, $null)
```

插入输入/输出变量

您可以将输入/输出变量添加到过程映像（“带映像”任务），然后可以将其与 I/O 设备或其他任务中的变量链接。您可以使用 TwinCAT XAE 中的相应对话框选择过程映像中输入/输出变量的数据类型和地址等。点击“Ok” (确定)，变量将添加到过程映像中。



也可以使用 ITcSmTreeItem [▶ 103]::CreateChild() [▶ 136] 方法通过自动化接口触发此过程，相应变量数据类型为 vInfo。在这种情况下，子类型指定“Start Address”(起始地址)，如上面的对话框所示。

代码片段 (C#)：

```
ITcSmTreeItem task1 = systemManager.LookupTreeItem("TIRT^Task 1 (With Image)^Inputs");
task1.CreateChild("bInput", -1, null, "BOOL");
```

代码片段 (Powershell)：

```
$task1 = $systemManager.LookupTreeItem("TIRT^Task 1 (With Image)^Inputs")
$task1.CreateChild("bInput", -1, $null, "BOOL")
```

通过使用 SubType = -1，TwinCAT 会在变量列表末尾自动附加新变量。

4.3.2.5 使用模板

下文描述如何使用模板，而不是通过自动化接口分别组态各项设置。使用模板具有许多优点：更易于维护，更易于替换为新模板，并在与多个团队合作执行一个 TwinCAT 项目时提供极大的可能性。本文档描述模板的使用，并涵盖以下主题：

- 模板的一般概念及不同层级
- 使用 I/O 模板
- 使用 Motion 模板 (轴)
- 使用 PLC 模板

模板的一般概念及不同层级

使用模板的目的是简化自动化接口代码和自动化接口应用程序。大多数用户不知道，在 TwinCAT 组态中，模板可以存在于多个层级。其中包括：

- “组态层级”模板
- 组态层级模板可作为多个 TwinCAT 组态存在 (*.sln 或 *.tszip 文件)。各组态可能包含不同内容，并且可以通过自动化接口代码打开和激活。
- “(PLC) 项目层级”模板
- (PLC) 项目层级模板可作为多个 TwinCAT PLC 项目存在，可以是解压缩文件夹结构 (*.plcproj 文件)，也可以是容器 (*.tpzip) 文件。然后，这些 PLC 项目可以根据需要通过自动化接口代码导入。
- “树项层级”模板

- 树项层级模板可作为所谓的 TwinCAT 导出文件 (*.xti) 存在。这些文件可以根据需要通过自动化接口代码导入，并包含树项的所有设置，例如对 EtherCAT Master I/O 设备进行的设置。

如上所述，所有不同的模板层级都有一个共同点：模板作为文件存在于文件系统中。最佳方案是，在开始实现自己的自动化接口应用程序时，明确定义所使用的“模板池”类型。模板池描述一个存储库，其中存储所有模板文件（也可能是不同模板层级的组合）。这可能只是本地（或远程）文件系统，也可能是源管理系统（例如 Team Foundation Server），由此自动检索、签出模板文件并将其组合到新的 TwinCAT 组态中。为方便起见并简化初始步骤，我们准备了专门的章节，说明如何使用 Visual Studio 对象模型 (DTE) 连接至 Team Foundation Server (TFS) 并通过程序代码执行这些操作。但是，为了获得关于通过自动化接口代码使用 TFS 的详细信息，强烈建议您阅读 Microsoft Developer Network (MSDN) 中的相应内容。

以下主题描述如何针对不同的 TwinCAT 区域（I/O、PLC…）在自动化接口中使用各个模板层级。

使用组态模板

组态层级模板提供最基本的模板使用方法。在这种情况下，模板池保存两个或多个预先组态的 TwinCAT 组态，并将其作为 TwinCAT 解决方案文件 (*.sln 或 *.tszip) 提供。这些文件可以使用 Visual Studio 方法 AddFromTemplate() 通过自动化接口打开。

代码片段 (C#)：

```
project = solution.AddFromTemplate(@"C:\TwinCAT Project.tszip", destination, projectName);
```

代码片段 (Powershell)：

```
$project = $solution.AddFromTemplate(@"C:\TwinCAT Project.tszip", $destination, $projectName)
```

或

代码片段 (C#)：

```
project = solution.Open(@"C:\TwinCAT Project 1.sln");
```

代码片段 (Powershell)：

```
$project = $solution.Open(@"C:\TwinCAT Project 1.sln")
```

使用 I/O 模板

使用 I/O 设备时，用户经常会在 I/O 设备上反复执行相同任务。这意味着，对采用各 TwinCAT 组态的 I/O 设备进行相同的设置。TwinCAT 提供一种机制，将所有这些设置存储为一种特殊的文件格式，即所谓的 XTI 文件 (*.xti)。此文件格式可由自动化接口代码使用，以通过使用 ITcSmTreeItem 接口中定义的 ImportChild() 方法将其导入新组态。

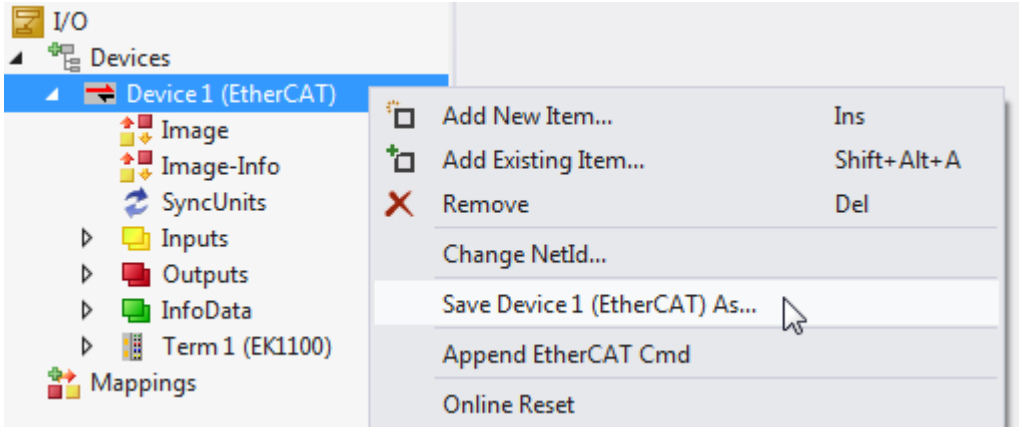
代码片段 (C#)：

```
ITcSmTreeItem io = systemManager.LookupTreeItem("TIID");
ITcSmTreeItem newIo = io.ImportChild(@"C:\IoTemplate.xti", "", true, "SomeName");
```

代码片段 (Powershell)：

```
$io = $systemManager.LookupTreeItem("TIID")
$newIo = $io.ImportChild(@"C:\IoTemplate.xti", "", true, "SomeName")
```

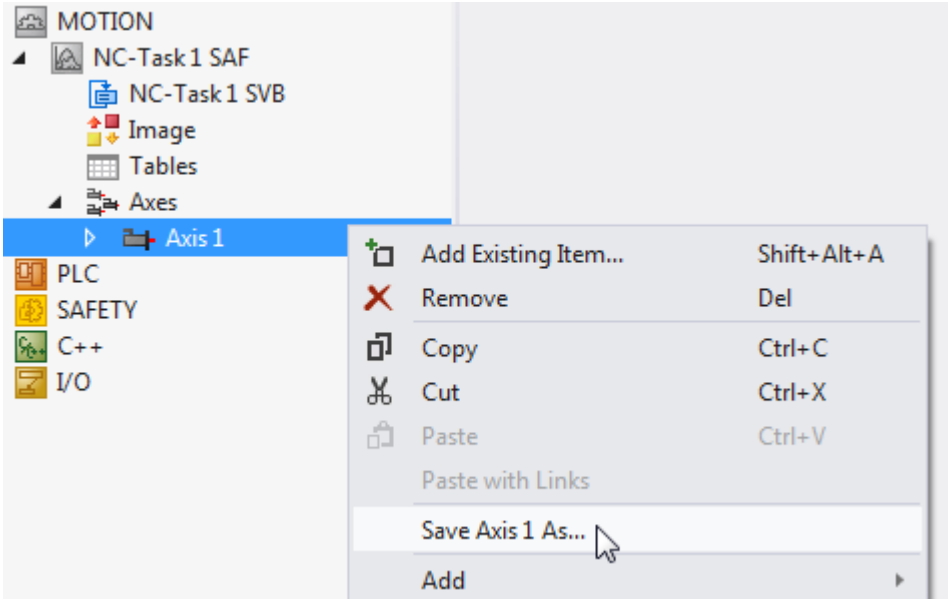
必须注意的是，如果从 TwinCAT XAE 中导出 I/O 设备，所有子设备会自动导出并包含在导出文件中。以下截屏显示如何将 I/O 设备导出到导出文件中。



请注意，您也可以使用自动化接口将 I/O 设备导出到 XTI 文件中。为此，将提供 ITcSmTreeItem 接口中的 ExportChild() 方法。

使用 Motion 模板 (轴)

Motion 轴模板的使用与 I/O 设备非常相似。轴也可以通过 TwinCAT XAE 或使用 ExportChild () 通过自动化接口代码导出到 XTI 文件中。



通过使用 ImportChild()，以后可以再次导入这些导出文件。

代码片段 (C#)：

```
ITcSmTreeItem motionAxes = systemManager.LookupTreeItem("TINC^NC-Task^Axes");
motionAxes.ImportChild(@"C:\AxisTemplates.xti", "", true, "Axis 1");
```

代码片段 (Powershell)：

```
$motionAxes = $systemManager.LookupTreeItem("TINC^NC-Task^Axes")
$motionAxes.ImportChild(@"C:\AxisTemplates.xti", "", true, "Axis 1")
```

使用 PLC 模板

PLC 模板可在两个单元使用：可以将整个 PLC 项目作为一个整体使用，也可以将每个 POU 单独用作模板。要将前者集成到现有 TwinCAT 项目中，只需使用 ITcSmTreeItem 接口的 CreateChild() 方法。

代码片段 (C#)：

```
ITcSmTreeItem plc = systemManager.LookupTreeItem("TIPC");
plc.CreateChild("NewPlcProject", 0, null, pathToTpzipOrTcProjFile);
```

代码片段 (Powershell)：

```
$plc = $systemManager.LookupTreeItem("TIPC")
$plc.CreateChild("NewPlcProject", 0, $null, $pathToTpzipOrTcProjFile)
```

对于导入现有 PLC 项目的更多选项，请参见有关“访问、创建和处理 PLC 项目”的最佳案例章节。

为 PLC 导入模板文件的下一个粒度级别是导入现有的 POU，例如功能块、结构、枚举、全局变量列表等。开发人员选择单个 POU 作为模板的原因之一是，构建现有功能池并将其封装在单独 POU 中更加容易，例如包含不同排序算法的不同功能块。创建项目时，开发人员只需选择在 TwinCAT 项目中需要的功能，然后访问模板工具来检索相应的 POU 并将其导入 PLC 项目。

代码片段 (C#)：

```
ITcSmTreeItem plcProject = systemManager.LookupTreeItem("TIPC^Name^Name Project");
plcProject.CreateChild("NameOfPou", 58, null, pathToPouFile);
plcProject.CreateChild(null, 58, null, stringArrayWithPathsToPouFiles);
```

代码片段 (Powershell)：

```
$plcProject = $systemManager.LookupTreeItem("TIPC^Name^Name Project")
$plcProject.CreateChild("NameOfPou", 58, $null, $pathToPouFile)
$plcProject.CreateChild($null, 58, $null, $stringArrayWithPathsToPouFiles)
```

注意 **POU 模板文件不仅可以是 .TcPou 文件，也可以是 DUT 和/或 GVL 的相应文件。上面的示例演示了导入 POU 模板文件的两种常见方法。一种是导入单个文件，另一种是通过将文件路径存储到字符串数组并将此字符串数组用作 CreateChild() 的 vInfo 参数来同时导入多个文件。**

4.3.2.6 访问 TwinCAT 远程管理器

本文描述如何通过自动化接口访问 TwinCAT 远程管理器功能。TwinCAT 远程管理器允许在同一工程计算机上安装的不同 TwinCAT 3 XAE 版本之间进行切换。要通过自动化接口访问远程管理器，只需执行以下代码片段。

代码片段 (C#)：

```
ITcRemoteManager remoteManager = dte.GetObject("TcRemoteManager");
remoteManager.Version = "3.1.4020.0";
```

代码片段 (Powershell)：

```
$remoteManager = $dte.GetObject("TcRemoteManager")
$remoteManager.Version = "3.1.4020.0"
```

4.3.2.7 将任务分配给 CPU 内核

步骤

- 启动内核以便实时使用。
- 通过 LoadLimit、基准时间 (BaseTime) 和 Latency Watchdog 对内核进行参数化
- 将任务分配给 CPU 资源。

以下代码片段也可以从示例部分下载。

示例 (C#)：

```
ITcSysManager3 systemManager = null;
[Flags()]
public enum CpuAffinity : ulong
{
CPU1 = 0x0000000000000001,
CPU2 = 0x0000000000000002,
CPU3 = 0x0000000000000004,
CPU4 = 0x0000000000000008,
CPU5 = 0x0000000000000010,
CPU6 = 0x0000000000000020,
CPU7 = 0x0000000000000040,
CPU8 = 0x0000000000000080,
None = 0x0000000000000000,
MaskSingle = CPU1,
MaskDual = CPU1 | CPU2,
MaskQuad = MaskDual | CPU3 | CPU4,
MaskHexa = MaskQuad | CPU5 | CPU6,
MaskOct = MaskHexa | CPU7 | CPU8,
MaskAll = 0xFFFFFFFFFFFFFFFF
}

public void AssignCPUCores()
{
ITcSmTreeItem realtimeSettings = systemManager.LookupTreeItem("TIRS");
// CPU Settings
// <TreeItem>
// <RTimeSetDef>
// <MaxCPUs>3</MaxCPUs>
// <Affinity>#x0000000000000007</Affinity>
// <CPUs>
// <CPU id="0">
// <LoadLimit>10</LoadLimit>
// <BaseTime>10000</BaseTime>
// <LatencyWarning>200</LatencyWarning>
// </CPU>
```

```
 // <CPU id="1">
// <LoadLimit>20</LoadLimit>
// <BaseTime>5000</BaseTime>
// <LatencyWarning>500</LatencyWarning>
// </CPU>
// <CPU id="2">
// <LoadLimit>30</LoadLimit>
// <BaseTime>3333</BaseTime>
// <LatencyWarning>1000</LatencyWarning>
// </CPU>
// </CPUs>
// </RTimeSetDef>
// </TreeItem>

string xml = null;
MemoryStream stream = new MemoryStream();
StringWriter stringWriter = new StringWriter();
using(XmlWriter writer = XmlTextWriter.Create(stringWriter))
{
writer.WriteStartElement("TreeItem");
writer.WriteStartElement("RTimeSetDef");
writer.WriteElementString("MaxCPUs", "4");
string affinityString = string.Format("#x{0}",((ulong)
cpuAffinity.MaskQuad).ToString("x16"));
writer.WriteElementString("Affinity", affinityString);
writer.WriteStartElement("CPUs");
writeCpuProperties(writer, 0, 10, 1000, 10000, 200);
writeCpuProperties(writer, 1, 20, 5000, 10000, 500);
writeCpuProperties(writer, 2, 30, 3333, 10000, 1000);
writer.WriteEndElement();     // CPUs
writer.WriteEndElement();     // RTimeSetDef
writer.WriteEndElement();     // TreeItem
}
xml = stringWriter.ToString();
realtimeSettings.ConsumeXml(xml);
ITcSmTreeItem tasks = systemManager.LookupTreeItem("TIRT");
ITcSmTreeItem task1 = tasks.CreateChild("TaskA",1);
setTaskProperties(task1,CpuAffinity.CPU1);
ITcSmTreeItem task2 = tasks.CreateChild("TaskB",1);
setTaskProperties(task2, CpuAffinity.CPU2);

ITcSmTreeItem task3 = tasks.CreateChild("TaskC", 1);
setTaskProperties(task3, CpuAffinity.CPU3);
}

private void setTaskProperties(ITcSmTreeItem task, CpuAffinity affinityMask)
{
// <TreeItem>
// <TaskDef>
// <CpuAffinity>#x0000000000000004</CpuAffinity>
// </TaskDef>
// </TreeItem>

StringWriter stringWriter = new StringWriter();
using(XmlWriter writer = new XmlTextWriter(stringWriter))
{
    writer.WriteStartElement("TreeItem");
    writer.WriteStartElement("TaskDef");
    string affinityString = string.Format("#x{0}", ((ulong)affinityMask).ToString("x16"));
    writer.WriteElementString("CpuAffinity",affinityString);
    writer.WriteEndElement();
    writer.WriteEndElement();
}

string xml = stringWriter.ToString();
task.ConsumeXml(xml);
}

private void writeCpuProperties(XmlWriter writer, int id, int loadLimit, int baseTime, int
latencyWarning)
{
writer.WriteStartElement("CPU");
writer.WriteAttributeString("id", id.ToString());
writer.WriteElementString("LoadLimit", loadLimit.ToString());
writer.WriteElementString("BaseTime", baseTime.ToString());
writer.WriteElementString("LatencyWarning", latencyWarning.ToString());
writer.WriteEndElement();
}
```

4.3.2.8 组态 TwinCAT Boot 设置

下文描述如何通过自动化接口组态 TwinCAT Boot 设置。为此，可以使用 ITcSmTreeItem::ProduceXml() 和 ITcSmTreeItem::ConsumeXml() 方法生成或导入以下 XML 结构，该结构表示 TwinCAT XAE 中的相应设置。

```
<TreeItem>
  <System>
    <BootSettings>
      <AutoRun>true</AutoRun>
      <AutoLogon>true</AutoLogon>
      <LogonUserName>UserName</LogonUserName>
      <LogonPassword>Password</LogonPassword>
      <BootFileEncryptionType>None</BootFileEncryptionType>
    </BootSettings>
  </System>
</TreeItem>
```

4.3.2.9 激活或禁用 IndependentProjectFile 设置

为增强与源管理集成相关的工程体验，TwinCAT 3 提供了将设置存储在单独项目文件（称为“IndependentProjectFile”）中的可能性。这种基于设置的树项也可以通过 TwinCAT 自动化接口激活/禁用。ITcSmTreeItem6 接口提供了必要的属性。

以下代码片段显示如何在此设置处于禁用状态时将其激活，例如在 EtherCAT Master 设备上。

代码片段 (C#)：

```
ITcSmTreeItem6 etherCatMaster = (ITcSmTreeItem6)systemManager.LookupTreeItem("TIID^Device 1
(EtherCAT)");
if (etherCatMaster.SaveInOwnFile == false)
  etherCatMaster.SaveInOwnFile = true;
```

代码片段 (Powershell)：

```
$etherCatMaster = $systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT)")
if ($etherCatMaster.SaveInOwnFile -eq $false)
{
  $etherCatMaster.SaveInOwnFile = $true
}
```

4.3.2.10 从离线组态到在线组态

本文说明如何通过 TwinCAT 自动化接口将“离线”创建的 TwinCAT 组态转换为在线组态。“离线”一词表示在创建组态时不存在物理 I/O，因此无法获得实际地址信息，例如，对于 EtherCAT Master。本文包含以下主题：

- 常规信息
- 创建离线组态
- 切换至在线组态

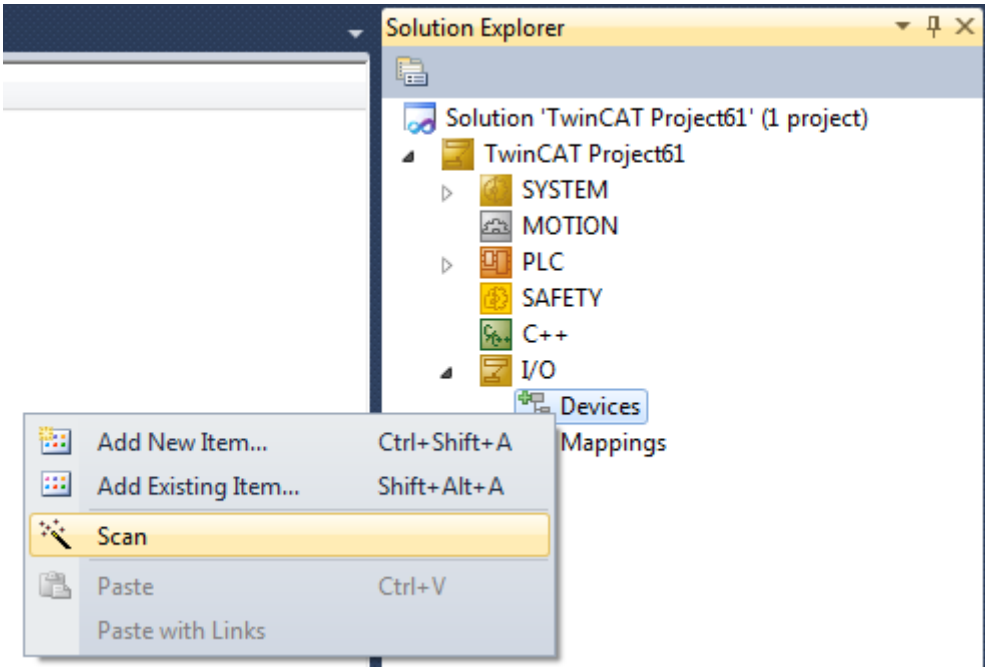
常规信息

创建 TwinCAT 组态时，有两种常见情况：

- 情况 1：这种情况基于实际物理设备，TwinCAT 组态之后会在该设备上运行。因此，所有 I/O 均已**在线**且可用，并已附加至设备。
- 情况 2：这种情况可能涉及**离线**创建组态（这意味着没有任何 I/O 附加到工程设备），之后在 I/O 可用时切换到在线组态。这是本文主要关注的情况。

两种情况均可通过 TwinCAT 自动化接口实现。但是，由于情况 2 缺少关于某些 I/O 的一些重要信息，例如其物理地址，因此可能稍微复杂一些，因为之后您需要提交所需的（地址）信息。

熟悉的 TwinCAT XAE 功能“Scan Devices”（扫描设备）在此用例中起重要作用，因为它将扫描所有可用的 I/O 设备，并自动将其以及所有需要的附加信息附加到组态中，例如物理地址。



在物理控制器上，此功能也可通过自动化接口调用，然后返回所有找到的 I/O 设备的 XML 描述，包括相应的地址信息。

在控制器上激活 TwinCAT 组态时，需要为组态中的所有 I/O 设备设置所需的地址信息。这可通过使用自动化接口调用“Scan Devices”（扫描设备）功能并在组态中设置 I/O 设备的地址信息来完成。现在将对此进行更详细的说明。

创建离线组态

多个章节对如何创建离线 TwinCAT 组态进行了说明。请参见产品描述 [▶ 9]页获得概述。

切换至在线组态

如果您最终创建了一个 TwinCAT 组态，现在应在物理控制器上将其激活，则需要采取以下步骤，以确保在下载组态之前所有必需的地址信息均可用：

- 步骤 1 [可选]：连接至目标设备
- 步骤 2：在设备上扫描可用 I/O
- 步骤 3：迭代 XML 并组态 I/O 的地址信息
- 步骤 4：激活组态

当然，也可以随时使用 ScanDevices 功能直接创建在线组态。示例 [▶ 79]将为您说明如何操作。

步骤 1 [可选]：连接至目标设备

如果物理控制器与运行自动化接口代码的机器不在同一台机器上，则可以使用 ITcSysManager [▶ 96]::SetTargetNetId() [▶ 100] 方法连接至远程设备，然后继续进行以下步骤。

步骤 2：在设备上扫描可用 I/O

通过在 I/O 设备节点上调用 ITcSmTreeItem::ProduceXml() 方法，可以通过自动化接口触发上述“Scan Devices”（扫描设备）功能。

代码片段 (C#)：

```
ITcSmTreeItem ioDevices = systemManager.LookupTreeItem("TIID");
string foundDevices = ioDevices.ProduceXml();
```

步骤 3：迭代 XML 并组态 I/O 的地址信息

在此示例中，我们想要更新离线组态中的 EtherCAT 设备地址信息。步骤 2 中的 ProduceXml() 已返回系统上可用的 I/O 设备，现在在“foundDevices”变量中可用。在此 XML 中，将通过项目子类型 (111) 标识 EtherCAT 设备，然后在组态中更新 EtherCAT Master 的地址信息。

代码片段 (C#)：

```
XmlDocument doc = new XmlDocument();
doc.LoadXml(foundDevices);
XmlNodeList devices = doc.SelectNodes("TreeItem/DeviceGrpDef/FoundDevices/Device");
foreach (XmlNode device in devices)
{
xmlNode typeNode = device.SelectSingleNode("ItemSubType");
int subType = int.Parse(typeNode.InnerText);
if (subType == 111)
    {
     XmlNode addressInfo = device.SelectSingleNode("AddressInfo");
     ITcSmTreeItem deviceToUpdate = systemManager.LookupTreeItem("TIID^EtherCAT Master");
     string xmlAddress = string.Format("<TreeItem><DeviceDef>{0}</DeviceDef></TreeItem>",
addressInfo.OuterXml);
     deviceToUpdate.ConsumeXml(xmlAddress);
}
}
```

步骤 4：激活组态

最后一步只涉及激活目标设备上的组态。操作时仅需使用 ITcSysManager::ActivateConfiguration() 方法。

代码片段 (C#)：

```
sysManager.ActivateConfiguration();
```

4.3.3 ADS

4.3.3.1 创建和处理 ADS 路径

可使用 ITcSmTreeItem 接口的 ConsumeXml() 方法通过自动化接口添加 ADS 路径。但是，在向远程目标添加路径之前，必须了解底层 XML 结构。

XML 结构

以下代码片段表示用于向远程目标添加路径的示例 XML 结构。请注意，您可以指定远程目标的 IP 地址或主机名。

此代码片段将向远程目标添加常规路径。

代码片段 (XML)：

```
<TreeItem>
  <ItemName>Route Settings</ItemName>
  <PathName>TIRR</PathName>
  <RoutePrj>
    <TargetList>
      <BroadcastSearch>true</BroadcastSearch>
    </TargetList>
    <AddRoute>
      <RemoteName>RouteName</RemoteName>
      <RemoteNetId>1.2.3.4.5.6</RemoteNetId>
      <RemoteIpAddr>1.2.3.4</RemoteIpAddr>
      <UserName>userName</UserName>
      <Password>password</Password>
      <NoEncryption></NoEncryption>
      <LocalName>LocalName</LocalName>
    </AddRoute>
  </RoutePrj>
</TreeItem>
```

此代码片段将向远程目标添加项目路径。

代码片段 (XML)：

```
<TreeItem>
  <ItemName>Route Settings</ItemName>
  <PathName>TIRR</PathName>
  <RoutePrj>
    <TargetList>
      <BroadcastSearch>true</BroadcastSearch>
    </TargetList>
    <AddProjectRoute>
      <Name>RouteName</Name>
      <NetId>1.2.3.4.5.6</NetId>
      <IpAddr>1.2.3.4</IpAddr>
    </AddProjectRoute>
  </RoutePrj>
</TreeItem>
```

以下代码片段将使用主机名而不是 IP 地址。

代码片段 (XML)：

```
<TreeItem>
  <ItemName>Route Settings</ItemName>
  <PathName>TIRR</PathName>
  <RoutePrj>
    <TargetList>
      <BroadcastSearch>true</BroadcastSearch>
    </TargetList>
    <AddRoute>
      <RemoteName>RouteName</RemoteName>
      <RemoteNetId>1.2.3.4.5.6</RemoteNetId>
      <RemoteHostName>CX-12345</RemoteHostName>
      <UserName>userName</UserName>
      <Password>password</Password>
      <NoEncryption></NoEncryption>
      <LocalName>LocalName</LocalName>
    </AddRoute>
  </RoutePrj>
</TreeItem>
```

用于项目路径。

代码片段 (XML)：

```
<TreeItem>
  <ItemName>Route Settings</ItemName>
  <PathName>TIRR</PathName>
  <RoutePrj>
    <TargetList>
      <BroadcastSearch>true</BroadcastSearch>
    </TargetList>
    <AddProjectRoute>
      <Name>RouteName</Name>
      <NetId>1.2.3.4.5.6</NetId>
      <HostName>1.2.3.4</HostName>
    </AddProjectRoute>
  </RoutePrj>
</TreeItem>
```

请注意，可以同时使用常规路径和项目路径的 XML 结构。

以下代码片段创建连接至远程目标的 ADS 路径，该目标已由其 IP 地址 (10.1.128.217) 和 AMS NetId (10.1.128.217.1.1) 指定。

代码片段 (C#)：

```
string xmlString ="<TreeItem><ItemName>Route Settings</ItemName><PathName>TIRR</
PathName><RoutePrj><TargetList><BroadcastSearch>true</BroadcastSearch></
TargetList><AddRoute><RemoteName>RouteName</RemoteName><RemoteNetId>10.1.128.217.1.1</
RemoteNetId><RemoteIpAddr>10.1.128.217</RemoteIpAddr><UserName>Administrator</UserName><Password>1</
Password><NoEncryption></NoEncryption></AddRoute></RoutePrj></TreeItem>";
ITcSmTreeItem routes = systemManager.LookupTreeItem("TIRR");
routes.ConsumeXml(xmlString);
```

代码片段 (Powershell)：

```
$xmlString ="<TreeItem><ItemName>Route Settings</ItemName><PathName>TIRR</
PathName><RoutePrj><TargetList><BroadcastSearch>true</BroadcastSearch></
TargetList><AddRoute><RemoteName>RouteName</RemoteName><RemoteNetId>10.1.128.217.1.1</
RemoteNetId><RemoteIpAddr>10.1.128.217</RemoteIpAddr><UserName>Administrator</UserName><Password>1</
```

```
Password><NoEncryption></NoEncryption></AddRoute></RoutePrj></TreeItem>"
$routes = $systemManager.LookupTreeItem("TIRR")
$routes.ConsumeXml($xmlString)
```

还请参阅有关此

示例下载 [▶ 158]

4.3.3.2 执行 ADS 广播搜索

要触发 TwinCAT 广播搜索并查找未知的远程 ADS 设备，可以使用 ITcSmTreeItem 接口中的 ConsumeXml() 和 ProduceXml() 方法。

常规广播搜索

代码片段 (C#)：

```
string xmlString = "<TreeItem><RoutePrj><TargetList><BroadcastSearch>true</BroadcastSearch></
TargetList></RoutePrj></TreeItem>";
ITcSmTreeItem routes = sysMan.LookupTreeItem("TIRR");
routes.ConsumeXml(xmlString);
string result = routes.ProduceXml();
```

代码片段 (Powershell)：

```
$xmlString = "<TreeItem><RoutePrj><TargetList><BroadcastSearch>true</BroadcastSearch></TargetList></
RoutePrj></TreeItem>"
$routes = $systemManager.LookupTreeItem("TIRR")
$routes.ConsumeXml($xmlString)
$result = $routes.ProduceXml()
```

“result”变量现在包含网络中找到的所有 ADS 设备的 XML 描述。要从该列表中选择指定设备，可以使用用于 XML 处理的常规 .NET 机制。

代码片段 (C#)：

```
XmlDocument xmlDocument = new XmlDocument();
xmlDocument.Load(result);
string amsNetId = xmlDocument.SelectSingleNode("//TreeItem/RoutePrj/TargetList/Target/IpAddr[text()=
\"" + ipAddress + "\"]/../NetId").InnerText;
string name = xmlDocument.SelectSingleNode("//TreeItem/RoutePrj/TargetList/Target/IpAddr[text()=\""
+ ipAddress + "\"]/../Name").InnerText;
```

代码片段 (Powershell)：

```
$xmlDocument = [xml]$result
$localAmsNetId = $xmlDocument.TreeItem.RoutePrj.Target
```

然后，可以使用此信息向 ADS 设备添加路径，按照专门的章节所述进行操作。

直接广播搜索

要求至少为 TwinCAT 3.1 Build 4020.10 或更高版本。

要使用指定的主机名或 IP 地址执行广播搜索，可以在 ConsumeXml() 中使用以下 XML 结构。

XML - 搜索主机名：

```
<TreeItem>
  <RoutePrj>
    <TargetList>
      <Search>CX-12345</Search>
    </TargetList>
  </RoutePrj>
</TreeItem>
```

XML - 搜索 IP 地址：

```
<TreeItem>
  <RoutePrj>
    <TargetList>
```

```
      <Search>172.17.60.153</Search>
    </TargetList>
  </RoutePrj>
</TreeItem>
```

随后的 ProduceXml() 将返回找到的主机，如下所示：

XML - 找到主机：

```
<TreeItem>
  <RoutePrj>
    <TargetList>
      <Target>
        <Name>CX-12345</Name>
        <NetId>172.17.60.153.1.1</NetId>
        <IpAddr>172.17.60.153</IpAddr>
        <Version>3.1.4020</Version>
        <OS>Windows 7</OS>
      </Target>
    </TargetList>
  </RoutePrj>
</TreeItem>
```

4.3.4 PLC

4.3.4.1 访问、创建和处理 PLC 项目

本章将详细说明如何创建、访问和处理 PLC 项目。以下列表显示本文中的所有章节：

- PLC 项目常规信息
- 创建和处理 PLC 项目
- 打开现有的 PLC 项目
- 嵌套项目及项目实例
- 将 PLC 项目另存为库
- 处理在线功能 (Login、StartPlc, StopPlc)
- 设置 Boot 项目选项
- 将项目和/或解决方案存档
- 调用 CheckAllObjects()

PLC 项目常规信息

PLC 项目由其所谓的项目模板指定。TwinCAT 目前部署了两个模板，由 TwinCAT 目录中的模板文件表示。下表显示可用的 PLC 模板及相应的模板文件：

模板名	模板文件
标准 PLC 模板	C:\TwinCAT\3.x\Components\Plc\PlcTemplate\Plc Templates\Standard PLC Template.plcproj
空 PLC 模板	C:\TwinCAT\3.x\Components\Plc\PlcTemplate\Plc Templates\Empty PLC Template.plcproj

创建和处理 PLC 项目

要通过自动化接口创建新的 PLC 项目，需要导航至 PLC 节点，然后以相应的模板文件作为参数执行 CreateChild() 方法。

代码片段 (C#)：

```
ITcSmTreeItem plc = systemManager.LookupTreeItem("TIPC");
ITcSmTreeItem newProject = plc.CreateChild("NameOfProject", 0, "", pathToTemplateFile);
```

代码片段 (Powershell)：

```
$plc = $systemManager.LookupTreeItem("TIPC")
$newProject = $plc.CreateChild("NameOfProject", 0, "", pathToTemplateFile)
```

请注意

使用 Beckhoff 提供的标准 PLC 模板时，请确保只使用模板名称而不是完整路径，例如“标准 PLC 模板”。

所有后续操作，例如创建和处理 POU 以及用代码填充，请参见专门的章节。

创建 PLC 项目后，可以通过将其强制转换到 ITcPlcIECProject [▶ 143] 专用接口来进行进一步处理，该接口提供更多功能以及对项目特定属性的访问权限：

代码片段 (C#)：

```
ITcSmTreeItem plcProject = systemManager.LookupTreeItem("TIPC^NameOfProject^NameOfProject Project");
ITcPlcIECProject iecProject = (ITcPlcIECProject) plcProject;
```

代码片段 (Powershell)：

```
$plcProject = $systemManager.LookupTreeItem("TIPC^NameOfProject^NameOfProject Project")
```

对象“iecProject”现在可用于访问 ITcPlcIECProject 接口的方法，例如将 PLC 项目另存为 PLC 库。

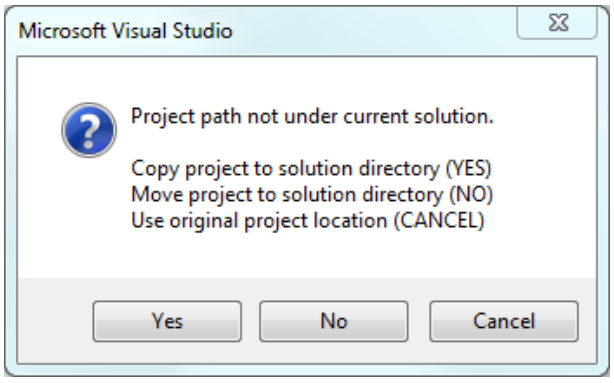
打开现有的 PLC 项目

要通过自动化接口打开现有的 PLC 项目，需要导航至 PLC 节点，然后以相应的 PLC 项目文件路径作为参数执行 CreateChild() 方法。

可以使用三个不同的值作为子类型：

- 0：将项目复制到解决方案目录
- 1：将项目移动到解决方案目录
- 2：使用原始项目位置（使用时，请将 "" 用作项目名参数）

基本上，这些值表示 TwinCAT XAE 中以下消息框中的功能（是、否、取消）：



您需要使用需要添加的 PLC 项目路径（至其 plcproj 文件）来代替模板文件。或者，也可以使用 PLC 项目存档（tpzip 文件）。

代码片段 (C#)：

```
ITcSmTreeItem plc = systemManager.LookupTreeItem("TIPC");
ITcSmTreeItem newProject = plc.CreateChild("NameOfProject", 1, "", pathToProjectOrTpzipFile);
```

代码片段 (Powershell)：

```
$plc = $systemManager.LookupTreeItem("TIPC")
$newProject = $plc.CreateChild("NameOfProject", 1, "", pathToProjectOrTpzipFile)
```

TwinCAT PLC 项目包含两个不同的区域 - 所谓的嵌套项目和项目实例。嵌套项目（树项子类型 56）包含 PLC 程序的源代码，而项目实例包含已声明的 PLC 程序的输入和输出变量。

以下代码片段显示在完整路径名未知的情况下以通用方式访问两个树项的常见方法。

代码片段 (C#)：

```
ITcSmTreeItem plc = sysManager.LookupTreeItem("TIPC");
foreach (ITcSmTreeItem plcProject in plc)
{
    ITcProjectRoot projectRoot = (ITcProjectRoot)plcProject;
    ITcSmTreeItem nestedProject = projectRoot.NestedProject;
    ITcSmTreeItem projectInstance = plcProject.get_Child(1);
}
```

代码片段 (Powershell)：

```
$plc = $sysManager.LookupTreeItem("TIPC")
ForEach( $plcProject in $plc)
{
    $nestedProject = $plcProject.NestedProject
    $projectInstance = $plcProject.get_Child(1)
}
```

请注意

访问 ITcProjectRoot 接口至少需要 TwinCAT 3.1 Build 4018。

将 PLC 项目另存为库

要将 PLC 项目保存为 PLC 库，需要使用 ITcPlcIECProject [▶ 143]::SaveAsLibrary() [▶ 146] 方法。

代码片段 (C#)：

```
iecProject.SaveAsLibrary(pathToLibraryFile, false);
```

代码片段 (Powershell)：

```
$plcProject.SaveAsLibrary(pathToLibraryFile, $false)
```

第二个参数确定将库保存为文件后是否应安装到默认存储库。

处理在线功能 (Login、StartPlc、StopPlc、ResetCold、ResetOrigin)

所需版本： TwinCAT 3.1 Build 4010 及以上版本

自动化接口还为您提供了 PLC 在线功能，例如Login到 PLC runtime 并启动/停止/重置 PLC 程序。这些功能均可通过 ITcSmTreeItem [▶ 103]::ProduceXml() [▶ 134] 和 ITcSmTreeItem [▶ 103]::ConsumeXml() [▶ 135] 方法访问。可以在 ITcPlcIECProject [▶ 143] 节点上使用这些功能。

XML 结构：

```
<TreeItem>
<IECProjectDef>
<OnlineSettings>
<Commands>
    <LoginCmd>false</LoginCmd>
    <LogoutCmd>false</LogoutCmd>
    <StartCmd>false</StartCmd>
    <StopCmd>false</StopCmd>
</Commands>
</OnlineSettings>
</IECProjectDef>
</TreeItem>
```

代码片段 (C#)：

```
string xml = "<TreeItem><IECProjectDef><OnlineSettings><Commands><LoginCmd>true</LoginCmd></
Commands></OnlineSettings></IECProjectDef></TreeItem>";
ITcSmTreeItem plcProject = systemManager.LookupTreeItem("TIPC^NameOfProject^NameOfProject Project");
plcProject.ConsumeXml(xml);
```

代码片段 (Powershell)：

下表更详细地描述了每个 XML 节点：

XML	描述
LoginCmd	true = in SPS-Laufzeit einloggen
LogoutCmd	true = aus SPS-Laufzeit ausloggen
StartCmd	true = Starten des aktuell in die Laufzeit geladenen SPS-Programms
StopCmd	true = Stoppen des aktuell in die Laufzeit geladenen SPS-Programms

请注意：为了使用 ResetOriginCmd 等命令，必须先执行 LoginCmd - 与 TwinCAT XAE 相似。

设置 Boot 项目选项

以下代码片段显示如何使用 ITcPlcProject 接口为 PLC 项目设置 Boot 项目选项。

代码片段 (C#)：

```
ITcSmTreeItem plcProjectRoot = systemManager.LookupTreeItem("TIPC^PlcGenerated");
ITcPlcProject plcProjectRootIec = (ITcPlcProject) plcProjectRoot;
plcProjectRootIec.BootProjectAutostart = true;
plcProjectRootIec.GenerateBootProject(true);
```

代码片段 (Powershell)：

```
$plcProject = $systemManager.LookupTreeItem("TIPC^PlcGenerated")
$plcProject.BootProjectAutostart = $true
$plcProject.GenerateBootProject($true)
```

将项目和/或解决方案存档

要将整个 TwinCAT 解决方案保存在 ZIP 兼容的存档文件 (*.tszip) 中，可以使用 ITcSysManager9 接口。

代码片段 (C#)：

```
ITcSysManager9 newSysMan = (ITcSysManager9)systemManager;
newSysMan.SaveAsArchive(@"C:\test.tszip");
```

代码片段 (Powershell)：

```
$systemManager.SaveAsArchive("C:\test.tszip")
```

要重新加载先前保存的 TSZIP 文件，可以使用 DTE 方法 AddFromTemplate()。

代码片段 (C#)：

```
dte.Solution.AddFromTemplate("C:\test.tszip",@"C:\tmp","CreatedFromTemplate");
```

代码片段 (Powershell)：

```
$dte.Solution.AddFromTemplate("C:\test.tszip","C:\tmp","CreatedFromTemplate")
```

要将指定的 PLC 项目保存在 ZIP 兼容的存档文件 (*.tpzip) 中，可以使用 ITcSmTreeItem::ExportChild() 方法。

代码片段 (C#)：

```
ITcSmTreeItem plc= sysManager.LookupTreeItem("TIPC");
plc.ExportChild("PlcProject",@"C:\PlcTemplate.tpzip");
```

代码片段 (Powershell)：

```
$plc = $systemManager.LookupTreeItem("TIPC")
$plc.ExportChild("PlcProject", "C:\PlcTemplate.tpzip")
```

要重新加载先前保存的 TPZIP 文件，可以使用 ITcSmTreeItem::CreateChild() 方法。

代码片段 (C#)：

```
plcConfig.CreateChild("PlcFromTemplate", 0, null, @"C:\PlcTemplate.tpzip");
```

代码片段 (Powershell)：

```
$plc.CreateChild("plcFromTemplate", 0, $null, "C:\PlcTemplate.tpzip")
```

调用 CheckAllObjects()

要在 PLC 嵌套项目上调用 CheckAllObjects() 方法，可以使用 ITcPlcIECProject2 接口中提供的相应方法。

代码片段 (C#)：

```
ITcSmTreeItem plcProject = systemManager.LookupTreeItem("TIPC^NameOfProject^NameOfProject Project");
ITcPlcIECProject2 iecProject = (ITcPlcIECProject2) plcProject;
iecProject.CheckAllObjects();
```

代码片段 (Powershell)：

```
$plcProject = $systemManager.LookupTreeItem("TIPC^NameOfProject^NameOfProject Project")
$plcProject.CheckAllObjects()
```

4.3.4.2 访问、创建和处理 PLC 库及占位符

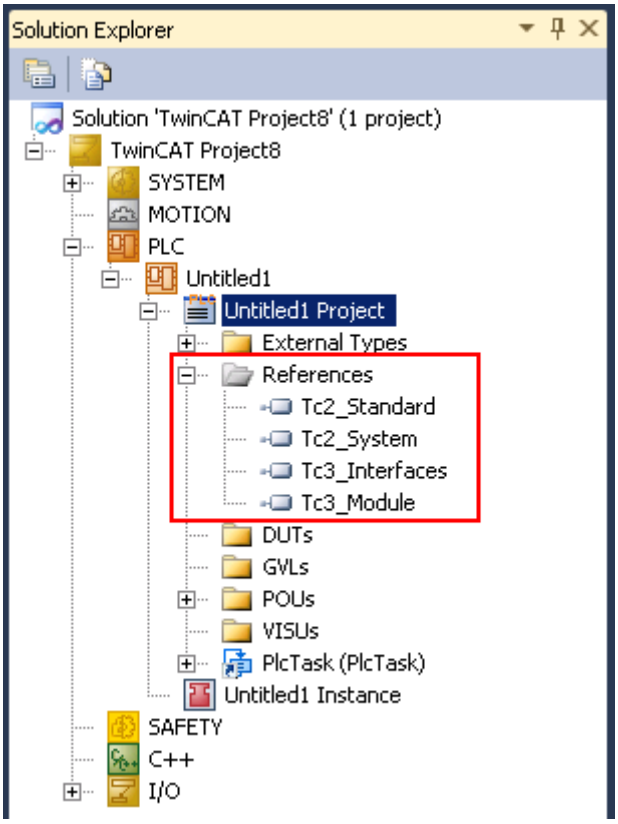
本章将详细说明如何访问和处理 PLC 库及 PLC 占位符。以下列表显示本文中的所有章节：

- PLC 库和占位符常规信息
- 浏览引用
- 添加引用
- 删除引用
- 扫描可用的库
- 冻结占位符版本
- 使用存储库

PLC 库和占位符常规信息

在 TwinCAT 3 中有两种类型的库对象：库和占位符。有关两种类型的详细信息，请参见关于库管理的 TwinCAT 3 文档。

在 TwinCAT 3 PLC 项目中，对库和占位符的引用作为子项添加到相应 PLC 项目下的“引用”节点。选择“Standard PLC Project”（标准 PLC 项目）模板时，一些库和占位符默认添加到项目中，例如 Tc2_Standard、Tc2_System....



通过自动化接口，仅需使用 ITcSysManager [▶ 96]::LookupTreeItem() [▶ 103] 方法，即可导航至“引用”节点。

代码片段 (C#)：

```
ITcSmTreeItem references = systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1
Project^References");
```

代码片段 (Powershell)：

```
$references = $systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1 Project^References")
```

要正确处理此对象，需要将其类型转换到 ITcPlcLibraryManager 接口。

```
ITcPlcLibraryManager libManager = (ITcPlcLibraryManager) references;
```

请注意，Windows Powershell 中无需此步骤。

浏览引用

您可以使用 ITcPlcLibraryManager [▶ 146]::References 属性对所有引用进行迭代。此属性可返回库对象 (即 ITcPlcLibrary [▶ 153]) 或占位对象 (即 ITcPlcPlaceholderRef) 的 ITcPlcReferences 集合。

代码片段 (C#)：

```
ITcSmTreeItem references = systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1
Project^References");
ITcPlcLibraryManager libManager = (ITcPlcLibraryManager) references;
foreach (ITcPlcLibRef libRef in libManager.References)
{
if (libRef is ITcPlcLibrary)
{
    ITcPlcLibrary library = (ITcPlcLibrary) libRef;
    // do something
}
else if (libRef is ITcPlcPlaceholderRef)
{
    ITcPlcPlaceholderRef placeholder = (ITcPlcPlaceholderRef) libRef;
    // do something
}
}
```

迭代对象 libRef 的类型为 ITcPlcLibRef。这是 ITcPlcLibrary 和 ITcPlcPlaceholderRef 对象的通用基础类。要使用这些特定类之一，需要对对象进行相应的类型转换，如上面的代码片段所示。

代码片段 (Powershell)：

```
$references = $systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1 Project^References")
ForEach( $libRef in $references )
{
  $libRef.LanguageIndependentName
}
```

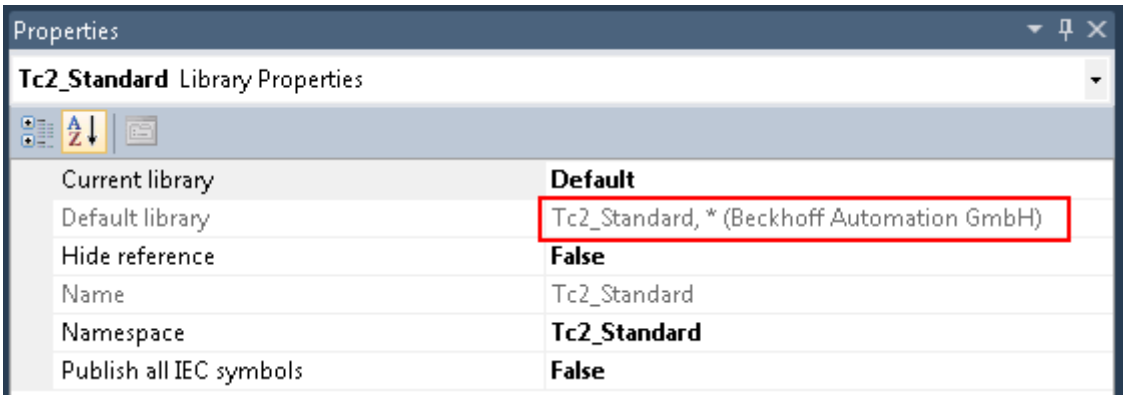
添加引用

ITcPlcLibraryManager [▶ 146] 类中提供两种方法，可为 PLC 项目添加库或占位符引用：AddLibrary() 和 AddPlaceholder()。

可以通过两种方法添加库：

- 指定库名称、版本和发布服务器
- 或使用库显示名称
- 或（如果是占位符）使用占位符名称

库显示名称可在库或占位符的属性窗口中确定：



添加库：

代码片段 (C#)：

```
ITcSmTreeItem references = systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1
Project^References");
ITcPlcLibraryManager libManager = (ITcPlcLibraryManager) references;
libManager.AddLibrary("Tc2_MDP", "*", "Beckhoff Automation GmbH"); // name, version, distribution
list
libManager.AddLibrary("Tc2_Math, * (Beckhoff Automation GmbH)"); //monitored name
```

代码片段 (Powershell)：

```
$references = $systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1 Project^References")
$references.AddLibrary("Tc2_MDP", "*", "Beckhoff Automation GmbH")
$references.AddLibrary("Tc2_Math, * (Beckhoff Automation GmbH)")
```

可以通过两种方法添加占位符：

- 指定占位符名称、库名称、库版本和库发布服务器
- 或使用占位符名称（若占位符已存在）

添加占位符：

代码片段 (C#)：

```
ITcSmTreeItem references = systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1
Project^References");
ITcPlcLibraryManager libManager = (ITcPlcLibraryManager) references;
libManager.AddPlaceholder("Tc2_MC2_Camming"); // add existing place holder with name
libManager.AddPlaceholder("Placeholder_NC", "Tc2_NC", "*", "Beckhoff Automation GmbH");
```

代码片段 (Powershell)：

```
$references = $systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1 Project^References")
$references.AddPlaceholder("Tc2_MC2_Camming")
$references.AddPlaceholder("Placeholder_NC", "Tc2_NC", "*", "Beckhoff Automation GmbH")
```

请注意：添加新的占位符时，AddPlaceholder() 方法的参数指定其默认分辨率。要设置有效分辨率，只需使用 ITcPlcLibraryManager [▶ 146]::SetEffectiveResoltion() 方法。

删除引用

要删除引用，只需使用 ITcPlcLibraryManager [▶ 146]::RemoveReference() 方法。由于此方法在 ITcPlcLibRef 项（ITcPlcLibrary 和 ITcPlcPlaceholderRef 对象的基础类）上运行，因此可以将此方法用于库引用和占位符引用。

库引用可以通过指定库名称、版本和发布服务器或指定其显示名称来删除。

占位符引用可以通过占位符名称来删除。

代码片段（C#）：

```
ITcSmTreeItem references = systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1
Project^References");
ITcPlcLibraryManager libManager = (ITcPlcLibraryManager) references;
libManager.RemoveReference("Tc2_Math"); // delete library
libManager.RemoveReference("Placeholder_NC"); // delete a placeholder
```

代码片段（Powershell）：

```
$references = $systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1 Project^References")
$references.RemoveReference("Tc2_Math")
$references.RemoveReference("Placeholder_NC")
```

扫描可用的库

要扫描系统所有可用 PLC 库，只需使用 ITcPlcLibraryManager [▶ 146]::ScanLibraries() [▶ 151] 方法。此方法将返回库的 ITcPlcReferences 集合（ITcPlcLibrary 类型）。

代码片段（C#）：

```
ITcSmTreeItem references = systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1
Project^References");
ITcPlcLibraryManager libManager = (ITcPlcLibraryManager) references;
ITcPlcReferences libraries = libManager.ScanLibraries();
foreach(ITcPlcLibrary library in libraries)
{
// do something
}
```

代码片段（Powershell）：

```
$references = $systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1 Project^References")
$libraries = $references.ScanLibraries()
ForEach( $lib in $libraries )
{
  $lib.Name
}
```

冻结占位符版本

可以将使用的占位符版本冻结为指定版本。这可使用 ITcPlcLibraryManager [▶ 146]::FreezePlaceholder() 方法实现。对指向引用节点的对象调用此方法。

代码片段（C#）：

```
ITcSmTreeItem references = systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1
Project^References");
ITcPlcLibraryManager libManager = (ITcPlcLibraryManager) references;
libManager.FreezePlaceholder(); // freezes the version of all place holders
libManager.FreezePlaceholder("Placeholder_NC"); // freezes the version of a specific place holder
```

代码片段（Powershell）：

```
$references = $systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1 Project^References")
$references.FreezePlaceholder()
$references.FreezePlaceholder("Placeholder_NC")
```

请注意：该版本冻结为有效分辨率。如果有效分辨率指向“*”，则使用系统中的最新版本。

使用存储库

自动化接口提供处理 PLC 库存储库的方法。所有 TwinCAT 安装均包含一个默认存储库。要创建其它存储库，可以使用 ITcPlcLibraryManager [▶ 146]::InsertRepository() [▶ 149] 方法。

代码片段 (C#)：

```
ITcSmTreeItem references = systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1
Project^References");
ITcPlcLibraryManager libManager = (ITcPlcLibraryManager) references;
libManager.InsertRepository("TestRepository", @"C:\Temp", 0);
```

代码片段 (Powershell)：

```
$references = $systemManager.LookupTreeItem("TIPC^Untitled1^Untitled1 Project^References")
$references.InsertRepository("TestRepository", "C:\Temp", 0)
```

在系统中安装新库时，需要将库包含在存储库中。此插入可以使用 ITcPlcLibraryManager [▶ 146]::InstallLibrary() [▶ 149] 方法实现。

代码片段 (C#)：

```
libManager.InstallLibrary("TestRepository", @"C:\SomeFolder\TcTestLibrary.library", false);
```

代码片段 (Powershell)：

```
$references.InstallLibrary("TestRepository", "C:\SomeFolder\TcTestLibrary.library", $false)
```

要从存储库中卸载一个库，使用 ITcPlcLibraryManager [▶ 146]::UninstallLibrary() [▶ 152] 方法。

代码片段 (C#)：

```
libManager.UninstallLibrary("TestRepository", "Tc2_MDP", "*", "Beckhoff Automation GmbH");
```

代码片段 (Powershell)：

```
$references.UninstallLibrary("TestRepository", "Tc2_MDP", "*", "Beckhoff Automation GmbH")
```

要删除一个存储库，使用 ITcPlcLibraryManager [▶ 146]::RemoveRepository() [▶ 150] 方法。

代码片段 (C#)：

```
libManager.RemoveRepository("TestRepository");
```

代码片段 (Powershell)：

```
$references.RemoveRepository("TestRepository")
```

4.3.4.3 访问、创建和处理 PLC POU

本章详细说明如何访问 PLC 对象，例如 POU、方法、转换、属性，以及如何访问相应的实现和声明区域以处理 PLC 代码。还包含如何在 PLCopen XML 中导入/导出 PLC 对象。以下列表显示本文中的所有章节：

- PLC 对象常规信息
- 访问 POU 的实现/声明区
- 访问 Sub-POU (动作、属性...)
- 创建 PLC 对象
- PLC 访问修饰符
- 导入/导出 PLCopen XML
- 导入现有的 POU (模板)

PLC 对象常规信息

尽管每个树项都被视为 ITcSmTreeItem [▶ 103] 类型，但有些项需要被强制转换到更特殊的接口才能访问其所有方法和属性，例如，POU 需要被强制转换到 ITcPlcPou [▶ 141] 接口才能访问其唯一的方法和属性。

代码片段 (C#)：

```
ITcSmTreeItem plcPousItem = systemManager.LookupTreeItem("TIPC^PlcGenerated^PlcGenerated
Project^POUs");
ITcSmTreeItem newFb = plcPousItem.CreateChild("FB_TEST", 604, "", IECLANGUAGETYPES.IECLANGUAGE_ST);
ITcPlcPou fbPou = (ITcPlcPou)newFb;
```

代码片段 (Powershell)：

```
$plcPousItem = $systemManager.LookupTreeItem("TIPC^PlcGenerated^PlcGenerated Project^POUs")
$newFb = $plcPousItem.CreateChild("FB_TEST", 604, "", 6)
```

本示例中，在 PLC 项目中创建了 POU *MAIN*。对象 *programPou* 引用此 POU，现在可以使用 ITcPlcPou 接口的相应方法/属性来访问 POU 的指定方法和属性。

访问 POU 的实现/声明区

还可以使用 ITcPlcImplementation [▶ 142] 或 ITcPlcDeclaration [▶ 142] 接口来获得对 POU 的实现或声明区域的读/写访问权限，如以下示例所示。

代码片段 (C#)：

```
ITcPlcDeclaration fbPouDecl = (ITcPlcDeclaration) fbPou;
ITcPlcImplementation fbPouImpl = (ITcPlcImplementation) fbPou;
string decl = fbPouDecl.DeclarationText;
string impl = fbPouImpl.ImplementationText;
string implXml = fbPouImpl.ImplementationXml;
```

代码片段 (Powershell)：

```
$decl = $newFb.DeclarationText
$impl = $newFb.ImplementationText
$implXml = $newFb.ImplementationXml
```

通过对比两个接口，将发现两个接口的可访问属性互不相同。例如，ITcPlcImplementation 接口提供属性“语言”，而 ITcPlcDeclaration 不提供。这是因为，根据 IEC，声明区域并非基于实际编程语言 (例如 ST)，因此在声明区域使用时，该属性没有意义。

访问 Sub-POU (动作、属性...)

POU 可能有更多子项，例如方法、转换、动作和属性。必须理解并非 POU 的每个子项都同时具有实现和声明区域。例如动作和转换仅有实现区域。因此，只有当相应的子对象具有此区域时，强制转换到上述接口之一才有效。下表概述了不同类型 POU 可用的区域。

树项目	类型	声明区域	实现区域
程序	POU	是	是
功能	POU	是	是
功能块	POU	是	是
动作	POU	否	是
方法	POU	是	是
属性 (获取/设置)	POU	是	是
转换	POU	否	是
Enum	DUT	是	否
Struct	DUT	是	否
Union	DUT	是	否
别名	DUT	是	否
接口	接口	是	否
属性 (获取/设置)	接口	是	是
全局变量	GVL	是	否

创建 PLC 对象

PLC 对象的创建操作简单，可以通过 ITcSmTreeItem [▶ 103] 接口的 CreateChild() [▶ 136] 方法实现。根据 POU 的类型，需要对此方法的参数进行不同的解释。下表提供了创建不同类型 POU 所必需的信息。请注意，如果需要多个参数，则 vInfo 参数可以是字符串数组。每个数组位置及其是否可选如下标记为 [...]。

树项目	类型	参数“nSubType”	参数“vInfo”
程序	POU	602	[0, optional]：IEC 编程语言，由 IECLANGUAGETYPES [▶ 141] 定义。默认使用 ST（结构化文本）。 [1, optional]：可用于衍生（关键词为“扩展”或“实现”）。如果应使用关键词“扩展”和“实现”，则此字段指定关键词“扩展”。 [2, optional]：应扩展/实现的接口或 POU 名称（使用 [1] 时为必填项）。如果应使用关键词“扩展”和“实现”，则此字段将库指定为“扩展”。 [3, optional]：如果应使用关键词“扩展”和“实现”，则此字段指定关键词“实现”。 [4, optional]：如果应使用关键词“扩展”和“实现”，则此字段指定用于衍生的接口。
功能	POU	603	[0]：IEC 编程语言，由 IECLANGUAGETYPES [▶ 141] 定义。 [1]：功能的返回数据类型。可以是 PLC 数据类型，例如 DINT、BOOL...
功能块	POU	604	[0, optional]：IEC 编程语言，由 IECLANGUAGETYPES [▶ 141] 定义。默认使用 ST（结构化文本）。 [1, optional]：可用于衍生（关键词为“扩展”或“实现”）。如果应使用关键词“扩展”和“实现”，则此字段指定关键词“扩展”。 [2, optional]：应扩展/实现的接口或 POU 名称（使用 [1] 时为必填项）。如果应使用关键词“扩展”和“实现”，则此字段将库指定为“扩展”。 [3, optional]：如果应使用关键词“扩展”和“实现”，则此字段指定关键词“实现”。 [4, optional]：如果应使用关键词“扩展”和“实现”，则此字段指定用于衍生的接口。
动作	POU	608	[0, optional]：IEC 编程语言，由 IECLANGUAGETYPES [▶ 141] 定义。默认使用 ST（结构化文本）。 [1, optional]：可能包含 PLCopen XML 字符串及动作代码
方法	POU	609	[0, optional]：IEC 编程语言，由 IECLANGUAGETYPES [▶ 141] 定义。默认使用 ST（结构化文本）。 [1, optional]：返回数据类型 [2, optional]：访问修饰符，例如 PUBLIC。默认使用 PUBLIC。 [3, optional]：可能包含 PLCopen XML 字符串及动作代码
属性	POU	611	[0]：IEC 编程语言，由 IECLANGUAGETYPES [▶ 141] 定义 [1]：返回数据类型 [2, optional]：访问修饰符，例如 PUBLIC。默认使用 PUBLIC。
获取属性	POU	613	[0, optional]：IEC 编程语言，由 IECLANGUAGETYPES [▶ 141] 定义。默认使用 ST（结构化文本）。 [1, optional]：访问修饰符，例如 PUBLIC。默认使用 PUBLIC。 [2, optional]：可能包含 PLCopen XML 字符串及动作代码
设置属性	POU	614	[0, optional]：IEC 编程语言，由 IECLANGUAGETYPES [▶ 141] 定义。默认使用 ST（结构化文本）。 [1, optional]：访问修饰符，例如 PUBLIC。默认使用 PUBLIC。 [2, optional]：可能包含 PLCopen XML 字符串及动作代码
转换	POU	616	[0, optional]：IEC 编程语言，由 IECLANGUAGETYPES [▶ 141] 定义。默认使用 ST（结构化文本）。 [1, optional]：可能包含 PLCopen XML 字符串及动作代码

树项目	类型	参数“nSubType”	参数“vInfo”
Enum	DUT	605	[0, optional]：可能包含声明文本
Struct	DUT	606	[0, optional]：可能包含声明文本
Union	DUT	607	[0, optional]：可能包含声明文本
别名	DUT	623	[0, optional]：可能包含声明文本
接口	接口	618	[0, optional]：扩展类型
属性	接口	612	[0]：返回数据类型
获取属性	接口	654	无需 vInfo 参数，可使用“null”。
设置属性	接口	655	无需 vInfo 参数，可使用“null”。
方法	接口	610	[0]：返回数据类型
全局变量	GVL	615	[0, optional]：可能包含声明文本
PLC 文件夹	文件夹	601	无需 vInfo 参数，可使用“null”。
参数列表	PL	629	[0, optional]：可能包含声明文本
UML 类图	POU	631	---

示例：以下代码片段显示如何使用 vInfo 参数创建功能块“FB_Test”，根据 bool 变量 *bExtends* 和 *bImplements* 的值使用关键词“扩展”和/或“实现”。扩展库为 *ADSRDSTATE*，实现接口为 *ITcADI*。

代码片段 (C#)：

```
string[] vInfo;
bool bExtends = true;
bool bImplements = true;

if (bExtends && bImplements)
{
  vInfo= new string[5];
}
else
{
  if (bExtends || bImplements)
  {
    vInfo= new string[3];
  }
  else
  {
    vInfo= new string[1];
  }
}
vInfo[0] = language.AsString();
if (bExtends && bImplements)
{
  vInfo[1] = "Extends";
  vInfo[2] = "ADSRDSTATE";
  vInfo[3] = "Implements";
  vInfo[4] = "ITcADI";
}
else
{
  if (bExtends)
  {
    vInfo[1] = "Extends";
    vInfo[2] = "ADSRDSTATE";
  }
  else
  {
    if (bImplements)
    {
      vInfo[1] = "Implements";
      vInfo[2] = "ITcADI";
    }
  }
}

ITcSmTreeItem newPOU = parent.CreateChild("FB_Test", 604,
"", fbVInfo);
```

请注意

如果参数 vInfo 只包含一个值（在上表中标记为 [0]），则不应创建大小为 1 的数组，而应使用常规变量。示例：使用 nSubType 618（接口）时，应如下所述使用 vInfo。

代码片段 (C#)：

```
ITcSmTreeItem interface1 = pou.CreateChild("NewInterface1", 618, "", null); // no expansion type
ITcSmTreeItem interface2 = pou.CreateChild("NewInterface2", 618, "", "ITcUnknown"); // expands
ITcUnknown interface
```

PLC 访问修饰符

以下访问修饰符有效并可用作 vInfo 参数，如上表所示：PUBLIC、PRIVATE、PROTECTED、INTERNAL。

导入/导出 PLCopen XML

还可以导入/导出 PLCopen XML 中的 PLC 元素，这可以通过 ITcPlcIECProject [▶ 143] 接口实现。此接口的方法和属性只能直接在 PLC 项目节点上执行，例如：

代码片段 (C#)：

```
ITcSmTreeItem plcProject =
systemManager.LookupTreeItem("TIPC^PlcGenerated^PlcGenerated
Project");
ITcPlcIECProject importExport = (ITcPlcIECProject)
plcProject;
importexport.PlcOpenExport(plcOpenExportFile,
"MyPous.POUProgram;MyPous.POUFunctionBlock");
importExport.PlcOpenImport(plcOpenExportFile,
(int)PLCIMPORTOPTIONS.PLCIMPORTOPTIONS_NONE);
```

代码片段 (Powershell)：

```
$plcProject = $systemManager.LookupTreeItem("TIPC^PlcGenerated^PlcGenerated
Project")
$plcProject.PlcOpenExport(plcOpenExportFile, "MyPous.POUProgram;MyPous.POUFunctionBlock")
$plcProject.PlcOpenImport(plcOpenExportFile,0)
```

此代码片段假定已有一个 PLC 项目添加到 TwinCAT 组态中，该组态通过 *plcProject* 对象引用。此引用被强制转换为类型 ITcPlcIECProject 的对象 *importexport*，随后用于将 POU POUProgram 和 POUFunctionBlock (均位于 PLC 文件夹“MyPOUs”中) 导出到 XML 文件中，然后再次将其导入。

可导入的选项为：None (0)、Rename (1)、Replace (2)、Skip (3)

导入现有的 POU (模板)

PLC 模板可在两个单元使用：可以将整个 PLC 项目作为一个整体使用，也可以将每个 POU 单独用作模板。本章将对后者进行介绍。开发人员选择单个 POU 作为模板的原因之一是，更易于构建现有功能池并将其封装在单独的 POU 中，例如包含不同排序算法的不同功能块。创建项目时，开发人员只需选择在 TwinCAT 项目中需要的功能，然后访问模板工具来检索相应的 POU 并将其导入 PLC 项目。

代码片段 (C#)：

```
ITcSmTreeItem plcProject = systemManager.LookupTreeItem("TIPC^PlcGenerated^PlcGenerated Project");
plcProject.CreateChild("NameOfImportedPOU", 58, null, pathToPouFile);
plcProject.CreateChild(null, 58, null, stringArrayWithPathsToPouFiles);
```

代码片段 (Powershell)：

```
$plcProject = $systemManager.LookupTreeItem("TIPC^PlcGenerated^PlcGenerated Project")
$plcProject.CreateChild("NameOfImportedPOU", 58, $null, $pathToPouFile)
$plcProject.CreateChild($null, 58, $null, $stringArrayWithPathsToPouFiles)
```

请注意：POU 模板文件不仅可以是 .TcPou 文件，也可以是 DUT 和/或 GVL 的相应文件。上面的示例演示了导入 POU 模板文件的两种常见方法。一种是导入单个文件，另一种是通过将文件路径存储到字符串数组并将此字符串数组用作 CreateChild() 的 vInfo 参数来同时导入多个文件。

4.3.5 I/O

4.3.5.1 创建和处理 EtherCAT 设备

本文说明如何通过 TwinCAT 自动化接口建立 EtherCAT 拓扑结构。它包括以下主题：

- 常规信息
- 创建 EtherCAT 设备
- 创建 EtherCAT 接线盒并插入拓扑结构中
- 创建 EtherCAT 终端并插入拓扑结构中
- ItemSubType 9099 的例外情况
- 更改 EtherCAT 终端的“前端口”
- 将 EtherCAT Slave 添加至 HotConnect 组
- 如何添加 EtherCAT SyncUnit
- 处理 EtherCAT 接线盒 (CU1128)

所有这些主题都涉及**离线**组态的情况，这意味着在创建组态时，所有设备的实际地址均未知。因此，本文的最后一个章节将说明如何

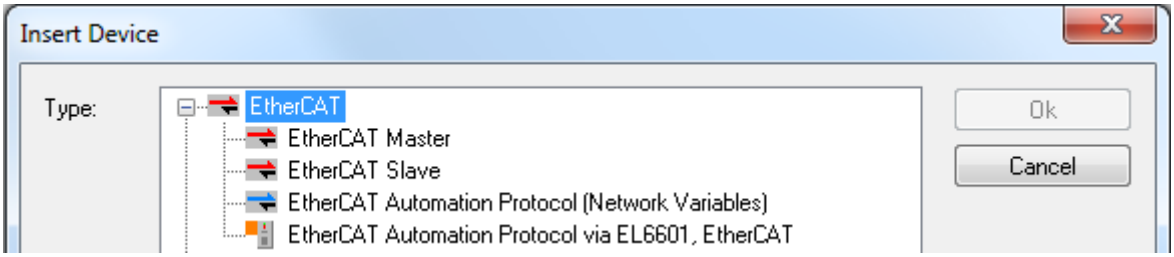
- 激活组态

常规信息

本文档将概述如何通过自动化接口创建和处理 EtherCAT 设备及其相应的拓扑结构。要深入了解 EtherCAT 的工作原理及其通常如何集成到 TwinCAT SystemManager/XAE 中，请参阅 EtherCAT 系统文档以及关于 TwinCAT 中的 EtherCAT 组态的相应章节。连接至 EtherCAT Master 的 EtherCAT 接线盒和终端共享一种通用创建方式，这将在关于 E-Bus 子类型 [▶ 109]的专门章节中进行介绍。

创建 EtherCAT 设备

创建 TwinCAT EtherCAT 组态时，第一步要创建 EtherCAT 设备，其中可能涉及创建 EtherCAT Master、Slave 或自动化协议 (例如使用网络变量，如专门章节 [▶ 66]中所述)。要创建 EtherCAT Master/Slave，只需使用 ITcSmTreeItem [▶ 136]::CreateChild() [▶ 136] 方法及子类型 (EtherCAT Master = 111，EtherCAT Slave = 130) 的相应参数。



EtherCAT Master - 代码片段 (C#)

```
ITcSmTreeItem devices = systemManager.LookupTreeItem("TIID");
ethercatMaster = devices.CreateChild("EtherCAT Master", 111, null, null);
```

EtherCAT Master - 代码片段 (Powershell)

```
$devices = $systemManager.LookupTreeItem("TIID")
$ethercatMaster = $devices.CreateChild("EtherCAT Master", 111, $null, $null)
```

EtherCAT Slave - 代码片段 (C#)

```
ITcSmTreeItem devices = systemManager.LookupTreeItem("TIID");
ethercatSlave = devices.CreateChild("EtherCAT Slave", 130, null, null);
```

EtherCAT Slave - 代码片段 (Powershell)

```
$devices = $systemManager.LookupTreeItem("TIID")
$ethercatSlave = $devices.CreateChild("EtherCAT Slave", 130, $null, $null)
```

创建 EtherCAT 接线盒

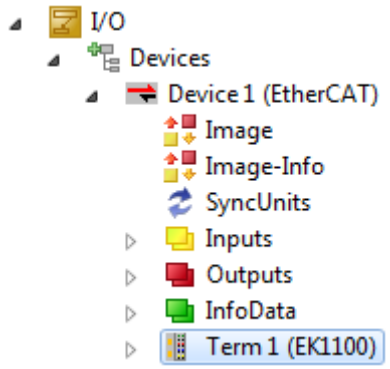
第二步是创建 EtherCAT 接线盒，例如 EK1100 EtherCAT 耦合器。如 E-Bus 子类型 [▶ 109] 章节中所述，EtherCAT Master 的每个子项 (少数例外情况请见下文) 均使用通用子类型 (9099) 并通过产品版本号标识，该版本号需用于 ITcSmTreeItem [▶ 103]::CreateChild() [▶ 136] 方法的 vInfo 参数中。

代码片段 (C#)

```
ITcSmTreeItem ethercatMaster = systemManager.LookupTreeItem("TIID^EtherCAT Master");
ethercatMaster.CreateChild("EK1100", 9099, "", "EK1100-0000-0017");
```

代码片段 (Powershell)

```
$ethercatMaster = $systemManager.LookupTreeItem("TIID^EtherCAT Master")
$ethercatMaster.CreateChild("EK1100", 9099, "", "EK1100-0000-0017")
```



请注意：除了完整的产品版本号之外，还可以使用“通配符”。如果只将“EK1100”指定为 vInfo，自动化接口将自动检测并使用最新版本号。示例：

代码片段 (C#)：

```
ITcSmTreeItem ethercatMaster = systemManager.LookupTreeItem("TIID^EtherCAT Master");
ethercatMaster.CreateChild("EK1100", 9099, "", "EK1100");
```

代码片段 (Powershell)：

```
$ethercatMaster = $systemManager.LookupTreeItem("TIID^EtherCAT Master")
$ethercatMaster.CreateChild("EK1100", 9099, "", "EK1100")
```

创建 EtherCAT 终端并插入拓扑结构中

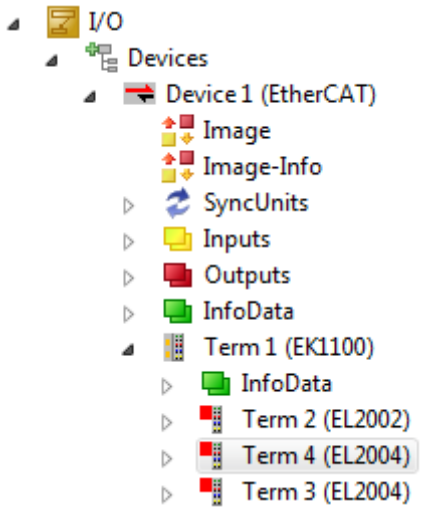
EtherCAT 终端基于与 EtherCAT 接线盒相同的概念创建。各终端也共享通用子类型并将由产品版本号标识，该版本号需用于 ITcSmTreeItem [▶ 103]::CreateChild() [▶ 136] 方法的 vInfo 参数中。可使用参数 bstrBefore 选择终端应插入组态的位置。

代码片段 (C#)：

```
ITcSmTreeItem ek1100 = systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT)^EK1100");
ek1100.CreateChild("Term 4 (EL2004)", 9099, "Term 3 (EL2004)", "EL2004-0000-0017");
```

代码片段 (Powershell)：

```
$ek1100 = $systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT)^EK1100")
$ek1100.CreateChild("Term 4 (EL2004)", 9099, "Term 3 (EL2004)", "EL2004-0000-0017")
```



请注意：如果在使用 bstrBefore 参数时遇到问题，请尝试将终端插入“设备层级”，例如：

代码片段 (C#)：

```
ITcSmTreeItem device= systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT)");
device.CreateChild("Term 4 (EL2004)", 9099, "Term 3 (EL2004)", "EL2004-0000-0017");
```

代码片段 (Powershell)：

```
$device= $systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT)")
$device.CreateChild("Term 4 (EL2004)", 9099, "Term 3 (EL2004)", "EL2004-0000-0017")
```

然后，新终端将插入“Term 3 (EL2004)”之前最后一次插入的 EtherCAT 接线盒下方。

请注意：除了完整的产品版本号之外，还可以使用“通配符”。如果只将“EL2004”指定为 vInfo，自动化接口将自动检测并使用最新版本号。示例：

代码片段 (C#)：

```
ITcSmTreeItem ek1100 = systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT)^EK1100");
ek1100.CreateChild("Term 4 (EL2004)", 9099, "Term 3 (EL2004)", "EL2004");
```

代码片段 (Powershell)：

```
$ek1100 = $systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT)^EK1100")
$ek1100.CreateChild("Term 4 (EL2004)", 9099, "Term 3 (EL2004)", "EL2004")
```

ItemSubType 9099 的例外情况

ItemSubType 9099 有少数例外情况，例如 RS232 终端 EP6002 (ItemSubType 9101) 和 EL600X (ItemSubType 9101)。下表概述了所有例外情况及其相应的 ItemSubType。

I/O	ItemSubType
EP6002	9101
EL6001	9101
EL6002	9101
EP6001-0002	9101
EP6002-0002	9101
EL6021	9103
EL6022	9103
EL6021-0021	9103
BK1120	9081
ILXB11	9086
EL6731	9093
EL6751	9094
EL6752	9095
EL6731-0010	9096
EL6751-0010	9097
EL6752-0010	9098
EL6601	9100
EL6720	9104
EL6631	9106
EL6631-0010	9107
EL6632	9108
EL6652-0010	9109
EL6652	9110

更改 EtherCAT 终端的“前端口”

EtherCAT 终端的前端口确定 EtherCAT 拓扑结构中的终端位置。

在 TwinCAT XAE 中，下拉框自动包含所有可用的前端口。通过自动化接口对此设置进行组态，可以使用 ITcSmTreeItem [▶ 103]::ProduceXml() [▶ 134] 和 ITcSmTreeItem [▶ 103]::ConsumeXml() [▶ 135] 方法修改相应 EtherCAT 终端的 XML 描述 [▶ 22]。因此，此 XML 描述包含一个或多个 XML 节点 <PreviousPort>，其属性“Selected=1”确定当前选择哪个前端口。各个前端口设备均由其物理地址进行标识。

示例 (XML 描述)

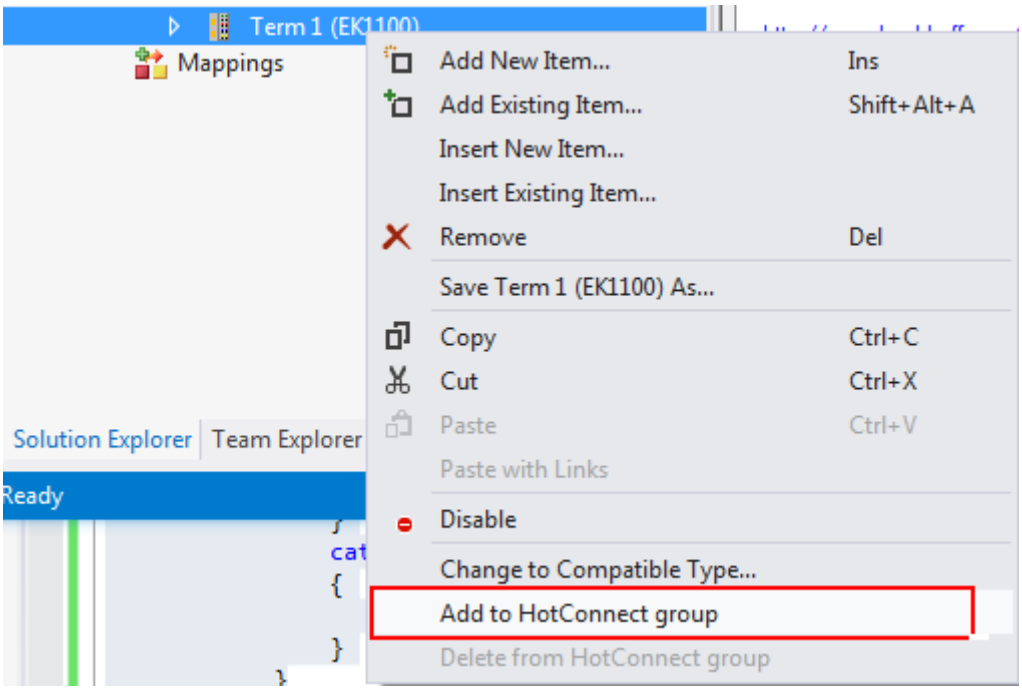
```
<TreeItem>
...
<EtherCAT>
        ...
    <Slave>
        ...
        <PreviousPort Selected="1">
                <Port>B</Port>
                <PhysAddr>1045</PhysAddr>
        </PreviousPort>
        <PreviousPort>
            <Port>B</Port>
                <PhysAddr>1023</PhysAddr>
        </PreviousPort>
        ...
    </Slave>
        ...
<EtherCAT>
...
</TreeItem>
```

如果要更改前端口，需要知道所需设备的 <PhysAddr>，该地址也可以通过其 XML 描述确定。

将 EtherCAT Slave 添加至 HotConnect 组

EtherCAT HotConnect 允许在系统启动之前或运行期间从数据通信中删除或添加预先组态的部分。有关 EtherCAT HotConnect 的详细信息，请阅读 EtherCAT 系统文档。

在 TwinCAT XAE 中，可以通过点击设备上下文菜单中的相应选项，将 EtherCAT Slave 添加到 HotConnect 组。



在 TwinCAT 自动化接口中，可以通过在 EtherCAT Slave 上使用以下 XML 结构实现相同的操作：

示例 (XML 描述)：

```
<TreeItem>
<EtherCAT>
<Slave>
<HotConnect>
    <GroupName>Term 1 (EK1101)</GroupName>
    <GroupMemberCnt>4</GroupMemberCnt>
    <IdentifyCmd>
     <Comment>HotConnect-Identität lesen</Comment>
     <Requires>cycle</Requires>
     <Cmd>1</Cmd>
     <Adp>0</Adp>
     <Ado>4096</Ado>
     <DataLength>2</DataLength>
     <Cnt>1</Cnt>
     <Retries>3</Retries>
     <Validate>
     <Data>0000</Data>
     <Timeout>5000</Timeout>
     </Validate>
    </IdentifyCmd>
</HotConnect>
</Slave>
</EtherCAT>
</TreeItem>
```

请注意：

- <GroupMemberCnt> 标签需要指定具体的终端数量及设备

如何设置 EtherCAT 同步单元

EtherCAT 同步单元可以通过 ITcSmTreeItem::ConsumeXml() 并使用以下 XML 描述进行设置。

示例 (XML 描述)：

```
<TreeItem>
  <EtherCAT>
    <Slave>
      <SyncUnits>
        <SyncUnit>SyncUnit1</SyncUnit>
      </SyncUnits>
    </Slave>
  </EtherCAT>
</TreeItem>
```

处理 EtherCAT 接线盒 (CU1128)

EtherCAT 接线盒的处理方式与所有其他树项相似。以下示例代码显示如何在添加一个 CU1128 接线盒后，再在下面添加两个 EK1100 接线盒：

代码片段 (C#)

```
ITcSmTreeItem cu1128 = ethercatMaster.CreateChild("CU1128", 9099, null, "CU1128");
ITcSmTreeItem cu1128_devA = cu1128.get_Child(1);
ITcSmTreeItem cu1128_devB = cu1128.get_Child(2);
ITcSmTreeItem ek1100_1 = cu1128_devA.CreateChild("EK1100-1", 9099, null, "EK1100");
ITcSmTreeItem ek1100_2 = cu1128_devB.CreateChild("EK1100-2", 9099, null, "EK1100");
```

代码片段 (Powershell)

```
$cu1128 = $ethercatMaster.CreateChild("CU1128", 9099, $null, "CU1128")
$cu1128_devA = $cu1128.get_Child(1)
$cu1128_devB = $cu1128.get_Child(2)
$ek1100_1 = $cu1128_devA.CreateChild("EK1100-1", 9099, $null, "EK1100")
$ek1100_2 = $cu1128_devB.CreateChild("EK1100-2", 9099, $null, "EK1100")
```

激活 EtherCAT 组态

要通过自动化接口激活已创建的 TwinCAT 组态，可以使用 ITcSysManager [▶ 96]::ActivateConfiguration() [▶ 98] 方法。然而，前面的章节只说明了如何创建离线组态，这意味着已创建的所有设备都没有实际地址，例如，EtherCAT Master 尚未链接到物理网络接口卡。因此，激活组态前，需要对具有在线地址的每台设备进行组态。要确定实际地址，可以在在线系统上运行 ScanDevices，通过 XML 描述 (ITcSmTreeItem [▶ 103]::ProduceXml() [▶ 134]) 确定实际地址，然后通过 ITcSmTreeItem [▶ 103]::ConsumeXml() [▶ 135] 将该地址信息导入已创建 (在线) 的组态。有两个“如何操作”示例可帮助您准确进行此类组态：

- 通过自动化接口 Scan Devices [▶ 79]（扫描设备）

4.3.5.2 创建和处理网络变量

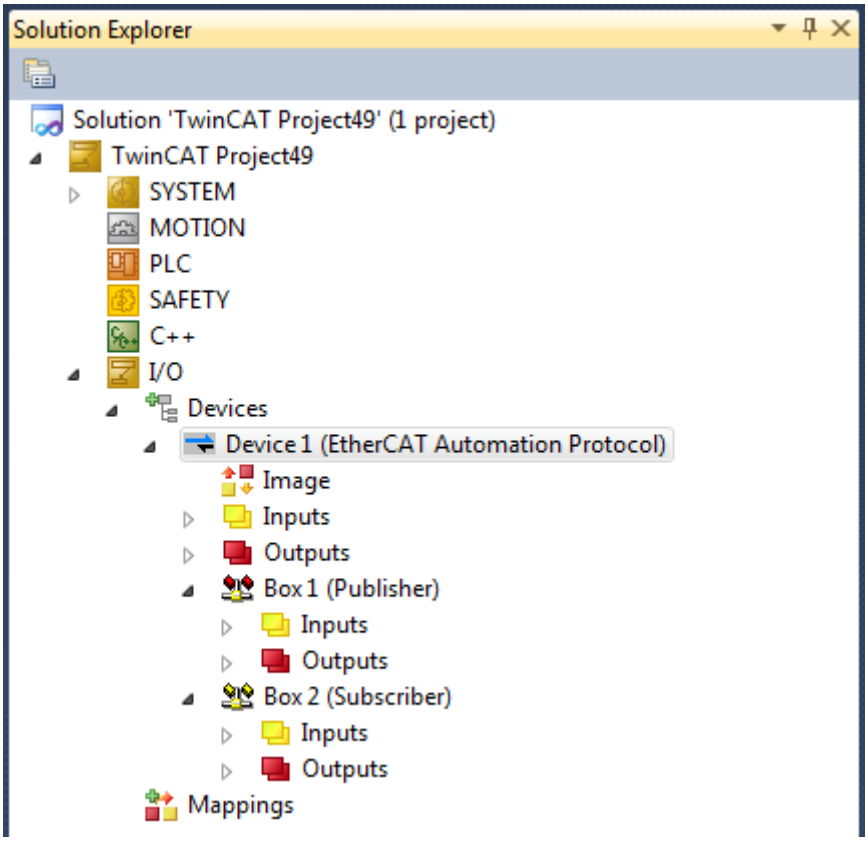
本章将详细说明如何创建和处理网络变量。以下列表显示本文中的所有章节：

- 网络变量常规信息
- 创建 EtherCAT 自动化协议设备
- 创建发布服务器接线盒
- 创建订阅服务器接线盒
- 设置发布服务器/订阅服务器接线盒的参数
- 创建发布服务器变量
- 创建订阅服务器变量
- 链接发布服务器/订阅服务器变量
- 读取发布服务器/订阅服务器变量 ID

请注意，脚本容器 [▶ 158]包含有关如何通过自动化接口创建和组态网络变量的详细示例。

网络变量常规信息

网络变量可用于通过基于 IP 的网络在两个 TwinCAT 设备之间发送数据。一个设备作为“发布服务器”（发送方）声明变量，另一个设备作为“订阅服务器”接收变量值。因此，也将其称为发布服务器/订阅服务器变量。TwinCAT 允许直接在 TwinCAT 项目中对网络变量进行灵活组态，以便将其映射到 PLC 或 I/O。



网络变量使用 EtherCAT 自动化协议设备通过本地网络进行通信。因此，需要添加此设备才能对发布服务器和/或订阅服务器接线盒及相应变量进行组态。

有关网络变量及如何在 TwinCAT 中对其进行组态的详细信息，请参见此处。

创建 EtherCAT 自动化协议设备

要创建 EtherCAT 自动化协议设备，可以结合此设备的相应子类型 (112) 使用 ITcSmTreeItem [▶ 103]::CreateChild() [▶ 136] 方法。

代码片段 (C#)：

```
ITcSmTreeItem devicesNode = systemManager.LookupTreeItem("TIID");
device = devicesNode.CreateChild("Device 1 (EtherCAT Automation Protocol)", 112, null, null);
```

代码片段 (Powershell)：

```
$devicesNode = $systemManager.LookupTreeItem("TIID")
$device = $devicesNode.CreateChild("Device 1 (EtherCAT Automation Protocol)", 112, $null, $null)
```

创建发布服务器接线盒

发布服务器接线盒是发布服务器变量的容器，定义对包含其中的变量应使用哪种类型的通信模式（单点传送、多点传送、广播传送）等。要添加发布服务器接线盒，只需再次结合相应子类型 (9051) 使用 ITcSmTreeItem [▶ 103]::CreateChild() [▶ 136] 方法。

代码片段 (C#)：

```
ITcSmTreeItem eapDevice = systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT Automation
Protocol)");
pubBox = eapDevice.CreateChild("Box 1 (Publisher)", 9051, null, null);
```

代码片段 (Powershell)：

```
$eapDevice = $systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT Automation Protocol)")
$pubBox = $eapDevice.CreateChild("Box 1 (Publisher)", 9051, $null, $null)
```

该小段代码片段将向之前创建的 EtherCAT 自动化协议设备添加发布服务器接线盒。要对此接线盒的通信模式进行组态，需要通过 ITcSmTreeItem [▶ 103]::ConsumeXml() [▶ 135] 方法对其进行自定义设置。有关详细信息，请参见脚本容器 [▶ 158]的 EtherCAT 自动化协议示例或本页下文。

创建订阅服务器接线盒

订阅服务器接线盒是订阅服务器变量的容器，定义对包含其中的变量应使用哪种类型的通信模式（单点传送、多点传送、广播传送）等。要添加发布服务器接线盒，只需再次结合相应子类型 (9052) 使用 ITcSmTreeItem [▶ 103]::CreateChild() [▶ 136] 方法。

代码片段 (C#)：

```
ITcSmTreeItem eapDevice = systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT Automation
Protocol)");
subBox = eapDevice.CreateChild("Box 1 (Subscriber)", 9052, null, null);
```

代码片段 (Powershell)：

```
$eapDevice = $systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT Automation Protocol)")
$subBox = $eapDevice.CreateChild("Box 1 (Subscriber)", 9052, $null, $null)
```

该小段代码片段将向之前创建的 EtherCAT 自动化协议设备添加订阅服务器接线盒。要对此接线盒的通信模式进行组态，需要通过 ITcSmTreeItem [▶ 103]::ConsumeXml() [▶ 135] 方法对其进行自定义设置。有关详细信息，请参见脚本容器 [▶ 158]的 EtherCAT 自动化协议示例或本页下文。

创建发布服务器变量

成功添加发布服务器接线盒后，可以结合子类型 (0) 和 vInfo 所需的参数使用 ITcSmTreeItem [▶ 103]::CreateChild() [▶ 136] 方法来将发布服务器变量添加到此接线盒中，vInfo 将指定发布服务器变量的数据类型，例如“BOOL”。

代码片段 (C#)：

```
ITcSmTreeItem pubBox = systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT Automation
Protocol)^Box 1 (Publisher)");
pubVar = pubBox.CreateChild("MAIN.bTestVar", 0, null, "BOOL");
```

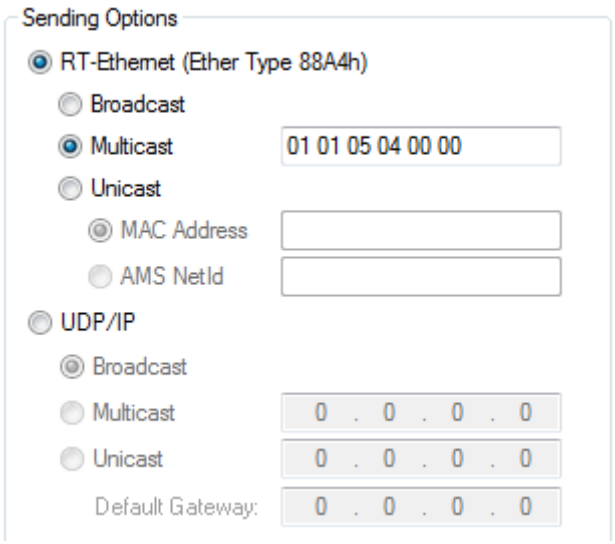
代码片段 (Powershell)：

```
$pubBox = $systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT Automation Protocol)^Box 1
(Publisher)")
$pubVar = $pubBox.CreateChild("MAIN.bTestVar", 0, $null, "BOOL")
```

有关详细信息，请参见脚本容器 [▶ 158]示例“EtherCAT 自动化协议”。

设置发布服务器/订阅服务器接线盒的参数

有两种通信模式需要在发布服务器和/或订阅服务器接线盒中进行组态：**RT-Ethernet** 或 **UDP/IP**。以下截屏显示 TwinCAT XAE 的相应组态选项卡。



有关这些选项的更多详细信息，请参见此处。

要对 **RT-Ethernet** 接线盒进行组态，需要使用 ITcSmTreeItem [▶ 103]::ConsumeXml() [▶ 135] 方法导入以下 XML 结构：

```
<TreeItem>
<BoxDef>
<FieldbusAddress>1</FieldbusAddress>
<AmsAddress>
<AmsPort>0</AmsPort>
<AmsPortTimeout>5</AmsPortTimeout>
</AmsAddress>
<NvPubDef>
<Udp Enabled="false"/>
     <MacAddress>00 00 00 00 00 00</MacAddress>
<IoDiv>
    <Divider>1</Divider>
    <Modulo>0</Modulo>
</IoDiv>
<VLAN>
    <Enable>false</Enable>
    <Id>0</Id>
    <Prio>0</Prio>
</VLAN>
<ArpInterval>1000</ArpInterval>
<DisableSubscriberMonitoring>false</DisableSubscriberMonitoring>
<TargetChangeable>false</TargetChangeable>
</NvPubDef>
</BoxDef>
</TreeItem>
```

下表显示如何根据所需通信模式调整**加粗**标记节点。

RT-Ethernet 通信模式	<PublisherNetId>	<MacAddress>
广播传送	0.0.0.0.0.0	FF FF FF FF FF FF
多点传送	0.0.0.0.0.0	需要包含多点传送 MAC 地址，请参见此处了解更多信息。
单点传送 - MAC 地址	0.0.0.0.0.0	需要包含单点传送 MAC 地址，请参见此处了解更多信息。
单点传送 - AmsNetId	包含目标计算机的 AmsNetId。	00 00 00 00 00 00

如果要使用 UDP/IP，则导入以下 XML 结构：

```
<TreeItem>
<BoxDef>
<FieldbusAddress>1</FieldbusAddress>
<AmsAddress>
<AmsPort>0</AmsPort>
<AmsPortTimeout>5</AmsPortTimeout>
</AmsAddress>
<NvPubDef>
<Udp Enabled="true">
    <Address>0.0.0.0</Address>
    <Gateway>0.0.0.0</Gateway>
</Udp>
<IoDiv>
    <Divider>1</Divider>
    <Modulo>0</Modulo>
</IoDiv>
<VLAN>
    <Enable>false</Enable>
    <Id>0</Id>
    <Prio>0</Prio>
</VLAN>
<ArpInterval>1000</ArpInterval>
<DisableSubscriberMonitoring>false</DisableSubscriberMonitoring>
<TargetChangeable>false</TargetChangeable>
</NvPubDef>
</BoxDef>
</TreeItem>
```

下表显示如何根据所需通信模式调整**加粗**标记节点。在节点 <Udp> 上，需要将属性“Enabled”设置为“true”。

RT-Ethernet 通信模式	<Address>	<Gateway>
广播传送	255.255.255.255	0.0.0.0
多点传送	需要包含多点传送 IP 地址。有关详细信息，请参见此处。	可能包含默认网关，数据包应发送到该网关以进行路由。否则可将其设置为 0.0.0.0
单点传送	需要包含（单点传送）IP 地址。有关详细信息，请参见此处。	可能包含默认网关，数据包应发送到该网关以进行路由。否则可将其设置为 0.0.0.0

创建订阅服务器变量

成功添加发布服务器接线盒后，可以结合子类型 (0) 和 vInfo 所需的参数使用 ITcSmTreeItem [▶ 103]::CreateChild() [▶ 136] 方法来将发布服务器变量添加到此接线盒中，vInfo 将指定发布服务器变量的数据类型，例如“BOOL”。

代码片段 (C#)：

```
ITcSmTreeItem subBox = systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT Automation
Protocol)^Box 1 (Subscriber)");
subVar = pubBox.CreateChild("MAIN.bTestVar", 0, null, "BOOL");
```

代码片段 (Powershell)：

```
$subBox = $systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT Automation Protocol)^Box 1
(Subscriber)")
$subVar = $pubBox.CreateChild("MAIN.bTestVar", 0, $null, "BOOL")
```

有关详细信息，请参见脚本容器 [▶ 158]示例“EtherCAT 自动化协议”。

链接发布服务器/订阅服务器变量

要将发布服务器/订阅服务器变量链接至 PLC 变量，只需使用 ITcSysManager [▶ 96]::Linkvariables() [▶ 99] 方法。

代码片段 (C#)：

```
systemManager.LinkVariales("TIPC^PLC Project^PLC Project Instance^PlcTask Outputs^MAIN.bTestVar",
"TIID^Device 1 (EtherCAT Automation Protocol)^Box 1 (Publisher)^MAIN.bTestVar^Outputs^VarData");
```

代码片段 (Powershell)：

```
$systemManager.LinkVariales("TIPC^PLC Project^PLC Project Instance^PlcTask Outputs^MAIN.bTestVar",
"TIID^Device 1 (EtherCAT Automation Protocol)^Box 1 (Publisher)^MAIN.bTestVar^Outputs^VarData")
```

有关详细信息，请参见脚本容器 [▶ 158]示例“EtherCAT 自动化协议”。

读取发布服务器/订阅服务器变量 ID

以下代码片段读取发布服务器接线盒的所有变量 ID 并将其存储在列表数组“ids”中。这也可用于订阅服务器变量。

代码片段 (C#)：

```
List<uint> ids = new List<uint>();
ITcSmTreeItem pubBox = systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT Automation
Protocol)^Box 1 (Publisher)");
foreach(ITcSmTreeItem var in pubBox)
{
if (var.ItemType == 35) // 35 = publisher variable, 36 = subscriber variable
{
string varStr = var.ProduceXml();
XmlDocument varXml = new XmlDocument();
varXml.LoadXml(varStr);
XmlNode id = varXml.SelectSingleNode("//TreeItem/NvPubVarDef/NvId");
ids.Add(Convert.ToUInt32(id.InnerXml));
}
}
```

代码片段 (Powershell)：

```
$ids = New-Object System.Collections.ArrayList
$pubBox = $systemManager.LookupTreeItem("TIID^Device 1 (EtherCAT Automation Protocol)^Box 1
(Publisher)")
foreach($var in $pubBox)
{
  if($var.ItemType -eq 35)
  {
    $varXml = [Xml]$var.ProduceXml()
    $id = $varXml.TreeItem.NvPubVarDef.NvId
    $ids.Add($id)
  }
}
```

4.3.5.3 创建和处理 Profinet 设备

本文说明如何通过 TwinCAT 自动化接口创建和处理 Profinet I/O 设备。对以下主题进行讨论：

- 创建 Profinet 设备
- 添加 Profinet 接线盒
- 添加 Profinet 模块
- 添加 Profinet 子模块

创建 Profinet 设备

要创建 Profinet I/O 设备 (控制器/设备)，可以使用 ITcSmTreeItem::CreateChild() 方法。子类型设置应添加的实际类型。

Name	子类型
Profinet 控制器 (RT)	113
Profinet 控制器 CCAT (RT)	140
Profinet 控制器 EL6631 (RT, EtherCAT)	119
Profinet 控制器 EL6632 (RT + IRT, EtherCAT)	126
Profinet 设备 (RT)	115
Profinet 设备 CCAT (RT)	142
Profinet 设备 CCAT (RT + IRT)	143
Profinet 设备 EL6631 (RT, EtherCAT)	118

代码片段 (C#)：

```
ITcSmTreeItem io = sysManager.LookupTreeItem("TIID");
ITcSmTreeItem profinetController = io.CreateChild("Profinet Controller", 113, null, null);
```

代码片段 (Powershell)：

```
$io = $sysManager.LookupTreeItem("TIID")
$profinetController = $io.CreateChild("Profinet Controller", 113, $null, $null)
```

添加 Profinet 接线盒

要在 Profinet 设备下创建接线盒，可以使用 ITcSmTreeItem::CreateChild()。子类型取决于应添加的接线盒。下表概述了可能的值：

Name	子类型
BK9102	9125
EK9300	9128
EL6631	9130

此外，要正确使用 vInfo 参数，还需要关于相应 Profinet GSD 文件的一些知识。vInfo 参数由以下语法组成：PathToGSDfile#ModuleIdentNumber#BoxFlags#DAPNumber

ModuleIdentNumber 可以在 GSD 文件中确定，例如通过 XML 结构 <ProfileBody><ApplicationProcess><DeviceAccessPointList><DeviceAccessPointItem ModuleIdentNumber>。ModuleIdentNumber 通常是唯一的。否则，DAPNumber 将指定 DeviceAccessPointList 中的位置。

当前解释以下 BoxFlag：

Name	Value	描述
GENERATE_NAME_FROM_PAB	0x0004	Profinet 名称将通过过程映像生成
GET_STATIONNAME	0x0400	将使用 TC config 中的 Profinet 名称 (树项名)
SET_NOT_IP_TO_OS	0x4000	仅 CE：Profinet IP 不在 OS 中注册

代码片段 (C#)：

```
ITcSmTreeItem profinetEL6631 = profinetController.CreateChild("EL6631", 9130, null, "C:\\TwinCAT\
\3.1\\Config\\Io\\Profinet\\GSDML-V2.31-beckhoff-EL6631-20140508.xml#0x3");
```

代码片段 (Powershell)：

```
$profinetEL6631 = $profinetController.CreateChild("EL6631", 9130, $null, "C:\\TwinCAT\\3.1\\Config\
\Io\\Profinet\\GSDML-V2.31-beckhoff-EL6631-20140508.xml#0x3")
```

添加 Profinet 模块

Profinet 模块在 Profinet 接线盒的 API 节点下创建。将 Profinet 接线盒添加到 TwinCAT 组态时，自动创建 API 节点。要添加 Profinet 模块，可以使用 ITcSmTreeItem::CreateChild() 方法。子类型确定模块在相应 GSD 文件的 <ProfileBody><ApplicationProcess><ModuleList> XML 结构中的位置。

代码片段 (C#)：

```
ITcSmTreeItem profinetEL6631api = profinetEL6631.get_Child(1);
ITcSmTreeItem profinetModule1 = profinetEL6631api.CreateChild("", 30, null, null); // SubType 30 =
200 Byte In-Out
ITcSmTreeItem profinetModule2 = profinetEL6631api.CreateChild("", 12, null, null); // SubType 12 = 8
Byte In-Out
ITcSmTreeItem profinetModule3 = profinetEL6631api.CreateChild("", 8, null, null); // SubType 8 = 4
Byte Out
```

代码片段 (Powershell)：

```
$profinetEL6631api = $profinetEL6631.get_Child(1)
$profinetModule1 = $profinetEL6631api.CreateChild("", 30, $null, $null)
$profinetModule2 = $profinetEL6631api.CreateChild("", 12, $null, $null)
$profinetModule3 = $profinetEL6631api.CreateChild("", 8, $null, $null)
```

添加 Profinet 子模块

Profinet 子模块可以添加到所谓的 Profinet Modular 模块下。处理方法与添加常规 Profinet 模块非常相似。子类型确定子模块在相应 GSD 文件的 <ProfileBody><ApplicationProcess><ModuleList> XML 结构中的位置。

代码片段 (C#)：

```
ITcSmTreeItem profinetModularModule = profinetEL6631api.CreateChild("", 98, null, null); // SubType
98 = Modular
ITcSmTreeItem modularModule1 = profinetModularModule.CreateChild("", 12, null, null); // SubType 12
= 8 Byte In-Out
ITcSmTreeItem modularModule2 = profinetModularModule.CreateChild("", 14, null, null); // SubType 14
= 16 Byte Out
ITcSmTreeItem modularModule3 = profinetModularModule.CreateChild("", 31, null, null); // SubType 31
= 8 Word In
```

代码片段 (Powershell)：

```
$profinetModularModule = $profinetEL6631api.CreateChild("", 98, $null, $null)
$modularModule1 = $profinetModularModule.CreateChild("", 12, $null, $null)
$modularModule2 = $profinetModularModule.CreateChild("", 14, $null, $null)
$modularModule3 = $profinetModularModule.CreateChild("", 31, $null, $null)
```

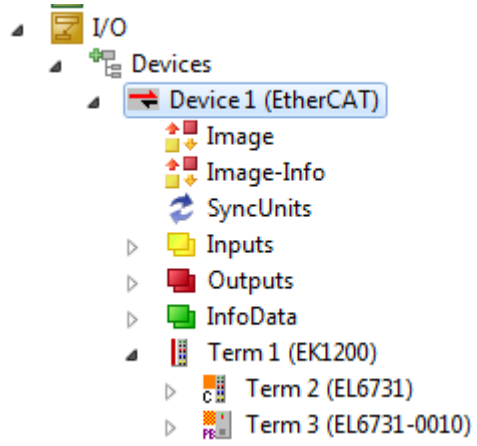
4.3.5.4 创建和处理 Profibus 设备

本文说明如何通过 TwinCAT 自动化接口创建和处理 Profibus Master 和 Slave 设备。包括以下要点：

- 创建和添加 Profibus Master
- 搜索合适的设备 (EL6731、FC310x 等) 并进行组态
- 创建和添加 Profibus Slave
- 搜索合适的 Slave 设备 (EL6731-0010 等) 并进行组态
- 更改现场总线地址

创建和添加 Profibus Master

1. 要创建 Profibus Master 设备，打开新的或现有的 TwinCAT 组态
2. 扫描所有设备。(也可以通过自动化接口执行这些动作。)



3. 创建系统管理器对象并导航至设备

代码片段 (C#)：

```
project = solution.Projects.Item(1);
sysman = (ItcSysManager)project.Object;
ITcSmTreeItem io = (ITcSmTreeItem)sysman.LookupTreeItem("TIID");
```

代码片段 (Powershell)：

```
$project = $sln.Projects.Items(1)
$sysman = $project.Object
$io = $sysman.LookupTreeItem("TIID")
```

要添加 Profibus Master，使用 ITcSmTreeItem:CreateChild 方法。

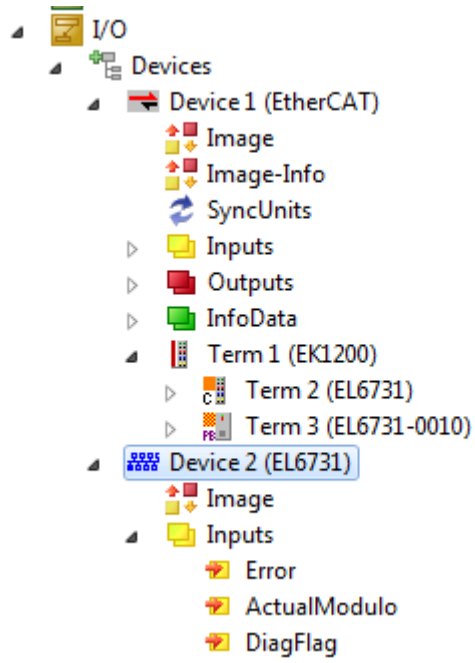
代码片段 (C#)：

```
ITcSmTreeItem5 profi_master = (ITcSmTreeItem5)io.CreateChild("Device 2 (EL6731)", 86, "", null);
```

代码片段 (Powershell)：

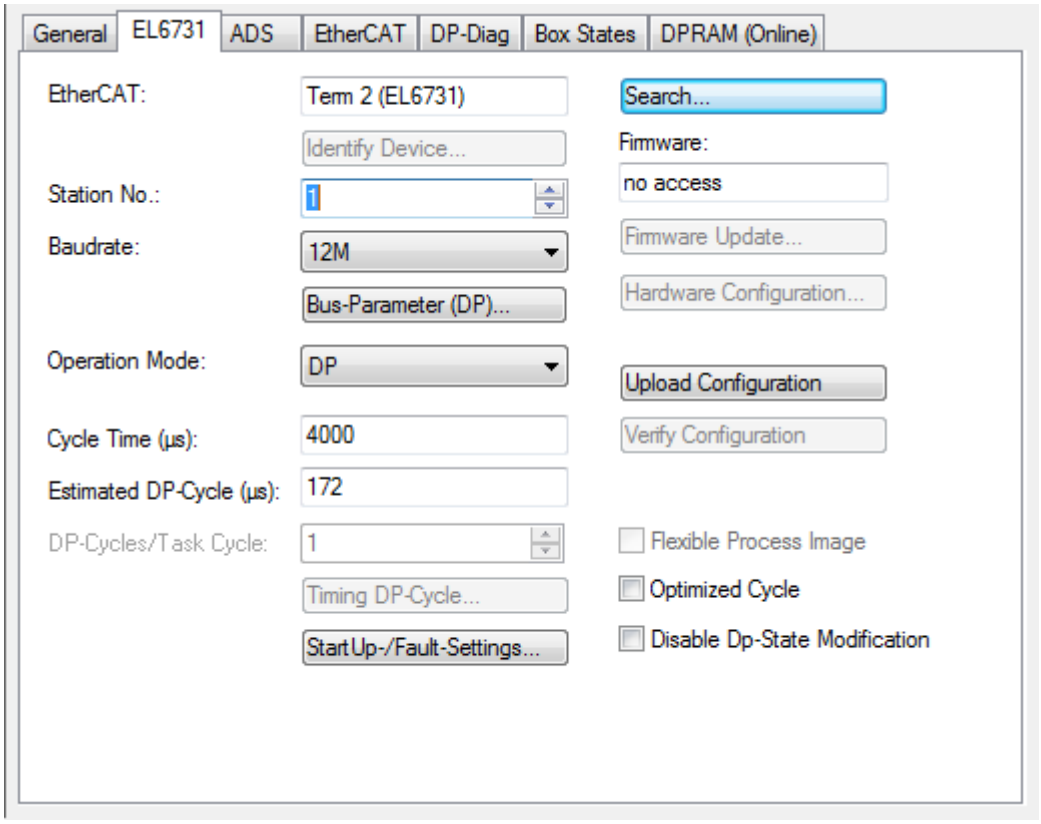
```
$profi_master = $io.CreateChild("Device 2 (EL6731) ", "86", "", $null)
```

对于其它 Profibus Master，输入此处列出的正确 ItemSubtype。这将添加新设备，如截屏所示：



从列表中搜索并声明 Profibus Master 设备：

需要对新添加的 Profibus Master 进行组态，通常在 TwinCAT 中通过按下搜索按钮并从列表中选择正确的设备来完成：



可通过自动化接口实现：

代码片段 (C#)：

```
string availableMaster = profi_master.ResourcesCount;
profi_master.ClaimResources(1);
```

代码片段 (Powershell)：

```
$availableMaster = $profi_master.ResourcesCount
$profi_master.ClaimResources(1)
```

ITcSmTreeItem5:ResourcesCount 给出兼容的 Profibus Master 设备数量，而 ITcSmTreeItem5:ClaimResources 获取要作为主站进行组态的 CANOpen 设备的索引。

创建和添加 Profibus Slave

可以将 Profibus Slave 添加到当前组态中，如下所示：

代码片段 (C#)：

```
ITcSmTreeItem5 profi_slave = (ITcSmTreeItem5)io.CreateChild("Device 3 (EL6731-0010)", 97, null);
```

代码片段 (Powershell)：

```
$profi_slave = $io.CreateChild("Device 3 (EL6731-0010)", "97", "", $null)
```

搜索并声明 Profibus Slave

与 Profibus Master 相似，可以通过以下代码发布 Profibus Slave 的数量：

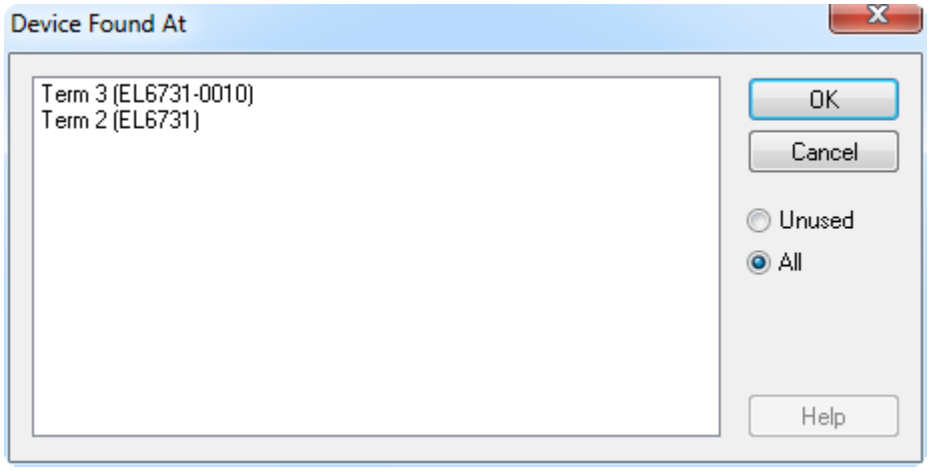
代码片段 (C#)：

```
string availableSlaves = profi_slave.ResourcesCount;
profi_slave.ClaimResource(1);
```

代码片段 (Powershell)：

```
$availableSlaves = $profi_slave.ResourcesCount
$profi_slave.ClaimResources(1)
```

最后一行代码将 EL6731-0010 声明为 Profibus Slave，与 TwinCAT 用户界面中出现的对话框相似。



附图 2:

更改现场总线地址

要在组态中更改 profibus 接线盒的现场总线地址（站号），必须创建 TwinCAT 系统管理器的实例并打开组态。ITcSysManager [▶ 96] 接口的 LookupTreeItem [▶ 101] 方法返回由其路径名 [▶ 9]引用的树项实现的 ITcSmTreeItem [▶ 103] 接口指针。此接口包含树项的 ConsumeXml [▶ 135] 方法。

步骤

有关创建 ITcSysManager [▶ 96] 接口（此处为“*sysMan*”实例）的步骤，请参见访问 TwinCAT 组态 [▶ 17]章节。此接口包含 LookupTreeItem [▶ 101] 方法，可返回指向由其路径名 [▶ 9]指定的特定树项的 ITcSmTreeItem [▶ 103] 指针。要将 profibus 接线盒“*TIID^Device 1 (FC310x)^Box 1 (BK3100)*”的现场总线地址（站号）更改为 44，可以使用以下代码：

代码片段 (C#)：

```
ItcSmTreeItem item = sysMan.LookupTreeItem("TIID^Device 1 (FC310x)^Box 1 (BK3100)");
item.ConsumeXml("44");
```

代码片段 (PowerShell)：

```
$item = $sysMan.LookupTreeItem("TIID^Device 1 (FC310x)^Box 1 (BK3100)")
$item.ConsumeXml("44")
```

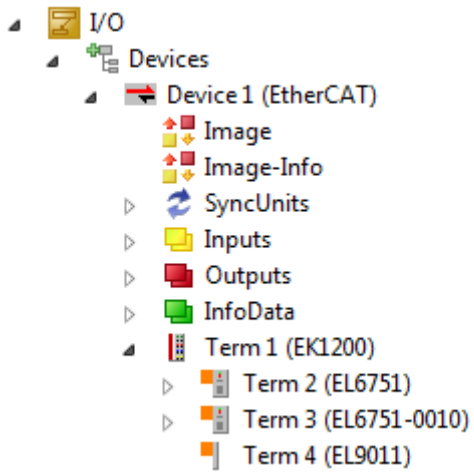
4.3.5.5 创建和处理 CANOpen 设备

本文说明如何通过 TwinCAT 自动化接口创建和处理 CANOpen Master 和 Slave 设备。包括以下要点：

- 创建和添加 CANOpen Master
- 搜索合适的设备 (EL6751、FC510x 等) 并进行组态
- 创建和添加 CANOpen Slave
- 搜索合适的设备 (EL6751-0010 等) 并进行组态
- 通过自动化接口导入 DBC 文件

创建和添加 CANOpen Master

要创建 CANOpen Master 设备，打开新的或现有的 TwinCAT 组态，扫描所有设备。请注意，也可以通过自动化接口执行这些动作。



创建系统管理器对象并导航至设备。

代码片段 (C#)：

```
project = solution.Projects.Item(1);
sysman = (ITcSysManager)project.Object;
ITcSmTreeItem io = (ITcSmTreeItem)sysman.LookupTreeItem("TIID");
```

代码片段 (Powershell)：

```
$project = $sln.Projects.Items(1)
$sysman = $project.Object
$io = $sysman.LookupTreeItem("TIID")
```

要添加 CANOpen Master，则使用 ITcSmTreeItem:CreateChild 方法：

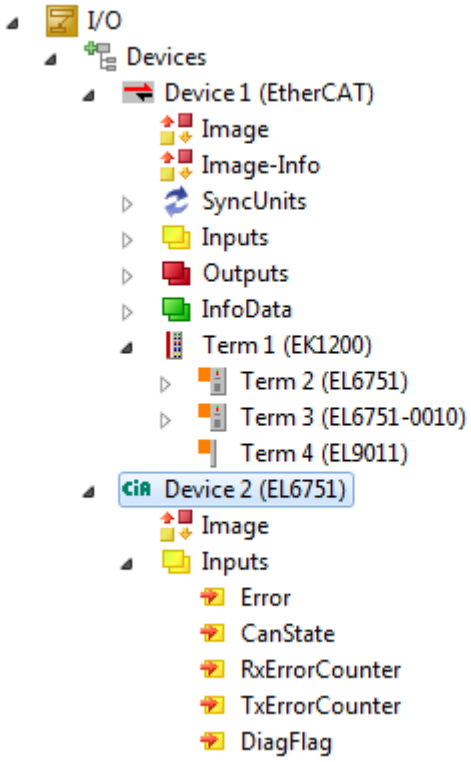
代码片段 (C#)：

```
ITcSmTreeItem5 can_master = (ITcSmTreeItem5)io.CreateChild("Device 2 (EL6751)", 87, "", null);
```

代码片段 (Powershell)：

```
$can_master = $io.CreateChild("Device 2 (EL6751)", "87", "", $null)
```

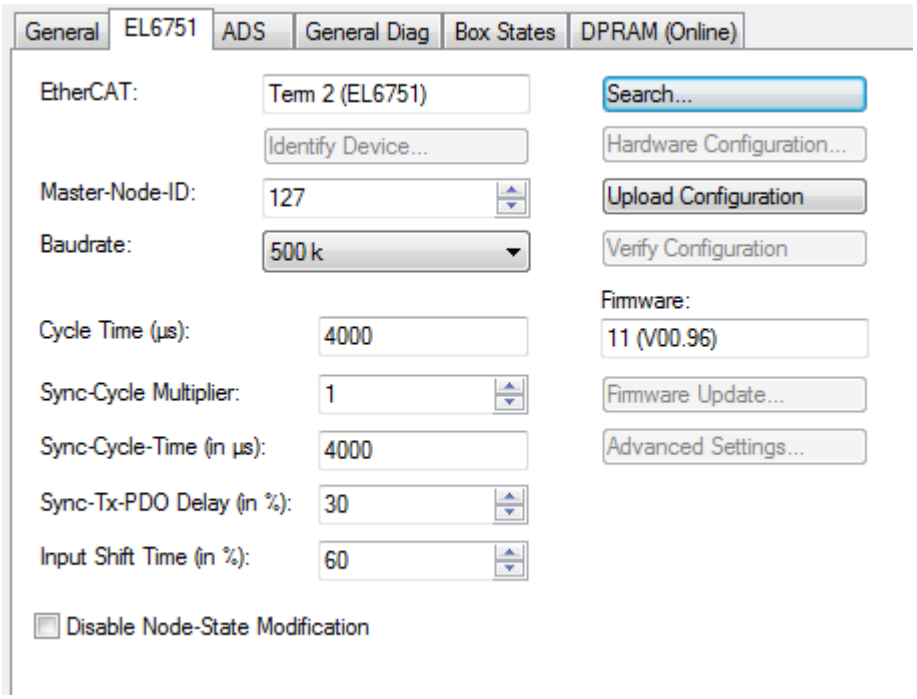
对于其它 CANOpen Master 设备，则输入此处 [▶ 111]列出的正确 ItemSubtype。这将添加新设备，如截屏所示：



附图 3: 已添加的 CANOpen Master

从列表中搜索并声明 CANOpen Master 设备：

需要对新添加的 CANOpen Master 进行组态，通常在 TwinCAT 中通过按下搜索按钮并从列表中选择正确的设备来完成：



附图 4: 通过搜索按钮选择 CANOpen Master

可通过自动化接口实现：

代码片段 (C#)：

```
String availableMaster = profi_master.ResourceCount;
can_master.ClaimResources(1);
```

代码片段 (Powershell)：

```
$availableMaster = $can_master.ResourceCount
$can_master.ClaimResources(1)
```

ITcSmTreeItem5:ResourceCount 给出兼容的 CANOpen Master 设备数量，而 ITcSmTreeItem5:ClaimResources 获取要作为主站进行组态的 CANOpen 设备的索引。

创建和添加 CANOpen Slave

可以将 CANOpen Slave 添加到当前组态中，如下所示：

代码片段 (C#)：

```
ITcSmTreeItem5 can_slave = (ITcSmTreeItem5)io.CreateChild("Device 3 (EL6751-0010)", "98", null);
```

代码片段 (Powershell)：

```
$can_slave = $io.CreateChild("Device 3 (EL6751-0010)", "98", $null)
```

<b />

搜索并声明 CANOpen Slave

与 CANOpen Master 相似，可以通过以下代码发布 CANOpen Slave 列表：

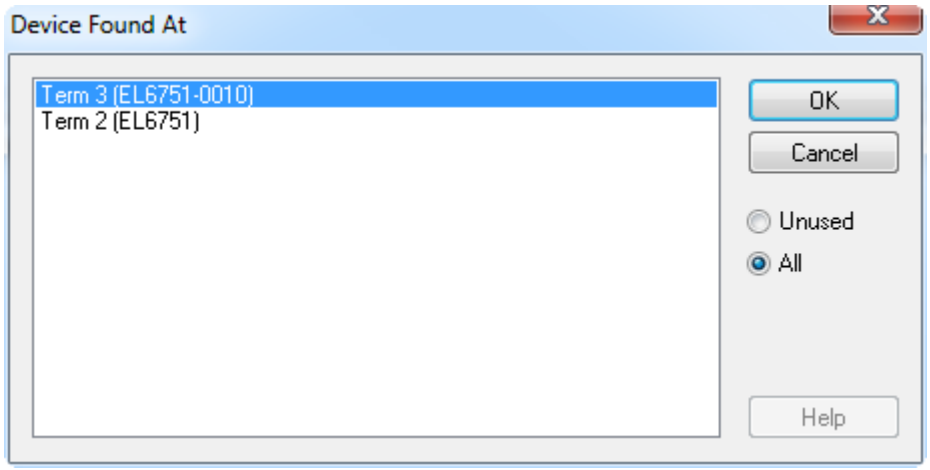
代码片段 (C#)：

```
string availableSlaves = can_slave.ResourceCount;
can_slave.Claimresources(1);
```

代码片段 (Powershell)：

```
$availableSlaves = $can_slave.ResourceCount
$can_slave.Claimresources(1)
```

最后一行代码将 EL6751-0010 声明为 CANOpen Slave，与 TwinCAT 用户界面中出现的对话框相似。

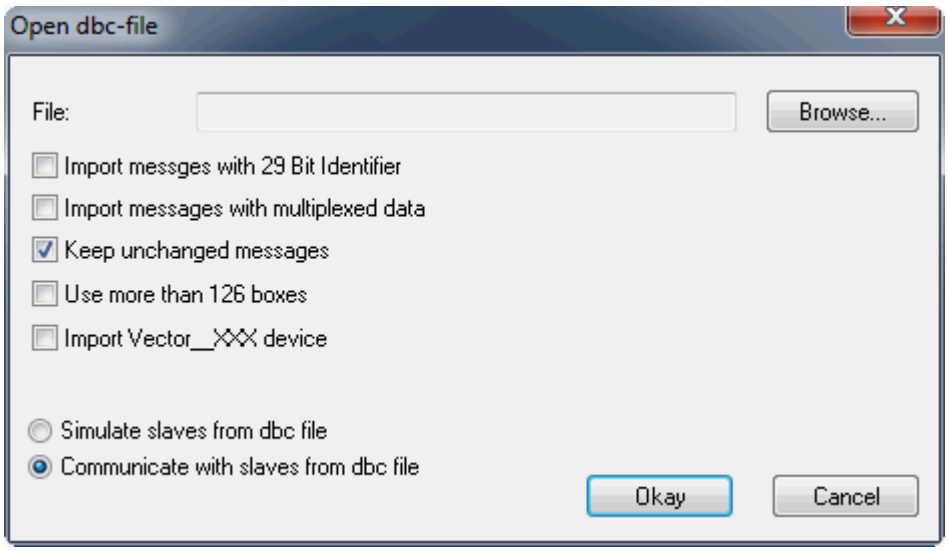


附图 5: CANOpen 通过对话框声明资源

通过自动化接口导入 DBC 文件

所需版本：TwinCAT 3.1 Build 4018 或更高版本

TwinCAT 提供要通过对话框导入的 DBC 文件，如截屏所示。右击 CANOpen Master，例如 EL6751，并选择“Import dbc-file” (导入 dbc 文件)，将有一个对话框提示用户浏览 DBC 文件。单击确认后，将自动加载 CANOpen 组态。



附图 6: CANOpen DBC 文件导入

自动化接口也支持 DBC 文件导入。为此，导航至 CANOpen Master 树项并通过 ITcSmTreeItem:ProduceXml(). 导出 Xml 文件。在 Xml 文件中，添加要导入的 DBC 文件的路径及上面所示对话框中的附加属性。

```
<DeviceDef>
<AmsPort>28673</AmsPort>
<AmsNetId>172.17.251.109.2.1</AmsNetId>
<AddressInfo>
<Ecat>
<EtherCATDeviceId>0</EtherCATDeviceId>
</Ecat>
</AddressInfo>
<MaxBoxes>126</MaxBoxes>
<ScanBoxes>false</ScanBoxes>
<CanOpenMaster>
<ImportDbcFile>
<FileName>c:\dbc_file_folder\dbc_file_to_be_imported.dbc</FileName>
<ImportExtendedMessages>false</ImportExtendedMessages>
<ImportMultiplexedDataMessages>false</ImportMultiplexedDataMessages>
<KeepUnchangedMessages>true</KeepUnchangedMessages>
<ExtBoxesSupport>false</ExtBoxesSupport>
<VectorXXXSupport>false</VectorXXXSupport>
<CommunicateWithSlavesFromDbcFile>true</CommunicateWithSlavesFromDbcFile>
</ImportDbcFile>
</CanOpenMaster>
<Fcxxxx>
<CalculateEquiTimes>0</CalculateEquiTimes>
<NodeId>127</NodeId>
<Baudrate>500 k</Baudrate>
<DisableNodeStateModification>false</DisableNodeStateModification>
</Fcxxxx>
</DeviceDef>
```

通过 ITcSmTreeImte:ConsumeXml() 将修改后的 Xml 文件导回 TwinCAT 组态中。现在，组态将加载到 CANOpen Master 下。

4.3.5.6 创建和处理 Devicenet 设备

与其它 I/O 组件一样，Devicenet 设备可以使用 ITcSmTreeItem 接口的 CreateChild() 方法通过 TwinCAT 自动化接口添加。子类型指定应添加的设备。

代码片段 (C#)：

```
ITcSmTreeItem io = sysManager.LookupTreeItem("TIID");
ITcSmTreeItem devicenet1 = io.CreateChild("Device 1 (EL6752)", 88, null, null);
ITcSmTreeItem devicenet2 = io.CreateChild("Device 2 (EL6752-0010)", 99, null, null);
```

代码片段 (Powershell)：

```
$io = $sysManager.LookupTreeItem("TIID")
$devicenet1 = $io.CreateChild("Device 1 (EL6752)", 88, $null, $null)
$devicenet2 = $io.CreateChild("Device 2 (EL6752-0010)", 99, $null, $null)
```

下表提供了所有 Devicenet I/O 设备及其相应子类型的概述。如果一台设备缺失，可以随时按照树项的 XML 描述 [▶ 22]章节所述找到子类型。

设备	子类型
Devicenet Master FC52xx PCI	41
Devicenet Master EL6752 EtherCAT	88
Devicenet Slave FC52xx PCI	62
Devicenet Slave EL6752 EtherCAT	99
Devicenet Master CX1500-M520 PC104	73
Devicenet Slave CX1500-B520-PC104	74
Devicenet Monitor FC52xx PCI	59

也可以通过 CreateChild() 附加 Devicenet 接线盒。子类型指定应添加的接线盒。

代码片段 (C#)：

```
ITcSmTreeItem devicenet1box = devicenet1.CreateChild("Box 2 (EL6752-0010)", 5203, null, null);
```

代码片段 (Powershell)：

```
$devicenet1box = $devicenet1.CreateChild("Box 2 (EL6752-0010)", 5203, $null, $null)
```

要将变量添加到 Devicenet 接线盒，只需使用以下代码片段。vInfo 参数指定此变量的数据类型。

代码片段 (C#)：

```
ITcSmTreeItem inputVars = devicenet1box.LookupChild("Inputs");
inputVars.CreateChild("TestVarInt", 0, null, "INT");
inputVars.CreateChild("TestVarBool", 0, null, "BOOL");
```

代码片段 (Powershell)：

```
$inputVars = $devicenet1box.LookupChild("Inputs")
$inputVars.CreateChild("TestVarInt", 0, $null, "INT")
$inputVars.CreateChild("TestVarBool", 0, $null, "BOOL")
```

4.3.5.7 扫描设备和接线盒

创建新组态时，通常需要根据实际可用的硬件调整 TwinCAT XAE 组态。为此，可以重新开始进行新的 TwinCAT XAE 组态，并进行以下步骤：

- 创建新的 TwinCAT XAE 组态
- 设置目标系统的地址
- 扫描可用的设备
- 添加要使用的设备并参数化
- 扫描并插入每台设备的接线盒

步骤

创建 ITcSysManager [▶ 96] 接口（此处为“*systemManager*”接口）的步骤请参见访问 TwinCAT 组态 [▶ 17]章节。此接口包含 LookupTreeItem [▶ 101] 方法，可返回指向由其路径名 [▶ 9]指定的特定树项的 ITcSmTreeItem [▶ 103] 指针，在本例中是引用 I/O 设备节点的快捷方式“TIID”。

代码片段 (C#)：

```
ITcSysManager3 systemManager = null;

public void ScanDevicesAndBoxes()
{
  systemManager.SetTargetNetId("1.2.3.4.5.6");
  ITcSmTreeItem ioDevicesItem = systemManager.LookupTreeItem("TIID");
  string scannedXml = ioDevicesItem.ProduceXml(false);
  XmlDocument xmlDoc = newXmlDocument();
  xmlDoc.LoadXml(scannedXml);
  XmlNodeList xmlDeviceList = xmlDoc.SelectNodes("TreeItem/DeviceGrpDef/FoundDevices/Device");
  List<ITcSmTreeItem> devices = newList<ITcSmTreeItem>();
  int deviceCount = 0;
  foreach (XmlNode node in xmlDeviceList)
  {
    int itemSubType = int.Parse(node.SelectSingleNode("ItemSubType").InnerText);
    string typeName = node.SelectSingleNode("ItemSubTypeName").InnerText;
    XmlNode xmlAddress = node.SelectSingleNode("AddressInfo");
    ITcSmTreeItem device = ioDevicesItem.CreateChild(string.Format("Device_{0}",+
+deviceCount),itemSubType,string.Empty,null);
    string xml = string.Format("<TreeItem><DeviceDef>{0}</DeviceDef></
TreeItem>",xmlAddress.OuterXml);
    device.ConsumeXml(xml);
    devices.Add(device);
  }
  foreach (ITcSmTreeItem device in devices)
  {
    string xml = "<TreeItem><DeviceDef><ScanBoxes>1</ScanBoxes></DeviceDef></TreeItem>";
    try
    {
      device.ConsumeXml(xml);
    }
    catch (Exception ex)
    {
      Console.WriteLine("Warning: {0}",ex.Message);
    }
    foreach (ITcSmTreeItem box in device)
    {
      Console.WriteLine(box.Name);
    }
  }
}
```

4.3.5.8 启用和禁用 I/O 设备

要在组态中禁用/启用树项，必须创建 TwinCAT 系统管理器的实例并打开组态。ITcSysManager [▶ 96] 接口的 LookupTreeItem [▶ 101] 方法返回由其路径名 [▶ 9]引用的树项实现的 ITcSmTreeItem [▶ 103] 接口指针。此接口包含树项的 Disabled [▶ 103] 属性。

步骤

创建 ITcSysManager [▶ 96] 接口（此处为“*sysMan*”实例）的步骤请参见访问 TwinCAT 组态 [▶ 17]章节。此接口包含 LookupTreeItem [▶ 101] 方法，可返回指向由其路径名 [▶ 9]指定的特定树项的 ITcSmTreeItem [▶ 103] 指针。要禁用/启用树项“*TIID^EtherCAT Master*”，可以使用以下代码片段。

示例 (CSharp)：

```
ITcSmTreeItem item = sysMan.LookupTreeItem("TIID^EtherCAT Master");
item.Disabled = DISABLED_STATE.SMDS_DISABLED;
```

请注意，在本示例中，需要为项目中的“Microsoft Developer Environment 10.0”和“Beckhoff TCatSysManager Library 1.1”添加引用。

示例 (PowerShell)：

```
$DISABLED_STATE = @{"SMDS_NOT_DISABLED" = "0"; "SMDS_DISABLED" = "1"}
$item = $systemManager.LookupTreeItem("TIID^EtherCAT Master")
$item.Disabled = $DISABLED_STATE.SMDS_DISABLED
```

示例 (VBScript):

```
dim dte,sln,proj,sysMan
set dte = CreateObject("VisualStudio.DTE.10.0")
set sln = dte.Solution
call sln.Open("C:\SolutionFolder\MySolution1.sln")
set proj = sln.Projects(1)
set sysMan = proj.Object
set item = sysMan.LookupTreeItem("TIID^EtherCAT Master")
item.Disabled = SMDS_DISABLED '(oder item.Disabled = SMDS_NOT_DISABLED um Strukturelement zu
aktivieren)
```

4.3.6 TcCOM

4.3.6.1 创建和处理 TcCOM 模块

本章说明如何将现有的 TcCOM 模块添加至 TwinCAT 组态并进行参数化。本章将简要介绍以下主题：

- 获取对“TcCOM Objects”节点的引用
- 添加现有的 TcCOM 模块
- 对已添加的 TcCOM 模块进行迭代
- 为参数设置 CreateSymbol 标签
- 为数据区域设置 CreateSymbol 标签
- 设置上下文 (任务)
- 链接变量

获取对“TcCOM Objects”节点的引用

在 TwinCAT 组态中，“TcCOM Objects”节点位于“SYSTEM^TcCOM Objects”下。因此，可以通过以下方式使用 ITcSysManager::LookupTreeItem() 方法获取对此节点的引用：

代码片段 (C#):

```
ITcSmTreeItem tcComObjects = systemManager.LookupTreeItem("TIRC^TcCOM Objects");
```

代码片段 (Powershell):

```
$tcComObjects = $systemManager.LookupTreeItem("TIRC^TcCOM Objects")
```

上述节点假定 AI 代码中已存在 systemManager 对象。

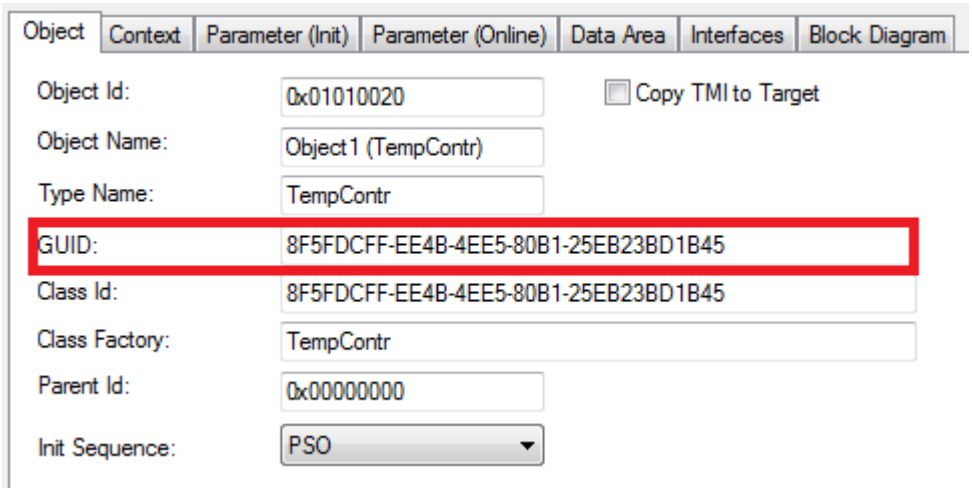
添加现有的 TcCOM 模块

要将现有的 TcCOM 模块添加到 TwinCAT 组态中，需要由 TwinCAT 检测到这些模块。使用如下方法之一可达此目的：

- 将 TcCOM 模块复制到文件夹 %TWINCAT3.XDIR”\CustomConfig\Modules\
- 编辑 %TWINCAT3.XDIR”\Config\Io\TcModuleFolders.xml，为所选文件夹添加路径，并将模块置于此文件夹中

两种方法都可以使 TwinCAT 检测到 TcCOM 模块。

TcCOM 模块由其 GUID 进行标识。此 GUID 可用于将 TcCOM 模块通过 ITcSmTreeItem::CreateChild() 方法添加到 TwinCAT 组态中。GUID 可以在 TwinCAT XAE 中通过 TcCOM 模块的属性页或通过编程方式来确定，这将在本章下文中介绍。



或者，也可以通过 TcCOM 模块的 TMC 文件确定 GUID。

```
<TcModuleClass>
  <Modules>
    <Module GUID="{8f5fdcff-ee4b-4ee5-80b1-25eb23bd1b45}">
    ...
    </Module>
  </Modules>
</TcModuleClass>
```

假定已具有在 TwinCAT 中注册并可由其检测到的 TcCOM 模块。现在要将这个具有 GUID {8F5FDCFF-EE4B-4EE5-80B1-25EB23BD1B45} 的 TcCOM 模块添加至 TwinCAT 组态。使用如下方法可达此目的：

代码片段 (C#)：

```
Dictionary<string,Guid> tcomModuleTable = new Dictionary<string,Guid>();
tcomModuleTable.Add("TempContr",Guid.Parse("{8f5fdcff-ee4b-4ee5-80b1-25eb23bd1b45}"));
ITcSmTreeItem tempController = tcComObjects.CreateChild("Test", 0, "",
tcomModuleTable["TempContr"]);
```

代码片段 (Powershell)：

```
$tcomModuleTable = @""
$tcomModuleTable.Add("TempContr", "{8f5fdcff-ee4b-4ee5-80b1-25eb23bd1b45}")
$tempController = $tcComObjects.CreateChild("Test", 0, "", $tcomModuleTable["TempContr"])
```

请注意，ItcSmTreeItem::CreateChild() 方法的 vInfo 参数包含 TcCOM 模块的 GUID，此 GUID 用于标识所有已在该系统中注册的 TcCOM 模块列表中的模块。

对已添加的 TcCOM 模块进行迭代

要对所有已添加的 TcCOM 模块实例进行迭代，可以使用 ITcModuleManager2 接口。以下代码片段显示如何使用此接口。

代码片段 (C#)：

```
ITcModuleManager2 moduleManager = (ITcModuleManager2)systemManager.GetModuleManager();
foreach (ITcModuleManager2 moduleInstance in moduleManager)
{
  string moduleType = moduleInstance.ModuleTypeName;
  string instanceName = moduleInstance.ModuleInstanceName;
  Guid classId = moduleInstance.ClassID;
  uint objId = moduleInstance.oid;
  uint parentObjId = moduleInstance.ParentOID;
}
```

代码片段 (Powershell)：

```
$moduleManager = $systemManager.GetModuleManager()
ForEach( $moduleInstance in $moduleManager )
{
  $moduleType = $moduleInstance.ModuleTypeName
  $instanceName = $moduleInstance.ModuleInstanceName
  $classId = $moduleInstance.ClassID
  $objId = $moduleInstance.oid
  $parentObjId = $moduleInstance.ParentOID
}
```

请注意，每个模块对象也可以解释为 ITcSmTreeItem，因此以下类型转换将生效：

代码片段 (C#)：

```
ITcSmTreeItem treeItem = moduleInstance As ITcSmTreeItem;
```

请注意：Powershell 默认使用动态数据类型。

为参数设置 CreateSymbol 标签

TcCOM 模块参数的 CreateSymbol (CS) 标签可以通过 XML 描述设置。以下代码片段显示如何激活参数“CallBy”的 CS 标签。

代码片段 (C#)：

```
bool activateCS = true;
// First step: Read all Parameters of TcCOM module instance
string tempControllerXml = tempController.ProduceXml();
XmlDocument tempControllerDoc = new XmlDocument();
tempControllerDoc.LoadXml(tempControllerXml);
XmlNode sourceParameters = tempControllerDoc.SelectSingleNode("TreeItem/TcModuleInstance/Module/
Parameters");

// Second step: Build target XML (for later ConsumeXml())
XmlDocument targetDoc = new XmlDocument();
XmlElement treeItemElement = targetDoc.CreateElement("TreeItem");
XmlElement moduleInstanceElement = targetDoc.CreateElement("TcModuleInstance");
XmlElement moduleElement = targetDoc.CreateElement("Module");
XmlElement parametersElement = (XmlElement) targetDoc.ImportNode(sourceParameters, true);
moduleElement.AppendChild(parametersElement);
moduleInstanceElement.AppendChild(moduleElement);
treeItemElement.AppendChild(moduleInstanceElement);
targetDoc.AppendChild(treeItemElement);

// Third step: Look for specific parameter (in this case "CallBy") and read its CreateSymbol
attribute
XmlNode destModule = targetDoc.SelectSingleNode("TreeItem/TcModuleInstance/Module ");
XmlNode callByParameter = destParameters.SelectSingleNode("Parameters/Parameter[Name='CallBy']");
XmlAttribute createSymbol = callByParameter.Attributes["CreateSymbol"];

createSymbol.Value = "true";

// Fifth step: Write prepared XML to configuration via ConsumeXml()
string targetXml = targetDoc.OuterXml;
tempController.ConsumeXml(targetXml);
```

代码片段 (Powershell)：

```
$tempControllerXml = [Xml]$tempController.ProduceXml()
$sourceParameters = $tempControllerXml.TreeItem.TcModuleInstance.Module.Parameters

[System.XML.XmlDocument] $targetDoc = New-Object System.XML.XmlDocument
[System.XML.XmlElement] $treeItemElement = $targetDoc.CreateElement("TreeItem")
[System.XML.XmlElement] $moduleInstanceElement = $targetDoc.CreateElement("TcModuleInstance")
[System.XML.XmlElement] $moduleElement = $targetDoc.CreateElement("Module")
[System.XML.XmlElement] $parametersElement = $targetDoc.ImportNode($sourceParameters, $true)
$moduleElement.AppendChild($parametersElement)
$moduleInstanceElement.AppendChild($moduleElement)
$treeItemElement.AppendChild($moduleInstanceElement)
$targetDoc.AppendChild($treeItemElement)

$destModule = $targetDoc.TreeItem.TcModuleInstance.Module
$callByParameter = $destmodule.SelectSingleNode("Parameters/Parameter[Name='CallBy']")

$callByParameter.CreateSymbol = "true"

$targetXml = $targetDoc.OuterXml
$tempController.ConsumeXml($targetXml)
```

为数据区域设置 CreateSymbol 标签

TcCOM 模块数据区域的 CreateSymbol (CS) 标签可以通过 XML 描述设置。以下代码片段显示如何激活数据区域“Input”的 CS 标签。请注意，该步骤与对参数的操作步骤基本相同。

代码片段 (C#)：

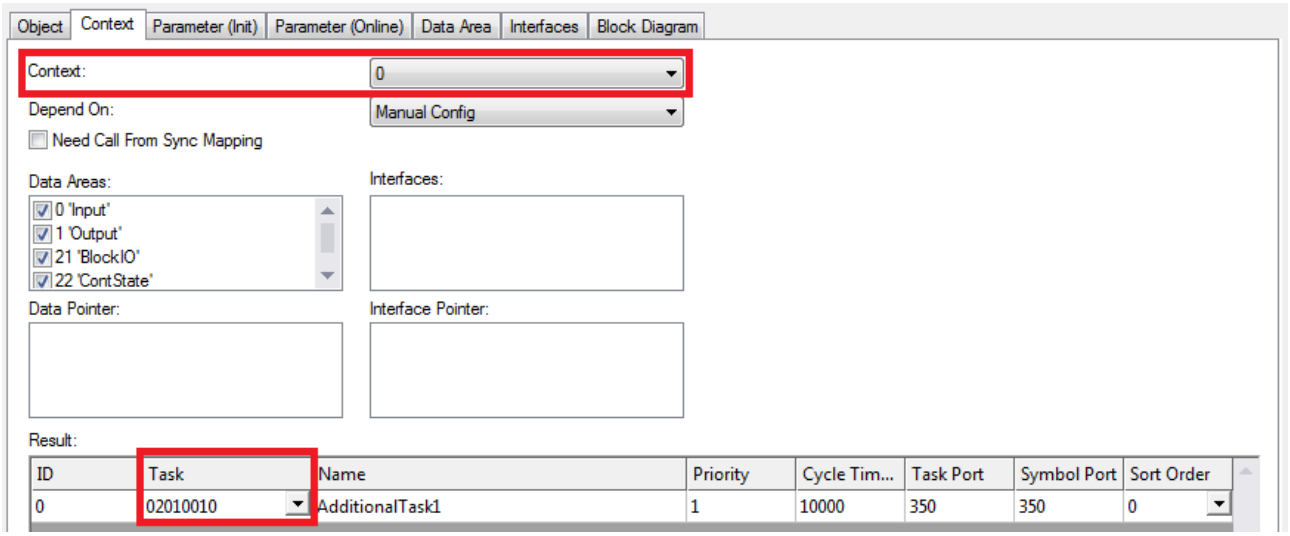
```
bool activateCS = true;
// First step: Read all Data Areas of a TcCOM module instance
string tempControllerXml = tempController.ProduceXml();
XmlDocument tempControllerDoc = new XmlDocument();
tempControllerDoc.LoadXml(tempControllerXml);
XmlNode sourceDataAreas = tempControllerDoc.SelectSingleNode("TreeItem/TcModuleInstance/Module/
DataAreas");

// Second step: Build target XML (for later ConsumeXml())
XmlDocument targetDoc = new XmlDocument();
XmlElement treeItem = targetDoc.CreateElement("TreeItem");
XmlElement moduleInstance = targetDoc.CreateElement("TcModuleInstance");
XmlElement module = targetDoc.CreateElement("Module");
XmlElement dataAreas = (XmlElement)
targetDoc.ImportNode(sourceDataAreas, true);
module.AppendChild(dataAreas);
moduleInstance.AppendChild(module);
treeItem.AppendChild(moduleInstance);
targetDoc.AppendChild(treeItem);

// Third step: Look for specific Data Area (in this case "Input") and read its CreateSymbol
attribute
XmlElement dataArea = (XmlElement)targetDoc.SelectSingleNode("TreeItem/TcModuleInstance/Module/
DataAreas/DataArea[ContextId='0' and Name='Input']");
XmlNode dataAreaNo = dataArea.SelectSingleNode("AreaNo");
XmlAttribute createSymbol = dataAreaNoNode.Attributes["CreateSymbols"];

// Fourth step: Set CreateSymbol attribute to true if it exists. If not, create attribute and set
its value
if (createSymbol != null)
string oldValue = createSymbol.Value;
else
{
createSymbol = targetDoc.CreateAttribute("CreateSymbols");
dataAreaNo.Attributes.Append(createSymbol);
}
createSymbol.Value = XmlConvert.ToString(activateCS);

// Fifth step: Write prepared XML to configuration via ConsumeXml()
string targetXml = targetDoc.OuterXml;
tempController.ConsumeXml(targetXml);
```

代码片段 (Powershell)：

```
$tempControllerXml = [Xml]$tempController.ProduceXml()
$sourceDataAreas = $tempControllerXml.TreeItem.TcModuleInstance.Module.DataAreas

[System.XML.XmlDocument] $targetDoc = New-Object System.XML.XmlDocument
[System.XML.XmlElement] $treeItem = $targetDoc.CreateElement("TreeItem")
[System.XML.XmlElement] $moduleInstance = $targetDoc.CreateElement("TcModuleInstance")
[System.XML.XmlElement] $module = $targetDoc.CreateElement("Module")
[System.XML.XmlElement] $dataAreas = $targetDoc.ImportNode($sourceDataAreas, $true)
$module.AppendChild($dataAreas)
$moduleInstance.AppendChild($module)
$treeItem.AppendChild($moduleInstance)
$targetDoc.AppendChild($treeItem)

$destModule = $targetDoc.TreeItem.TcModuleInstance.Module
[System.XML.XmlElement] $dataArea = $destModule.SelectSingleNode("DataAreas/DataArea[ContextId='0'
and Name='Input']")
$dataAreaNo = $dataArea.SelectSingleNode("AreaNo")
$dataAreaNo.CreateSymbols = "true"

// Fifth step: Write prepared XML to configuration via ConsumeXml()
$targetXml = $targetDoc.OuterXml
$tempController.ConsumeXml($targetXml)
```

设置上下文（任务）

每个 TcCOM 模块实例都需要在特定上下文（任务）中运行。这可通过 ITcModuleInstance2::SetModuleContext() 方法实现。该方法等待两个参数：ContextId 和 TaskObjectId。均等同于 TwinCAT XAE 中的相应参数：



请注意，TaskObjectId 在 TwinCAT XAE 中以十六进制显示。

代码片段 (C#)：

```
ITcModuleInstance2 tempControllerMi = (ITcModuleInstance2) tempController;
tempControllerMi.SetModuleContext(0, 33619984);
```

可以通过相应任务的 XML 描述确定 TaskObjectId，例如：

代码片段 (C#)：

```
ITcSmTreeItem someTask = systemManager.LookupTreeItem("TIRT^SomeTask");
string someTaskXml = someTask.ProduceXml();
XmlDocument someTaskDoc = new XmlDocument();
someTaskDoc.LoadXml(someTaskXml);
XmlNode taskObjectIdNode = someTaskDoc.SelectSingleNode("TreeItem/ObjectId");
string taskObjectIdStr = taskObjectId.InnerText;
uint taskObjectId = uint.Parse(taskObjectIdStr, NumberStyles.HexNumber);
```

链接变量

通过使用常规自动化接口机制，例如 ITcSysManager::LinkVariables()，可以将 TcCOM 模块实例的变量链接到 PLC/IO 或其它 TcCOM 模块。

4.3.7 C++

4.3.7.1 创建和处理 C++ 项目和模块

本章将详细说明如何创建、访问和处理 TwinCAT C++ 项目。以下列表显示本文中的所有章节：

- C++ 项目常规信息
- 创建新的 C++ 项目
- 在 C++ 项目中创建新模块
- 打开现有的 C++ 项目
- 创建模块实例
- 调用 TMC 代码生成器
- 调用发布模块命令
- 设置 C++ 项目属性
- 构建项目

C++ 项目常规信息

C++ 项目通过由“TwinCAT C++ Project Wizard” (TwinCAT C++ 项目向导) 使用的所谓项目模板指定。在一个项目中，可以通过由“TwinCAT Class Wizard” (TwinCAT 类向导) 使用的模块模板定义多个模块。

TwinCAT 定义的模板参见章节 C++ /向导。

客户可以自行定义模板，请参见 C++ /向导的相应小节。

创建 C++ 项目

要通过自动化接口创建新的 C++ 项目，需要导航至 C++ 节点，然后以相应的模板文件作为参数执行 CreateChild() 方法。

代码片段 (C#)：

```
ITcSmTreeItem cpp = systemManager.LookupTreeItem("TIXC");
ITcSmTreeItem cppProject = cpp.CreateChild("NewCppProject", 0, "", pathToTemplateFile);
```

代码片段 (Powershell)：

```
$cpp = $systemManager.LookupTreeItem("TIXC")
$newProject = $cpp.CreateChild("NewCppProject", 0, "", $pathToTemplateFile)
```

要对驱动程序项目进行实例化，请将“TcDriverWizard”用作 pathToTemplateFile.。

在 C++ 项目中创建新模块

在 C++ 项目中，通常使用 TwinCAT 模块向导来使向导程序通过模板创建模块。

代码片段 (C#)：

```
ITcSmTreeItem cppModule = cppProject.CreateChild("NewModule", 1, "", pathToTemplateFile);
```

代码片段 (Powershell)：

```
$cppModule = $cppProject.CreateChild("NewModule", 0, "", $pathToTemplateFile);
```

例如，要对 Cyclic IO 模块项目进行实例化，请将“TcModuleCyclicCallerWizard”用作 pathToTemplateFile.。

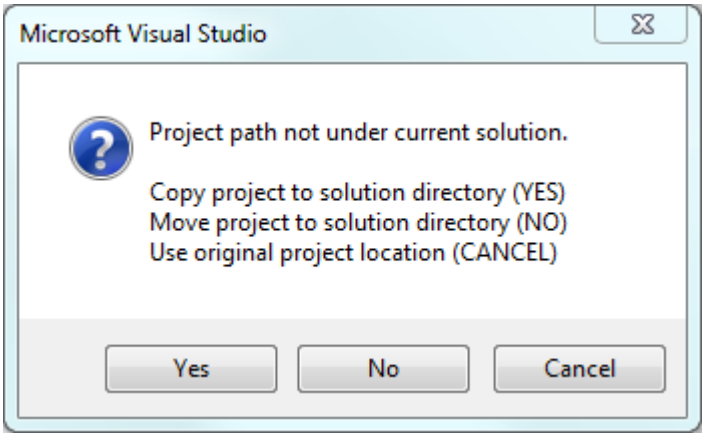
打开现有的 C++ 项目

要通过自动化接口打开现有的 C++ 项目，需要导航至 C++ 节点，然后以相应的 C++ 项目文件路径作为参数执行 CreateChild() 方法。

可以使用三个不同的值作为子类型：

- 0：将项目复制到解决方案目录
- 1：将项目移动到解决方案目录
- 2：使用原始项目位置（指定“”为 NameOfProject 参数）

基本上，这些值表示 TwinCAT XAE 中以下消息框中的功能（是、否、取消）：



需要使用要添加的 C++ 项目的路径（至其 vcxproj 文件）来代替模板文件。或者，也可以使用 C++ 项目存档（tczip 文件）。

代码片段 (C#)：

```
ITcSmTreeItem cpp = systemManager.LookupTreeItem("TIXC");
ITcSmTreeItem newProject = cpp.CreateChild("NameOfProject", 1, "", pathToProjectOrTczipFile);
```

代码片段 (Powershell)：

```
$cpp = $systemManager.LookupTreeItem("TIXC")
$newProject = $cpp.CreateChild("NameOfProject", 1, "", $pathToProjectOrTczipFile)
```

请注意，C++ 项目无法重命名，因此需要指定原始项目名。(cmp.)

创建模块实例

可以在系统 -> TcCOM 模块节点处创建 TcCOM 模块。请参见此处的文档 [▶ 81]。

也可以对 C++ 项目节点采用相同的步骤来在相应位置（本页顶部代码处的 $newProject）添加 TcCOM 实例。

调用 TMC 代码生成器

对 C++ 项目的 TMC 文件进行修改后，可以调用 TMC 代码生成器来生成 C++ 代码。

代码片段 (C#)：

```
string startTmcCodeGenerator = @"<?xml version=""1.0"" encoding=""UTF-16""?>
<TreeItem>
<CppProjectDef>
<Methods>
<StartTmcCodeGenerator>
<Active>true</Active>
</StartTmcCodeGenerator>
</Methods>
</CppProjectDef>
</TreeItem>";
cppProject.ConsumeXml(startTmcCodeGenerator);
```

代码片段 (Powershell)：

```
$startTmcCodeGenerator = @"<?xml version=""1.0"" encoding=""UTF-16""?>
<TreeItem>
<CppProjectDef>
<Methods>
<StartTmcCodeGenerator>
<Active>true</Active>
</StartTmcCodeGenerator>
</Methods>
</CppProjectDef>
</TreeItem>"
$cppProject.ConsumeXml($startTmcCodeGenerator)
```

调用发布模块命令

发布包括构建用于所有平台的项目。将提供用于导出的已编译模块，如 C++ 模块处理部分所述。

代码片段 (C#)：

```
string publishModules = @"<?xml version=""1.0"" encoding=""UTF-16""?>
<TreeItem>
<CppProjectDef>
<Methods>
<PublishModules>
<Active>true</Active>
</PublishModules>
</Methods>
</CppProjectDef>
</TreeItem>";
cppProject.ConsumeXml(publishModules);
```

代码片段 (Powershell)：

```
$publishModules = @"<?xml version=""1.0"" encoding=""UTF-16""?>
<TreeItem>
<CppProjectDef>
<Methods>
<PublishModules>
<Active>true</Active>
</PublishModules>
</Methods>
```

```
</CppProjectDef>
</TreeItem>"
$cppProject.ConsumeXml($publishModules)
```

设置 C++ 项目属性

C++ 项目为构建和部署过程提供不同的选项。
可通过自动化接口对其进行设置。

代码片段 (C#)：

```
string projProps = @"<?xml version=""1.0"" encoding=""UTF-16""?>
<TreeItem>
<CppProjectDef>
<BootProjectEncryption>Target</BootProjectEncryption>
<TargetArchiveSettings>
<SaveProjectSources>false</SaveProjectSources>
</TargetArchiveSettings>
<FileArchiveSettings>
<SaveProjectSources>false</SaveProjectSources>
</FileArchiveSettings>
</CppProjectDef>
</TreeItem>";
cppProject.ConsumeXml(projProps);
```

代码片段 (Powershell)：

```
$projProps = @"<?xml version=""1.0"" encoding=""UTF-16""?>
<TreeItem>
<CppProjectDef>
<BootProjectEncryption>Target</BootProjectEncryption>
<TargetArchiveSettings>
<SaveProjectSources>false</SaveProjectSources>
</TargetArchiveSettings>
<FileArchiveSettings>
<SaveProjectSources>false</SaveProjectSources>
</FileArchiveSettings>
</CppProjectDef>
</TreeItem>"
$cppProject.ConsumeXml($projProps)
```

对于 BootProjectEncryption，“None”和“Target”值有效。
另两个设置为“false”和“true”值。

构建项目

要构建项目或解决方案，可以使用此处 [▶ 27]所述的 Visual Studio API 的相应类和方法。

4.3.8 测量

4.3.8.1 创建和处理 TwinCAT Measurement 项目

TwinCAT 自动化接口提供创建和访问 TwinCAT Measurement 项目的方法和属性。以下章节介绍如何使用该类 TwinCAT 项目解决一些基本任务，并包括有关以下主题的信息：

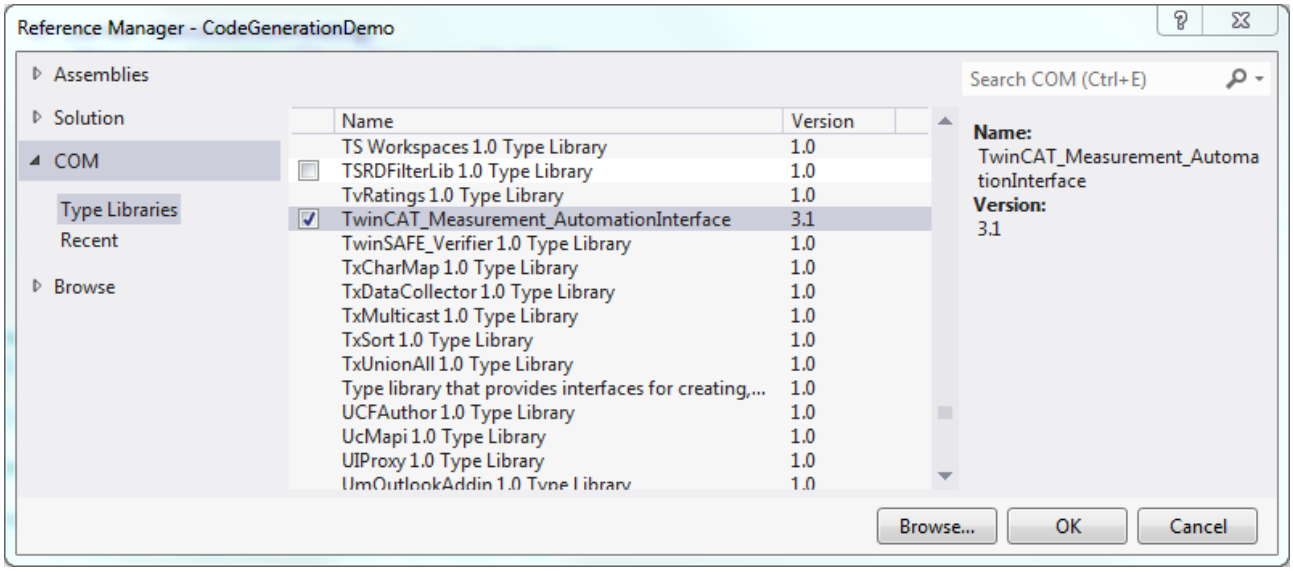
- 要求
- 创建 TwinCAT Measurement 项目
- 创建 TwinCAT Scope 组态
- 创建、访问和处理图表
- 创建、访问和处理轴
- 创建、访问和处理通道
- 开始和停止记录
- 有关 TwinCAT Measurement 的详细信息，请访问信息系统中的相应网页。

要求

从 TwinCAT 3.1 Build 4013 开始，可以通过自动化接口集成 TwinCAT Measurement 项目。

对以下 COM 接口的引用是使用 Scope AI 功能的一个必要条件：

- TwinCAT Measurement 自动化接口 (注册为 COM 库)
- TwinCAT Scope2 Communications (.NET 程序集：C:\Windows\Microsoft.NET\assembly\GAC_MSIL\TwinCAT.Scope2.Communications\v4.0_3.1.3121.0__180016cd49e5e8c3\)



创建 TwinCAT Measurement 项目

TwinCAT Measurement 是一个全局“容器”，可以承载一个或多个测量项目，例如 TwinCAT Scope 组态。与常规 TwinCAT 组态相似，每个项目首先通过模板文件进行描述。在向解决方案中添加新项目时会使用此模板文件，可以通过从 Visual Studio DTE 对象调用 AddFromTemplate() 方法来实现。请注意，在添加常规 TwinCAT 项目时，该步骤相同。以下代码片段假定已获得 DTE 实例并已创建 Visual Studio 解决方案，请参见有关访问 TwinCAT 组态的章节。对 DTE 实例的引用存储在对象“dte”中。

代码片段 (C#)：

```
EnvDTE.Project scopeProject = dte.Solution.AddFromTemplate(template, destination, name);
```

[template]：默认模板文件存储在 C:\TwinCAT\Functions\TE130X-Scope-View\Templates\Projects\ 下，文件类型为“tcmproj”。

[destination]：新组态在硬盘上的存储路径。

[name]：新组态的名称，如 TwinCAT XAE 中所示。

创建 TwinCAT Scope 组态

TwinCAT Scope 项目代表记录的组态。这表示插入此项目的所有元素都将采用相同的记录设置。如前所述，在使用 AddFromTemplate() 方法添加项目时，可以通过自动化接口指定相应“TwinCAT Scope 项目”模板来添加 Scope 项目。

创建、访问和处理图表

一个 Scope 组态中可同时存在多个图表。要将图表添加到现有 Scope 项目中，只需使用 IMeasurementScope 接口中的 CreateChild() 方法。以下代码片段假定已创建 TwinCAT Measurement 项目且对此项目的引用存储在对象“scopeProject”中。

代码片段 (C#)：

```
((IMeasurementScope)scopeProject.ProjectItems.Item(1).Object).ShowControl();
EnvDTE.ProjectItem newChart;
((IMeasurementScope)scopeProject.ProjectItems.Item(1).Object).CreateChild(out newChart);
IMeasurementScope newChartObj = (IMeasurementScope)newChart.Object;
newChartObj.ChangeName("NC Chart");
```

现在，对象“newChartObj”存储了对新添加图表的引用。请记住，此时仍需要原始对象“newChart”，以备以后使用，例如用于设置图表属性。

在 TwinCAT XAE 中，图表属性通过 Visual Studio 属性窗口进行设置。对象“newChart”允许通过其“属性”集合访问这些属性。以下代码片段对图表的所有属性进行迭代，并将属性“StackedYAxis”设置为“true”。

代码片段 (C#)：

```
foreach (EnvDTE.Property prop in newChart.Properties)
{
If (prop.Name = "StackedYAxis")
prop.Value = true;
}
```

创建、访问和处理轴

以下代码片段显示如何添加轴。

代码片段 (C#)：

```
EnvDTE.ProjectItem newAxis;
newChartObj.CreateChild(out newAxis);
IMeasurementScope newAxisObj = (IMeasurementScope)newAxis.Object;
newAxisObj.ChangeName(“Axis 1”);
```

创建、访问和处理通道

Scope 通道记录与运行时符号的连接，例如 PLC 变量。要通过自动化接口添加通道，可以使用 IMeasurementScope 接口的 CreateChild() 方法。

代码片段 (C#)：

```
EnvDTE.ProjectItem newChannel;
newAxisObj.CreateChild(out newChannel);
IMeasurementScope newChannelObj = (IMeasurementScope)newChannel.Object;
newChannelObj.ChangeName("Signals.Rectangle");
```

现在，对象“newChannelObj”存储了对新添加通道的引用。请记住，此时仍需要原始对象“newChannel”，以备以后使用，例如用于设置通道属性。

在 TwinCAT XAE 中，通道属性通过 Visual Studio 属性窗口进行设置。对象“newChannel”允许通过其“属性”集合访问这些属性。以下代码片段对通道的所有属性进行迭代，并设置多个属性。

代码片段 (C#)：

```
foreach (EnvDTE.Property prop in newChannel.Properties)
{
switch (prop.Name)
{
case "TargetPorts":
prop.Value = "851";
break;

case "Symbolbased":
prop.Value = true;
break;

case "SymbolDataType":
prop.Value = TwinCAT.Scope2.Communications.Scope2DataType.BIT;
break;

case "SymbolName":
prop.Value = "MAIN.bSignal";
break;

case "SampleState":
prop.Value = 1;
break;

case "LineWidth":
prop.Value = 3;
break;
}
}
```

如您所见，Scope2DataType 枚举定义组态通道时支持的多个数据类型。创建此文档时，此枚举定义如下：

代码片段 (C#)：

```
public enum Scope2DataType
{
VOID = 0,
BIT = 1,
INT16 = 2,
INT32 = 3,
INT64 = 4,
INT8 = 5,
REAL32 = 6,
REAL64 = 7,
UINT16 = 8,
UINT32 = 9,
UINT64 = 10,
UINT8 = 11,
BIT_ARRAY_8 = 12,
}
```

请使用 DLL 资源管理器（例如 Visual Studio）获得最新列表。

开始和停止记录

要开始/停止记录经过组态的 Scope，可以使用 IMeasurementScope 接口中的相应方法：

代码片段 (C#)：

```
((IMeasurementScope)scopeProject.ProjectItems.Item(1).Object).StartRecord()
((IMeasurementScope)scopeProject.ProjectItems.Item(1).Object).StopRecord();
```

4.3.9 Motion

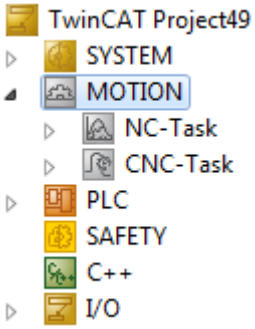
4.3.9.1 创建和处理 Motion 项目

本章将详细说明如何创建和处理 Motion 项目。以下列表显示本文中的所有章节：

- TwinCAT 3 Motion 常规信息
- 创建 NC 任务
- 创建轴
- 对轴进行参数化

TwinCAT Motion 常规信息

TwinCAT 3 Motion 由以下三部分组成：NC-I、NC-PTP 和 CNC。因此它是用于控制和调整轴或同步轴组的功能组程序集。有关 TwinCAT Motion 的详细信息，请参见 TwinCAT 3 Motion 文档。



创建 NC 任务

创建 TwinCAT 3 Motion 项目时，首先要创建所谓的 NC 任务。在 TwinCAT 自动化接口中，只需调用 ITcSmTreeItem [▶ 103]::CreateChild() [▶ 136] 方法并使用 SubType 参数 1 即可创建此任务，如以下代码片段所示。

代码片段 (C#)：

```
ITcSmTreeItem ncConfig = systemManager.LookupTreeItem("TINC");
ncConfig.CreateChild("NC-Task", 1);
```

代码片段 (Powershell)：

```
$ncConfig = $systemManager.LookupTreeItem("TINC")
$ncConfig.CreateChild("NC-Task", 1)
```

创建轴

由于 NC 任务默认不包含任何轴，因此需要再次使用 CreateChild() 方法进行添加 - 这次是在 NC 任务下对轴节点的引用上。

代码片段 (C#)：

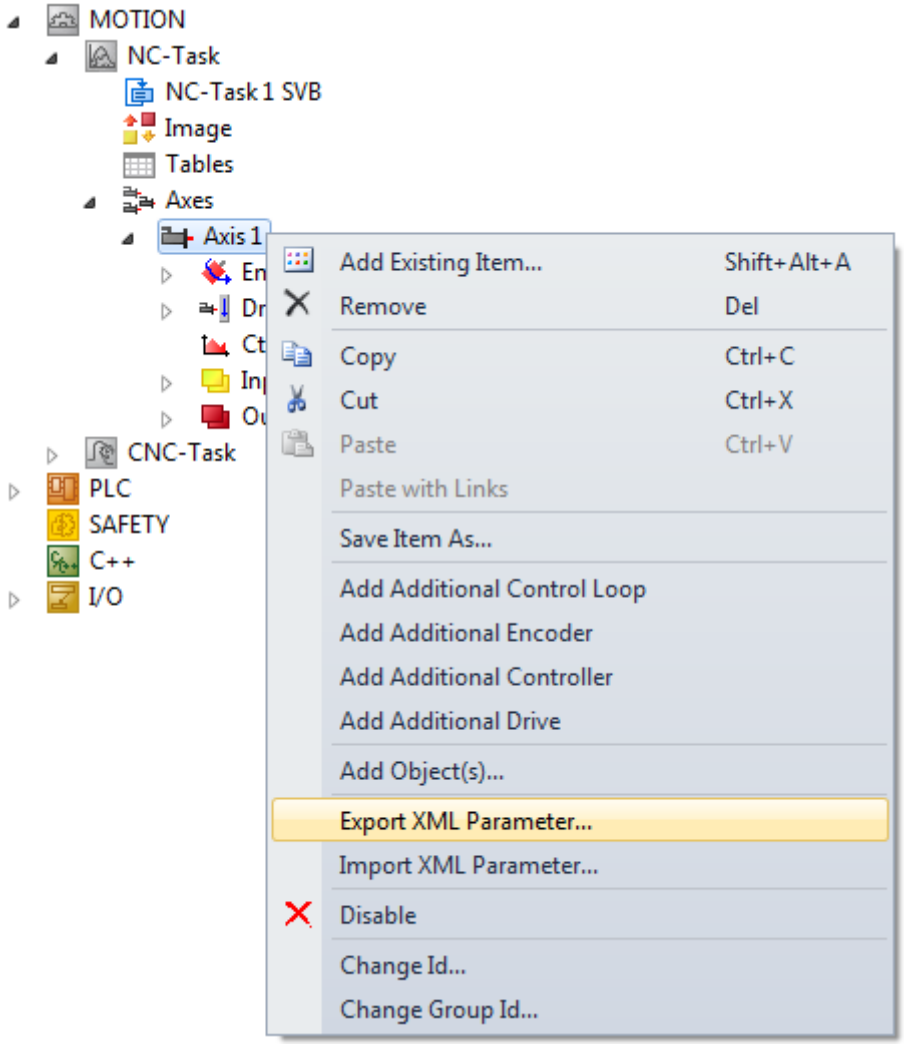
```
ITcSmTreeItem axes = systemManager.LookupTreeItem("TINC^NC-Task^Axes");
axes.CreateChild("Axis 1", 1);
```

代码片段 (Powershell)：

```
$axes = $systemManager.LookupTreeItem("TINC^NC-Task^Axes")
$axes.CreateChild("Axis 1", 1)
```

对轴进行参数化

轴可以通过其 XML 描述 [▶ 22]进行参数化。要获得 XML 描述示例，可以在 TwinCAT XAE 中手动添加轴并使用上下文菜单中的相应条目，或通过自动化接口添加轴，并使用 ITcSmTreeItem [▶ 103]::ProduceXml() [▶ 134] 方法来获得。



代码片段 (C#)：

```
ITcSmTreeItem axis = systemManager.LookupTreeItem("TINC^NC-Task^Axes^Axis 1");
string xmlDescription = axis.ProduceXml();
```

代码片段 (Powershell)：

```
$axis = $systemManager.LookupTreeItem("TINC^NC-Task^Axes^Axis 1")
$xmlDescription = $axis.ProduceXml()
```

可以根据需要修改此 XML 描述，然后通过 ITcSmTreeItem [▶ 103]::ConsumeXml() [▶ 135] 方法再次将其导入，以对轴进行参数化。

代码片段 (C#)：

```
ITcSmTreeItem axis = systemManager.LookupTreeItem("TINC^NC-Task^Axes^Axis 1");
axis.ConsumeXml(xmlDescription);
```

代码片段 (Powershell)：

```
$axis = $systemManager.LookupTreeItem("TINC^NC-Task^Axes^Axis 1")
$axis.ConsumeXml($xmlDescription)
```

4.3.10 安全

4.3.10.1 创建和处理 Safety 项目

本章将详细说明如何创建、访问和处理 Safety 项目。以下列表显示本文中的所有章节：

- Safety 项目常规信息
- 打开现有的 Safety 项目

Safety 项目常规信息

- TwinCAT 自动化接口允许将现有的 TwinCAT Safety 项目导入 TwinCAT 组态。为此，用户可以将相应的 *.splcproj 文件或容器格式 *.tfzip 作为源模板使用。

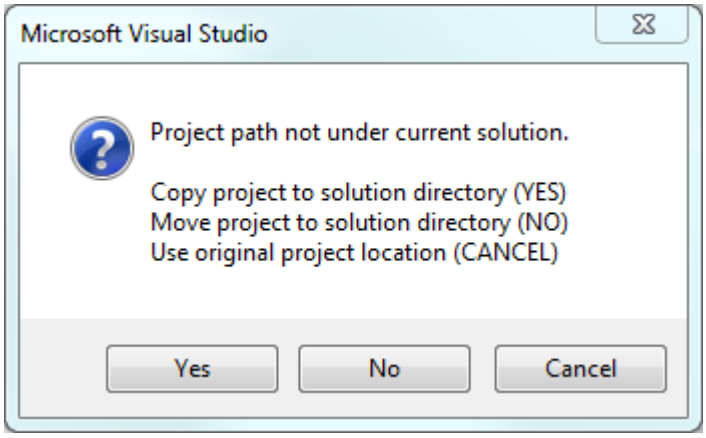
打开现有的 Safety 项目

要通过自动化接口打开现有的 Safety 项目，需要导航至 Safety 节点，然后以相应的现有 Safety 项目文件路径作为参数执行 CreateChild() 方法。

可以使用三个不同的值作为子类型：

- 0：将项目复制到解决方案目录
- 1：将项目移动到解决方案目录
- 2：使用原始项目位置（使用时，请将 "" 用作项目名参数）

基本上，这些值表示 TwinCAT XAE 中以下消息框中的功能（是、否、取消）：



可以使用需要添加的 Safety 项目路径（至其 *.splcproj 文件），也可以使用 Safety 项目存档 (*.tfzip)。<b />

代码片段 (C#)：

```
ITcSmTreeItem safety = systemManager.LookupTreeItem("TISC");
ITcSmTreeItem newProject = safety.CreateChild("NameOfProject", 0, null, pathToProjectOrTfzipFile);
```

代码片段 (Powershell)：

```
$safety = $systemManager.LookupTreeItem("TISC")
$newProject = $safety.CreateChild("NameOfProject", 0, "", pathToProjectOrTfzipFile)
```

5 API

5.1 引用

本章包含 TwinCAT 自动化接口所有类和方法的文档。所提供的接口可以分为不同的“层级”，其中较高层级接口代表主接口，因此与自动化接口进行基本交互。请注意，这种区别仅来自逻辑观点，以便更好地理解哪些接口是最重要的，哪些接口是次要的。

1 级接口

如介绍 [▶ 9]中所述，在 TwinCAT 组态中，只有两个主要接口用于导航和引用树项。

主类	描述	可用版本
ITcSysManager [▶ 96]	用于创建 TwinCAT 组态并参数化的基础类	TwinCAT 2.11
ITcSmTreeItem [▶ 103]	表示 TwinCAT 组态中的树项	TwinCAT 2.11

2 级接口

这些接口被视为“helper 类”，通常与 1 级类一起使用，例如将 ITcSmTreeItem 对象强制转换为更具体的树项类型，例如 POU (ITcPlcPou) 或链接任务 (ITcTaskReference)。

帮助程序类	描述	可用版本
ITcPlcLibraryManager [▶ 146]	定义 PLC 库管理的方法和属性	TwinCAT 3.1
ITcPlcPou [▶ 141]	定义处理 PLC POU 的方法和属性	TwinCAT 3.1
ITcPlcDeclaration [▶ 142]	定义 PLC POU 声明区域的读/写方法	TwinCAT 3.1
ITcPlcImplementation [▶ 142]	定义 PLC POU 实现区域的读/写方法	TwinCAT 3.1
ITcPlcProject [▶ 140]	定义有关 PLC 项目的方法和属性，例如将项目设置为 boot 项目	TwinCAT 3.1
ITcPlcIECProject [▶ 143]	定义在 PLCopen XML 中导入/导出 PLC 项目并将其作为 PLC 库安装所需的方法	TwinCAT 3.1
ITcPlcTaskReference [▶ 156]	定义将 PLC 项目链接到任务的方法和属性	TwinCAT 3.1
ITcPlcLibrary [▶ 153]	表示单个 PLC 库的 Helper 类	TwinCAT 3.1
ITcPlcLibraries [▶ 153]	表示 PLC 库集合的 Helper 类	TwinCAT 3.1
ITCPlcReferences [▶ 153]	表示 ITcPlcLibRef 对象集合的 Helper 类 (即 PLC 项目中的引用)	TwinCAT 3.1
ITcPlcLibRef [▶ 154]	表示 ITcPlcLibrary 和 ITcPlcPlaceholderRef 对象基础类的 Helper 类	TwinCAT 3.1
ITcPlcPlaceholderRef [▶ 154]	表示单个 PLC 占位符的 Helper 类	TwinCAT 3.1
ITcPlcLibRepository [▶ 155]	表示单个 PLC 存储库的 Helper 类	TwinCAT 3.1
ITcPlcLibRepositories [▶ 155]	表示 PLC 库集合存储库的 Helper 类	TwinCAT 3.1

5.2 ITcSysManager

5.2.1 ITcSysManager

ITcSysManager 是 TwinCAT 自动化接口的主接口。此接口允许组态 TwinCAT 3 XAE 的基本操作并包含多种方法。多年来，ITcSysManager 接口通过更多功能进行了扩展，为客户提供更好的方法来使用所有自动化接口功能。但是，由于 COM 对象模型中的限制，需要将这些功能作为单独的接口添加到自动化接口中。因此，每次添加一个新功能集时，这些功能都会被组合到一个名为 ITcSysManagerX 的新接口中，其中 X 是一个数字，会在每次添加一个新接口时递增。下表说明哪些方法包含在 ITcSysManager 接口中及哪些已添加到各个新的“功能集”接口。

方法

ITcSysManager 方法	描述	可用版本
NewConfiguration [▶ 97]	生成新组态	TwinCAT 2.11
OpenConfiguration [▶ 97]	加载先前创建的组态文件 (WSM 文件)	TwinCAT 2.11
SaveConfiguration [▶ 98]	将组态保存在指定名称或当前名称的文件中	TwinCAT 2.11
ActivateConfiguration [▶ 98]	激活组态 (与“保存至注册表”相同)	TwinCAT 2.11
LookupTreeItem [▶ 101]	通过名称查找组态项目 (树项) 并返回 ITcSmTreeItem [▶ 103] 接口	TwinCAT 2.11
StartRestartTwinCAT [▶ 99]	启动或重新启动 TwinCAT 系统	TwinCAT 2.11
IsTwinCATStarted [▶ 98]	评估 TwinCAT 系统是否正在运行	TwinCAT 2.11
LinkVariables [▶ 99]	链接通过名称指定的两个变量	TwinCAT 2.11
UnlinkVariables [▶ 99]	删除通过名称指定的两个变量之间的链接或来自一个变量的所有链接。	TwinCAT 2.11

ITcSysManager2 方法	描述	可用版本
SetTargetNetId [▶ 100]	设置当前打开的 TwinCAT 组态的目标 NetId。	TwinCAT 2.11
GetTargetNetId [▶ 100]	获得当前打开的 TwinCAT 组态的目标 NetId。	TwinCAT 2.11
GetLastErrorMessages [▶ 101]	获得 TwinCAT 子系统中发生的最后一条错误消息。	TwinCAT 2.11

ITcSysManager3 方法	描述	可用版本
LookupTreeItemById [▶ 102]	通过指定项目 id 查找组态树项。	TwinCAT 2.11
ProduceMappingInfo [▶ 102]	生成实际组态映射的 Xml 描述。	TwinCAT 3.1
ClearMappingInfo	清除映射信息。	TwinCAT 2.11

注释

ITcSysManager 接口包含两种用于在 TwinCAT XAE 中导航的方法：ITcSysManager::LookupTreeItem [▶ 101] 和 ITcSysManager3::LookupTreeItemById [▶ 102]。有关浏览 TwinCAT XAE 的详细说明，请参见 TreeItem 浏览模式 [▶ 20]章节。

警告：ITcSysManager::NewConfiguration [▶ 97]、ITcSysManager::OpenConfiguration [▶ 97] 和 ITcsSysManager::SaveConfiguration [▶ 98] 三种方法仅在兼容模式 [▶ 17]下可用。在标准模式下对其进行调用将抛出 E_NOTSUPPORTED 异常。

ITcSysmanager 和 ITcSmTreeItem [▶ 103] 接口允许完全访问 TwinCAT 组态。在本文档的“如何操作...”章节中，提供了一个长示例列表 (但并不完整)，说明如何自动操作 TwinCAT 组态。

版本信息

要求

所需 TwinCAT 版本
TwinCAT 2.11 及以上版本支持此接口

5.2.2 ITcSysManager::NewConfiguration

NewConfiguration() 方法将生成一个新的 TwinCAT 组态文件。

```
HRESULT NewConfiguration();
```

返回值

S_OK	功能返回一个值。
E_ACCESSDENIED	实际文档在系统管理器实例中锁定。这是指至少打开一个对系统管理器对象的引用或一个树项。
E_FAIL	功能失败。

注释

警告：ITcSysManager::NewConfiguration [▶ 97]、ITcSysManager::OpenConfiguration [▶ 97] 和 ITcsSysManager::SaveConfiguration [▶ 98] 三种方法仅在兼容模式 [▶ 17]下可用。在标准模式下对其进行调用将抛出 E_NOTSUPPORTED 异常。

5.2.3 ITcSysManager::OpenConfiguration

OpenConfiguration() 方法将加载先前创建的 TwinCAT 组态文件。

```
HRESULT OpenConfiguration(BSTRbstrFile);
```

Parameters (参数)

bstrFile	[in, defaultvalue(L"")] 包含应加载组态文件的文件路径或者空字符串 (如果应生成新组态)。当前运行的目标设备组态也可以通过“CURRENTCONFIG”读取。

返回值

S_OK	功能返回一个值。
E_ACCESSDENIED	实际文档在系统管理器实例中锁定。这是指至少打开一个对系统管理器对象的引用或一个树项。
E_INVALIDARG	此路径未指向有效组态文件。

注释

警告：ITcSysManager::NewConfiguration [▶ 97]、ITcSysManager::OpenConfiguration [▶ 97] 和 ITcsSysManager::SaveConfiguration [▶ 98] 三种方法仅在兼容模式 [▶ 17]下可用。在标准模式下对其进行调用将抛出 E_NOTSUPPORTED 异常。

还请参阅有关此

ITcSysManager [▶ 96]

5.2.4 ITcSysManager::SaveConfiguration

SaveConfiguration() 方法将 TwinCAT 组态保存在具有指定名称的文件中。

```
HRESULT SaveConfiguration(BSTRbstrFile);
```

Parameters (参数)

bstrFile	[in, defaultvalue(L"")] 包含组态文件应保存的路径。当 *bstrFile* 是空字符串时，则使用实际文件名。

返回值

S_OK	功能返回一个值。
E_INVALIDARG	文件路径无效。

注释

警告：ITcSysManager::NewConfiguration [▶ 97]、ITcSysManager::OpenConfiguration [▶ 97] 和 ITcsSysManager::SaveConfiguration [▶ 98] 三种方法仅在兼容模式 [▶ 17]下可用。在标准模式下对其进行调用将抛出 E_NOTSUPPORTED 异常。

还请参阅有关此

ITcSysManager [▶ 96]

5.2.5 ITcSysManager::ActivateConfiguration

ActivateConfiguration() 方法将激活 TwinCAT 组态（与“保存至注册表”相同）。之后，必须执行以下 TwinCAT 系统启动或重新启动，以实际激活组态。

```
HRESULT ActivateConfiguration();
```

返回值

S_OK	功能返回一个值。
E_FAIL	功能失败。

还请参阅有关此

ITcSysManager [▶ 96]

5.2.6 ITcSysManager::IsTwinCATStarted

IsTwinCATStarted() 方法将评估 TwinCAT 系统是否正在运行。

```
HRESULT IsTwinCATStarted(VARIANT_BOOL*pStarted);
```

Parameters (参数)

pStarted	[out, retval] 指向包含结果的布尔值的存储位置。

返回值

S_OK	功能返回一个值。

还请参阅有关此

ITcSysManager [▶ 96]

5.2.7 ITcSysManager::StartRestartTwinCAT

StartRestartTwinCAT() 方法启动或重新启动 TwinCAT 系统。如果 TwinCAT 已启动，则功能执行重新启动，如果 TwinCAT 停止，则执行启动。

```
HRESULT StartRestartTwinCAT();
```

返回值

要求

S_OK	功能返回一个值。
E_FAIL	TwinCAT 无法启动。

还请参阅有关此

ITcSysManager [▶ 96]

5.2.8 ITcSysManager::LinkVariables

LinkVariables() 方法将链接通过名称指定的两个变量。由树路径名表示的两个变量将被链接。路径名的语法必须与 ITcSysManager::LookupTreeItem [▶ 101] 中的描述一致。可以使用相同的快捷方式 [▶ 101]。

```
HRESULT LinkVariables(BSTRbstrV1, BSTRbstrV2, longoffs1, longoffs2, longsize);
```

Parameters (参数)

bstrV1	[in] 第一个变量的路径名。需要完整的路径名，各分支之间必须用扬抑符“^”或制表符分隔。
bstrV2	[in] 第二个变量的路径名。需要完整的路径名，各分支之间必须用扬抑符“^”或制表符分隔。
offs1	[in, defaultvalue(0)] 第一个变量的位偏移（在两个变量大小不同或不应链接整个变量时使用）。
offs2	[in, defaultvalue(0)] 第二个变量的位偏移。
Größe	[in, defaultvalue(0)] 要链接的位数。如果 *size* 为 0，则使用变量 1 和变量 2 中的最小变量值。

返回值

S_OK	函数成功返回。
TSM_E_ITEMNOTFOUND (0x98510001)	一个或两个路径名不限定现有树项。
TSM_E_INVALIDITEMTYPE (0x98510002)	一个或两个树项不是变量。
TSM_E_MISMATCHINGITEMS (0x98510004)	两个变量无法进行链接。可能是您试图将一个任务的输出与另一个任务的输出进行链接，或者将一个任务的输出与设备的输入或同一用户的变量进行链接。
E_INVALIDARG	*offs1*、*offs2* 和/或 *size* 的值与变量不匹配。

还请参阅有关此

ITcSysManager [▶ 96]

5.2.9 ITcSysManager::UnlinkVariables

UnlinkVariables() 方法将解除通过其名称指定的两个变量之间的链接，或清除来自第一个变量的所有链接（如果第二个变量的名称 *bstrV2* 为空）。将解除由其树路径名指定的两个变量之间的链接。路径名的语法必须与 ITcSysManager::LookupTreeItem [▶ 101] 中的描述一致。可以使用相同的快捷方式 [▶ 101]。

```
HRESULT UnlinkVariables(BSTRbstrV1, BSTRbstrV2);
```

Parameters (参数)

bstrV1	[in] 第一个变量的路径名。需要完整的路径名，各分支之间必须用扬抑符“^”或制表符分隔。
bstrV2	[in, defaultvalue(L"")] 第二个变量的路径名。如果需要设置完整路径名，各分支之间必须用扬抑符“^”或制表符分隔。

返回值

S_OK	函数成功返回。
S_FALSE	两个变量之间未进行链接。
TSM_E_ITEMNOTFOUND (0x98510001)	一个或两个路径名不限定现有树项。
TSM_E_INVALIDITEMTYPE (0x98510002)	一个或两个树项不是变量。
TSM_E_CORRUPTEDLINK (0x98510005)	无法解除两个变量之间的链接。

注释

如果 *bstrV2* 是空字符串，此功能将清除通过 *bstrV1* 指定的所有变量链接。如果 *bstrV2* 不为空，将只删除两个变量之间的现有链接。

还请参阅有关此

ITcSysManager [▶ 96]

5.2.10 ITcSysManager2::GetTargetNetId

GetTargetNetId() 方法返回当前 TwinCAT 系统的 NetId。

```
HRESULT GetTargetNetId();
```

Parameters (参数)

无

返回值

STRING	返回目标的 NetId

还请参阅有关此

ITcSysManager [▶ 96]

5.2.11 ITcSysManager2::SetTargetNetId

SetTargetNetId() 方法设置当前 TwinCAT 系统的 NetId。

```
HRESULT SetTargetNetId(STRING netId);
```

Parameters (参数)

netId	表示目标的 NetId。

返回值

S_OK	函数成功返回。

还请参阅有关此

ITcSysManager [▶ 96]

5.2.12 ITcSysManager2::GetLastErrorMessages

GetLastErrorMessages() 方法返回最后一条错误消息。

```
HRESULT GetLastErrorMessages();
```

Parameters (参数)

无

返回值

STRING	返回最后一条错误消息。

还请参阅有关此

ITcSysManager [▶ 96]

5.2.13 ITcSysManager::LookupTreeItem

LookupTreeItem() 方法返回由其完整路径名指定的树项的 *ITcTreeItem* 指针。

```
HRESULT LookupTreeItem(BSTRbstrItem, ITcSmTreeItem**pipItem);
```

Parameters (参数)

bstrItem	[in] 要查找树项的路径名。需要完整的路径名，各分支之间必须用扬抑符“^”或制表符分隔。以下列出主树项的快捷方式列表。
pipItem	[out, retval] 指向返回的 ITcSmTreeItem [▶ 103] 接口指针的位置。接口指针允许访问属于树项的特定方法。

返回值

S_OK	函数成功返回。
TSM_E_ITEMNOTFOUND (0x98510001)	路径名不限定现有树项。

快捷键

可以通过快捷键访问每个组态文件中存在的主树项。这些快捷键与语言无关，并且需要较少的内存：

```
"TIIC": shortcut for "I/O Configuration"
"TIID": shortcut for "I/O Configuration^I/O Devices" or "I/O Configuration" TAB "I/O Devices"
"TIRC": shortcut for "Real-Time Configuration"
"TIRR": shortcut for "Real-Time Configuration^Route Settings"
"TIRT": shortcut for " Real-Time Configuration^Additional Tasks" or " Real-Time Configuration" TAB
"Additional Tasks"
"TIRS": shortcut for " Real-Time Configuration^Real-Time Settings" or " Real-Time Configuration" TAB
"Real-Time Settings"
"TIPC": shortcut for "PLC Configuration"
"TINC": shortcut for "NC Configuration"
"TICC": shortcut for "CNC Configuration"
"TIAC": shortcut for "CAM Configuration"
```

示例 (C++):

```
ITcSmTreeItem* ipItem;

BSTR bstrItem = L"TIID^Device 1 (C1220)";

if ( SUCCEEDED(spTsm->LookupTreeItem( bstrItem, &ipItem ))
)

{

// do anything with ipItem
```

```
ipItem->Release();

}
```

示例 (VB)： Dim ipItem As ITcSmTreeItem
set ipItem = spTsm.LookupTreeItem("TIID^Device 1 (C1220)")
' do anything with ipItem

注释

还请参阅有关此

ITcSysManager [▶ 96]

5.2.14 ITcSysManager3::LookupTreeItemById

LookupTreeItemById() 方法将返回由其完整路径名指定的树项的 *ITcTreeItem* 指针。

```
HRESULT LookupTreeItemById(longitemType, longitemId, ITcSmTreeItem**pipItem);
```

Parameters (参数)

itemType	[in] 要查找的 TreeItem 的项目类型。
itemId	[in] TreeItem 的 ID
pipItem	[out, retval] 指向返回的 ITcSmTreeItem [▶ 103] 接口指针的位置。接口指针允许访问属于树项的特定方法。

返回值

S_OK	函数成功返回。
TSM_E_ITEMNOTFOUND (0x98510001)	*itemType itemId* 组合不限定有效树项。

还请参阅有关此

ITcSysManager [▶ 96]

5.2.15 ITcSysManager3::ProduceMappingInfo

生成包括所有当前已组态映射的 XML 输出，例如 PLC 和 I/O 之间。

```
HRESULT ProduceMappingInfo();
```

Parameters (参数)

无

返回值

STRING：返回包括所有已组态映射的 XML 结构。以下片段显示此结构的示例：

```
<VarLinks>
    <OwnerA Name="TIID^Device 1 (EtherCAT)">
        <OwnerB Name="TIXC^Untitled2^Untitled2_Obj1 (CModule1)">
            <Link VarA="Term 1 (EK1100)^Term 3 (EL1008)^Channel 5^Input" VarB="Inputs^Value" />
            <Link VarA="Term 1 (EK1100)^Term 2 (EL2008)^Channel 4^Output" VarB="Outputs^Value" />
        </OwnerB>
    </OwnerA>
    <OwnerA Name="TIPC^Untitled1^Untitled1 Instance">
        <OwnerB Name="TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 2 (EL2008)">
            <Link VarA="PlcTask Outputs^MAIN.bOutput1" VarB="Channel 1^Output" />
            <Link VarA="PlcTask Outputs^MAIN.bOutput3" VarB="Channel 3^Output" />
            <Link VarA="PlcTask Outputs^MAIN.bOutput2" VarB="Channel 2^Output" />
        </OwnerB>
        <OwnerB Name="TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 3 (EL1008)">
```

```
            <Link VarA="PlcTask Inputs^MAIN.bInput1" VarB="Channel 1^Input" />
            <Link VarA="PlcTask Inputs^MAIN.bInput3" VarB="Channel 3^Input" />
            <Link VarA="PlcTask Inputs^MAIN.bInput2" VarB="Channel 2^Input" />
            <Link VarA="PlcTask Inputs^MAIN.bInput4" VarB="Channel 4^Input" />
        </OwnerB>
    </OwnerA>
</VarLinks>
```

此示例显示 PLC <--> I/O 和 TcCOM (C++) <--> I/O 之间的映射。

还请参阅有关此

ITcSysManager [▶ 96]

5.2.16 ITcSysManager3::ConsumeMappingInfo

使用包括项目映射信息的 XML 结构。

```
HRESULT ConsumeMappingInfo(BSTR bstrXml);
```

Parameters (参数)

bstrXml

[in]：采用 XML 结构的字符串。XML 映射信息可以使用 ITcSysManager3::ProduceMappingInfo() [▶ 102] 获得。

还请参阅有关此

ITcSysManager [▶ 96]

5.3 ITcSmTreeItem

5.3.1 ITcSmTreeItem

TwinCAT XAE 组态中的每个树项均由 *ITcSmTreeItem* 接口的实例表示，该接口支持与树项之间的各种交互。

例如，此接口的树项将通过调用用于浏览树形结构的 ITCatSysManager::LookupTreeItem [▶ 101] 方法返回。

属性

ITcSmTreeItem 属性	类型	访问	描述
Name	BSTR	RW	树项名
注释	BSTR	RW	注释
禁用	BOOL	RW	获取/设置树项状态，可能是以下其中一个枚举值： • SMDS_NOT_DISABLED（项目启用） • SMDS_DISABLED（项目禁用） • SMDS_PARENT_DISABLED（**只读**，在其中一个父项禁用时设置）
PathName	BSTR	R	TwinCAT XAE 中的树项路径。各分支用“'^”分隔。路径名可以用于其它方法调用，例如 ITCatSysManager::LookupTreeItem [▶ 101]。请注意，此属性对 TwinCAT XAE 中的树项进行唯一标识。
ItemType	ENUM	R	树项分类，例如设备、接线盒、PLC...。由项目类型 [▶ 105] 定义。
ItemSubType	LONG	RW	树项子类型 [▶ 108]。
Parent	ITcSmTreeItem*	R	指向父树项的指针。
ChildCount	LONG	R	子项数量。此属性计数的子项仅包含树项的主子项（例如，接线盒是设备的主子项，而不是设备过程映像）。要访问所有子项，则使用 _NewEnum 属性。
Child(LONG n)	ITcSmTreeItem*	R	n-th 子项的 ITcSmTreeItem 指针
VarCount((LONG x)	LONG	R	属于树项的变量数量。x = 0 对输入变量计数，x = 1 对输出变量计数
Var(LONG x, LONG n)	ITcSmTreeItem*	R	n-th 变量的 ITcSmTreeItem 指针。x = 0 使用输入变量，x = 1 使用输出变量
_NewEnum	IUnknown* (IEnumVariant*)	R	返回枚举接口，此接口枚举当前树项的**所有**子树项。此属性可能由 For-Each 等语句使用。

方法

ITcSmTreeItem 方法	描述	可用版本
CreateChild [▶ 136]	创建子树项。	TwinCAT 2.11
DeleteChild [▶ 138]	删除子树项。	TwinCAT 2.11
ImportChild [▶ 138]	从剪贴板或先前导出的文件中导入子项。	TwinCAT 2.11
ExportChild [▶ 139]	将子项导出至剪贴板或文件中。	TwinCAT 2.11
ProduceXml [▶ 134]	返回一个字符串，包含项目的 XML 描述及其所有特定于项目的数据和参数。	TwinCAT 2.11
ConsumeXml [▶ 135]	使用一个字符串，包含树项的 XML 描述及其所有特定于项目的数据和参数。	TwinCAT 2.11
GetLastXmlError [▶ 139]	获得最后一条错误的 ConsumeXml() 调用的错误消息。	TwinCAT 2.11
LookupChild [▶ 139]	搜索指定相对路径的子项。	TwinCAT 2.11

ITcSmTreeItem2 方法	描述	可用版本
ResoucesCount	只供内部使用	TwinCAT 2.11
ChangeChildSubType	更改 ITcSmTreeItem 的子类型。	TwinCAT 2.11
ClaimResources	只供内部使用	TwinCAT 2.11

版本信息

要求

所需 TwinCAT 版本
TwinCAT 2.11 及以上版本支持此接口

5.3.2 ITcSmTreeItem 项目类型

TwinCAT 系统管理器/TwinCAT XAE 中的每个树项均被**分类**到各个**组**，例如设备、接线盒、任务...。可通过将树项手动添加到 TwinCAT 系统管理器或 XAE 中，然后通过相应菜单条目导出其 XML 描述来查看树项类型。

- **TwinCAT 系统管理器：**动作 --> 导出 XML 描述
- **TwinCAT XAE：** TwinCAT --> 所选项目 --> 导出 XML 描述

```
<TreeItem>
    <ItemName>Device 1 (EtherCAT)</ItemName>
    <PathName>TIID^Device 1 (EtherCAT)</PathName>
    <ItemType>2</ItemType>
    <ItemId>1</ItemId>
    <ObjectId>#x03010010</ObjectId>
    <ItemSubType>111</ItemSubType>
```

在生成的 XML 文件中，项目类型由节点 <ItemType> 表示。

常规项目类型

项目类型	选项卡	描述
0	TREEITEMTYPE_UNKNOWN	---
1	TREEITEMTYPE_TASK	---
9	TREEITEMTYPE_IECPRJ	---
10	TREEITEMTYPE_CNCPRJ	---
11	TREEITEMTYPE_GSDMOD	Profibus GSD 设备的模块
12	TREEITEMTYPE_CDL	---
13	TREEITEMTYPE_IECLZS	---
14	TREEITEMTYPE_LZSGRP	---
15	TREEITEMTYPE_IODEF	---
16	TREEITEMTYPE_ADDTASKS	---
17	TREEITEMTYPE_DEVICEGRP	---
18	TREEITEMTYPE_MAPGRP	---
30	TREEITEMTYPE_CANPDO	---
31	TREEITEMTYPE_RTIMESET	---
32	TREEITEMTYPE_BCPLC_VARS	---
33	TREEITEMTYPE_FILENAME	---
34	TREEITEMTYPE_DNETCONNECT	---
37	TREEITEMTYPE_FLBCMD	---
43	TREEITEMTYPE_EIPCONNECTION	---
44	TREEITEMTYPE_PNIOAPI	---
45	TREEITEMTYPE_PNIOMOD	---
46	TREEITEMTYPE_PNIOSUBMOD	---
47	TREEITEMTYPE_ETHERNETPROTOCOL	---
200	TREEITEMTYPE_CAMDEF	---
201	TREEITEMTYPE_CAMGROUP	---
202	TREEITEMTYPE_CAM	---
203	TREEITEMTYPE_CAMENCODER	---
204	TREEITEMTYPE_CAMTOOLGRP	---
205	TREEITEMTYPE_CAMTOOL	---
300	TREEITEMTYPE_LINEDEF	---
400	TREEITEMTYPE_ISGDEF	---
401	TREEITEMTYPE_ISGCHANNEL	---
402	TREEITEMTYPE_ISGAGROUP	---
403	TREEITEMTYPE_ISGAXIS	---
500	TREEITEMTYPE_RTSCONFIG	---
501	TREEITEMTYPE_RTSAPP	---
502	TREEITEMTYPE_RTSAPPTASK	---
503	TREEITEMTYPE_RTSADI	---
504	TREEITEMTYPE_CPPCONFIG	---
505	TREEITEMTYPE_SPLCCONFIG	---

I/O 项目类型

项目类型	选项卡	描述
2	TREEITEMTYPE_DEVICE	I/O 设备
3	TREEITEMTYPE_IMAGE	过程映像
4	TREEITEMTYPE_MAPPING	---
5	TREEITEMTYPE_BOX	I/O 接线盒（例如“BK2000”，I/O 设备的子项）
6	TREEITEMTYPE_TERM	I/O 终端（终端耦合器（接线盒）的子项）
7	TREEITEMTYPE_VAR	Variable
8	TREEITEMTYPE_VARGRP	变量组（例如“输入”）
35	TREEITEMTYPE_NVPUBLISHERVAR	---
36	TREEITEMTYPE_NVSUBSCRIBERVAR	---

PLC 项目类型

项目类型	选项卡	描述
600	TREEITEMTYPE_PLCAPP	PLC 应用程序（根 PLC 对象） [1]
601	TREEITEMTYPE_PLCFOLDER	PLC 文件夹 [1]
602	TREEITEMTYPE_PLCPOUPROG	POU 程序 [1]
603	TREEITEMTYPE_PLCPOUFUNC	POU 功能 [1]
604	TREEITEMTYPE_PLCPOUFB	POU 功能块 [1]
605	TREEITEMTYPE_PLCDUTENUM	DUT 枚举数据类型 [1]
606	TREEITEMTYPE_PLCDUTSTRUCT	DUT 结构数据类型 [1]
607	TREEITEMTYPE_PLCDUTUNION	DUT 联合数据类型 [1]
608	TREEITEMTYPE_PLCACTION	PLC 动作 [1]
609	TREEITEMTYPE_PLCMETHOD	PLC 方法 [1]
610	TREEITEMTYPE_PLCITFMETH	PLC 接口方法 [1]
611	TREEITEMTYPE_PLCPROP	PLC 属性 [1]
612	TREEITEMTYPE_PLCITFPROP	PLC 接口属性 [1]
613	TREEITEMTYPE_PLCPROPGET	PLC 属性获取器 [1]
614	TREEITEMTYPE_PLCPROPSET	PLC 属性设置器 [1]
615	TREEITEMTYPE_PLCGVL	GVL（全局变量列表） [1]
616	TREEITEMTYPE_PLCTRANS	PLC 转换 [1]
617	TREEITEMTYPE_PLCLIBMAN	PLC 库管理器 [1]
618	TREEITEMTYPE_PLCITF	PLC 接口 [1]
619	TREEITEMTYPE_PLCVISOBJ	PLC 可视对象 [1]
620	TREEITEMTYPE_PLCVISMAN	PLC 可视管理器 [1]
621	TREEITEMTYPE_PLCTASK	PLC 任务对象 [1]
622	TREEITEMTYPE_PLCPROGREF	PLC 程序引用 [1]
623	TREEITEMTYPE_PLCDUTALIAS	DUT 别名
624	TREEITEMTYPE_PLCEXTDATATYPECONT	PLC 外部数据类型容器 [1]
625	TREEITEMTYPE_PLCTMCDESCRIPTION	PLC TMC 描述文件 [1]
654	TREEITEMTYPE_PLCITFPROPGET	PLC 接口属性获取器
655	TREEITEMTYPE_PLCITFPROPSET	PLC 接口属性设置器

NC 项目类型

项目类型	选项卡	描述
19	TREEITEMTYPE_NCDEF	---
20	TREEITEMTYPE_NCAXISES	---
21	TREEITEMTYPE_NCCHANNEL	NC 通道
22	TREEITEMTYPE_NCAXIS	NC 轴
23	TREEITEMTYPE_NCENCODER	---
24	TREEITEMTYPE_NCDRIVE	---
25	TREEITEMTYPE_NCCONTROLLER	---
26	TREEITEMTYPE_NCGROUP	---
27	TREEITEMTYPE_NCINTERPRETER	---
40	TREEITEMTYPE_NCTABLEGRP	---
41	TREEITEMTYPE_NCTABLE	---
42	TREEITEMTYPE_NCTABLESLAVE	---

要求

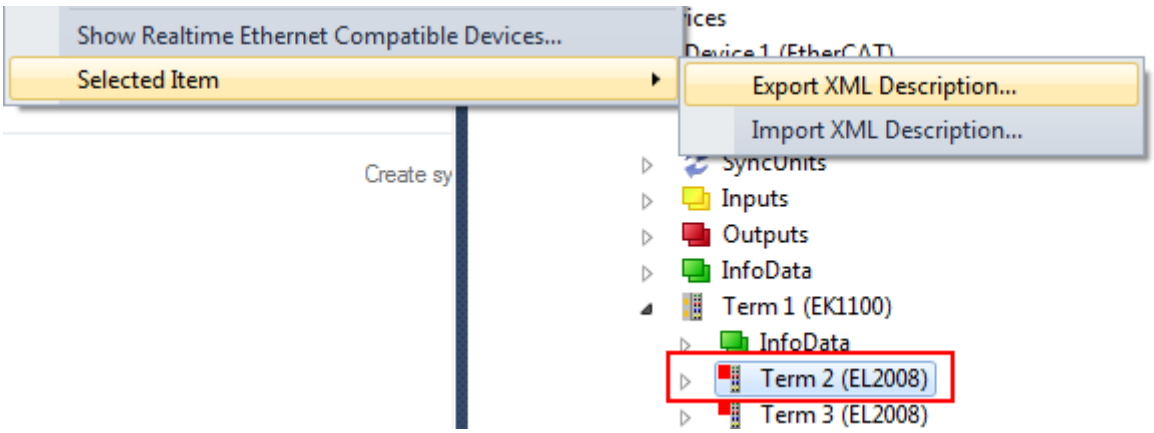
注意事项	
[1]	需要 TwinCAT 3.1

5.3.3 树项子类型

5.3.3.1 ITcSmTreeItem 项目子类型

项目子类型指定正在使用的**设备**、**接线盒**或**终端**类型，例如 2408 子类型标识 KL2408 数字输出终端。可通过将项目手动添加到 TwinCAT 系统管理器或 XAE 中，然后通过相应菜单条目导出其 XML 描述，来查看项目子类型。

- **TwinCAT 系统管理器：** 动作 --> 导出 XML 描述
- **TwinCAT XAE：** TwinCAT --> 所选项目 --> 导出 XML 描述



在生成的 XML 文件中，子类型由节点 <ItemSubType> 表示。在上述示例中，已导出 EL2008 终端的 XML 描述。XML 文件显示此终端具有子类型 9099。

```
<TreeItem>
  <ItemName>Term 2 (EL2008)</ItemName>
  <PathName>TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 2 (EL2008)</PathName>
  <ItemType>5</ItemType>
  <ItemId>2</ItemId>
  <ObjectId>#x03020002</ObjectId>
  <ItemSubType>9099</ItemSubType>
  <ItemSubTypeName>EL2008 8Ch. Dig. Output 24V, 0.5A</ItemSubTypeName>
  <ChildCount>0</ChildCount>
  <Disabled>false</Disabled>
```

下表提供了部分可用子类型的概述。如果您的设备未列出，请执行上述步骤来确定指定设备的子类型。

快捷键：

- 设备 [▶ 111]
- 接线盒 [▶ 117]
- 终端：E-Bus [▶ 109] (ELxxxx)
- 终端：K-Bus 数字输入 [▶ 122] (KL1xxx)
- 终端：K-Bus 数字输出 [▶ 124] (KL2xxx)
- 终端：K-Bus 模拟输入 [▶ 127] (KL3xxx)
- 终端：K-Bus 模拟输出 [▶ 129] (KL4xxx)
- 终端：K-Bus 位置测量 [▶ 129] (KL5xxx)
- 终端：K-Bus 通信 [▶ 130] (KL6xxx)
- 终端：K-Bus 电源 [▶ 131] (KL8xxx)
- 终端：K-Bus 安全 [▶ 133] (KLx90x)
- 终端：K-Bus 系统 [▶ 132] (KL9xxx)

5.3.3.2 ITcSmTreeItem 项目子类型：E-Bus

由于架构不同，E-Bus **接线盒**、**终端**和**模块**的处理方式与其 K-Bus 对应项不同，例如使用 CreateChild() [▶ 136] 方法创建时。每个 K-Bus 终端将根据其特定子类型指定，而 E-Bus 终端由一个通用子类型识别，然后通过其“**产品版本号**”指定，该版本号将在 CreateChild() 方法中用作 vInfo 参数。

子类型	描述
9099	所有 EtherCAT 终端的通用子类型。如果是 CreateChild() [▶ 136]，特定终端将通过 vInfo 参数定义。

此规则有一些例外，例如 RS232 终端。下表提供了这些例外情况的概述：

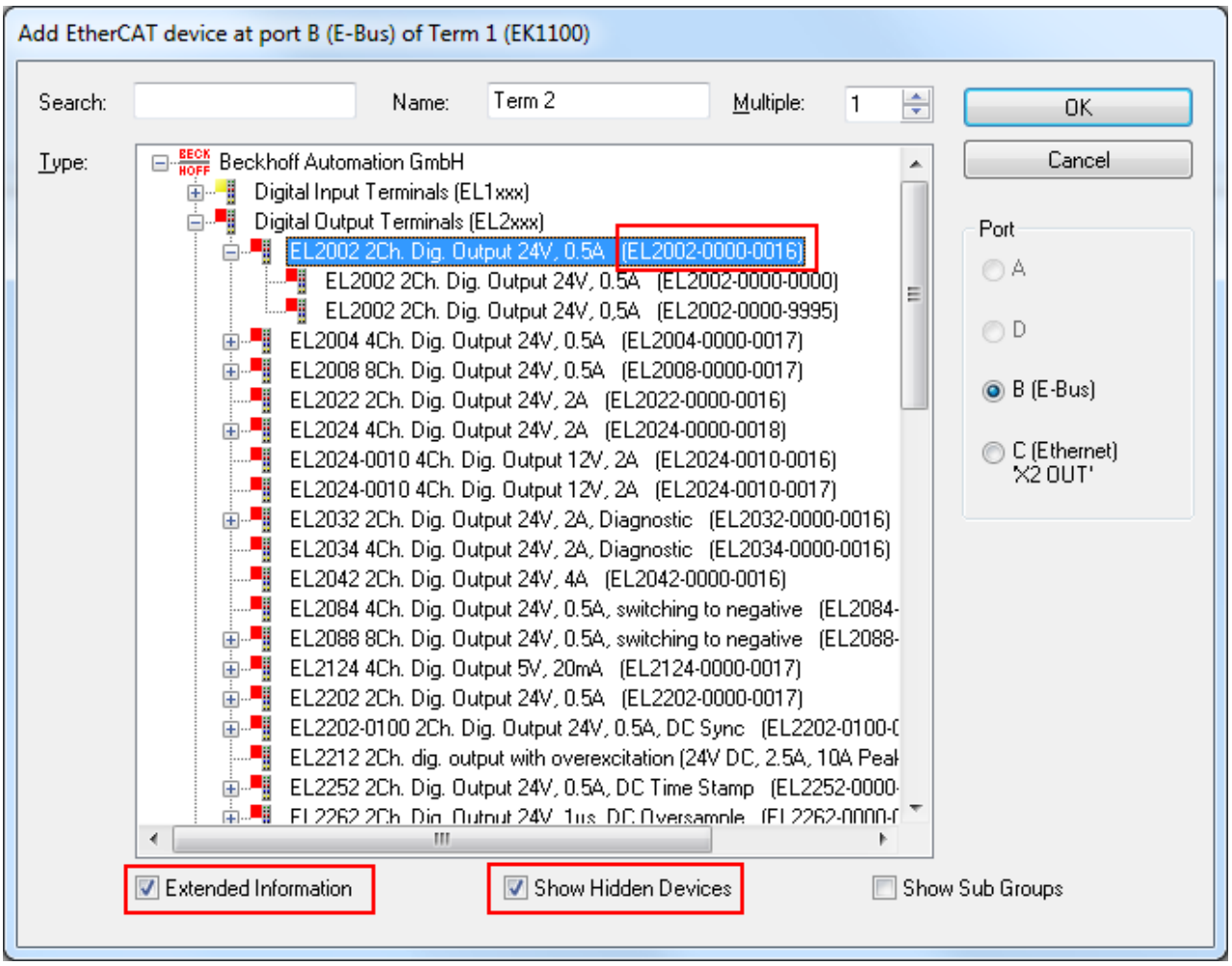
I/O	ItemSubType	描述
EP6002	9101	RS232 / RS422 / RS485 接口终端
EL6001	9101	RS232 接口终端
EL6002	9101	RS232 接口终端 (2 通道)
EL6021	9103	RS422 / RS485 接口终端
EL6022	9103	RS422 / RS485 接口终端 (2 通道)

代码片段 (C#)

```
ITcSmTreeItem ek1100 = systemManager.LookupTreeItem("TIID^EtherCAT Master^EK1100");
ek1100.CreateChild("EL2002 - 1", 9099, "", "EL2002-0000-0016");
```

产品版本号

每个 E-Bus 接线盒/终端/模块都有自己的产品版本号，此版本号可通过导出其 XML 描述 [▶ 22]或在 TwinCAT XAE 中的“Add Device” (添加设备) 对话框中查看。



例如，EL2002 数字输出终端带有产品版本号 EL2002-0000-0016，此版本号也可以在其 XML 描述中查看：

```
<EtherCAT>
- <Slave>
  - <Info>
    - <Name>
        <![CDATA[ Term 3 (EL2002) ]]>
      </Name>
      <PhysAddr>1002</PhysAddr>
      <AutoIncAddr>65535</AutoIncAddr>
      <Physics>KK</Physics>
      <VendorId>2</VendorId>
      <ProductCode>131215442</ProductCode>
      <RevisionNo>1048576</RevisionNo>
      <SerialNo>0</SerialNo>
      <ProductRevision>EL2002-0000-0016</ProductRevision>
      <Type>EL2002</Type>
    </Info>
```

要获得树项的 XML 描述，只需进行以下操作：

- **TwinCAT 2**：将项目添加到系统管理器并选定，然后从菜单中选择“Actions” (动作) --> “Export XML description” (导出 XML 描述)

- **TwinCAT 3**：将项目添加到 XAE 并选定，然后从菜单中选择“TwinCAT” --> “Selected item” (所选项目) --> “Export XML description” (导出 XML 描述)

5.3.3.3 设备

5.3.3.3.1 ITcSmTreeItem 项目子类型：设备

设备：其它项

子类型	选项卡	描述
0	IODEVICETYPE_UNKNOWN	---
6	IODEVICETYPE_BKHFPC	Beckhoff 工业PC C2001
9	IODEVICETYPE_LPTPORT	LPT 端口
10	IODEVICETYPE_DPRAM	通用 DPRAM
11	IODEVICETYPE_COMPORT	COM 接口
18	IODEVICETYPE_FCXXXX	Beckhoff-FeldbusCard
32	IODEVICETYPE_SMB	主板系统管理总线
43	IODEVICETYPE_BKHFNCBP	Beckhoff NC Rückwand
44	IODEVICETYPE_SERCANSPCI	Sercos Master (SICAN/IAM PCI)
46	IODEVICETYPE_SERCONPCI	Sercon 410B 或 816 Chip Master 或 Slave (PCI)
53	IODEVICETYPE_BKHFAH2000	Beckhoff AH2000 (液压背板)
55	IODEVICETYPE_AH2000MC	Beckhoff-AH2000 mit Profibus-MC

设备：Beckhoff CX 终端设备

子类型	选项卡	描述
120	IODEVICETYPE_CX5000	CX5000 终端设备
135	IODEVICETYPE_CX8000	CX8000 终端设备
105	IODEVICETYPE_CX9000_BK	CX9000 终端设备
65	IODEVICETYPE_CX1100_BK	CX1100 终端设备
124	IODEVICETYPE_CCAT	Beckhoff CCAT 适配器

设备：Beckhoff CP 设备

子类型	选项卡	描述
14	IODEVICETYPE_BKHFCP2030	Beckhoff CP2030 (Pannel-Link)
31	IODEVICETYPE_BKHFCP9030	Beckhoff CP9030 (带 UPS 和 ISA 的 Pannel-Link)
52	IODEVICETYPE_BKHFCP9040	Beckhoff CP9040 (CP-PC)
54	IODEVICETYPE_BKHFCP9035	Beckhoff CP9035 (带 UPS 和 PCI 的 Pannel-Link)
116	IODEVICETYPE_BKHFCP6608	Beckhoff CP6608 (IXP PC)

设备：Beckhoff BC/BX 设备

子类型	选项卡	描述
77	IODEVICETYPE_BX_BK	BX Klemmenbus 接口
78	IODEVICETYPE_BX_M510	BX SSB-Master
103	IODEVICETYPE_BC8150	BCXX50 Serial Slave
104	IODEVICETYPE_BX9000	BX9000 Ethernet Slave
107	IODEVICETYPE_BC9050	BC9050 Etherent Slave
108	IODEVICETYPE_BC9120	BC9120 Etherent Slave
110	IODEVICETYPE_BC9020	BC9020 Etherent Slave

设备：Beckhoff EtherCAT

子类型	选项卡	描述
94	IODEVICETYPE_ETHERCAT	弃用：EtherCAT Master。使用“111”代替。
111	IODEVICETYPE_ETHERCATPROT	EtherCAT Master
130	IODEVICETYPE_ETHERCATSLV	EtherCAT Slave
106	IODEVICETYPE_EL6601	通过 EL6601 的 EtherCAT 自动化协议
112	IODEVICETYPE_ETHERNETNVPROT	EtherCAT 自动化协议 (网络变量)

设备：Beckhoff Lightbus Master/Slave

子类型	选项卡	描述
67	IODEVICETYPE_CX1500_M200	PC104 Lightbus-Master
68	IODEVICETYPE_CX1500_B200	PC104 Lightbus-Slave
36	IODEVICETYPE_FC200X	Beckhoff-Lightbus-I/II-PCI 卡
114	IODEVICETYPE_EL6720	Beckhoff-Lightbus-EtherCAT-Klemme
1	IODEVICETYPE_C1220	Beckhoff Lightbus ISA 接口卡 C1220
2	IODEVICETYPE_C1200	Beckhoff Lightbus ISA 接口卡 C1200

设备：Beckhoff Profibus Master/Slave

子类型	选项卡	描述
69	IODEVICETYPE_CX1500_M310	PC104 ProfiBus-Master
70	IODEVICETYPE_CX1500_B310	PC104 ProfiBus-Slave
33	IODEVICETYPE_PBMON	Beckhoff-PROFIBUS 监视器
38	IODEVICETYPE_FC3100	Beckhoff-Profibus-PCI 卡
56	IODEVICETYPE_FC3100MON	Beckhoff-Profibus-Monitor-PCI-Karte
60	IODEVICETYPE_FC3100SLV	Beckhoff-Profibus-PCI-Karte als Slave
79	IODEVICETYPE_BX_B310	BX ProfiBus-Slave
83	IODEVICETYPE_BC3150	BCxx50 ProfiBus-Slave
86	IODEVICETYPE_EL6731	Beckhoff-Profibus-EtherCAT-Klemme
97	IODEVICETYPE_EL6731SLV	Beckhoff-Profibus-Slave-EtherCAT-Klemme

设备：Beckhoff CANopen Master/Slave

子类型	选项卡	描述
71	IODEVICETYPE_CX1500_M510	PC104 CANopen-Master
72	IODEVICETYPE_CX1500_B510	PC104 CANopen-Slave
39	IODEVICETYPE_FC5100	Beckhoff-CanOpen-PCI 卡
58	IODEVICETYPE_FC5100MON	Beckhoff-CANopen-Monitor-PCI-Karte
61	IODEVICETYPE_FC5100SLV	Beckhoff-CanOpen-PCI-Karte als Slave
81	IODEVICETYPE_BX_B510	BX CANopen-Slave
84	IODEVICETYPE_BC5150	BCxx50 CANopen-Slave
87	IODEVICETYPE_EL6751	Beckhoff-CanOpen-EtherCAT-Klemme
98	IODEVICETYPE_EL6751SLV	Beckhoff-CanOpen-Slave-EtherCAT-Klemme

设备：Beckhoff DeviceNet Master/Slave

子类型	选项卡	描述
73	IODEVICETYPE_CX1500_M520	PC104 DeviceNet-Master
74	IODEVICETYPE_CX1500_B520	PC104 DeviceNet-Slave
41	IODEVICETYPE_FC5200	Beckhoff-DeviceNet-PCI 卡
59	IODEVICETYPE_FC5200MON	Beckhoff-DeviceNet-Monitor-PCI-Karte
62	IODEVICETYPE_FC5200SLV	Beckhoff-DeviceNet-PCI-Karte als Slave
82	IODEVICETYPE_BX_B520	BX DeviceNet-Slave
85	IODEVICETYPE_BC5250	BCxx50 DeviceNet-Slave
88	IODEVICETYPE_EL6752	Beckhoff-DeviceNet-EtherCAT-Klemme
99	IODEVICETYPE_EL6752SLV	Beckhoff-DeviceNet-Slave-EtherCAT-Klemme

设备：Beckhoff Sercos Master/Slave

子类型	选项卡	描述
75	IODEVICETYPE_CX1500_M750	PC104 Sercos-Master
76	IODEVICETYPE_CX1500_B750	PC104 Sercos-Slave
48	IODEVICETYPE_FC7500	Beckhoff-SERCOS-PCI 卡

设备：以太网

子类型	选项卡	描述
66	IODEVICETYPE_ENETRTMP	以太网实时微端口
109	IODEVICETYPE_ENETADAPTER	实时以太网适配器（多协议处理器）
138	---	实时以太网协议（BK90xx，AX2000-B900）
45	IODEVICETYPE_ETHERNET	虚拟以太网接口

设备：USB

子类型	选项卡	描述
57	IODEVICETYPE_USB	虚拟 USB 接口
125	---	虚拟 USB 接口（远程）

设备：Hilscher

子类型	选项卡	描述
4	IODEVICETYPE_CIF30DPM	ISA ProfiBus-Master 2 kByte (Hilscher 卡)
5	IODEVICETYPE_CIF40IBSM	ISA Interbus-S-Master 2 kByte (Hilscher 卡)
12	IODEVICETYPE_CIF30CAN	ISA CANopen-Master (Hilscher 卡)
13	IODEVICETYPE_CIF30PB	ISA ProfiBus-Master 8 kByte (Hilscher 卡)
16	IODEVICETYPE_CIF30IBM	ISA Interbus-S-Master (Hilscher 卡)
17	IODEVICETYPE_CIF30DNM	ISA DeviceNet-Master (Hilscher 卡)
19	IODEVICETYPE_CIF50PB	PCI ProfiBus-Master 8 kByte (Hilscher 卡)
20	IODEVICETYPE_CIF50IBM	PCI Interbus-S-Master (Hilscher 卡)
21	IODEVICETYPE_CIF50DNM	PCI DeviceNet-Master (Hilscher 卡)
22	IODEVICETYPE_CIF50CAN	PCI CANopen-Master (Hilscher 卡)
23	IODEVICETYPE_CIF60PB	PCMCIA ProfiBus-Master (Hilscher 卡)
24	IODEVICETYPE_CIF60DNM	PCMCIA DeviceNet-Master (Hilscher 卡)
25	IODEVICETYPE_CIF60CAN	PCMCIA CANopen-Master (Hilscher 卡)
26	IODEVICETYPE_CIF104DP	PC104 ProfiBus-Master 2 kByte (Hilscher 卡)
27	IODEVICETYPE_C104PB	PC104 ProfiBus-Master 8 kByte (Hilscher 卡)
28	IODEVICETYPE_C104IBM	PC104 Interbus-S-Master 2 kByte (Hilscher 卡)
29	IODEVICETYPE_C104CAN	PC104 CANopen-Master (Hilscher 卡)
30	IODEVICETYPE_C104DNM	PC104 DeviceNet-Master (Hilscher 卡)
35	IODEVICETYPE_CIF60IBM	PCMCIA Interbus-S-Master (Hilscher 卡)
49	IODEVICETYPE_CIF30IBS	ISA Interbus-S-Slave (Hilscher 卡)
50	IODEVICETYPE_CIF50IBS	PCI Interbus-S-Slave (Hilscher 卡)
51	IODEVICETYPE_C104IBS	PC104 Interbus-S-Slave (Hilscher 卡)
89	IODEVICETYPE_COMPB	COM ProfiBus-Master 8 kByte (Hilscher 卡)
90	IODEVICETYPE_COMIBM	COM Interbus-S-Master (Hilscher 卡)
91	IODEVICETYPE_COMDNM	COM DeviceNet-Master (Hilscher 卡)
92	IODEVICETYPE_COMCAN	COM CANopen-Master (Hilscher 卡)
93	IODEVICETYPE_COMIBS	COM CANopen-Slave (Hilscher 卡)
100	IODEVICETYPE_C104PPB	PC104+ ProfiBus-Master 8 kByte (Hilscher 卡)

子类型	选项卡	描述
101	IODEVICETYPE_C104PCAN	PC104+ CANopen-Master (Hilscher 卡)
102	IODEVICETYPE_C104PDNM	PC104+ DeviceNet-Master (Hilscher 卡)

设备：Profinet / Profibus

子类型	选项卡	描述
3	IODEVICETYPE_SPC3	ProfiBus Slave (Siemens 卡)
7	IODEVICETYPE_CP5412A2	ProfiBus-Master (Siemens 卡)
34	IODEVICETYPE_CP5613	PCI ProfiBus-Master (Siemens 卡)
113	IODEVICETYPE_PROFINETIOCONTROLLER	PROFINET Master
115	IODEVICETYPE_PROFINETIODEVICE	PROFINET Slave

设备：Indramat

子类型	选项卡	描述
8	IODEVICETYPE_SERCANSISA	Sercos Master (Indramat)

设备：Phoenix

子类型	选项卡	描述
15	IODEVICETYPE_IBSSCIT	Interbus-S-Master (Phoenix 卡)
47	IODEVICETYPE_IBSSCRIRTLK	带 Slave-Part LWL Basis 的 Interbus-S-Master (Phoenix 卡)
63	IODEVICETYPE_IBSSCITPCI	PCI Interbus-S-Master (Phoenix 卡)
64	IODEVICETYPE_IBSSCRILKPCI	PCIInterbus-S-Master mit Slave-Teil auf LWL Basis (Phoenix 卡)
80	IODEVICETYPE_IBSSCRIRTPCI	PCIInterbus-S-Master mit Slave-Teil auf Kupfer Basis (Phoenix 卡)

5.3.3.4 接线盒

5.3.3.4.1 ITcSmTreeItem 项目子类型：接线盒

子类型	选项卡	描述
0	BOXTYPE_UNKNOWN	---
1	BOXTYPE_BK2000	BK2000 Lightbus 耦合器
2	BOXTYPE_M1400	M1400 Lightbus 数字输入/输出模块
3	BOXTYPE_M2400	M2400 Lightbus 输入/输出模块
4	BOXTYPE_M3120_1	M3120 Lightbus 增量编码器接口
5	BOXTYPE_M3120_2	M3120 Lightbus 增量编码器接口
6	BOXTYPE_M3120_3	M3120 Lightbus 增量编码器接口
7	BOXTYPE_M3120_4	M3120 Lightbus 增量编码器接口
8	BOXTYPE_M3000	M3000 绝对/增量编码器
9	BOXTYPE_C1120	---
10	BOXTYPE_BK2010	BK2010 Lightbus 耦合器
11	BOXTYPE_AX2000	---
12	BOXTYPE_M2510	---
20	BOXTYPE_BK2020	BK2020 Lightbus 耦合器
21	BOXTYPE_BC2000	---
31	BOXTYPE_FOX20	---
32	BOXTYPE_FOX50	---
33	BOXTYPE_FOXRK001	---
34	BOXTYPE_FOXRK002	---
35	BOXTYPE_CP1001	---
40	BOXTYPE_IPXB2	---
41	BOXTYPE_ILXB2	---
42	BOXTYPE_ILXC2	---
50	BOXTYPE_TSMBOX_200	---
51	BOXTYPE_BX2000	---
52	BOXTYPE_CX1500_B200	---
1001	BOXTYPE_BK3000	---
1002	BOXTYPE_BK3100	---
1003	BOXTYPE_PBDP_GSD	---
1004	BOXTYPE_BK3010	---
1005	BOXTYPE_BK3110	---
1006	BOXTYPE_BK3500	---
1007	BOXTYPE_LC3100	---
1008	BOXTYPE_PBDP_DRIVE	---
1009	BOXTYPE_BK3020	---
1010	BOXTYPE_BK3120	---
1011	BOXTYPE_BC3100	---
1012	BOXTYPE_PBDP_DRIVE2	---
1013	BOXTYPE_PBDP_DRIVE3	---
1014	BOXTYPE_PBDP_DRIVE4	---
1015	BOXTYPE_PBDP_DRIVE5	---
1016	BOXTYPE_PBDP_DRIVE6	---
1017	BOXTYPE_PBDP_DRIVE7	---
1018	BOXTYPE_PBDP_DRIVE8	---
1019	BOXTYPE_BK3150	---
1020	BOXTYPE_BC3150	---
1021	BOXTYPE_BK3XXX	---
1022	BOXTYPE_BC3XXX	---

子类型	选项卡	描述
1030	BOXTYPE_IPXB3	---
1031	BOXTYPE_ILXB3	---
1032	BOXTYPE_ILXC3	---
1040	BOXTYPE_TSMBOX_310	---
1041	BOXTYPE_BX3100	---
1042	BOXTYPE_CX1500_B310	---
1043	BOXTYPE_FC310X_SLAVE	---
1044	BOXTYPE_EL6731_SLAVE	---
1051	BOXTYPE_AX2000_B310	---
1100	BOXTYPE_TCPBDPSLAVE	---
1101	BOXTYPE_TCFDLAGAG	---
1102	BOXTYPE_TCMPI	---
1103	BOXTYPE_TCPBMCSLAVE	---
1104	BOXTYPE_TCPBMCSLAVE2	---
1105	BOXTYPE_TCPBMCSLAVE3	---
1106	BOXTYPE_TCPBMCSLAVE4	---
1107	BOXTYPE_TCPBMCSLAVE5	---
1108	BOXTYPE_TCPBMCSLAVE6	---
1109	BOXTYPE_TCPBMCSLAVE7	---
1110	BOXTYPE_TCPBMCSLAVE8	---
1111	BOXTYPE_TCPBMONSLAVE	---
2001	BOXTYPE_BK4000	---
2002	BOXTYPE_IBS_GENERIC	---
2003	BOXTYPE_IBS_BK	---
2004	BOXTYPE_BK4010	---
2005	BOXTYPE_BK4500	---
2006	BOXTYPE_BK4510	---
2007	BOXTYPE_IBS_SLAVEBOX	---
2008	BOXTYPE_BC4000	---
2009	BOXTYPE_BK4020	---
2020	BOXTYPE_CP2020	---
2030	BOXTYPE_IPXB4	---
2031	BOXTYPE_ILXB4	---
2032	BOXTYPE_ILXC4	---
3001	BOXTYPE_SERCOSAXIS	---
3011	BOXTYPE_BK7500	---
3012	BOXTYPE_BK7510	---
3013	BOXTYPE_BK7520	---
3021	BOXTYPE_SERCOSMASTERBOX	---
3031	BOXTYPE_SERCOSSLAVEBOX	---
4001	BOXTYPE_BK8100	BK8100 总线耦合器
4002	BOXTYPE_BK8110	BK8110 总线耦合器
4003	BOXTYPE_BK8000	BK8000 总线耦合器
4004	BOXTYPE_BK8010	BK8010 总线耦合器
4005	BOXTYPE_CP9040	CP9040 PCB
4011	BOXTYPE_BC8000	BC8000 总线终端控制器
4012	BOXTYPE_BC8100	BC8100 总线终端控制器
4030	BOXTYPE_IPXB80	---
4031	BOXTYPE_ILXB80	---

子类型	选项卡	描述
4032	BOXTYPE_ILXC80	---
4040	BOXTYPE_IPXB81	---
4041	BOXTYPE_ILXB81	---
4042	BOXTYPE_ILXC81	---
4050	BOXTYPE_BC8150	BC8150 总线终端控制器
5001	BOXTYPE_BK5100	BK5100 总线耦合器
5002	BOXTYPE_BK5110	BK5110 总线耦合器
5003	BOXTYPE_CANNODE	---
5004	BOXTYPE_BK5120	BK5120 总线耦合器
5005	BOXTYPE_LC5100	---
5006	BOXTYPE_CANDRIVE	---
5007	BOXTYPE_AX2000_B510	---
5008	BOXTYPE_BK5150	BK5150 总线耦合器
5009	BOXTYPE_BK5151	BK5151 总线耦合器
5011	BOXTYPE_BC5150	BC5150 总线终端控制器
5030	BOXTYPE_IPXB51	---
5031	BOXTYPE_ILXB51	---
5032	BOXTYPE_ILXC51	---
5040	BOXTYPE_TSMBOX_510	---
5041	BOXTYPE_BX5100	BX5100 CANopen 总线终端控制器
5042	BOXTYPE_CX1500_B510	---
5043	BOXTYPE_FC510XSLV	---
5050	BOXTYPE_TCCANSLAVE	---
5051	BOXTYPE_CANQUEUE	CAN 接口
5201	BOXTYPE_BK5200	---
5202	BOXTYPE_BK5210	---
5203	BOXTYPE_DEVICENET	---
5204	BOXTYPE_BK5220	---
5205	BOXTYPE_LC5200	---
5211	BOXTYPE_BK52XX	---
5212	BOXTYPE_BC52XX	---
5230	BOXTYPE_IPXB52	---
5231	BOXTYPE_ILXB52	---
5232	BOXTYPE_ILXC52	---
5250	BOXTYPE_TCDNSLAVE	---
9001	BOXTYPE_BK9000	BK9000 以太网 TCP/IP 总线耦合器
9002	BOXTYPE_BK9100	BK9100 以太网 TCP/IP 总线耦合器
9005	BOXTYPE_BK9050	BK9050 以太网 TCP/IP 总线耦合器
9011	BOXTYPE_BC9000	BC9000 以太网 TCP/IP 总线终端控制器
9012	BOXTYPE_BC9100	BC9100 以太网 TCP/IP 总线终端控制器
9013	BOXTYPE_BX9000	BX9000 以太网 TCP/IP 总线终端控制器
9014	BOXTYPE_BX9000SLV	---
9015	BOXTYPE_BC9050	BC9050 以太网 TCP/IP 总线终端控制器
9016	BOXTYPE_BC9050SLV	BC9050 以太网 TCP/IP 总线终端控制器从站

子类型	选项卡	描述
9017	BOXTYPE_BC9120	BC9120 以太网 TCP/IP 总线终端控制器
9018	BOXTYPE_BC9120SLV	BC9120 以太网 TCP/IP 总线终端控制器从站
9019	BOXTYPE_BC9020	BC9020 以太网 TCP/IP 总线终端控制器
9020	BOXTYPE_BC9020SLV	BC9020 以太网 TCP/IP 总线终端控制器从站
9030	BOXTYPE_IPXB9	---
9031	BOXTYPE_ILXB9	---
9032	BOXTYPE_ILXC9	---
9041	BOXTYPE_REMOTETASK	---
9051	BOXTYPE_NV_PUB	网络发布服务器。
9052	BOXTYPE_NV_SUB	网络订阅服务器。
9053	BOXTYPE_NV_PUBVAR	发布服务器变量。
9054	BOXTYPE_NV_SUBVAR	订阅服务器变量。
9055	BOXTYPE_NV_PUBDATA	发布服务器数据对象。
9056	BOXTYPE_NV_SUBDATA	订阅服务器数据对象。
9061	BOXTYPE_AX2000_B900	---
9071	BOXTYPE_FLB_FRAME	---
9081	BOXTYPE_BK1120	---
9085	BOXTYPE_IPXB11	---
9086	BOXTYPE_ILXB11	---
9087	BOXTYPE_ILXC11	---
9105	BOXTYPE_FSOESLAVE	---
9121	BOXTYPE_PNIODEVICE	---
9122	BOXTYPE_PNIOTCDEVICE	---
9123	BOXTYPE_PNIODEVICEINTF	Profinet TwinCAT 设备接口
9124	BOXTYPE_PNIO_DRIVE	---
9125	BOXTYPE_PNIOBK9103	---
9126	BOXTYPE_PNIOILB903	---
9132	BOXTYPE_BK9053	BK9053 K-Bus 耦合器，PROFINET IO RT
9133	BOXTYPE_EIPSLAVEINTF	---
9143	BOXTYPE_PTPSLAVEINTF	---
9151	BOXTYPE_RAWUDPINTF	---
9500	BOXTYPE_BK9500	BK9500
9510	BOXTYPE_BK9510	BK9510
9520	BOXTYPE_BK9520	BK9520
9591	BOXTYPE_CPX8XX	---
9700	BOXTYPE_CX1102	---
9701	BOXTYPE_CX1103	---
9702	BOXTYPE_CX1190	---

5.3.3.5 终端

5.3.3.5.1 ITcSmTreeItem 项目子类型：终端：K-Bus 数字输入 (KL1)

子类型	描述
1002	KL1002 2 通道数字输入终端
1012	KL1012 2 通道数字输入终端
1032	KL1032 2 通道数字输入终端
1052	KL1052 2 通道数字输入终端
1104	KL1104 4 通道数字输入终端
1114	KL1114 4 通道数字输入终端
1124	KL1124 4 通道数字输入终端
1154	KL1154 4 通道数字输入终端
1164	KL1164 4 通道数字输入终端
1184	KL1184 4 通道数字输入终端
1194	KL1194 4 通道数字输入终端
1212	KL1212 2 通道数字输入终端
1232	KL1232 2 通道数字输入终端
1302	KL1302 2 通道数字输入终端
1304	KL1304 4 通道数字输入终端
1312	KL1312 2 通道数字输入终端
1314	KL1314 4 通道数字输入终端
1352	KL1352 2 通道数字输入终端
1362	KL1362 2 通道数字输入终端
1382	KL1382 2 通道数字输入终端
1402	KL1402 2 通道数字输入终端
1404	KL1404 4 通道数字输入终端
1408	KL1408 8 通道数字输入终端
1412	KL1412 2 通道数字输入终端
1414	KL1414 4 通道数字输入终端
1418	KL1418 8 通道数字输入终端
1434	KL1434 4 通道数字输入终端
1488	KL1488 8 通道数字输入终端
1498	KL1498 8 通道数字输入终端
1501	KL1501 单通道数字输入终端
1512	KL1512 2 通道数字输入终端
1702	KL1702 2 通道数字输入终端
1712	KL1712 2 通道数字输入终端
16778928	KL1712-0060 2 通道数字输入终端
1722	KL1722 2 通道数字输入终端
1804	KL1804 4 通道数字输入终端
1808	KL1808 8 通道数字输入终端
1809	KL1809 16 通道数字输入终端
1814	KL1814 4 通道数字输入终端
1819	KL1819 16 通道数字输入终端
1859	KL1859 8 通道数字输入终端
1862	KL1862 16 通道数字输入终端
16779078	KL1862-0010 16 通道数字输入终端
1872	KL1872 16 通道数字输入终端
1889	KL1889 16 通道数字输入终端
1202	KL1202 2 通道数字输入终端

5.3.3.5.2 ITcSmTreeItem 项目子类型：终端：K-Bus 数字输出 (KL2)

子类型	描述
2408	KL2408 8 通道数字输出终端
2012	KL2012 2 通道数字输出终端
2022	KL2022 2 通道数字输出终端
2032	KL2032 2 通道数字输出终端
2114	KL2114 4 通道数字输出终端
2124	KL2124 4 通道数字输出终端
2134	KL2134 4 通道数字输出终端
2184	KL2184 4 通道数字输出终端
2212	KL2212 2 通道数字输出终端
2404	KL2404 4 通道数字输出终端
2212	KL2212 2 通道数字输出终端
2404	KL2404 4 通道数字输出终端
2408	KL2408 8 通道数字输出终端
2284	KL2284 4 通道数字输出终端
2424	KL2424 4 通道数字输出终端
2442	KL2442 2 通道数字输出终端
2488	KL2488 8 通道数字输出终端
2502	KL2502 2 通道 PWM 输出终端
2512	KL2512 2 通道 PWM 输出终端
2521	KL2521 单通道脉冲串输出终端
2531	KL2531 单通道步进电机终端
16779747	KL2531-1000 单通道步进电机终端
2532	KL2532 2 通道直流电动机放大器输出终端
2535	KL2535 2 通道 PWM 放大器输出终端
2541	KL2541 单通道步进电机终端
16779757	KL2541-1000 单通道步进电机终端
2542	KL2542 2 通道直流电动机放大器终端
2545	KL2545 2 通道 PWM 放大器输出终端
2552	KL2552 2 通道直流电机放大器终端
2602	KL2602 2 通道输出继电器终端
2612	KL2612 2 通道输出继电器终端
2622	KL2622 2 通道输出继电器终端
2631	KL2631 单通道电源输出继电器终端
2641	KL2641 单通道电源输出继电器终端
2652	KL2652 2 通道电源输出继电器终端
2701	KL2701 单通道固态负载继电器终端
2702	KL2702 2 通道输出固态继电器终端
16779918	KL2702-0002 2 通道输出固态继电器终端
33557134	KL2702-0020 2 通道输出固态继电器终端
2712	KL2712 2 通道三端双向可控硅输出终端
2722	KL2722 2 通道三端双向可控硅输出终端
2732	KL2732 2 通道三端双向可控硅输出终端
2744	KL2744 4 通道输出固态继电器
2751	KL2751 单通道通用调光器终端
33557183	KL2751-1200 单通道通用调光器终端
2761	KL2761 单通道通用调光器终端
2784	KL2784 4 通道输出终端
2791	KL2791 单通道速度控制器终端
33557223	KL2791-1200 单通道速度控制器终端

子类型	描述
2794	KL2794 4 通道输出终端
2808	KL2808 8 通道输出终端
2809	KL2809 16 通道输出终端
2872	KL2872 16 通道输出终端
2889	KL2889 16 通道输出终端

5.3.3.5.3 ITcSmTreeItem 项目子类型：终端：K-Bus 模拟输入 (KL3)

子类型	描述
3001	KL3001 单通道模拟输入终端
3002	KL3002 3 通道模拟输入终端
3011	KL3011 单通道模拟输入终端
3012	KL3012 2 通道模拟输入终端
3021	KL3021 单通道模拟输入终端
3022	KL3022 2 通道模拟输入终端
3041	KL3041 单通道模拟输入终端
3042	KL3042 2 通道模拟输入终端
3044	KL3044 4 通道模拟输入终端
3051	KL3051 单通道模拟输入终端
3052	KL3052 2 通道模拟输入终端
3054	KL3054 4 通道模拟输入终端
3061	KL3061 单通道模拟输入终端
3062	KL3062 2 通道模拟输入终端
3064	KL3064 4 通道模拟输入终端
3102	KL3102 2 通道模拟输入终端
3112	KL3112 2 通道模拟输入终端
3122	KL3122 2 通道模拟输入终端
3132	KL3132 2 通道模拟输入终端
3142	KL3142 2 通道模拟输入终端
3152	KL3152 2 通道模拟输入终端
3158	KL3158 8 通道模拟输入终端
3162	KL3162 2 通道模拟输入终端
3172	KL3172 2 通道模拟输入终端
33557604	KL3172-0500 2 通道模拟输入终端
67112036	KL3172-1000 2 通道模拟输入终端
3182	KL3182 2 通道模拟输入终端
3201	KL3201 单通道模拟输入终端
3202	KL3202 2 通道模拟输入终端
3204	KL3204 4 通道模拟输入终端
33557640	KL3208-0010 8 通道模拟输入终端
3222	KL3222 2 通道模拟输入终端
3228	KL3228 8 通道模拟输入终端
3302	KL3302 2 通道模拟输入终端
3311	KL3311 单通道模拟输入终端
3312	KL3312 2 通道模拟输入终端
3314	KL3314 4 通道模拟输入终端
3351	KL3351 单通道电阻桥终端
50334999	KL3351-0001 单通道电阻桥终端
3356	KL3356 单通道精密电阻桥终端
3361	KL3361 单通道示波器终端
3362	KL3362 2 通道示波器终端
3403	KL3403 3 相电源测量终端
3404	KL3404 4 通道模拟输入终端
3408	KL3408 8 通道模拟输入终端
3444	KL3444 4 通道模拟输入终端
3448	KL3448 8 通道模拟输入终端
3454	KL3454 4 通道模拟输入终端
3458	KL3458 8 通道模拟输入终端

子类型	描述
3464	KL3464 4 通道模拟输入终端
3468	KL3468 8 通道模拟输入终端

5.3.3.5.4 ITcSmTreeItem 项目子类型：终端：K-Bus 模拟输出 (KL4)

子类型	描述
4001	KL4001 单通道模拟输出终端
4002	KL4002 2 通道模拟输出终端
4004	KL4004 4 通道模拟输出终端
4011	KL4011 单通道模拟输出终端
4012	KL4012 2 通道模拟输出终端
4021	KL4021 单通道模拟输出终端
4022	KL4022 2 通道模拟输出终端
4031	KL4031 单通道模拟输出终端
4032	KL4032 2 通道模拟输出终端
4034	KL4034 4 通道模拟输出终端
4112	KL4112 2 通道模拟输出终端
4122	KL4122 2 通道模拟输出终端
4132	KL4132 2 通道模拟输出终端
4404	KL4404 4 通道模拟输出终端
4408	KL4408 8 通道模拟输出终端
4414	KL4414 4 通道模拟输出终端
4418	KL4418 8 通道模拟输出终端
4424	KL4424 4 通道模拟输出终端
4428	KL4428 8 通道模拟输出终端
4434	KL4434 4 通道模拟输出终端
4438	KL4438 8 通道模拟输出终端
4494	KL4494 2 通道模拟输出终端

5.3.3.5.5 ITcSmTreeItem 项目子类型：终端：K-Bus 测量 (KL5)

子类型	描述
5001	KL5001 单通道 SSI 编码器终端
5051	KL5051 单通道 Bi-SSI 编码器终端
16782267	KL5051-0010 单通道 Bi-SSI 编码器终端
5101	KL5101 增量编码器 5V 终端
5111	KL5111 增量编码器 24V 终端
5121	KL5121 4 通道直线运动控制器终端
5151	KL5151 单通道增量编码器终端
33559583	KL5151-0021 单通道增量编码器终端
16782367	KL5151-0050 2 通道增量编码器终端
5152	KL5152 2 通道增量编码器终端

5.3.3.5.6 ITcSmTreeItem 项目子类型：终端：K-Bus 通信 (KL6)

子类型	描述
16783217	KL6001 接口 RS232C 终端
50337649	KL6001 接口 RS232 终端
16783227	KL6011 接口 TTY 终端
50337659	KL6011 接口 TTY 终端
16783237	KL6021 接口 RS422/485 终端
50337669	KL6021 接口 RS485 终端
100669317	KL6021 接口 RS485 终端
100669327	KL6031 接口 RS232 终端
50337679	KL6031 接口 RS232 终端
100669337	KL6041 接口 RS485 终端
50337689	KL6041 接口 RS485 终端
16783257	KL6041-0100 接口 RS485 终端
6051	KL6051 数据交换终端
335550521	KL6201 ASI-Master 终端
369104953	KL6201 ASI-Master 终端
402659385	KL6201 ASI-Master 终端
335550531	KL6211 ASI-Master 终端
369104963	KL6211 ASI-Master 终端
402659395	KL6211 ASI-Master 终端
6224	KL6224 I/O-Link Master 终端
16783440	KL6224 I/O-Link Master 终端
33560656	KL6224 I/O-Link Master 终端
50337872	KL6224 I/O-Link Master 终端
33560733	KL6301 EIB 终端
33560833	KL6401 LON 终端
33561013	KL6581 EnOcean 终端
33561203	KL6771 MP-Bus Master 终端
6781	KL6781 M-Bus 接口终端
6811	KL6811 DALI-Master 终端
6821	KL6821 eDALI-Master 终端

5.3.3.5.7 ITcSmTreeItem 项目子类型：终端：K-Bus 电源 (KL8)

子类型	描述
33562433	KL8001 单通道电源终端
8001	KL8001 3 通道电源终端
8519	KL8519 16 数字输入终端
8524	KL8524 2x4 数字输出终端
8528	KL8528 8 数字输出终端
8548	KL8548 8 通道模拟输出终端
8610	KL8610 单通道适配器终端 KL8601
16785826	KL8610 2 通道适配器终端 KL8601
33563042	KL8610 3 通道适配器终端 KL8601
50340258	KL8610 4 通道适配器终端 KL8601
67117474	KL8610 5 通道适配器终端 KL8601
83894690	KL8610 6 通道适配器终端 KL8601
100671906	KL8610 7 通道适配器终端 KL8601
117449122	KL8610 8 通道适配器终端 KL8601

5.3.3.5.8 ITcSmTreeItem 项目子类型：终端：K-Bus 系统 (KL9)

子类型	描述
9010	KL9010 线端终端
9020	KL9020 总线扩展线端终端
9050	KL9050 总线扩展耦合器终端
9060	KL9060 适配器终端
9070	KL9070 屏蔽终端
9080	KL9080 分离终端
9100	KL9100 供电器终端
9110	KL9110 供电器终端
9150	KL9150 供电器终端
9160	KL9160 供电器终端
9180	KL9180 电位连接终端
9181	KL9181 电位连接终端
9182	KL9182 电位连接终端
9183	KL9183 电位连接终端
9184	KL9184 电位连接终端
9185	KL9185 电位连接终端
9186	KL9186 电位连接终端
9187	KL9187 电位连接终端
9188	KL9188 电位连接终端
9189	KL9189 电位连接终端
9190	KL9190 馈电终端
9195	KL9195 屏蔽终端
9200	KL9200 供电器终端
9210	KL9210 供电器终端
9250	KL9250 供电器终端
9260	KL9260 供电器终端
9290	KL9290 供电器终端
9300	KL9300 4 通道二极管阵列终端
9301	KL9301 7 通道二极管阵列终端
9302	KL9302 7 通道二极管阵列终端
9309	KL9309 KL85xx 接口终端
9400	KL9400 K-Bus 供电器终端
9505	KL9505 供电器终端
167781665	KL9505-0010 供电器终端
9508	KL9508 供电器终端
167781668	KL9508-0010 供电器终端
9510	KL9510 供电器终端
167781670	KL9510-0010 供电器终端
9512	KL9512 供电器终端
167781672	KL9512-0010 供电器终端
9515	KL9515 供电器终端
167781675	KL9515-0010 供电器终端
9528	KL9528 供电器终端
9540	KL9540 电涌滤波器磁场供电终端
9550	KL9550 电涌滤波器系统和磁场供电终端
9560	KL9560 供电器终端
9570	KL9570 缓冲电容器终端

5.3.3.5.9 ITcSmTreeItem 项目子类型：终端：K-Bus 安全 (KLx90x)

子类型	描述
1904	KL1904 4 通道安全输入终端
1908	KL1908 8 通道安全输入终端
2904	KL2904 4 通道安全输出终端
6904	KL6904 安全逻辑 (7 个 TwinSAFE 接头) 终端
16784120	KL6904 安全逻辑 (15 个 TwinSAFE 接头) 终端

5.3.3.6 模块

5.3.3.6.1 ITcSmTreeItem 项目子类型：模块：K-Bus 数字输入 (KM1)

子类型	描述
838861802	KM1002 16 通道数字输入模块
838861804	KM1004 32 通道数字输入模块
838861808	KM1008 64 通道数字输入模块
838861812	KM1012 16 通道数字输入模块
838861814	KM1014 32 通道数字输入模块
838861818	KM1018 64 通道数字输入模块
838862444	KM1644 4 通道数字输入模块

5.3.3.6.2 ITcSmTreeItem 项目子类型：模块：K-Bus 数字输出 (KM2)

子类型	描述
838862802	KM2002 16 通道数字输出模块
838862804	KM2004 32 通道数字输出模块
838862808	KM2008 64 通道数字输出模块
838862822	KM2022 16 通道数字输出模块
838862842	KM2042 16 通道数字输出模块
838863404	KM2604 4 通道数字输出继电器模块
838863414	KM2614 4 通道数字输出继电器模块
838863442	KM2642 2 通道数字电源输出继电器模块
838863452	KM2652 2 通道数字电源输出继电器模块
16779990	KM2774 4 通道盲源控制终端
2774	KM2774-1001 4 通道盲源控制终端

5.3.3.6.3 ITcSmTreeItem 项目子类型：模块：K-Bus 模拟输入 (KM3)

子类型	描述
838864501	KM3701 单通道差压测量
872418933	KM3701-0340 单通道差压测量
838864502	KM3702 2 通道绝对压力测量
838864512	KM3712 2 通道绝对压力测量

5.3.3.6.4 ITcSmTreeItem 项目子类型：模块：K-Bus 模拟输出 (KM4)

子类型	描述
838865402	KM4602 2 通道模拟输出终端

5.3.3.6.5 ITcSmTreeItem 项目子类型：模块：K-Bus 通信 (KM6)

子类型	描述
6551	KM6551 IEEE802.15.4 终端
838867463	KM6663 以太网转换开关终端

5.3.4 ITcSmTreeItem::ProduceXml

ProduceXml() 方法返回一个 XML 字符串，包含项目特定信息和参数。

```
HRESULT ProduceXml(VARIANT_BOOLbRecursive, BSTR* pXML);
```

Parameters (参数)

bRecursive	[in, defaultvalue(0)] 供以后使用的可选参数。
pXML	[out, retval] 包含项目特定数据和参数的 XML 描述。

返回值

S_OK	函数成功返回。
E_POINTER	pXML 指针无效。

注释

如果树项是类型 *SERCOS Master/Slave FC750x* 的 I/O 设备，则以下 XML 字符串是生成信息的不完整示例。此字符串可以用作 IXMLDOMDocument::loadXML 方法的输入，以创建 XML DOM 文档。

```
<?xml version="1.0"?>
<TreeItem>
<ItemName>Device 1 (FC750x)</ItemName>
<PathName>TIID^Device 1 (FC750x)</PathName>
<ItemType>2</ItemType>
<ItemId>1</ItemId>
<ItemSubType>48</ItemSubType>
<ItemSubTypeName>SERCOS Master/Slave FC750x, PCI [Beckhoff FC7502 PCI]</ItemSubTypeName>
<ChildCount>0</ChildCount>
<Disabled>0</Disabled>
<DeviceDef>
<AmsPort>0</AmsPort>
<AddressInfo>
<Pci>
<BusNo>0</BusNo>
<SlotNo>16</SlotNo>
<IrqNo>9</IrqNo>
<FcChannel>0</FcChannel>
</Pci>
</AddressInfo>
<SercosMaster>
<Baudrate>0</Baudrate>
<OperationMode>0</OperationMode>
<SendPower>1</SendPower>
<AccessTime>200</AccessTime>
<ShiftTime>50</ShiftTime>
<StartupToPhase4>1</StartupToPhase4>
<CheckTiming>1</CheckTiming>
</SercosMaster>
</DeviceDef>
</TreeItem>
```

参见

ITcSmTreeItem::ConsumeXML [▶ 135]

5.3.5 ITcSmTreeItem::ConsumeXml

ITcSmTreeItem [▶ 103]

ConsumeXml() 方法使用一个 BSTR，其中包含项目特定数据的 XML 描述，并更新发现的参数。此方法用于更改 ITcSmTreeItem [▶ 103] 接口无法直接访问的项目参数。

```
HRESULT ConsumeXml(BSTRbstrXML);
```

Parameters (参数)

bstrXML	[in] 包含项目特定参数的 XML 描述的字符串。相应参数将在系统管理器数据库中进行更新

返回值

S_OK	函数成功返回。
E_FAIL	*bstrXML* 字符串不包含有效 xml 文档。

注释

文档仅可包含应更改的特定参数。文档结构必须匹配项目特定的 XML 树形结构，但可以忽略不应更改的参数。以下文档是一个可用于更改项目特定参数 *CheckNumberBoxes* 的最小示例（本示例中为 C1220 现场总线卡）。如果文档中的参数对于项目是未知的，则将被忽略。

<TreeItem><DeviceDef><DevC1220Def><CheckNumberBoxes>0</CheckNumberBoxes></DevC1220Def></DeviceDef></TreeItem>

特定树项的参数集在 TwinCAT 系统管理器附带的 xml 格式文档中定义。也可以调用 ITcSmTreeItem::ProduceXML [▶ 134] 方法评估特定项目的参数。生成的 xml 字符串包含此项目的所有参数，但并非所有参数都可以更改。xml 字符串可能包含在 xml 格式下有效的任意数量和层级顺序的 xml 元素。允许一次只更改一个参数（如以上示例所示）、同时更改一个参数集或传递 ITcSmTreeItem::ProduceXML [▶ 134] 返回的完整参数集（通常更改相同的参数）。

有些特定 xml 元素与参数不对应，这些元素将“执行”一项功能。以 PLC 项目树项的 <Rescan> 元素为例进行说明。字符串

<TreeItem><PlcDef><ReScan>1</ReScan></PlcDef></TreeItem>

（作为 **ConsumeXml** 的参数）将使系统管理器重新扫描 PLC 项目（如同通过按下 PLC 项目上的“Rescan”（重新扫描）按钮进行手动重新扫描）。可用的参数和功能记录在 xml 格式文件中。

参见

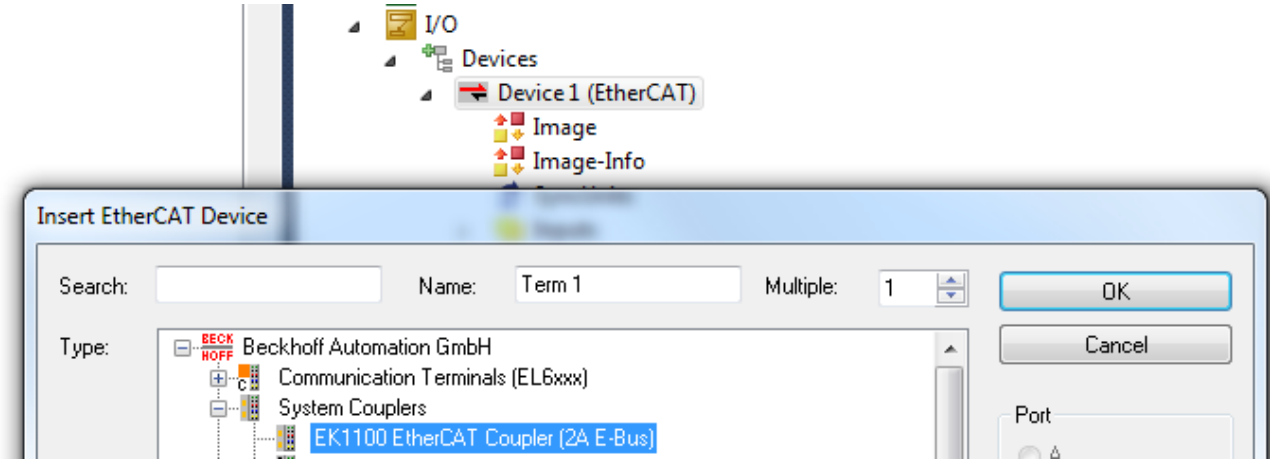
ITcSmTreeItem::ProduceXML [▶ 134]

5.3.6 ITcSmTreeItem::CreateChild

在父节点上创建子项。子项由其子类型 [▶ 108]指定。

```
HRESULT CreateChild(BSTRbstrName,long nSubType, BSTRbstrBefore, VARIANT vInfo,
ITcSmTreeItem**pipItem);
```

以下示例更详细地说明此特性。



在本示例中，为 TwinCAT XAE 中的 EtherCAT Master 设备（设备 1）添加了一个 EK1100 耦合器。在自动化接口中，这可以通过 CreateChild() 方法实现。父节点（设备 1 EtherCAT）具有项目类型“2”（TREEITEMTYPE_DEVICE）。子类型指定子项，因此 EK1100 是子类型“9099”（BOXTYPE_EXXXXX）。

Parameters (参数)

bstrName	[in] 新子项的项目名。
nSubtype:	[in] [▶ 108]新子项的子类型 [▶ 108]。可用的子类型取决于父树项的项目类型 [▶ 105] (类别)。例如，PLC Functionblock 只可添加到项目类型 PLCFOLDER，不能添加到 DEVICE。
bstrBefore	[in, defaultvalue("")] 如果已设置，则参数包含另一子项的名称 (在此之前应插入新项目)。
vInfo	[in, optional] 含附加创建信息的可选参数，取决于子类型 [▶ 108]。下表列出了不同的依赖项。
pipItem	[out, retval] 指向接收结果的 ITcSmTreeItem [▶ 103] 接口指针的位置。

返回值

S_OK	函数成功返回。
E_POINTER	返回指针的位置无效
TSM_E_INVALIDITEMSUBTYPE (0x98510003)	指定子类型无效。
TSM_E_ITEMNOTFOUND (0x98510001)	未找到 *bstrBefore* 项。

可选 vInfo 参数

根据新子项的项目子类型，创建子项时可能需要一些其它信息。此信息可以通过 *vInfo* 参数提供。

子功能元素：Element-Subtyp	vInfo 参数
E-Bus Box / Klemme / Modul (Element-Subtyp 9099)	包含 EtherCAT 接线盒的标识对象 (CoE 1018h、VendorId、ProductCode 以及可选的 RevisionNo 和 SerialNo)。变量类型必须为 LONG 数组 (VT_I4|VT_ARRAY，2-4 个元素)。或者包含以下格式的 BSTR (其中 %X = 十六进制值，例如“V00000002_P044c2c52_R00000000”)： 特别是对于 Beckhoff E-Bus 终端/模块，请阅读相应的 E-Bus 章节 [▶ 109]。
Interbus 接线盒。(Element-Subtyp 2002)	包含 Interbus 接线盒的 IdentCode 和 LengthCode。变量类型必须为无符号的短格式 (VT_I2)，低字节包含 IdentCode，高字节包含 LengthCode。
AX2000 (Element-Subtyp 5007) CANDrive (Element-Subtyp 5006)	(可选) 包含 bool 值 (VT_BOOL)，如果已设置，将创建“Following Error” (以下错误) 的附加变量。
DeviceNET (Element-Subtyp 5203) TcDNSSlave (Element-Subtyp 5250) CX1500 (Element-Subtyp 1042) FC520X Slave (Element-Subtyp 1043)	(可选) 包含 EDS 文件的文件路径 (VT_BSTR)。
BK3000 und alle übrigen PROFIBUS-Box-Typen	(可选) 包含 GSD 文件的文件路径 (VT_BSTR)。
Variable	(可选) 包含新变量的位地址。变量类型可为 SHORT 或 LONG (VT_I2 或 VT_I4)。
SPS-POU Funktionsblock (Element-Subtyp 604)	包含 IEC 语言类型 Integer，由 IECLANGUAGETYPES [▶ 141] 定义。
SPS-POU Funktion (Element-Subtyp 603)	包含字符串 [2] 数组，保留以下值： • array[0] = IEC 语言类型 Integer，由 IECLANGUAGETYPES [▶ 141] 定义。 • array[1] = 功能返回数据类型。可以是任何 PLC 数据类型，例如 DINT、BOOL...

还请参阅有关此

ITcSmTreeItem [▶ 103]

5.3.7 ITcSmTreeItem::DeleteChild

删除一个子项。

```
HRESULT DeleteChild(BSTRbstrName);
```

Parameters (参数)

bstrName	[in] 应删除的子项的项目名。

返回值

S_OK	函数成功返回。
E_ACCESSDENIED	不允许删除子项。
TSM_E_ITEMNOTFOUND (0x98510001)	未找到 *bstrBefore* 项。

还请参阅有关此

ITcSmTreeItem [▶ 103]

5.3.8 ITcSmTreeItem::ImportChild

从剪贴板或先前导出的文件中导入子项。

```
HRESULT ImportChild(BSTRbstrFile, BSTRbstrBefore, VARIANT_BOOLbReconnect, BSTRbstrName,
ITcSmTreeItem**pipItem);
```

Parameters (参数)

bstrFile	[in, defaultvalue(L"")] 要从中导入新子项的文件的文件名。如果未指定文件名 (空字符串)，则将从剪贴板导入子项。
bstrBefore	[in, defaultvalue(L"")] 如果已设置，则此参数包含另一子项的名称 (在此之前应插入新项目)。如果未设置，子项将附加在末尾。
bReconnect	[in, defaultvalue(VARIANT_TRUE)] 一个可选标签，指示系统管理器尝试将导入项中的变量重新连接到组态中的其它变量 (按名称)。
bstrName	[in, defaultvalue(L"")] 如果已设置，则使用导入文件中的子项名称覆盖该子项名称。
pipItem	[out, retval] 指向接收结果的 ITcSmTreeItem [▶ 103] 接口指针的位置。

返回值

S_OK	函数成功返回。
NTE_NOT_FOUND (0x80090011)	无法找到/打开文件。
NTE_BAD_SIGNATURE (0x80090006)	文件不包含有效树项。
TSM_E_MISMATCHINGITEMS (0x98510004)	文件中的项不是有效的子项。

还请参阅有关此

ITcSmTreeItem [▶ 103]

5.3.9 ITcSmTreeItem::ExportChild

将子项导出至剪贴板或文件中。

```
HRESULT ExportChild(BSTRbstrName, BSTRbstrFile);
```

Parameters (参数)

bstrName	[in] 要导出子项的名称。
bstrFile	[in, defaultvalue("")] 子项将导出到的文件的文件名。如果未指定文件名 (空字符串)，则子项将导出到剪贴板。

返回值

S_OK	函数成功返回。
TSM_E_ITEMNOTFOUND (0x98510001)	未找到 *bstrName* 项。

还请参阅有关此

ITcSmTreeItem [▶ 103]

5.3.10 ITcSmTreeItem::LookupChild

返回由其相对路径名指定的子代树项的 *ITcTreeItem* 指针。

```
HRESULT LookupChild(BSTRbstrName, ITcSmTreeItem**pipItem);
```

Parameters (参数)

bstrName	[in] 要查找树项的相对路径。需要相对路径名，各分支之间必须用扬抑符 "^" 或制表符分隔。
pipItem	[out, retval] 指向返回的 ITcSmTreeItem [▶ 103] 接口指针的位置。接口指针允许访问属于树项的特定方法。

返回值

S_OK	函数成功返回。
TSM_E_ITEMNOTFOUND (0x98510001)	路径名不限定现有树项。

还请参阅有关此

ITcSmTreeItem [▶ 103]

5.3.11 ITcSmTreeItem::GetLastXmlError

获得最后一条错误的 ConsumeXml [▶ 135] 调用的项目路径/错误消息。

```
HRESULT GetLastXmlError(BSTR *pXML);
```

Parameters (参数)

pXML	[out, retval] 错误消息。

返回值

S_OK	函数成功返回。

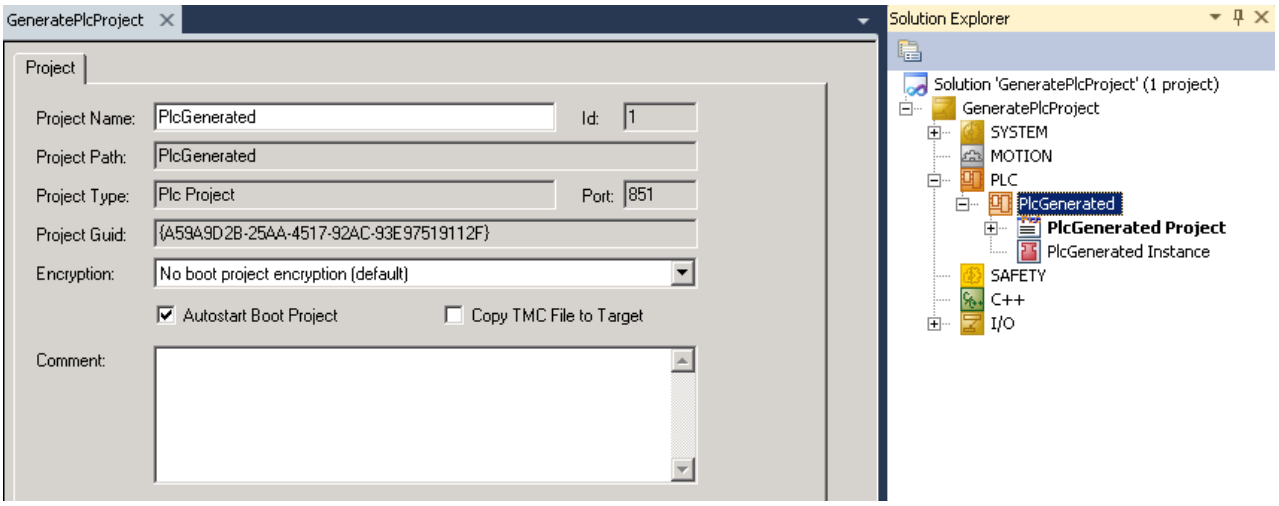
还请参阅有关此

ITcSmTreeItem [▶ 103]

5.4 ITcPlcProject

5.4.1 ITcPlcProject

ITcPlcProject 类使开发人员能够为 PLC 项目设置属性。它通常以 PLC 项目的根节点为目标，如下图所示。



以下 C# 代码片段显示有关如何在自动化接口代码中使用该类的示例：

```
ITcSmTreeItem plcProjectRootItem = systemManager.LookupTreeItem("TIPC^PlcGenerated");
ITcPlcProject iecProjectRoot = (ITcPlcProject)plcProjectRootItem;
iecProjectRoot.BootProjectAutostart = true;
iecProjectRoot.GenerateBootProject (true);
```

请注意：如果要**编译**一个 PLC 项目，请使用 Visual Studio COM objectEnvDTE 的编译器功能，如多个示例 [▶ 158]部分中所示。

方法

ITcPlcProject 方法	描述	可用版本
GenerateBootProject() [▶ 141]	相当于 TwinCAT XAE 上下文菜单中的条目“Activate Boot project” (激活 Boot 项目)	TwinCAT 3.1

属性

ITcPlcProject 属性	获取/设置	描述	可用版本
BootProjectAutoStart	是 / 是	相当于上面所示对话框中的复选框“Autostart Boot Project”(自动启动 Boot 项目)	TwinCAT 3.1
BootProjectEncryption	是 / 是	相当于上面所示对话框中的下拉框“Encryption”(加密)	TwinCAT 3.1
TmcFileCopy	是 / 是	相当于上面所示对话框中的复选框“Copy TMC File to Target”(将 TMC 文件复制到目标)	TwinCAT 3.1

版本信息

所需 TwinCAT 版本
TwinCAT 3.1 及以上版本支持此接口

5.4.2 ITcPlcProject::GenerateBootProject

根据指定的 bool 参数，激活或禁用作为 boot 项目的 PLC 项目。

```
HRESULT GenerateBootProject(VARIANT_BOOL bActivate);
```

Parameters (参数)

bActivate	[in, optional, defaultvalue(-1)] 指定是否应激活 boot 项目

5.5 ITcPlcPou

5.5.1 ITcPlcPou

要在 TwinCAT 3 项目中处理 POU 及其内容，可以使用接口 ITcPlcPou、ITcPlcDeclaration [▶ 142] 和 ITcPlcImplementation [▶ 142]。例如，如果要为 PLC 项目创建一个新功能块并填充代码，可以使用这些接口来完成。有关详细信息，请参见有关“如何访问和创建 PLC 对象”的最佳方案章节。

属性

ITcPlcPou 属性	获取/设置	描述	可用版本
DocumentXml	是 / 是	以 XML 格式获取/设置 POU 的 PLC 代码 (非 PLCopen XML)	TwinCAT 3.1
ReturnType	是 / 否	POU 的返回类型，例如函数的返回类型。可以是 PLC 已知的任何数据类型，例如 BOOL、DINT...。通过 CreateChild() [▶ 136] 方法创建 POU 时会设置返回类型。	TwinCAT 3.1

版本信息

要求

所需 TwinCAT 版本
TwinCAT 3.1 及以上版本支持此接口

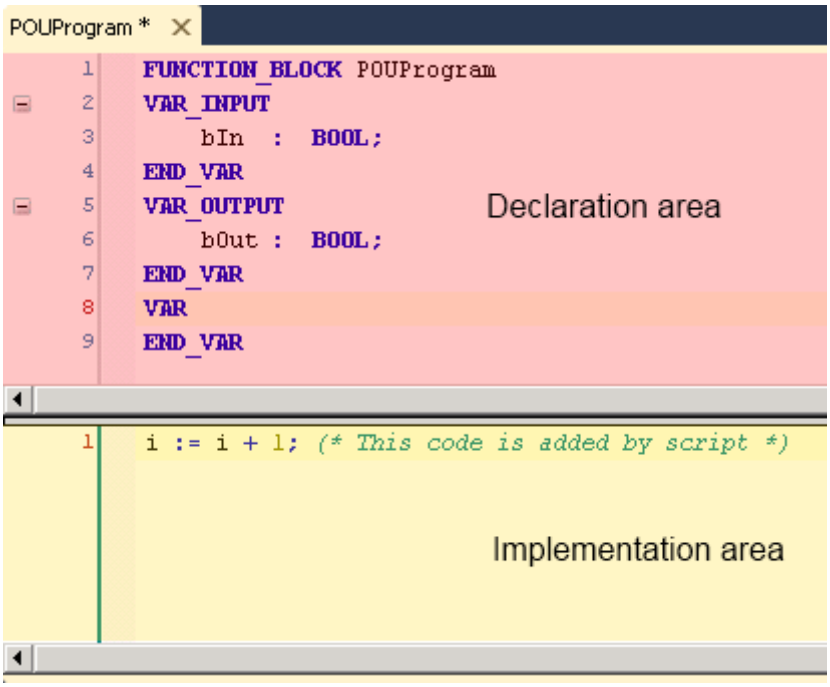
5.5.2 IECLanguageTypes

枚举 IECLanguageTypes 根据 IEC 标准定义不同的编程语言，并可在通过 ITcSmTreeItem [▶ 103]::CreateChild() [▶ 136] 方法创建新的 POU 时使用。

条目	Value	描述
IECLANGUAGE_NONE	0	---
IECLANGUAGE_ST	1	结构化文本
IECLANGUAGE_IL	2	指令表
IECLANGUAGE_SFC	3	顺序功能图表
IECLANGUAGE_FBD	4	功能块图
IECLANGUAGE_CFC	5	连续功能图表
IECLANGUAGE_LD	6	梯形图

5.6 ITcPlcDeclaration

ITcPlcDeclaration 接口可访问 PLC POU 及其子节点（例如动作、方法...）的声明区域。有关如何使用此接口的详细信息，另请参见“处理 PLC 对象”的最佳方案章节。



属性（按 Vtable 顺序）

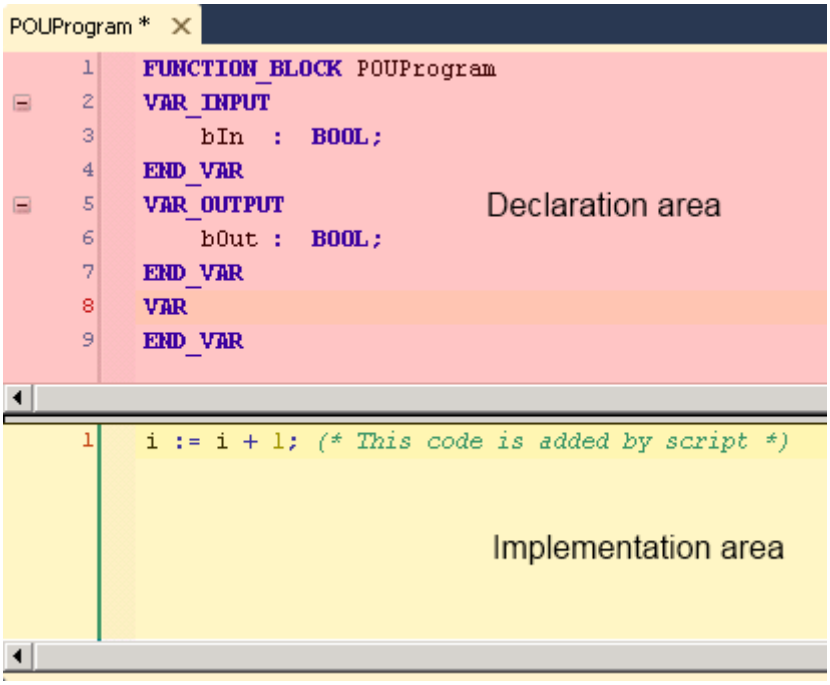
ITcPlcDeclaration 属性	获取/设置	描述	可用版本
DeclarationText	是 / 是	表示项目的声明区域，并以明文形式获取/设置其内容	TwinCAT 3.1

版本信息

所需 TwinCAT 版本
TwinCAT 3.1 及以上版本支持此接口

5.7 ITcPlcImplementation

ITcPlcImplementation 接口可访问 PLC POU 及其子节点（例如动作、方法...）的实现区域。有关如何使用此接口的详细信息，另请参见有关“处理 PLC 对象”的最佳方案章节。



属性（按 Vtable 顺序）

ITcPlcImplementation 属性	获取/设置	描述	可用版本
ImplementationText	是 / 是	表示项目的声明区域，并以明文形式获取/设置其内容	TwinCAT 3.1
ImplementationXml	是 / 是	以 XML 格式获取/设置内容（**非** PLCopen XML）	TwinCAT 3.1
语言	是 / 否	获取实现区域使用的 IEC 语言	TwinCAT 3.1

版本信息

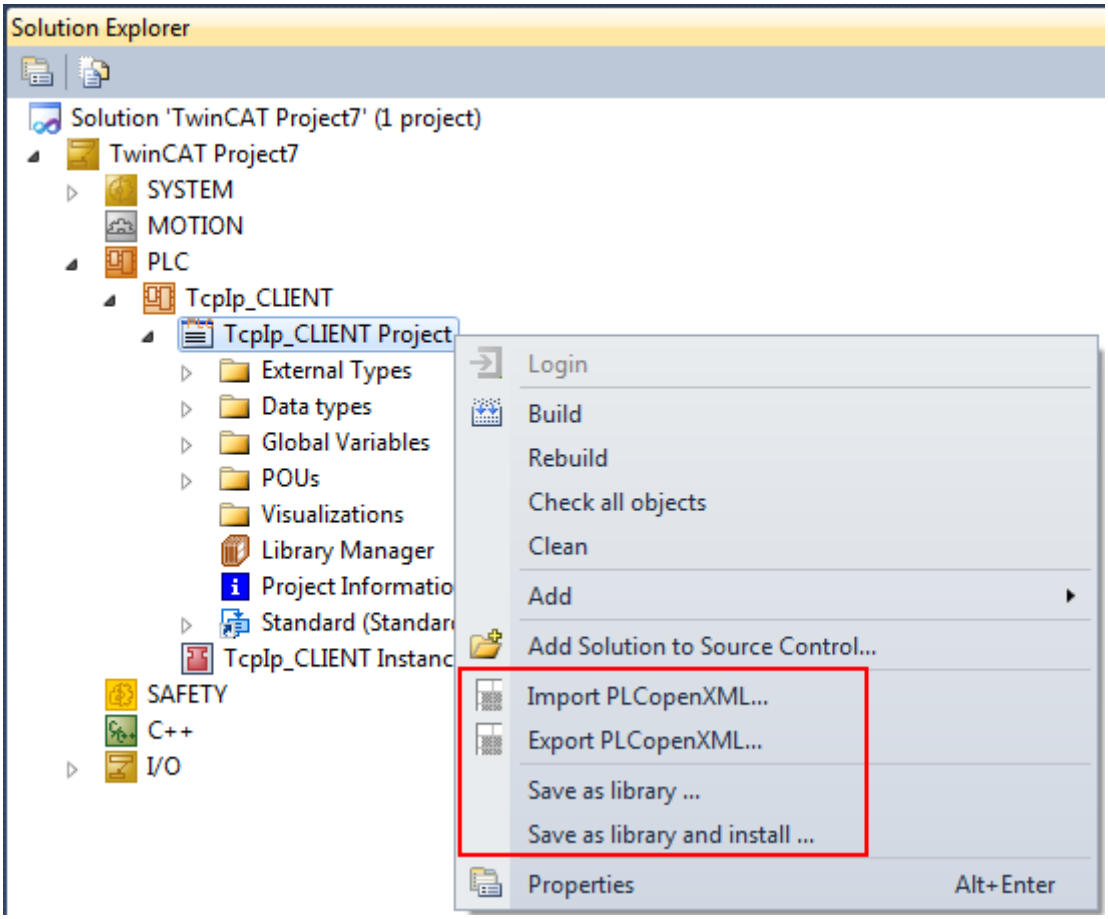
要求

所需 TwinCAT 版本
TwinCAT 3.1 及以上版本支持此接口

5.8 ITcPlcIECProject

5.8.1 ITcPlcIECProject

ITcPlcIECProject 接口提供根据 PlcOpen XML 标准导入和导出 PLC 项目以及将 PLC 项目另存为 PLC 库的方法。与 TwinCAT XAE GUI 相比，此接口表示以下四个选项：



方法（按 Vtable 顺序）

ITcPlcIECProject 方法	描述	可用版本
PlcOpenExport() [▶ 145]	将指定的树节点及其内容导出到符合 PlcOpen 的 XML 文件中	TwinCAT 3.1
PlcOpenImport() [▶ 145]	导入符合 PlcOpen 的 XML 文件及其内容	TwinCAT 3.1
SaveAsLibrary() [▶ 146]	将选定的 PLC 项目另存为 PLC 库，也可以选择对其进行安装。	TwinCAT 3.1

版本信息

要求

所需 TwinCAT 版本
TwinCAT 3.1 及以上版本支持此接口

5.8.2 PlcImportOptions

此枚举包含以下选项，并在通过 ITcPlcIECProject [▶ 143]::PlcOpenImport() [▶ 145] 方法导入符合 PlcOpen 的 XML 文件时使用。

元素	描述	支持版本
PLCIMPORTOPTIONS_NONE	---	TwinCAT 3.1
PLCIMPORTOPTIONS_RENAME	如果导入项已存在于 PLC 项目中，则会自动重命名	TwinCAT 3.1
PLCIMPORTOPTIONS_REPLACE	如果导入项已存在于 PLC 项目中，则替换现有项目	TwinCAT 3.1
PLCIMPORTOPTIONS_SKIP	如果导入项已存在于 PLC 项目中，则跳过	TwinCAT 3.1

5.8.3 ITcPlcIECProject::PlcOpenExport

将指定的树节点及其内容导出到符合 PlcOpen 的 XML 文件中。XML文件和树节点的路径将在该方法的参数中指定。

```
HRESULT PlcOpenExport(BSTR bstrFile, BSTR bstrSelection);
```

Parameters (参数)

bStrFile

[in]：XML 文件的路径

bstrSelection

[in]：选择应导出的树项。各项用分号分隔，例如 z.B。“POUs.FBItem1;POUs.FBItem2”

返回值

S_OK	项目已成功导出到 PlcOpen XML 文件中。

5.8.4 ITcPlcIECProject::PlcOpenImport

导入符合 PlcOpen 的 XML 文件及其内容。XML 文件的路径将通过参数传递给方法。

```
HRESULT PlcOpenImport(BSTR bstrFile, int options, BSTR bstrSelection, VARIANT_BOOL
bFolderStructure);
```

Parameters (参数)

bstrFile

[in]：XML 文件的路径。

options

[in, optional, defaultvalue(0)]：导入选项，取决于枚举 _PLCIMPORTOPTIONS [▶ 144]。

bstrSelection

[in, optional, defaultvalue("")]：选择应从 XML 结构中导入哪些项

bFolderStructure

[in, optional, defaultvalue(-1)]：如果设置为 true (-1)，则导入对象的文件夹结构保持不变 (如果 XML 文件中提供了此信息)

返回值

S_OK	PlcOpen XML 文件已成功导入。

还请参阅有关此

ITcPlcIECProject [▶ 143]

5.8.5 ITcPlcIECProject::SaveAsLibrary

将 PLC 项目另存为 PLC 库，也可以选择对其进行安装。

```
HRESULT SaveAsLibrary(BSTR bstrLibraryPath, VARIANT_BOOL binstall);
```

Parameters (参数)

bStrLibraryPath

[in]：应将 PLC 库保存到的位置的路径

binstall

[in]：如果还应安装 PLC 库，则设置为“true”

返回值

S_OK	PLC 项目成功保存为 PLC 库。

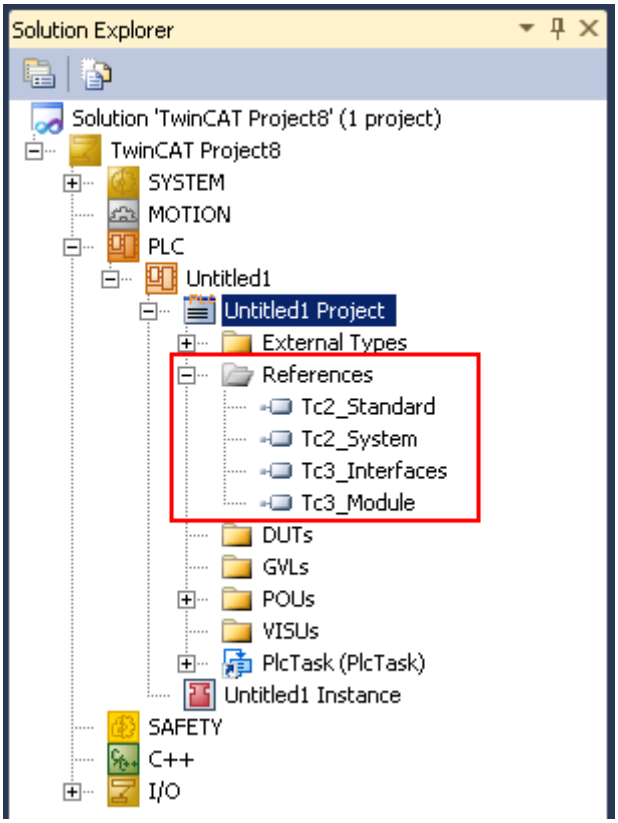
还请参阅有关此

ITcPlcIECProject [▶ 143]

5.9 ITcPlcLibraryManager

5.9.1 ITcPlcLibraryManager

ITcPlcLibraryManager 接口通过访问 TwinCAT 3 PLC 项目的 PLC 库或 TwinCAT 系统的 PLC 存储库来扩展自动化接口。



方法（按 Vtable 顺序）

ITcPlcLibraryManager 方法	描述	可用版本
AddLibrary() [▶ 148]	将库添加到 PLC 项目中。	TwinCAT 3.1
AddPlaceholder() [▶ 149]	将占位符添加到 PLC 项目中	TwinCAT 3.1
InsertRepository() [▶ 149]	创建存储库，此存储库表示多个库的逻辑容器。	TwinCAT 3.1
InstallLibrary() [▶ 149]	将库安装到存储库中。	TwinCAT 3.1
MoveRepository() [▶ 150]	更改存储库在存储库位置列表中的位置。	TwinCAT 3.1
RemoveReference() [▶ 150]	从 PLC 项目中删除库。	TwinCAT 3.1
RemoveRepository() [▶ 150]	删除存储库。	TwinCAT 3.1
ScanLibraries() [▶ 151]	返回在系统中找到的**所有**库的列表。返回对象为类型 ITcPlcLibraries [▶ 153]，是 ITcPlcLibrary [▶ 153] 对象的集合。	TwinCAT 3.1
SetEffectiveResolution() [▶ 151]	设置占位符的有效分辨率	TwinCAT 3.1
UninstallLibrary() [▶ 152]	从存储库中卸载库。	TwinCAT 3.1

属性（按 Vtable 顺序）

ITcPlcLibraryManager 属性	获取/设置	描述	可用版本
引用	是 / 否	获取类型 ITcPlcReferences [▶ 153] 的对象，该对象是 ITcPlcLibrary [▶ 153] 或 ITcPlcPlaceholderRef 对象的集合。表示已添加到 PLC 项目的所有引用的列表。	TwinCAT 3.1
存储库	是 / 否	获取类型 ITcPlcLibRepositories [▶ 155] 的对象，该对象是 ITcPlcLibRepository [▶ 155] 对象的集合。表示所有当前已组态存储库的列表。	TwinCAT 3.1

版本信息

要求

所需 TwinCAT 版本
TwinCAT 3.1 及以上版本支持此接口

5.9.2 ITcPlcLibraryManager::AddLibrary

将库添加到 PLC 项目中。库可以通过其属性（名称、版本、公司）或显示文本添加。

```
HRESULT AddLibrary(
BSTR bstrLibName,
BSTR bstrVersion,
BSTR bstrCompany
);
```

```
HRESULT AddLibrary(BSTR bstrLibName, BSTR bstrVersion, BSTR bstrCompany);
```

Parameters（参数）

bstrLibName	[in] 库名。
bstrVersion	[in, optional, defaultvalue("")] 库版本号。
bstrCompany	[in, optional, defaultvalue("")] 创建库的公司。

返回值

S_OK	已成功添加库。

注释

添加 PLC 库的两种常见方法包括：

- libraryManager.AddLibrary("Tc2_MDP", "3.2.0.0", "Beckhoff Automation GmbH"); // 通过其属性添加库
- libraryManager.AddLibrary("Tc2_MDP, 3.2.0.0 (Beckhoff Automation GmbH)"); // 通过其显示名称添加库

其中 librayManager 是类型 ITcPlcLibraryManager 的对象。

5.9.3 ITcPlcLibraryManager::AddPlaceholder

将占位符添加至 PLC 项目。占位符可以通过其属性（占位符名称、库名、库版本号、库分发服务器）或其名称（如果占位符已存在）添加。

```
HRESULT AddPlaceholder(BSTR bstrPlaceholderName, BSTR bstrDefaultLibName, BSTR bstrDefaultVersion,
BSTR bstrDefaultDistributor);
```

Parameters（参数）

bstrPlaceholderName	[in] 占位符名称。
bstrDefaultLibName	[in] 占位符指向的默认库名。
bstrVersion	[in, optional, defaultvalue("")] 默认库版本号。
bstrCompany	[in, optional, defaultvalue("")] 创建库的公司。

返回值

S_OK	已成功添加占位符。

还请参阅有关此

ITcPlcLibraryManager [▶ 146]

5.9.4 ITcPlcLibraryManager::InsertRepository

在指定位置添加新的库存储库。此位置表示存储库在 TwinCAT 3 中存储库列表中的位置的索引。除了索引之外，存储库还可以通过其名称和文件系统中目录的路径来标识。

```
HRESULT InsertRepository(BSTR bstrName, BSTR rootFolder, int iIndex);
```

Parameters（参数）

bstrName	[in] 存储库名称。
rootFolder	[in] 存储库路径（文件系统）。
iIndex	[in] 指示存储库在存储库列表中的位置。

返回值

S_OK	已成功插入存储库

还请参阅有关此

ITcPlcLibraryManager [▶ 146]

5.9.5 ITcPlcLibraryManager::InstallLibrary

将库安装到现有库存储库中。

```
HRESULT InstallLibrary(BSTR bstrRepositoryName, BSTR bstrLibPath, VARIANT_BOOL bOverwrite);
```

Parameters（参数）

bstrRepositoryName	[in]：应插入库的存储库名称
bstrLibPath	[in]：库路径
bOverwrite	[in]：如果已经存在另一个库，则将此参数设置为对其进行覆盖

返回值

S_OK	库安装成功。

还请参阅有关此

ITcPlcLibraryManager [▶ 146]

5.9.6 ITcPlcLibraryManager::MoveRepository

将存储库移至存储库列表中的新位置。此位置由其索引进行标记，从 0 开始。

```
HRESULT MoveRepository(BSTR bstrRepositoryName, int iNewIndex);
```

Parameters (参数)

bstrRepositoryName	[in]: 存储库名称
iNewIndex	[in]: 存储库索引

还请参阅有关此

ITcPlcLibraryManager [▶ 146]

5.9.7 ITcPlcLibraryManager::RemoveReference

从实际 PLC 项目中删除引用。引用可以是库，也可以是占位符。

请注意：如果是，这只会从存储库中删除引用，而不删除实际库文件。为此，需要使用 UninstallLibrary() [▶ 152] 方法。

与 AddLibrary() 方法相似，可以通过指定库的属性（名称、版本号、公司）或显示文本来删除库。

```
HRESULT RemoveReference(BSTR bstrLibName, BSTR bstrVersion, BSTR bstrCompany);
```

Parameters (参数)

bstrLibName	[in]: 库名
bstrVersion	[in, optional, defaultvalue("")]:
bstrCompany	[in, optional, defaultvalue("")]

注释

删除 PLC 库的两种常见方法包括：

- libraryManager.RemoveReference("Tc2_MDP", "3.2.0.0", "Beckhoff Automation GmbH"); // 通过其属性删除库
- libraryManager.RemoveReference("Tc2_MDP, 3.2.0.0 (Beckhoff Automation GmbH)"); // 通过其显示名称删除库

其中 librayManager 是类型 ITcPlcLibraryManager 的对象。

还请参阅有关此

ITcPlcLibraryManager [▶ 146]

5.9.8 ITcPlcLibraryManager::RemoveRepository

删除库存储库。存储库由其唯一名称指定。

```
HRESULT RemoveRepository(BSTR bstrName);
```

Parameters (参数)

bstrName

[in] 存储库名称。

还请参阅有关此

ITcPlcLibraryManager [▶ 146]

5.9.9 ITcPlcLibraryManager::ScanLibraries

返回所有存储库中所有已注册库的集合。

```
HRESULT ScanLibraries(ITcPlcLibraries** ppEnumLibraries);
```

Parameters (参数)

ppEnumLibraries	[out, retval] 返回类型 ITcPlcLibraries [▶ 153] 的对象，这是 ITcPlcLibrary [▶ 153] 对象的集合。

注释

此方法提供的功能与通过树节点“库管理器”上的 ConsumeXml() [▶ 135] 导入以下 XML 结构相同：

```
<TreeItem>
<PlcLibDef>
    <ScanLibraries>
     <Active>true</Active>
    </ScanLibraries>
</PlcLibDef>
</TreeItem>
```

还请参阅有关此

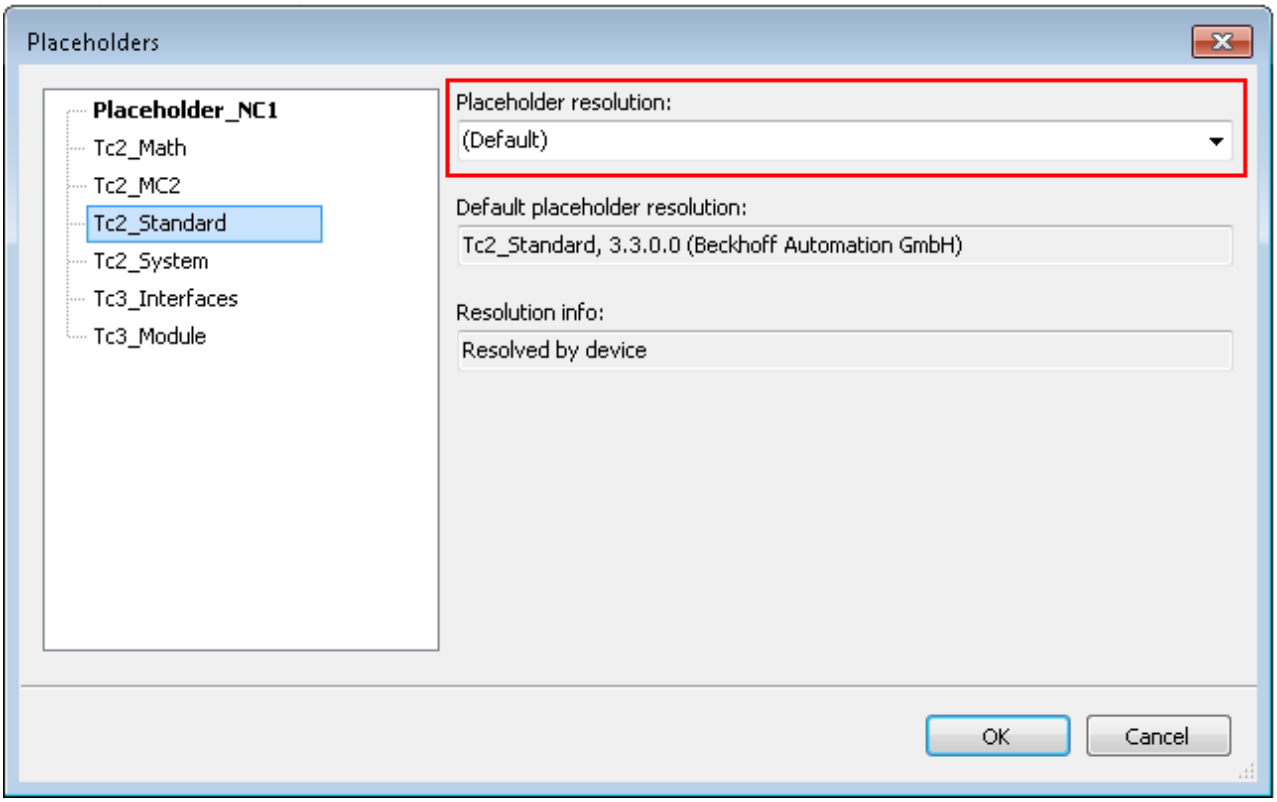
ITcPlcLibraryManager [▶ 146]

5.9.10 ITcPlcLibraryManager::SetEffectiveResolution

设置占位符的有效分辨率，即有效库。

```
HRESULT SetEffectiveResolution(BSTR bstrPlaceholderName, BSTR strLibName, BSTR bstrVersion, BSTR
bstrDistributor);
```

在 TwinCAT XAE 中，可以通过占位符组态窗口设置有效分辨率。



请注意：通过 ITcPlcLibraryManager [▶ 146]::AddPlaceholder() [▶ 149] 添加占位符时，会设置占位符的默认分辨率。

Parameters (参数)

bstrPlaceholderName	[in] 应设置有效分辨率的占位符名称。
bstrLibName	[in] 有效分辨率的库名。
bstrVersion	[in, optional, defaultvalue("")] 库版本号。
bstrDistributor	[in, optional, defaultvalue("")] 创建库的公司。

返回值

S_OK	已成功设置有效分辨率。

5.9.11 ITcPlcLibraryManager::UninstallLibrary

从存储库中卸载库。

```
HRESULT UninstallLibrary(BSTR bstrRepositoryName, BSTR bstrLibraryName, BSTR bstrVersion, BSTR bstrDistributor);
```

Parameters (参数)

bstrRepositoryName	[in]：存储库名称
bstrLibraryName	[in]：库名
bstrVersion	[in, optional]：库版本号
bstrDistributor	[in, optional]：库分发服务器

还请参阅有关此

ITcPlcLibraryManager [▶ 146]

5.10 ITcPlcReferences

ITcPlcReferences 表示 ITcPlcLibRef [▶ 154] 对象的集合，例如，在使用 ITcPlcLibraryManager [▶ 146]::References 属性时返回该集合。

方法（按 Vtable 顺序）

要求

ITcPlcLibRepositories 方法	描述	可用版本
get_Item() [▶ 156]	返回位于指定位置的类型 ITcPlcLibRef [▶ 154] 的项。	TwinCAT 3.1

属性（按 Vtable 顺序）

ITcPlcLibRepositories 属性	获取/设置	描述	可用版本
计数	是 / 否	返回集合中 ITcPlcLibRef [▶ 154] 对象的数量	TwinCAT 3.1

版本信息

所需 TwinCAT 版本
TwinCAT 3.1 及以上版本支持此接口

5.11 ITcPlcLibrary

表示与 ITcPlcLibraries [▶ 153] 集合和 ITcPlcLibraryManager [▶ 146]::ScanLibraries() [▶ 151] 方法或 ITcPlcLibraryManager [▶ 146]::References 属性一起使用时的单个 PLC 库。

属性（按 Vtable 顺序）

ITcPlcLibrary 属性	获取/设置	描述	可用版本
显示名称	是 / 否	在库列表中标识库的显示名称	TwinCAT 3.1
分发服务器	是 / 否	库创建者	TwinCAT 3.1
Name	是 / 否	库名	TwinCAT 3.1
版本	是 / 否	库版本号	TwinCAT 3.1

版本信息

要求

所需 TwinCAT 版本
TwinCAT 3.1 及以上版本支持此接口

5.12 ITcPlcLibraries

5.12.1 ITcPlcLibraries

ITcPlcLibraries 表示 ITcPlcLibrary [▶ 153] 对象的集合，例如，在使用 ITcPlcLibraryManager [▶ 146]::ScanLibraries() [▶ 151] 方法或 ITcPlcLibraryManager [▶ 146]::Libraries 属性时。

方法 (按 Vtable 顺序)

要求

ITcPlcLibraries 方法	描述	可用版本
get_Item() [▶ 154]	返回位于指定位置的类型 ITcPlcLibrary [▶ 153] 的项。	TwinCAT 3.1

属性 (按 Vtable 顺序)

ITcPlcLibraries 属性	获取/设置	描述	可用版本
计数	是 / 否	返回集合中 ITcPlcLibrary [▶ 153] 对象的数量	TwinCAT 3.1

版本信息

所需 TwinCAT 版本
TwinCAT 3.1 及以上版本支持此接口

5.12.2 ITcPlcLibraries::get_Item

返回指定位置的 ITcPlcLibrary 对象。

```
HRESULT AddLibrary(long n, ITcPlcLibrary** pipType);
```

Parameters (参数)

n	[in] 项在列表中的位置。
pipType	[out, retval] 返回类型 ITcPlcLibrary 的对象

5.13 ITcPlcLibRef

ITcPlcLibRef 表示 ITcPlcLibrary [▶ 153] 或 ITcPlcPlaceholderRef [▶ 154] 对象的基础类。

属性 (按 Vtable 顺序)

要求

ITcPlcLibRepositories 属性	获取/设置	描述	可用版本
Name	是 / 否	返回 ITcPlcLibRef 对象的名称	TwinCAT 3.1

版本信息

所需 TwinCAT 版本
TwinCAT 3.1 及以上版本支持此接口

5.14 ITcPlcPlaceholderRef

表示与 ITcPlcReferences [▶ 153] 集合和 ITcPlcLibraryManager [▶ 146]::ScanLibraries() [▶ 151] 方法或 ITcPlcLibraryManager [▶ 146]::References 属性一起使用时的单个 PLC 占位符。

属性（按 Vtable 顺序）

ITcPlcLibrary 属性	获取/设置	描述	可用版本
显示名称	是 / 否	在库列表中标识库的显示名称	TwinCAT 3.1
分发服务器	是 / 否	库创建者	TwinCAT 3.1
Name	是 / 否	库名	TwinCAT 3.1
版本	是 / 否	库版本号	TwinCAT 3.1

版本信息

要求

所需 TwinCAT 版本
TwinCAT 3.1 及以上版本支持此接口

5.15 ITcPlcLibRepository

ITcPlcLibRepository 接口表示单个存储库，例如，在与 ITcPlcLibRepositories [▶ 155] 集合和 ITcPlcLibraryManager [▶ 146]::Repositories 属性一起使用时。

属性

ITcPlcLibRepository 属性	获取/设置	描述	可用版本
文件夹	是 / 否	存储库路径（文件系统）	TwinCAT 3.1
Name	是 / 否	存储库名称。	TwinCAT 3.1

版本信息

要求

所需 TwinCAT 版本
TwinCAT 3.1 及以上版本支持此接口

5.16 ITcPlcLibRepositories

5.16.1 ITcPlcLibRepositories

ITcPlcLibraries 表示 ITcPlcLibRepository [▶ 155] 对象的集合，例如，在使用 ITcPlcLibraryManager [▶ 146]::Repositories 属性时。

方法（按 Vtable 顺序）

要求

ITcPlcLibRepositories 方法	描述	可用版本
get Item() [▶ 156]	返回位于指定位置的类型 ITcPlcLibRepository [▶ 155] 的项。	TwinCAT 3.1

属性（按 Vtable 顺序）

ITcPlcLibRepositories 属性	获取/设置	描述	可用版本
计数	是 / 否	返回集合中 ITcPlcLibRepository [▶ 155] 对象的数量	TwinCAT 3.1

版本信息

所需 TwinCAT 版本
TwinCAT 3.1 及以上版本支持此接口

5.16.2 ITcPlcLibRepositories::get_Item

返回指定位置的 ITcPlcLibRepository 对象。

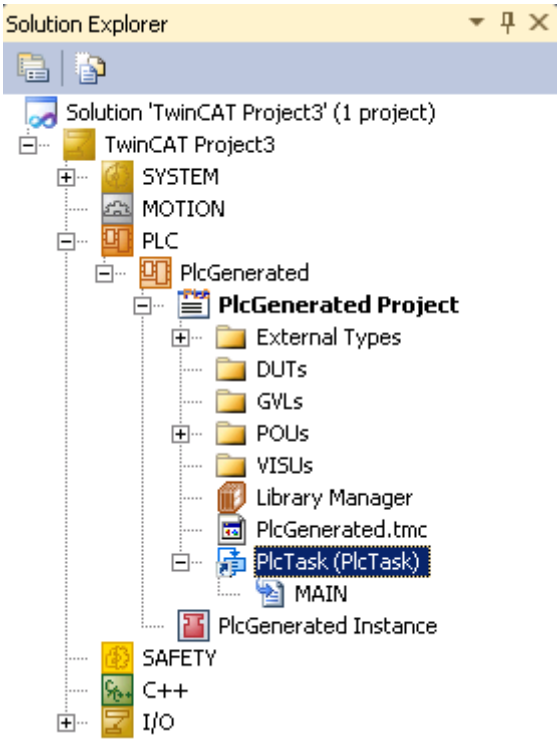
```
HRESULT AddLibrary(long n, ITcPlcLibRepository** ppRepo);
```

Parameters（参数）

n	[in] 项在列表中的位置。
pipType	[out, retval] 返回类型 ITcPlcLibRepository 的对象

5.17 ITcPlcTaskReference

ITcPlcTaskReference 接口使程序员能够获取或设置 PLC 项目的当前链接任务。这与以下 TwinCAT XAE 条目匹配：



属性（按 Vtable 顺序）

要求

ITcPlcTaskReference 属性	获取/设置	描述	可用版本
LinkedTask	是 / 是	获取或设置 PLC 项目的链接任务。设置新的链接任务时，任务将被指定为一个字符串，表示 TwinCAT XAE 树中任务的路径。	TwinCAT 3.1

6 实例

6.1 示例下载

在“最佳方案”部分，大多数示例和“如何操作”作为代码片段提供，以方便重复使用。而此下载部分还提供了可随时使用的二进制或源代码示例。

重要说明：

- 所有 C# 示例均基于 .NET 4.0 及以上版本的 Visual Studio 项目
- 所有 C# 示例均包含对 2.1 版本中“TwinCAT XAE Base”类型库的引用。根据安装的 TwinCAT 版本，可能需要更新对此库的引用，请参见安装章节 [▶ 15]
- 请注意，所有 Visual Studio 插件示例仅在 Visual Studio 2010 和 2012 下运行。要为 Visual Studio 2013 开发加载项，Microsoft 建议使用 VSPackage Extension，请参见相应 MSDN 网站上关于创建加载项和向导的内容。

SampleNo.	描述	编程/脚本语言	最低 TwinCAT 版本	下载
1	脚本容器 [▶ 158]	C#	取决于脚本	仅限 ScriptingContainer 二进制文件 ScriptingContainer 源
2	CodeGeneration Demo [▶ 162]	C#	3.1 Build 4014.0	仅限 CodeGenerationDemo 二进制文件 CodeGenerationDemo 源
3	Visual Studio 插件 PlcVersionInfo [▶ 163]	C#	3.1 Build 4016.6	PluginSample PlcVersionInfo.zip

还请参阅有关此

- 处理不同的 Visual Studio 版本 [▶ 26]
- 实现 COM 消息过滤器 [▶ 23]

6.2 脚本容器

6.2.1 脚本容器

脚本容器（在此处下载 [▶ 158]）是一个 C# WPF 应用程序，表示所有可用自动化接口示例的集合。大部分示例也可以作为独立的示例下载 - 但是，如果提供新示例，在创建单独的示例之前，它们将首先在脚本容器中发布。因此，建议熟悉此应用程序并定期检查是否有新示例。

本文介绍脚本容器应用程序的常规结构，并包含以下主题：

- 基本结构
- 首次设置

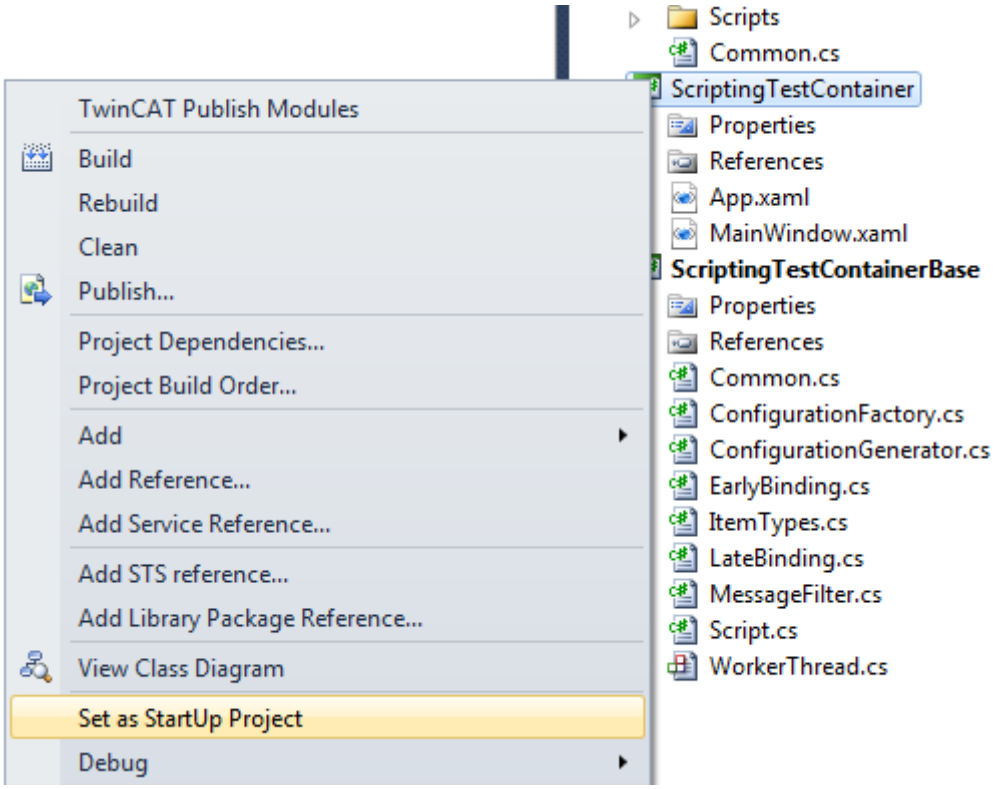
基本结构

如上所述，ScriptingContainer 由自动化接口示例的集合组成，这些示例由 ScriptingContainer 中的不同项目（每个项目自带 UI）表示。下表概述了所有可用项目以及指向相应章节的链接。

项目名称	描述
CodeGenerationDemo [▶ 162]	实现从 XML 文件中读取三种不同 TwinCAT 组态的 AI 代码。
CopyProjectDemo	将现有 TwinCAT 组态中的 I/O 和轴组态复制到新的组态中。
ScriptingTestContainer [▶ 159]	提供可从图形用户界面执行的可用自动化接口示例的集合。

首次

首次打开 ScriptingContainer 时，请通过右击此项目并选择“Set as StartUp Project”（设置为启动项目），将启动项目设置为所需的用户界面，例如“ScriptingTestContainer”。这可确保在应用程序启动时加载正确的 GUI。



现在可以通过进入“Debug”（调试）菜单并单击“Start Debugging”（开始调试）来启动应用程序。通过这种方式启动应用程序，可以设置断点或逐步执行代码，以便轻松计算执行的脚本。

6.2.2 项目

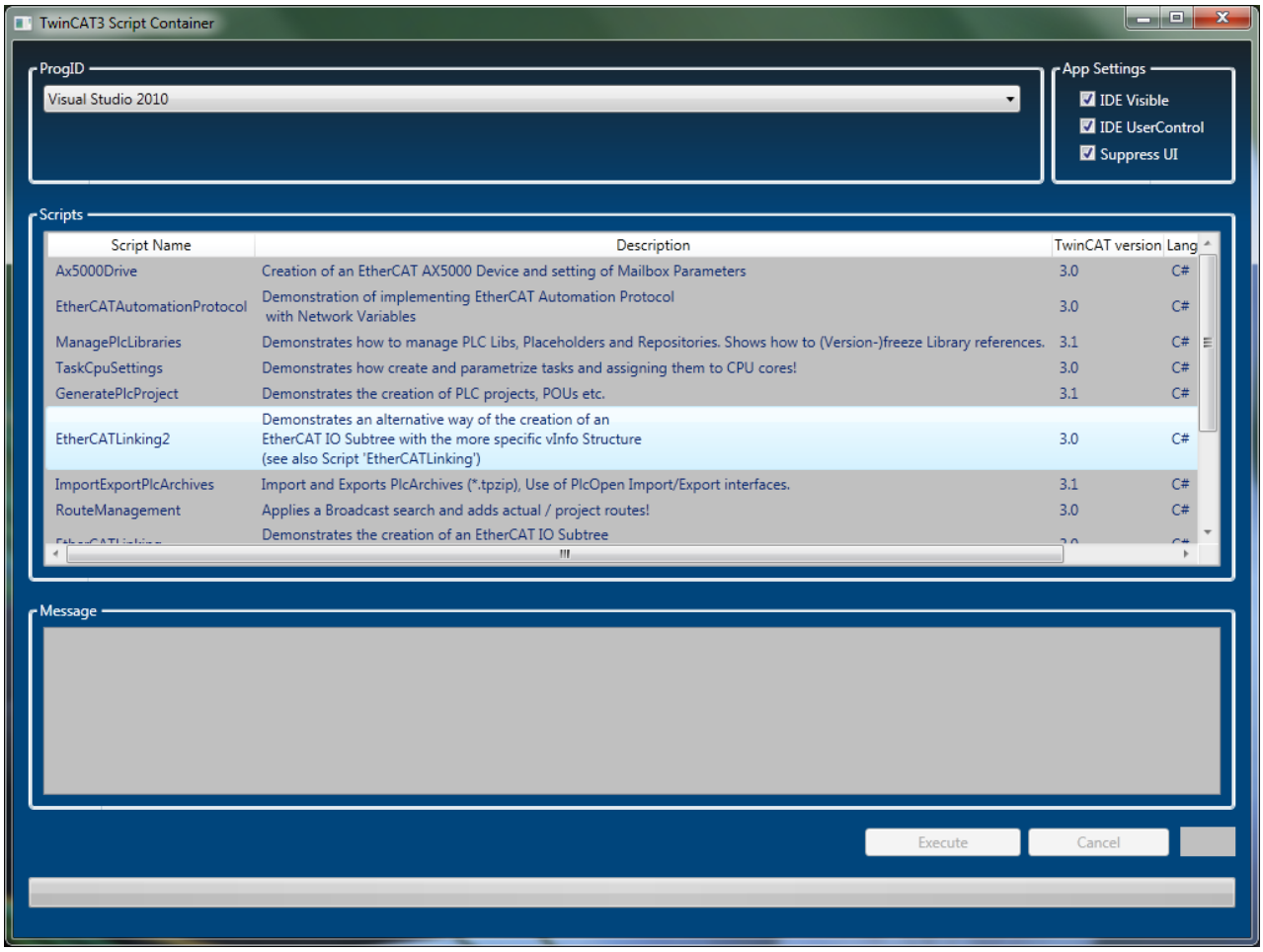
6.2.2.1 脚本容器：ScriptingTestContainer

本文介绍 ScriptingTestContainer 项目的基本结构，并包含以下主题：

- 图形用户界面 (GUI)
- 早期和后期绑定脚本示例
- 示例脚本的位置（**“自动化接口代码的位置???”** ））
- 后台：类结构
- 后台：方法结构
- 后台：实现自带示例

图形用户界面 (GUI)

启动脚本容器应用程序后，其 GUI 将如下所示：



可以在窗口顶部的下拉框中选择要用其创建 TwinCAT 组态的 Visual Studio 版本 [▶ 26]。

有多种可用的示例脚本，可以在窗口中央的表中进行选择。

要执行脚本，只需从表中进行选择并单击“Execute”(执行)。执行代码时，脚本在脚本表下的消息日志窗口中显示状态信息。

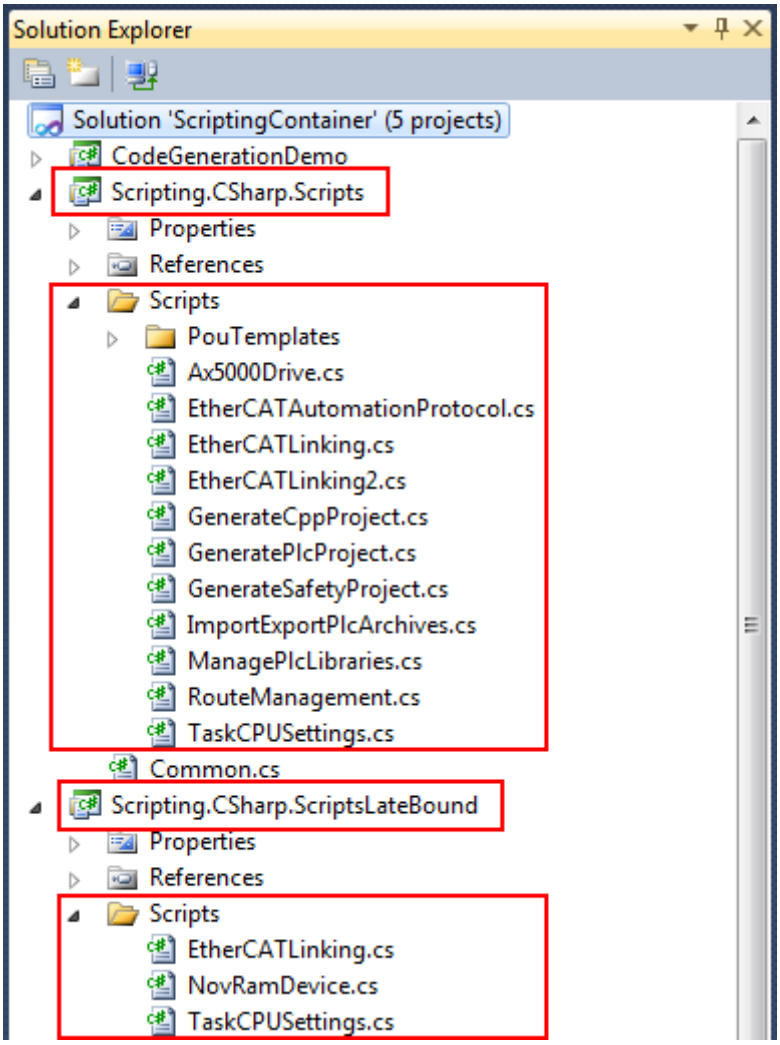
要更改 Visual Studio 的显示特性，可以切换窗口右上角的复选框。在默认情况下，Visual Studio 在脚本执行期间显示，且不会在脚本完成后关闭。

早期和后期绑定脚本示例

此应用程序包含早期和后期绑定的脚本示例。后期绑定的示例使用 .NET 数据类型“动态”，因此在运行时确定对象数据类型，而早期绑定的脚本在创建对象时使用对象的相应数据类型。这两种方法各有利弊，取决于开发人员及项目的首选类型处理方法。

示例脚本的位置 (“自动化接口代码的位置???”)

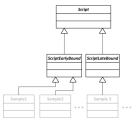
早期和后期绑定的所有脚本示例均位于 Visual Studio 解决方案中的自带项目容器内：
Scripting.CSharp.Scripts 用于早期绑定的示例代码，**Scripting.CSharp.ScriptsLateBound** 用于后期绑定的示例代码。



每个脚本文件实现一个 OnExecute() 方法，在其中实现 TwinCAT 自动化接口脚本代码。

后台：类结构

每个代码示例均由自带的类表示，该类由抽象类“ScriptEarlyBound”或“ScriptLateBound”派生而来 - 取决于示例中使用的数据类型处理方法。这两个类之间的区别在于 ScriptEarlyBound 类使用 DTE 和解决方案对象的静态类型。而 ScriptLateBound 类使用动态类型。下面的 UML 图更详细地解释了类层次结构。灰色类表示实际示例脚本，由类 ScriptEarlyBound（应使用静态类型时）或类 ScriptLateBound（应使用动态类型时）派生而来。



抽象类“Script”定义了 ScriptEarlyBound 和 ScriptLateBound 类通用的方法和属性。

后台：方法结构

每个示例类包含三个方法，这些方法用于在脚本容器中执行自动化接口代码。下表所示为它们的含义。

派生方法的签名	描述
OnInitialize (dynamic dte, dynamic solution, IWorker worker)	OnInitialize() 方法通常用于打开或准备新 TwinCAT XAE 组态的自动化接口代码。
OnCleanUp (IWorker worker)	OnCleanUp() 方法可以用于在执行代码后清除 TwinCAT XAE 组态。
OnExecute (IWorker worker)	此方法执行实际自动化接口脚本代码。

另请比较现有示例脚本的实现，以便更好地了解每个方法的工作方式。

后台：实现自带示例

在脚本容器中实现自带自动化接口示例非常容易。要实现自带自动化接口代码，开发人员只需确定要使用哪个绑定，并实现从其中一个主类（ScriptEarlyBound 或 ScriptLateBound）派生而来的新类，然后实现派生方法。

6.3 CodeGenerationDemo

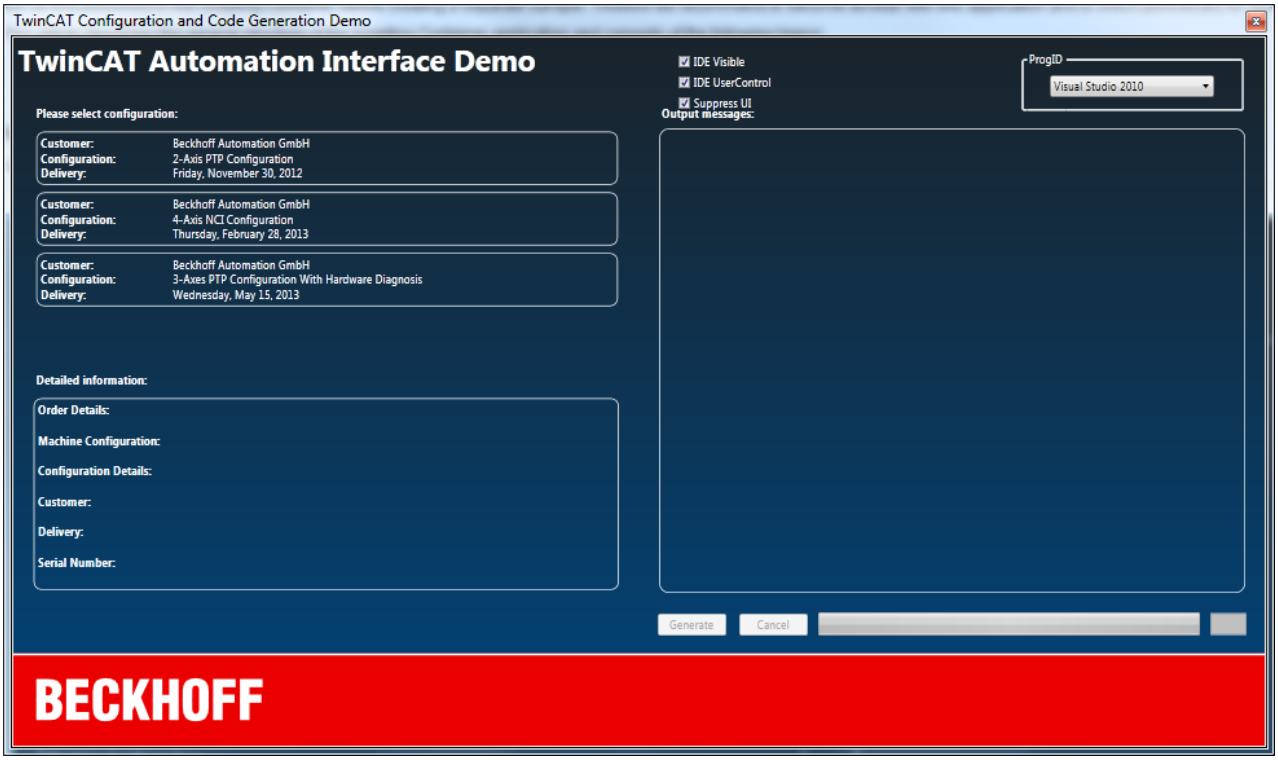
脚本容器（在此处下载 [▶ 158]）是一个 C# WPF 应用程序，表示所有可用自动化接口示例的集合。大部分示例也可以作为独立的示例下载 - 但是，如果提供新示例，在创建单独的示例之前，它们将首先在脚本容器中发布。因此，建议熟悉此应用程序并定期检查是否有新示例。

本文介绍脚本容器应用程序的常规结构，并包含以下主题：

- 图形用户界面（GUI）
- 示例脚本的位置（“自动化接口代码的位置???”）
- 数据位置（“所有数据来自哪里？”）

图形用户界面（GUI）

启动 CodeGenerationDemo 应用程序后，其 GUI 将如下所示：



在左侧，可以从三种不同的 TwinCAT 组态中进行选择，各组态都有自带的 I/O、轴和 PLC 组态。

通过右上角的下拉框，可以选择脚本代码应使用哪个 Visual Studio 版本来创建组态 - 如果已在系统中安装了不同的 Visual Studio 版本。

要开始创建组态，只需选择一个组态并按下“Generate”（生成）。

示例脚本的位置（“自动化接口代码的位置???” ））

实际的自动化接口代码可以在以下类中找到：

- CodeGenerationScript.cs

- ConfigurationScriptA.cs
- ConfigurationScriptB.cs
- ConfigurationScriptC.cs

根据所选组态，CodeGenerationDemo 应用程序将对三个 ConfigurationScriptX.cs 类中的其中一个进行实例化，这些类均由 CodeGenerationScript.cs 派生而来。

每个类都提供相应的方法来创建 PLC、轴或 I/O 组态。

数据位置（“所有数据来自哪里？”）

各组态有自带的 I/O、轴和 PLC 设置。您可能已经注意到，此组态并未硬编码到自动化接口代码中（正如在其他 AI 示例中一样）。在此演示中，这些设置均存储在一个 XML 文件中，该文件可在“CodeGenerationDemo\Data\Orders.xml”下找到。此 XML 文件指定两个重要的 XML 子结构，用于存储 TwinCAT 设置。

组态的主要描述：

```
<MachineOrders>
<Order id="1">
...
</Order>
</MachineOrders>
```

此结构定义一个组态，并设置其全局描述性属性，例如名称、描述...。这些属性将显示在组态选项下的 GUI 上。

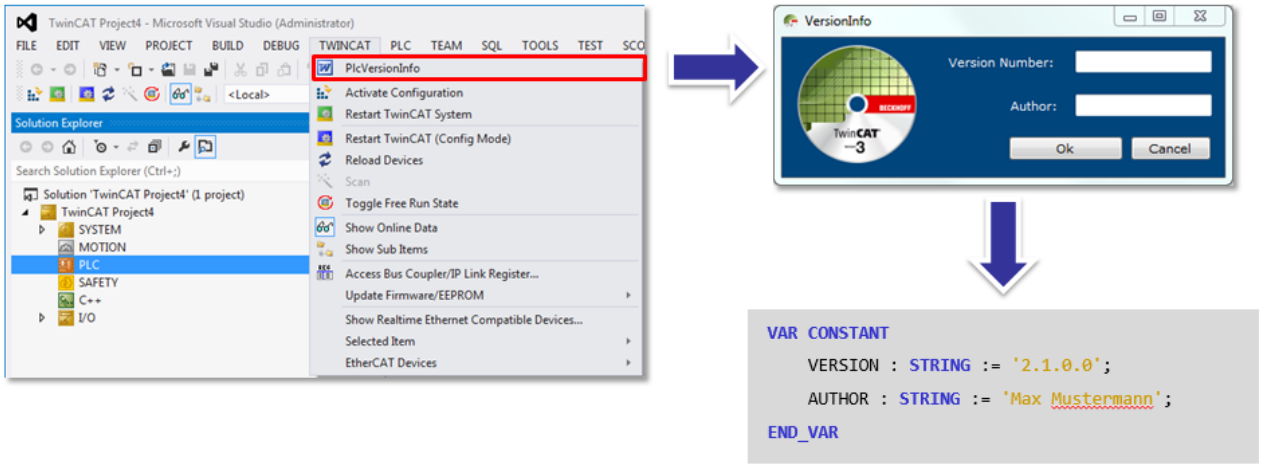
实际组态及对其描述的引用：

```
<AvailableConfigurations>
<Configuration id="..." name="...">
...
</Configuration>
</AvailableConfigurations>
```

此结构指定 I/O、轴、PLC 库及链接。

6.4 Visual Studio 插件 - PlcVersionInfo

PlcVersionInfo 示例显示如何在 Visual Studio 插件中使用 TwinCAT 自动化接口。其主要目的是展示如何创建一个通过更多功能扩展 PLC engineering 的简单工具。小型用户界面允许输入 PLC 功能块的版本及作者信息。执行示例时，此信息位于 PLC 项目内所有功能块的 VAR_CONSTANT 区域中。尽管这看起来可能是一个非常简单的示例，但清楚地展示了 Visual Studio 插件的强大功能及其如何极大地增强用户的工程体验。



有关 Visual Studio 插件的详细信息，请查阅 MSDN 网页。

基本概念

如前所述，此工具开始对一个 PLC 项目的所有 PLC 树项进行迭代。如果当前树项是一个功能块、功能或程序，则通过 VAR_CONSTANT 块确定并扩展树项的声明文本。

关于执行和调试 Visual Studio 插件的说明

Visual Studio 插件可能位于不同的文件夹中，以便 Visual Studio 对其进行识别。本文档假定插件只对一个特定用户帐户可用，因此插件将置于用户配置文件中的一个目录中。有关详细信息，请查阅 MSDN 网页的加载项注册部分。

要在不调试的情况下执行插件，请将已编译的 DLL 和 .AddIn 文件复制到目录 c:\Users\username\Documents\Visual Studio 201x\Addins\，并重新启动 Visual Studio。

要通过调试来执行插件，只需在插件解决方案中执行调试程序。如果这导致出现问题，也可以将 .AddIn 文件复制到 Addins 目录（如前所述），并使用所选的文本编辑器编辑该文件，以将 <Assembly> 节点更改为已编译的调试 DLL 的路径。然后启动一个 Visual Studio 实例并将调试程序附加到此实例/过程。

Visual Studio 插件将在 Visual Studio 的 TwinCAT 菜单中提供。

7 附录

7.1 各类错误代码

以下错误代码表示自动化接口方法的 HRESULT 值，如 API 引用 [▶ 95]中的说明。

```
typedefenum TCSYSMANAGERHRESULTS
{
    [helpstring("ITcSmTreeItem not found!" (ITcSmTreeItem nicht gefunden!))]
    TSM_E_ITEMNOTFOUND          = 0x98510001,
    [helpstring("Invalid Item Type!" (Ungültiger Elementtyp!))]
    TSM_E_INVALIDITEMTYPE         = 0x98510002,
    [helpstring("Invalid SubItem Type!" (Ungültiger Unterelementtyp!))]
    TSM_E_INVALIDITEMSUBTYPE        = 0x98510003,
    [helpstring("Mismatching Items!" (Nicht übereinstimmende Elemente!))]
    TSM_E_MISMATCHINGITEMS         = 0x98510004,
    [helpstring("Corrupted Link" (Fehlerhafte Verknüpfung))]
    TSM_E_CORRUPTEDLINK          = 0x98510005,
    [helpstring("Item still referenced!" (Element immer noch referenziert!))]
    TSM_E_ITEMREFERENCED         = 0x98510006,
    [helpstring("Item already deleted!" (Element bereits gelöscht!))]
    TSM_E_ITEMDELETED          = 0x98510007,
    [helpstring("XML Error" (XML-Fehler))]
    TSM_E_XMLERROR             = 0x98510008,
} TCSYSMANAGERHRESULTS;
```

请注意，以下 COM 错误代码列表只是代码片段，并不完整：

错误	描述
RPC_E_CALL_REJECTED	COM 服务器已拒绝该调用。请阅读有关如何实现自带消息过滤器 [▶ 23]的章节，以避免此错误。
返回方法 CoRegisterMessageFilter() <> 0 的值	此错误的一个原因可能是 COM 消息过滤器已应用于 MTA 单元。消息过滤器仅用于 STA 线程。有关消息过滤器及 STA 和 MTA 的详细信息，请查看消息过滤器 [▶ 23]章节。
在 Visual Studio 中引用类型库时，出现错误消息“A reference to Beckhoff TwinCAT XAE Base 2.0 Type Library could not be added”(无法添加对 Beckhoff TwinCAT XAE Base 2.0 类型库的引用)。	此类型库未正确注册。请通过执行以下命令重新注册类型库： *C:\Windows\Microsoft .NET\Framework\v4.0.xxxxx\regtlibv12.exe C:\TwinCAT3\Components\Base\TCatSysManager.tlb* xxxxxx 是当前安装的 .NET Framework 4.0 的版本号。