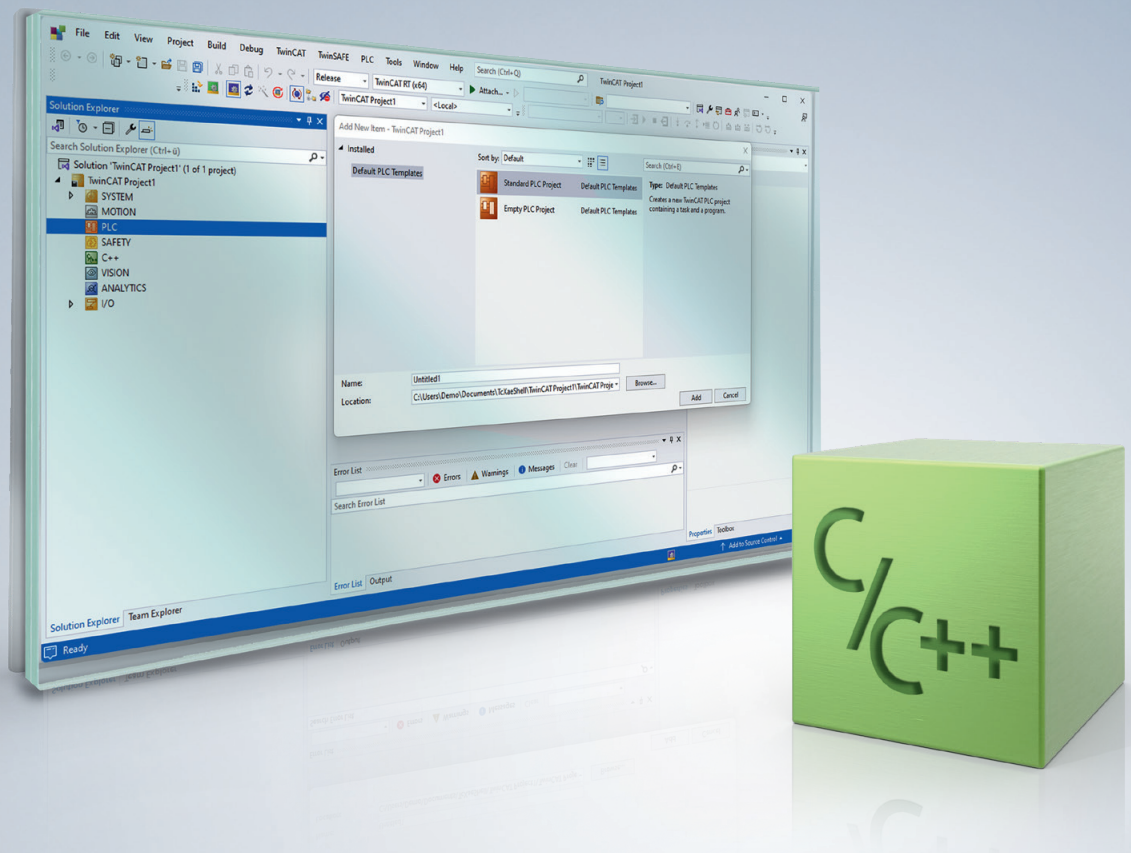
BECKHOFF New Automation Technology

手册 | ZH

TwinCAT 3

C/C++



10.02.2026 | 版本： 1.18.0

目录

1 前言

1.1 文档说明

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

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

免责声明

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®、ATRO®、EtherCAT®、EtherCAT G®、EtherCAT G10®、EtherCAT P®、MX-System®、Safety over EtherCAT®、TC/BSD®、 TwinCAT®、TwinCAT/BSD®、TwinSAFE®、XFC®、XPlanar® 和 XTS®是 Beckhoff Automation GmbH
的注册商标并由其授权使用。本出版物中所使用的其它名称可能是商标名称，任何第三方出于其自身目的使用它们可能会侵犯商标所有者的权利。

EtherCAT®是注册商标和专利技术，由 Beckhoff Automation GmbH 授权使用。

版权所有

第三方商标

本文档可能使用了第三方商标。有关商标信息，可以访问：https://www.beckhoff.com/trademarks。

1.2 安全信息

安全规范

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

责任免除

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

人员资格

本说明仅供熟悉适用国家标准的控制、自动化和驱动工程专家使用。

警示性词语

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

人身伤害警告

⚠ 危险
存在死亡或重伤的高度风险。

⚠ 警告
存在死亡或重伤的中度风险。

⚠ 谨慎
存在可能导致中度或轻度伤害的低度风险。

财产或环境损害警告

注意
可能会损坏环境、设备或数据。

操作产品的信息

这些信息包括：
有关产品的操作、帮助或进一步信息的建议。

1.3 信息安全说明

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

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

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

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

1.4 文档发行状态

版本	修改
1.18.x	**新增：** 使用 TwinCAT Task Dumps [▶ 80] 安装 [▶ 19] 根据 ADS 进行在线更改 [▶ 229] Third-party components [▶ 335]

2 概述

本章介绍在 C/C++ 中实现 TwinCAT 3。最重要的章节是：

- 基础内容
  支持哪些平台？实现 TwinCAT 3 C++ 模块是否需要额外程序？
  在要求 [▶ 19]和准备工作 [▶ 21]中可以找到答案，需遵循以下限制条件 [▶ 149]。
- 快速入门 [▶ 57]
  此为“五分钟内示例”，介绍如何用 C++ 实现简单的周期性计数器。该示例还介绍了对计数器值进行监控和覆盖，以及调试选项等。
- 模块 [▶ 37]
  模块化是 TwinCAT 3 的基本理念。了解 TwinCAT 3 的模块化理念是基本要求，尤其是对于 C++ 模块而言。
  需要了解 TwinCAT 模块结构的基础知识。
- 向导 [▶ 83]
  TwinCAT C++ 环境可视化组件的文档。
  其中包括用于创建项目的工具以及用于编辑模块和配置模块实例的工具。
- 编程参考 [▶ 139]
  本章包含有关 TwinCAT C++ 编程的详细信息。在此处，您可以找到用于 ADS 通信接口、其他TwinCAT 功能以及辅助方法。
- “如何.....？ [▶ 211]” 章节包含使用 TwinCAT C++ 的实用技巧。
- 示例
  以可执行程序的形式详细介绍一些接口及其使用方法，该程序可作为源代码和解决方案的下载文件提供。

3 简介

将可编程逻辑控制器 (PLC) 和数字控制器 (NC) 等传统自动化设备作为软件在功能强大的标准硬件上实现仿真的方法，多年来一直是最先进的技术，现在已被许多制造商所采用。

优点有很多，但最重要的无疑是该软件在很大程度上与硬件无关。这意味着，首先，硬件的性能能专门针对应用进行适配；其次，用户可以自动从硬件的进一步开发中获益。

尤其适用于 PC 硬件，其性能仍在以惊人的速度持续提升。这种软硬件分离所带来的供应商相对独立性，对用户来说也非常重要。

由于采用此方法时，PLC 和运动控制（可能还有其他自动化组件）仍然是独立的逻辑功能块，因此与传统自动化技术相比，应用架构的改动极少。

PLC 可确定机器的逻辑过程，并将特定轴功能的实现任务交给运动控制模块。由于控制器的高性能和使用更高级编程语言 (IEC 61131-3) 的可能性，即使是复杂的机器也可以通过这种方式实现自动化。

模块化

为了掌握现代机器的复杂性，同时减少必要的工程支出，许多机器制造商开始对机器进行模块化设计。因此，各个功能、组件或机器单元均可视为模块，尽可能独立使用，并通过统一的接口嵌入到整个系统中。

理想情况下，机器采用分层结构，最底层的模块代表最简单且可持续重复使用的基本元件。它们结合在一起，形成越来越复杂的机器单元，最终整合为完整的机器整体。在机器模块化的控制系统方面，遵循不同的方法。这些方法大致可分为分散式和集中式两种。

在本地方法中，每个机器模块都配有各自的控制器，控制器会决定 PLC 功能，也可能决定模块的运动功能。

各个模块可以单独运行和维护，并相对独立地进行扩展。控制器之间必要的交互通过通信网络（现场总线或以太网）进行协调，并通过适当的配置文件进行标准化设置。

集中式方法将所有模块的所有控制功能都集中在共用控制器中，仅在本地 I/O 设备中使用很少的预处理智能。由于通信轨迹变得更短，可以更直接地在中央控制单元内进行交互。不会出现死机时间，控制硬件的使用更加均衡，从而降低了总体成本。

然而，集中式方法也有缺点，即无法自动指定控制软件的必要模块化。与此同时，在中央控制器中能够从程序的其他部分获取信息的可能性，阻碍了模块的形成以及该控制软件在其他应用中的可重复使用性。由于控制单元之间不存在通信通道，因此控制单元的合理配置和标准化工作经常被忽视。

两全其美

模块化机器的理想控制器采用了分散式和集中式控制架构的元素。一款通用型的中央高性能计算机平台，“一如既往”地充当控制硬件。

集中控制技术的优势：

- 总成本低
- 具备可用性
- 高速、模块化的现场总线系统（关键词：EtherCAT）
- 以及在无通信的情况下访问系统中所有信息的可能性

这些是决定性参数。

通过对控制软件进行适当的模块化设计，可以在集中控制系统中实现上述分散式方法的优势。

许多小型“控制器”能够在同一硬件的通用运行时环境中相对独立地共存，无需运行大型复杂的PLC程序以及多轴数控（NC）系统。各个控制模块自成一体，通过标准接口向环境提供功能，或者使用其他模块或运行时的相应功能。

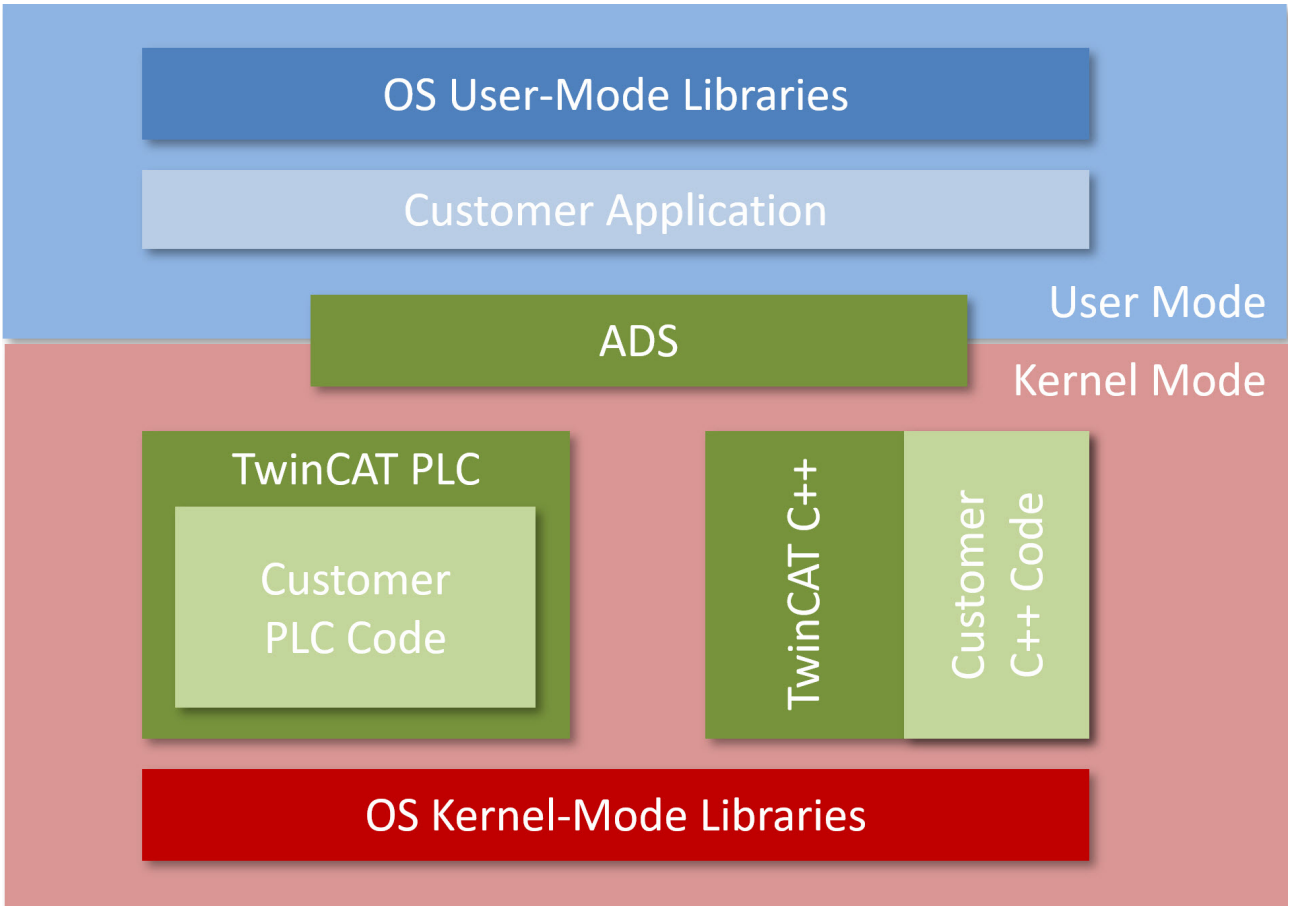
通过对这些接口的定义以及相应参数和过程数据的标准化，创建重要的配置文件。由于各个模块会在运行时中执行，因此也可以通过相应的标准接口直接调用其他模块。这样，模块化便可在合理的范围内进行，而不会出现通信中断。

在开发或调试各个机器模块过程中，可在具有相应运行时的控制硬件上创建和测试相关控制模块。在此阶段，缺失的模块连接可以被模拟。在整机上，会统一将其运用在中央运行时上进行实例化，只需确保其资源配置能满足所有已实例化模块（内存、任务和计算能力）的资源需求即可。

TwinCAT 3 Runtime

TwinCAT Runtime 可为 TwinCAT 模块的加载、实现和管理提供一个软件环境。可提供更多的基本功能，以便使用系统资源（内存、任务、现场总线和硬件访问等）。各个模块无需使用相同的编译器来创建。这意味着模块可以相互独立，采购自不同的供应商。

运行时启动时，会自动加载一些系统模块，以便其他模块可以使用其属性。然而，系统模块的属性访问方式与常规模块的属性访问方式相同，那么对于模块而言，相应的属性是由系统模块还是常规模块提供并不重要。

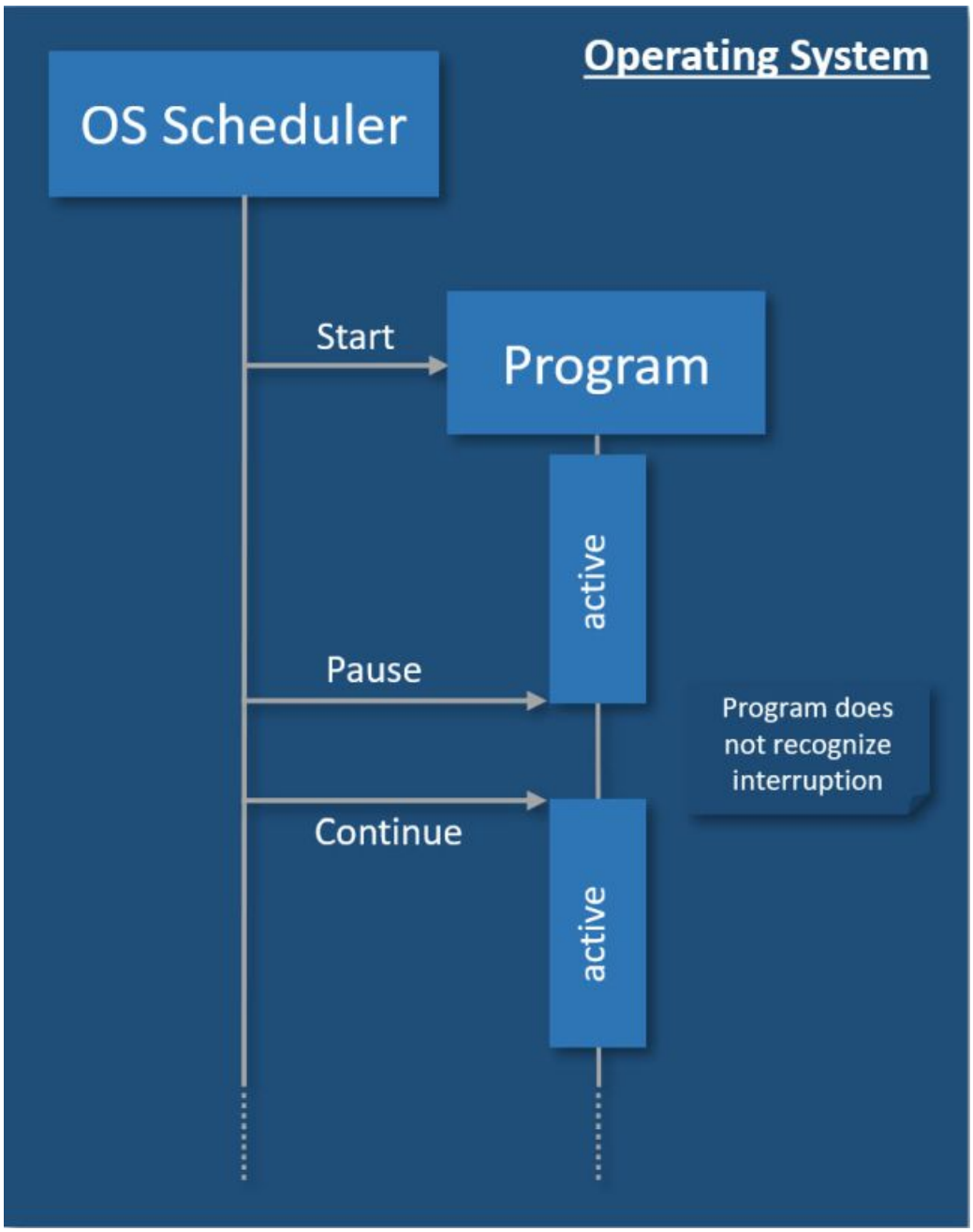


与在运行时环境中执行用户定义程序的 PLC 不同，TwinCAT C++ 模块并不处于此类托管环境中。这会导致 TwinCAT C++ 模块 (.tmx) 作为内核模块执行，并由 TwinCAT 加载。

3.1 用户模式与实时编程的区别

本文介绍了使用 C++、C# 或 Java 等编程语言进行标准用户模式编程与在 TwinCAT 中进行实时编程在概念上的区别。

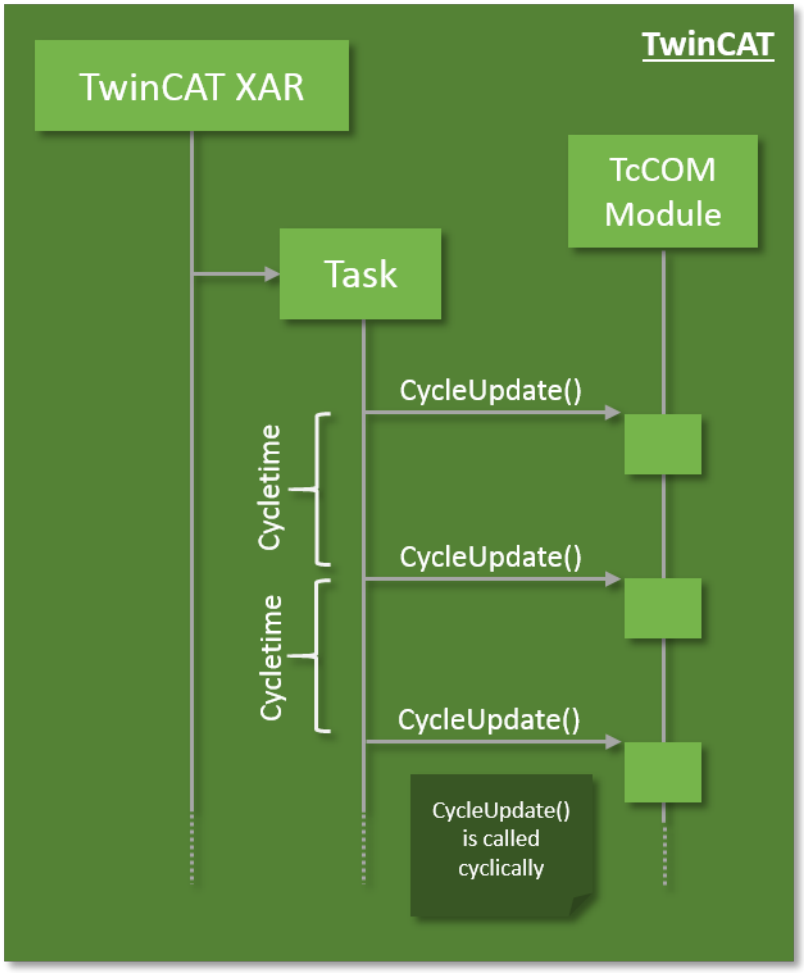
本文重点关注使用 TwinCAT C++ 进行实时编程的问题，因为在这个问题上，先前掌握的 C++ 编程知识显得尤为重要，而且必须考虑到 TwinCAT 实时系统的序列特征。



通过传统的用户模式编程，例如使用 C# 创建一个程序，然后由操作系统执行。

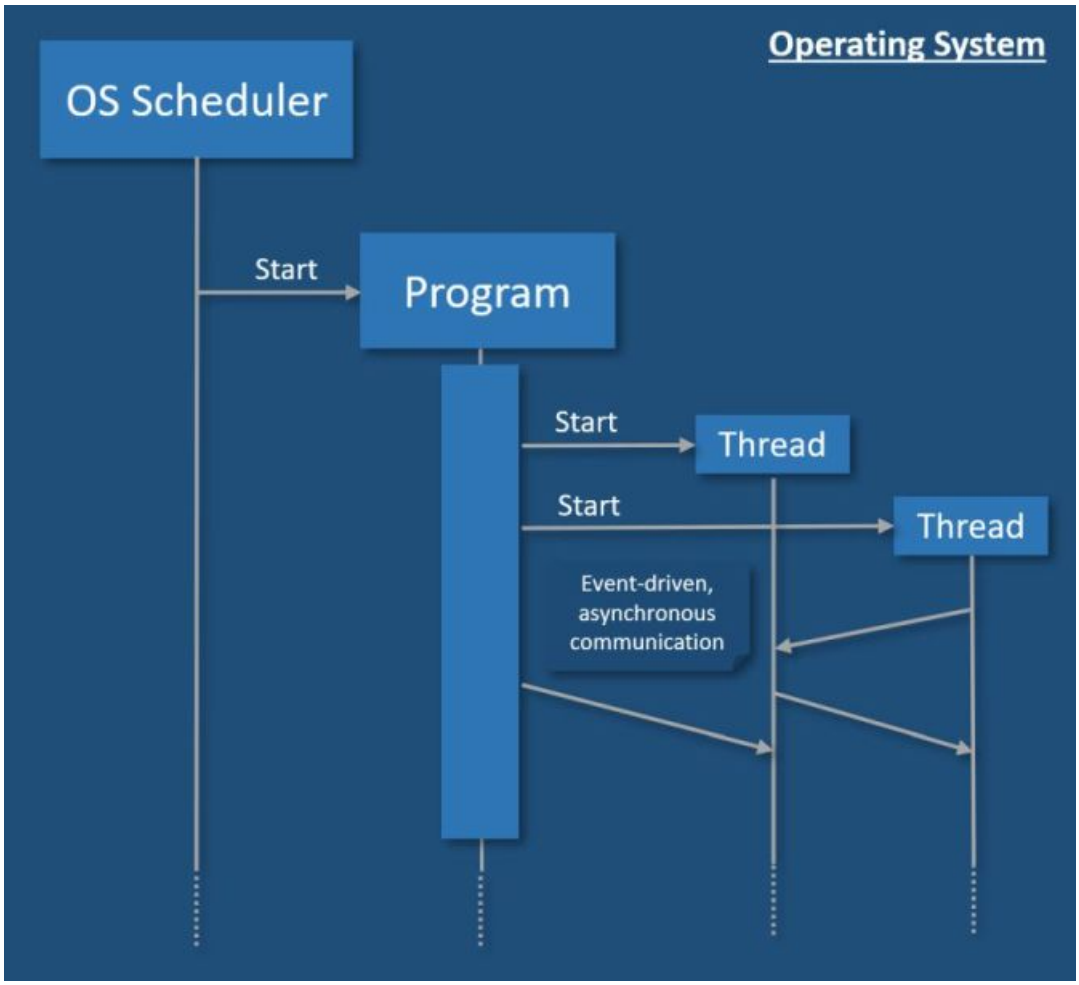
该程序由操作系统启动，可以独立运行，即完全自主执行，包括线程和内存管理等方面。为了实现多任务处理，操作系统会在任何时间、任何时段中断该程序。程序不会记录这种中断。操作系统必须确保这种中断不会被用户察觉。程序与其环境之间的数据交换为事件驱动型，即非确定性且常会导致阻塞。

这种行为不适用实时条件下执行，因为实时应用程序本身必须能够依赖可用资源，才能确保实时特性（响应保证）。



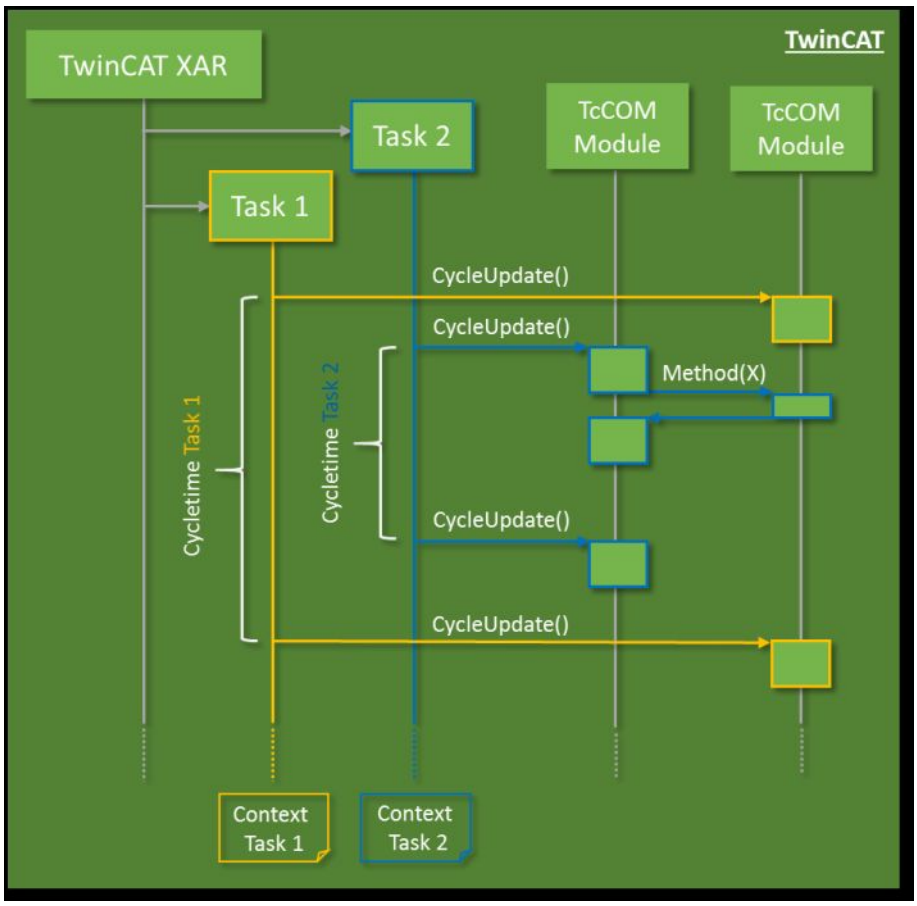
因此，TwinCAT C++ 采用 PLC 的基本理念：TwinCAT 实时系统负责管理实时任务、进行调度并周期性调用程序代码中的入口点。程序执行必须在可用周期时长内完成，并将控制权交回系统。TwinCAT 系统可在过程映像中提供 I/O 区域的数据，从而保证数据访问的一致性。这意味着程序代码本身不能使用线程等机制。

并发性



在用户模式下进行传统编程时，并发性由程序控制。线程即在此启动，线程之间可以相互通信。所有这些机构均需占用资源，而资源的分配与启用过程，可能会影响实时性能。线程之间的通信基于事件，这样调用线程便无法控制被调用线程的处理时间。

在 TwinCAT 中，任务用于调用模块，因此体现出并发性。任务分配至一个内核；具有周期时间和优先级，因此优先级较高的任务可以中断优先级较低的任务。如果使用多个内核，任务实际上是并行执行的。



模块之间可以相互通信，因此在并行模式下必须确保数据的一致性。
例如，可通过映射实现跨任务边界的数据交换。例如，当使用方法直接访问数据时，必须通过临界区加以保护。

启动/停机行为

TwinCAT C++ 代码在所谓的“内核环境”和“TwinCAT 实时环境”中执行，而不是作为用户模式应用程序执行。

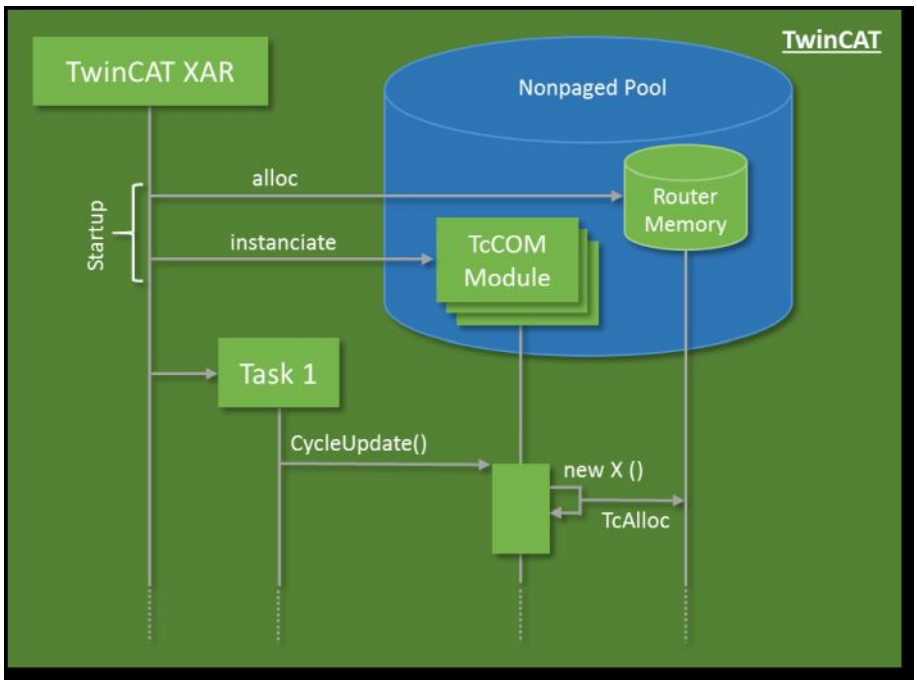
在模块的启动/关闭过程中，用于（反）初始化的代码最初在内核环境中执行；只有最后阶段和循环调用才会在 TwinCAT 实时环境中执行。

详情请参见模块状态机 [▶ 43]一章。

内存管理

TwinCAT 有独立的内存管理机制，也可以在实时环境中使用。此内存从操作系统提供的所谓“非分页池”中获取。在此内存中，TcCOM 模块根据其内存需求进行实例化。

此外，TwinCAT 还在该内存区域提供了所谓的“路由器内存”，TcCOM 模块可以在实时环境中（例如使用 New操作符）从该内存区域动态分配内存。



如果可能，一般应提前分配内存，而不是在周期代码中分配。在每次分配过程中，都需要进行检查，以确认内存是否确实可用。因此，周期代码中的分配执行取决于内存的可用性。

4 安装

4.1 系统要求

工程(XAE)

技术数据	要求
操作系统	Windows 10/11
目标平台	x64
TwinCAT 版本	TwinCAT 3.1 Build 4024 | Build 4026
所需 TwinCAT 授权	无（Microsoft 提供的 Visual Studio 授权）

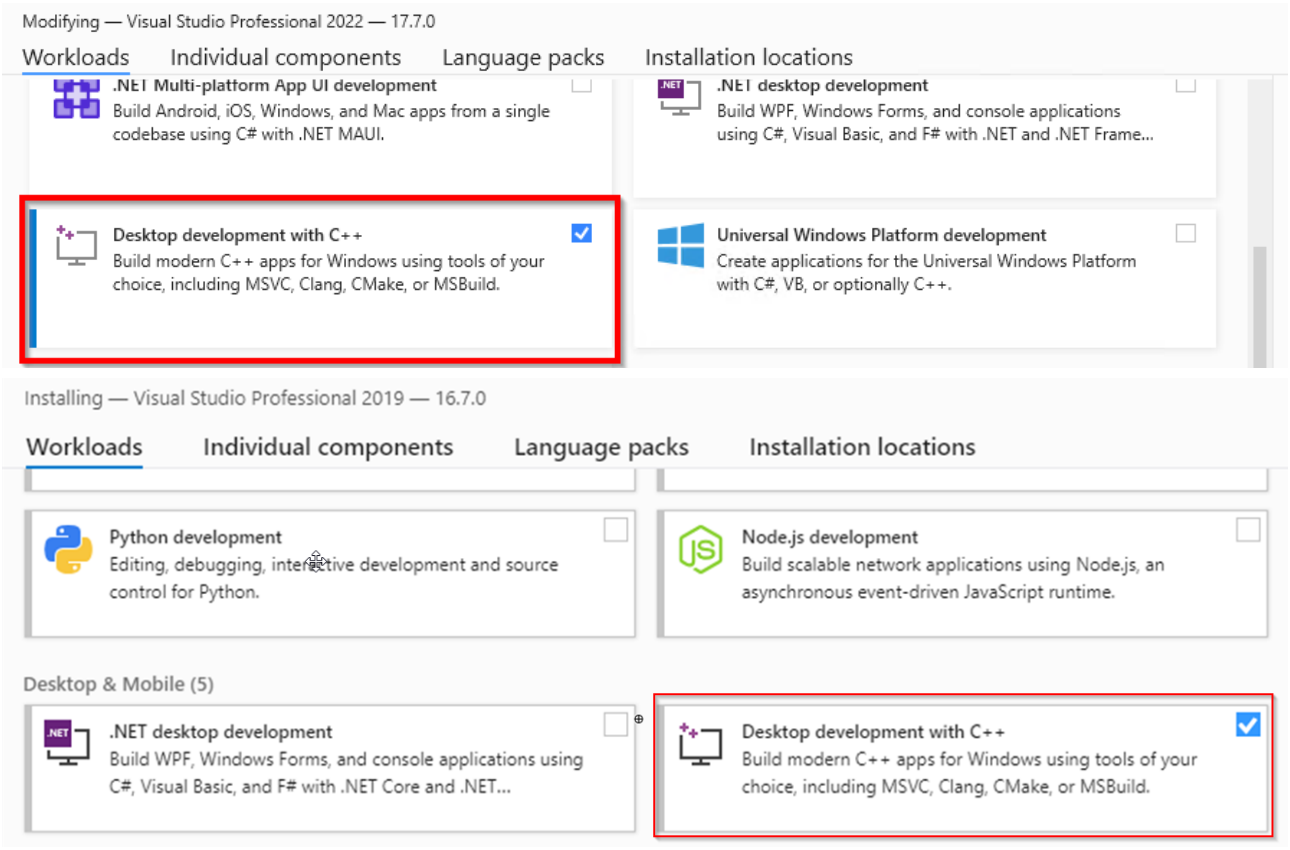
运行时 (XAR)

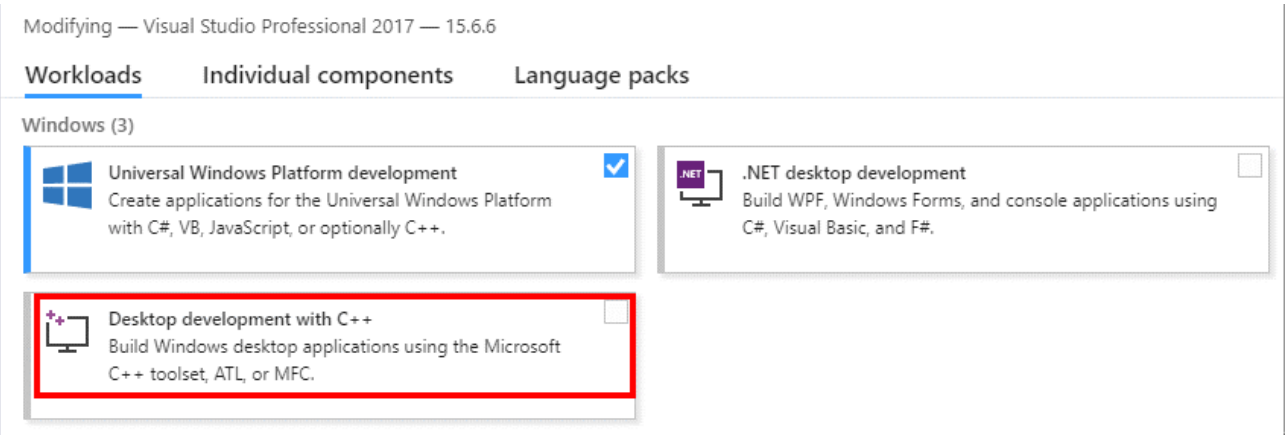
技术数据	要求
操作系统	Windows 10，TwinCAT/BSD
目标平台	x64
TwinCAT 版本	TwinCAT 3.1 Build 4024 | Build 4026
所需 TwinCAT 授权	TC1300

4.2 安装

工程前提条件

开发在PC 上必须安装 Microsoft Visual Studio® Professional/Enterprise 2017、2019 或 2022。
在 Visual Studio® 安装程序中，必须选择 **Desktop development with C++** 选项，因为自动安装时未选择该选项：





XAE Shell 足以在 TwinCAT 3 PLC 环境中集成和使用现有的二进制 C++ 模块（不需要 Visual Studio®）。

运行时前提条件

无需安装 Microsoft Visual Studio®。

- 适用于 TwinCAT 3.1 Build 4024.x：TwinCAT 3.1 XAR
- 适用于 TwinCAT 3.1 Build 4026.x：TwinCAT Standard Runtime

TwinCAT Package Manager：安装 (TwinCAT 3.1 Build 4026)

有关安装产品的详细说明，请参阅 TwinCAT 3.1 Build 4026 安装说明中的安装工作负载章节。

安装以下工作负载才能使用产品：

TC1300 | TwinCAT 3 C/C++

TwinCAT 设置：安装（TwinCAT 3.1 Build 4024 及更早版本）

包含在 TwinCAT 3.1 Full 设置中。

TwinCAT/BSD：

无需安装

5 准备工作 - 仅此一次

必须准备一台用于 TwinCAT C++ 模块工程设计的 PC。这些步骤只需执行一次：

- 配置 TwinCAT Basis [▶ 21] 以及配置和平台工具栏。
- 签名模块以便执行；请参见文档，设置测试签名 [▶ 21]。

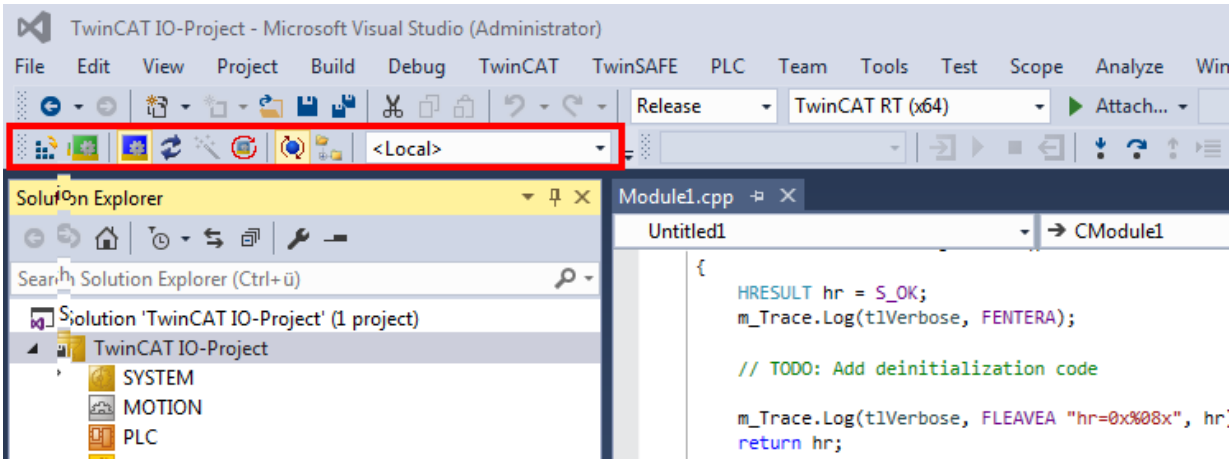
5.1 Visual Studio - TwinCAT XAE Base 工具栏

通过 TwinCAT XAE Base 工具栏实现高效的工程设计

TwinCAT 3 在 Visual Studio 菜单中集成自身工具栏，以提高效率。可以协助创建 C++ 项目。通过 TwinCAT 3 设置，该工具栏会自动添加到 Visual Studio 菜单中。如果您想手动添加，请执行以下操作：

1. 打开**视图**菜单，选择**工具栏\TwinCAT XAE Base**

⇨ 选定的工具栏会显示在菜单下方。



5.2 模块签名

TwinCAT C++ 模块必须使用证书签名才能执行。

该签名可确保对生产系统仅执行可追溯来源的 C++ 软件。

C++ 模块由 TwinCAT 运行时系统加载，必须使用 TwinCAT 用户证书签名。

出于测试目的，可使用无法验证的证书进行签名。然而，只有在操作系统处于测试模式时才可行，从而避免在生产系统上使用这些证书。

工程开发无需签字

只有执行需要使用证书，而工程设计则不需要。

开发软件和生产软件的组织隔离

倍福建议在组织结构中使用（至少）两个证书。

1. 无交叉签名的证书，因此在开发过程中需要使用测试模式。该证书也可由各开发者单独颁发。然后将测试系统设置为测试模式。
2. 只有通过相应最终测试的软件才由会签证书签署。因此，该软件也可以安装在机器上并交付使用。

这种开发与操作分离的做法可确保只有经过测试的软件才能在生产系统上运行。

5.2.1 TwinCAT

版本化 C++ 项目以二进制形式存储在 TMX 文件 (TwinCAT Module Executeable) 中。

对于 TwinCAT 3 C++ 模块的实现，如果要由 TwinCAT Runtime 加载，该编译后的可执行 TMX 文件必须用 TwinCAT 用户证书签名。

签名

只有在成功签名后才能加载 TwinCAT TMX 文件。

注意

在 32 位和 64 位系统上签名

与操作系统签名不同，TwinCAT 签名适用于 32 位和 64 位系统。因此，在 32 位系统上进行测试签名时也假定使用测试模式。

要签署 TMX 文件，需要使用 TwinCAT 用户证书 [▶ 22]，该证书在待签署项目 [▶ 144]中进行了相应配置。

测试签名

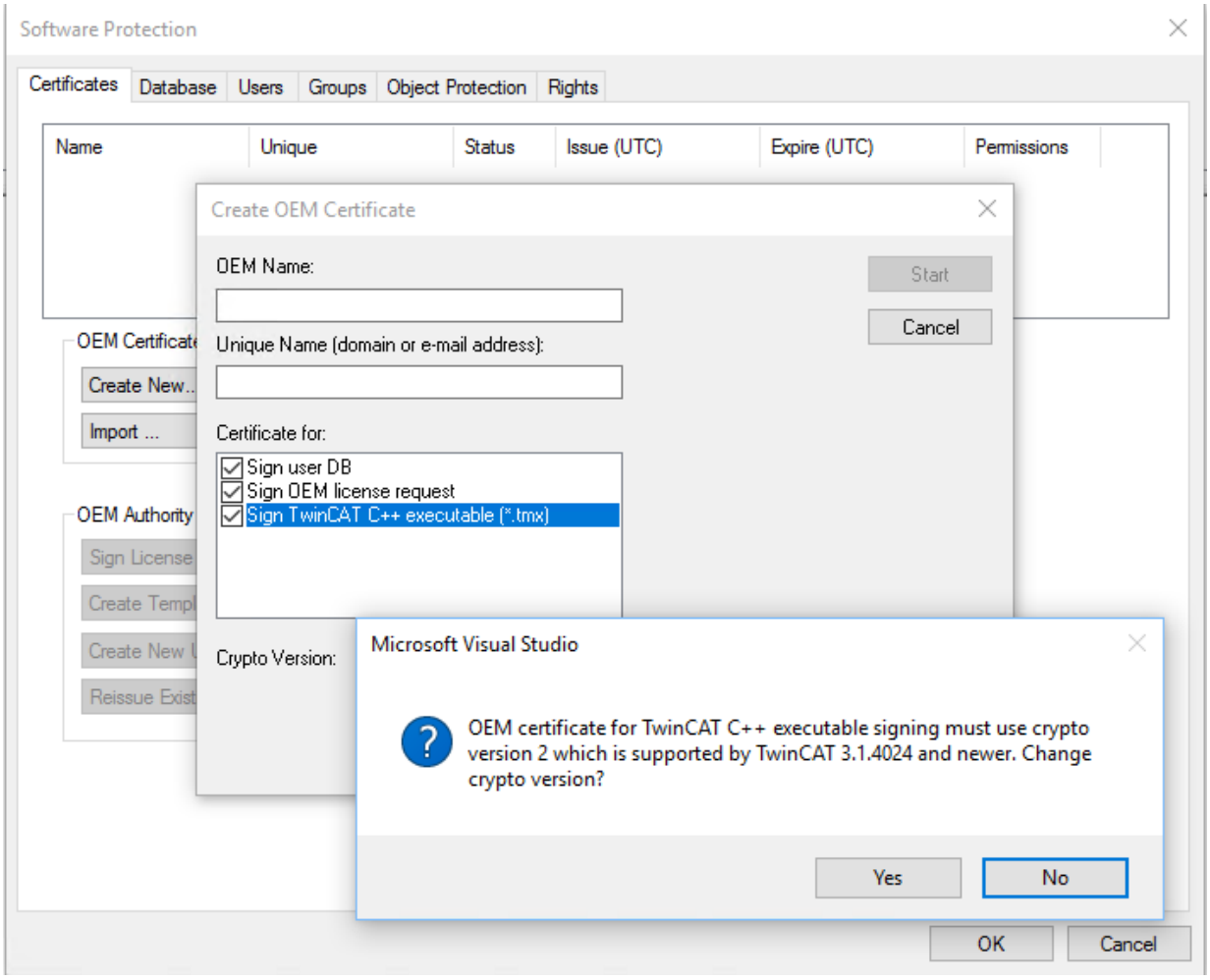
可在 TwinCAT 中执行用户证书本地创建。只要未经倍福会签，就必须启用测试模式 [▶ 22]。

一旦倍福对 TwinCAT 用户证书进行会签 [▶ 24]，便可以相应地取消测试模式。停用方式与启用方式相同。

5.2.1.1 测试签名

可以使用与实际交付相同的 TwinCAT 用户证书对 TwinCAT 进行测试签名（见 申请 TwinCAT 3 用户证书 [▶ 25]）。

1. 对于测试运行，例如在软件开发期间，只需创建一个 TwinCAT 用户证书 [▶ 26]即可。确保选择用途“签署 TwinCAT C++ 可执行文件 (*.tmx)”。为此，需要使用 Crypto 版本 2，此时会显示一条消息。

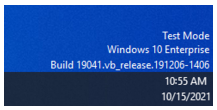


在 XAR（和 XAE，若为本地测试）上，启用测试模式，以便操作系统可以接受自签名证书。可以在工程系统 (XAE) 和运行时系统 (XAR) 上完成操作。

适用于 Windows

利用管理员提示执行以下操作：
`bcdedit /set testsigning yes`
并重启目标系统。
为此，必须关闭“SecureBoot”，可在 BIOS 中完成操作。

如果已启用测试签名模式，则会在桌面右下方显示。现在，系统接受所有已签名的执行驱动程序。



适用于 TwinCAT/BSD

在 `/usr/local/etc/TwinCAT/3.1/TcRegistry.xml` 文件中，在密钥“`System`”下输入 "`<Value Name="EnableTestSigning" Type="DW"1</Value>`"。

```
<Key Name="System">
  <Value Name="RunAsDevice" Type="DW">1</Value>
  <Value Name="RTimeMode" Type="DW">0</Value>
  <Value Name="AmsNetId" Type="BIN">052445B00101</Value>
```

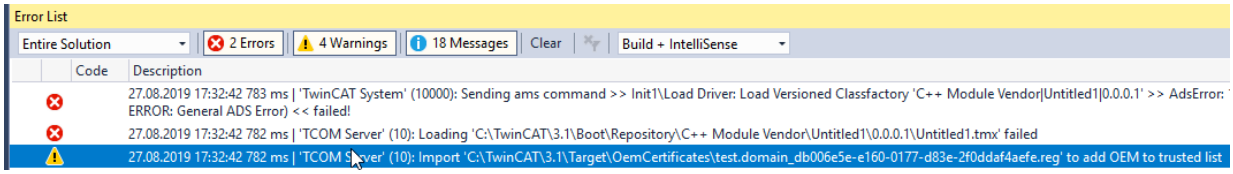
```
  <Value Name="LockedMemSize" Type="DW">33554432</Value>
 <Value Name="EnableTestSigning" Type="DW">1</Value>
</Key>
```

然后重启 TwinCAT 系统服务：

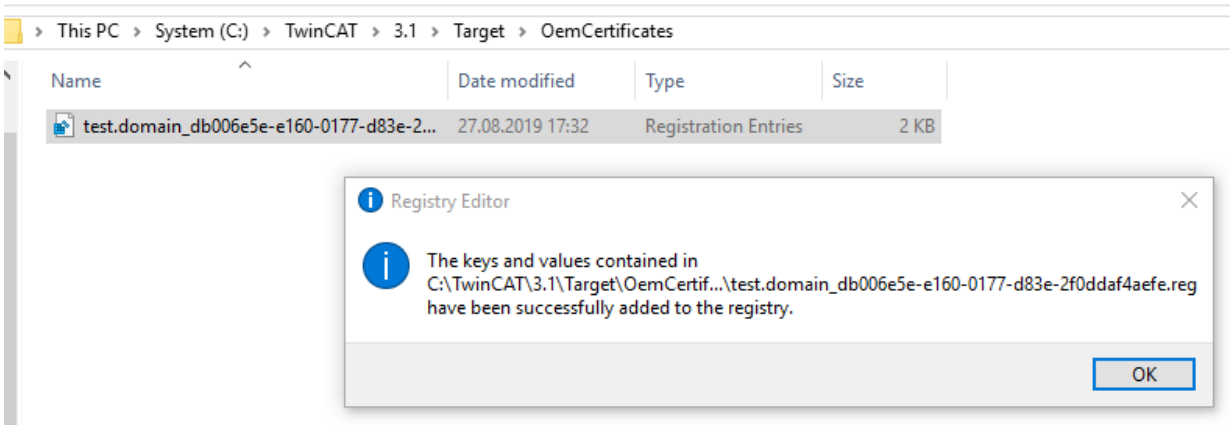
```
doas service TcSystemService restart
```

完成相应程序后，系统会接受所有已签名的执行驱动程序。

1. 在使用 TwinCAT 用户证书执行首次启用 (Activate Configuration) 时，目标系统会检测到证书不受信任，启用过程中止：



适用于 Windows：
具有管理权限的本地用户可以通过创建的 REG 文件信任证书，只需执行该文件即可：



适用于 TwinCAT/BSD：
如果未安装“Tcimportcert”软件包，请安装：`pkg install TcImportCert`
通过 `doas tcimportcert /usr/local/etc/TwinCAT/3.1/Target/OemCertificates/<CreatedFile>.reg` 信任证书。
然后重启 TwinCAT 系统服务或重新启动系统：

```
doas service TcSystemService restart
```

⇨ 此过程只允许运行具有受信任 TwinCAT 用户证书签名的 C++ 模块。

2. 在此过程之后，您可以使用 TwinCAT 用户证书以操作系统的测试模式进行签名。
在项目属性 [▶ 144]中对此进行配置。
使用 TcSignTool [▶ 54] 可避免在项目中存储 TwinCAT 用户证书的密码，例如，在版本管理中也会出现这种情况。

如果要使用不带 TestMode 的 TwinCAT 用户证书进行交付，则必须由倍福对证书进行会签 [▶ 24]。

5.2.1.2 在非测试模式下交付的用户证书

系统要求
- 至少为 TwinCAT 3.1 Build 4024
- 至少为 Windows 10 或 TwinCAT/BSD®（在目标系统上）

随着 TwinCAT Build 4024 的推出，倍福为现有客户提供了“TwinCAT 3 OEM 用户证书”，可对使用 C++ 代码通过 TwinCAT 3 创建的 TMX 文件进行签名。

- 由于是在 Windows 环境下使用，因此该证书需要对申请人数据进行安全验证。因此，必须正式下订单订购 TwinCAT 3 用户证书，以验证地址和联系信息，并且只发放给倍福现有客户。
- 订单号：**TC0008**
- 此 TwinCAT 3 用户证书申请在中国地区需要收费。

- 证书保存目录：**C:\TwinCAT\3.1\CustomConfig\Certificates**

使用 TwinCAT 3 TMX 文件不需要 TwinCAT 3 用户证书

TwinCAT 3 用户证书仅用于 TMX 文件的一次性签名，使用已通过该证书签名的 TMX 文件时不需要该证书。

哪些计算机需要 TwinCAT 3 用户证书 TC0008?

TwinCAT 3 用户证书应仅位于对 TMX 文件签名的编程计算机上，**而不是**每个目标系统上。

TwinCAT 3 用户证书的有效性

出于安全考虑，TwinCAT 3 用户证书的有效期限定为两年。

证书期满后会怎样?

您将无法再对新的 TMX 文件进行签名。
不过，使用已签名的 TMX 文件不受任何限制。

您可以在两年期满前（甚至期满后）申请证书续期。

延长 TwinCAT 3 用户证书的过程与申请新证书相同。在这种情况下，也必须订购证书（证书延期的订单号与新证书申请的订单号相同）。

与新证书不同的是，您不需要生成一个新的“OEM 证书申请文件”，而是将您现有的证书发送给倍福证书部门进行更新。请在电子邮件中告知我们，您想要为证书办理延期，而不是申请新的证书。否则，会以申请新证书的标准来处理电子邮件内容。

现有证书将获得一个新的到期日，然后重新签发，有效期为 2 年。

因此，新签发的证书与原始版本完全兼容。

5.2.1.2.1 申请 TwinCAT 3 用户证书

订购和验证流程概述

申请 TwinCAT OEM 证书时，需要正式的订单。

- 订单号：**TC0008**（TwinCAT 3 证书延期验证）
- 在中国地区 TwinCAT 3 用户证书将收费进行发放（和续期）。
- 由于 TwinCAT 3 用户证书是一种数字 ID 卡，因此需要根据常用的市场标准对询问者的联系资料进行验证。
- 因此，TwinCAT 3 用户证书只发放给倍福现有客户。

订购和验证流程概述

您的电子邮件地址必须是公司电子邮件账户（不允许使用 GMail 或类似的免费电子邮件），并符合询问者的公司名称。

1. 请联系您的倍福销售联系专员，并指明申请 TwinCAT 3 OEM 证书。订购“TC0007“或“TC0008”。
2. 重要提示：作为询问者，请提供您的详细联系方式作为交付地址（=联系人姓名和电子邮件地址）和证书使用区域（公司名称、地址）。
3. 订单中提供的详细联系信息将会予以核实，倍福销售部会与您（交付地址中指定的询问者）联系。
4. 申请新的 OEM 证书时，创建 TC0008 的证书申请文件 [▶ 26]。

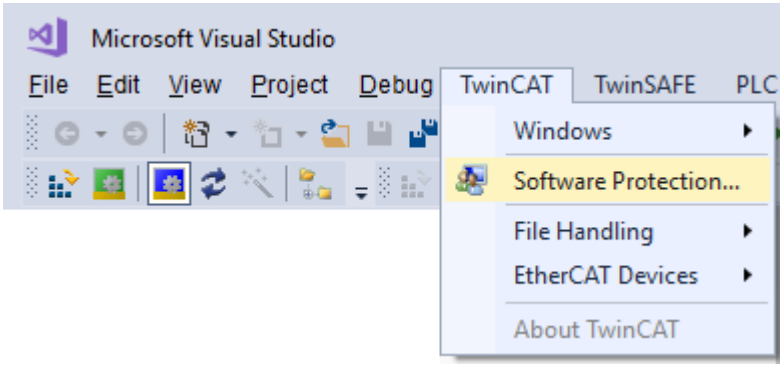
5. 使用 TwinCAT Engineering 确定 OEM 证书文件的“文件指纹”（参见确定 OEM 证书文件的文件指纹 [▶ 29]）。请将此文件指纹告知倍福销售联系专员，作为您联系资料验证的一部分。传输文件指纹时，必须使用与发送 OEM 证书请求文件不同的通信渠道。
6. 现在，将“OEM 证书文件“发送给倍福销售联系专员。
7. 在倍福总部签名证书文件后，您会通过电子邮件从您联系人那里收到该文件。

请注意，可能需要数天时间来验证您的联系信息并签发证书。

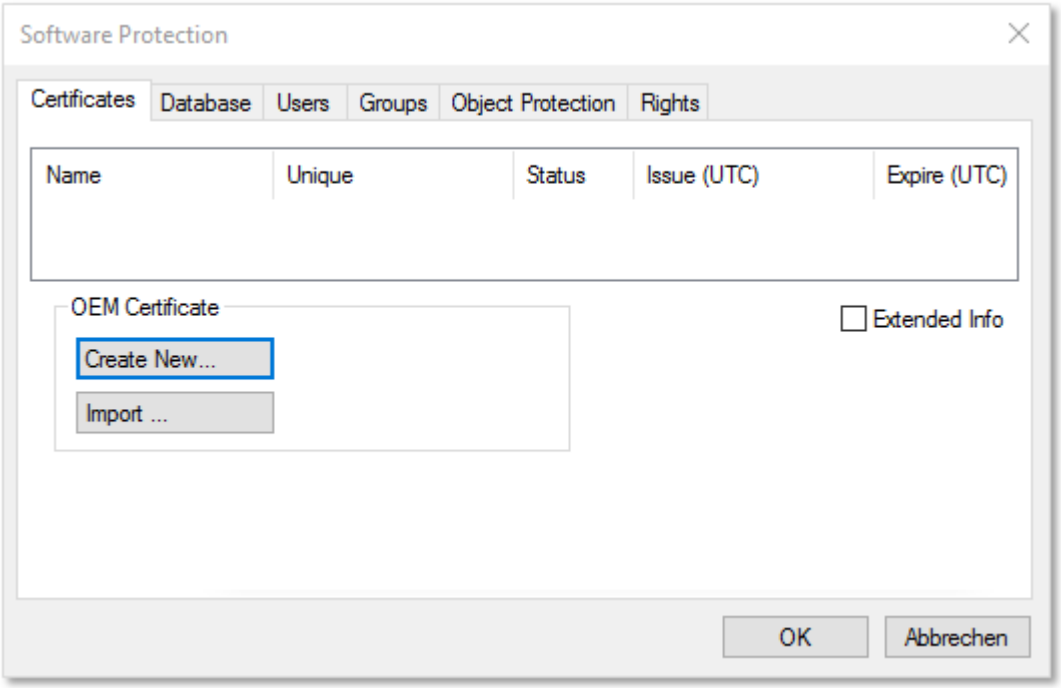
5.2.1.2.2 创建 TC0008 的证书申请文件

系统要求
- 至少为 TwinCAT 3.1 Build 4024
- 至少为 Windows 10 或 TwinCAT/BSD®（在目标系统上）

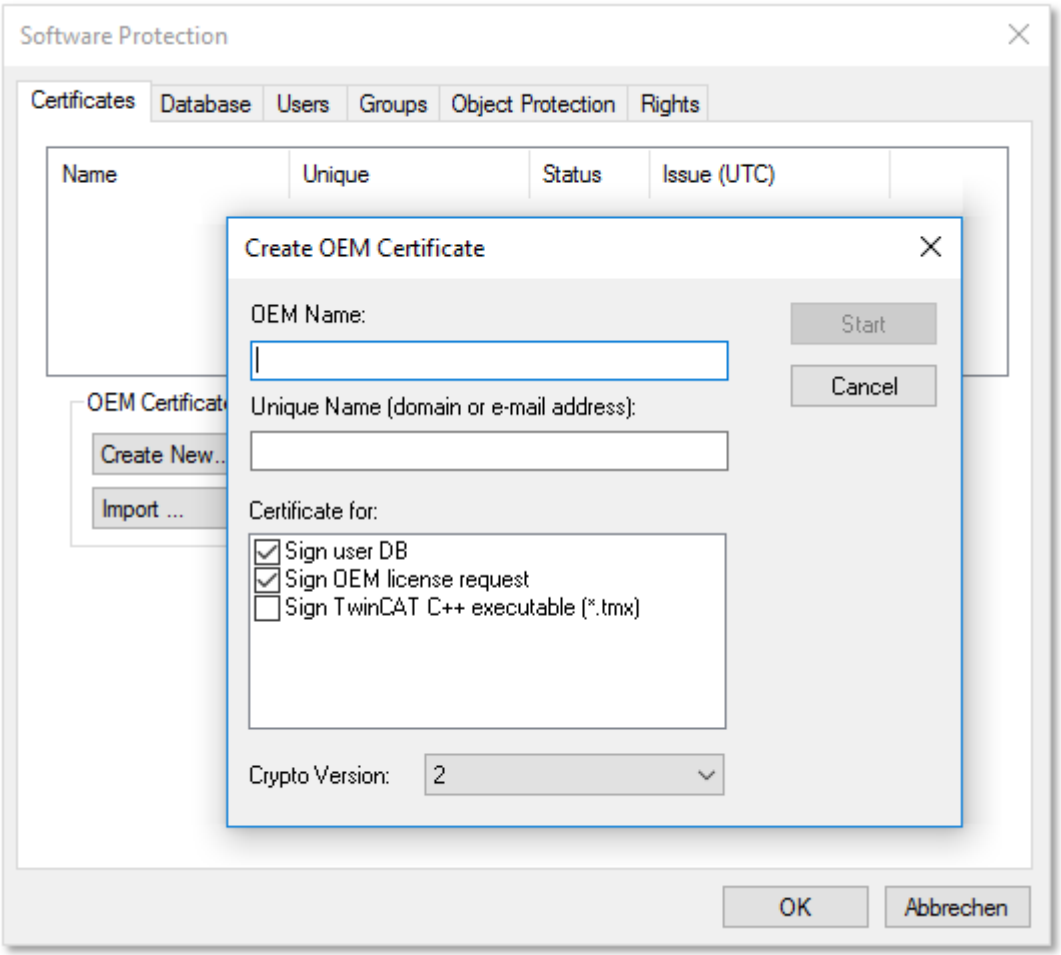
1. 调用软件保护配置器。为此，请在主菜单中选择 **TwinCAT** 项下面的菜单项**软件保护**：



2. 在打开的窗口中，选择**证书**选项卡。
点击**新建.....**：

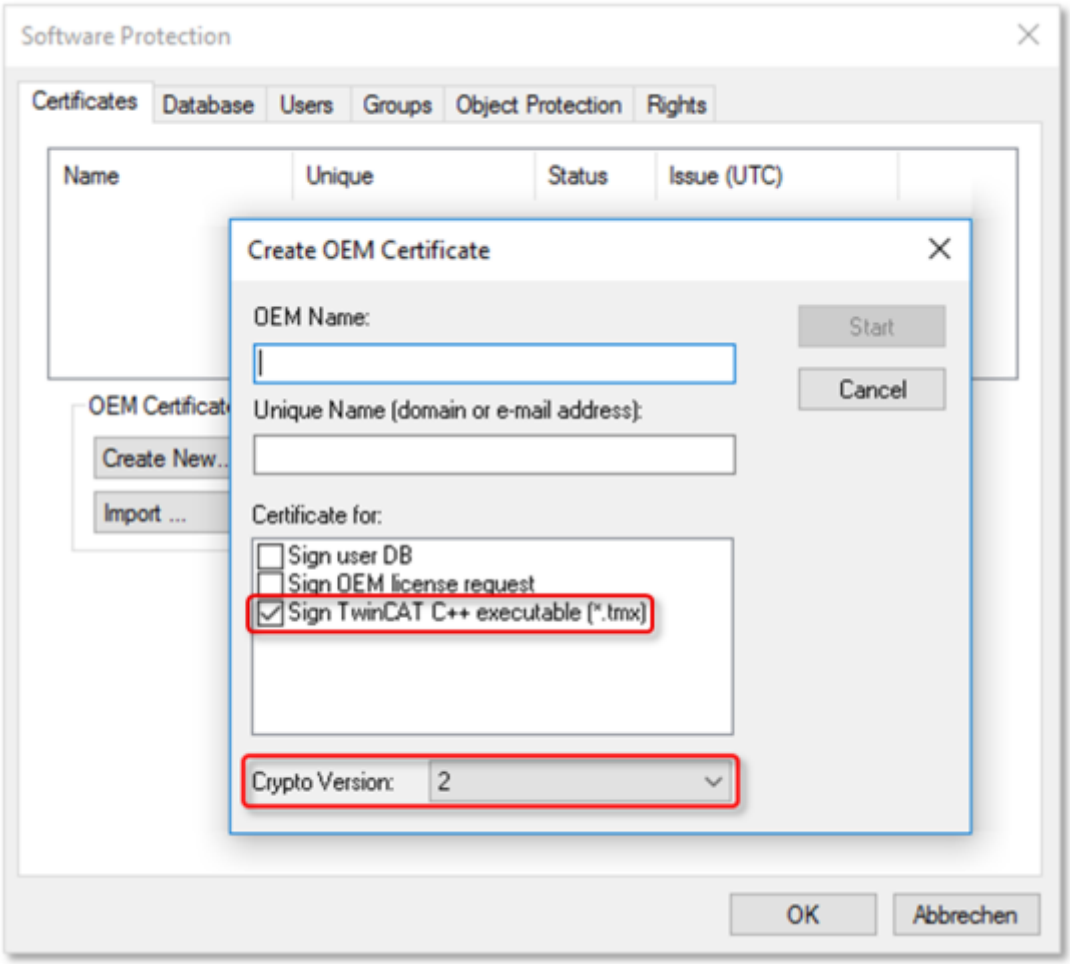


3. **创建 OEM 证书**输入窗口打开：



4. 输入所需数据：
   - 在 **OEM 名称**文本框中输入贵公司名称。该名称必须明确提及贵公司或您的业务单位。
   - 输入一个**唯一名称**。“OEM 唯一名称”必须是全球范围内可唯一标识证书所有者的名称，最好是贵公司网站的 URL 或您的电子邮件地址。电子邮件地址必须是公司的电子邮件地址，也就是说，必须能够明确将其分配给一家公司。

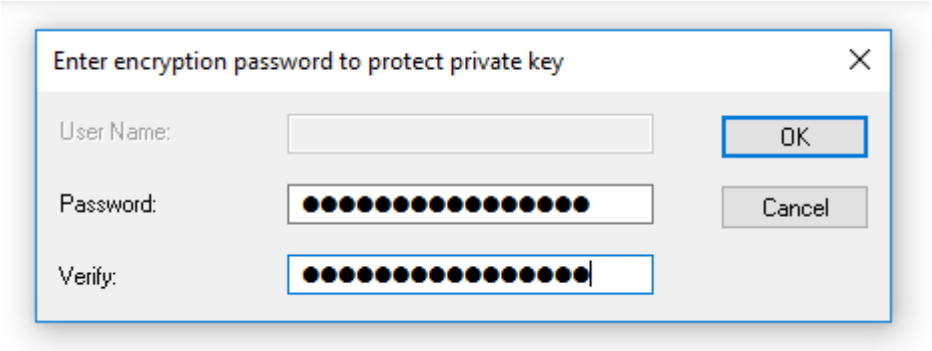
- 选中复选框**签署 TwinCAT C++ 可执行文件**：



如果只想用该证书签署 TwinCAT 驱动程序软件，则取消选中其他两个复选框。（这些仅用于 PLC 区域）

- 确保已设置加密版本 2（用于证书内容的加密）。（默认设置）

5. 输入数据之后，点击**开始**，并选择一个目录来保存文件。
   可以简单地接受建议的目录“c:\twincat\3.1\customconfig\certificates”。需要将新创建的文件保存在该目录下，以便能够在后续步骤中读取该文件的“文件指纹” [▶ 29]。
   ⇨ 出现一个用来为 OEM 私钥选择密码的对话框。
6. 设置一个 OEM 私钥的密码。
   密码安全 - 请确保为您的证书设置高强度的密码！采取适当措施保护密码，防止泄露！
   密码丢失后无法恢复，倍福无法恢复或重置您的密码。如果忘记或丢失了证书密码，则无法再使用证书，必须申请新的证书。
7. 再次输入密码进行确认，然后点击**确定**，关闭对话框。



⇨ 文件被保存。

以这种方式生成的“证书请求文件“现在必须经倍福证书部门签名才能生效。该程序的说明详见申请证书 [▶ 25]章节。

5.2.1.2.3 确定 OEM 证书文件的文件指纹

需要这项功能来申请 **TwinCAT OEM 证书扩展验证**（TC0008）。

系统要求

这项功能需要 TwinCAT 3.1 Build 4024 以上版本。

倍福一旦签名，“OEM 证书申请文件“即成为 TwinCAT OEM 证书。除此签名之外，这些文件没有任何区别。因此，在以下章节中，“TwinCAT OEM 证书文件”这一术语用于两个文件版本。

通过 TwinCAT 3 Engineering 读取 OEM 证书文件的“文件指纹”。

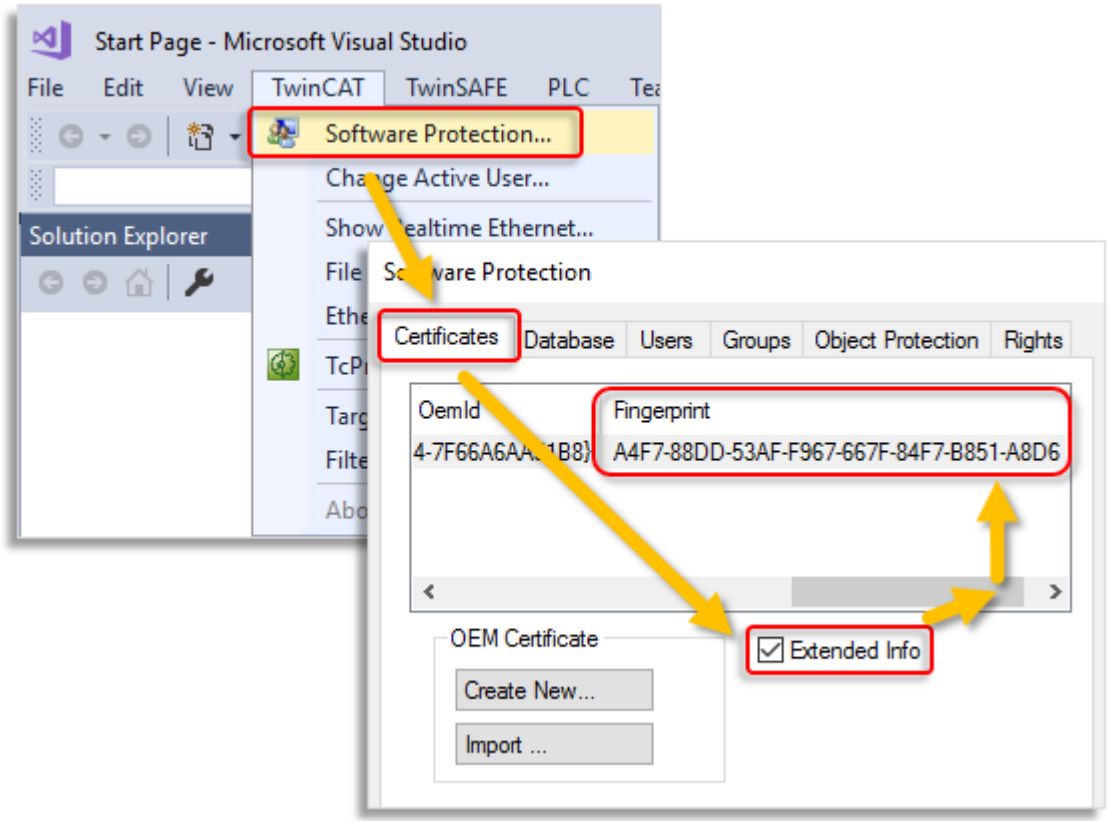
为了能使用此功能，OEM 证书文件必须存放在以下文件夹中：“c:\twincat\3.1\customconfig\certificates”。

如果您已经有证书并想更新证书，那么证书需要被保存在这个目录下。

如果在创建“OEM 证书请求文件“时没有改变建议的文件夹，则文件已在该文件夹中。

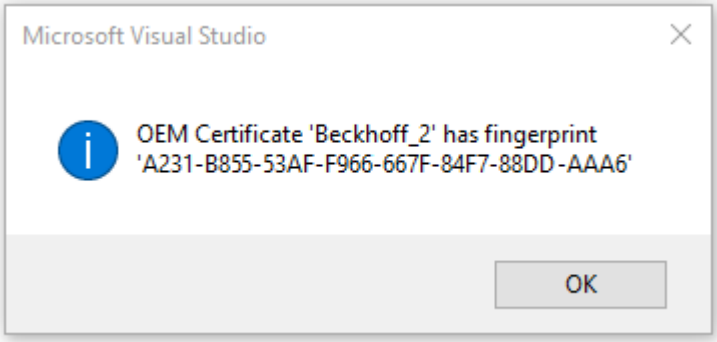
程序：

1. 调用 TwinCAT 3 软件保护配置器。



2. 选择**证书**（Certificates）选项卡。
3. 勾选**扩展信息**（Extended Info）复选框。

4. 在窗口中向右滚动，直至看到**指纹**（Fingerprint）列。或者，只双击证书行即可。然后，指纹文件会显示在一个弹出窗口中：



可使用快捷键 [Ctrl] + [C] 将指纹数据从信息窗口复制到 Windows 剪贴板。

5.2.1.2.4 保存已签名的 TwinCAT 用户证书

建议的证书保存目录：**C:\TwinCAT\3.1\CustomConfig\Certificates**

系统要求

- 至少为 TwinCAT 3.1 Build 4024
- 至少为 Windows 10 或 TwinCAT/BSD®（在目标系统上）

使用 TwinCAT 3 TMX 文件不需要 TwinCAT 3 用户证书

TwinCAT 3 用户证书仅用于 TMX 文件的一次性签名，使用已通过该证书签名的 TMX 文件时不需要该证书。

哪些计算机需要 TwinCAT 3 用户证书 TC0008?

TwinCAT 3 用户证书应仅位于对 TMX 文件签名的编程计算机上，**而不是**每个目标系统上。

5.2.2 操作系统

注意

建议使用 TwinCAT 加载器迁移至 TMX

由于 TwinCAT 3.1 4024.0 版本的 C++ 项目可用，其二进制文件可直接从 TwinCAT 中加载。建议进行迁移！

在 x64 平台上实现 TwinCAT 3 C++ 模块时，如果要由操作系统加载驱动程序（*.sys 文件），则必须使用证书签名。

该签名在 TwinCAT 3 生成过程中自动执行，被 64 位 Windows 操作系统用于驱动程序的身份验证。

需要使用证书签名驱动程序。本 Microsoft 文档介绍了获取 64 位 Windows 操作系统接受的测试和发布证书的相关流程和背景知识。

要在 TwinCAT 3 中使用此类证书，请按照“为测试模式创建测试证书 [▶ 31]”中的说明，在编译 x64 版本目标后配置该步骤。

测试证书

出于测试目的，可以创建和使用自签名测试证书，不受技术限制。

以下教程将介绍如何启用该选项。
要为生产机器创建具有真实证书的驱动程序，必须禁用该选项。

- 为测试模式创建测试证书 [▶ 31]
- 删除（测试）证书 [▶ 33]

更多参考案例：

MSDN、MakeCert 测试证书（Windows 驱动程序）、
https://docs.microsoft.com/en-us/windows-hardware/drivers/install/makecert-test-certificate

5.2.2.1 测试签名

概述

为 x64 平台实现 TwinCAT 3 C++ 模块需要用证书对驱动程序进行签名。

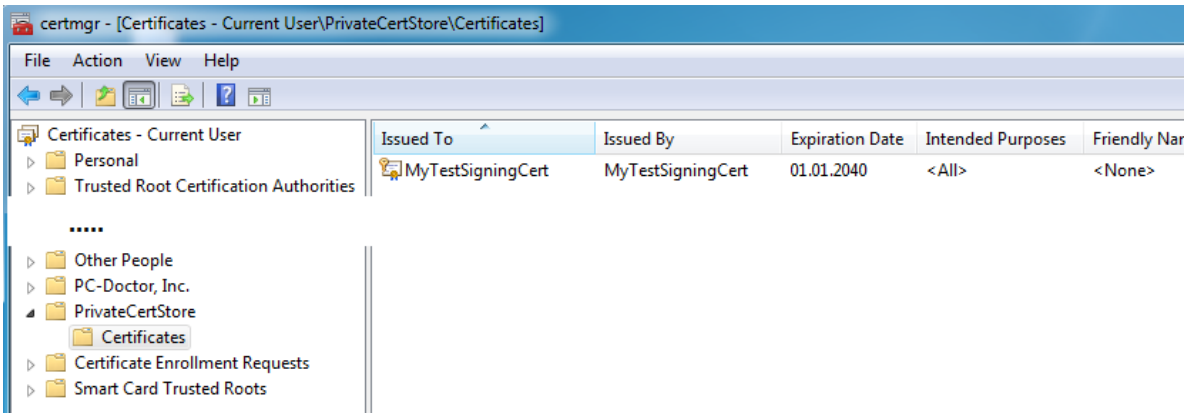
本文介绍如何创建和安装用于测试 C++ 驱动程序的测试证书。

请注意创建测试证书时的程序
开发人员可以使用多种工具自行创建证书。请完全按照此说明激活测试证书机制。

必须从任意一种方式打开的命令行中执行以下命令：

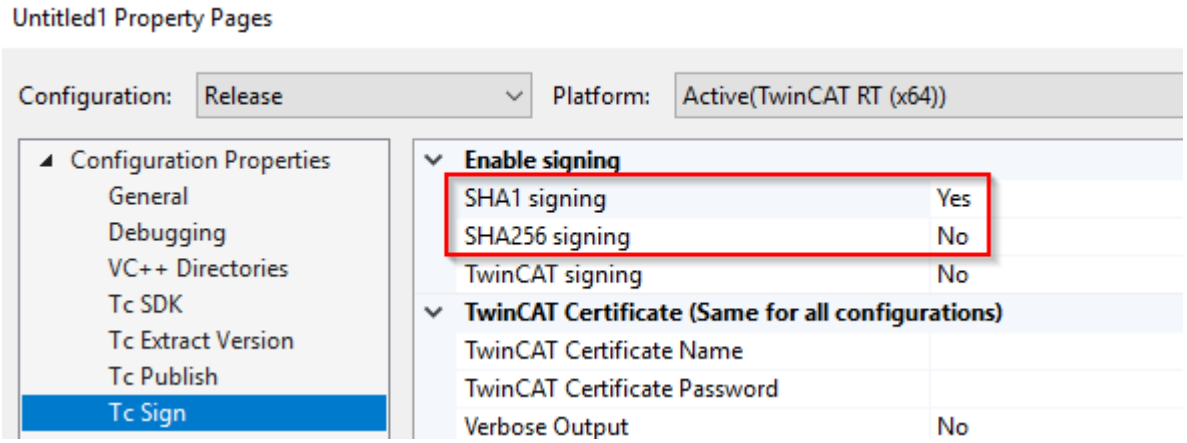
- **具有管理员权限的 Microsoft Visual Studio® 2010/2012 命令提示符。**（通过：**所有程序** -> **Microsoft Visual Studio 2010/201**2 -> **Visual Studio 工具** -> Visual Studio 命令提示符，然后右键单击**以管理员身份运行**）
- **具有管理员权限的 Microsoft Visual Studio® 2017/2019 开发人员命令提示符。**（通过：**所有程序** -> **Visual Studio 2017** -> 对比 2017/2019 的 Visual Studio 命令提示符，然后右键单击**以管理员身份运行**）
- 仅在已安装 WINDDK 的情况下：
  具有管理员权限的提示符（**开始** -> 命令提示符），然后切换至目录 %WINDDK7%\bin\x86\，其中包含相应工具。

1. 在 XAE 上：
   在工程系统中，在 Microsoft Visual Studio® 2010/2012 命令提示符下以管理员权限输入以下命令（见上文注释）：
   ```
   makecert -r -pe -ss PrivateCertStore -n CN=MyTestSigningCert
   MyTestSigningCert.cer
   ```
   （如果没有 PrivateCertStore 的访问权限，可以使用其他存储位置。还必须在 PostBuild 事件中 [▶ 34]**使用。）**
   ⇨ 然后创建自签名证书，该证书存储在“MyTestSigningCert.cer”文件和 Windows Certificate Store 中。
   ⇨ 使用 mmc 验证结果（**使用文件** -> **添加/删除管理单元** -> **证书**）：



2. 在 XAE 上：
   配置证书，使其能被工程系统上的 TwinCAT XAE 识别。
   在工程系统中将环境变量 TWINCATTESTCERTIFICATE 设为“MyTestSigningCert”，或编辑 Debug|TwinCAT RT (x64) 和 Release|TwinCAT RT (x64) 的构建后事件。
   变量的名称不是证书文件的名称，而是 CN 名称（在本例中为 MyTestSigningCert）。

从 TwinCAT 3.1 4024.0 开始，在项目属性中的 Tc Sign 下执行待用证书的配置。如本文所述，要通过操作系统使用签名，请注意项目设置。



在 XAR（和 XAE，若为本地测试）上，启用测试模式，以便操作系统可以接受自签名证书。可以在工程系统 (XAE) 和运行时系统 (XAR) 上完成操作。

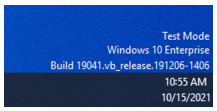
适用于 Windows

利用管理员提示执行以下操作：

```
bcdedit /set testsigning yes
```

并重启目标系统。
为此，必须关闭“SecureBoot”，可在 BIOS 中完成操作。

如果已启用测试签名模式，则会在桌面右下方显示。现在，系统接受所有已签名的执行驱动程序。



完成相应程序后，系统会接受所有已签名的执行驱动程序。

1. 测试在目标系统上能否启用和开启在 TwinCAT C++ 驱动程序中实现的 TwinCAT 模块配置。

⇨ 编译 x64 驱动程序会产生以下输出：

```
Output
Show output from: Build
1>------ Build started: Project: Untitled2, Configuration: Debug TwinCAT RT (x64) ------
1>  header file << C:\TwinCAT\3.1\SDK\\_products\TwinCAT RT (x64)\Debug\Untitled2\\Untitled2Version.h >> is up-to-date!
1>  TcPch.cpp
1>  Module1.cpp
1>  Untitled2ClassFactory.cpp
1>  Untitled2Driver.cpp
1>  Untitled2.vcxproj -> C:\TwinCAT\3.1\SDK\\_products\TwinCAT RT (x64)\Debug\Untitled2.sys
1>  The following certificate was selected:
1>      Issued to: MyTestSigningCert
1>
1>      Issued by: MyTestSigningCert
1>
1>      Expires:   Sun Jan 01 00:59:59 2040
1>
1>      SHA1 hash: E27A66E6A0C7BC0C86DFDD093DDF2486D1EE502E
1>
1>
1>  Done Adding Additional Store
1>  Successfully signed and timestamped: C:\TwinCAT\3.1\SDK\\_products\TwinCAT RT (x64)\Debug\Untitled2.sys
1>
1>
1>  Number of files successfully Signed: 1
1>
1>  Number of warnings: 0
1>
1>  Number of errors: 0
1>
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

参考案例：
MSDN、MakeCert 测试证书（Windows 驱动程序）、
https://docs.microsoft.com/en-us/windows-hardware/drivers/install/makecert-test-certificate

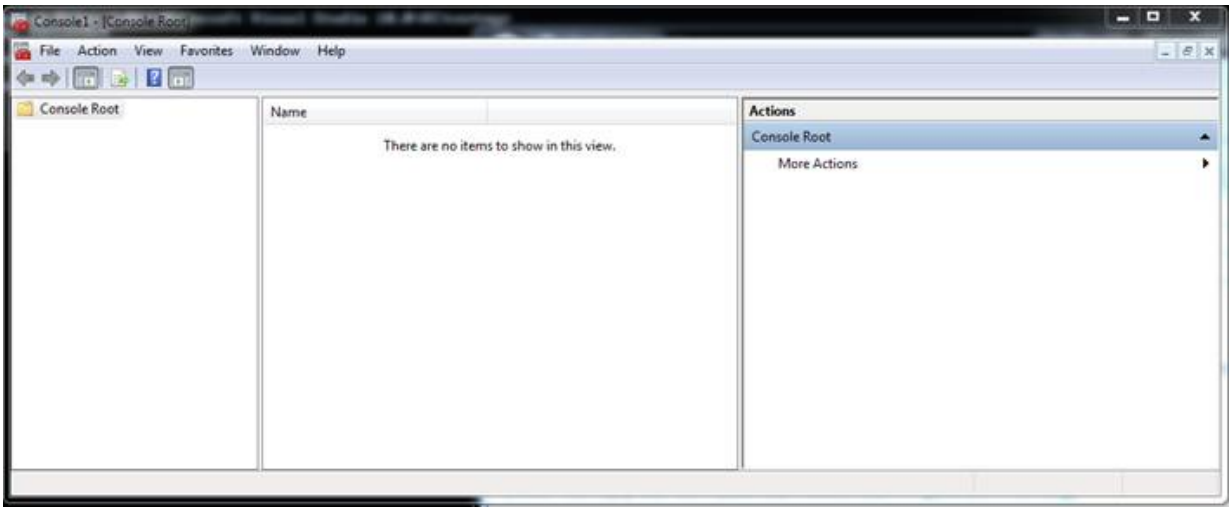
5.2.2.2 删除测试证书

本文介绍如何删除测试证书。

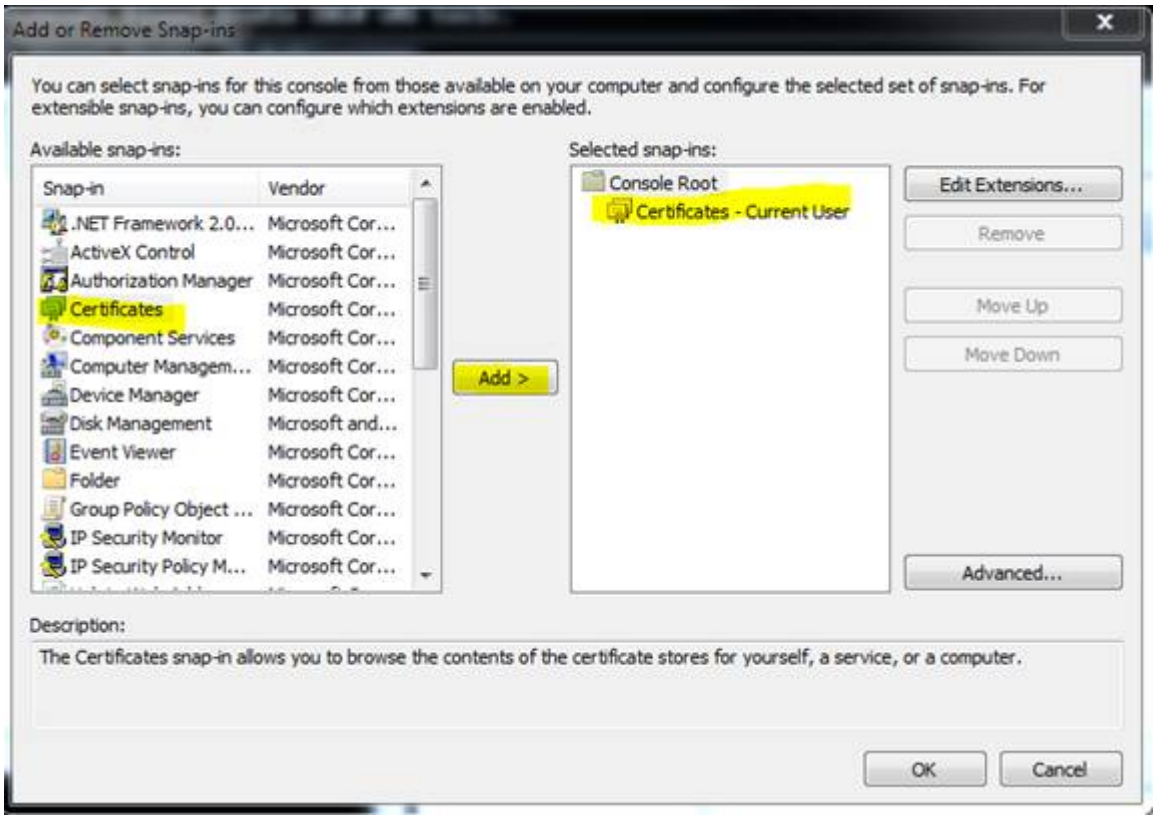
概述

可通过 Microsoft Management Console 删除证书：

1. 通过“开始”菜单或用户界面启动管理控制台 MMC.exe。

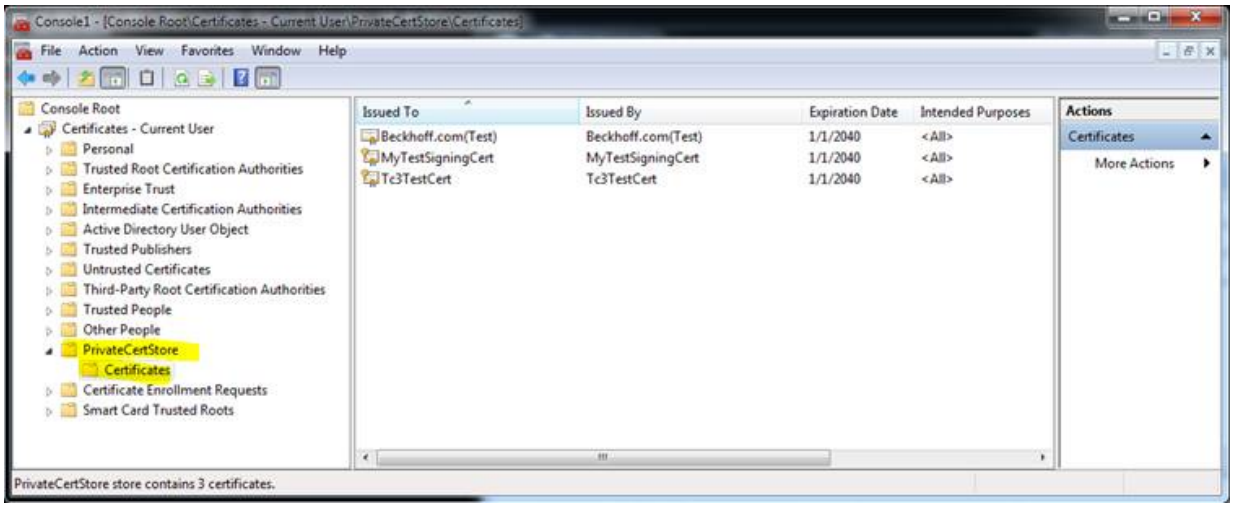


2. 点击菜单中的**文件** -> **添加/删除管理单元**，选择当前用户的证书管理单元，最后点击**确定**。



⇨ 证书列在 **PrivateCertStore/Certificates** 下的节点中。

3. 选择要删除的证书。



5.2.2.3 无测试模式的 Windows 驱动程序

对于 Windows 操作系统，必须通过“认证签名”对驱动程序进行签名。这需要 EV 证书。
Microsoft 提供相关说明：https://docs.microsoft.com/en-us/windows-hardware/drivers/dashboard/attestation-signing-a-kernel-driver-for-public-release
以这种方式创建的驱动程序也适用于已启用安全启动的设备。

Microsoft 宣布，从 2021 年 7 月起将停用先前使用 CrossSigning 证书的程序（带有参数/ac 的签名工具）。不得再使用该程序（取决于各个 CrossSigning 证书的有效期）。
通过 TwinCAT 加载器 [▶ 49]加载的版本控制 C++ 项目已经可供 TwinCAT C++ 使用一段时间，因此它们并不属于操作系统意义上的驱动程序。因此，倍福建议使用版本控制C++ 项目。

有关将 TwinCAT C++ 驱动程序迁移到版本控制 C++ 项目中的操作，已在“如何操作”部分提供指南。

6 模块

TwinCAT 模块概念是现代机器模块化的核心要素之一。本章介绍的是模块概念和模块的使用。

模块概念适用于所有 TwinCAT 模块，而不仅仅是 C++ 模块，尽管大多数细节仅涉及 C++ 模块的工程设计。

6.1 TwinCAT 组件对象模型 (TcCOM) 概念

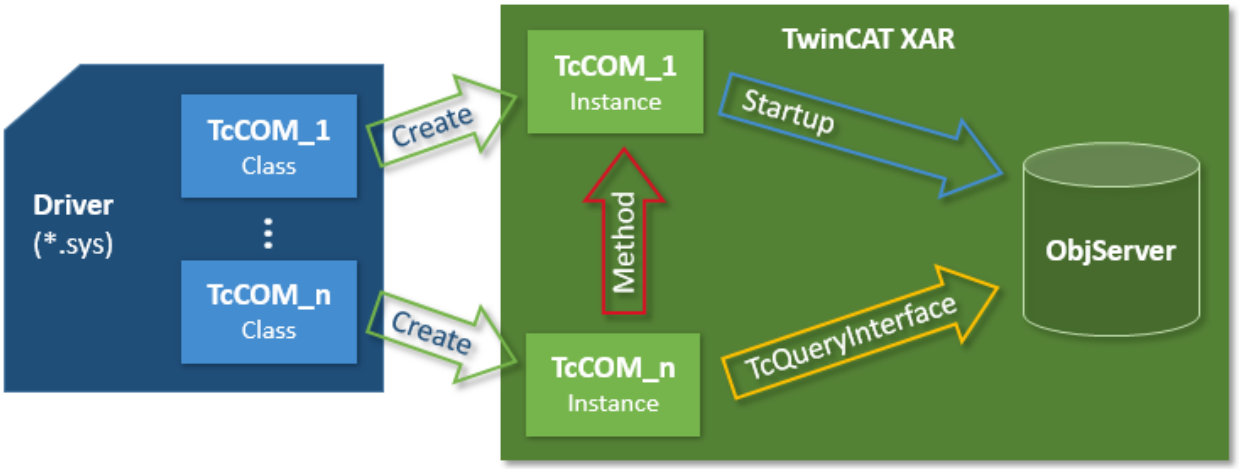
TwinCAT 组件对象模型定义了模块的特性和行为。该模型源自 Microsoft Windows 的“组件对象模型”(COM)，功能说明了各种独立开发和编译的软件组件之间的协作的方式。为了达成这一目标，必须明确界定模块的行为模式并遵循其接口规范，使其能够相互作用。例如，这种接口还可以便捷的实现不同制造商开发模块之间的交互。

在某种程度上，TcCOM 以 COM（Microsoft Windows 体系的组件对象模型）为基础，尽管只使用 COM 的一个子集。然而，与 COM 相比，TcCOM 包含更多超出 COM 的定义，例如状态机模块。

TcCOM 模块概述和应用

本介绍性概述旨在使各个主题更容易理解。

一个或多个 TcCOM 模块集成在一个驱动程序中。该驱动程序由 TwinCAT 工程使用 MSVC 编译器创建。TMC（TwinCAT 模块类）文件对模块和接口进行了说明。如今，驱动程序及其 TMC 文件可以在工程系统之间进行交换和组合。



现在可以使用Twincat工程工具创建这些模块的实例。它们与 TMI 文件相关联。这些实例可进行参数设置，并且可以相互关联，或与其他模块组合形成 I/O系统。相应的配置会传输到目标系统，并在该系统上执行。

相应的模块会被启动，并在 TwinCAT ObjectServer 上注册。TwinCAT XAR 还会提供过程映像。模块可以查询 TwinCAT ObjectServer，以获得与特定接口有关的另一个对象的引用。如果有这样的引用，便可在模块实例上调用接口方法。

以下章节将对各个主题进行论证。

ID 管理

不同类型的 ID 用于模块之间以及模块内部的交互。TcCOM 使用 GUID（128 位）和 32 位长整数。

TcCOM 使用

- GUID 作为：ModulID、ClassID 和 InterfaceID。
- 32 位长整数用于：ParameterID、ObjectID、ContextID、CategoryID。

接口

COM 的一个重要组成部分是接口，TcCOM 也是如此。

接口定义了一组方法，这些方法组合在一起可以执行某项任务。接口通过唯一的 ID (InterfaceID) 进行引用，只要接口不改变，这个 ID 就永远不能修改。该 ID 使模块能够决定其是否能与其他模块协作。同时，如果接口定义明确，那么开发过程可以独立进行。因此，修改接口会导致生成不同的 ID。TcCOM 理念旨在使 InterfaceID 可以叠加其他（旧的）InterfaceID（TMC 功能说明/TMC 编辑器中的“隐藏项”）。这样，两个版本的接口均可使用，而另一方面，同时始终明确最新InterfaceID。同样的概念也适用于数据类型。

TcCOM 已经已定义一系列的接口，这些接口在某些情况下是预先规定的（如 ITComObject），但在大多数情况下是可选的。许多接口只有在某些应用领域才有意义。其他接口则为通用型，通常可以重复使用。系统支持用户自定义接口，例如可实现两个第三方模块之间的相互交互。

- 所有接口都源自基本接口 ItcUnknown，该接口与 COM 的IUnknown类似，接口查询服务 (TcQueryInterface) 和模块生命周期管理（TcAddRef 和 TcRelease）提供基本服务。
- 每个模块都必须实现 ITComObject 接口，该接口包含用于访问模块名称、ObjectID、父模块的 ObjectID、参数和状态机的方法。

许多模块都使用多个通用接口：

- ITcCyclic接口由需周期性调用的模块实现（"CycleUpdate"）。该模块可通过 TwinCAT 任务的 ITcCyclicCaller 接口注册，以获取周期调用。
- ITcADI 接口可用于访问模块的数据区。
- 默认情况下，ITcWatchSource 已实现；它为 ADS DeviceNotification 和其他功能提供了便利。
- ITcTask 接口由实时系统的任务实现，提供有关周期时间、优先级和其他任务信息。
- ITComObjectServer 接口由 ObjectServer 实现，并被所有模块引用。

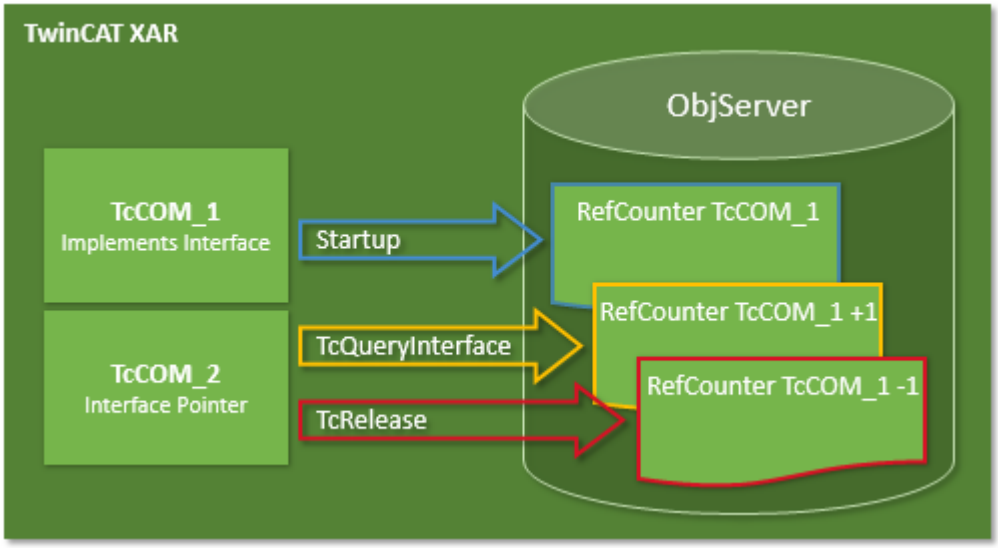
倍福已提供整个系列的通用接口。通用接口的优点是可以支持模块的交换和循环使用。只有在没有适当通用接口的情况下，才应定义自己的接口。

类工厂

“类工厂”用于在 C++ 中创建模块。驱动程序中包含的所有模块都有一个共同的类工厂。类工厂向 ObjectServer 注册一次，并为创建某些模块类提供服务。模块类由模块的唯一 ClassID 标识。当 ObjectServer 请求新模块时（基于配置器的初始化数据或在运行时通过其他模块），模块会根据 ClassID 选择合适的类工厂，并通过其 ITcClassFactory 接口触发模块的创建。

模块生命周期管理

与 COM 类似，模块的生命周期通过引用计数器 (RefCounter) 来确定。每次查询模块接口时，引用计数器都会递增。释放接口时，计数器会递减。当模块登录 ObjectServer（ITComObject 接口）时，也会查询接口，这样引用计数器至少为 1。注销时计数器递减。当计数器为 0 时，模块会自动删除，通常是在注销 ObjectServer 之后。如果另一个模块已维护一个引用（具有一个接口指针），则该模块继续存在，而接口指针保持有效，直至释放该指针。



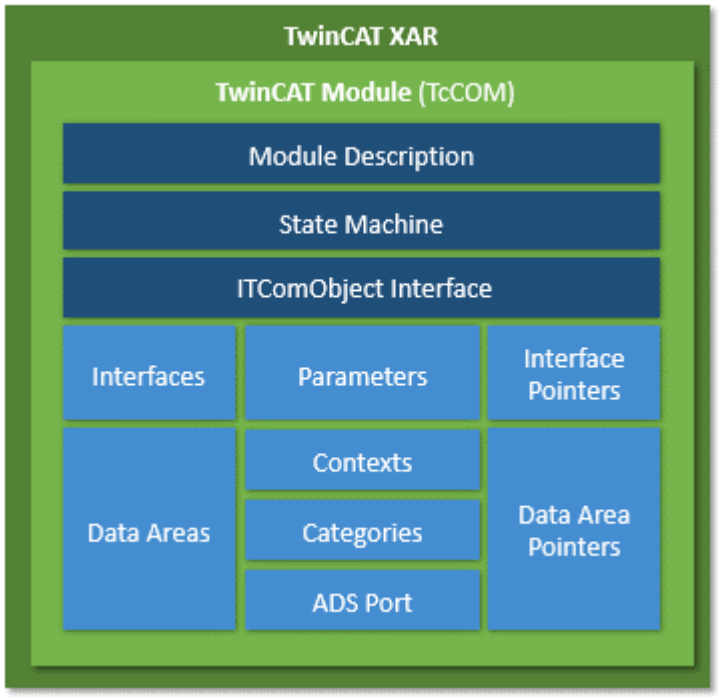
6.1.1 TwinCAT 模块属性

TcCOM 模块包含一组形式化定义的属性，分为强制属性与可选属性两类。这些属性通过标准化定义实现互换应用。每个模块都具有模块说明，用于功能说明模块属性。它们用于配置模块及其相互之间的关系。

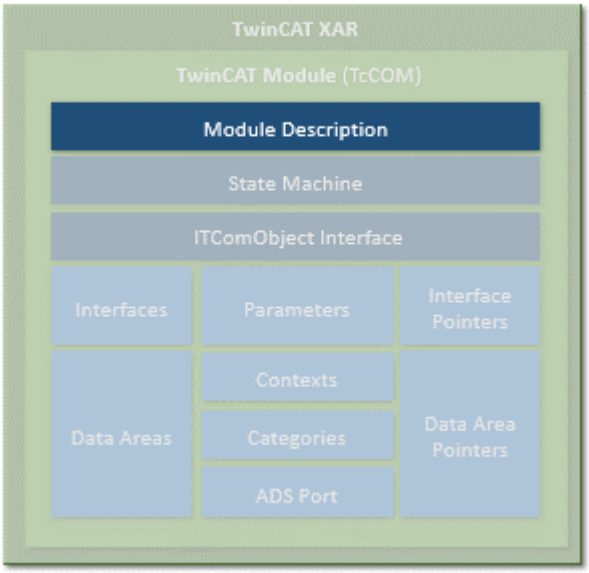
如果模块在 TwinCAT 运行时中实例化，它就会在中央系统实例 (ObjectServer) 中自行注册。这样，其他模块和通用工具便可对其进行访问和参数设置。模块可以独立编译，因此也可以独立开发、测试和更新。模块可以非常简单，例如可能仅包含低通滤波器等基本功能。或者其内部也可以非常复杂，包含机器子组件的整个控制系统。

模块的应用非常广泛；自动化系统的所有任务都可以在模块中指定。因此，对于主要代表自动化系统基本功能（如实时任务、现场总线驱动程序或 PLC 运行时系统）的模块，以及用于控制机器单元的特定用户或应用算法，不作任何区分。

下图显示的是常见 TwinCAT 模块及其主要属性。深蓝色块定义了强制属性，而浅蓝色块则定义了可选属性。



模块说明



每个 TcCOM 模块都有一些一般功能说明参数。其中包括一个 ClassID，可以明确引用模块类。由相应的 ClassFactory 实例化。每个模块实例都有一个 ObjectID，该 ID 在 TwinCAT 运行时中是唯一的。此外，还有一个父级 ObjectID，指的是可能的逻辑父对象。

通过 ITComObject 接口可以获取下文所述模块的信息、状态机和参数（请参见“接口”章节）。

类功能说明文件 (*.tmc)

模块类别信息集路在类功能说明文件（TwinCAT Modole Class；*.tmc）中。

开发人员使用这些文件来功能说明模块属性和接口，以便其他人调用和集成模块。除常规信息（供应商数据、模块类 ID 等）之外，文件中还会功能说明模块的可选属性。

- 支持类别
- 已实现的接口
- 带有相应符号的数据区
- 参数
- 接口指针
- 可以设置的数据指针

系统配置器主要将类功能说明文件用作以下操作的基础：将模块实例整合到配置中，指定参数并配置与其他模块的关联。

同时包括模块中所有数据类型的功能说明，然后由配置器在其通用数据类型系统中采用。这样，系统中存在的 TMC 功能说明的所有接口都可以被所有模块使用。

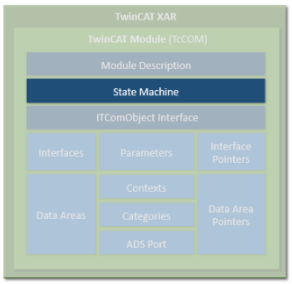
涉及多个模块的较复杂配置也可在类功能说明文件中有所功能说明，这些文件已针对特定应用进行预配置和关联。因此，在开发阶段，内部由多个子模块组成的复杂机器单元的模块，可以作为实体进行定义和预配置。

实例功能说明文件 (*.tmi)

特定模块的实例在实例功能说明文件（TwinCAT Moudule Instance；*.tmi）中有所功能说明。实例功能说明文件采用与类功能说明类似的格式，但与类功能说明文件不同，实例功能说明文件已包含项目中特殊模块实例的参数、接口指针等具体规范。

实例功能说明文件由 TwinCAT 工程 (XAE) 在为特定项目创建类功能说明实例时创建。其主要用于配置涉及的所有工具之间的数据交换。然而，实例功能说明也可以跨项目使用，例如，在新项目中再次使用专门进行参数设置的模块。

状态机

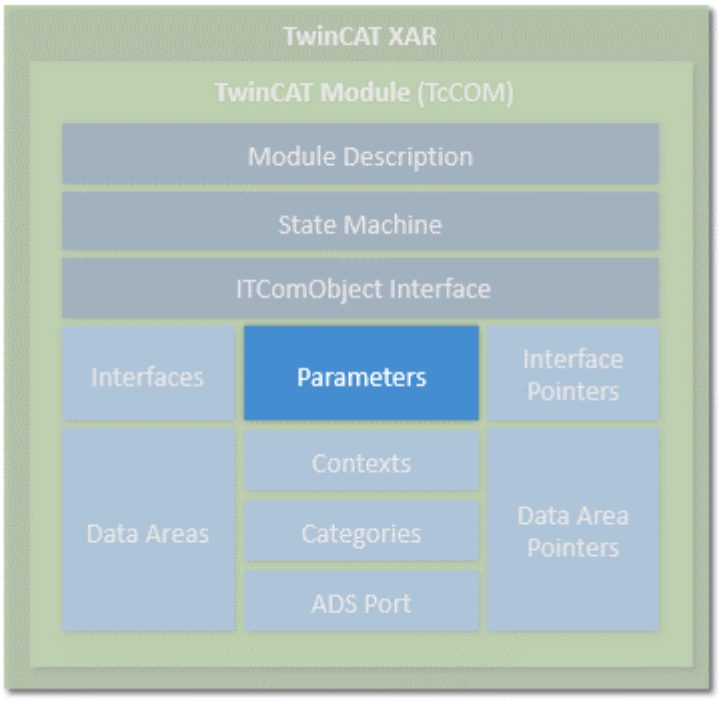


每个模块都包含状态机，该机器功能说明的是模块的初始化状态以及从外部修改该状态的方法。状态机功能说明的是模块启动和停止时的状态。其中涉及模块创建、参数设置以及与其他模块的协同生产。

应用特定状态（如现场总线或驱动程序的状态）可以用各自的状态机来功能说明。TcCOM 模块的状态机定义了 INIT、PREOP、SAFEOP 和 OP 状态。虽然状态名称与 EtherCAT 现场总线的状态名称相同，但实际状态却不同。当 TcCOM 模块为 EtherCAT 执行现场总线驱动程序时，它配有两个状态机（模块和现场总线状态机），且二者需按顺序执行。模块状态机必须达到运行状态 (OP) 后，现场总线状态机才能启动。

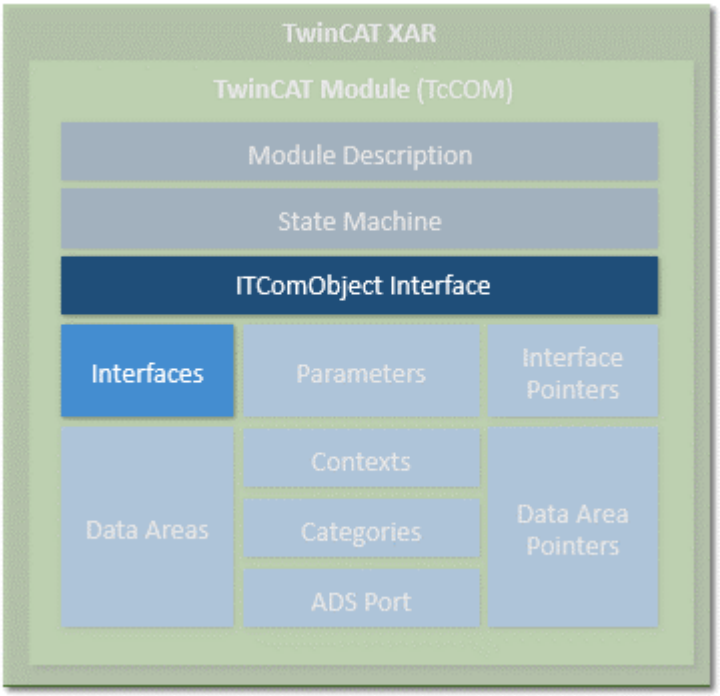
状态机的详细介绍 [▶ 43]将单独阐述。

参数



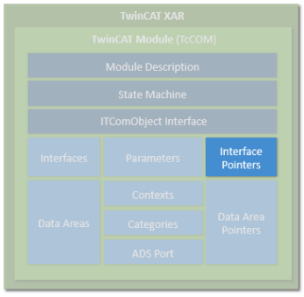
模块可以包含参数，这些参数可以在初始化时或之后的运行时（OP 状态）读取或写入。每个参数都有一个参数 ID。参数 ID 的唯一性可以分为三种类型：全局性、受限全局性或模块特定性。有关更多详情，请参见“ID 管理”章节。除参数 ID 外，参数还包含当前数据；数据类型取决于参数，并根据相应的参数 ID 明确定义。

接口



接口由一组明确方法（函数）组成，这些方法为模块提供交互通道，使其他模块能够与该模块建立通信。如上所述，接口具有唯一 ID。模块必须至少支持 ITComObject 接口，此外还可根据需要包含多个接口。通过调用“TcQueryInterface”方法，并指定相应的接口 ID，即可查询接口引用。

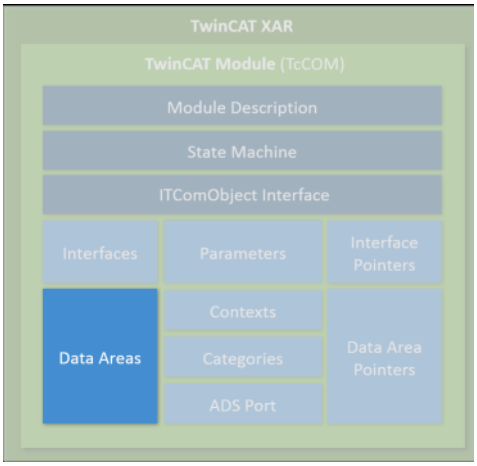
接口指针



接口指针可视为接口的对应交互对象。如果模块要使用另一个模块的接口，则必须具有相应接口类型的接口指针，并确保其指向目标模块。之后方可调用目标模块的方法。

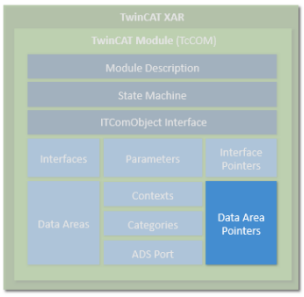
接口指针通常会在状态机启动时设置。在 INIT 向 PREOP (IP) 转换期间，模块通过相应接口接收其他模块的对象 ID；在 PREOP 向 SAFEOP (PS) 或 SAFEOP 向 OP (SO) 转换期间，通过 ObjectServer 搜索其他模块的实例，并通过方法查询接口设置相应接口。在状态逆向切换时，即从 SAFEOP 到 PREOP (SP) 或从 OP 到 SAFEOP (OS) 期间，必须再次启用接口。

数据区



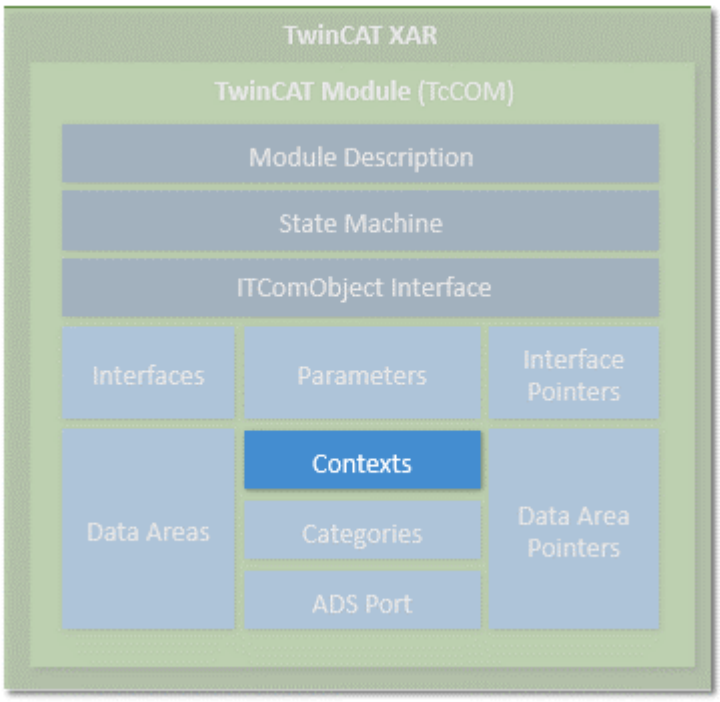
模块可以包含数据区，这些数据区可供环境（如其他模块或 TwinCAT 的 IO 区域）调用。这些数据区可以储存任何数据。它们通常用于过程映像数据（输入和输出）。数据区的结构在模块的设备功能说明中定义。如果模块拥有数据区，并希望其他模块也能访问这些数据区，那么该模块就需要实现 ITcADI 接口，以便能够访问这些数据。数据区可以包含符号信息，这些信息更详细地功能说明了相应数据区的结构。

数据区指针



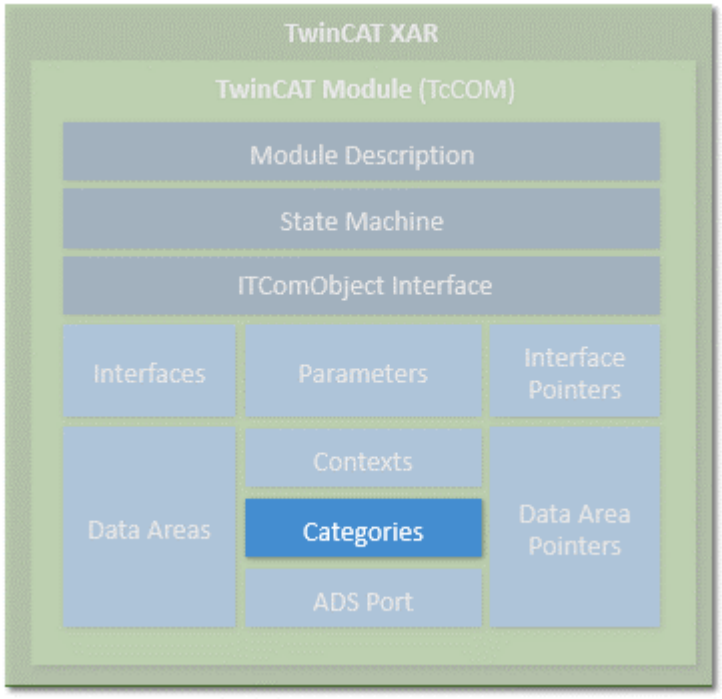
如果模块要访问其他模块的数据区，则可包含数据区指针。这些通常在状态机初始化时指向其他模块的数据区或数据区段。访问权限为直接进入内存区，因此必要时因此必要时需实现相应的保护机制，以应对并发访问冲突。在许多情况下，最好使用相应的接口。

环境



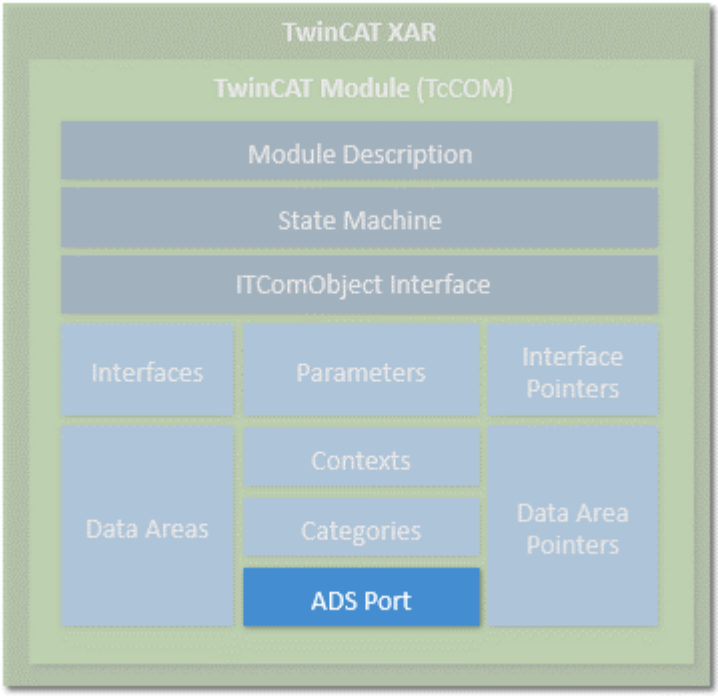
环境应视为实时任务环境。例如，在配置模块时需要环境。简单模块通常在单一时间环境中运行，因此无需详细说明。其他模块可能在多个环境中部分处于活动状态（例如，EtherCAT 主站可以支持多个独立的实时任务，或者控制回路可以在另一个循环时间内处理下一层的控制回路）。如果模块具有多个随时间变化的环境，则必须在模块说明中加以说明。

类别



模块可以通过实现接口 ITComObjectCategory 来提供类别。ObjectServer 会对类别进行列举，可通过 ObjectServer (ITComObjectEnumPtr) 来查询与类别相关联的对象。

ADS



在 ObjectServer 中输入的每个模块都可以通过 ADS 访问。例如，ObjectServer 调用模块的 ITComObject 接口，实现读写参数，或访问状态机。此外，还可以实现专用的 ADS 端口，通过该端口接收专用的 ADS 命令。

系统模块

TwinCAT 运行时还可提供许多系统模块，使其他模块也能使用基本的运行时服务。这些系统模块具有固定不变的 ObjectID，其他模块可以通过它来访问。这种系统模块的其中一个示例为实时系统，通过 ITcRTime 接口提供基本的实时系统服务，即生成实时任务。ADS 路由器同时作为系统模块实现，因此其他模块可以在此注册其 ADS 端口。

创建模块

模块可以通过 C++ 和 IEC 61131-3 两种语言创建。创建过程需使用 TwinCAT PLC的面向对象扩展功能。这两种语言模块可以通过接口进行交互，交互方式与纯 C++ 模块相同。面向对象的扩展功能可提供与 C++ 相同的接口。

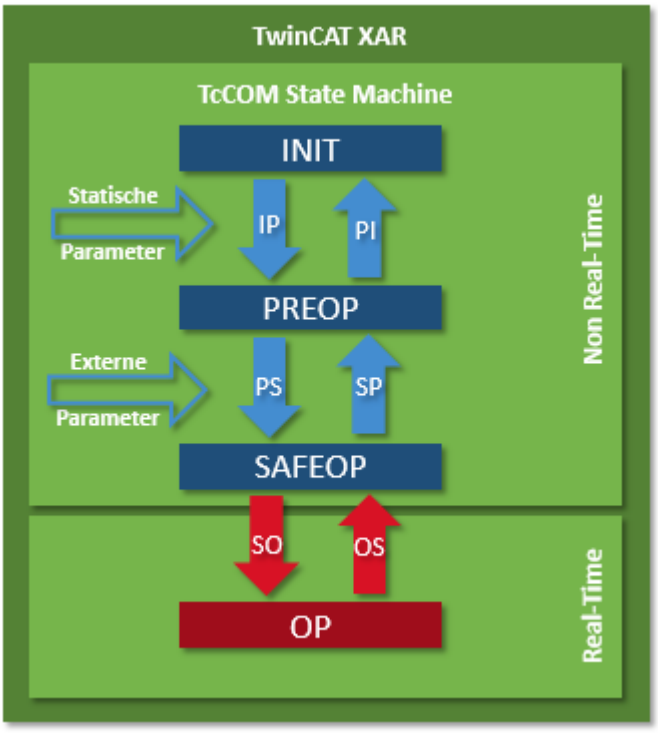
PLC 模块同时通过 ObjectServer 注册，因此可以通过 ObjectServer 访问 PLC 模块。PLC 模块的复杂程度各不相同。无论是只生成小型滤波模块，还是将完整的 PLC 程序打包到一个模块中，都没有区别。由于实现了自动化，每个 PLC 程序都是 TwinCAT 模块意义上的一个模块。所有传统 PLC 程序都会自动封装到一个模块中，并在ObjectServer及一个或多个任务模块中完成注册。对 PLC 模块过程数据的访问（例如与现场总线驱动程序的映射）也可通过定义的数据区和 ITcADI 进行控制。

只要 PLC 程序员决定将 PLC 程序的某些部分明确定义为 TwinCAT 模块，以便可以适当灵活地调用这些模块，那么对于 PLC 程序员来说，这种行为仍然是透明且不可见的。

6.1.2 TwinCAT 模块状态机

除了四种状态（INIT、PREOP、SAFEOP 和 OP）外，还有相应的状态转换，在这些状态转换中，必须执行或可以执行一般或特定模块的操作。状态机的设计简洁。在任何情况下，都只能转换到下一步或上一步。

正向转换：INIT 到 PREOP (IP)、PREOP 到 SAFEOP (PS) 以及 SAFEOP 到 OP (SO)。反向转换，存在以下状态转换：OP 到 SAFEOP (OS)、SAFEOP 到 PREOP (SP) 以及 PREOP 到 INIT (PI)。包括 SAFEOP 状态在内，所有状态和状态转换都在非实时环境中进行。只有从 SAFEOP 到 OP 的转换、OP 状态以及从 OP 到 SAFEOP 的转换是在实时环境中进行的。在分配或启用资源时、模块与其他模块注册或注销注册时具有重要意义。



状态：INIT

INIT 状态只是一种虚拟状态。创建模块后，模块状态立即从 INIT 变为 PREOP，即执行 IP 状态转换。实例化和 IP 状态转换总是同时进行，因此模块绝不会停留在 INIT 状态。只有在移除模块时，模块才会短暂保持 INIT 状态。

转换：INIT 到 PREOP (IP)

在 IP 状态转换期间，模块使用其唯一的 ObjectID 向 ObjectServer 注册。初始化参数也是在创建对象时分配的，并被传输到模块中。在此转换期间，模块无法与其他模块建立连接，因为不清楚其他模块是否已经存在并在 ObjectServer 注册。当模块需要系统资源（如内存）时，可在状态转换期间分配这些资源。在从 PREOP 转换到 INIT (PI) 期间，必须再次释放所有已分配的资源。

状态：PREOP

在 PREOP 状态下，模块创建完成，模块通常已进行完全参数设置，即使在从 PREOP 转换到 SAFEOP 的过程中也可能会进一步添加参数。虽然尚未创建与其他模块的连接，但该模块已在 ObjectServer 中注册。

转换：PREOP 到 SAFEOP (PS)

在这种状态转换中，模块可以与其他模块建立连接。为此，模块通常会收到其他模块的 ObjectID 和初始化数据等，现在通过 ObjectServer 将这些数据转换为与这些模块的实际连接。

转换一般可由系统根据配置器触发，或由其他模块（如父模块）触发。在此状态转换过程中，可以传输更多参数。例如，父模块可以将自己的参数传送给子模块。

状态：SAFEOP

模块仍处于非实时环境中，等待系统或其他模块切换到 OP 状态。

转换：从 SAFEOP 转换到 OP (SO)

从 SafeOP 到 OP 的状态转换、状态 OP 以及从 OP 到 SAFEOP 的转换都是在实时环境中进行的。可能不会再分配系统资源。但其他模块现在可以申请资源，模块也可以向其他模块注册，例如在任务执行期间获得周期性调用。

该转换过程不允许执行长时间运行的任务。例如，文件操作应在 PS 期间执行。

状态：OP

在 OP 状态下，模块开始工作，在 TwinCAT 系统中处于完全激活状态。

转换：OP 到 SAFEOP (OS)

这种状态转换是在实时环境中进行的。SO 转换的所有操作均会撤销，SO 转换期间申请的所有资源都会再次释放。

转换：从 SAFEOP 到 PREOP (SP)

PREOP→SAFEOP （PS ）转换的所有操作均会撤销，PS 转换期间申请的所有资源都会再次释放。

转换：从 PREOP 到 INIT (PI)

IP 转换的所有操作均会撤销，IP 转换期间申请的所有资源都会再次释放。模块从 ObjectServer 注销，通常会触发自销毁流程（请参见“Service life”）。

6.2 模块间通信

TcCOM 模块可以相互通信。本文旨在概述各种选项。模块间通信有四种方法：

- IO 映射（输入/输出符号的链接）
- IO 数据指针
- 通过接口的方法调用
- ADS

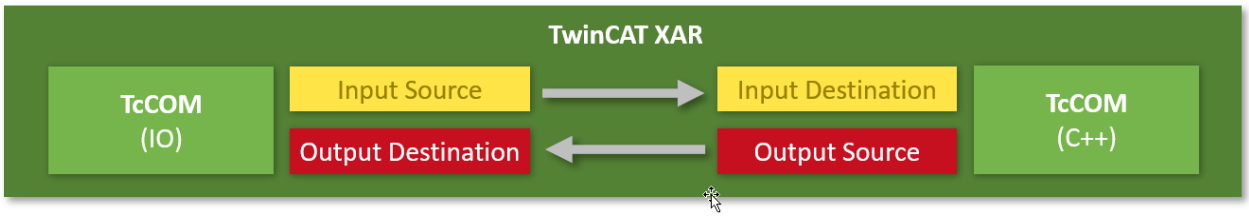
下面将介绍这四种方法。

IO 映射（输入/输出符号的链接）

TcCOM 模块的输入和输出可通过 IO 映射进行关联，关联方式与现场总线层级的物理符号关联方式相同。为此，会在 TMC 编辑器中创建数据区 [▶ 117]，其中功能说明相应的输入/输出。然后在 TwinCAT 解决方案中将其互相关联。

映射用途是在模块间提供或传输数据。通过同步或异步映射来确保数据一致性，映射在实际循环程序序列之后或之前执行。

该通信方式的实现与所使用的编程语言（PLC、C++、Matlab®）无关。



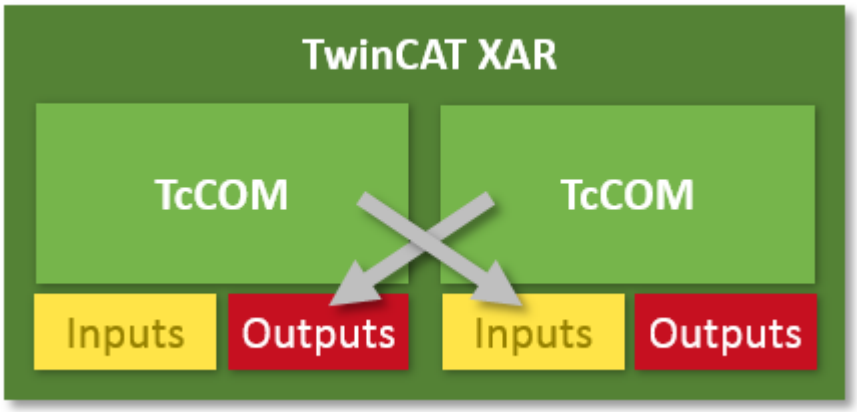
下方示例显示的是实现情况：

Sample12：模块通信：IO 映射的应用 [▶ 284]

IO 数据指针

在任务中还可以通过数据区指针直接访问内存，数据区指针在 TMC 编辑器中创建。

如果出现一个任务的多个调用者或其他任务的调用者，用户必须通过适当的机制确保数据一致性。数据指针可用于 C++ 和 MATLAB®。



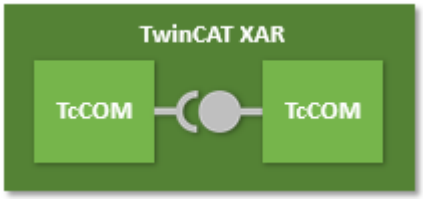
下方示例显示的是实现情况：

Sample10：模块通信：数据指针的应用 [▶ 255]

通过接口的方法调用

如前所述，TcCOM 模块可以提供在 TMC 编辑器中同样定义的接口。如果模块可实现这些接口（TMC 编辑器中的“已实现接口” [▶ 109]），则会提供相应的方法。然后，调用模块会设定一个指向该模块的“接口指针”，以便调用这些方法。

这些都是拦截调用，即调用者会拦截，直到被调用的方法返回，因此可以直接使用这些方法的返回值。如果出现一个任务的多个调用者或其他任务的调用者，用户必须通过适当的机制确保数据一致性。



下方示例显示的是实现情况：

Sample11：模块通信：PLC 模块调用 C 模块的方法 [▶ 256]

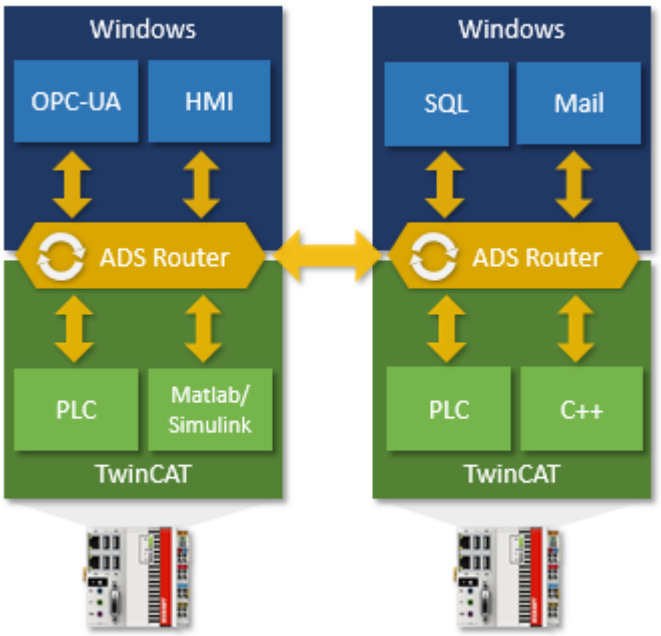
Sample11a：模块通信：C 模块调用另一个 C 模块的方法 [▶ 283]

在与 PLC 的通信方面还有更多的示例 [▶ 302]。

ADS

正如一般的 TwinCAT 系统内部通信，ADS 也可用于模块之间的通信。这种情况下的通信是非周期的事件控制型通信。

同时，ADS 还可用于收集或提供 UserMode 的数据，并与其他控制器进行通信（即通过网络）。ADS 还可用于确保数据一致性通信，例如任务/内核/CPU 之间的通信。在这种情况下，TcCOM 模块既可以是客户端（请求者），也可以是服务端（提供者）。该通信方式的实现与所使用的编程语言（PLC、C++、Matlab®）无关。



下方示例显示的是实现情况：

Sample03：C++ 用作 ADS 服务器 [▶ 238]

Sample06：UI-C#-ADS 客户端从模块上传符号 [▶ 248]

Sample07：接收 ADS 通知 [▶ 252]

Sample08：提供 ADS-RPC [▶ 253]

7 模块 - 处理

TcCOM 模块会在构建后实现并加载。

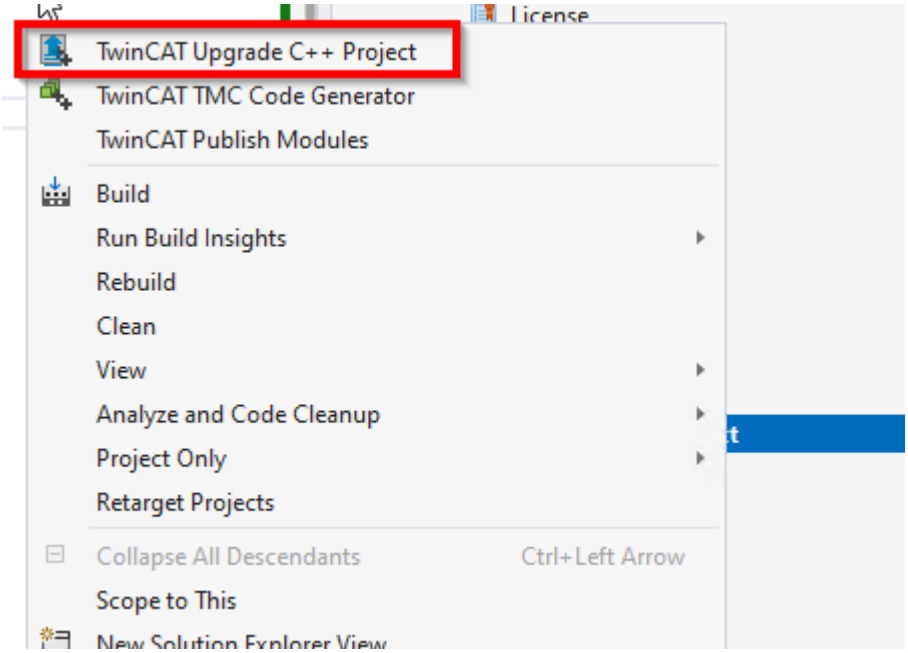
本节介绍的是系统间交换模块时的处理方法。这些模块会在版本控制 C++ 项目 [▶ 48]中编程，生成一个 tmx 文件，以便用 TwinCAT 加载器 [▶ 49]（自 TwinCAT 3.1 Build 4024 起）进行加载。

驱动程序项目

对于旧项目，倍福建议迁移至版本控制 C++ 项目 [▶ 48]，因为这些项目具有以下优势：

- 通过可从倍福获得的 OEM 证书进行驱动程序签名 [▶ 22]。
- 二进制文件版本化存储 [▶ 49]。
- 如有需要，可进行在线更改 [▶ 148]。

TwinCAT 3.1 Build 4026 安装后，可在 C++ 节点上进行转换，以便从旧项目（“驱动程序项目”）迁移。可通过环境菜单访问：



迁移过程会生成一份 HTML 文件报告，该报告内容需重点关注。

7.1 版本控制的 C++ 项目

TwinCAT 3.1 版本 4024.0 及以上

此处描述的功能从 TwinCAT 3.1 及以上版本可用。4024.0.

版本控制的 TwinCAT C++ 项目在构建过程中会生成一个与架构相关的 TMX 文件，并通过 TwinCAT 加载器 [▶ 49]加载。必须由 TwinCAT 用户证书签署。

如果 C++ 项目是使用“版本控制的 C++ 项目”模板创建的，则二进制文件会通过“发布”功能存储在 TwinCAT 工程库中，储存路径（在 TwinCAT 3.1 Build 4024 中为 C:\TwinCAT\3.1\Repository，在 TwinCAT 3.1 Build 4026 中为 C:\ProgramData\Beckhoff\TwinCAT\3.1\Repository），存储路径因 TwinCAT 版本不同而有所差异。

如果需要，模块会从此处转移到目标系统：

- 搭载 TwinCAT 3.1 Build 4024 的 Windows 系统：C:\TwinCAT\3.1\Boot\Repository

- 搭载 TwinCAT 3.1 Build 4026 的 Windows 系统：C:\ProgramData\Beckhoff\TwinCAT\3.1\Boot\Repository
- TwinCAT/BSD：/usr/local/etc/TwinCAT/3.1/Boot/Repository

可以在激活时（**Activate Configuration**）或Online Change [▶ 148]时执行。

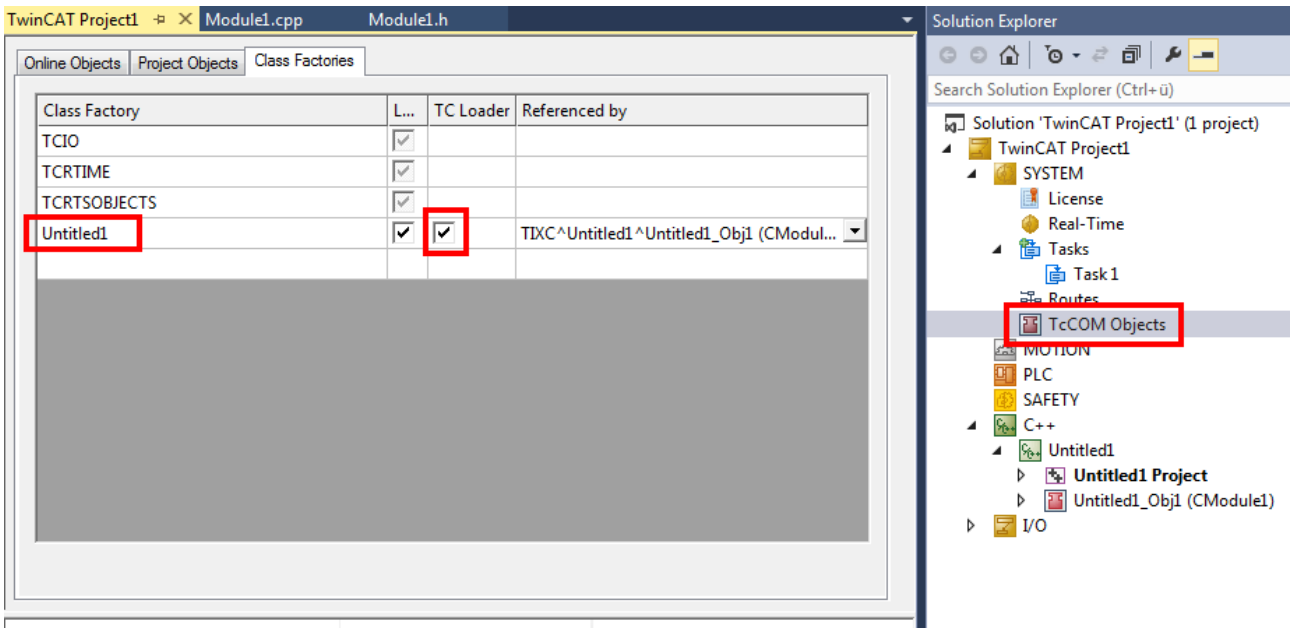
还可以为该项目二进制版本在工程系统之间的传输创建一个归档文件，归档文件由项目属性 [▶ 143]进行配置。

7.2 启动模块

TwinCAT C++ 模块可以通过两种方式启动：

- 操作系统：操作系统将 TwinCAT 模块作为常规驱动程序启动。
  建议迁移到带有 TMX 文件的 TwinCAT 加载程序。
- TwinCAT 加载器： [▶ 49]TwinCAT 加载器可启动 TwinCAT 模块。
  - TwinCAT 加载器要求使用 TwinCAT 用户证书签名 [▶ 21]。
  - 加密模块 [▶ 51]必须使用该选项。
  - 版本控制的 C++ 项目 [▶ 83]需要使用 TwinCAT 加载程序。

通过**系统** -> **TcCOM 模块** -> **类工厂**选项卡，可以查看使用的是 TwinCAT 加载程序还是操作系统：



7.3 TwinCAT 加载程序

TwinCAT 3.1 版本 4024.0 及以上

此处描述的功能从 TwinCAT 3.1 及以上版本可用。4024.0.

TwinCAT 3 集成了模块加载功能。

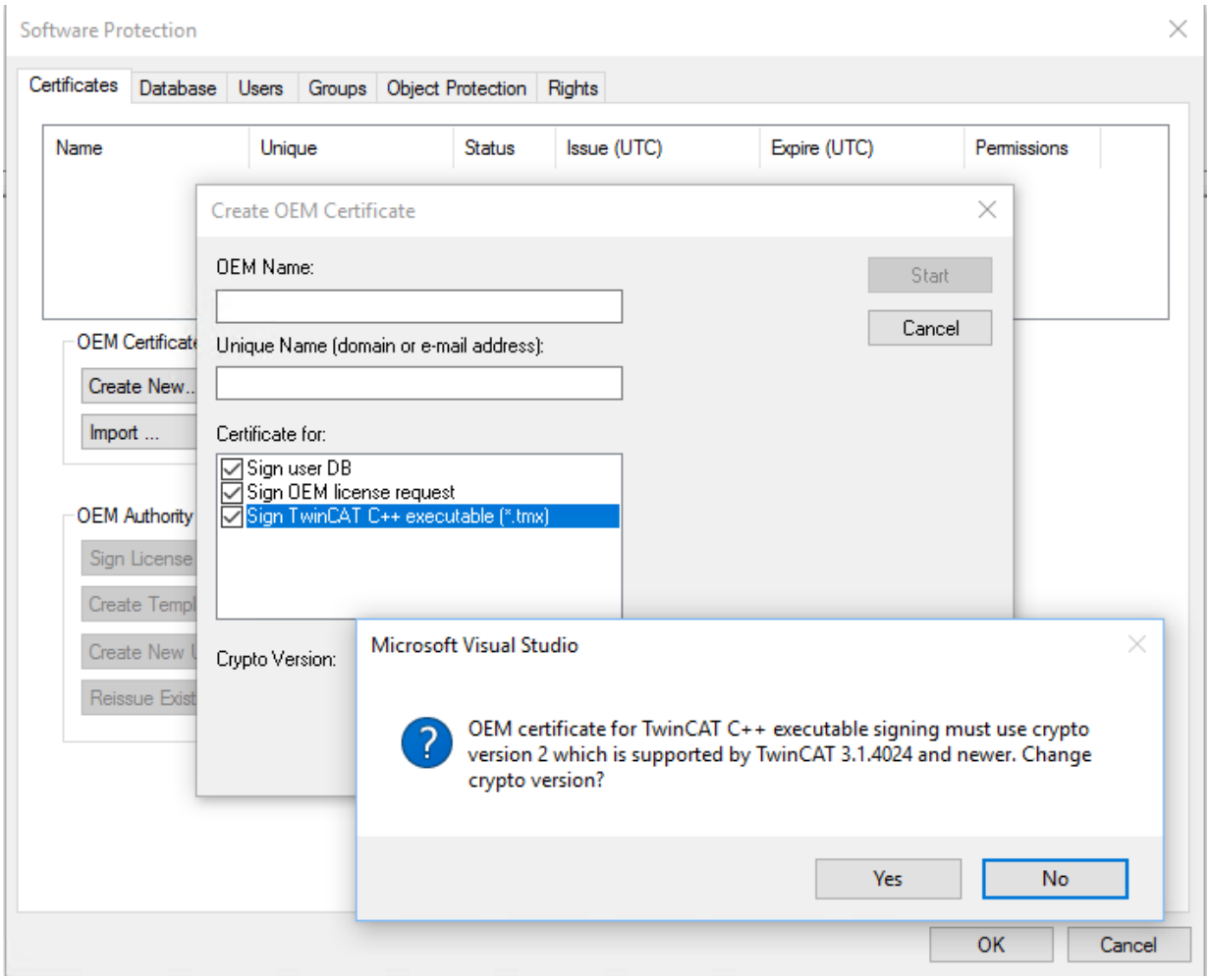
用 TwinCAT 加载器加载的模块：

- **必须**签署：测试签名 [▶ 50]。
- **可以**加密：加密模块 [▶ 51]，其中 TwinCAT Software Protection 必须使用用户数据库配置。

7.3.1 测试签名

可以使用与实际交付相同的 TwinCAT 用户证书对 TwinCAT 进行测试签名（见 申请 TwinCAT 3 用户证书 [▶ 25]）。

1. 对于测试运行，例如在软件开发期间，只需创建一个 TwinCAT 用户证书 [▶ 26]即可。确保选择用途“签署 TwinCAT C++ 可执行文件 (*.tmx)”。为此，需要使用 Crypto 版本 2，此时会显示一条消息。



在 XAR（和 XAE，若为本地测试）上，启用测试模式，以便操作系统可以接受自签名证书。可以在工程系统 (XAE) 和运行时系统 (XAR) 上完成操作。

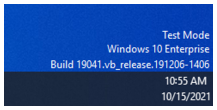
适用于 Windows

利用管理员提示执行以下操作：

```
bcdedit /set testsigning yes
```

并重启目标系统。
为此，必须关闭“SecureBoot”，可在 BIOS 中完成操作。

如果已启用测试签名模式，则会在桌面右下方显示。现在，系统接受所有已签名的执行驱动程序。



适用于 TwinCAT/BSD

在 `/usr/local/etc/TwinCAT/3.1/TcRegistry.xml` 文件中，在密钥“System”下输入 "<Value Name="EnableTestSigning" Type="DW"1</Value>"。

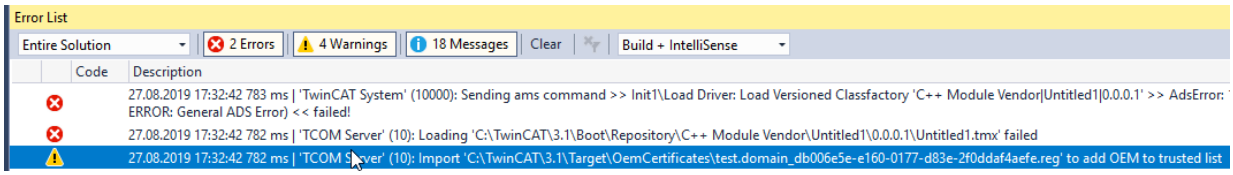
```
<Key Name="System">
  <Value Name="RunAsDevice" Type="DW">1</Value>
  <Value Name="RTimeMode" Type="DW">0</Value>
  <Value Name="AmsNetId" Type="BIN">052445B00101</Value>
  <Value Name="LockedMemSize" Type="DW">33554432</Value>
 <Value Name="EnableTestSigning" Type="DW">1</Value>
</Key>
```

然后重启 TwinCAT 系统服务：

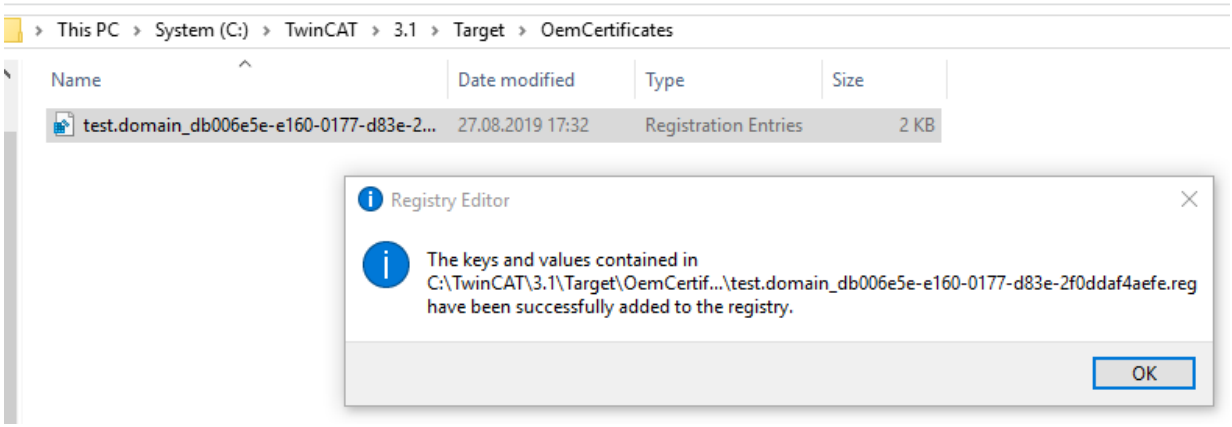
```
doas service TcSystemService restart
```

完成相应程序后，系统会接受所有已签名的执行驱动程序。

1. 在使用 TwinCAT 用户证书执行首次启用 (Activate Configuration) 时，目标系统会检测到证书不受信任，启用过程中止：



适用于 Windows：
具有管理权限的本地用户可以通过创建的 REG 文件信任证书，只需执行该文件即可：



适用于 TwinCAT/BSD：
如果未安装“Tcimportcert”软件包，请安装：`pkg install TcImportCert`
通过 `doas tcimportcert /usr/local/etc/TwinCAT/3.1/Target/OemCertificates/<CreatedFile>.reg` 信任证书。
然后重启 TwinCAT 系统服务或重新启动系统：

```
doas service TcSystemService restart
```

⇨ 此过程只允许运行具有受信任 TwinCAT 用户证书签名的 C++ 模块。

2. 在此过程之后，您可以使用 TwinCAT 用户证书以操作系统的测试模式进行签名。
在项目属性 [▶ 144]中对此进行配置。
使用 TcSignTool [▶ 54] 可避免在项目中存储 TwinCAT 用户证书的密码，例如，在版本管理中也会出现这种情况。

如果要使用不带 TestMode 的 TwinCAT 用户证书进行交付，则必须由倍福对证书进行会签 [▶ 24]。

7.3.2 加密模块

通过 TwinCAT 加载程序加载的 TwinCAT C++ 模块（TMX 文件）可以加密，即通过密钥保护驱动程序的内容，防止文件级的篡改和逆向工程设计。

禁止调试

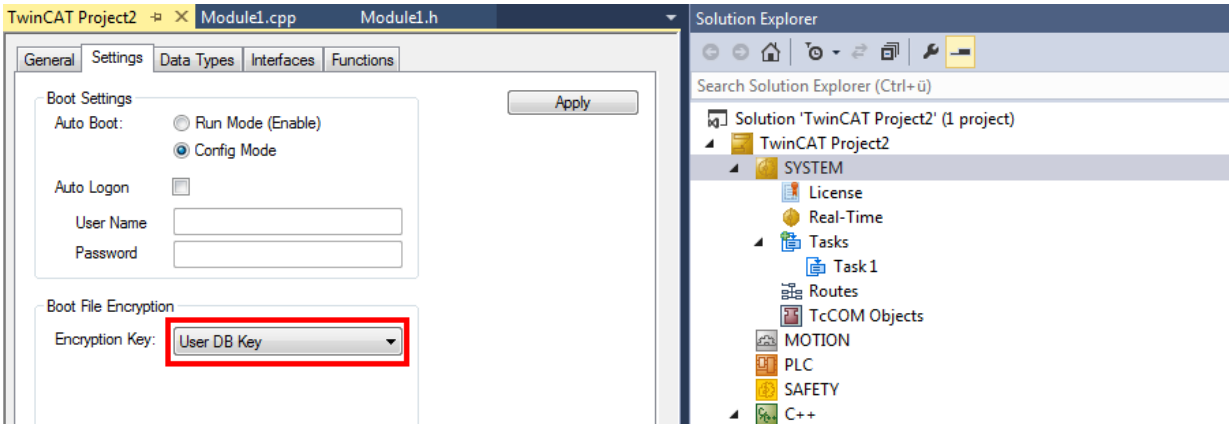
无法排查加密模块的错误。调试器中不显示加密模块。

模块加密的启用方式如下：

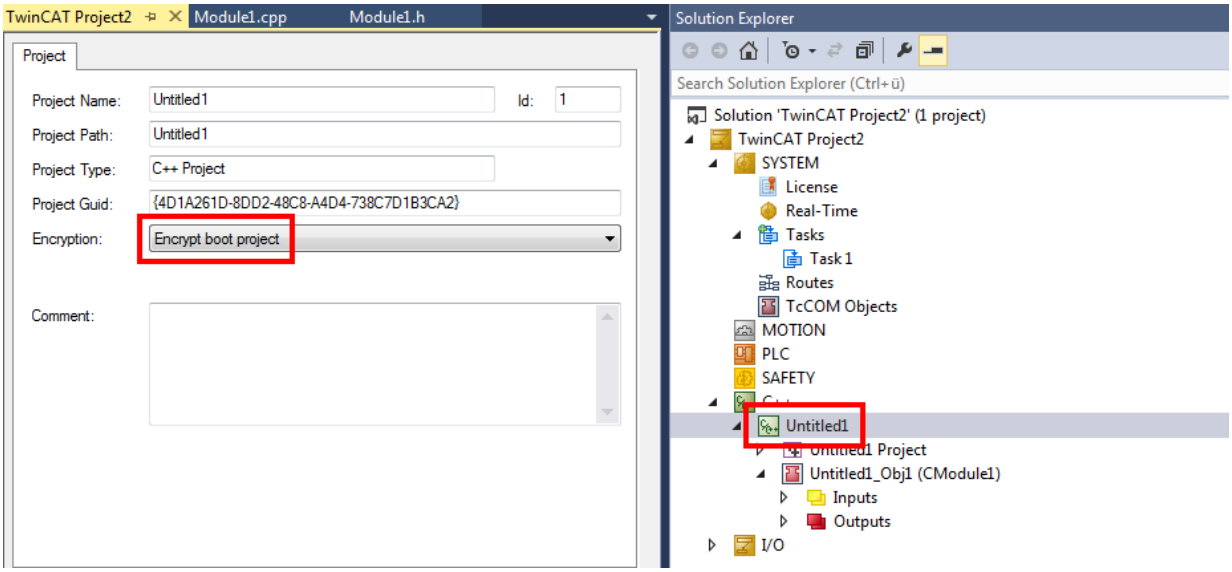
✓ 必须配置 TwinCAT 软件保护。

✓ 需要具有“签署 UserDB”权限的 TwinCAT 用户证书。

1. 在system tree中，选择解决方案**用户 DB 密钥**作为“启动文件加密密钥”。



2. 选择 C++ 项目并激活加密：



3. 开始时，必须使用 TwinCAT 加载程序（而不是操作系统）加载加密模块。

⇨ 对于非版本控制的驱动程序：驱动程序在传输到项目的 _deployment 目录时会被加密。

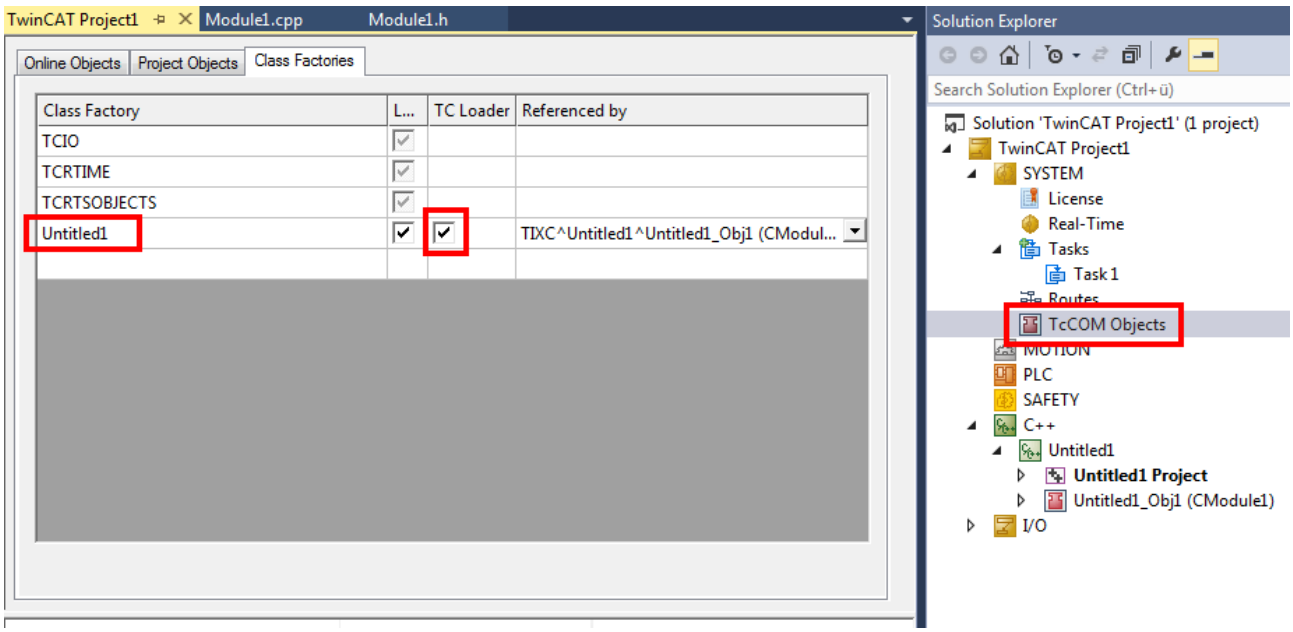
⇨ 对于版本控制的 TMX：驱动程序未加密存储在 XAE 中，而在目标系统上激活时进行加密。

⇨ 如果该功能用于版本控制的 C++ 项目，则 TMX 文件会像往常一样存储在库 [▶ 48]中。

TwinCAT C++ 模块可以通过两种方式启动：

- 操作系统：操作系统将 TwinCAT 模块作为常规驱动程序启动。
  建议迁移到带有 TMX 文件的 TwinCAT 加载程序。
- TwinCAT 加载器： [▶ 49]TwinCAT 加载器可启动 TwinCAT 模块。
  - TwinCAT 加载器要求使用 TwinCAT 用户证书签名 [▶ 21]。
  - 加密模块 [▶ 51]必须使用该选项。
  - 版本控制的 C++ 项目 [▶ 83]需要使用 TwinCAT 加载程序。

通过**系统** -> **TcCOM 模块** -> **类工厂**选项卡，可以查看使用的是 TwinCAT 加载程序还是操作系统：



7.3.3 返回代码

由于各种原因，使用 TwinCAT 加载器加载模块可能会失败。

Windows 下的返回代码

十六进制	描述
0xC1	文件已损坏；PE 文件校验和错误。
0x241	签名错误：TwinCAT 用户证书与文件哈希值不匹配。
0x4FB	签名错误：文件未签署。
0x1772	文件已加密，但无法用已知密钥解码。

TwinCAT/BSD 下的返回代码

十六进制	描述
0xC0000001 0xC0000004 0xC000007B 0xC0000173 0xC0000221	文件已损坏；PE 文件校验和错误。
0xC0000102 0xC0000424	签名错误：TwinCAT 用户证书与文件哈希值不匹配。
0x4FB	签名错误：文件未签署。
0xC0000428	签名错误：所用证书未经会签。需要测试模式。
0xC0000603	签名错误：证书不受信任。需要添加（请参见“tcinsertcert”）
0xC0000385	签名错误：文件未签署。检查“工程设计”中的设置。
0xC0000293	文件已加密，但无法用已知密钥解码。
0xC0000225	找不到文件。

7.3.4 TcSignTool - 在项目外存储证书密码

TcSignTool 支持在注册表中存储 TwinCAT 用户证书的密码。这样，项目便不再需要密码，可避免密码因疏忽被纳入版本控制系统。

TcSignTool 是一个命令行工具，位于 C:\TwinCAT\3.1\sdk\Bin\ 或 C:\Program Files (x86)\Beckhoff\TwinCAT\3.1\SDK\Bin 路径下。

通过以下参数存储密码：

```
tcsigntool grant /f "…CustomConfig\Certificates\MyCertificate.tccert" /p MyPassword
```

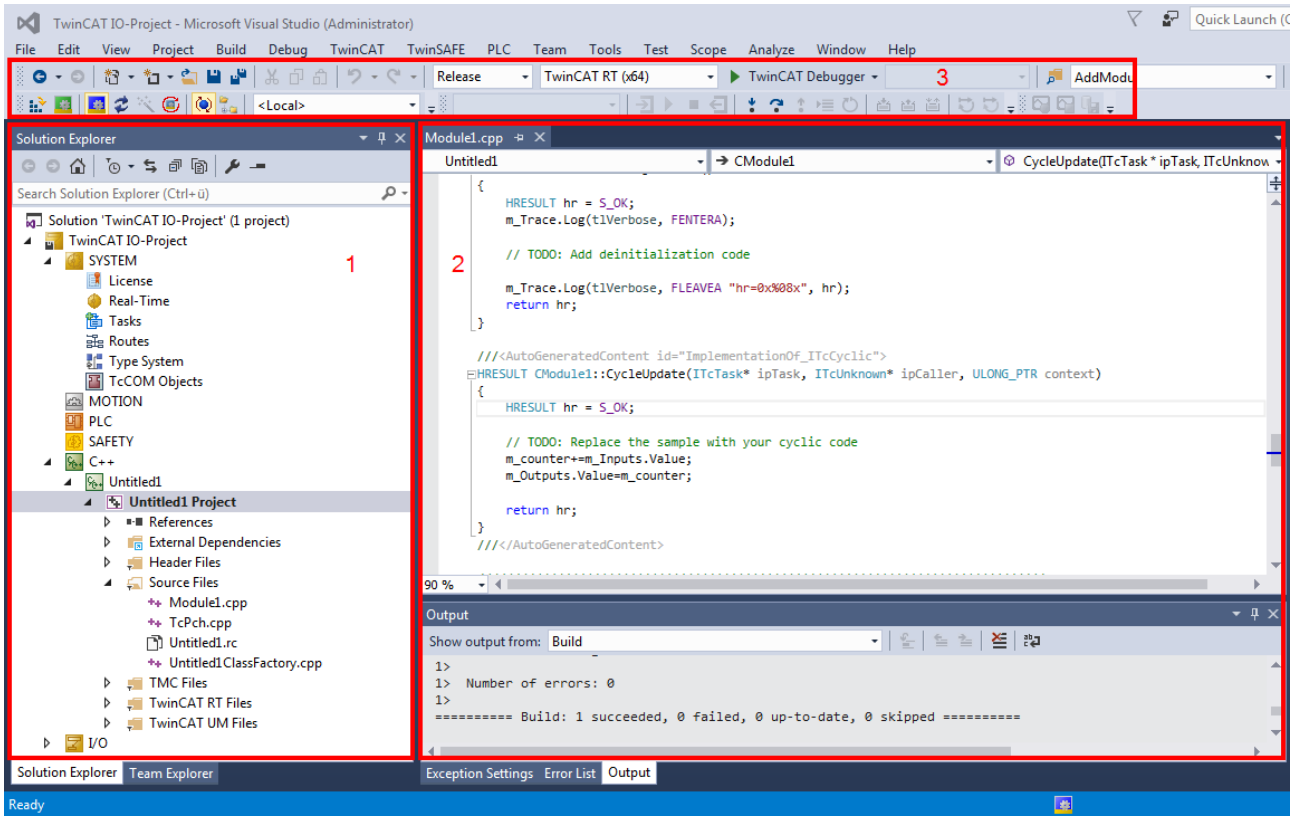
通过以下参数删除密码：

```
tcsigntool grant /f "…CustomConfig\Certificates\MyCertificate.tccert" /r
```

未加密的密码存储在：`HKEY_CURRENT_USER\SOFTWARE\Beckhoff\TcSignTool\`

8 TwinCAT C++ 开发

开发环境概览



Visual Studio 的布局具有灵活性和适应性，因此在此只能简要介绍常见的配置。用户可根据需要自由配置窗口和安排。

1. 在 TwinCAT 解决方案中，右键单击 C++ 图标即可创建 TwinCAT C++ 项目。该项目包含多个模块 [▶ 35] 的源代码（“未命名项目”）（如适用），并可创建模块实例（“Untitled1_Obj1 (CModule1)”）。模块实例有输入/输出，可以通过常规方式（“链接”）进行关联。还有更多选项 [▶ 45]可进行模块交互。
2. Visual C++ 的 Visual Studio 编辑器用于编程。此处应特别注意下拉框，以便在文件内快速导航。下部区域用于输出编译过程的结果。也可以切换为 TwinCAT 消息（见 工程的模块消息（日志记录/追溯） [▶ 208]）。
   在编辑器中可以使用断点等常用功能（见 调试 [▶ 74]）。
3. 可自由配置的工具栏通常包含 TwinCAT XAE Base 的工具栏。**Activate Configuration（激活配置）**、**RUN（运行）**、**CONFIG（配置）**、Choose Target System（选择目标系统）（在本例中为**<Local>**）和其他几个按钮可快速访问常用功能。就 C++ 模块而言，**TwinCAT 调试器**是与目标系统建立连接的按钮（PLC 使用独立调试器）。与其他 C++ 程序一样，与 PLC 不同的是，在 TwinCAT C++ 中必须区分“发布”和“调试”。在“发布”的构建过程中，代码经过优化，调试器可能无法再可靠地到达断点，并且可能会显示错误数据。

程序

本节介绍的是 TwinCAT C++ 项目的编程、编译和启动过程。

其中提供 TwinCAT C++ 项目工程设计流程的总体概述，并参考相应的详细文档。快速入门 [▶ 57]指南介绍了各个常见步骤。

1. 类型声明和模块类型：
   TwinCAT 模块类编辑器 (TMC) [▶ 86] 和 TMC Code Generator 用于定义数据类型和接口，以及使用这些类型和接口的模块。

TMC Code Generator 根据处理过的 TMC 文件生成源代码，并准备数据类型/接口，以便在其他项目（如 PLC）中使用。
编辑和启动代码生成器的次数不限，代码生成会关注并保留已编程的用户代码。

2. 编程
   熟悉的 Visual Studio C++ 编程环境用于开发和调试 [▶ 74]代码模板中的用户定义代码。
3. 实例化模块 [▶ 35]
   该程序功能说明的是被实例化为对象的类。TwinCAT 模块实例配置器 [▶ 129] 用于配置实例。一般配置要素包括：分配任务、下载运行时的符号信息（TwinCAT 模块实例 (TMI) 文件）或指定参数/接口指针。
4. 变量映射
   使用标准 TwinCAT 系统管理器，可以将一个对象的输入和输出变量与其他对象或 PLC 项目的变量关联起来。
5. 构建
   在构建（编译和链接）TwinCAT C++ 项目过程中，所有组件均针对所选平台编译。平台在选择目标系统时自动确定。
6. 发布
   在发布模块时，会创建适用于所有平台的驱动程序，并为模块的发布做好准备。在存储库中创建的目录无需传输源代码即可分发。仅传输带有接口说明的二进制代码。
7. 签名（见 模块签名 [▶ 21]）
   TwinCAT C++ 项目必须签名。签名过程可以由用户定义 [▶ 34]。
8. 激活
   TwinCAT C++ 驱动程序可以像任何其他 TwinCAT 项目一样通过**激活配置**进行激活。然后对话框请求将 TwinCAT 切换至运行模式。
   对于 TwinCAT C++ 模块，可以进行实时调试 [▶ 74]（在基于 IEC61131 的系统中很常见）并设置（条件）断点。

⇨ 该模块在实时条件下运行。

9 快速入门

本快速入门会展示如何在短时间内熟悉 TwinCAT C++ 模块工程设计。下面将详细介绍创建实时运行模块的每个步骤。

讨论两种不同的方案：

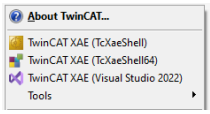
- TwinCAT 版本控制的 C++ 项目。
  基于此类项目的模块由 TwinCAT 加载，并以二进制形式存储。
- 我们将以版本控制的 C++ 项目为基础，说明如何通过“Online Change”在不同版本之间进行切换。

了解快速入门之前，请注意准备工作 - 只需一次！ [▶ 21]特别是要准备好相应的驱动程序签名。

9.1 创建 TwinCAT 3 项目

启动 TwinCAT 工程环境 (XAE)

Microsoft Visual Studio® 可通过 TwinCAT SysTray 图标启动。

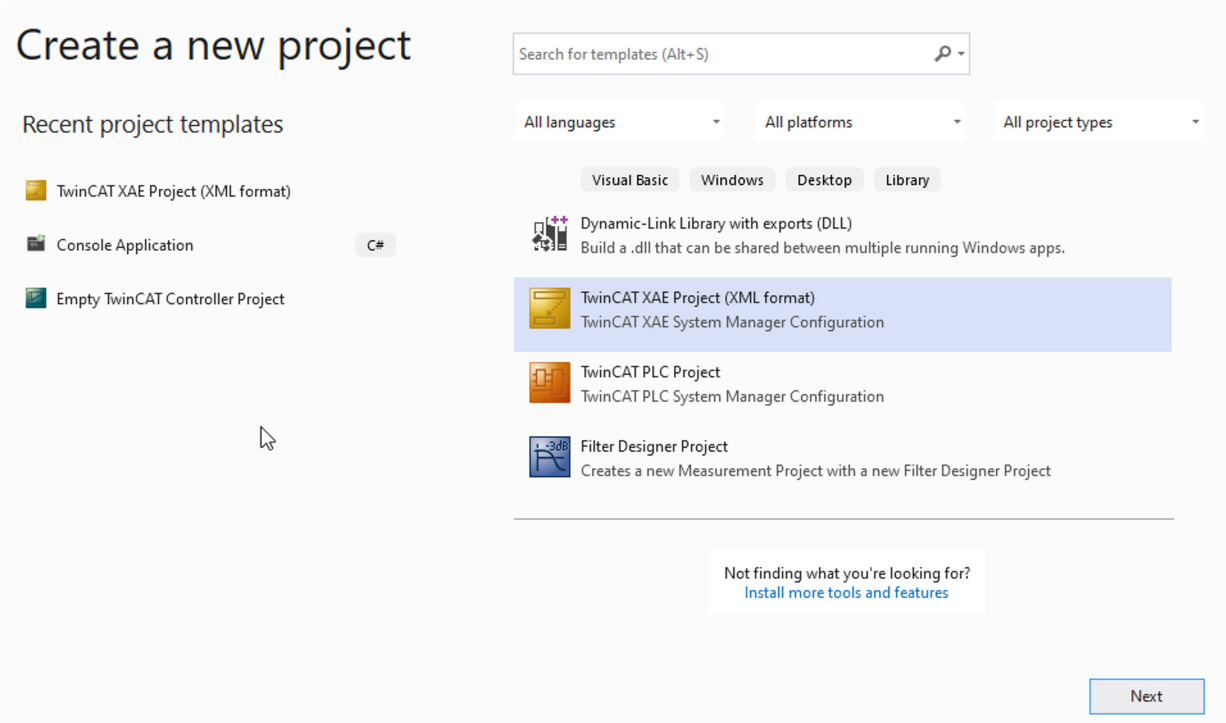


所提供的 Microsoft Visual Studio® 版本为已激活 TwinCAT 的版本。
或者，也可通过“开始”菜单启动 Microsoft Visual Studio®。

创建 TwinCAT 3 C++ 项目

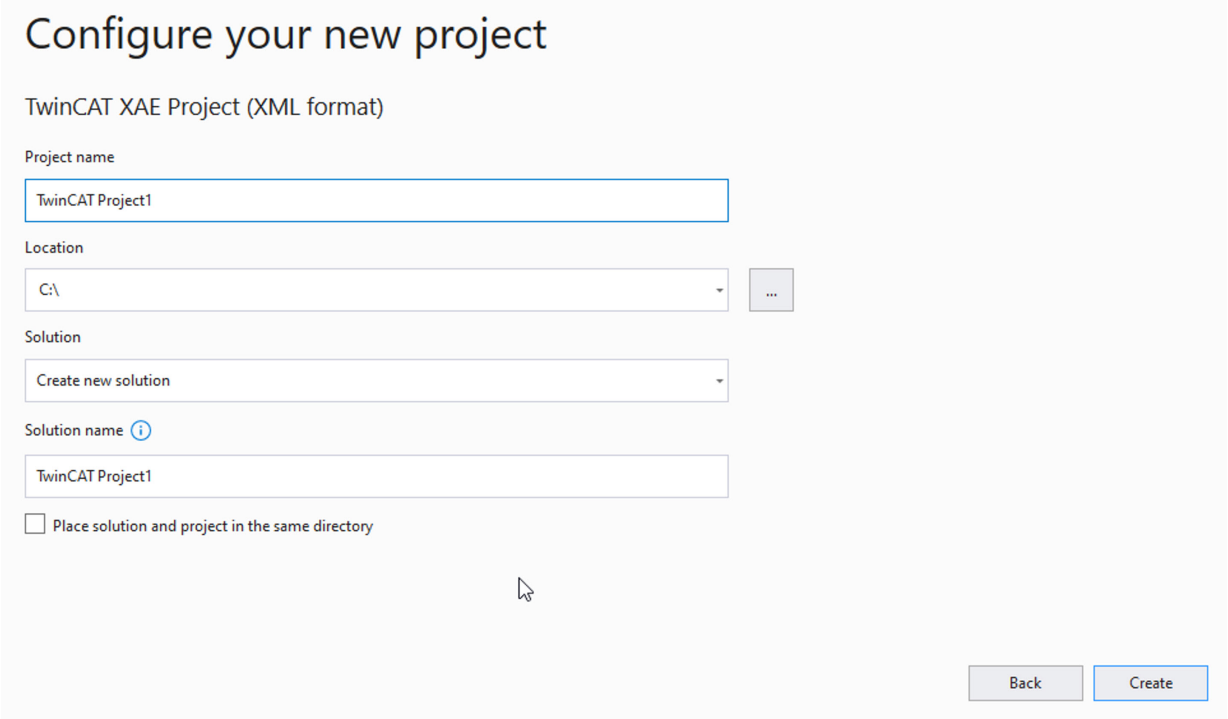
执行以下步骤创建 TwinCAT C++ 项目：

1. 通过“开始”页面选择 **New TwinCAT Project（新建 TwinCAT 项目.....）**。

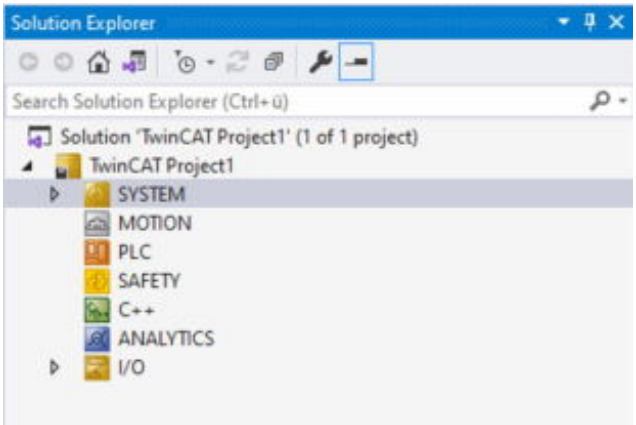


2. 或者，您也可以通过点击：**文件** -> **新建** -> **项目**来创建项目。
   ⇨ 显示所有现有项目模板。
3. 选择 **TwinCAT XAE 项目**，并可根据需要输入合适的项目名称。

4. 点击**确定**。



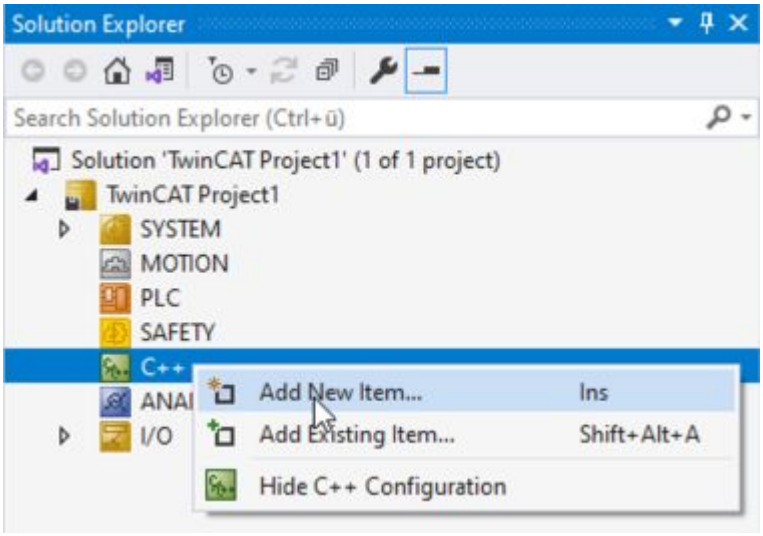
⇨ Visual Studio 解决方案资源管理器随后会显示 TwinCAT 3 项目。



9.2 创建 TwinCAT 3 C++ 项目

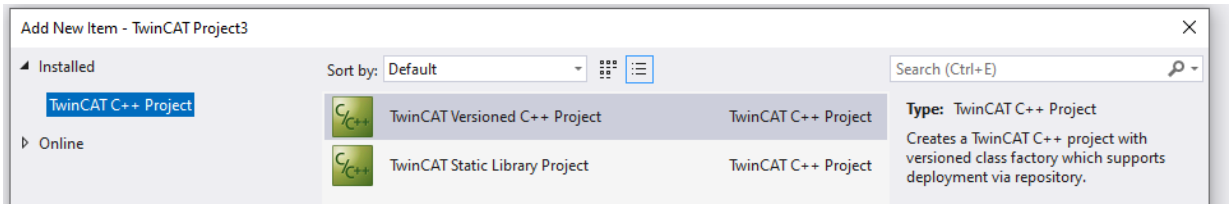
创建 TwinCAT 3 项目后，打开 C++ 节点并按以下步骤操作：

1. 右键单击 **C++** 并选择**添加新项目……**。
   如果绿色 C++ 符号未列出，则意味着要么选择的目标设备不支持 TwinCAT C++，要么 TwinCAT 解决方案当前在不支持 C++ 的 Visual Studio 版本中打开（见 安装 [▶ 19]）。



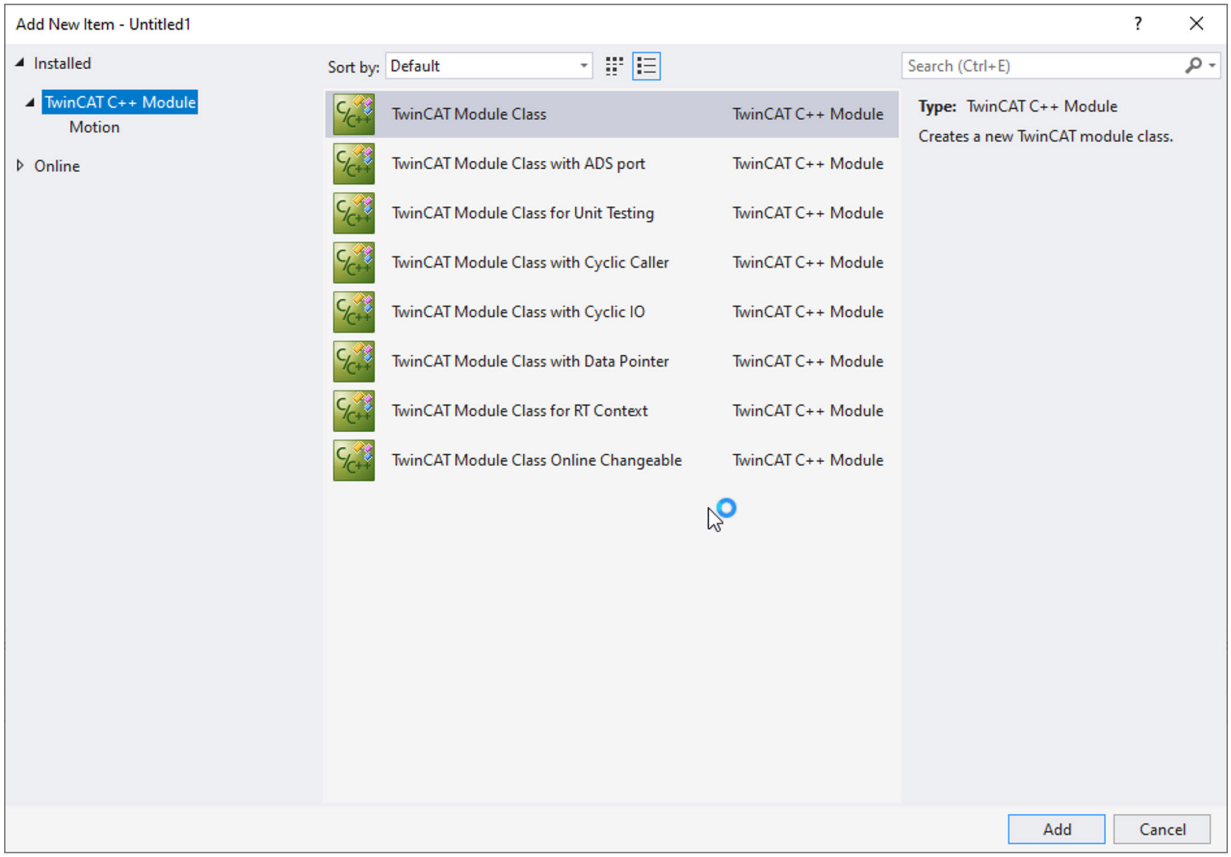
   ⇨ 显示 TwinCAT C++ 项目向导 [▶ 83]，并列出所有现有项目模板。

2. 选择 **TwinCAT 版本控制 C++ 项目**，可根据需要输入相关项目名称，然后点击**确定**。
   或者，也可使用 **TwinCAT 静态库项目**，为静态 TwinCAT C++ 库的编程提供一个环境（见示例 25 [▶ 295]）。



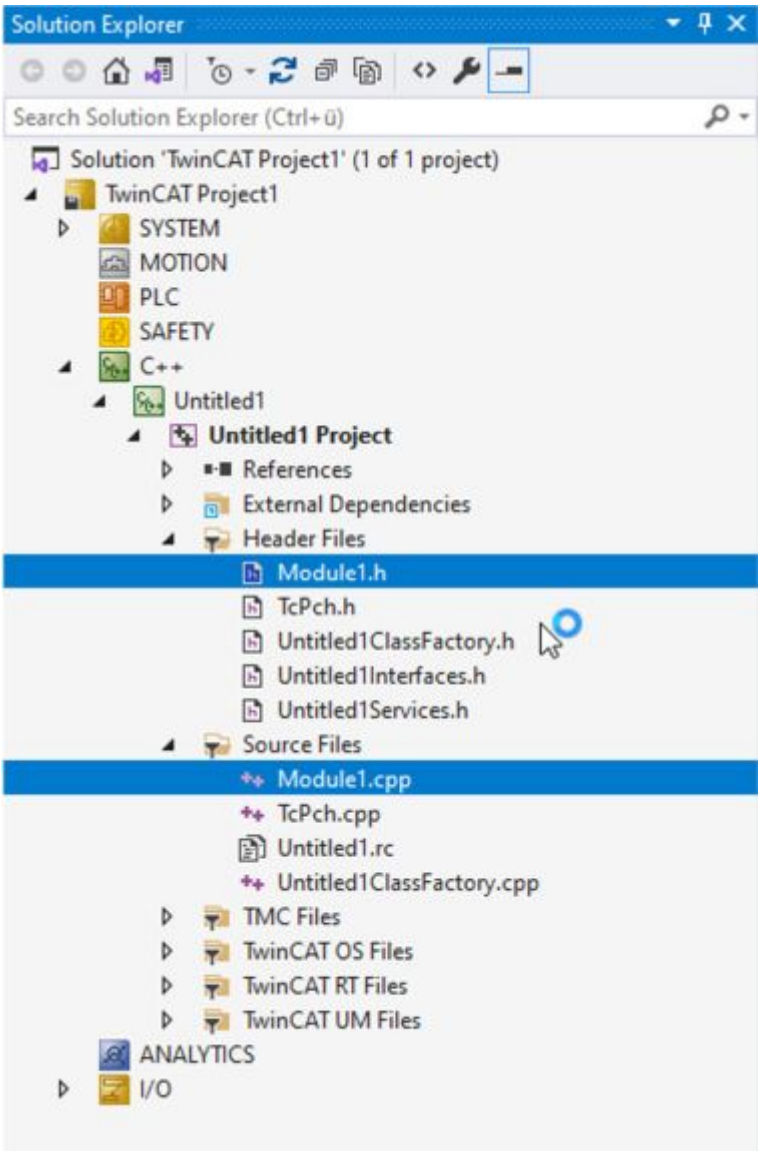
   ⇨ 显示 TwinCAT 模块向导 [▶ 83]。

3. 在这种情况下，选择**带循环 IO 的 TwinCAT 模块类**，然后点击**确定**。如果要使用“在线更改”功能，请选择 **TwinCAT 模块类在线更改**。



4. 在 **TwinCAT 类向导**对话框中输入一个唯一名称，或者按照建议继续输入“Module1”。

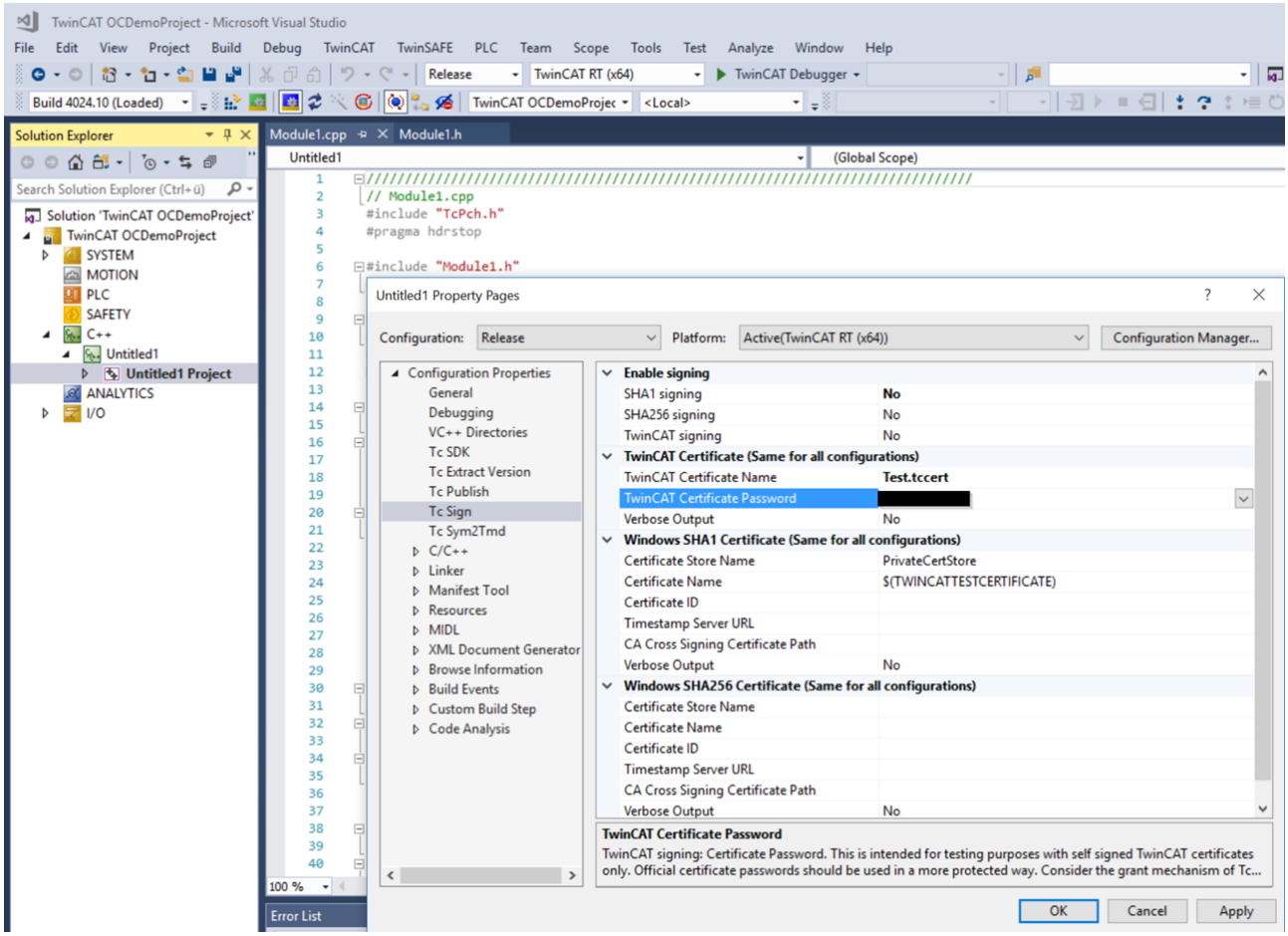
⇨ 然后将根据所选模板创建一个 TwinCAT 3 C++ 项目：



9.3 TwinCAT 3 C++ 配置项目

✓ 您已创建一个 TwinCAT 版本控制 C++ 项目，并通过向导创建了一个模块

1. 右键单击，转至项目的属性。
2. 在 **Tc Sign**选项卡中激活 TwinCAT 签名。
3. 如果尚未创建 TwinCAT 用户证书，请按照说明操作，并注意选择**签署 TwinCAT C++ 可执行文件**。
4. 输入 TwinCAT 用户证书的文件名和密码。
   （请注意，证书以未加密方式存储在解决方案中，因此也会通过版本控制等方式加载到服务器上。如有必要，请使用 TcSignTool [▶ 54]。）

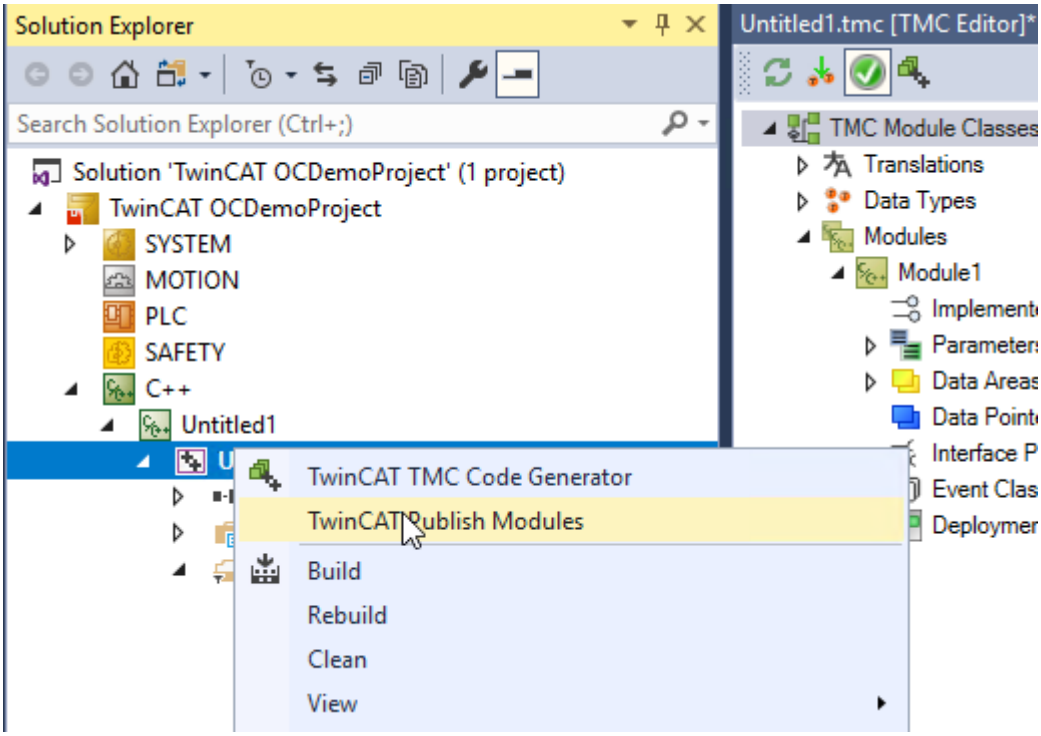


9.4 实现 TwinCAT 3 C++ 项目

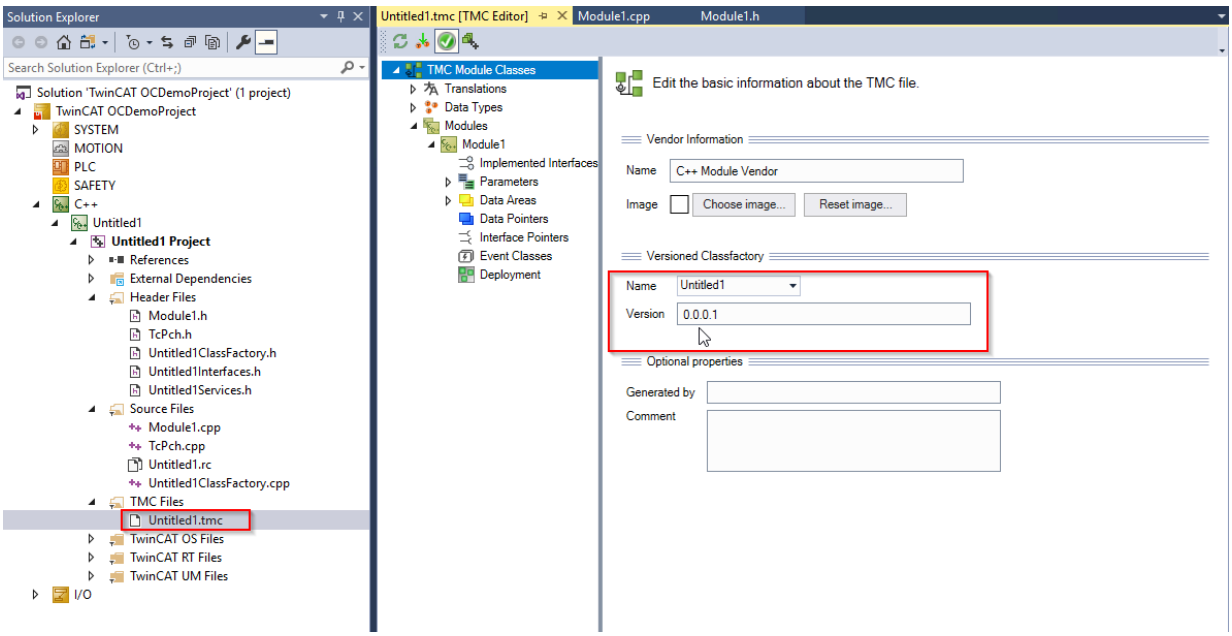
本文将介绍如何更改示例项目。

创建一个 TwinCAT C++ 项目并打开 <MyClass>.cpp（本示例中为 Module1.cpp）后，开始实现。

1. <MyClass>::CycleUpdate() 方法被循环调用，此为循环逻辑的定位点。此时，添加整个循环代码。使用编辑器顶部的下拉菜单进行导航。



2. 在这种情况下，计数器会根据输入图像 (`m_Inputs`) 中变量Value的值递增。替换一行，以便在不依赖输入图像值的情况下递增计数器。
   将这一行：
   ```
   m_counter+=m_Inputs.Value;
   ```
   替换为这一行：
   ```
   m_counter++;
   ```
3. 保存修改。
4. 如果您已经为“在线更改”准备好模块，请注意此处已实现 0.0.0.1 版，您可以在 TMC 编辑器中看到。

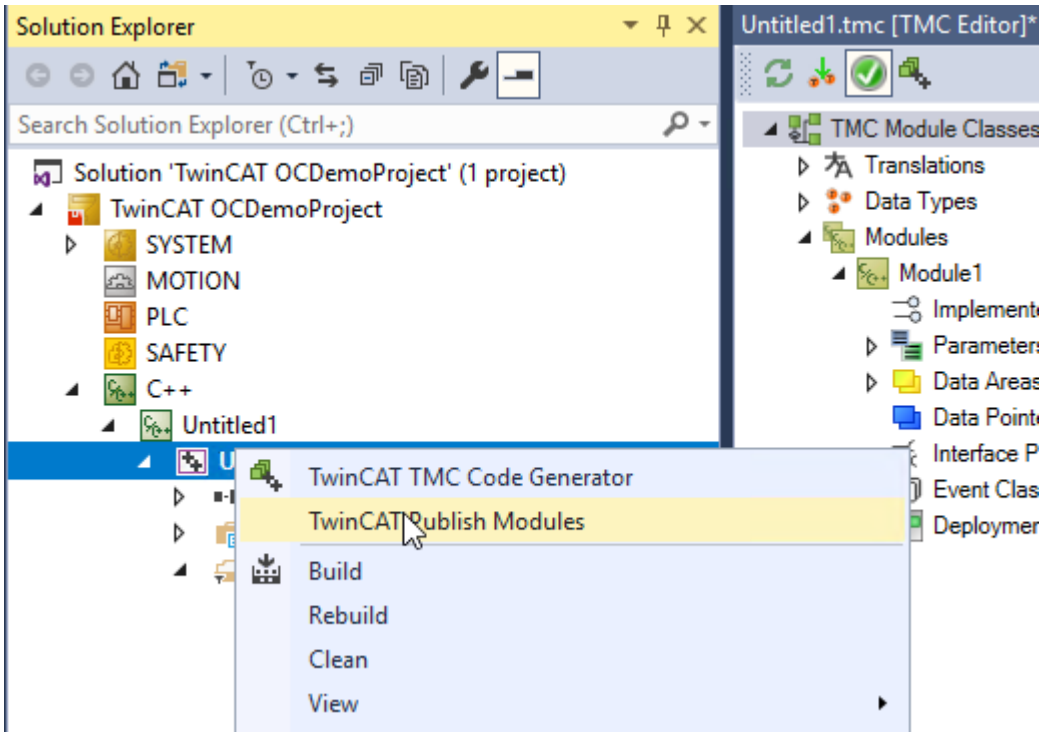


5. 现在可以构建项目，并在本地测试成功后再进行实际实现。

9.5 发布 0.0.0.1 版 TwinCAT 3 C++ 项目

在创建、编译和测试 TwinCAT C++ 项目之后，可以发布该项目，即为分发做好准备。

1. 点击环境菜单中的 **TwinCAT 发布模块**发布模块，使其作为版本 0.0.0.1



发布到本地存储库

⇨ 该模块发布的版本为 0.0.0.1，工程设备配有增量计数器。

9.6 实现并发布 TwinCAT 3 C++ 项目 0.0.0.2 版

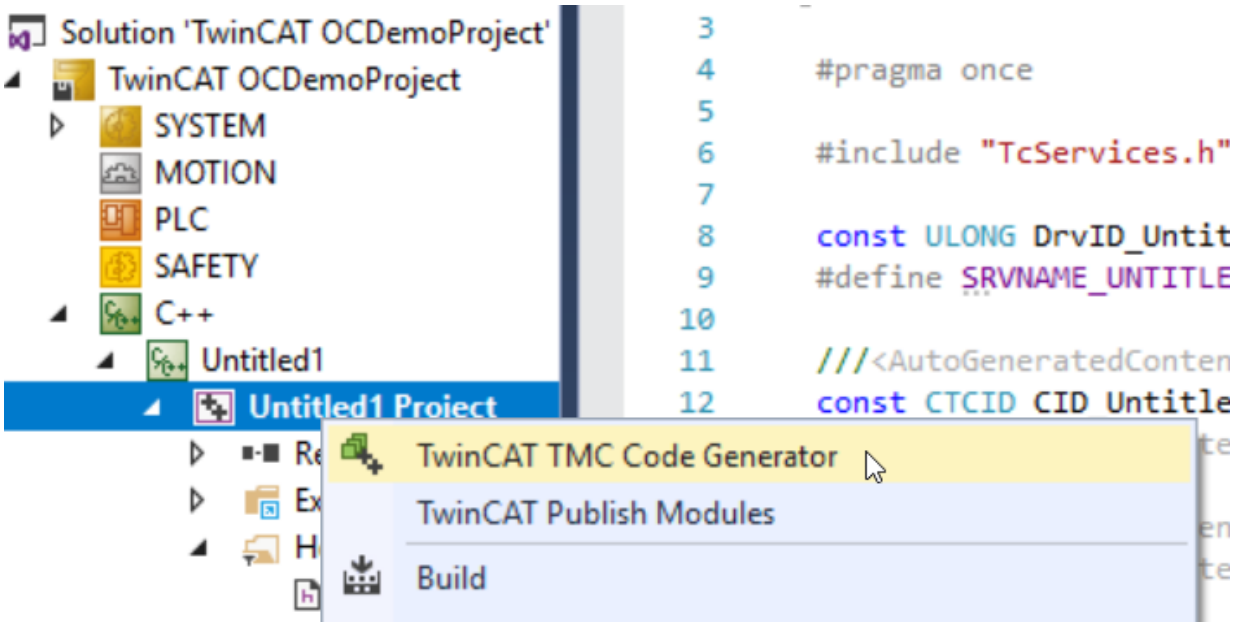
只有在先前已为“在线更改”准备好模块的情况下，才需要执行这些步骤。

本文介绍如何更改示例项目以创建 0.0.0.2 版。随后可通过“在线更改”在目标系统上进行交换

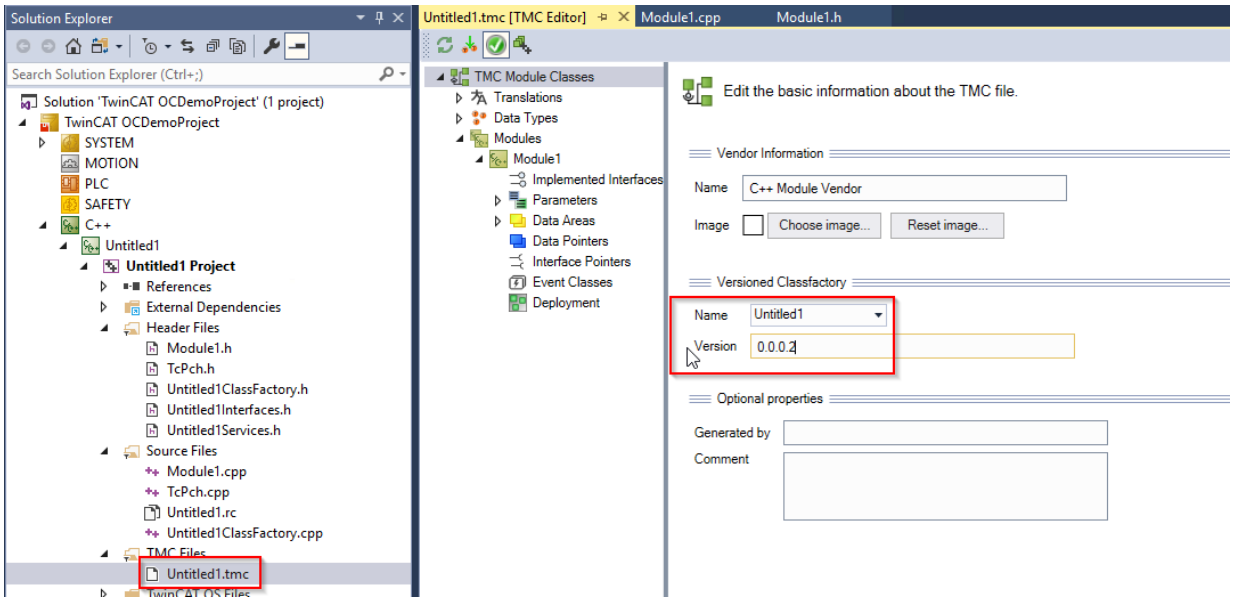
在 <MyClass>.cpp（本示例中为 Module1.cpp）中，可以更改实现方式。

1. 将某一行代码替换为使计数器递减而不是递增。
   将这一行
   ```
   m_counter++;
   ```
2. 替换此行
   ```
   m_counter--;
   ```
3. 保存修改。

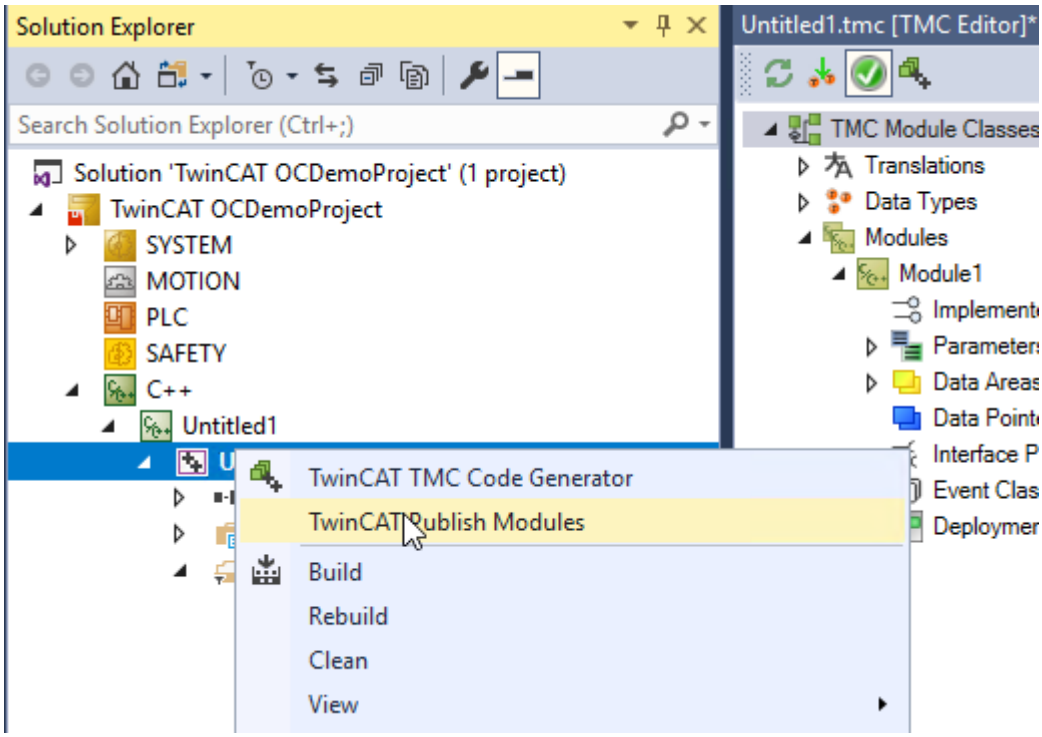
4. 启动代码生成器，处理可能的更改。



5. 在 TMC 编辑器中对 0.0.0.2 版进行参数设置。



6. 同时发布此版本：



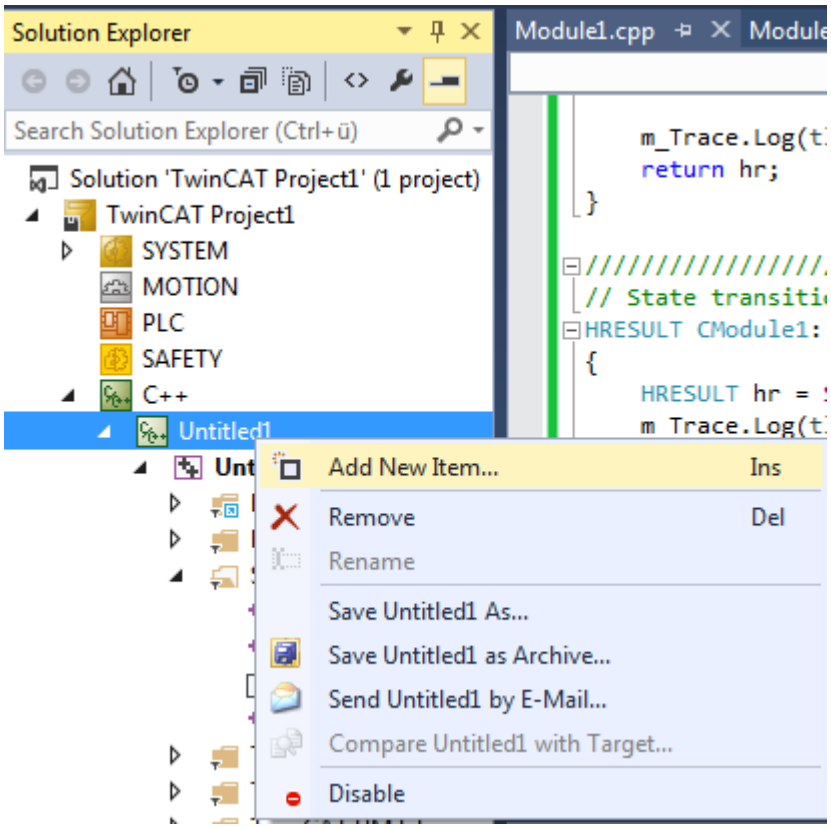
⇨ 一个模块有两个版本，可以在运行时期间交换。

9.7 创建 TwinCAT 3 C++ 模块实例

必须创建一个模块实例才能执行该模块。一个模块可以有多个实例。

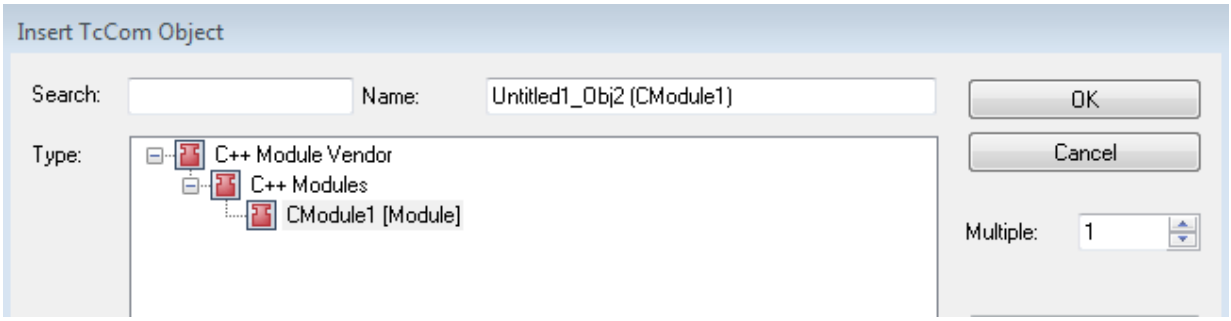
创建 TwinCAT C++ 模块后，打开 **C++** 节点并按照以下步骤创建实例。

1. 右键单击 C++ 模块（本例中为“Untitled1”），选择**添加新项目.....**



⇨ 已列出所有现有的 C++ 模块。

2. 选择 C++ 模块。您可以使用默认名称，也可以输入新的实例名称，然后点击**确定**确认（本例中选择的是默认名称）。



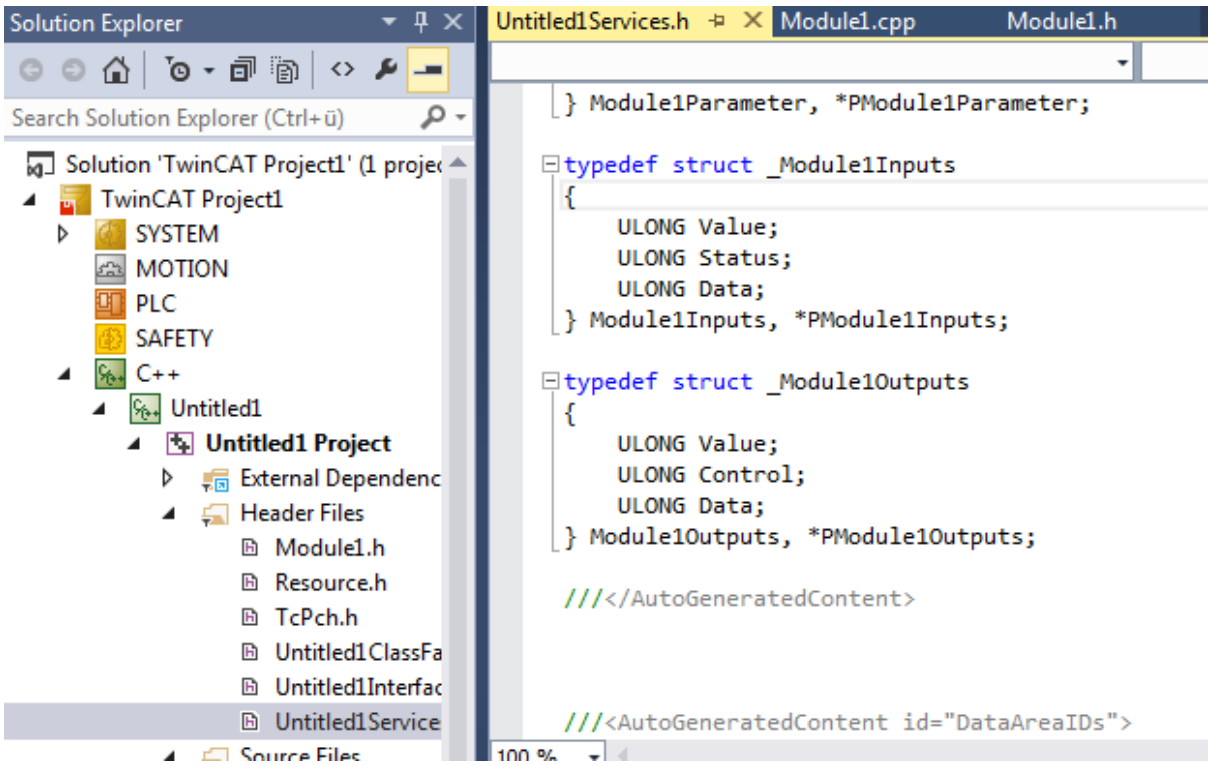
⇨ 新实例“Untitled1_Obj2 (CModule1)”成为 TwinCAT 3 解决方案的一部分：新节点可精准定位在 TwinCAT 3 C++ 源“Untitled1 项目”下。

该模块已提供一个简单的 I/O 接口，每种情况下有 3 个变量：

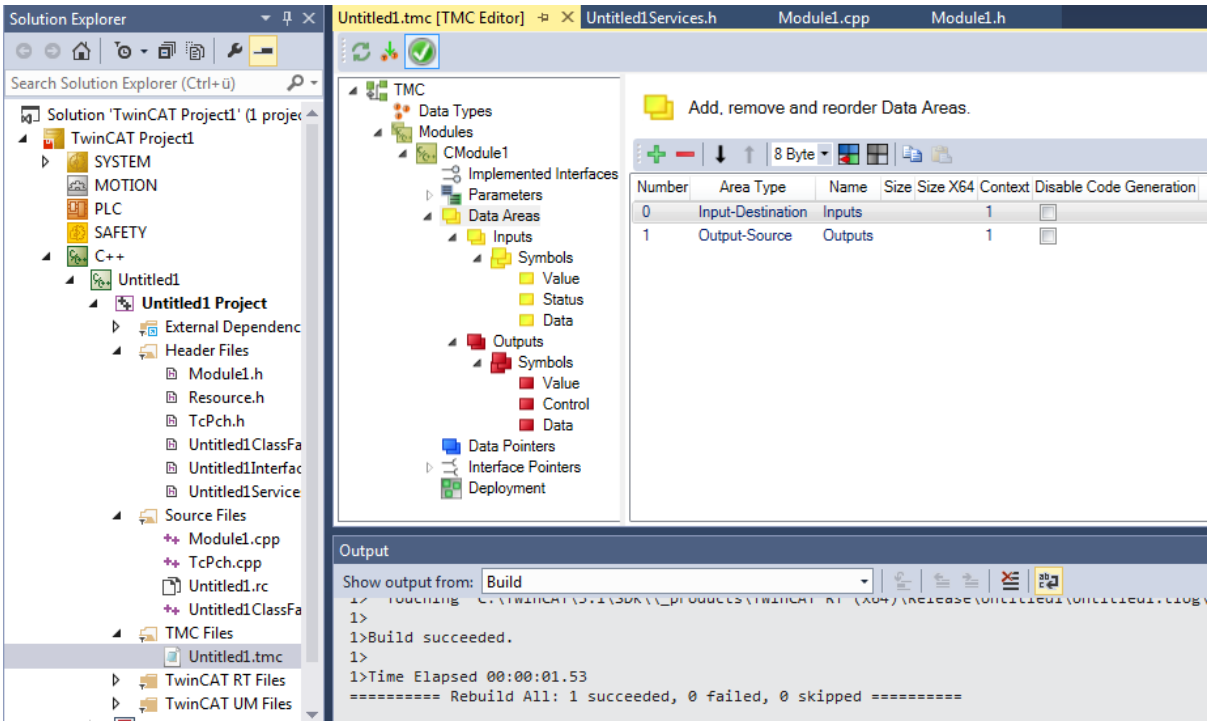
- 输入区域：值、状态、数据
- 输出区域：值、控制、数据

对这些接口的说明在两个位置保持一致：

- “<Classname>Services.h”（本例中为“Untitled1Services.h”）



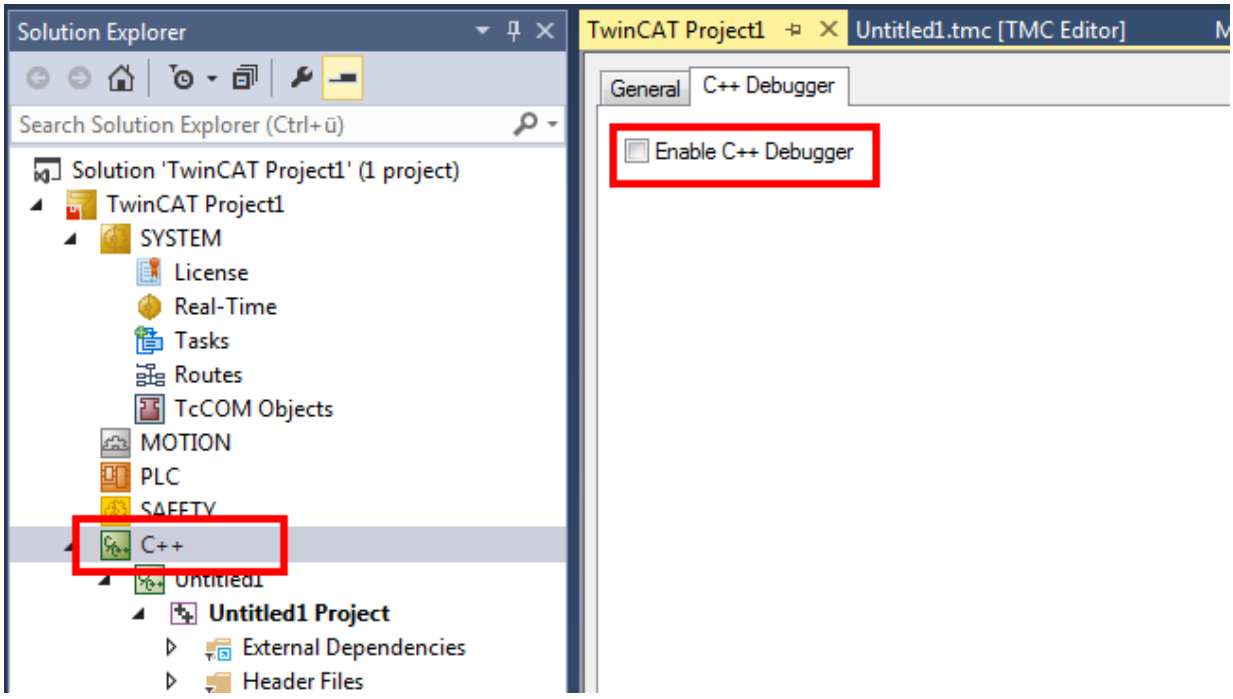
- “TwinCAT 模块配置”.tmc 文件（本例中为“Untitled1.tmc”）



9.8 TwinCAT 3 启动 C++ 调试器

为避免在调试 [▶ 74]时加载所有依赖项，该功能默认关闭，必须在激活配置前激活一次。

1. 在“解决方案”选项卡的 C++ 节点上选择 **C++ 调试器**。
2. 选择**启用 C++ 调试器**。
3. 开启**启用 C++ 调试器**。



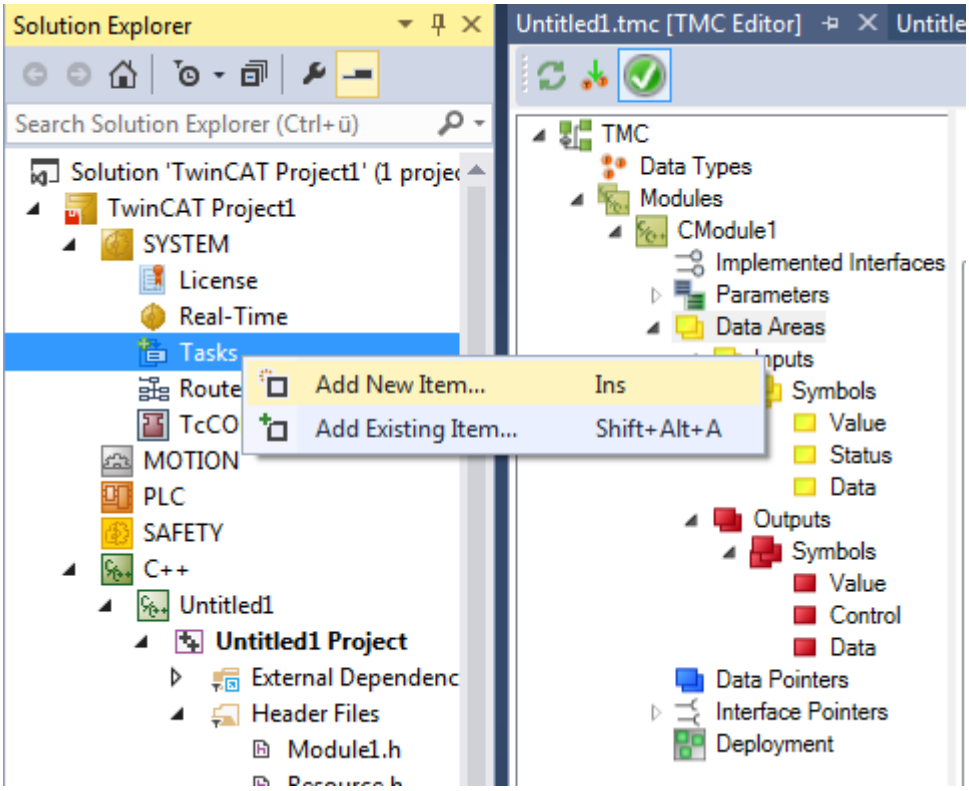
9.9 创建 TwinCAT 任务并将其应用于模块实例

本页介绍模块实例与任务的关联，以便 TwinCAT 实时系统调用模块的循环接口。

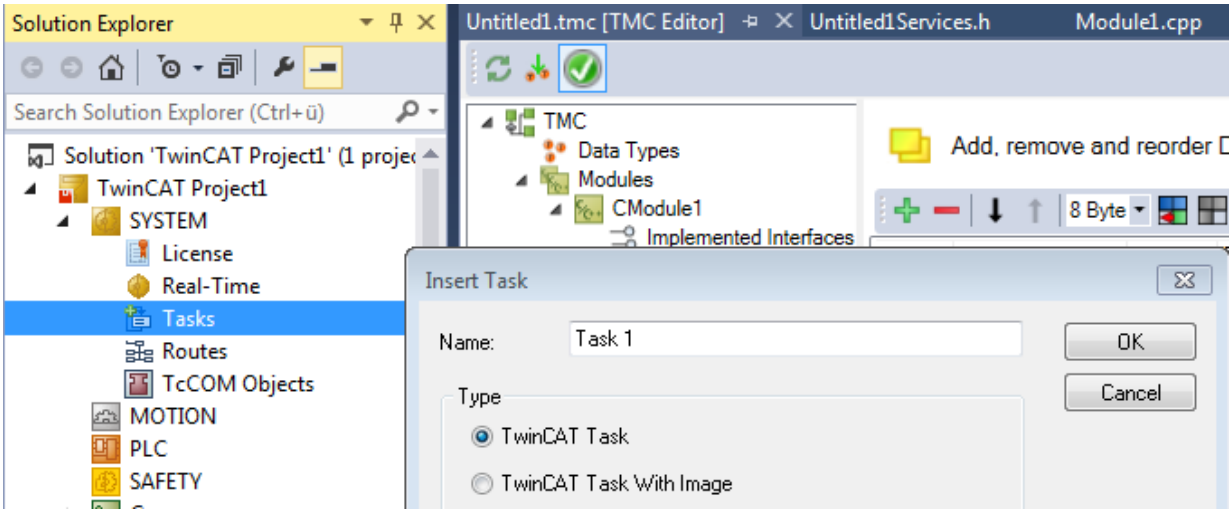
此配置步骤只需执行一次。在同一项目中后续创建/新编译 C++ 模块无需配置新任务。

创建 TwinCAT 3 任务

1. 打开**系统**，右键单击**任务**并选择**添加新项目……**。

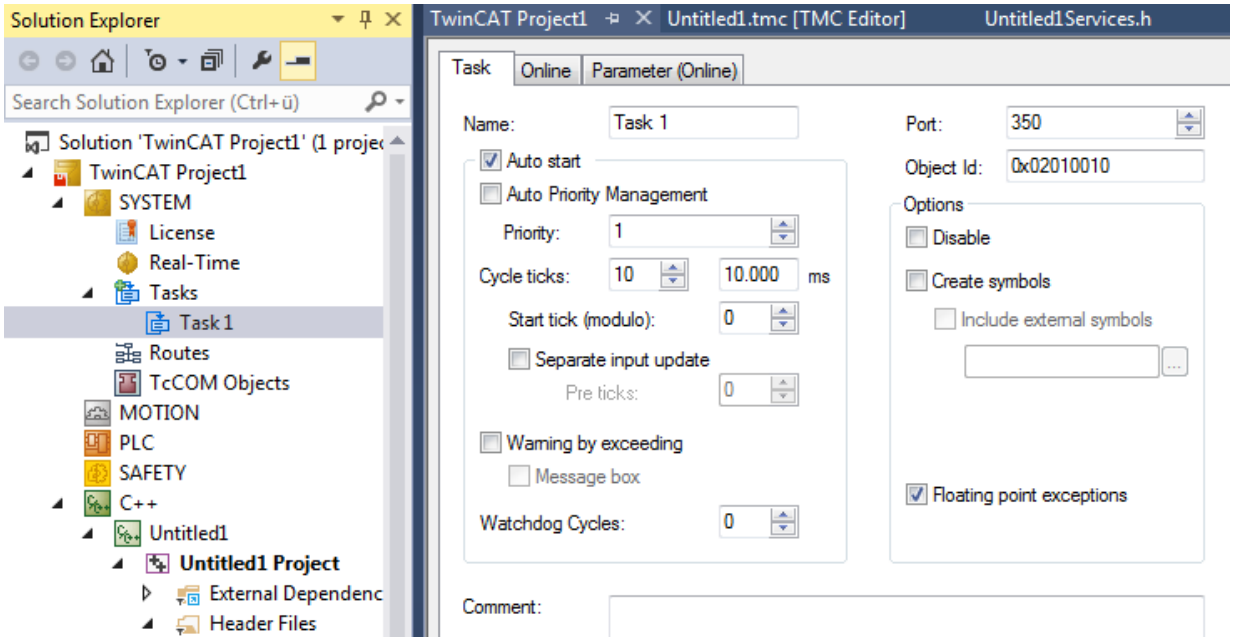


2. 输入任务的唯一名称（或保留默认名称）。
在本例中，I/O 映像接口由 C++ 模块实例提供，因此在任务中无需额外的映像来触发 C++ 模块实例的执行。



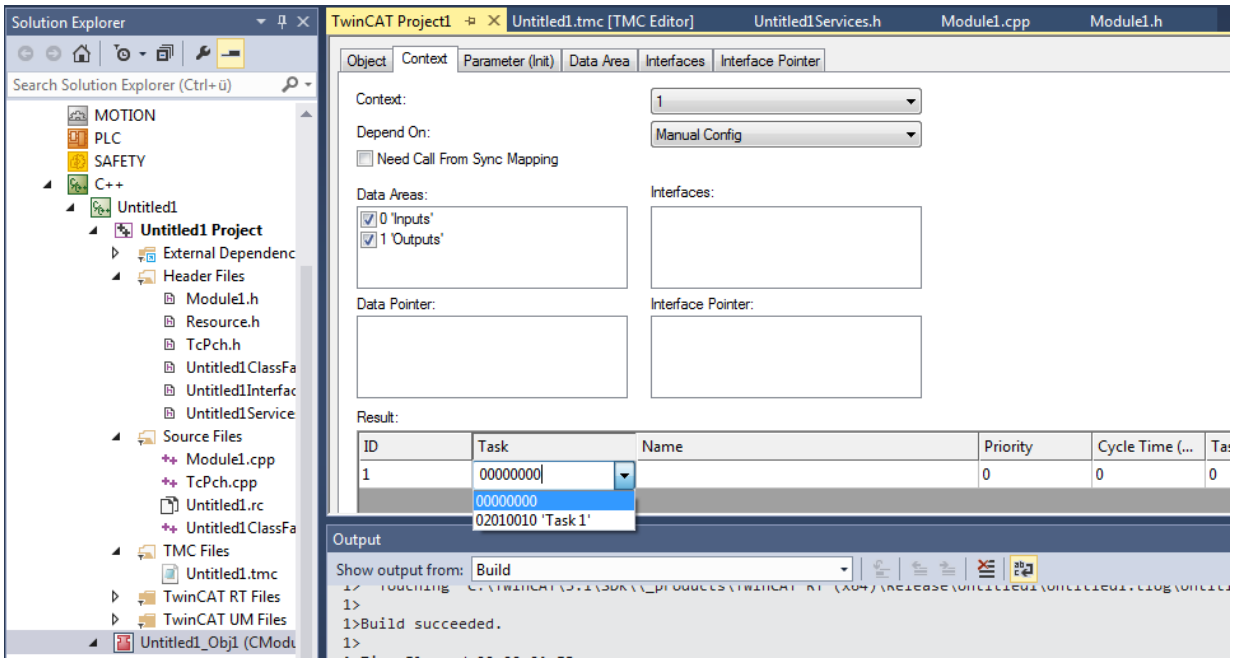
⇨ 已创建名称为“任务 1”的新任务。

3. 现在可以对任务进行配置；双击任务即可完成配置。
最重要的参数是**自动启动**和**优先级**：
必须激活**自动启动**，才能自动启动循环执行的任务。**Cycle ticks**定义了时钟相对于基准时钟的时间（请参见实时设置）。



配置从任务中调用的 TwinCAT 3 C++ 模块实例

1. 在解决方案树中选择 C++ 模块实例。
2. 在右侧工作区选择**Context**选项卡。
3. 在任务下拉菜单中为先前创建的Context任务。
   选择示例中的默认**Task1**。

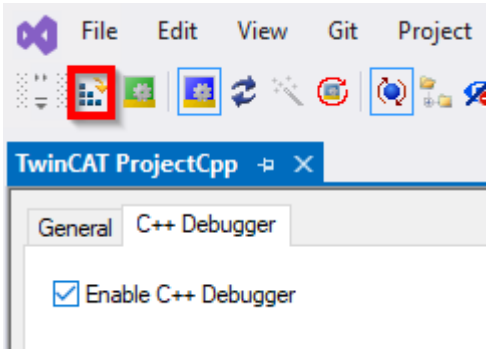


⇨ 完成该步骤后，**接口指针**将被配置为 **CyclicCaller**。至此，配置工作就完成了！

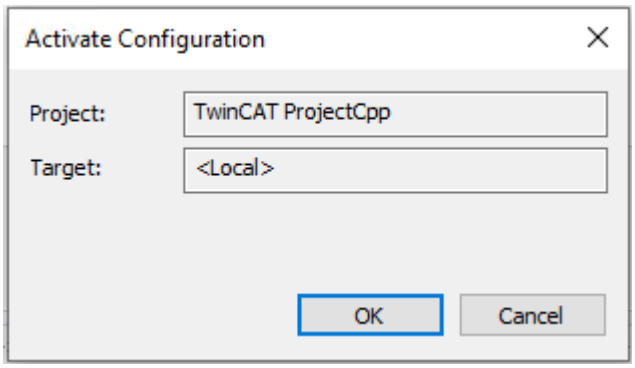
9.10 激活 TwinCAT 3 项目

创建、编译并提供 TwinCAT C++ 项目之后，就必须激活配置：

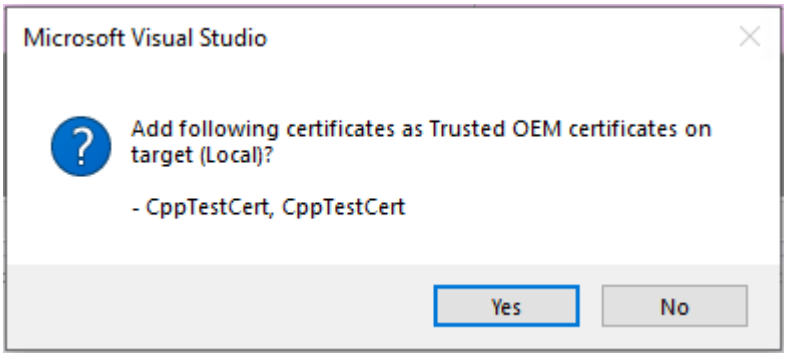
1. 点击**激活配置**符号，TwinCAT 项目所需的所有文件会传输到目标系统：



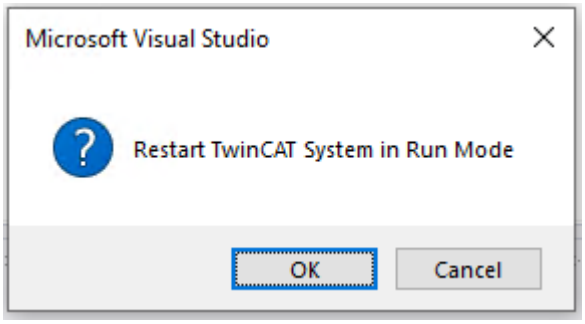
2. 下一步，确认激活新配置。先前的旧配置将被覆盖。



⇨ 执行所需的证书会显示出来，并可自动获批：



3. 如果目标系统上没有授权，则会提供创建 7 天试用授权的选项。这些授权码可以重复使用任意次数。
4. TwinCAT 3 会自动询问是否应将模式切换为“运行”模式。



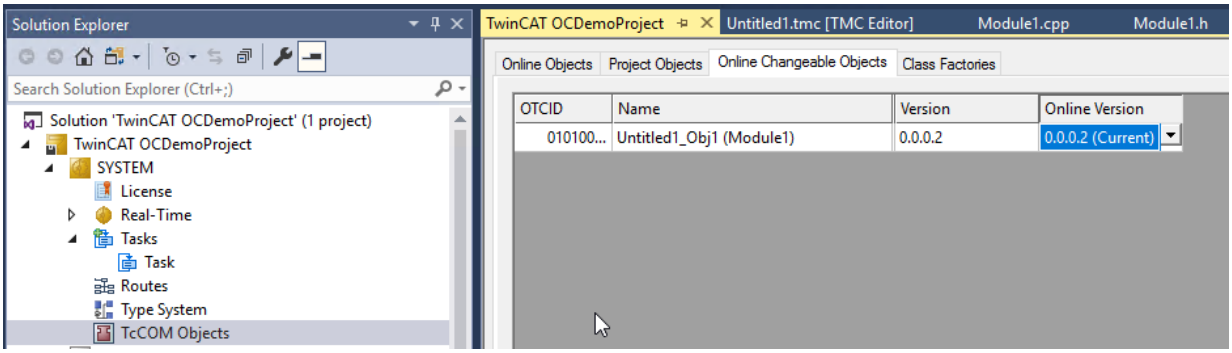
⇨ 点击**确定**后，TwinCAT 3 项目改为“运行”模式。
点击**取消**后，TwinCAT 3 将保持在**“配置”模式**。

⇨ 切换为“运行”模式后，Visual Studio 中底部的 TwinCAT 系统服务符号会亮起绿灯。

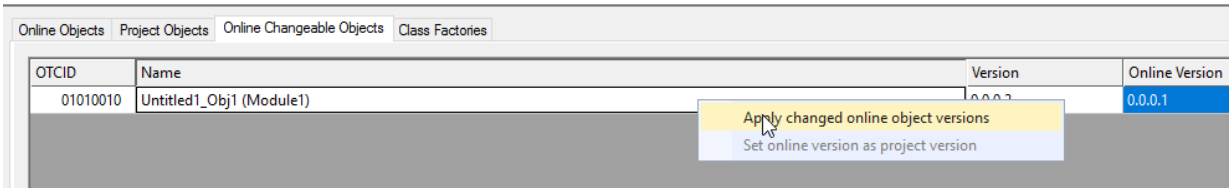
9.11 TwinCAT 3 C++ 实现项目在线更改

只有在先前已为“在线更改”准备好模块的情况下，才需要执行这些步骤。

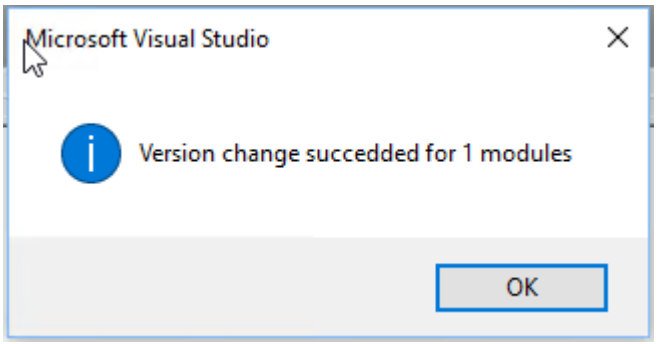
✓ 如上所述，运行 TwinCAT C++ 项目。

1. 切换到**系统**区域的 **TcCOM 对象**概览，然后切换到**在线可更改对象**选项卡。
2. 

3. 在“**在线版本**”一栏中，已预选当前运行的版本，并标有扩展名（当前）。
4. 设置不同的版本。
5. 右键单击以激活此次更改，然后在目标上**应用更改后的在线对象版本**。



⇨ 在目标上已进行版本更改



还请参阅有关此

激活 TwinCAT 3 项目 [▶ 71]

1 调试

0

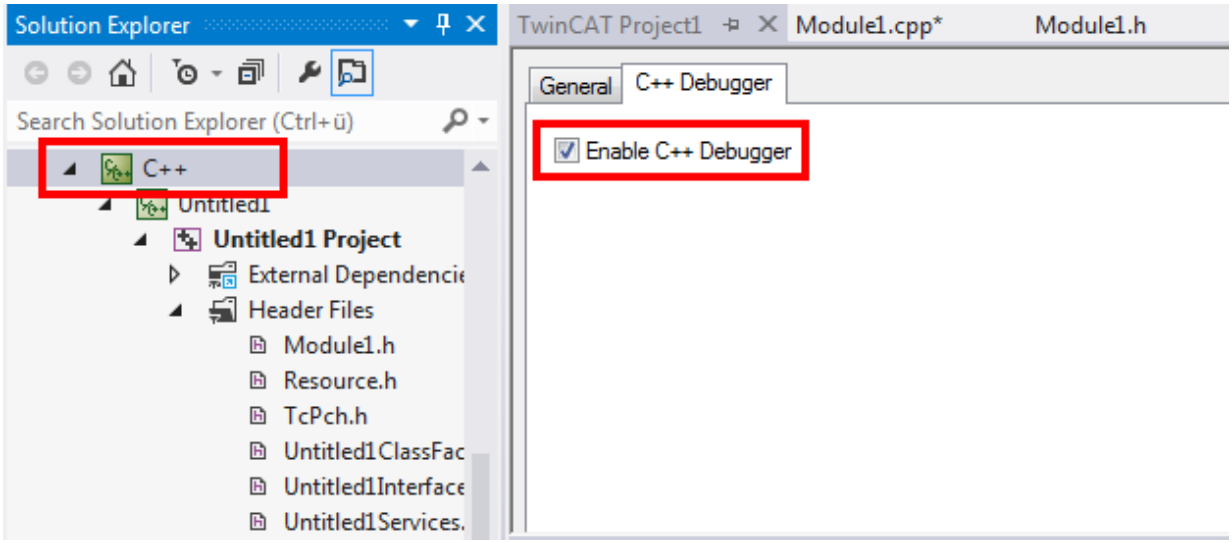
TwinCAT C++ 可为调试实时条件下运行的 TwinCAT C++ 模块提供各种机制。
其中大多数机制都与常规 C++ 开发环境中熟悉的机制相对应。自动化领域需要额外不同的调试机构，相关内容已在此处记录。

此外，我们还概述了可在 TwinCAT 3 中使用的 Visual Studio 工具。我们对这些工具进行了扩展，以便显示来自目标系统的数据。

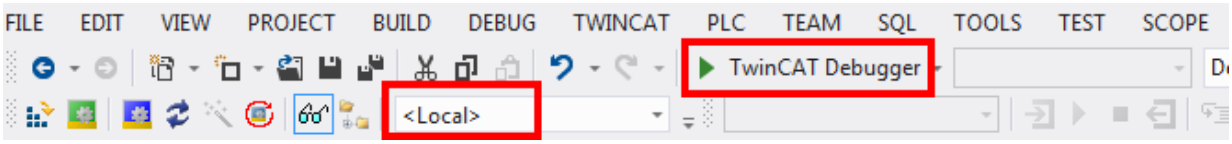
必须启用调试功能。

可通过解决方案的 C++ 节点进行配置：

1. 双击 C++ 节点，切换为 **C++ 调试器**选项卡，访问复选框。



2. 对于 TwinCAT C++ 中的所有调试，请通过 **TwinCAT 调试器**按钮将 TwinCAT 工程与运行时系统 (XAR) 连接。

3. 

断点和逐步执行

在大多数情况下，调试 C++ 程序时都会设置断点，然后在观察变量、指针等项目的同时逐步执行代码。

在 Visual Studio 调试环境中，TwinCAT 提供了逐步运行实时执行代码的选项。要设置断点，可以浏览代码并点击左侧与代码相邻的灰色列，或者使用快捷键（通常为 F9）。

⚠ 警告

由于机器/工厂的意外行为而造成的工厂受损和人身伤害

断点可改变机器或工厂的行为。根据不同的受控设备，可能会损坏机器或工件，或者可能会危及人员的健康和生命。

确保控制系统的行为改变不会造成任何损害，并且务必注意工厂文档。

```
///<AutoGeneratedContent id="ImplementationOf_ITcCyclic">
HRESULT CModule1::CycleUpdate(ITcTask* ipTask, ITcUnknown* ipCaller, ULONG_PTR context)
{
    HRESULT hr = S_OK;

    // TODO: Replace the sample with your cyclic code
    m_counter+=m_Inputs.Value;
```

当到达断点（箭头所示）时，代码会停止执行。

```
///<AutoGeneratedContent id="ImplementationOf_ITcCyclic">
HRESULT CModule1::CycleUpdate(ITcTask* ipTask, ITcUnknown* ipCaller, ULONG_PTR context)
{
    HRESULT hr = S_OK;

    // TODO: Replace the sample with your cyclic code
```

按下 **Step Over（单步执行）**（调试菜单、工具栏或快捷键 F10），会逐步执行代码。此外，还可以使用常用的 Visual Studio 功能**步入** (F11) 和**步出** (Shift + F11)。

条件断点

更先进的技术允许设置条件断点，只有在满足某个条件时，代码执行才会在断点处停止。

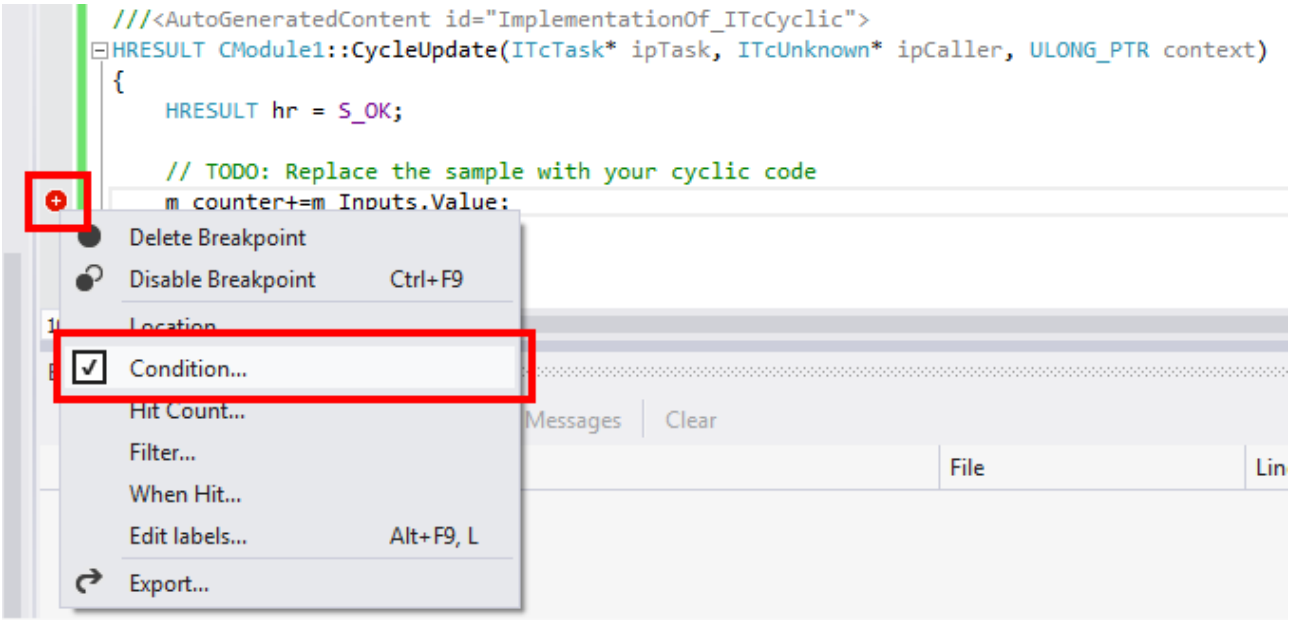
作为 Visual Studio 集成的一部分，TwinCAT 可提供条件断点的实现功能。要设置条件，首先要设置一个常规断点，然后右键单击断点列中的红点。

⚠ 警告

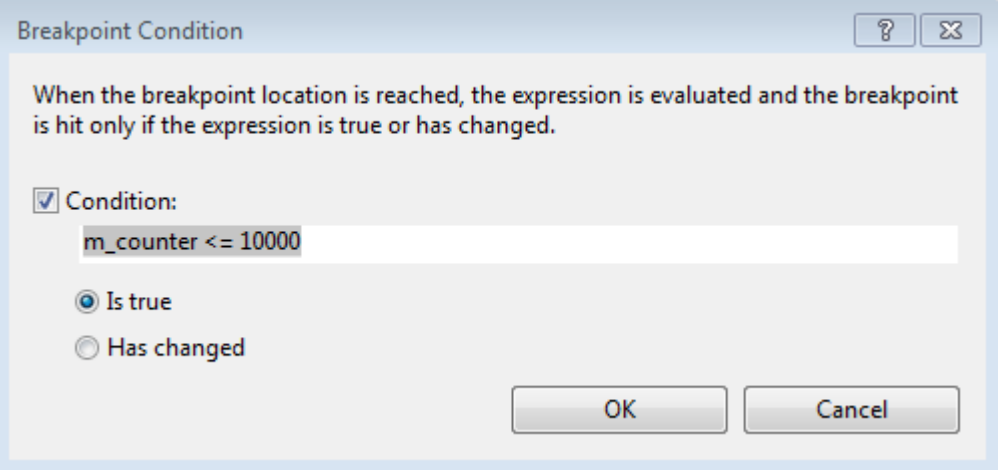
由于机器/工厂的意外行为而造成的工厂受损和人身伤害

断点可改变机器或工厂的行为。根据不同的受控设备，可能会损坏机器或工件，或者可能会危及人员的健康和生命。

确保控制系统的行为改变不会造成任何损害，并且务必注意工厂文档。



选择 **Condition...（条件.....）**打开条件窗口：



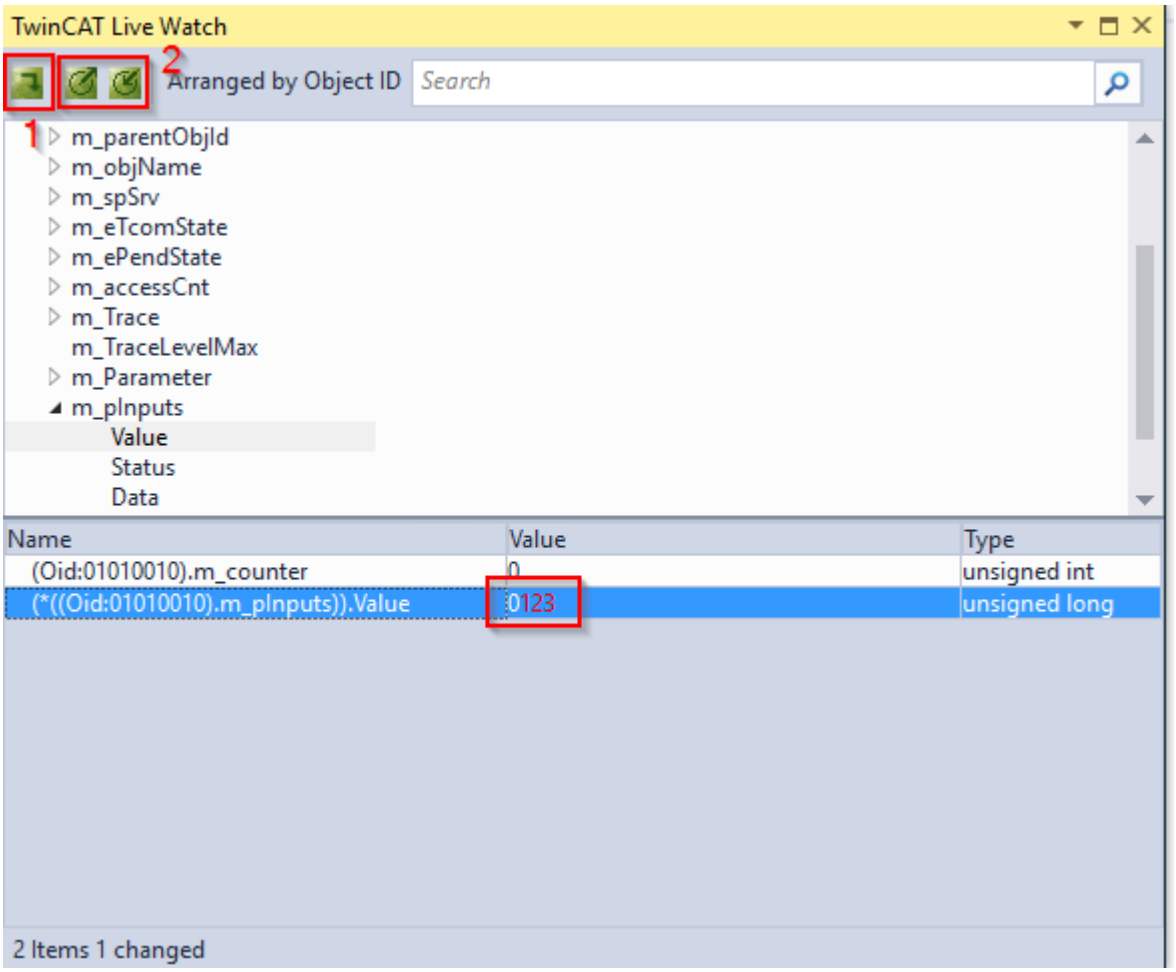
有关更多信息，请参见章节：条件断点详细信息 [▶ 77]。

实时监视

在对机器进行工程设计和开发时，并不总是建议在断点处停止系统，因为这会影响行为。
TwinCAT PLC 项目可在线查看和处理“运行”状态下的变量，无需中断实时的运行。

TwinCAT C++ 项目通过**实时监视（Live Watch）**窗口为 C++ 代码提供类似功能。

Live Watch（实时监视）窗口可通过 **Debug（调试）** -> **Windows** -> **TwinCAT Live Watch（TwinCAT 实时监视）**打开。
要打开该窗口，首先要与实时系统建立连接（按下 **TwinCAT Debugger（TwinCAT 调试器）**按钮），此时 Visual Studio 会切换为调试视图，否则无法提供数据。



TwinCAT 实时监视（Live Watch）窗口分为两个区域。

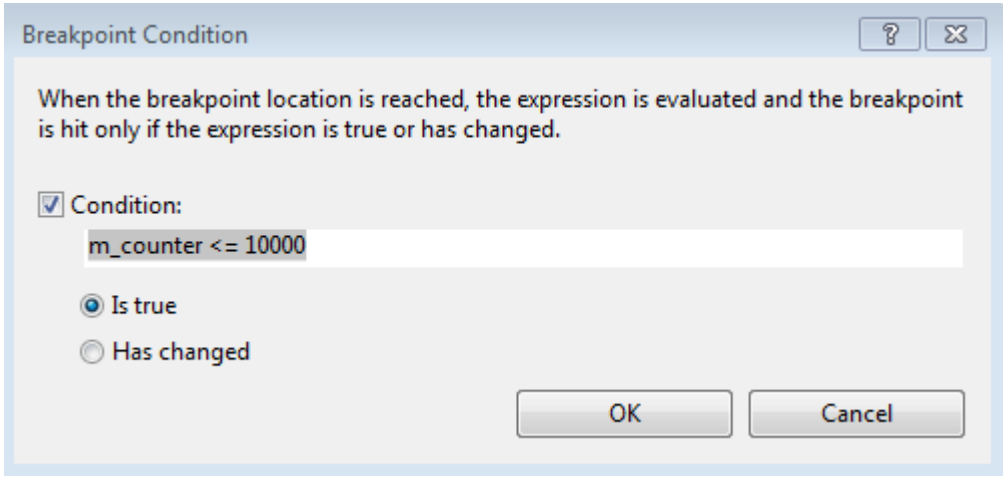
在上部区域，可以检索所有成员变量。双击后，便会添加到下部区域，并在该区域显示当前值。

点击**值**字段中的值，即可编辑这些值。新值以红色突出显示。要写入值，请按下左上角的符号 (1)。

使用 (2) 下的导入和导出符号，可以保存所选成员变量并在稍后恢复。

10.1 条件断点详细信息

TwinCAT C++ 可提供条件断点。有关这些条件的制定细节，请在此处查阅。



与 Visual Studio C++ 条件断点不同，TwinCAT 条件经过编译，随后传输到目标系统，因此可以在较短的周期内使用。

⚠ 警告

由于机器/工厂的意外行为而造成的工厂受损和人身伤害

断点可改变机器或工厂的行为。根据不同的受控设备，可能会损坏机器或工件，或者可能会危及人员的健康和生命。

确保控制系统的行为改变不会造成任何损害，并且务必注意工厂文档。

选项按钮可提供两个选项，将分别进行说明。

说明：条件为真

借助逻辑术语定义条件，类似于合取范式（Conjunctive Normal Forms）。
其由与“&&”相连的 Maxterm 组合而成：

```
(Maxterm1 && Maxterm2 && ... && MaxtermN)
```

其中每个 Maxterm 表示 || 相关条件的组合：

```
(condition1 ||condition2 || ... || conditionN )
```

可能的比较运算符：`==`, `!=`, `<=`, `>=`, `<`, `>`

观察“实时监控（Live Watch）”窗口，确定可用变量。所有列出的变量都可用于条件设定。其中包括 TMC 定义的符号以及本地成员变量。

示例：

```
m_counter == 123 && hr !=  0
```

```
m_counter == 123 || m_counter2 == 321 && hr == 0
```

```
m_counter == 123
```

监控模块实例

对象的 OID 存储在 `m_objId` 中，因此对 OID 的监控如下所示 `m_objId == 0x01010010`

任务监控

提供一个特殊变量 `#taskId`，用于访问调用任务的 OID。例如：`#taskID == 0x02010010`

选项：已更改

“已更改”选项很容易理解：通过提供变量名，如果该值与前一周期相比发生变化，则会对该值进行监控并保持执行。

示例：

```
m_counter
```

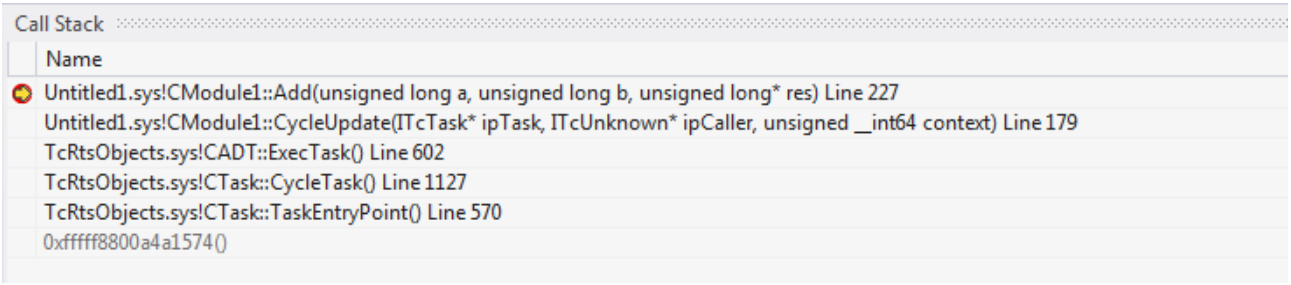
```
m_counter && m_counter2
```

10.2 Visual Studio 工具

Visual Studio 可为 C++ 开发人员提供常用的开发和调试工具。TwinCAT 3 可对这些 Visual Studio 工具进行扩展，因此在 TwinCAT 3 工程中也可以使用 Visual Studio 工具调试在目标系统上运行的 C++ 代码。
在此简要介绍相应的高级工具。如果相应窗口在 Visual Studio 中不可见，可通过菜单项 **Debug（调试）** -> **Windows** 进行添加。菜单取决于环境，即此处功能说明的许多窗口仅在调试器与目标系统建立连接后，才可进行配置。

调用堆栈

达到断点时，**调用堆栈**工具窗口会显示“调用堆栈（Call Stack）”。



自动/本地与观察

达到断点时，相应的变量和值会显示在**自动**/**本地**窗口中。更改项以红色显示。

Locals

Name	Value	Type
this	0xfffffa801a1d57b0	CModule1*
ITComObject	{}	ITComObject
ITcADI	{}	ITcADI
ITcWatchSource	{}	ITcWatchSource
ITcCyclic	{}	ITcCyclic
Calc	{}	Calc
m_refCnt	{value=2 }	AUTO_ULONG
m_objId	{value=16842768 }	AUTO_ULONG
m_parentObjId	{value=0 }	AUTO_ULONG
m_objName	{str={...} }	AUTO_NAMESTR
m_spSrv	{m_pInterface={...} m_oid=0 }	_tc_com_ptr_t<_tc_com_IID<ITCo
m_eTcomState	{value={...} }	AUTO_TCOM_STATE
m_ePendState	{value={...} }	AUTO_TCOM_STATE_INVALID
m_accessCnt	{value=1 }	AUTO_ULONG
m_Trace	{m_TraceLevelMax={...} m_spSrv={...} }	CTcTrace
m_TraceLevelMax	tlAlways (0)	TcTraceLevel
m_Parameter	{data1=0 data2=0 data3=0.0 }	_Module1Parameter
m_Inputs	{Value=123 Status=0 Data=0 }	_Module1Inputs
m_Outputs	{Value=108117 Control=0 Data=0 }	_Module1Outputs
m_spCyclicCaller	{m_info={...} }	_tc_com_ptr_t_listinfo<_tc_com_III
m_counter	0	unsigned int
hr	21	HRESULT
a	123	unsigned long
b	108117	unsigned long
res	0xfffffa801a1d5824	unsigned long*
	108117	unsigned long

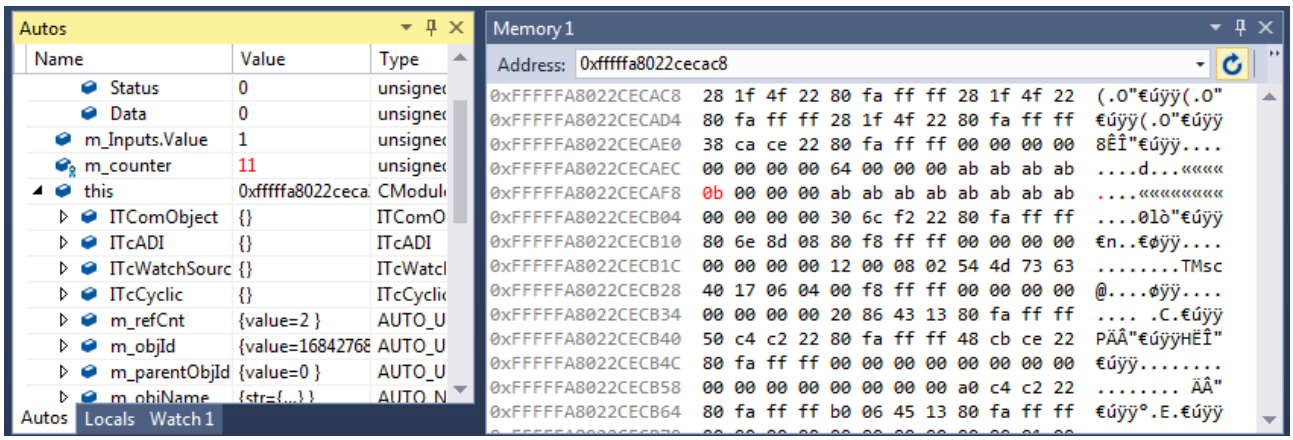
在此处，可以通过右键单击将值应用到**监视**窗口：

Watch 1

Name	Value
a	123
b	108117
*(res)	108117

内存视图

可直接监控内存。更改项以红色显示。



10.3 使用 TwinCAT Task Dumps

TwinCAT 3.1 Build 4026 及以上版本

TwinCAT 3.1 Build 4026 及以上版本可用该功能。

如果出现异常，TwinCAT 运行时会自动将“TwinCAT 任务转储”写入启动目录（通常为 C:\ProgramData\Beckhoff\TwinCAT\3.1\Boot），前提是应用程序未对其进行处理。

TwinCAT C++ 开发人员可以在搭载用于 TwinCAT C++ 开发的 Visual Studio 开发系统上传输和评估这些“任务转储”。

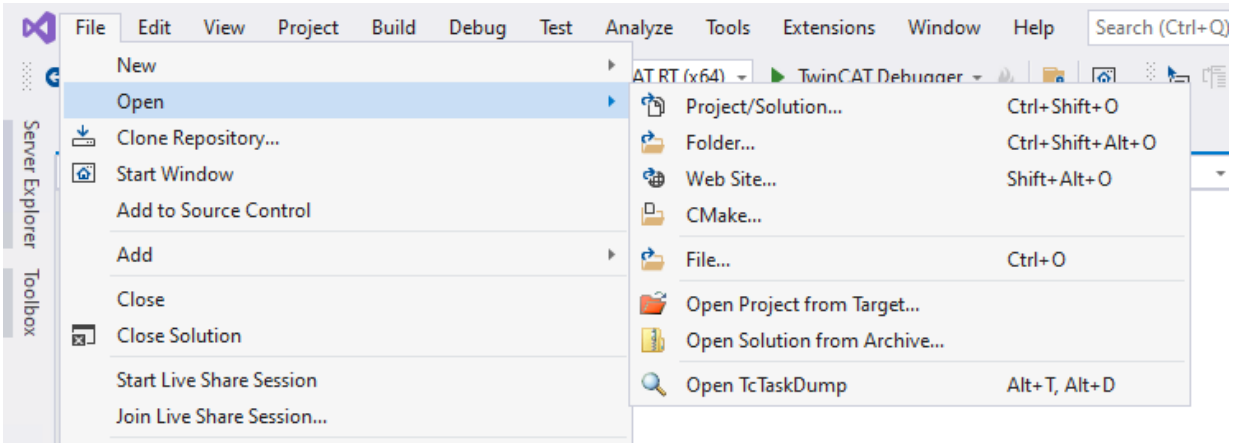
为了进行调查，除了 TwinCAT 任务转储外，还必须提供准确对应的 TMX 和 PDB 文件。

如果直接从相应的解决方案中加载 TwinCAT 任务转储，则会自动找到这些文件。如果 TwinCAT 任务转储是从一个空“解决方案”中加载，那么相关 TMX 和 PDB 文件可以与 TwinCAT 任务转储放在同一目录下。或者，也可以在 Visual Studio 选项（工具 -> 选项 -> 调试 -> 符号：添加符号文件 (.pdb) 位置的新路径）中指定搜索路径。

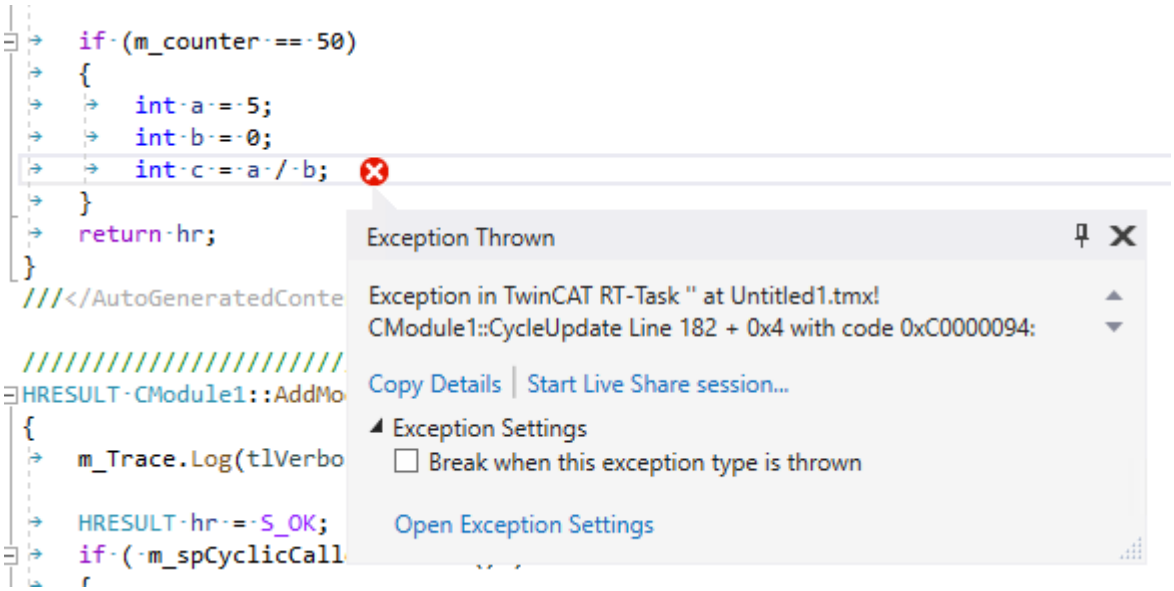
操作步骤如下：

✓ 目标系统出现异常。

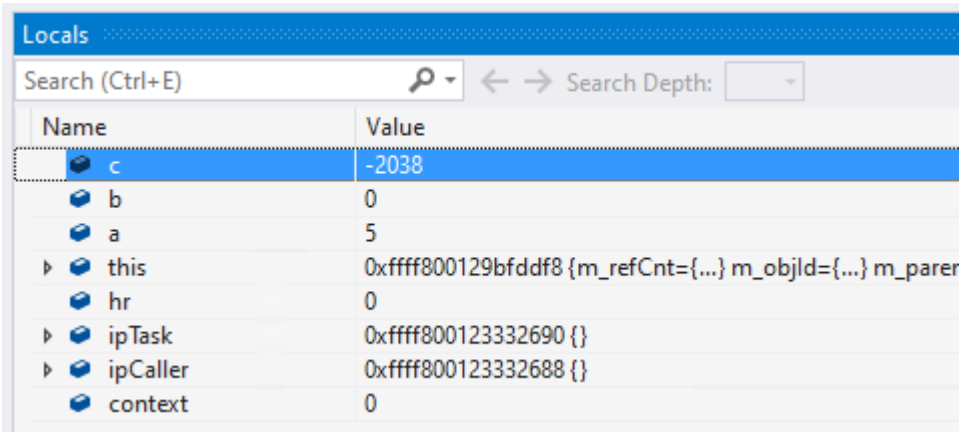
1. TwinCAT 任务转储文件从目标系统传输到工程系统。其名称类似 taskmemory-*.dmp
2. 理想情况下，TwinCAT 项目的相应源代码已可供使用，并已在 Visual Studio 中打开。
3. 该文件可通过 File（文件） -> Open-（打开-） > Open TcTaskDump（打开 TcTaskDump）打开：



4. 编辑器会直接跳转到触发异常的相应代码位置：



此外，还可以使用标准 Visual Studio 窗口来控制变量值，例如：



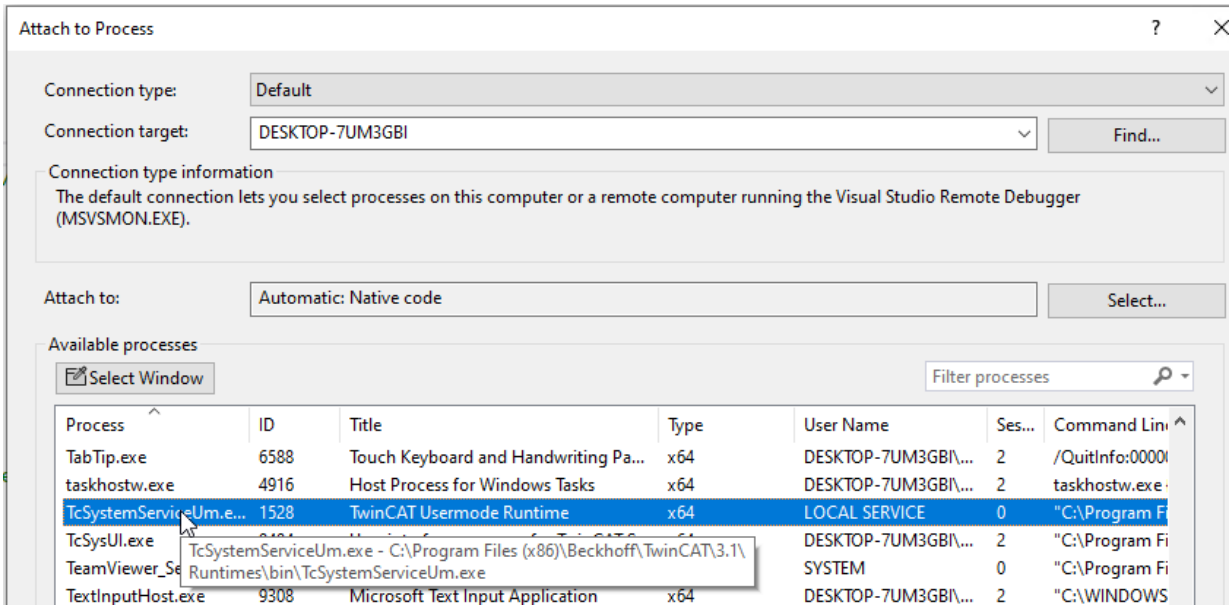
10.4 使用 Usermode Runtime 调试状态机

在实时环境中无法调试 TcCOM 模块的状态机，因为此时调试器本身尚未启动。
为此，可以使用 Usermode Runtime。

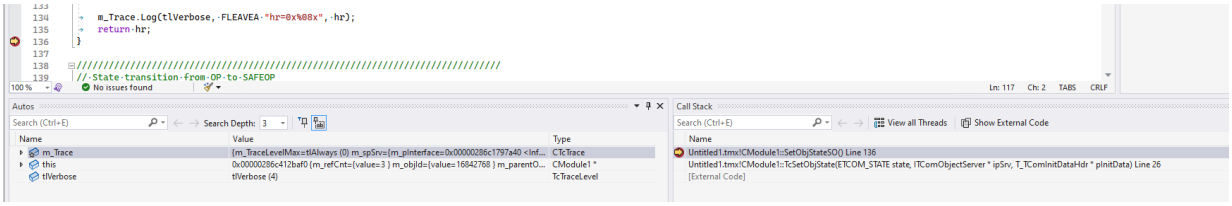
✓ 在 Visual Studio 中加载 TwinCAT C++ 项目

1. 在 Usermode Runtime 中激活项目
2. 打开第二个 Visual Studio 实例

3. 使用常规 Visual Studio C++ 调试器连接 Usermode Runtime 进程。Debug（调试） -> Attach to Process（连接到进程） ... ... -> Selection of TcSystemServiceUm.exe（选择 TcSystemServiceUm.exe）



4. 在这种情况下，添加要调试的 TwinCAT C++ 项目。在转换过程中设置断点。
5. 启动 Usermode Runtime - 到达断点。



1 向导

1

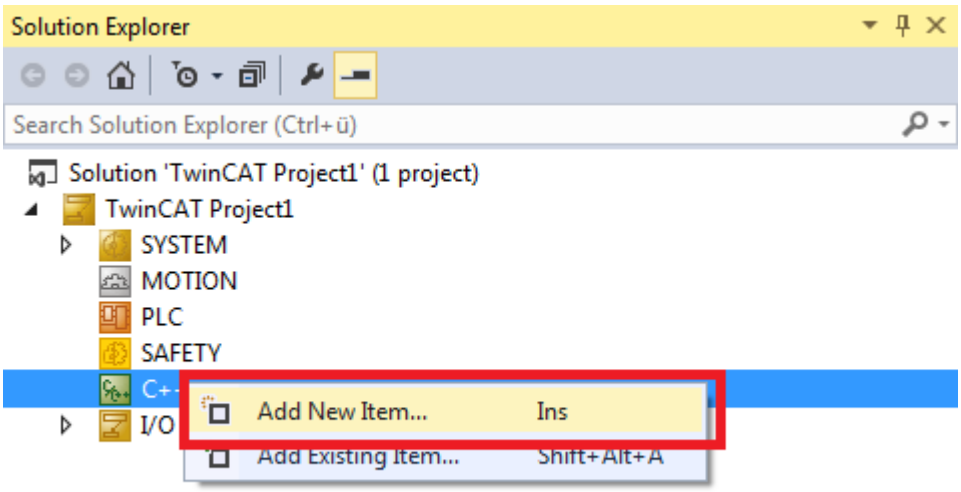
为便于熟悉 TwinCAT C++ 系统的工程设计程序，可使用向导工具。

- TwinCAT 项目向导 [▶ 83]会创建一个 TwinCAT C++ 项目。对于驱动程序项目，则启动 TwinCAT 类向导。
- 创建 C++ 模块时，会自动启动 TwinCAT 模块类向导 [▶ 83]。
  该向导会提供各种“即用型”项目，作为您自行开发的起点。
- TwinCAT 模块类编辑器 [▶ 86] (TMC) 是用于定义数据结构、参数、数据区、接口和指针的图形编辑器。它会创建一个 TMC 文件，供 TMC Code Generator 使用。
- 实例根据定义的类生成，可通过 TwinCAT 模块实例配置器 [▶ 129] 进行配置。

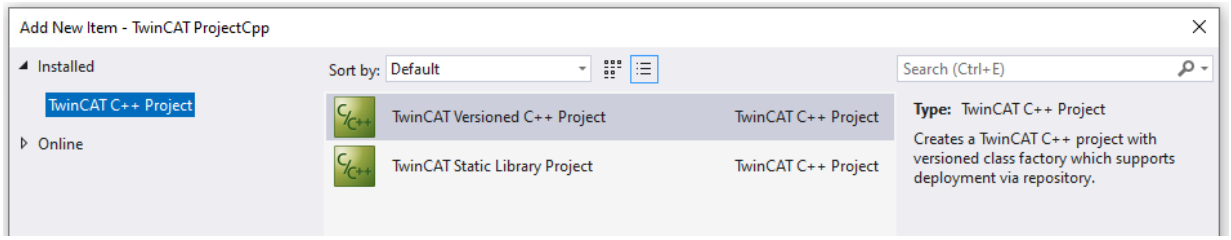
11.1 TwinCAT C++ 项目向导

创建 TwinCAT 项目后，您可以借助 TwinCAT C++ 项目向导添加一个 C++ 项目：

1. 右键单击 C++ 图标，选择 **添加新项目.....**，启动 C++ 项目图标。



TwinCAT 可提供两种 C++ 项目：
- 版本控制 C++ 项目：涉及版本控制的项目还可以选择在运行时在不同版本之间进行切换。
- 静态库：带有由（不同）TwinCAT C++ 驱动程序使用的 C++ 功能的项目。



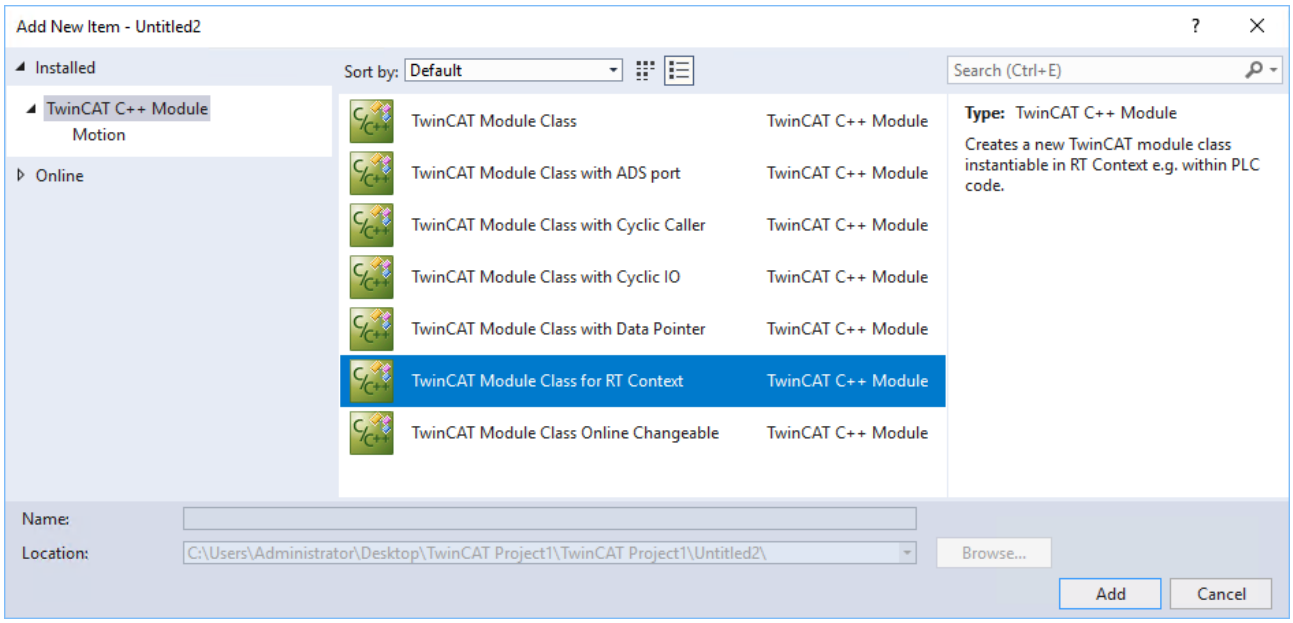
2. 选择其中一个项目模板，并指定名称和位置。
   ⇨ 已创建 TwinCAT C++ 项目

⇨ 如果是驱动程序，则启动 TwinCAT C++ 类向导 [▶ 83]。

11.2 TwinCAT 模块类向导

TwinCAT 3 可提供各种类模板，用于创建 TcCOM 对象类：

- TwinCAT 模块类
- 带 ADS 端口的 TwinCAT 模块类
- 带周期性调用的 TwinCAT 模块类
- 带循环输入/输出的 TwinCAT 模块类
- 带数据指针的 TwinCAT 模块类
- 用于实时环境的 TwinCAT 模块类
- 具有在线更改功能的 TwinCAT 模块类



TwinCAT 模块类

创建一个新的 TwinCAT 模块类。

此为生成基本核心模块的模板。在TwinCAT C++开发中，若需创建一个无周期性调用（Cyclic Caller）且无数据区（Data Area）的模块类，通常适用于实现按需调用（On-Demand）的服务。

例如，当创建将从 PLC 模块或其他 C++ 模块调用的 C++ 方法时。

请参见 Sample11 [▶ 256]

带 ADS 端口的 TwinCAT 模块类

该模板会提供 C++ 模块以及 ADS 服务器和 ADS 客户端的功能。

- ADS server：
  可以作为 C++ 模块的该模板的单个实例运行，并可预先配置特定 ADS 端口号（如 25023）。
  可启用该模板的多个实例，从而由 TwinCAT 3 为每个 C++ 模块分配各自唯一的 ADS 端口号（如 25023、25024、25025.....）。
  由于 C++ 模块的实现，可以分析和处理 ADS 消息。
  访问输入/输出数据区的 ADS 处理不必使用自己的 ADS 消息处理来实现。
- ADS 客户端：
  该模板可提供向 ADS 通信伙伴发送 ADS 消息以发起 ADS 调用的示例代码。

由于模块的行为类似于通过 ADS 消息相互通信的 ADS 客户端或 ADS 服务器，因此两个模块（Caller=Client 和 Called=Server）可以在相同或不同的实时环境中在相同或不同的 CPU 内核上运行。
由于 ADS 可以跨网络作业，因此这两个模块也可以在网络中的不同机器上运行。

请参见 Sample03 [▶ 238]， ADS 通信 [▶ 188]

带周期性调用的 TwinCAT 模块类

支持对与外部系统隔离的 C++ 程序周期调用。

（该模式）较少使用，通常更推荐采用**同时具备周期性调用（Cyclic Caller）和周期性I/O（Cyclic I/O）功能的模块类。

带循环输入/输出的 TwinCAT 模块类

创建一个新 TwinCAT 模块类，实现周期调用接口，并设立输入和输出数据区。

输入和输出数据区可与其他输入/输出映像或物理 I/O 端子模块相连。

重要信息：

C++ 模块拥有自己的逻辑输入/输出数据存储区。模块的数据区可通过系统管理器进行配置。

如果模块采用周期性接口（cyclic interface）进行映射，则输入和输出数据区的副本同时存在于两个模块（调用方和被调用方）中。这样模块便可在不同的实时环境下运行，甚至可以在与另一个模块相关的另一个 CPU 内核上运行。
TwinCAT 将在模块之间不断复制数据。

请参见 快速入门 [▶ 57]，sample 01 [▶ 237]。

带数据指针的 TwinCAT 模块类

正如带循环 IO 的 TwinCAT 模块类，该模板也会生成一个新 TwinCAT 模块类，可实现带有输入和输出数据区的调用接口，用于与其他逻辑输入/输出映像或物理 I/O 端子模块相连接。

此外，该模板还会提供数据指针，可用于通过指针访问其他模块的数据区。

重要信息：

在周期性 I/O 数据区中，数据在模块间周期复制，而在使用 C++ 数据指针的情况下则不同，只有一个数据区，属于目标模块。当通过数据指针机构从另一个 C++ 模块写入目标模块时，将立即影响目标模块的数据区。(无需等待周期结束）。

如果模块在运行时被调用，操作会立即执行并阻塞原进程（此为指针.....）。因此，两个模块（调用方和被调用方）必须处于同一实时环境中，并在同一个 CPU 内核上。

数据指针在 TwinCAT 模块实例配置器 [▶ 129] 中进行配置。

请参见 sample10 [▶ 255]

用于实时环境的 TwinCAT 模块类

该模板创建了一个模块，可在实时环境中对其进行实例化。

如 TwinCAT 模块状态机 [▶ 43] 章所述，其他模块在非实时环境中存在启动和关闭的转换。在某些情况下，必须在实时状态已经运行的情况下启动模块，以便在实时环境中执行所有转换。此为对应的模板。
具有这种（修改后）状态机的模块也可在启动 TwinCAT 时直接用于实例化。在这种情况下，转换的执行方式与常规模块相同。

TcCOM 03 示例 [▶ 316]说明了这种模块的应用。

具有Online Change（在线更改）功能的 TwinCAT 模块类

只有将模块添加到版本控制的 C++ 项目中时，才能使用该选项。

该模板创建了一个可进行在线更改的模块。由于项目的修订控制，这些模块在运行时可以交换，因此可以在运行时交换不同版本的模块。

在线更改 [▶ 148] 章介绍了Online Change的程序。

否则，模块本身就相当于一个具有循环输入/输出的模块。

11.3 TwinCAT 模块类编辑器 (TMC)

TwinCAT 模块类编辑器（TMC 编辑器）用于定义模块的类信息。其中包括数据类型定义及其应用、提供和实现的接口，以及数据区和数据指针。

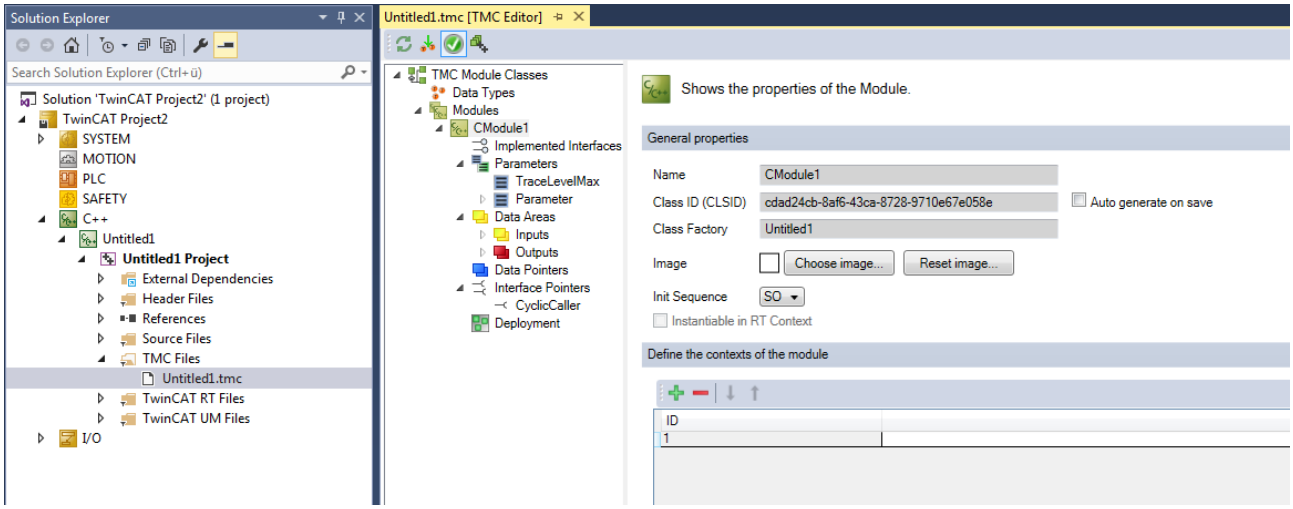
简而言之：所有外部可见的项目都必须用此编辑器来定义。

基本理念是：

1. TMC 编辑器可用于修改模块说明文件（TMC 文件）。其中包含 TwinCAT 系统本身可访问的所有信息。例如符号、实现的接口和参数。
2. TwinCAT 代码生成器也可从 TMC 编辑器中调用，用于生成所有需要的 C++ 代码，即报头文件和 cpp 文件。

启动 TMC 编辑器

双击模块的 TMC 文件，打开编辑器。打开图形编辑器：



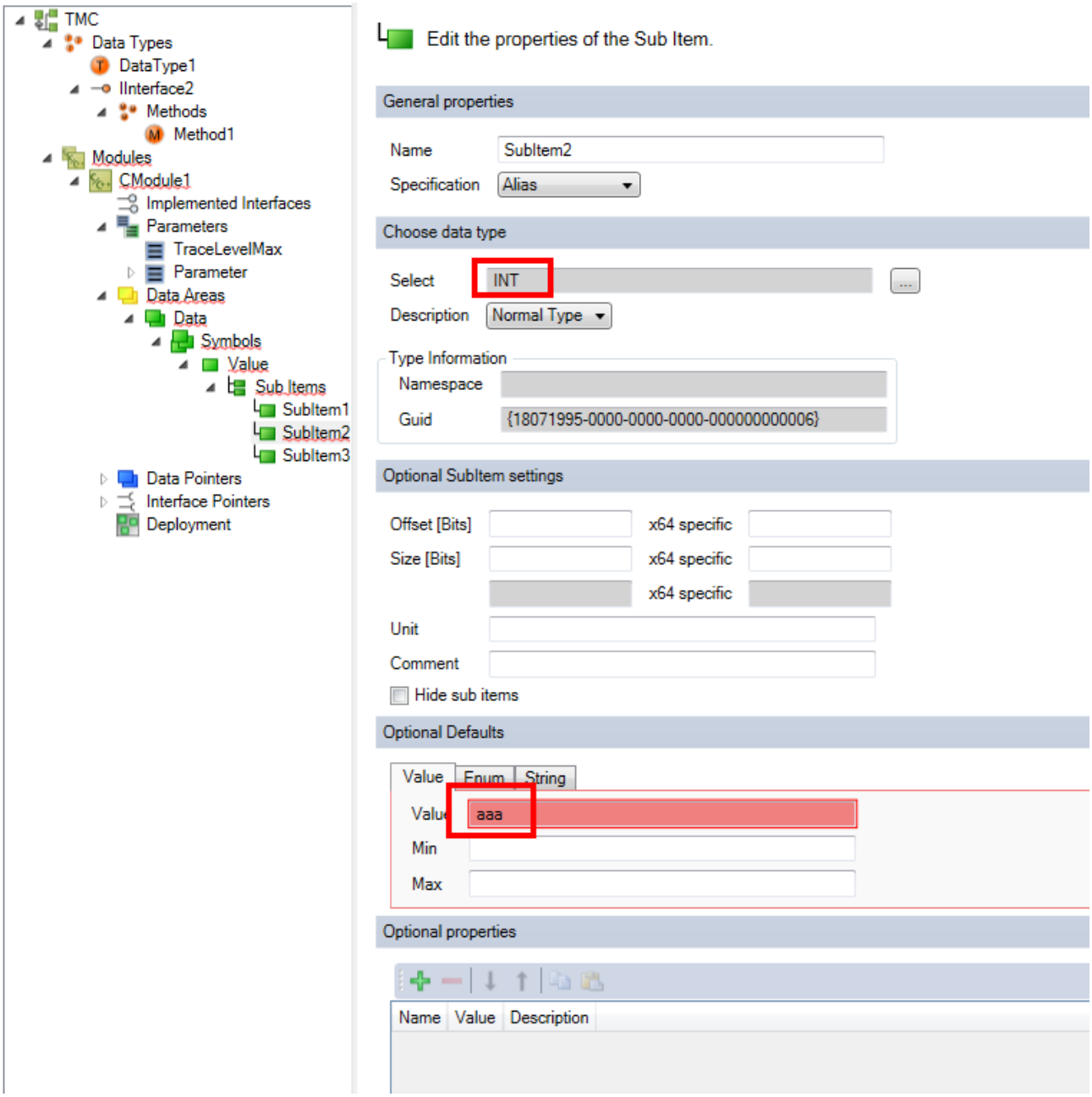
TMC 编辑器的功能：

- 在数据区创建/删除/编辑符号，例如模块的逻辑输入或输出过程映像。
- 创建/删除/编辑用户定义的数据类型定义。
- 创建/删除/编辑模块参数列表中的符号。

用户帮助

TMC 编辑器支持用户定义数据类型和 C++ 模块。

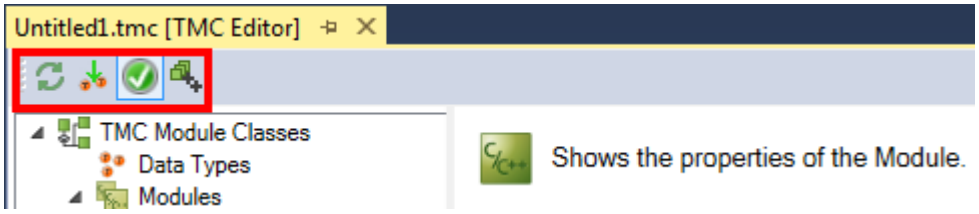
例如，当 TMC 内部出现问题（对齐、无效标准定义.....）时，会通过 TMC 树内的红色标记引导用户找到相关位置：



用户仍然可以直接编辑 TMC，由于其为 XML 文件，因此可以由用户创建和编辑。

工具

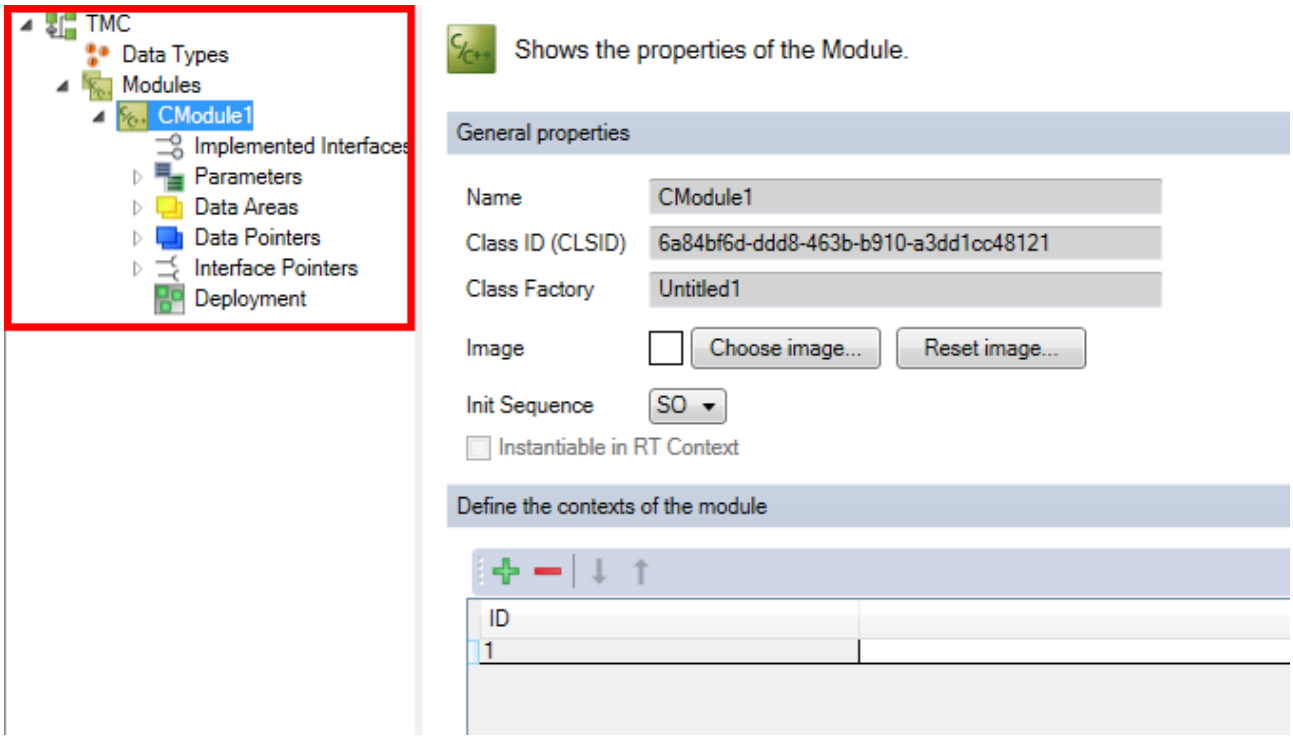
TMC 编辑器上部包含所需操作的符号。



- 从类型系统重新加载 TMC 文件和类型。
- 更新高级数据类型。
- 打开/关闭用户帮助 [▶ 86]。
- 启动 TwinCAT TMC Code Generator：

编辑器会将输入的信息存储到 TMC 文件中。TwinCAT TMC Code Generator 将此 TMC 功能说明转换为源代码，也可在 TwinCAT C++ 项目的环境菜单中找到源代码。

11.3.1 概述

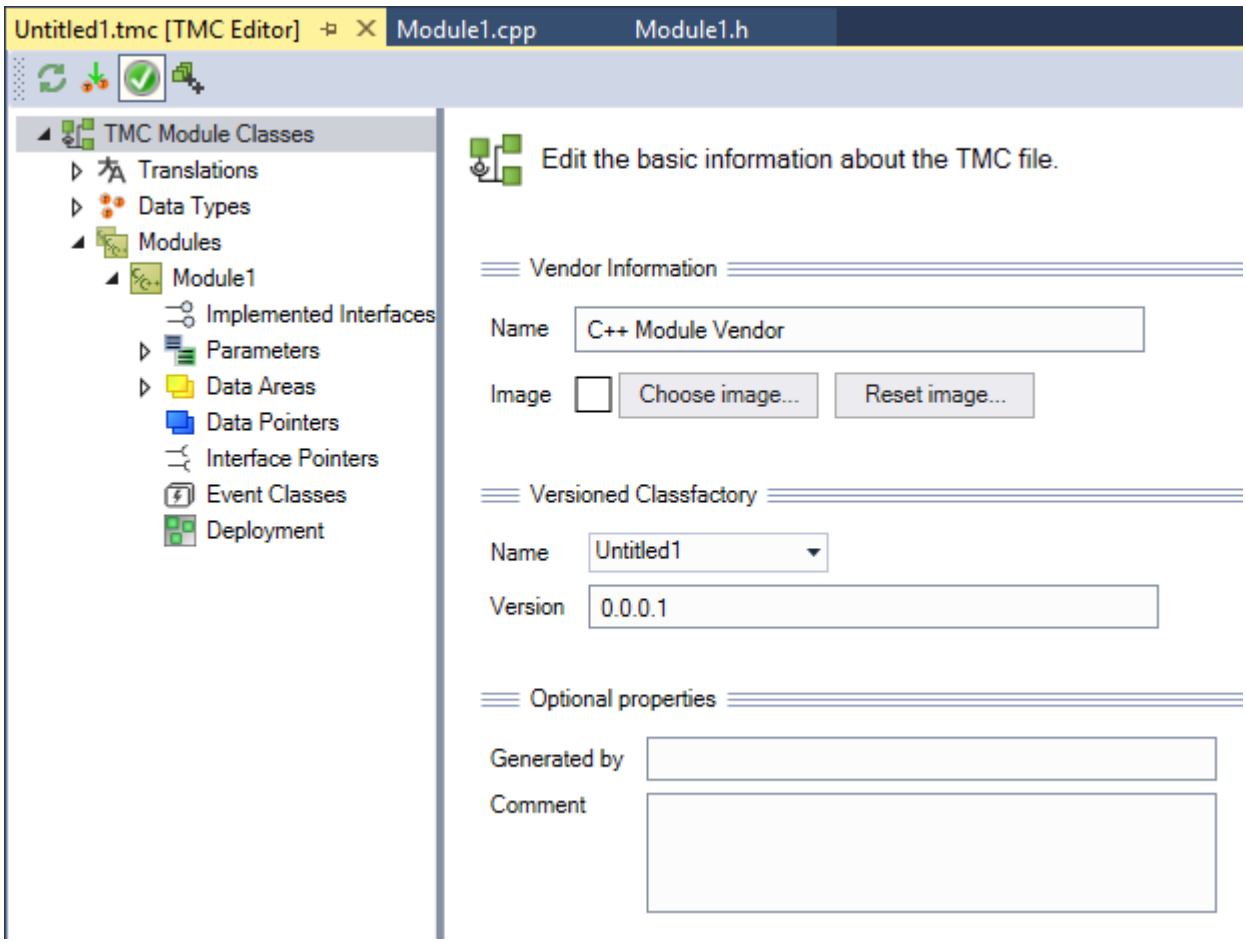


用户界面

- TMC [▶ 88]：在此可以编辑 C++ 模块供应商的基本信息并添加图像。
- 数据类型 [▶ 90]：在此添加、删除或重新排列数据类型。
- 模块 [▶ 107]：显示驱动程序的模块。
- 已实现接口 [▶ 109]：显示模块的已实现接口。
- 参数 [▶ 110]：在此添加、删除或重新排列参数。
  - TraceLevelMax [▶ 116]：控制记录消息数量的参数；（几乎）每个模块都已预定义。
- 数据区 [▶ 117]：在此添加、删除或重新排列数据区。
- 数据指针 [▶ 124]：数据指针在此添加、删除或重新排列。
- 接口指针 [▶ 126]：接口指针可在此处添加、删除或重新排列。
- 部署 [▶ 127]：确定提供的文件。

11.3.2 基本信息

有关 TMC 文件的基本信息，请参见此处：



关于提供方的信息

名称：此为提供方的名称。

选择图像：在此输入一个 16 x 16 像素的位图图标。

重置图像：将模块图像重置为标准值。

版本化的类工厂

名称：显示 TMC 文件引用的所有类工厂。必须设置实现 C++ 项目的类工厂。通常是项目的名称。
仅用于版本控制的 C++ 项目，否则此处显示“未设置”。

版本：当前版本：由四位数字组成，每个数字之间用“ ”分隔开来。. 至少有一个数字不能为 0。

可选功能

生成者：该字段表示创建和维护该文件的人员。请注意，在 TMC 编辑器中填写该字段时，无法再进行更改（停用所有编辑程序）。

注释：您可以选择在此处输入注释。

11.3.3 数据类型

用户可以通过 TwinCAT 模块类 (TMC) 编辑器定义数据类型。

这些数据类型可以是类型定义、结构、区域、枚举或接口，例如方法及其签名。

TwinCAT 工程系统 (XAE) 会发布与 TwinCAT 项目的所有其他嵌套项目相关的这些数据类型，因此这些数据类型也可用于 PLC 项目等（如 Sample11：模块通信：PLC 模块到 C++ 模块的方法调用 [▶ 256] 章所述）。

注意

名称冲突

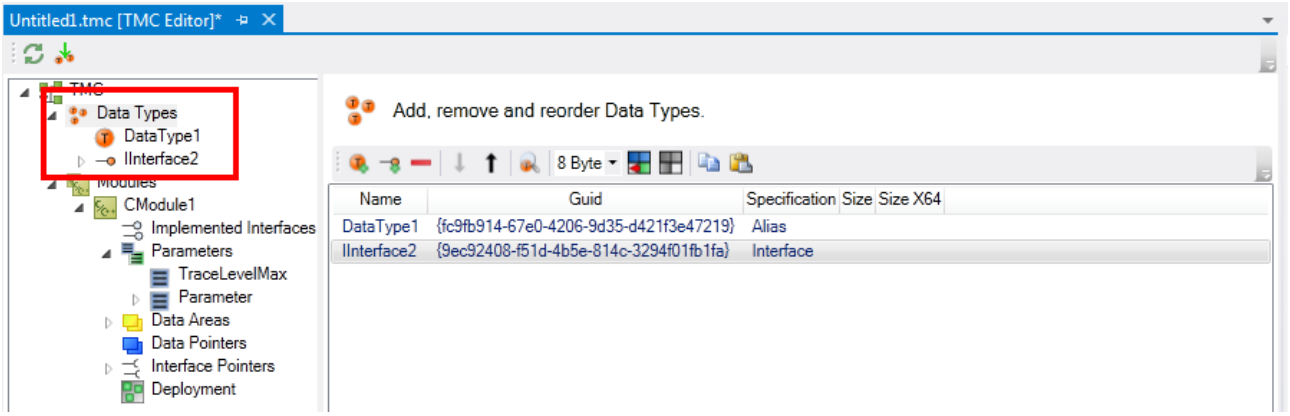
如果将驱动程序与 PLC 模块结合使用，则会发生名称冲突。

- 不要使用为 PLC 保留的关键词作为名称。

本章介绍如何使用 TMC 编辑器的功能来定义数据类型。

11.3.3.1 概述

用户界面



符号	功能
	添加新数据类型。
	添加新接口。
	删除所选类型。
	将所选元素向下移动一位。
	将所选元素向上移动一位。
	搜索未使用的类型。
	选择字节对齐。
	对齐所选数据类型（对齐）。 此功能会运行所有使用过的数据类型（递归）。如果不希望这样做，可以通过使用数据类型中的功能来逐步实现。
	重置所选数据类型的数据格式。
	复制
	粘贴

数据类型属性

名称：用户定义的数据类型名称。

GUID：数据类型的唯一 ID。

规范：数据类型的规范。

大小：数据类型的大小（如果明确指定）。

大小 X64：x64 平台数据类型的不同大小。

11.3.3.2 添加/修改/删除数据类型

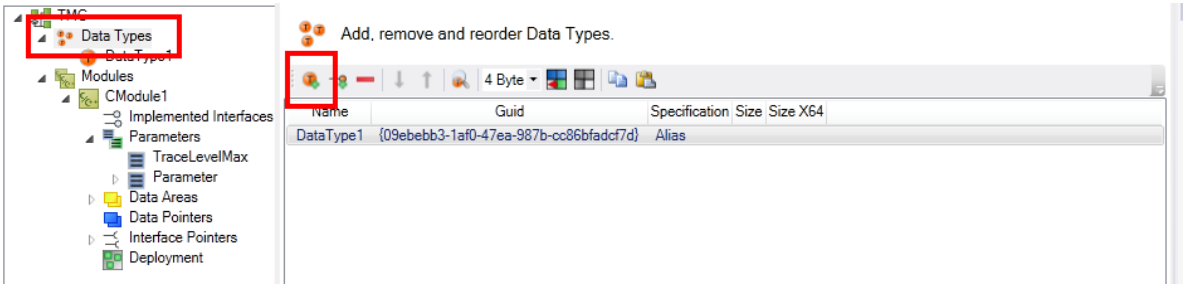
借助 TwinCAT 模块类 (TMC) 编辑器，可以添加、编辑和删除 TwinCAT C++ 模块使用的数据类型。

本文介绍：

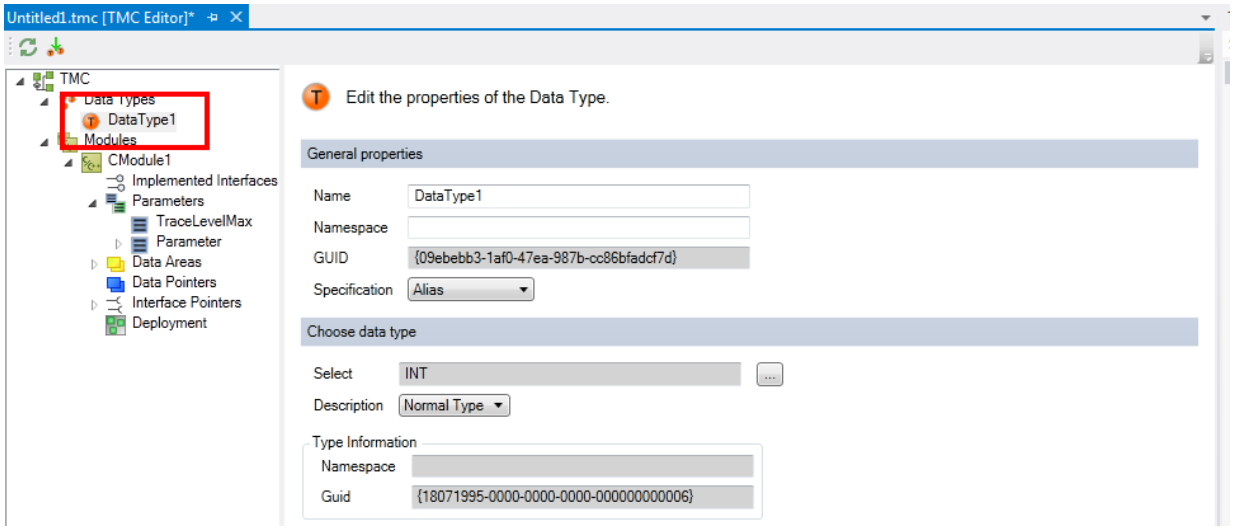
- 第 1 步：在 TMC 文件中 [▶ 91]创建新数据类型。
- 第 2 步：启动 TwinCAT TMC Code Generator [▶ 94]，以便根据 TMC 文件中的模块说明生成 C++ 代码。
- 使用 [▶ 107]数据类型。

第 1 步：生成新数据类型

1. 启动 TMC 编辑器后，选择**数据类型**节点。
2. 点击**添加新数据区**按钮（即 + 按钮），在数据类型和接口列表中添加一个新的数据类型。
   ⇨ 新**数据类型**便会列为新条目：

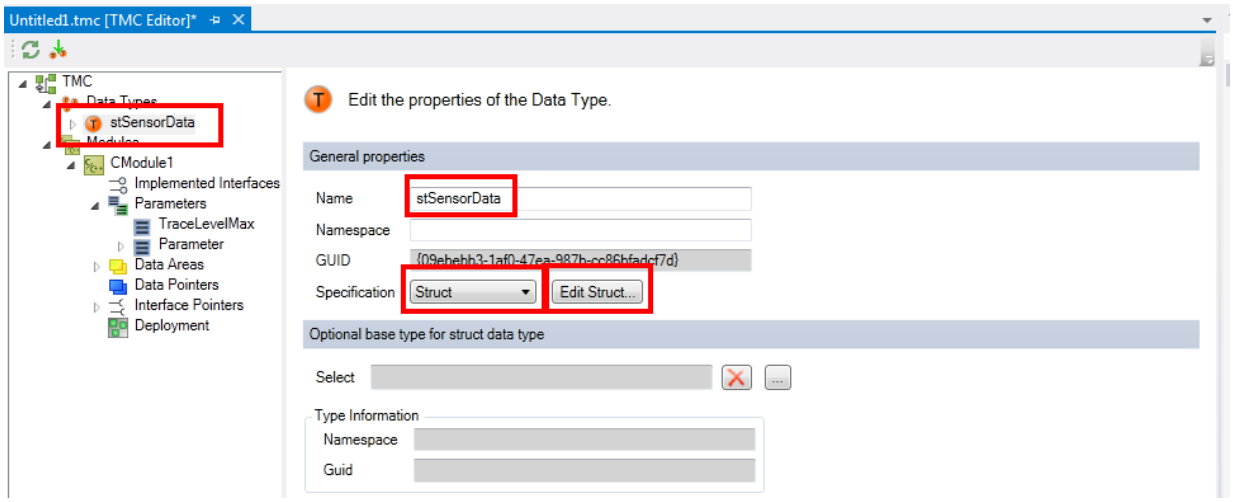


3. 选择生成的“数据类型 1”，以获取新数据类型的详细信息。

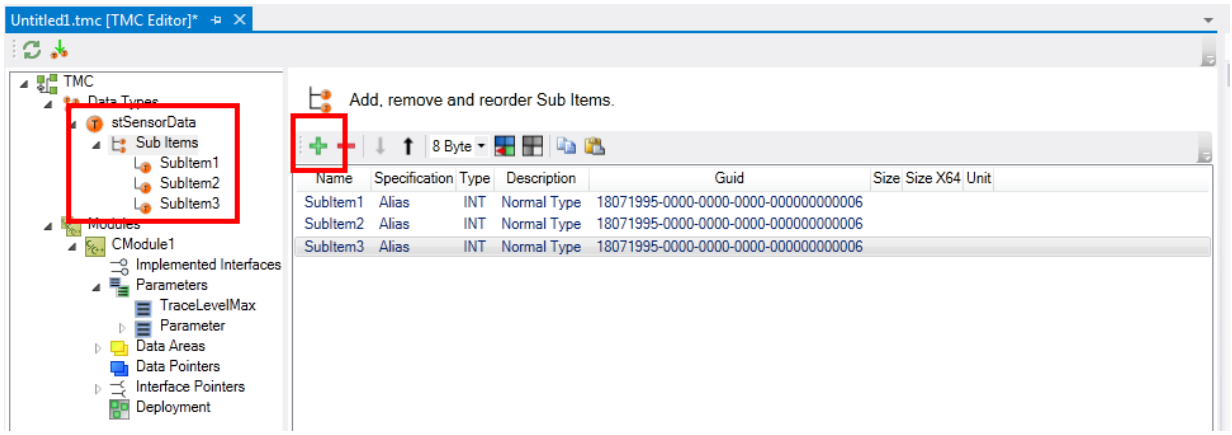


4. 指定 [▶ 101]数据类型。

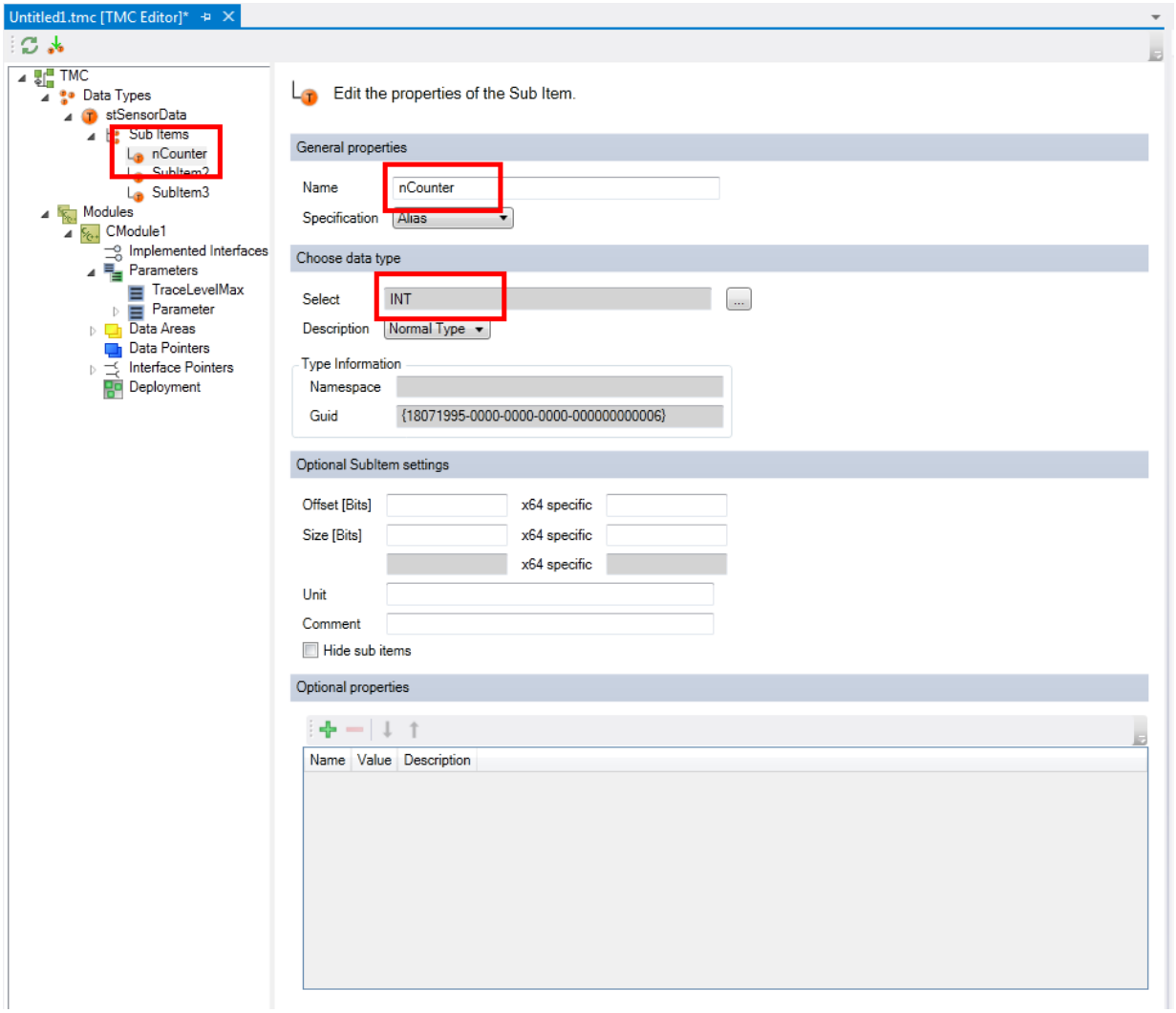
5. 重新命名数据类型。
   在本示例 **stSensorData** 中，选择规范**结构体**，然后点击**编辑结构体**。



6. 点击**添加新子项**按钮，在结构中插入新子项目。



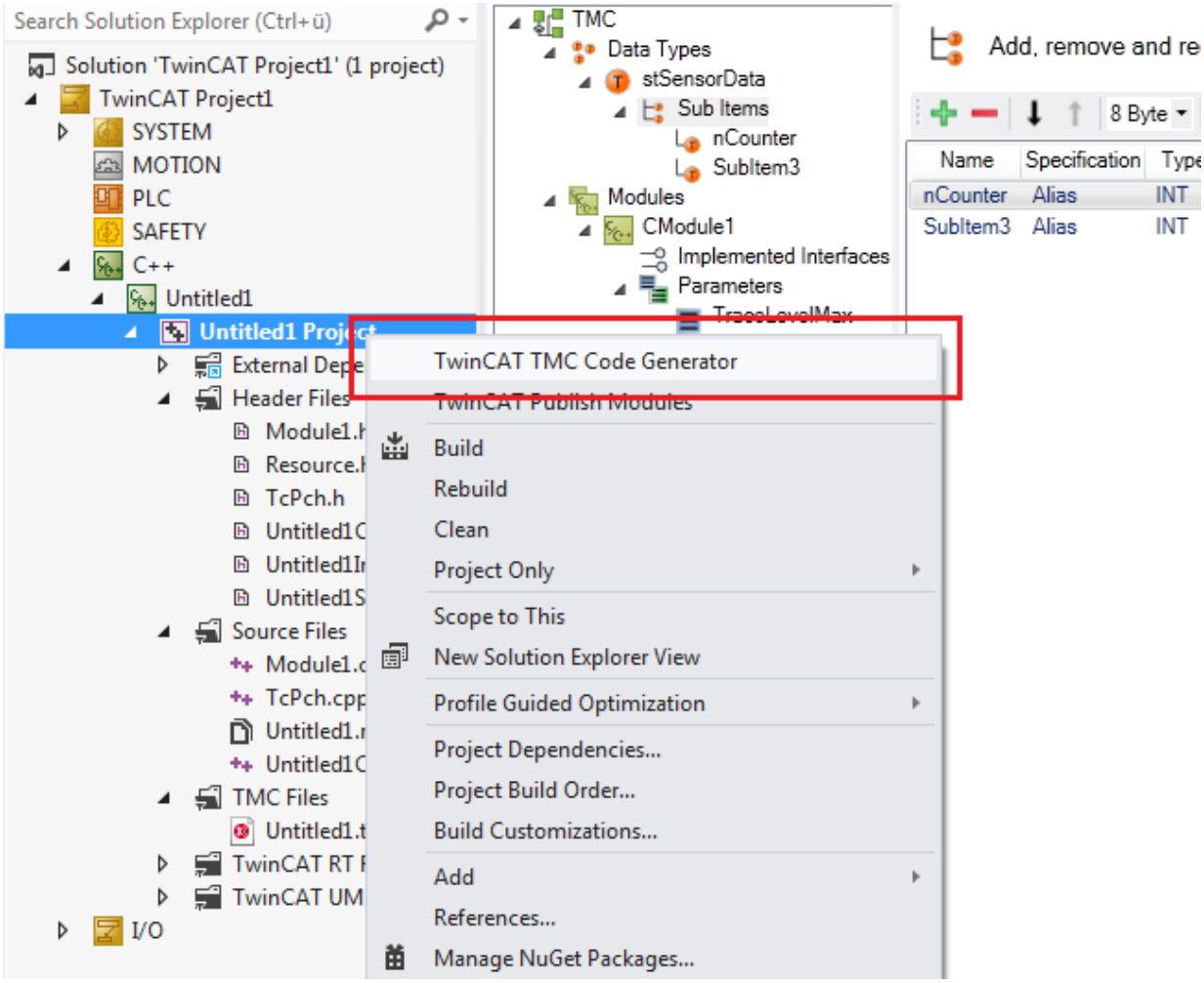
7. 双击子项目可编辑属性。为子项命名并选择合适的数据类型。



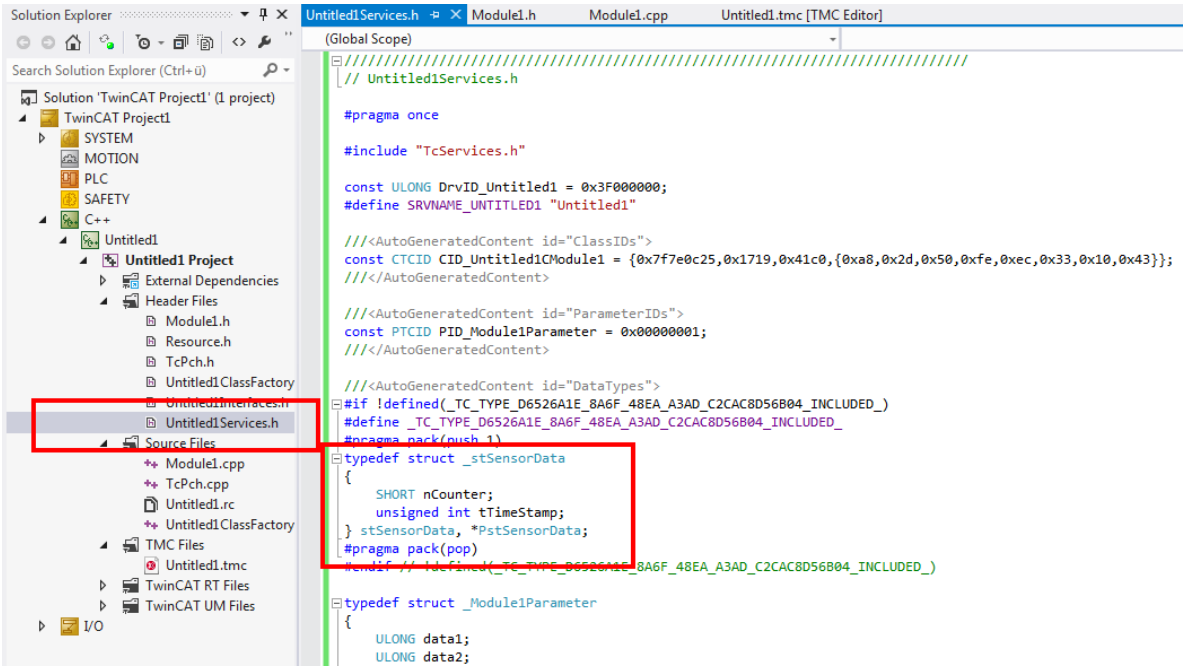
8. 为其他子项命名，并选择合适的数据类型。
9. 保存 TMC 文件中的更改项。

第 2 步：启动 TwinCAT TMC Code Generator，生成模块说明代码。

1. 右键单击项目文件，选择 **TwinCAT TMC Code Generator**，生成数据类型的源代码：



⇨ 您可以在模块报头文件“Untitled1Services.h”



中查看数据类型声明

⇨ 如果要添加其他数据类型或子元素，请再次运行 TwinCAT TMC Code Generator。

11.3.3.3 添加/修改/删除接口

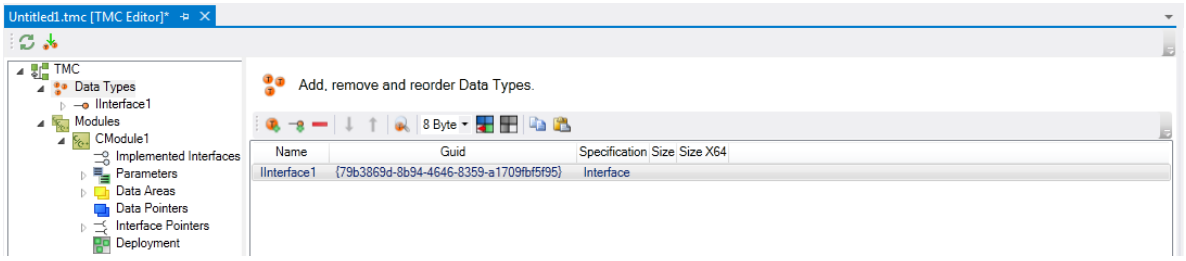
借助 TwinCAT 模块类 (TMC) 编辑器，可以添加、编辑和删除 TwinCAT 模块的接口。

本文介绍：

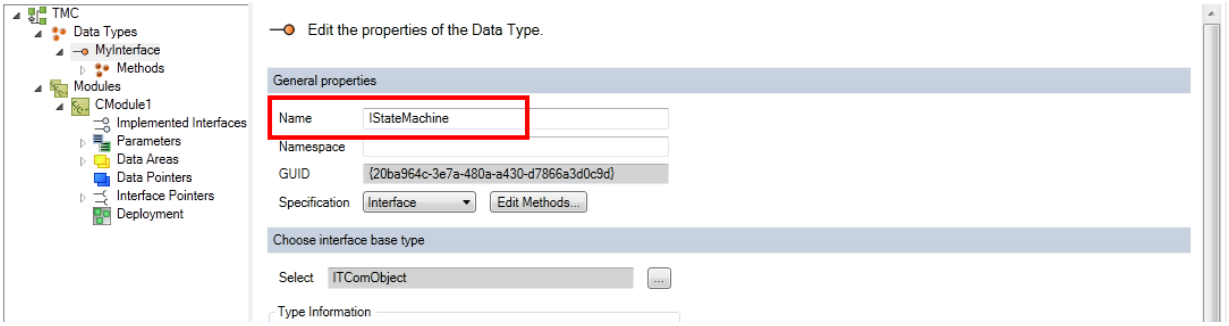
- 第 1 步：在 TMC 文件中 [▶ 95]创建一个新接口。
- 第 2 步： [▶ 95]在 TMC 文件中 [▶ 95]为接口添加方法。
- 第 3 步：将接口添加到 [▶ 97]模块的“已实现接口”中，以使用该接口。
- 第 4 步：启动 TwinCAT TMC [▶ 99] Code Generator，生成模块说明代码。
- 接口的可选更改 [▶ 99]。

第 1 步：生成新接口

1. 启动 TMC 编辑器后，选择**数据类型**节点。
2. 点击**添加新接口**，在接口列表中添加新接口。
   ⇨ 然后会列出一个新条目 **IInterface1**：



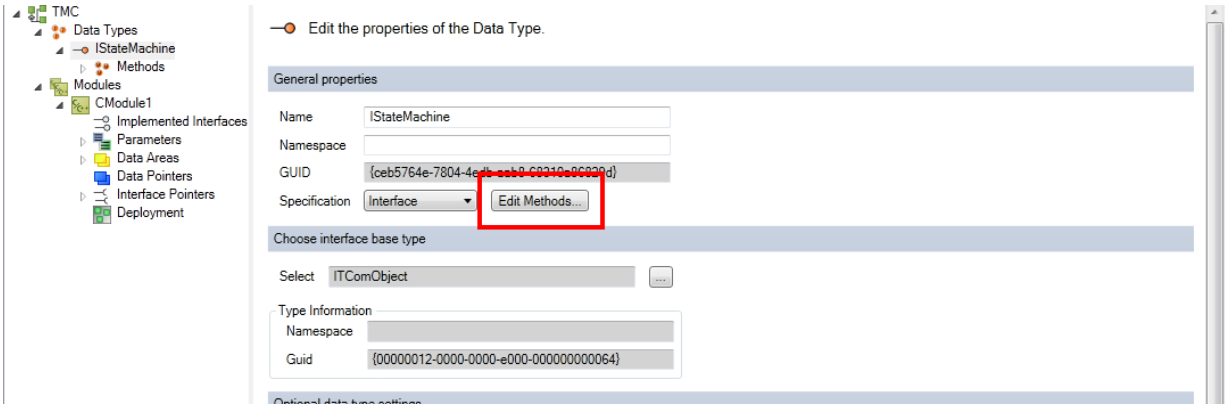
3. 要打开详细信息，可以选择树中的相应节点，或双击表格中的行。



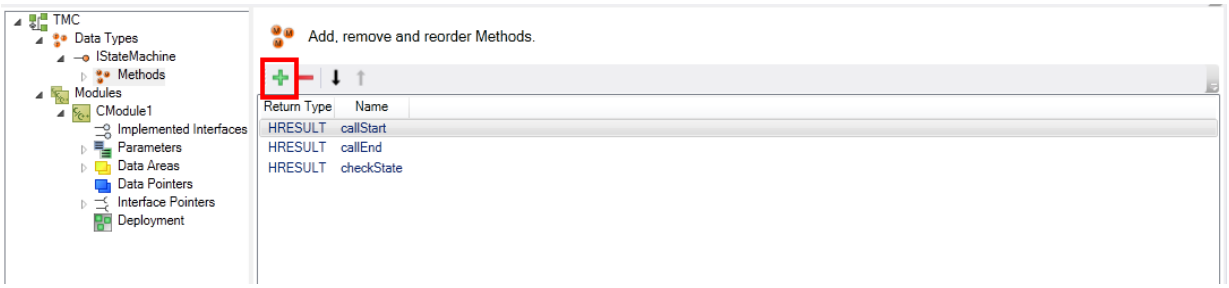
4. 输入一个有意义的名称，在本示例中为“IStateMachine”。

第 2 步：向接口添加方法

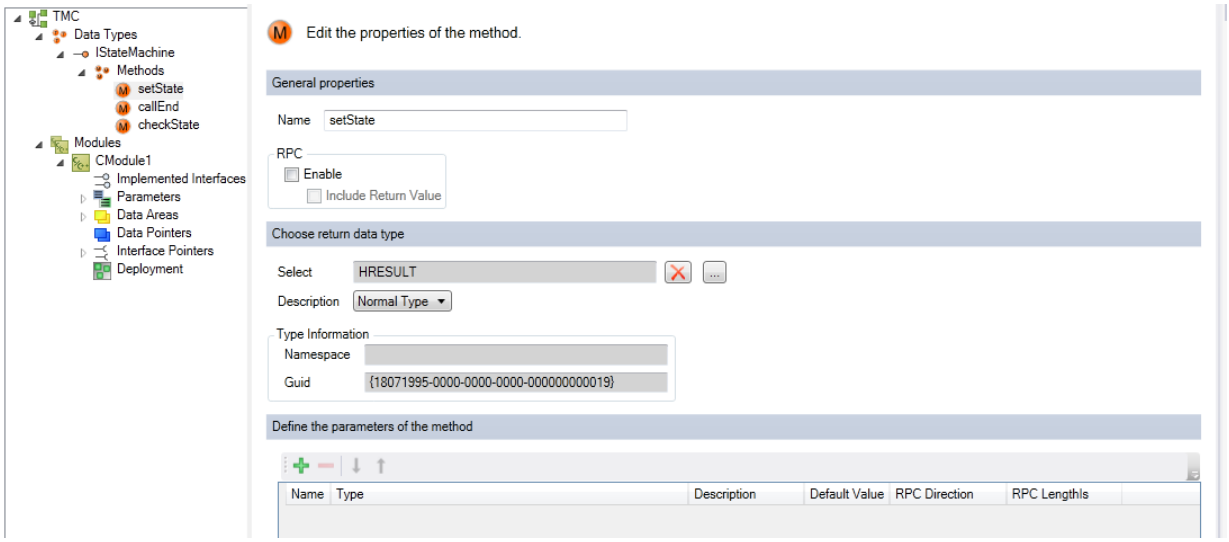
1. 点击**编辑方法.....**，获取该接口的方法列表：



2. 点击 + 按钮，生成新的默认方法“Method1”。



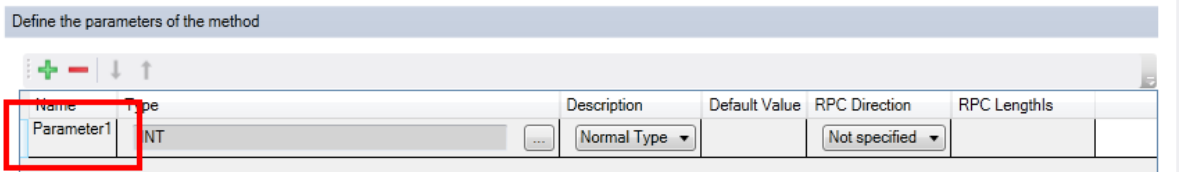
3. 双击方法或选择树中的节点，打开详细信息。
4. 为默认的“Method1”重命名为更有意义的名称。
5. 随后，您可以点击**添加新参数**来添加参数，或编辑“SetState”方法的参数。



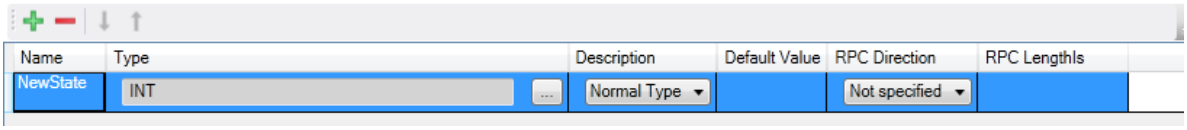
⇨ 新参数“Parameter1”默认生成为“常规类型” “INTEGER”。

6. 点击参数名称“Parameter1”编辑参数。

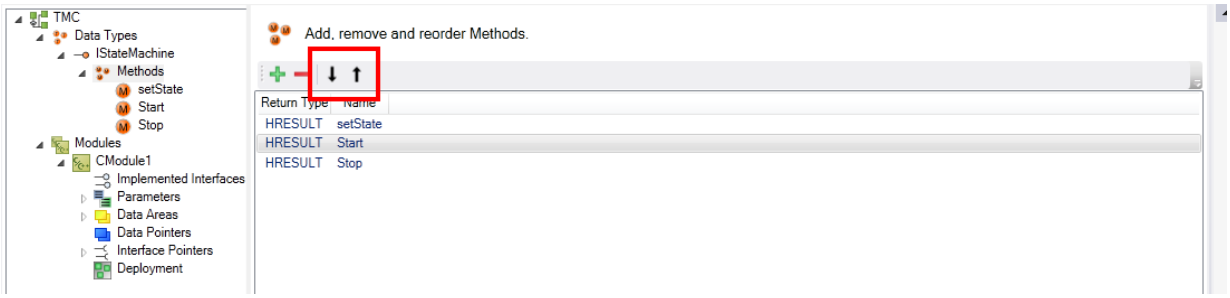
⇨ “常规类型”也可以更改为“指针”等，数据类型本身也可以选择。



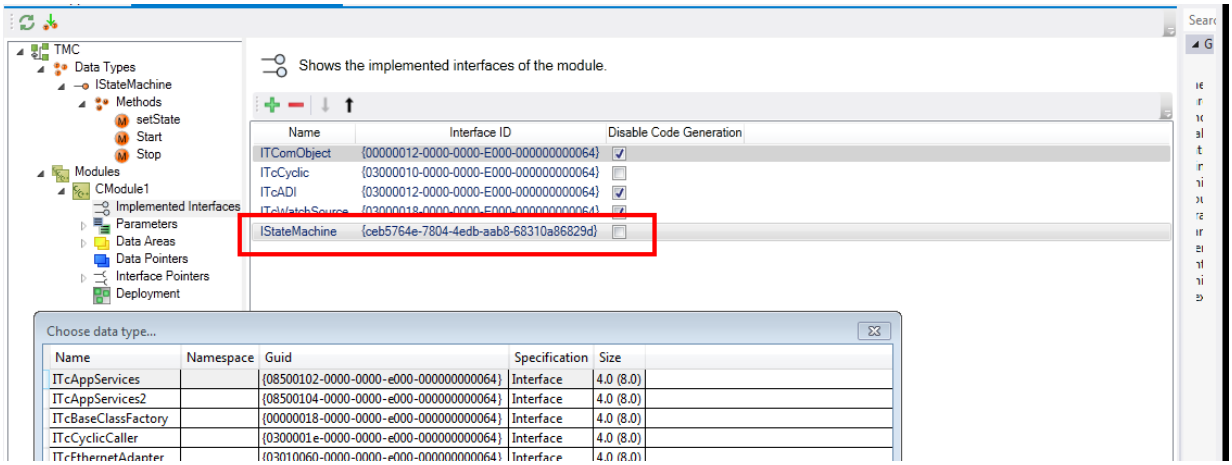
⇨ 在这种情况下，” NewState”即为新名称，其他设置不会改变。



7. 重复第 2 步“向接口添加方法”，所有方法都会列出，您可以通过**上移**/**下移**按钮重新排列方法的顺序。

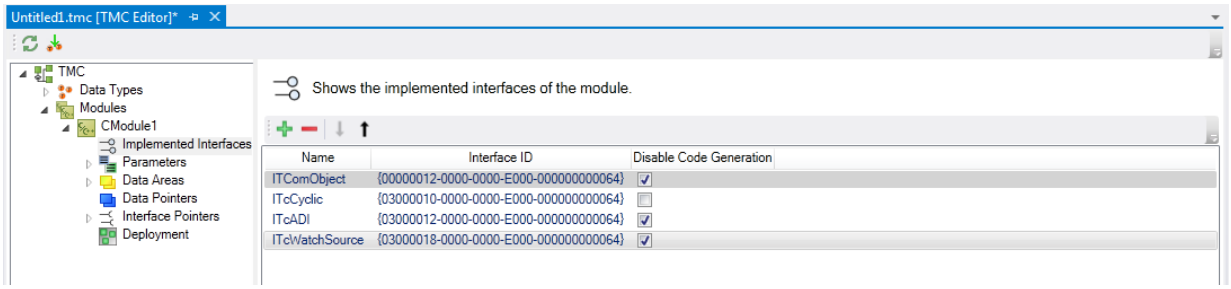


8. 您的模块即可实现该接口。

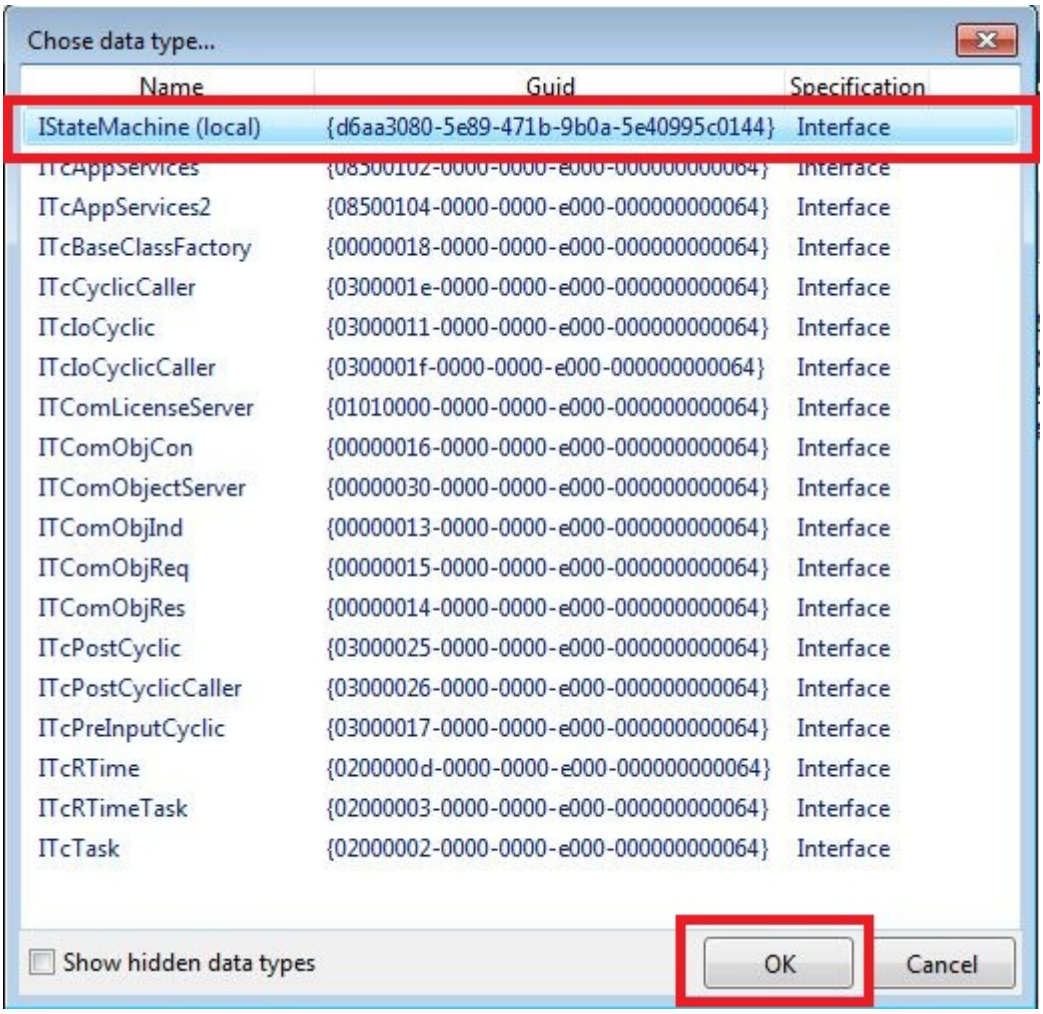


第 3 步：将新接口添加到“已实现接口”中

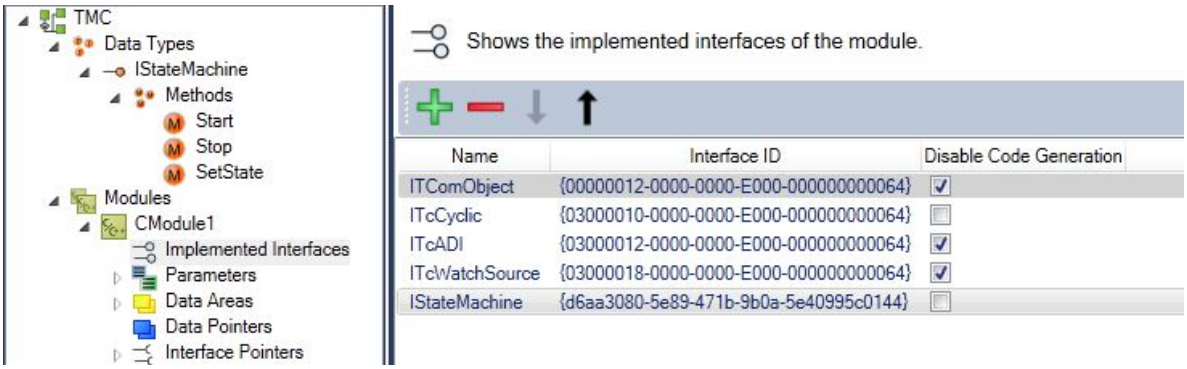
1. 选择要由新接口扩展的模块，本例中选择目标 **（Modules）模块 -> CModule1**。
2. 点击 **+** 按钮，使用**将新接口添加至模块**功能，在已实现接口的列表中添加一个新的接口。



3. 所有可用接口均已列出，选择新模板“IStateMachine”，然后点击**确定**结束。

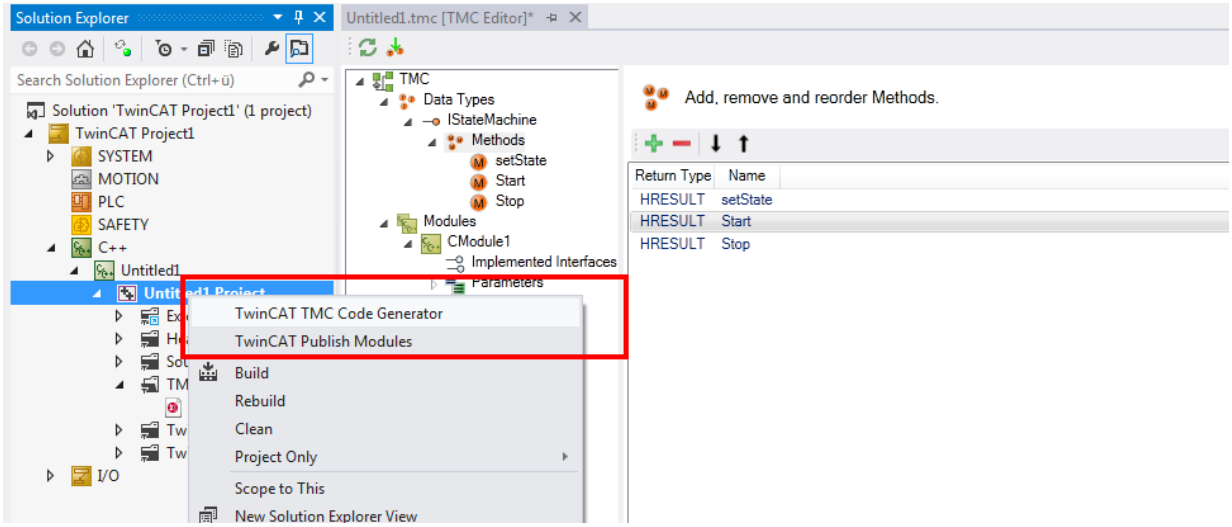


⇨ 新接口“IStateMachine”是模块说明的一部分。

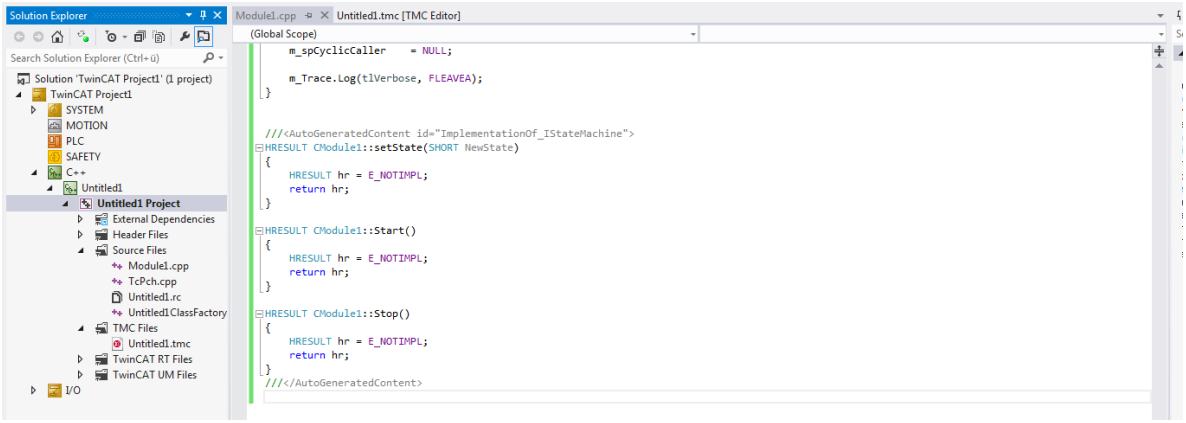


第 4 步：启动 TwinCAT TMC Code Generator，生成模块说明代码。

1. 为了在此模块说明的基础上生成 C/C++ 代码，请在 C/C++ 项目中右键单击，然后选择 **TwinCAT TMC Code Generator**。



⇨ 这样，模块“Module1”便会包含新的接口
CModule1：Start()
CModule1：Stop()
CModule1：SetState(SHORT NewState)。



⇨ 完成 - 现在可以在该区域插入用户自定义代码。

接口的可选更改

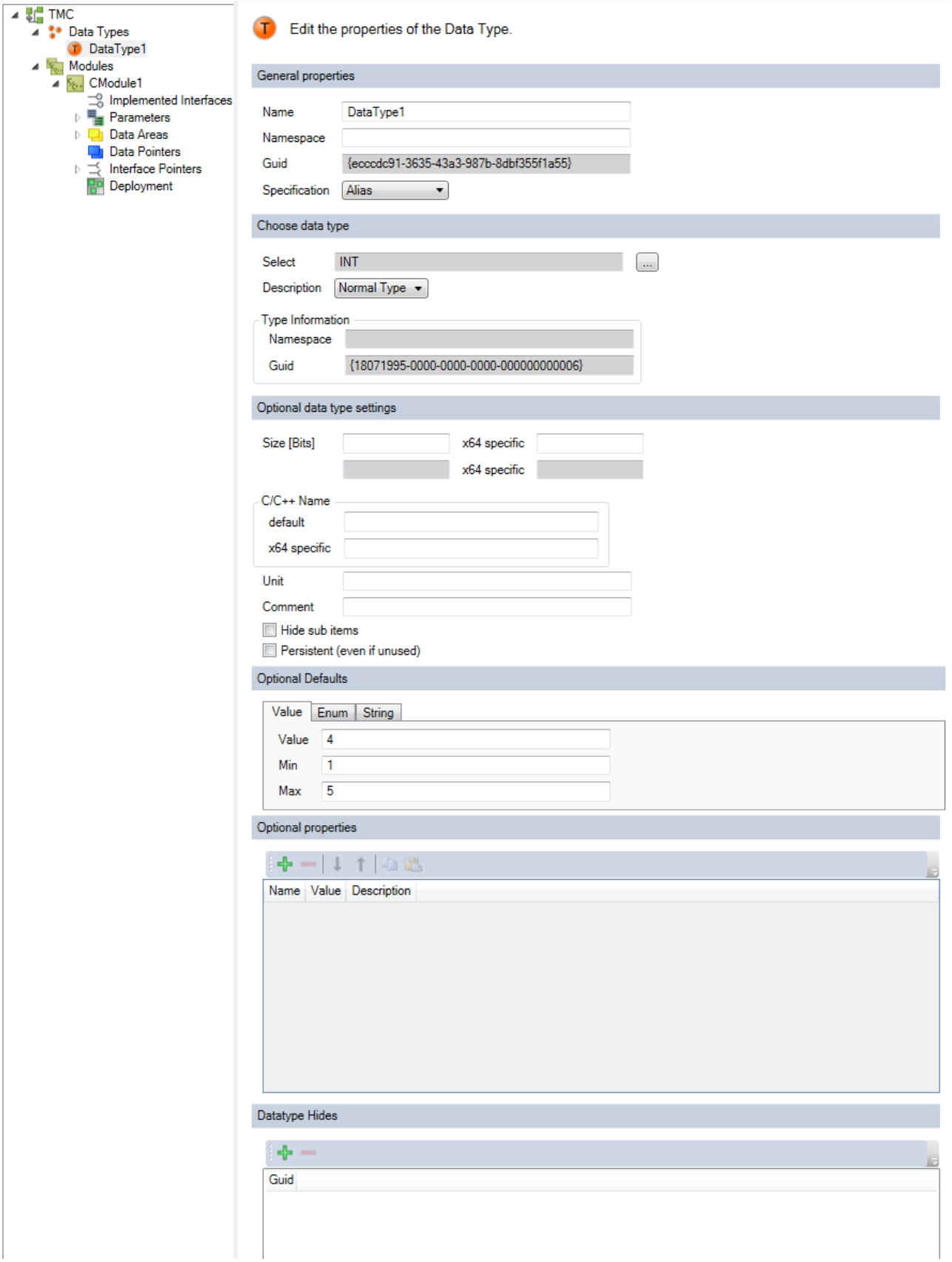
用户定义的代码永不删除

在接口发生变化的情况下（例如，方法的参数会稍后扩展），用户定义的代码永远不会删除。相反，如果 TMC Code Generator 无法映射这些方法，现有方法只会提供一个注释。

```
///<AutoGeneratedContent id="ImplementationOf_IStateMachine">
HRESULT CModule1::SetState(SHORT SetState, bool bRun)
{
    HRESULT hr = E_NOTIMPL;
    return hr;
}
///</AutoGeneratedContent>

///<AutoGeneratedContent id="Obsolete_ImplementationOf_IStateMachine">
//HRESULT CModule1::SetState(SHORT SetState)
//{
//  HRESULT hr = E_NOTIMPL;
//
//  //custom code
//  nState = SetState;
//
//  return hr;
//}
//
///</AutoGeneratedContent>
```

11.3.3.4 数据类型属性

编辑数据类型的属性



一般属性

名称：用户定义的数据类型名称。

注意

名称冲突

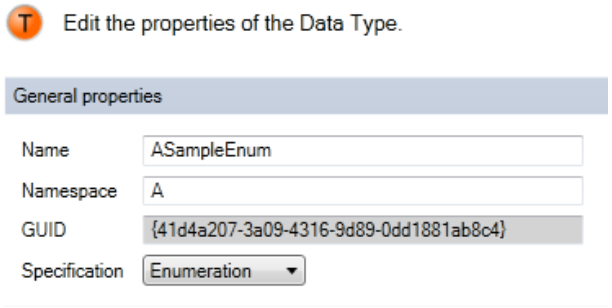
如果将驱动程序与 PLC 模块结合使用，则会发生名称冲突。

- 不要使用为 PLC 保留的关键词作为名称。

命名空间：用户定义的数据类型命名空间。

请注意，**不会**将其分配给 C 命名空间。用作数据类型的前缀。

示例：命名空间为“A”的枚举：



生成的代码如下：

```
///<AutoGeneratedContent id="DataTypes">
#if !defined(_TC_TYPE_41D4A207_3A09_4316_9D89_0DD1881AB8C4_INCLUDED_)
#define _TC_TYPE_41D4A207_3A09_4316_9D89_0DD1881AB8C4_INCLUDED_
enum A_ASampleEnum : SHORT {
One,
Two,
Three
};
#endif // !defined(_TC_TYPE_41D4A207_3A09_4316_9D89_0DD1881AB8C4_INCLUDED_)
```

您可以手动将命名空间名称作为前缀添加到枚举元素中：

```
#if !defined(_TC_TYPE_C26FED5F_AC13_4FD3_AC6F_B658CB5604E0_INCLUDED_)
#define _TC_TYPE_C26FED5F_AC13_4FD3_AC6F_B658CB5604E0_INCLUDED_
enum B_BSampleEnum : SHORT {
B_one,
B_two,
B_three
};
#endif // !defined(_TC_TYPE_C26FED5F_AC13_4FD3_AC6F_B658CB5604E0_INCLUDED_)
```

GUID：数据类型的唯一 ID。

规范：数据类型的规范。

- **别名**：生成标准数据类型（如 INT）的别名。
- 数组 [▶ 104]：创建用户定义的数组。
- 枚举 [▶ 104]：创建用户定义的枚举。
- 结构体 [▶ 105]：生成用户定义的结构。
- 接口 [▶ 106]：生成新的接口。

选择数据类型

选择：选择数据类型，可以是基本 TwinCAT 数据类型，也可以是用户定义的数据类型。
定义与 PLC 数据类型等效的数据类型（如 TIME、LTIME）。有关更多信息，请参见 PLC 的 。

说明：通过适当的选择将类型定义为指针、引用或值。

- 常规类型
- 指针

- 指针指向指针
- 指针指向指针指向指针
- 引用

类型信息

- **命名空间：**为所选数据类型定义。
- **GUID**：所选数据类型的唯一 ID。

可选数据类型设置

大小 [位]：以位（白色字段）和“Byte.Bit”表示法（灰色字段）为单位的大小。可为 x64 平台定义不同的大小。

C/C++ 名称：生成的 C++ 代码中所用名称。TMC Code Generator 不会生成该声明，因此可以为该数据类型提供用户定义的代码。除此之外，还可以为 x64 定义不同的名称。

单位：变量的单位。

注释：例如在实例配置器中可见的注释。

隐藏子项：如果数据类型有子元素，则系统管理器不允许访问子元素。例如，在数组较大的情况下应使用这种方法。

持久类型（即使未使用）：全局类型系统中的持久类型（请参见 System（系统） -> Type System（类型系统） -> Data Types（数据类型））。

可选默认值

可根据数据类型定义默认值。

可选属性

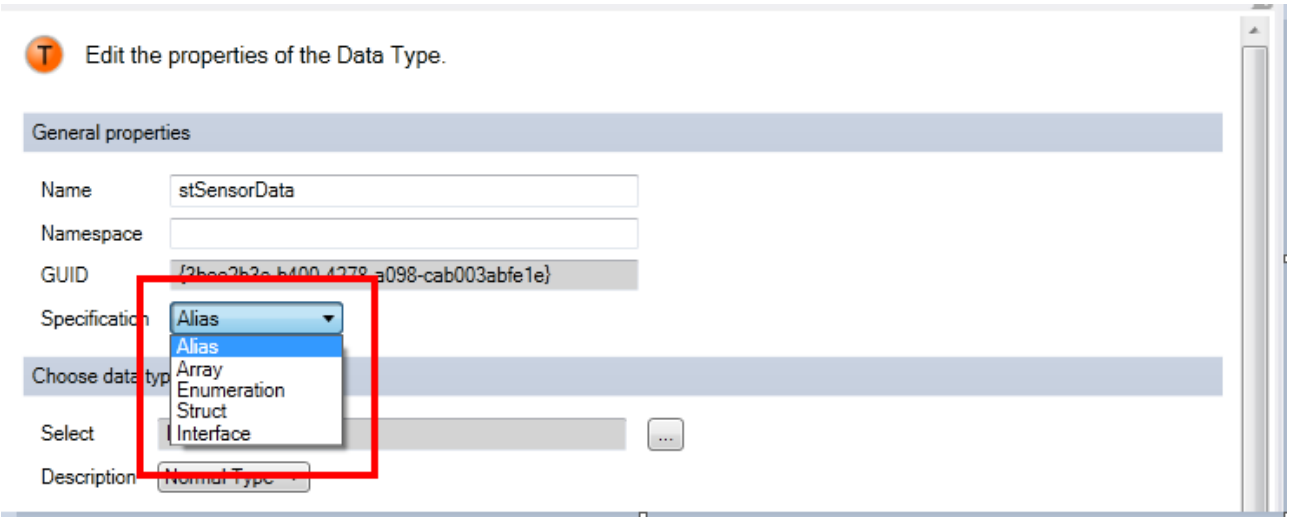
用于注释数据类型的名称、值和说明的表格。
这些信息可见于 TMC 和 TMI 文件。
TwinCAT 功能和客户程序均可使用这些属性。

数据类型隐藏

列出的 GUID 指的是该数据类型隐藏的数据类型。通常情况下，每次更改时都会自动插入该数据类型先前版本的 GUID。

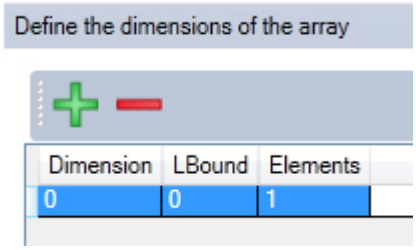
11.3.3.5 规范

本节介绍数据类型的规范。



11.3.3.5.1 数组

数组：创建用户定义的数组。



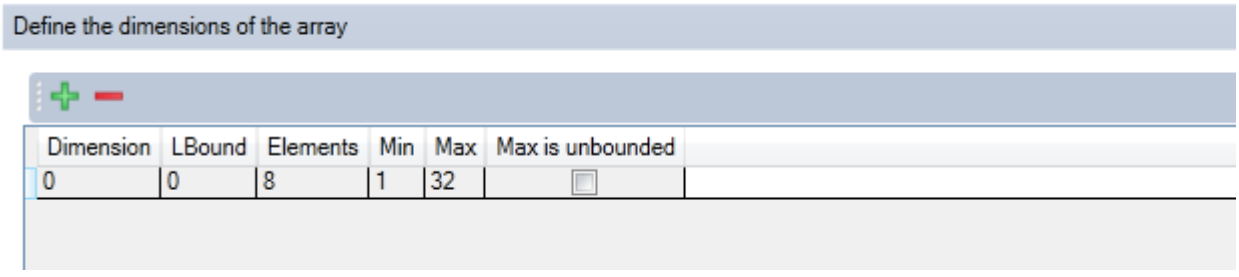
显示新的对话框，用于添加 (+) 或删除 (-) 数组元素。

维度：数组的维度。

LBound：数组的左限值（默认值 = 0）。

元素：元素数量。

参数和数据指针的动态数组



对于参数 [▶ 110]和数据指针 [▶ 124]，TwinCAT 3 支持具有动态大小的数组。

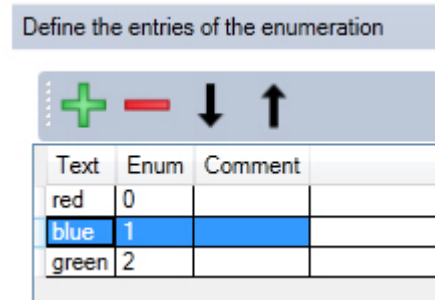
最小值：数组的最小大小。

最大值：数组的最大大小。

最大值无限制：表示数组大小没有上限。

11.3.3.5.2 枚举

枚举：创建用户定义的枚举。



将显示一个新的对话框，用于添加 (+) 或删除 (-) 元素。借助箭头调整顺序。

注意

枚举元素需要唯一名称

请注意，枚举元素必须有唯一名称，否则生成的 C++ 代码无效。

文本：枚举元素

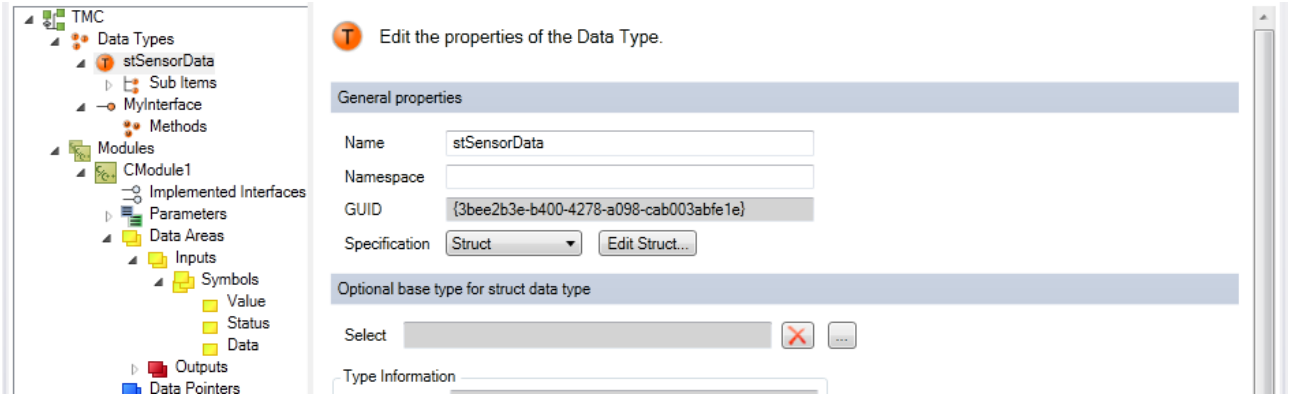
枚举：合适的整数值。

注释：可选注释。

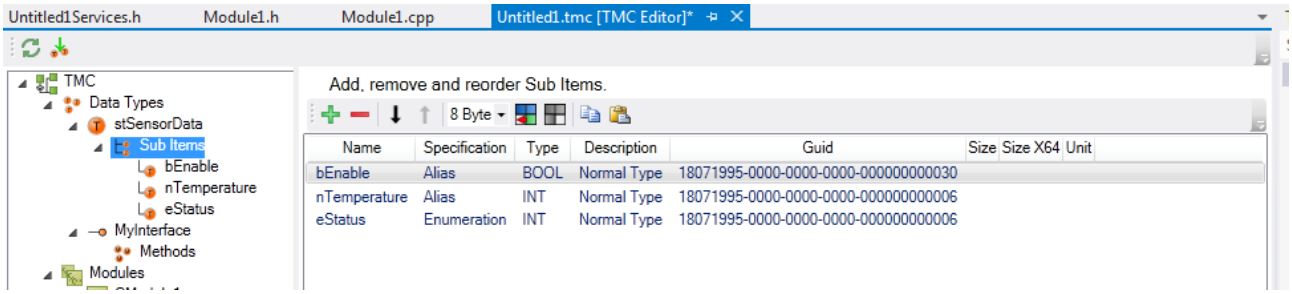
11.3.3.5.3 结构体

结构体：创建用户定义的结构。

选择**子项**节点或点击**编辑结构体**按钮，切换到该表：



将显示一个新的对话框，用于添加 (+) 或删除 (-) 元素。借助箭头调整顺序。



名称：元素名称。

规范：结构体可以包含别名、数组或枚举。

类型：变量的类型。

大小：子元素的大小和偏移。

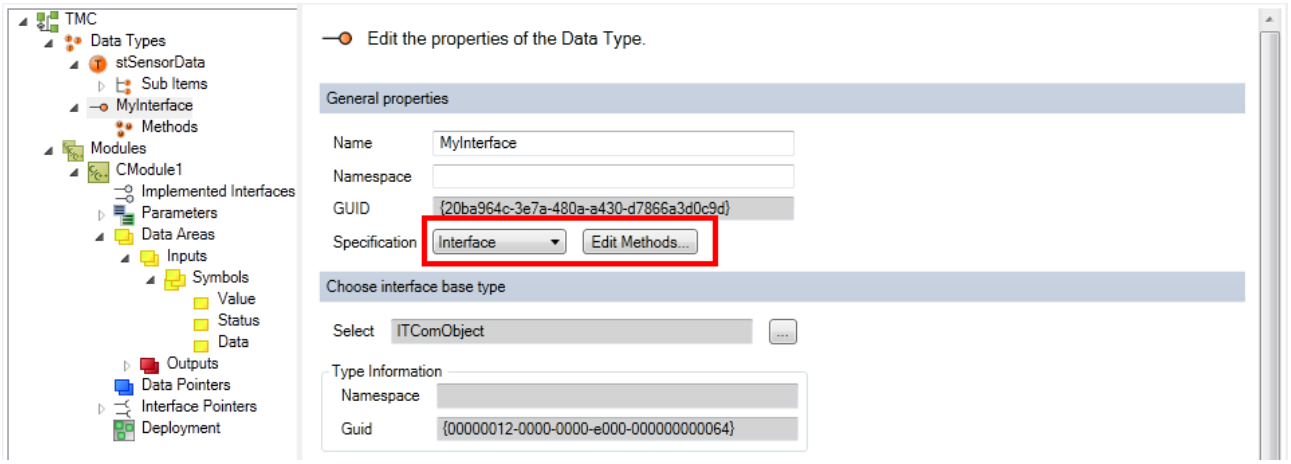
大小 X64：另外还将提供 x64 平台的其他大小。

单位：可选单位。

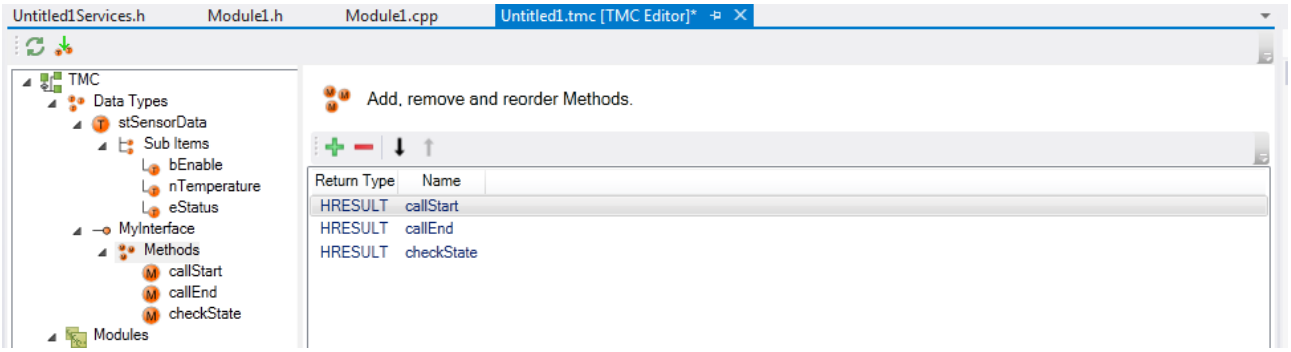
通过选择数据类型或双击表项，可显示子元素配置页面的详细信息。类似于 数据类型属性 [▶ 101]。

11.3.3.5.4 接口

接口：创建用户定义的接口。

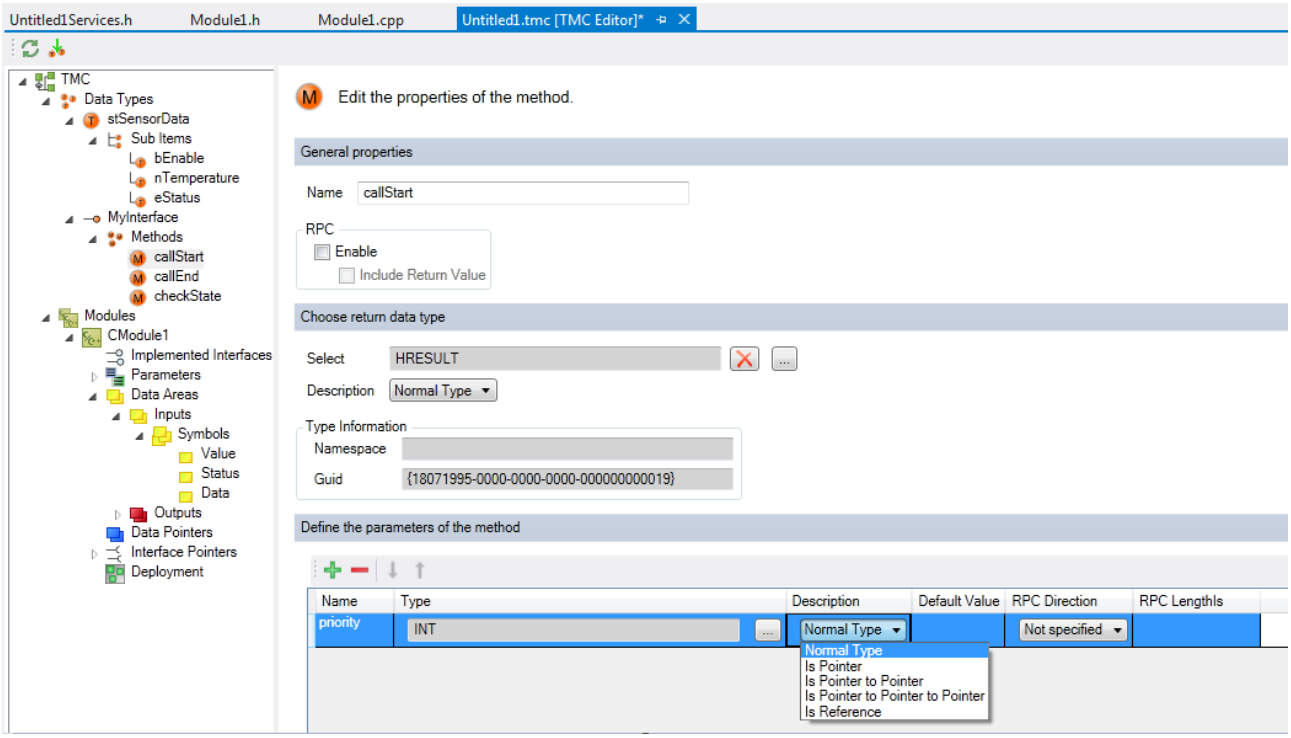


选择**方法**节点或点击**编辑方法**按钮，切换到该表：



方法参数

选择“方法”节点或双击条目，查看方法的详细信息。



名称：方法的名称。

启用 RPC：启用本方法外部的远程程序调用。

包含返回值：启用共享方法的返回值。

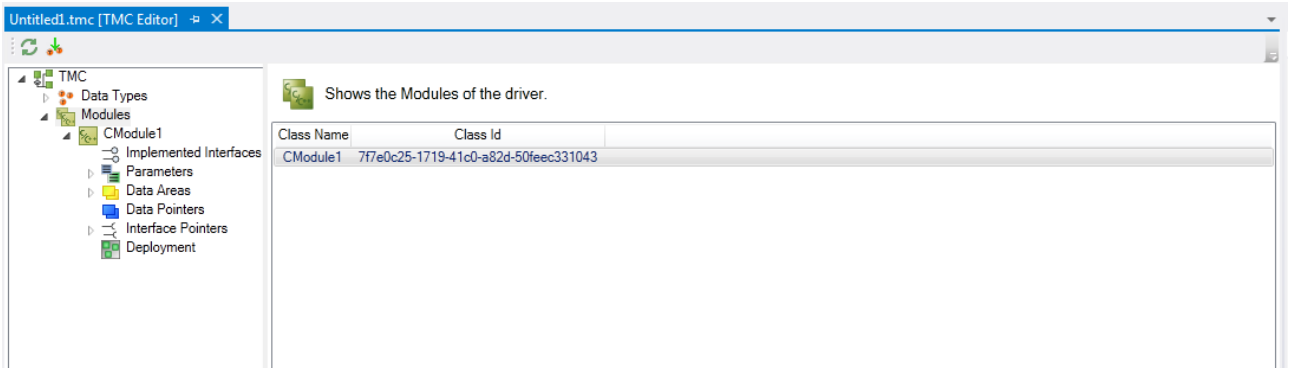
这些字段与 数据类型属性 [▶ 101] 的字段相对应。

定义方法参数

- **名称：**
- **类型：**从 数据类型属性 [▶ 101] 中得知。
- **说明：**从 数据类型属性 [▶ 101] 中得知。
- **默认值：**该参数的默认值；只允许使用数字。
- **RPC 方向：**与 PLC 功能块一样，每个参数可以是 IN、OUT 或 INOUT。此外，还可以将其定义为“无”，以便在远程程序调用 (RPC) 中忽略该参数。

11.3.4 模块

模块：显示驱动程序的模块。

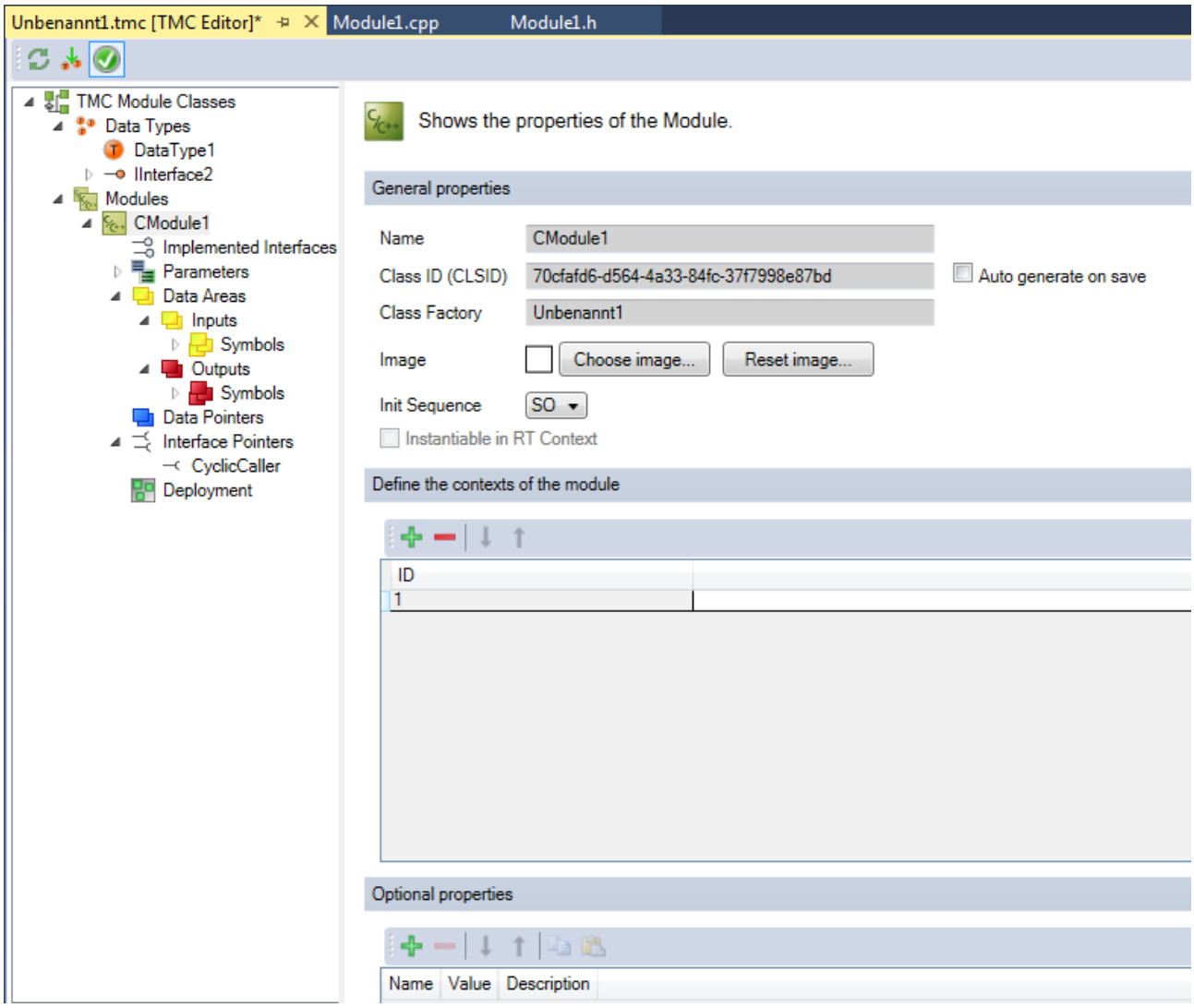


类名：模块名称。

类 ID：模块的唯一 ID。

模块属性：

点击树中的节点或表中的行，打开模块属性。



一般属性

名称：模块名称。

类 ID：唯一模块 ID。

保存时自动生成：使 TwinCAT 能够在保存时通过模块参数生成 ClassID。如果 ClassID 在导入二进制模块时发生变化，则必须调整相应的 ClassID。因此，TwinCAT 可以检测到接口变化。

选择图像：添加一个 16x16 像素的位图符号。

重置图像：将模块图像重置为默认值。

Init 序列：启动状态机。名称中带有“late”的选择项为内部选择项。（有关更多信息，请参见实例配置器的对象 [▶ 130]。）

可在实时环境中实例化：表示此模块是否可在实时环境中实例化；请参见 TwinCAT 模块类向导 [▶ 83]

定义模块的环境

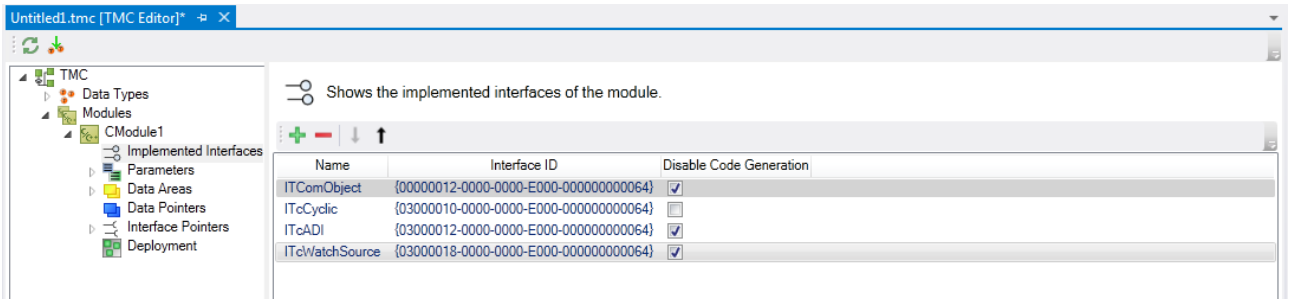
您可以为模块添加 (+) 或删除 (-) 环境。借助箭头调整顺序。
环境 ID 必须是 0 以外的整数。

可选属性

用于注释模块的名称、值和说明的表格。
这些信息可见于 TMC 和 TMI 文件。
TwinCAT 功能和客户程序均可使用这些属性。

11.3.4.1 已实现接口

已实现接口：查看和编辑模块的已实现接口。

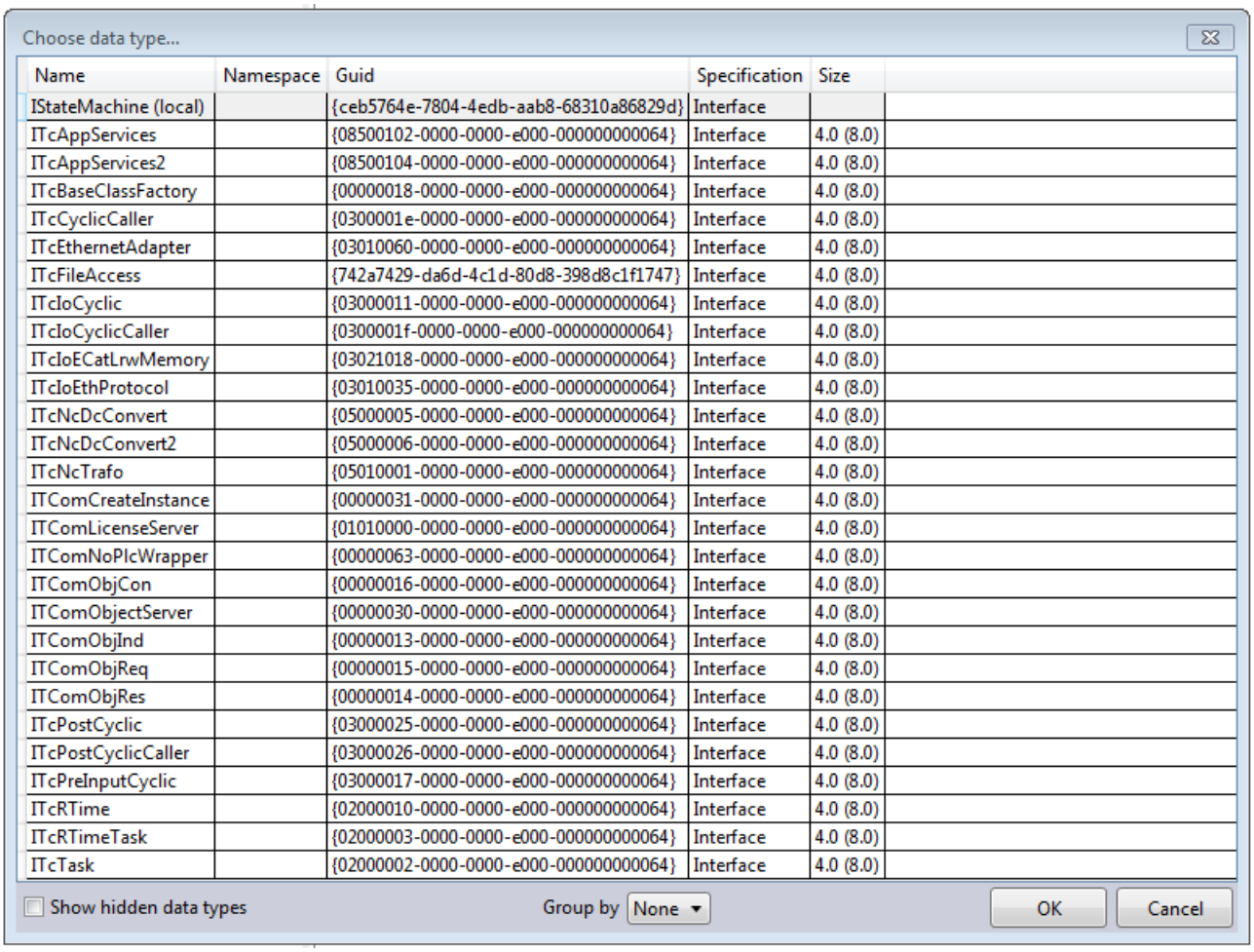


名称：接口名称。

接口 ID：接口的唯一 ID。

禁用代码生成：启用/禁用代码生成。

您可以为模块添加 (+) 或删除 (-) 环境。借助箭头调整顺序。



11.3.4.2 参数

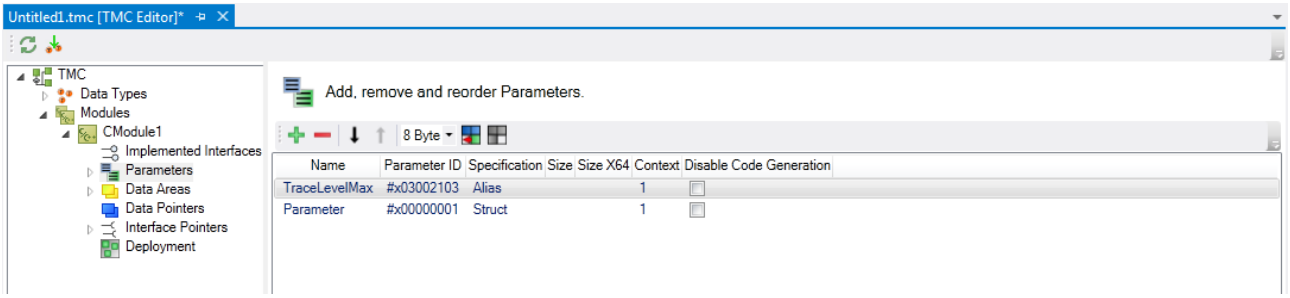
TcCOM 模块实例通过各种参数定义。

TwinCAT 支持三种类型的参数 ID (PTCID)，请参见参数 ID 的配置 [▶ 115]部分。

- “用户定义”（新参数的默认值）：会生成一个唯一的参数 ID，可在用户代码或实例配置中用于指定参数。
- “预定义.....”：TwinCAT 3 系统提供的特殊 PTCID（如 TcTraceLevel）。
- “基于环境.....”：自动为该参数分配配置环境 [▶ 131]的值。所选属性应用于 PTCPID。它会覆盖定义的标准参数和实例配置参数（参数 (Init)）。

下文将详细介绍这些参数及其配置。

参数：显示模块的执行参数。



符号	功能
	添加新的参数
	删除所选类型
	将所选元素下移一个位置
	将所选元素上移一个位置
	选择字节对齐
	对齐所选数据类型
	重置所选数据类型的数据格式

名称：接口名称。

参数 ID：参数的唯一 ID。

规范：参数的数据类型。

大小：参数的大小。x64 可以使用其他大小。

环境：参数的环境 ID。

禁用代码生成：启用/禁用代码生成。

11.3.4.2.1 添加/修改/删除参数

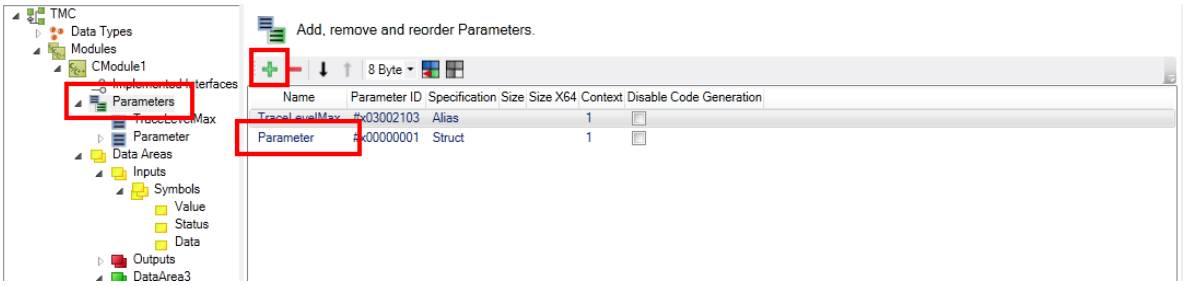
借助 TwinCAT 模块类 (TMC) 编辑器，可以添加、编辑和删除 TwinCAT 类的属性和功能。

本文介绍：

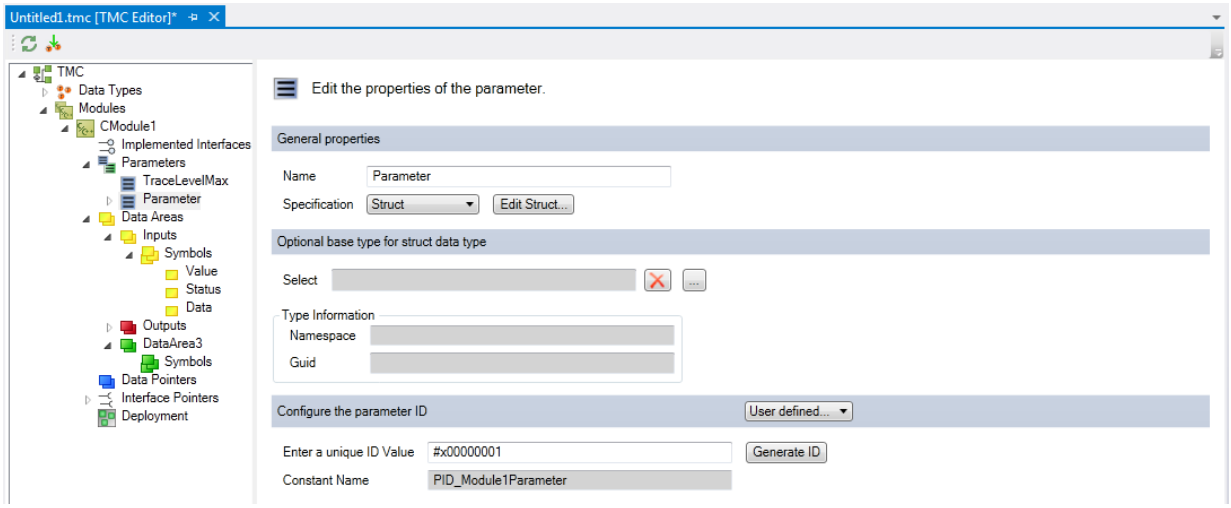
- 第 1 步：在 TMC 文件中 [▶ 111]创建一个新的参数。
- 第 2 步：启动 TwinCAT TMC Code Generator [▶ 112]，为 TMC 文件中的模块说明生成代码。
- 第 3 步：观察状态机的转换 [▶ 113]

第 1 步：创建新参数

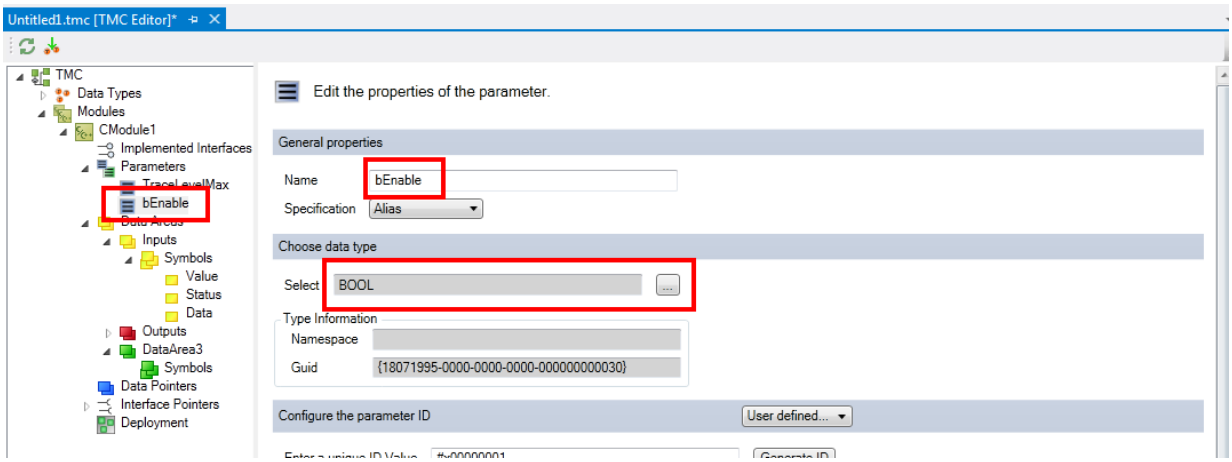
1. 启动 TMC 编辑器后，选择目标**参数**。
2. 点击**添加新参数**按钮（即 + 按钮），在参数列表中添加一个新的参数。
   ⇨ 新的“参数”便会列为新条目：



3. 选择左侧树中的**参数**，或双击红色标记“Parameter3”，或选择树中的节点，以获取新参数的详细信息。



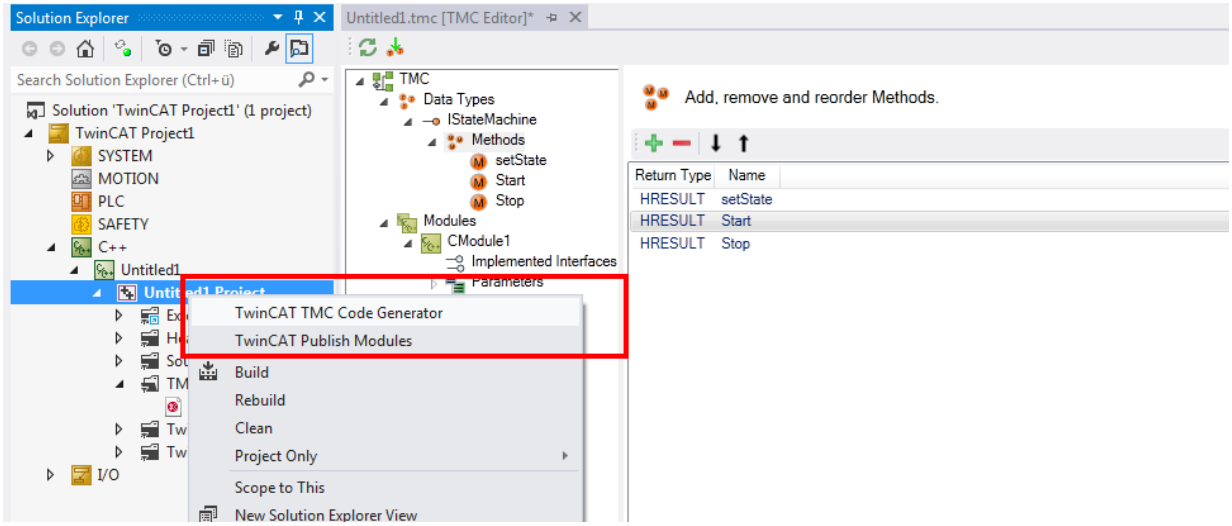
4. 配置参数以及 数据类型 [▶ 89]。
5. 为其重命名为更有意义的名称，在本示例中为“bEnable”，然后选择数据类型“BOOL”。



6. 保存 TMC 文件中的更改项。

第 2 步：启动 TwinCAT TMC Code Generator，生成模块说明代码。

1. 右键单击项目文件，选择 **TwinCAT TMC Code Generator**，接收源代码中的参数：



⇨ 您可以在模块的报头文件“Module1.h”中查看参数声明。

```
///<AutoGeneratedContent id="Members">
    TcTraceLevel m_TraceLevelMax;
    bool m_bEnable;
    Module1Inputs m_Inputs;
    Module1Outputs m_Outputs;
    Module1DataArea3 m_DataArea3;
    ITcCyclicCallerInfoPtr m_spCyclicCaller;
///</AutoGeneratedContent>
```

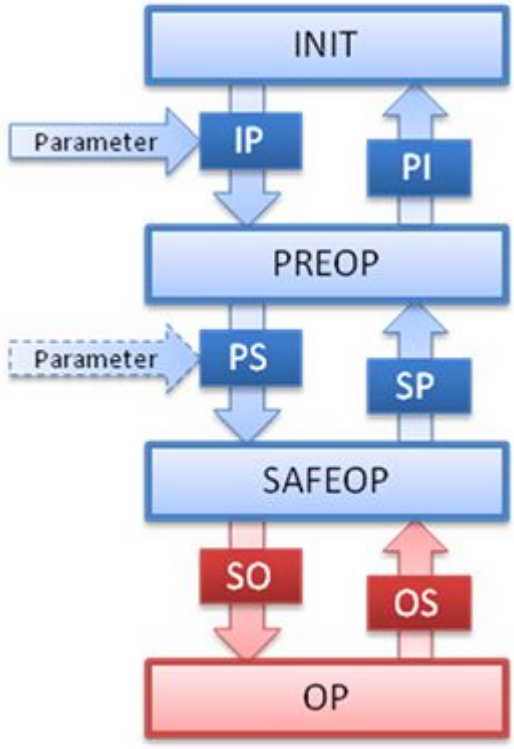
⇨ 新参数的实现可以在模块类“module1.cpp”的获取和设置方法中找到。

```
IMPLEMENT_ITCOMOBJECT(CModule1)
IMPLEMENT_ITCOMOBJECT_SETSTATE_LOCKOP2(CModule1)
IMPLEMENT_ITCADI(CModule1)
IMPLEMENT_ITCWATCHSOURCE(CModule1)


////////////////////////////////////////////////////////////////////////////////
// Set parameters of CModule1
BEGIN_SETOBJPARA_MAP(CModule1)
    SETOBJPARA_DATAAREA_MAP()
///<AutoGeneratedContent id="SetObjectParameterMap">
    SETOBJPARA_VALUE(PID_TcTraceLevel, m_TraceLevelMax)
    SETOBJPARA_VALUE(PID_Module1bEnable, m_bEnable)
    SETOBJPARA_ITFPTR(PID_Ctx_TaskOid, m_spCyclicCaller)
///</AutoGeneratedContent>
END_SETOBJPARA_MAP()

////////////////////////////////////////////////////////////////////////////////
// Get parameters of CModule1
BEGIN_GETOBJPARA_MAP(CModule1)
    GETOBJPARA_DATAAREA_MAP()
///<AutoGeneratedContent id="GetObjectParameterMap">
    GETOBJPARA_VALUE(PID_TcTraceLevel, m_TraceLevelMax)
    GETOBJPARA_VALUE(PID_Module1bEnable, m_bEnable)
    GETOBJPARA_ITFPTR(PID_Ctx_TaskOid, m_spCyclicCaller)
///</AutoGeneratedContent>
END_GETOBJPARA_MAP()
```

如需添加其他参数，请再次使用 TwinCAT TMC Code Generator。

第 3 步：状态机的转换

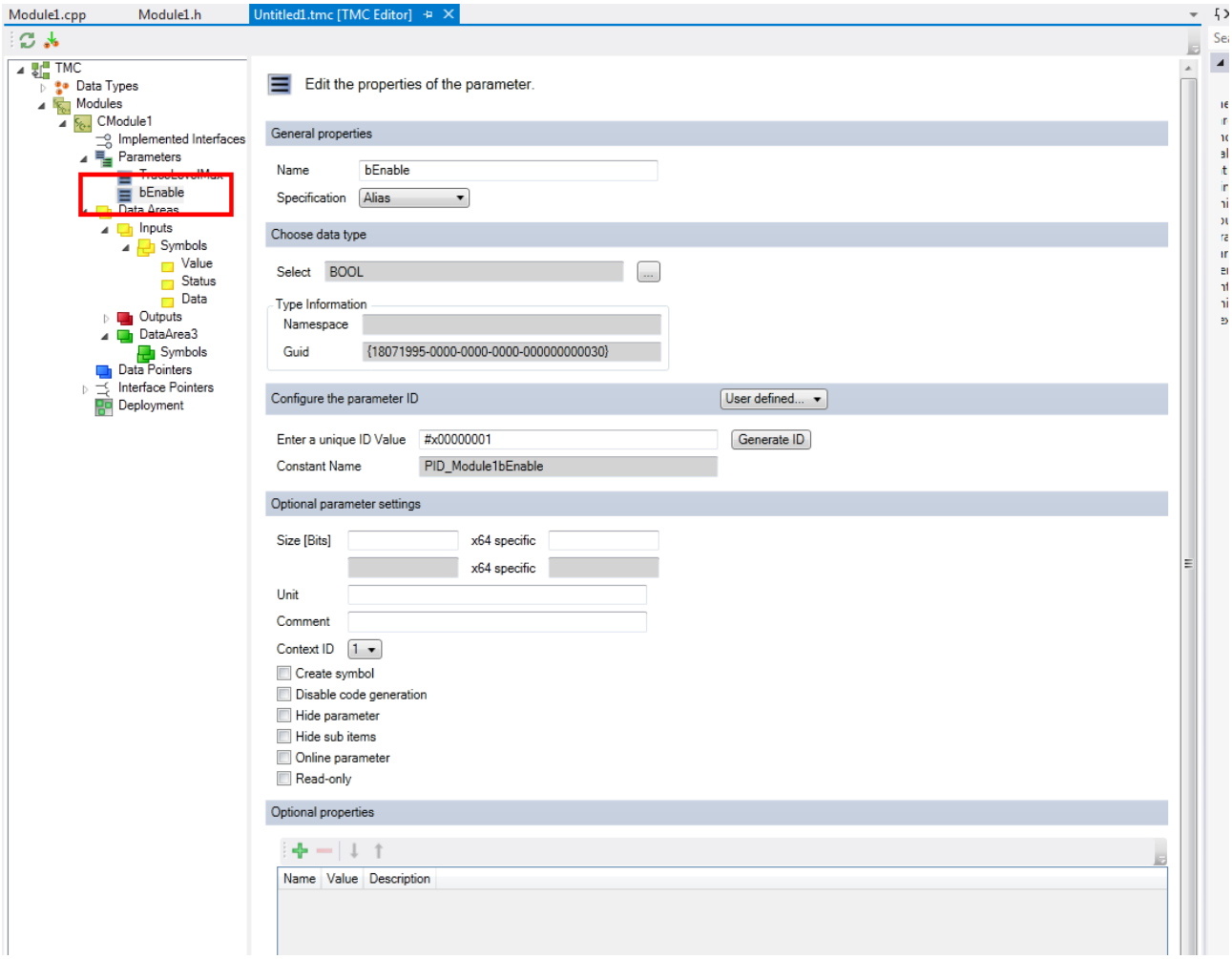
注意状态机 [▶ 43]的不同状态转换：



这些参数是在从 Init 到 Preop 以及（如适用）从 Preop 到 Safeop 的转换期间设置的。

11.3.4.2.2 参数属性

参数属性：编辑参数属性。



一般属性

名称：接口名称。

规范：参数的数据类型，请参见规范。

选择数据类型

选择：选择数据类型。

类型信息

- **命名空间**：用户定义的数据类型命名空间。
- **GUID**：数据类型的唯一 ID。

输入唯一 ID 值：输入唯一 ID 值，请参见参数 [▶ 110]。

常量名称：参数 ID 的源代码名称。

可选参数设置

大小 [位]：以位（白色字段）和“Byte.Bit”表示法（灰色字段）为单位计算的大小。已为 x64 提供特殊大小配置。

单位：变量的单位。

注释：例如在实例配置器中可见的注释。

环境 ID：ADS 访问参数时使用的环境。

创建符号：用于创建 ADS 图标的默认设置。

禁用代码生成：启用/禁用代码生成。

隐藏参数：在系统管理器视图中的显示/隐藏参数之间切换。

隐藏子项：如果数据类型有子元素，则系统管理器不允许访问子元素。例如，在数组较大的情况下应使用这种方法。

在线参数：定义为在线参数。

只读：为系统管理器切换为只读访问模式。

可选属性

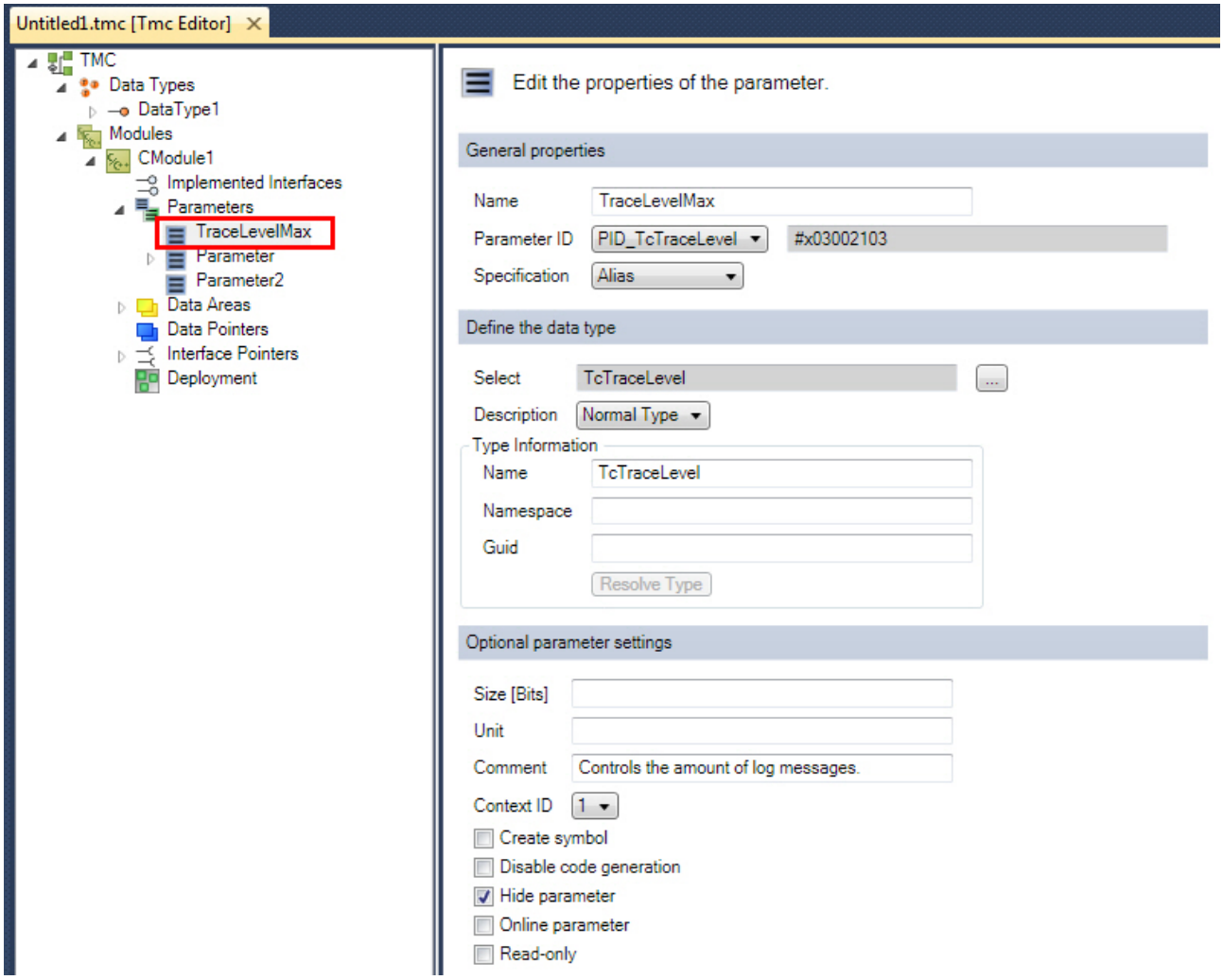
用于注释参数的名称、值和说明的表格。
这些信息可见于 TMC 和 TMI 文件。
TwinCAT 功能和客户程序均可使用这些属性。

11.3.4.2.3 TraceLevelMax

TraceLevelMax：定义跟踪级别的参数。
此为大多数 TwinCAT 模块模板提供的预定义参数（空 TwinCAT 模块模板除外）。

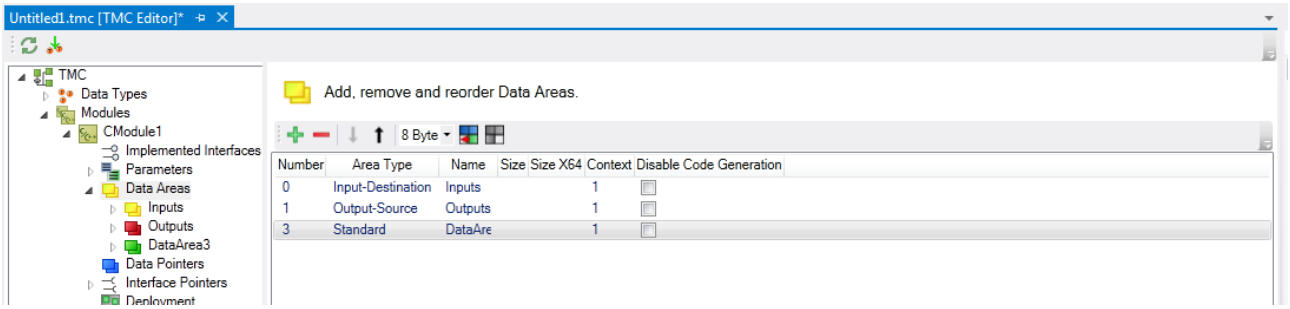
不得更改该参数的设置。

请参见 工程的模块消息（日志记录/追溯） [▶ 208]



11.3.4.3 数据区

数据区：用于编辑模块数据区的对话框。



符号	功能
	添加新的数据区
	删除所选数据区
	将所选元素下移一个位置
	将所选元素上移一个位置
	选择字节对齐
	对齐所选数据类型
	重置所选区域的数据格式

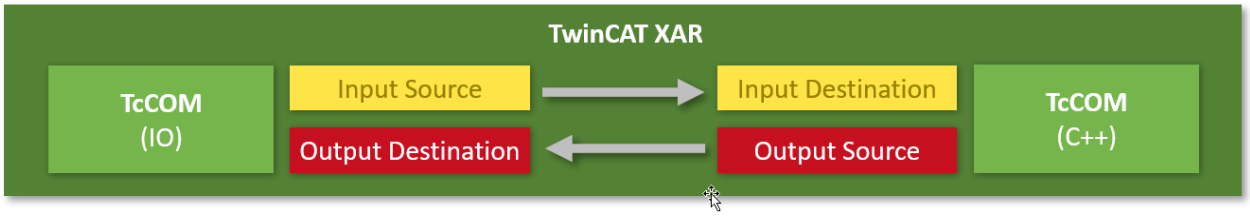
注意

设置对齐方式时的递归

设置数据区的对齐方式时，所有元素（符号及其子元素）均以此为基础。用户定义的对齐方式会被覆盖。

编号：数据区编号。

类型：定义数据区的用途和位置。
用于映射（到其他模块以及 I/O）的数据区分为输入和输出以及源和目标。图表会说明以下内容：



名称：数据区名称。

大小：参数大小；x64 可以使用其他大小。

环境：显示环境 ID。

禁用代码生成：启用/禁用代码生成。

11.3.4.3.1 添加/修改/删除数据区和变量

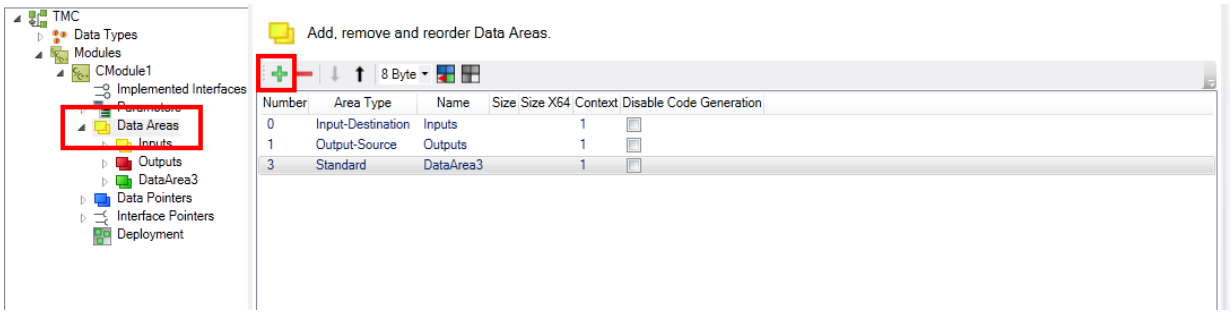
借助 TwinCAT 模块类 (TMC) 编辑器，可以添加、编辑和删除 TwinCAT 类的属性和功能。

本文介绍：

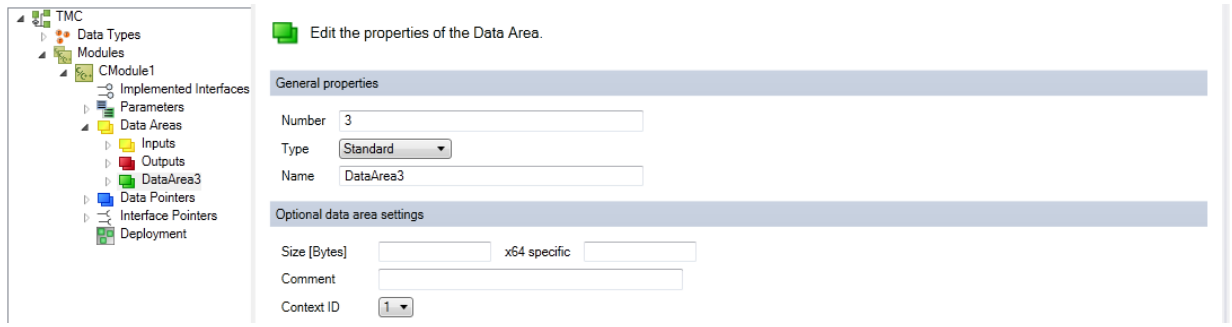
- 在 TMC 文件中创建新的数据区。
- 在数据区创建 [▶ 123]新的变量。
- 例如，编辑 TMC 文件中 [▶ 123]已有变量的名称或数据类型。
- 从 TMC 文件中删除现有变量 [▶ 124]。

创建新数据区

1. 启动 TMC 编辑器后，选择模块的**数据区**节点。
2. 点击 + 按钮，创建一个新数据区。



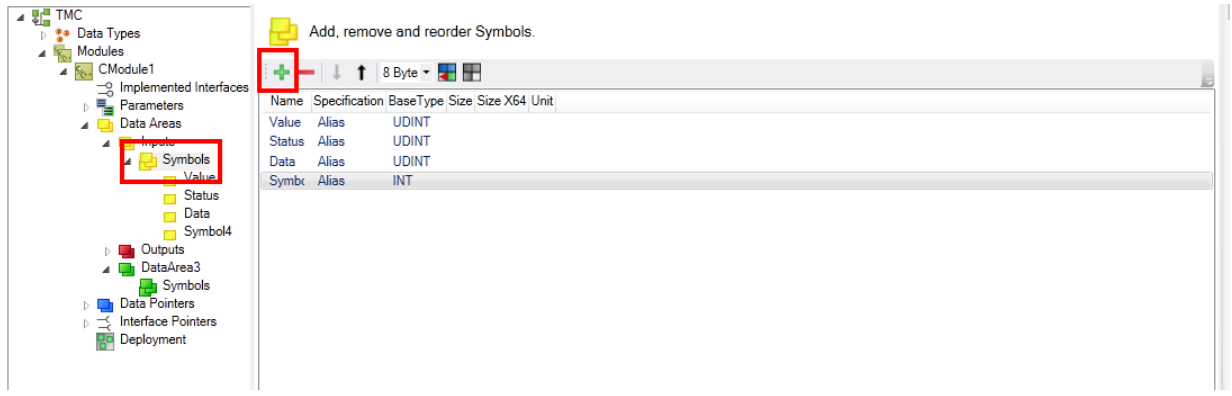
3. 要获取数据区的属性，请双击表格或节点。



4. 重新命名数据区。

创建新变量

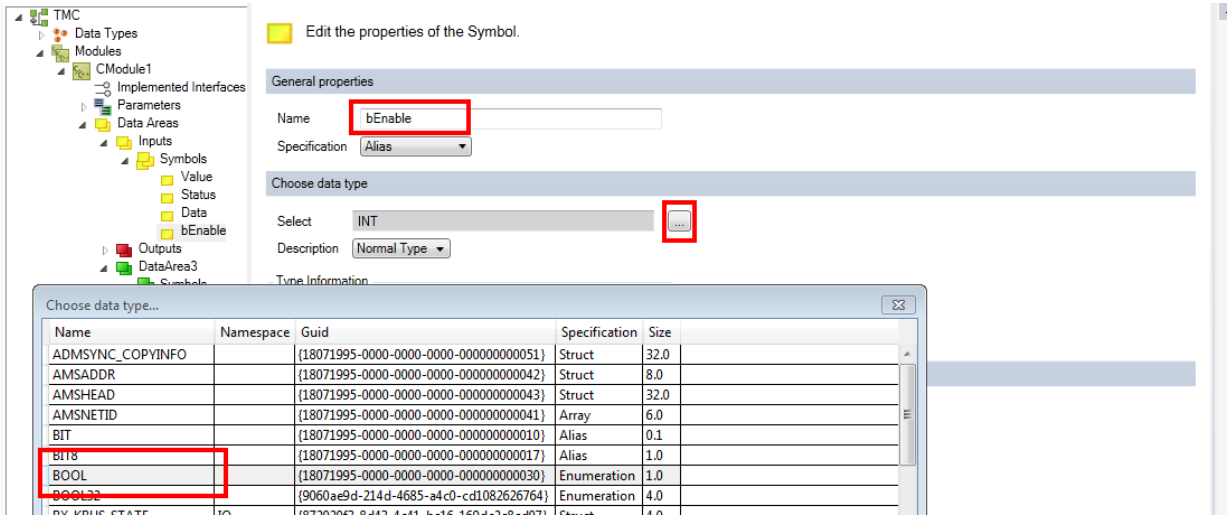
1. 选择数据区的子节点**符号**。
2. 点击 + 按钮，用新变量扩展该数据区。然后会列出一个新条目“Symbol4”。



编辑现有变量的名称或数据类型

1. 选择子节点 **Symbol4** 或双击该行。显示变量属性。

2. 输入新名称，例如“bEnableJob”，并将类型更改为 BOOL。

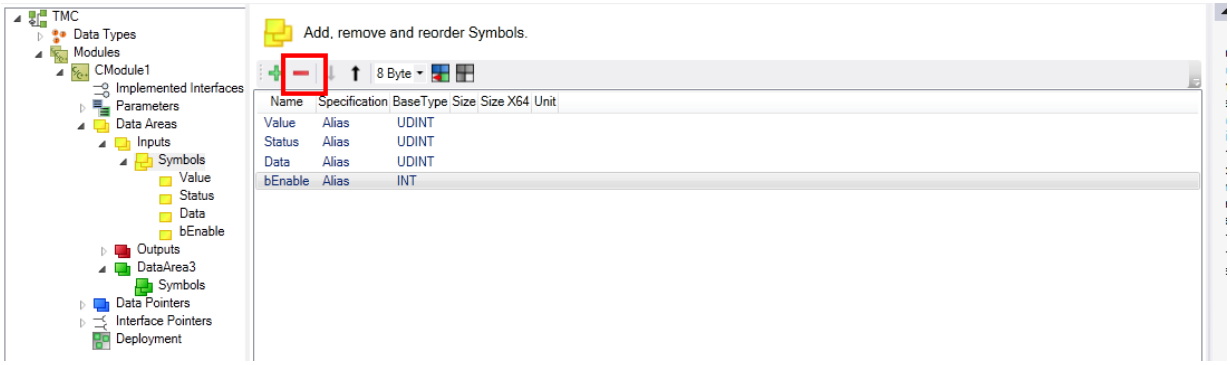


⇨ 在“输入”数据区创建新变量“bEnableJob”。

切记再次运行 TMC Code Generator。

删除现有变量

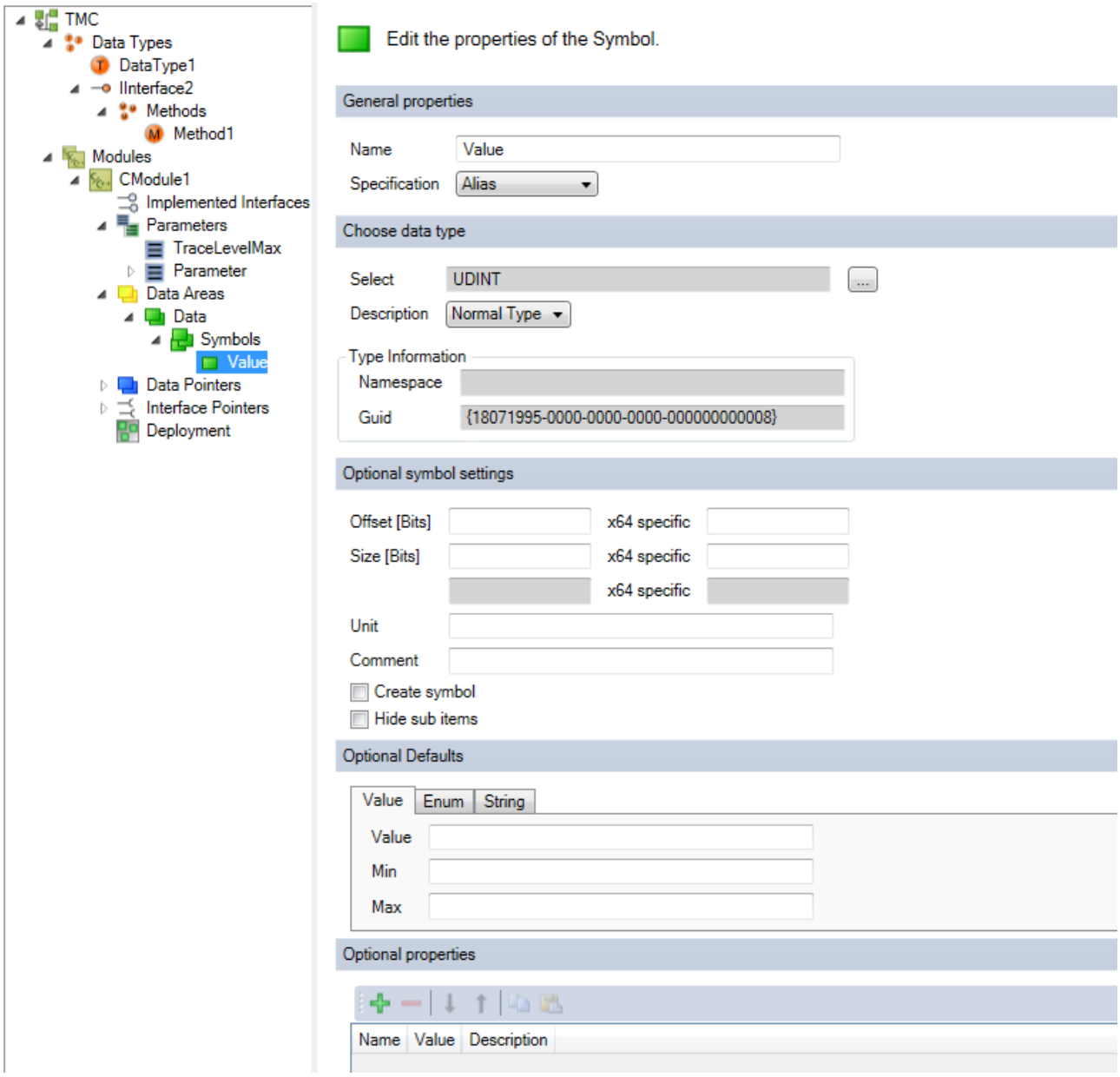
1. 要从数据区删除现有变量，请选择变量，然后点击删除图标：
   在本示例中，选择“MachineStatus1”，然后点击**删除图标**。



2. 再次运行 TMC Code Generator。

11.3.4.3.2 数据区属性

数据区属性：用于编辑数据区属性的对话框。



一般属性

编号：数据区编号。

类型：定义数据区的用途和位置。提供以下选项：

系统管理器中的可链接数据区：

- 输入源
- 输入目标
- 输出源
- 输出目标
- 保留源（适用于 NOV-RAM 内存，请参见附录 [▶ 325]）
- 保留目标（供内部使用）

更多数据区：

- 标准（在系统管理器中可见但不可链接）
- 内部（适用于内部模块符号，可通过 ADS 进入，但在系统管理器中不可见）
- MArea（供内部使用）

- 未指定（与标准相同）

名称：数据区名称。

可选参数设置

大小 [字节]：大小（字节）。已为 x64 提供特殊大小配置。

注释：可选注释，例如在实例配置器中可见。

环境 ID：该数据区所有符号的环境 ID；用于确定映射。

数据类型名称：如果指定，会在类型系统中创建具有指定名称的数据类型。

创建符号：用于创建 ADS 图标的默认设置。

禁用代码生成：启用/禁用代码生成。

可选默认值

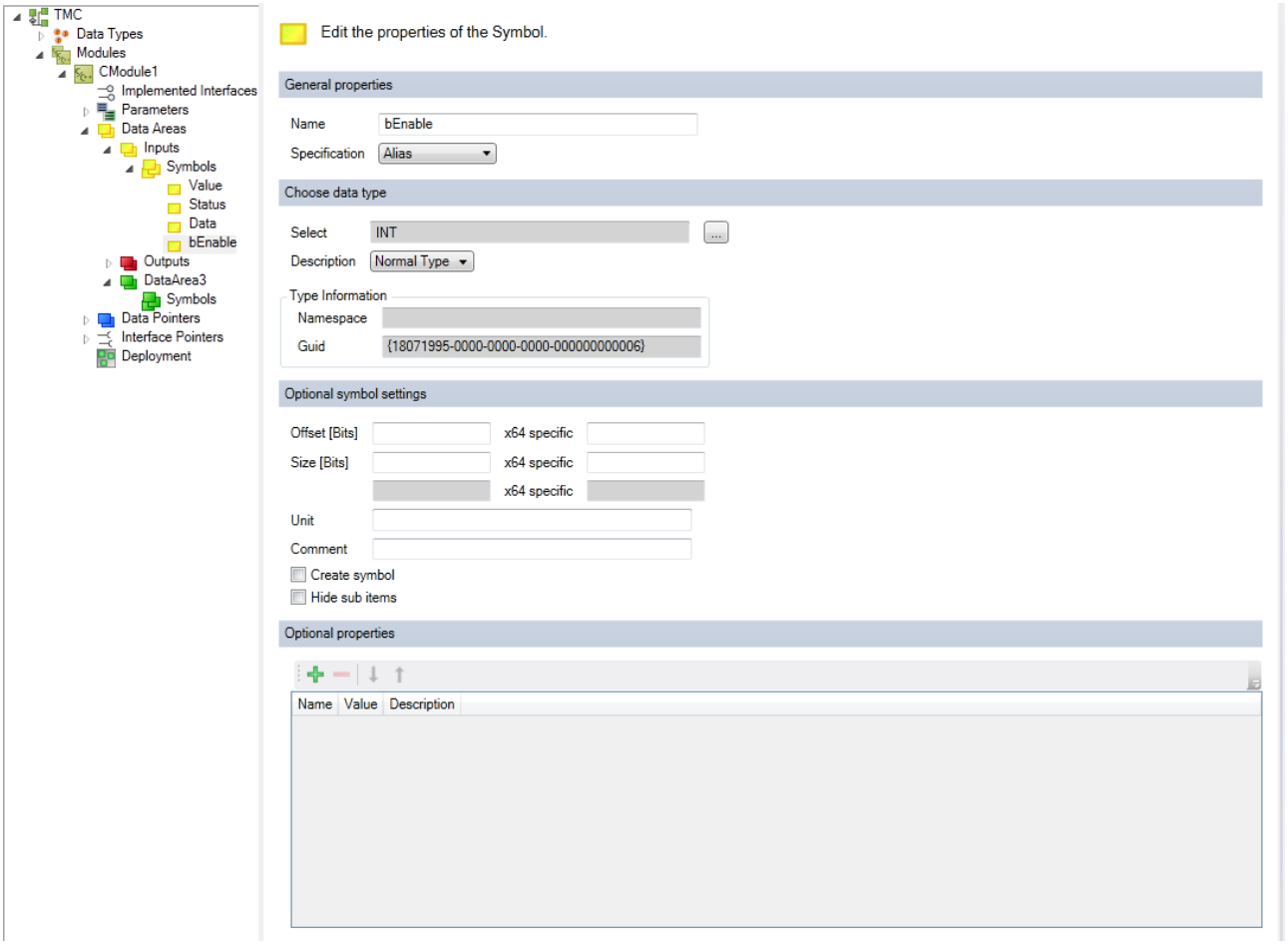
可根据数据类型定义默认值。

可选属性

用于注释数据区的名称、值和说明的表格。
这些信息可见于 TMC 和 TMI 文件。
TwinCAT 功能和客户程序均可使用这些属性。

11.3.4.3.3 符号属性

符号：用于编辑数据区符号的对话框。



一般属性

名称：符号的名称。

规范：符号的数据类型，请参见 数据类型属性 [▶ 101]。

选择数据类型

选择：选择数据类型，可以是基本 TwinCAT 数据类型，也可以是用户定义的数据类型。

说明：定义是否为如下类型：

- 常规类型
- 指针
- 指针指向指针
- 指针指向指针指向指针
- 引用

类型信息

- **命名空间：**所选数据类型的命名空间。
- **GUID**：数据类型的唯一 ID。

可选数据类型设置

偏移 [位]：数据区内符号的偏移量；可为 x64 平台定义不同的偏移量。

大小 [位]：以位为单位的大小（如果指定）。可为 x64 平台定义不同的大小。

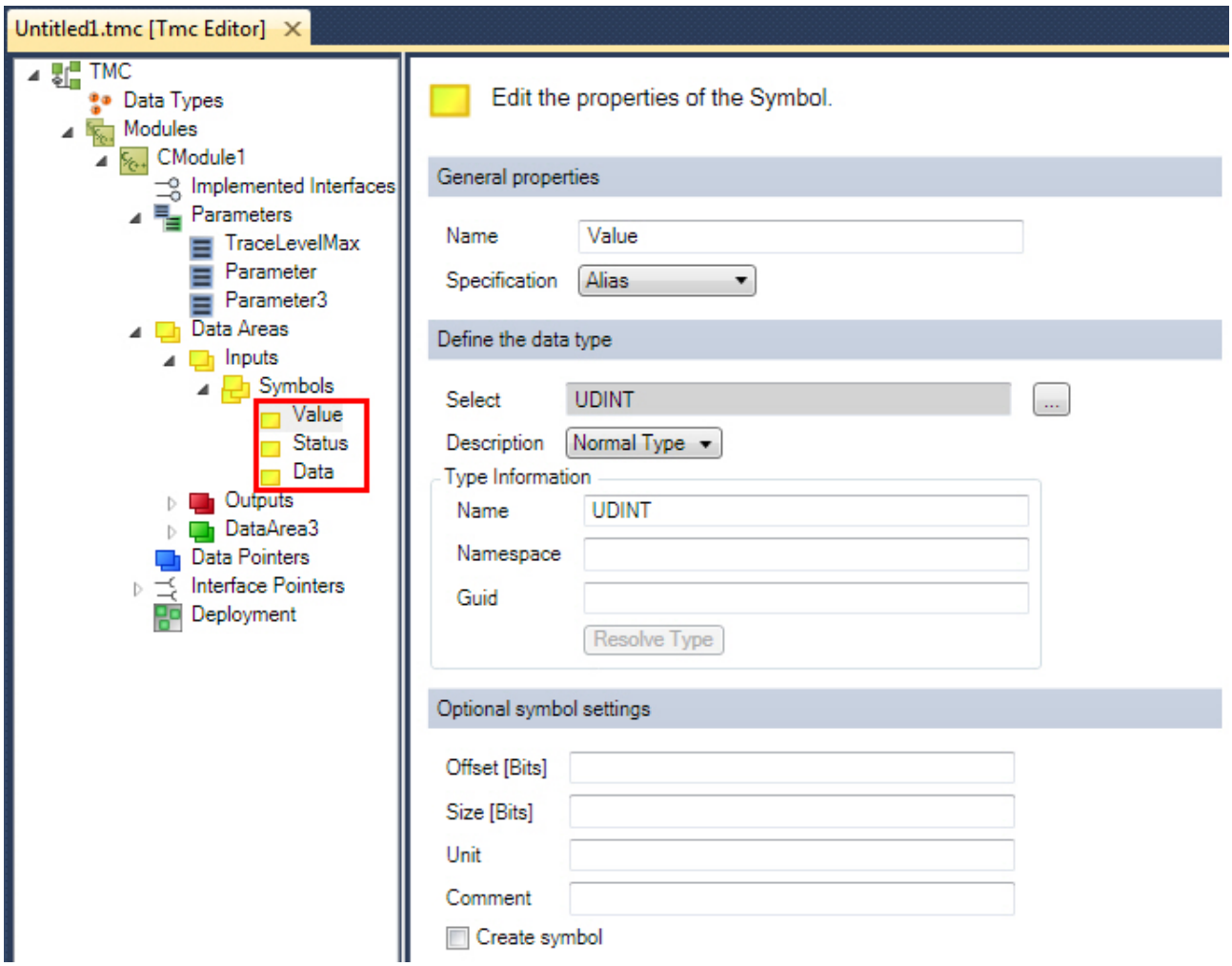
注释：可选注释，例如在实例配置器中可见。

创建符号：用于创建 ADS 图标的默认设置。

隐藏子项：如果变量有子元素，则系统管理器不允许访问子元素。例如，在数组较大的情况下应使用这种方法。

11.3.4.3.3.1 符号属性

数据区符号属性：用于编辑符号属性的对话框。



一般属性

名称：接口名称

规范：参数的数据类型

可提供的规范如下：

- **别名**：创建默认数据类型（如 INT）的别名
- **数组**：创建用户专用数组
- **枚举**：创建用户特定枚举
- **结构体**：创建用户特定结构
- **接口**：创建新接口

定义数据类型

选择：选择数据类型

说明：定义说明

类型信息

名称：所选默认类型的名称

命名空间：用户定义的数据类型命名空间

GUID：数据类型的唯一 ID

可选数据类型设置

偏移 [位]：内存偏移

大小 [位]：计算大小（位）

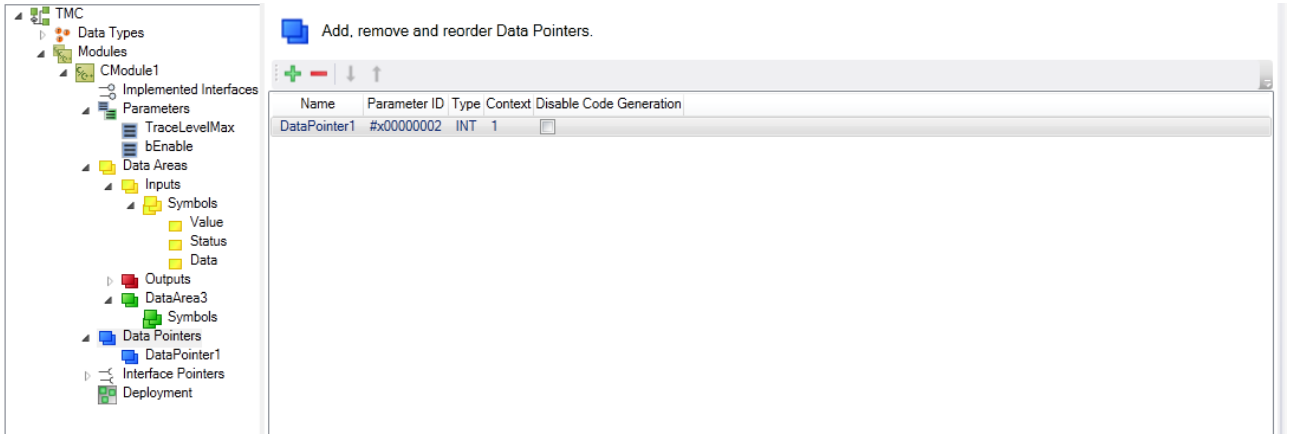
单位：可选

注释：可选

创建符号：创建 ADS 符号的默认设置

11.3.4.4 数据指针

数据指针：用于编辑模块数据指针的对话框。



符号	功能
	添加新数据指针
	删除所选数据指针
	将所选元素下移一个位置
	将所选元素上移一个位置

名称：数据指针的名称。

参数 ID：参数的唯一 ID。

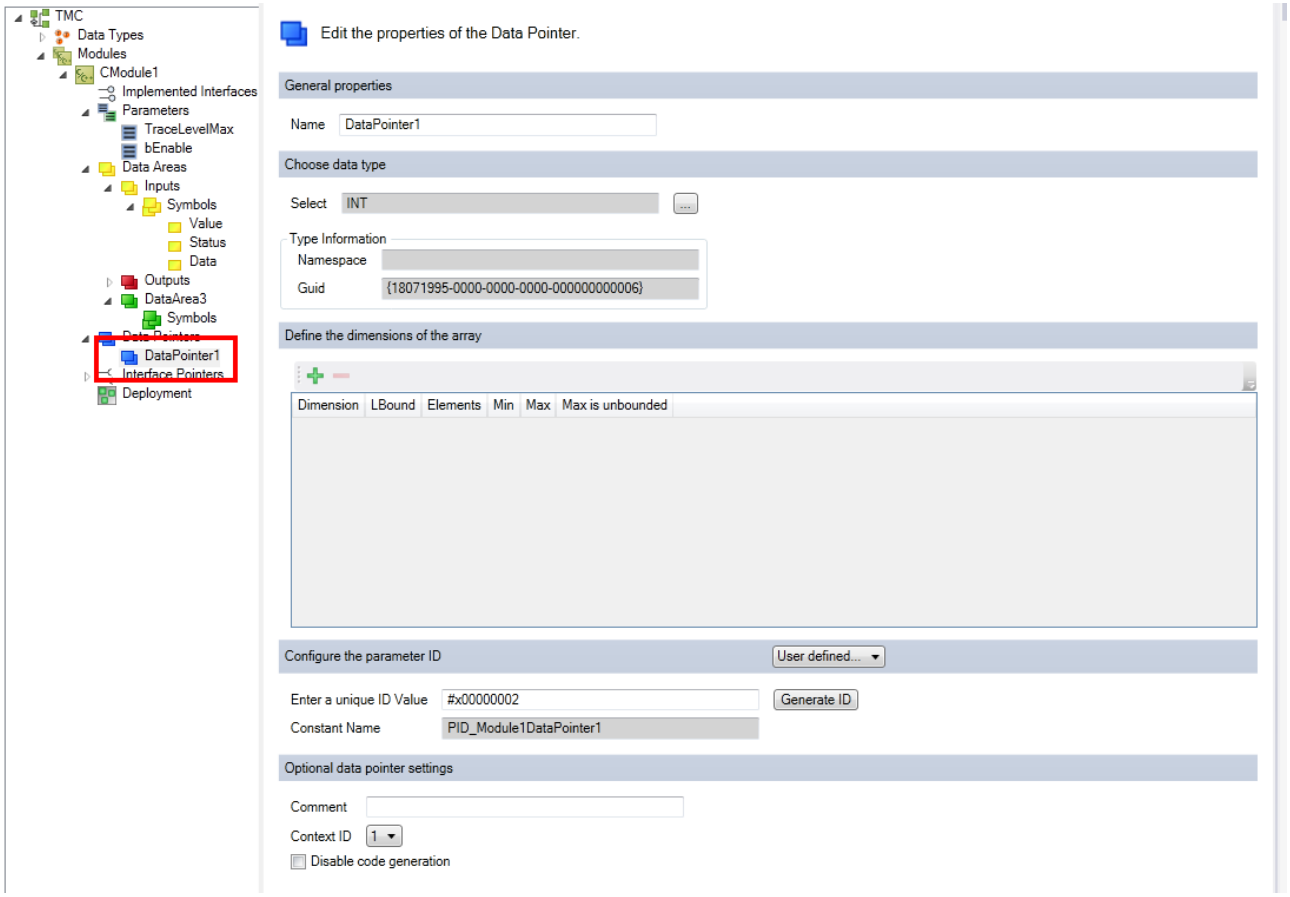
类型：定义指针类型。

环境：显示环境 ID。

禁用代码生成：启用/禁用代码生成。

11.3.4.4.1 属性

数据指针属性：编辑数据指针的属性。



一般属性

名称：数据指针的名称。

定义数据类型

选择：选择数据类型。

类型信息

- **名称**：所选数据类型的名称。
- **GUID**：数据类型的唯一 ID。

定义数组的维度

请参见 数组 [▶ 104] 章。

配置参数 ID

输入唯一 ID 值：输入唯一 ID 值，请参见参数 [▶ 110]。

常量名称：参数 ID 的源代码名称。

可选数据指针设置

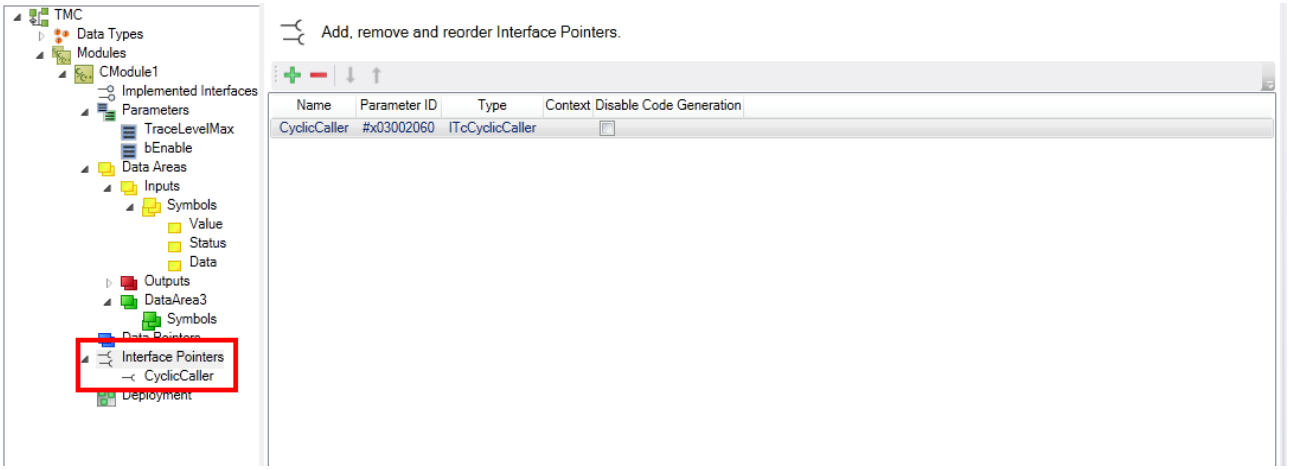
注释：例如在实例配置器中可见的注释。

环境 ID：数据指针的环境 ID。

禁用代码生成：启用/禁用代码生成。

11.3.4.5 接口指针

接口指针：添加、删除和重新排列接口指针。



符号	功能
	添加接口指针
	删除所选指针
	将所选元素下移一个位置
	将所选元素上移一个位置

名称：接口名称。

参数 ID：接口指针的唯一 ID。

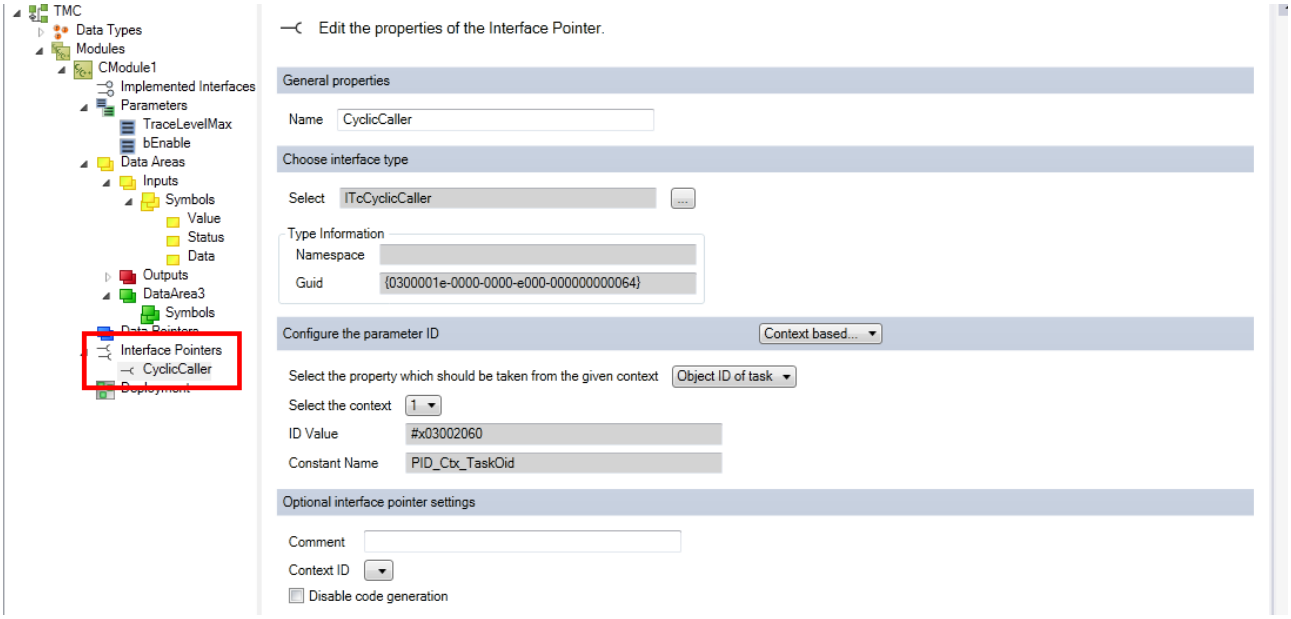
类型：接口指针类型。

环境：接口的环境。

禁用代码生成：启用/禁用代码生成。

11.3.4.5.1 属性

接口指针属性：编辑接口指针的属性。



一般属性

名称：接口指针的名称。

选择基本接口

选择：选择接口。

类型信息

- **命名空间：**接口的命名空间。
- **GUID**：接口的唯一 ID。

配置参数 ID

请参见参数 [▶ 110]。

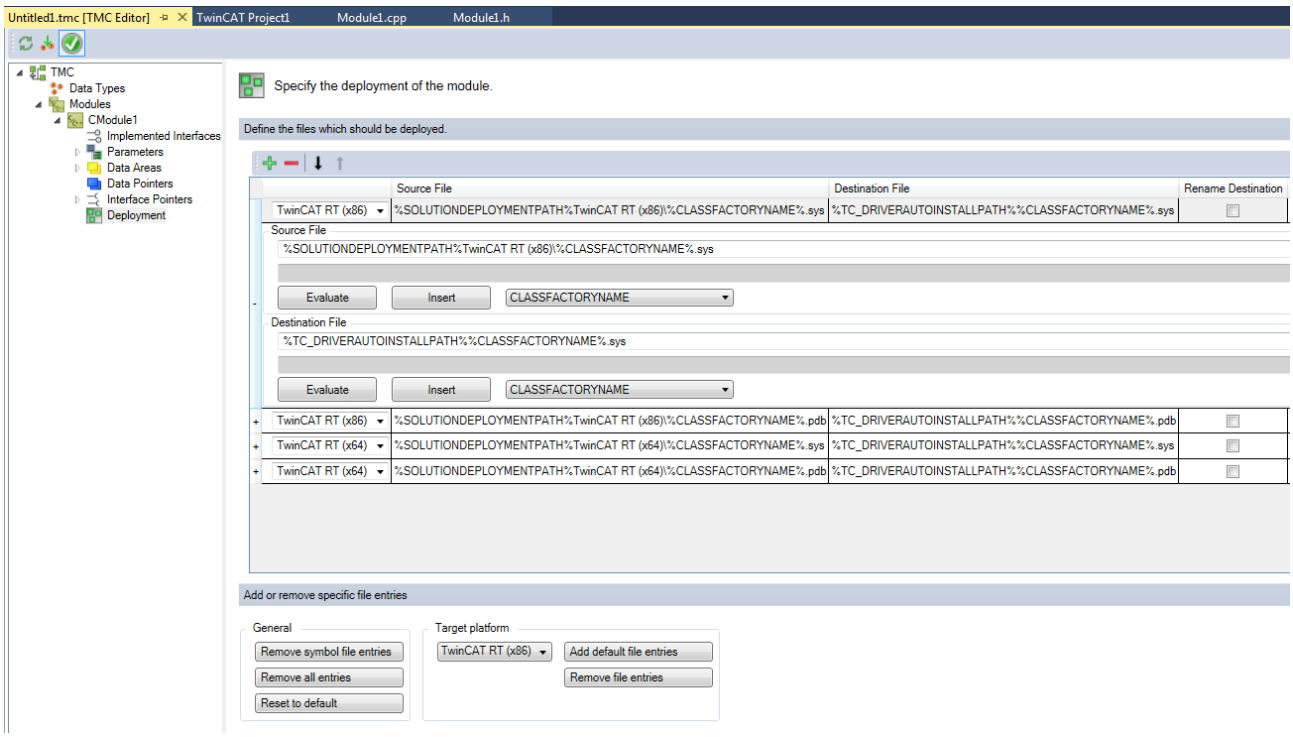
注释：可选

环境 ID：接口指针的环境 ID。

禁用代码生成：启用/禁用代码生成。

11.3.4.6 部署

部署：指定目标系统上所提供模块的存储位置。对于带模块的版本控制 C++ 项目，这些条目为空，并且并不需要。



符号	功能
	添加新文件条目
	删除文件条目
	将所选元素下移一个位置
	将所选元素上移一个位置

通过此对话框可以配置源文件和目标文件，并将其传输到相应平台的目标系统中。

定义应部署的文件。

源文件：源文件的路径。

目标文件：二进制文件的路径。

重命名目标：在传输新文件之前会重命名目标文件。由于此为 Windows 10 的必要条件，因此以隐式方式完成该操作。

各个条目可分别通过开头的 + 或 - 进行展开或折叠。

评估：将计算值放入文本字段进行验证。

插入：添加在下拉列表中选择的变量名。

添加或删除特定文件条目

删除符号文件条目：删除提供符号文件 (PDB) 的条目。

删除所有条目：删除所有条目。

重置为默认值：设置标准条目。

添加默认文件条目：添加所选平台的条目。

删除文件条目：删除所选平台的条目。

分配的源路径和目标路径可能包含虚拟环境变量，这些变量由 TwinCAT XAE/XAR 系统进行解析。

下表列出了这些支持的虚拟环境变量。

虚拟环境变量	注册表条目 (REG_SZ) 位于键 \HKLM\Software\Beckhoff\TwinCAT3 下	默认值 (Windows)	默认值 (TwinCAT/BSD)
%TC_INSTALLPATH%	InstallDir	C:\TwinCAT\3.**1**\	/usr/local/etc/TwinCAT/3.**1**/
%TC_CONFIGPATH%	ConfigDir	C:\TwinCAT\3.**1**\Config\	/usr/local/etc/TwinCAT/3.**1**/Config/
%TC_TARGETPATH%	TargetDir	C:\TwinCAT\3.**1**\Target\	/usr/local/etc/TwinCAT/3.**1**/Target/
%TC_SYSTEMPATH%	SystemDir	C:\TwinCAT\3.**1**\System\	/usr/local/etc/TwinCAT/3.**1**/System/
%TC_BOOTPRJPATH%	BootDir	C:\TwinCAT\3.**1**\Boot\	/usr/local/etc/TwinCAT/3.**1**/Boot/
%TC_RESOURCEPATH%	ResourceDir	C:\TwinCAT\3.**1**\Target\Resource\	/usr/local/etc/TwinCAT/3.**1**/Target/Resource/
%TC_REPOSITORYPATH%	RepositoryDir	C:\TwinCAT\3.**1**\Repository\	/usr/local/etc/TwinCAT/3.**1**/Repository
%TC_DRIVERPATH%	DriverDir	C:\TwinCAT\3.**1**\Driver\	不可用
%TC_DRIVERAUTOINSTALLPATH%	DriverAutoInstallDir	C:\TwinCAT\3.**1**\Driver\AutoInstall\	/usr/local/etc/TwinCAT/3.**1**/
%CLASSFACTORYNAME%		<Name of the Class Factory>	<Name of the Class Factory>

11.4 TwinCAT 模块实例配置器

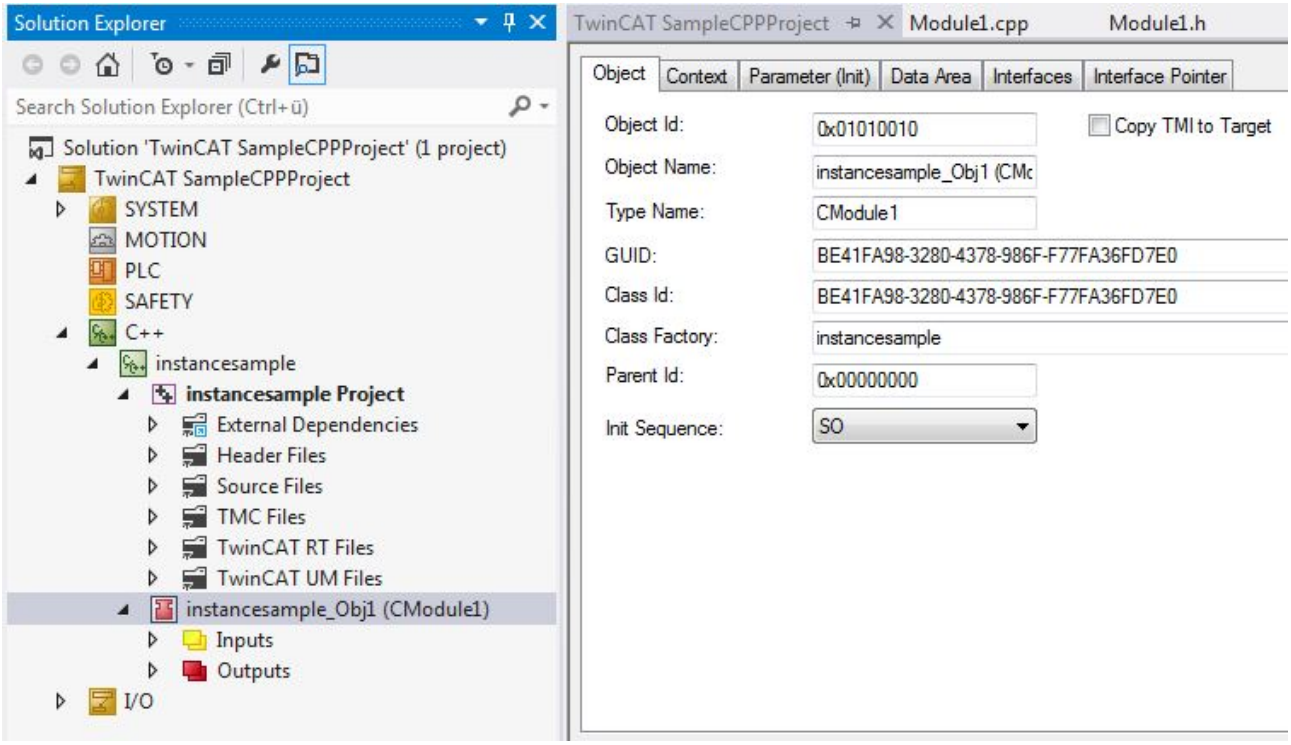
上述 TwinCAT 3 模块类 (TMC) 编辑器在类级别定义驱动程序。这些均为实例化，必须通过 TwinCAT 3 实例配置器进行配置。

该配置适用于环境（包括调用模块的任务）、参数和指针。

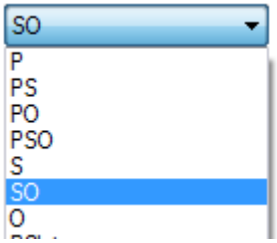
右键单击 C++ 项目文件夹可创建 C++ 类实例；请参见快速入门 [▶ 57]。本章将详细介绍这些实例的配置。

双击生成的实例，打开包含多个窗口的配置对话框。

11.4.1 对象



- **对象 ID：**用于在 TwinCAT 系统中识别该实例的对象 ID。
- **对象名称：**用于在解决方案资源管理器树中显示实例的对象名称。
- **类型名称：**实例的类型信息（类名）。
- **GUID：**模块类 GUID。
- **类 ID：**实现类的类 ID（GUID 和 ClassId 通常相同）。
- **类工厂：**指提供用于创建模块实例的类工厂的驱动程序。
- **父类 ID：**包含父类的 ObjectID（若有）。
- **Init 序列：**指定初始化状态，以确定交互模块的启动行为。有关状态机的详细说明，请参见 TwinCAT 模块状态机 [▶ 43]。



指定多个 TcCOM 实例的启动行为

TcCOM 实例可以相互引用，例如通过数据或接口指针进行交互。为了确定启动行为，**Init 序列**可为所有其他模块指定每个 TcCOM 实例要“保持”的状态。

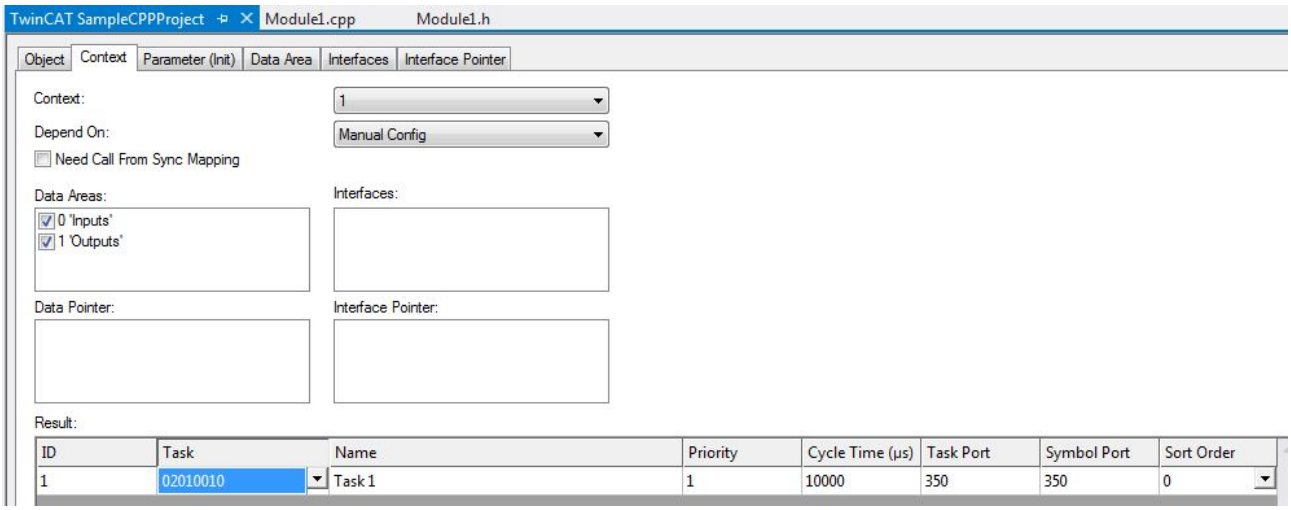
“Init 序列”的名称由 TcCOM 状态机的简称组成。如果“Init 序列”的名称中包含某个状态（I、P、S、O）的简称，则模块会在该状态下等待，直到所有其他模块都至少达到该状态。在接下来的转换中，模块可以参考所有其他模块实例的信息，以便至少处于这种状态。

例如，如果某个模块的 Init 序列为“PS”，则其他所有模块的 IP 转换都会执行，这样所有模块均处于“Preop”状态。

随后是模块的 PS 转换，模块可以确信其他模块处于“Preop”状态。

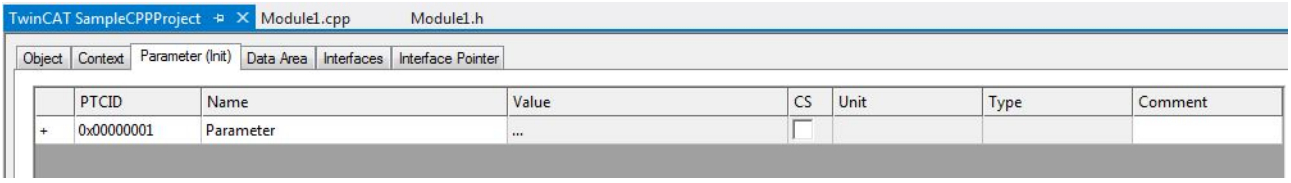
- **将 TMI 复制到目标：**生成 TMI（TwinCAT 模块实例）文件并将其传输到目标。

11.4.2 环境



- **环境：**选择要配置的环境（有关添加不同环境的信息，请参见 TMC 编辑器）。
  一个数据区会分配至一个环境。
- **数据区/接口/数据指针和接口指针：**每个实例都可配置为具有或不具有 TMC 中定义的元素。
- **结果表：**需要配置的 ID 列表。至少必须对任务的环境（“任务”列）进行相应配置。

11.4.3 参数 (Init)

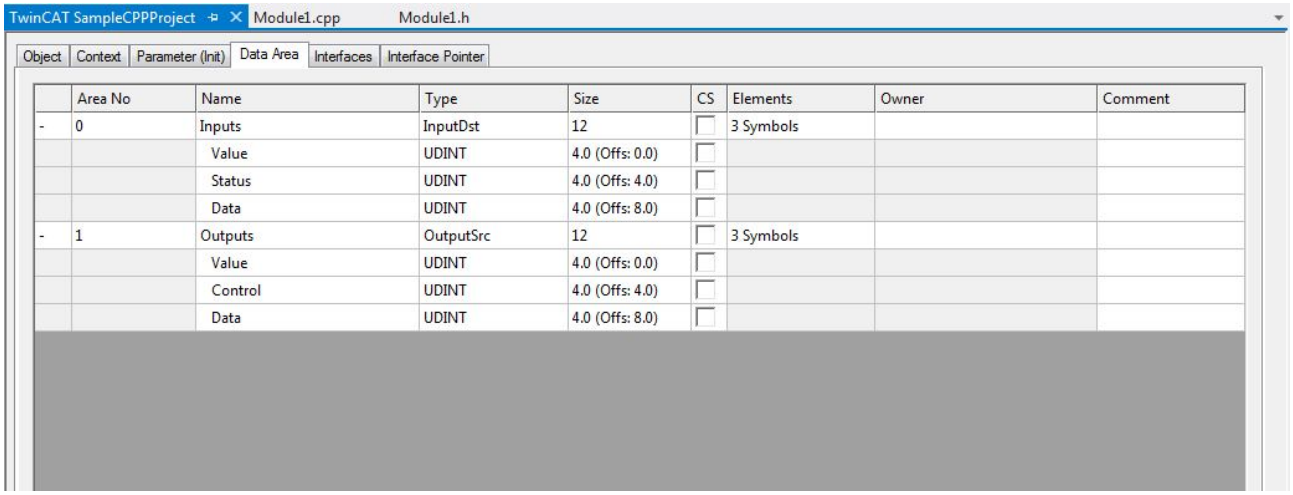


所有参数的列表（如 TMC 中所定义）均可通过每个实例的值进行初始化。

专用参数 ID (PTCID) 用于自动设置值。如 参数 [▶ 110] 所述，可使用 TMC 编辑器的参数对话框窗口配置这些参数。

CS (CreateSymbol) 复选框可为每个参数创建 ADS 符号，以便从外部访问。

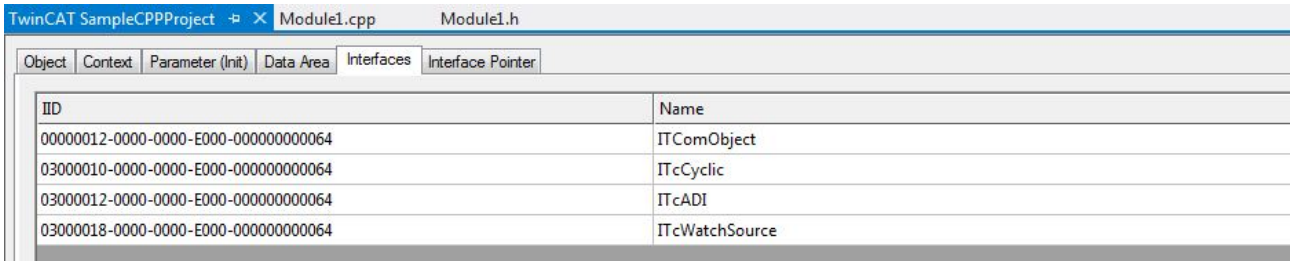
11.4.4 数据区



所有数据区及其变量的列表（如 TMC 中所定义）。

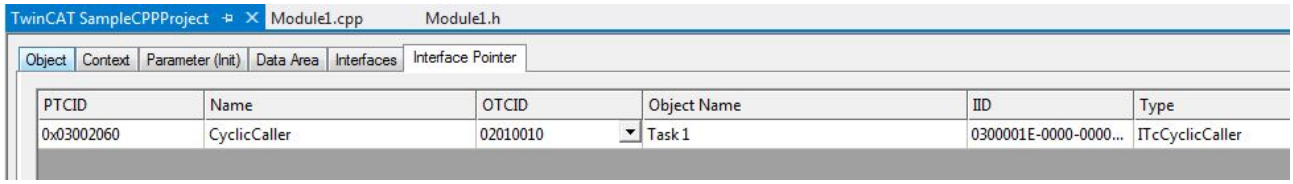
CS (CreateSymbol) 复选框会为每个参数创建 ADS 符号，因此可以从外部访问变量。

11.4.5 接口



所有已实现接口的列表（如 TMC 中所定义）。

11.4.6 接口指针

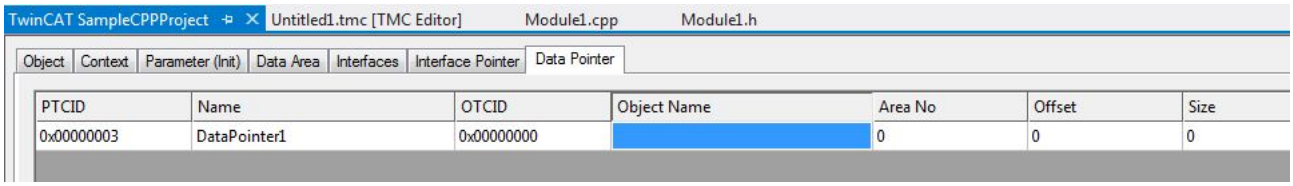


所有接口指针的列表（如 TMC 中所定义）。

专用参数 ID (PTCID) 用于自动设置值。如 参数 [▶ 110] 所述，可使用 TMC 编辑器的参数对话框窗口配置这些参数。

OTCID 列定义了要使用的指向实例的指针。

11.4.7 数据指针



所有数据指针的列表（如 TMC 中所定义）。

专用参数 ID (PTCID) 用于自动设置值。如 参数 [▶ 110] 所述，可使用 TMC 编辑器的参数对话框窗口配置这些参数。

OTCID 列定义了要使用的指向实例的指针。

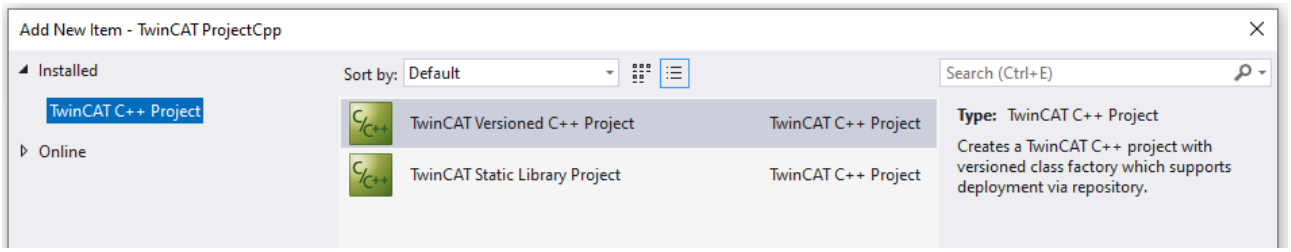
11.5 客户特定项目模板

TwinCAT 3 嵌入了 Visual Studio，因此也会使用所提供的项目管理。TwinCAT 3 C++ 项目是 TwinCAT 项目文件夹（TwinCAT 解决方案）中的“嵌套项目”。

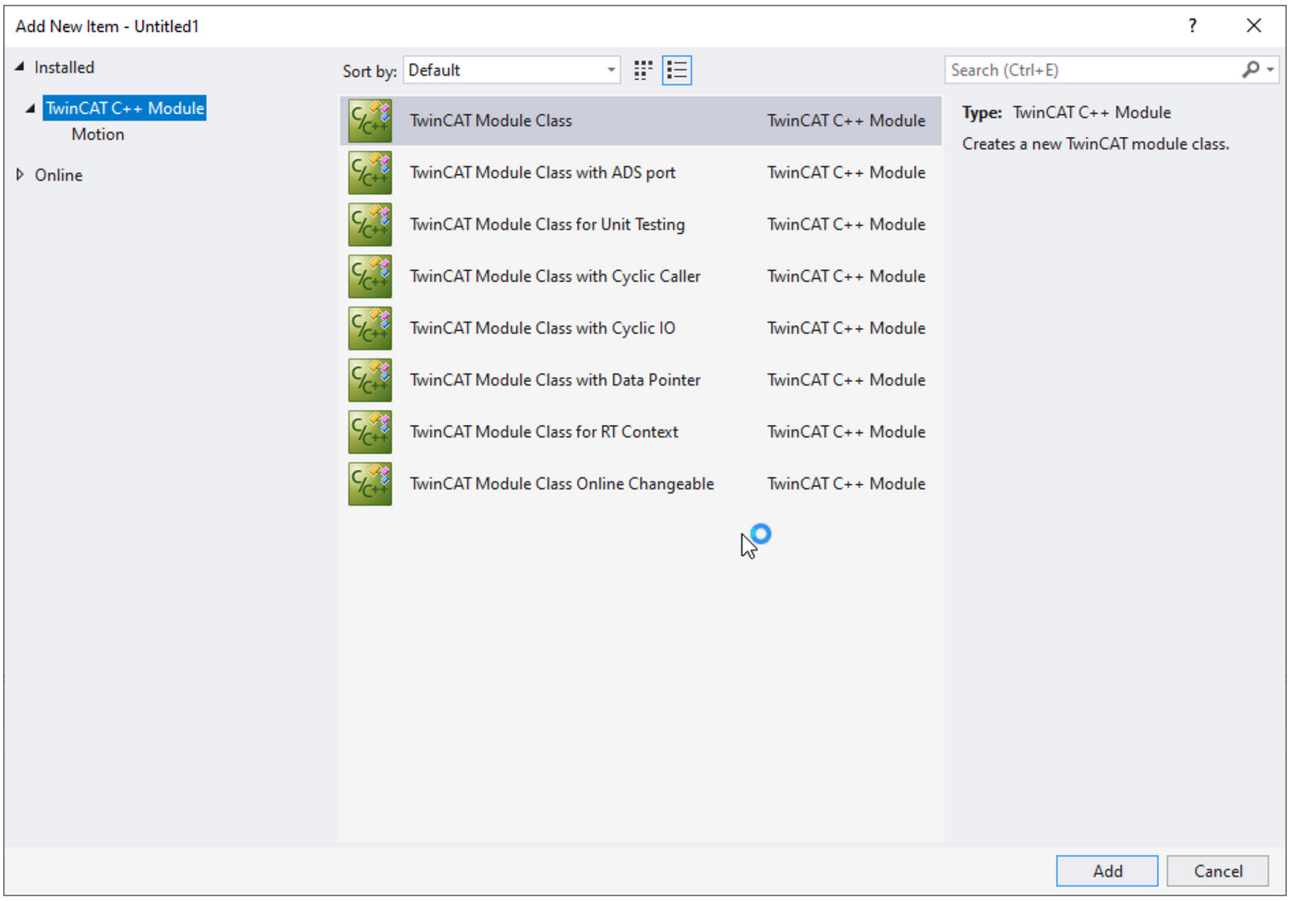
本节文档介绍了客户如何实现自己的项目模板。

11.5.1 概述

创建 TwinCAT C/C++ 项目时，首先要启动 TwinCAT C++ 项目向导。后者可为 TwinCAT 版本控制 C++ 项目生成一个框架。实际功能会在 TwinCAT 模块中实现。



在创建新项目时会自动启动 TwinCAT 类向导，以便添加第一个模块。不同的模块由同一个 TwinCAT 类向导生成，但模块的具体设计是通过模板实现的。



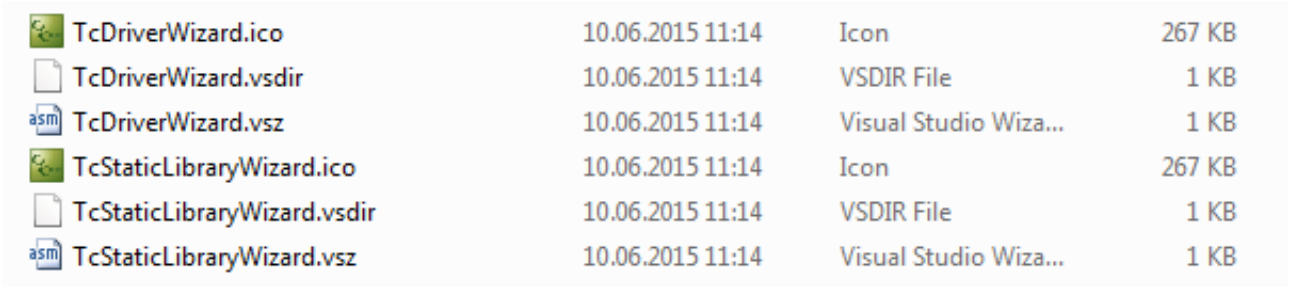
11.5.2 涉及的文件

几乎所有相关信息都包含在 C:\TwinCAT\3.1\Components\Base\CppTemplate 目录中：

如果要创建驱动程序项目，TwinCAT C++ 项目向导会调用 TwinCAT 模块类向导。

目录：驱动程序和类

在驱动程序目录 （用于 TwinCAT C++ 项目向导）和 类目录（用于 TwinCAT 模块类向导）中定义了相应的项目类型，每个项目类型包含 3 个文件：



.vsdir 文件提供的信息会在启动相关助手向导时使用。其本质上是名称、简要说明以及类型为 .vsz 的文件名，其中包含此项目类型的详细信息。
MSDN 中的一般功能说明可见于 https://msdn.microsoft.com/de-de/library/Aa291929%28v=VS.71%29.aspx。

.vsdir 文件中引用的 .vsz 文件提供了向导所需的信息。
此处最重要的信息是要启动的向导和参数列表。

这两个向导都有 .xml 文件作为参数，用于说明从模板到特定项目的源文件转换。这些模板与源代码模板等位于模块目录中。
若要创建驱动程序，TwinCAT C++ 项目向导会通过 **TriggerAddModule** 参数启动 TwinCAT 模块类向导。

MSDN 中的一般说明可见于：https://msdn.microsoft.com/de-de/library/Aa291929%28v=VS.71%29.aspx。

.ico 文件仅提供一个图标。

目录：模板

源代码模板和用于 TwinCAT 模块类向导的 .vsz 文件中命名的 .xml 文件均位于模板目录下的相应子目录中。

该 .xml 文件介绍的是从模板到实际项目的程序。

11.5.3 转换

转换说明（XML 文件）

配置文件以 XML 格式说明将模板文件转换到项目文件夹的过程。通常情况下，这些文件是 .cpp/.h 文件，如果适用，还有项目文件；然而，也可以处理所有类型的文件。

根节点是 <ProjectFileGeneratorConfig> 元素。可直接在此节点设置 useProjectInterface="true" 属性。将 Visual Studio 模式下的处理程序设置为生成项目（而非 TC-C++ 模块）。

此处有多个 <FileDescription> 元素，每个元素说明的都是文件转换。在这些元素之后，还可以在 <Symbols> 元素中定义可用于转换的符号。

模板文件的转换

<FileDescription> 元素的结构如下：

```
<FileDescription openFile="true">
<SourceFile>FileInTemplatesDirectory.cpp</SourceFile>
<TargetFile>[!output SYMBOLNAME].cpp</TargetFile>
<Filter>Source Files</Filter>
</FileDescription>
```

- 模板目录中的源文件指定为 <SourceFile>。
- 项目目录中的目标文件指定为 <TargetFile>。符号通常通过 [!output...] 命令使用。
- 属性“copyOnly”可用于指定是否对文件进行转换，即是否执行源文件所述的转换。否则，只会复制文件。
- 属性“openFile”可用于指定是否在 Visual Studio 中创建项目后打开文件。

- 滤波：在项目中创建一个滤波。
  为此，必须在 <ProjectFileGeneratorConfig> 中设置 useProjectInterface="true" 属性。

转换说明

模板文件中使用的是说明转换本身的命令。

可提供以下命令：

- `[!output SYMBOLNAME]`
  该命令用符号的值替换命令。有许多预定义的符号可供选择。
- `[!if SYMBOLNAME]`, `[!else]` 和 `[!endif]` 介绍的是在转换过程中仅在特定情况下整合相应文本的可能性。

符号名称

可以通过三种方式为转换说明提供符号名称。
上述命令使用这些名称执行替换。

1. 配置文件中直接包含的一系列预定义符号：
   XML 文件中提供了 <Symbols> 列表。可在此处定义符号：
   ```
   <Symbols>
   <Symbol>
   <Name>CustomerSymbol</Name>
   <Value>CustomerString</Value>
   </Symbol>
   </Symbols>
   ```
2. 生成的目标文件名可通过添加“symbolName”属性来提供：
   ```
   <TargetFile symbolName="CustomerFileName">[!output SYMBOLNAME].txt</
   TargetFile>
   ```
3. 重要符号由系统本身提供。

符号名称（项目）	描述
PROJECT_NAME	Visual Studio 对话框中的项目名称。
PROJECT_NAME_UPPERCASE	以大写字母表示的项目名称。
WIN32_WINNT	0x0400
DRVID	驱动程序 ID 格式为 0x03010000
PLATFORM_TOOLSET	工具集版本，如 v100
PLATFORM_TOOLSET_ELEMENT	作为 XML 元素的工具集版本，例如 <PlatformToolset>v100</PlatformToolset>
NEW_GUID_REGISTRY_FORMAT	创建新的 GUID，格式为：{48583F97-206A-4C7C-9EF2-D5C8A31F7BDC}

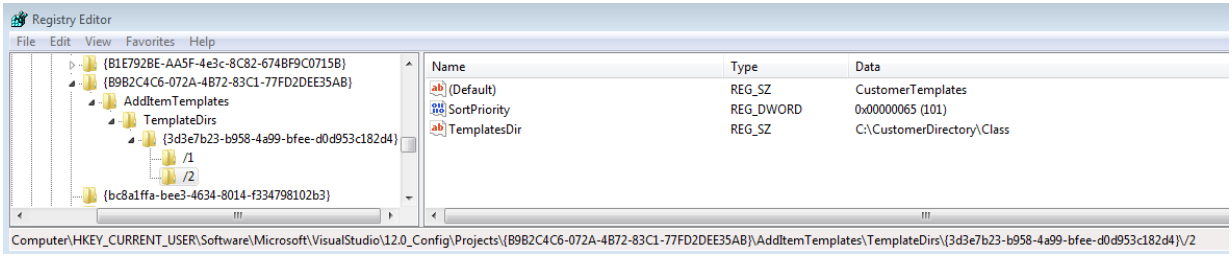
符号名称（类别）	描述
PROJECT_NAME	Visual Studio 对话框中的项目名称。
HEADER_FILE_NAME	由用户在向导对话框中输入。
SOURCE_FILE_NAME	由用户在向导对话框中输入。
CLASS_NAME	由用户在向导对话框中输入。
CLASS_SHORT_NAME	由用户在向导对话框中输入。
CLASS_ID	向导创建的新 GUID。
GROUP_NAME	C++
TMC_FILE_NAME	用于识别 TMC 文件。
NEW_GUID_REGISTRY_FORMAT	创建新的 GUID，格式为：{48583F97-206A-4C7C-9EF2-D5C8A31F7BDC}

11.5.4 处理注意事项

客户特定目录中的模板

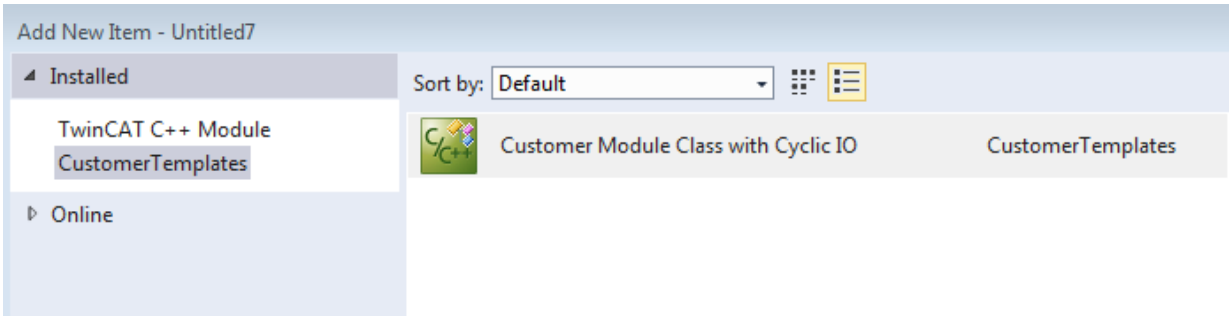
模板也可以存储在常规 TwinCAT 目录之外。

1. 在注册表中，展开创建节点 /2 的搜索路径（本例中为 V12.0，即 VS 2013）：
   注册表键：
   `HKEY_CURRENT_USER\Software\Microsoft\VisualStudio\12.0_Config\Projects\{B9B2C4C6-072A-4B72-83C1-77FD2DEE35AB}\AddItemTemplates\TemplateDirs\{3d3e7b23-b958-4a99-bfee-d0d953c182d4}\`



2. 提升 SortPriority。
3. 建议：在目录中创建一个名为“类”的子目录（在注册表中输入）和一个名为“模板”的子目录，以便将 .vsz/.vsdir/.ico 文件与模板分开。
4. 调整文件中的路径。

⇨ 因此，模板有专门规定：



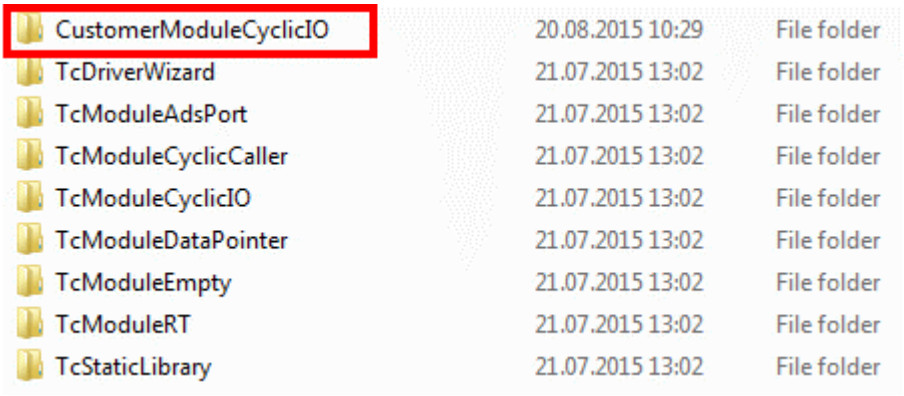
例如，该目录或目录结构现在可以在版本管理系统中进行版本控制，并且不会受到 TwinCAT 安装/更新的影响。

快速入门

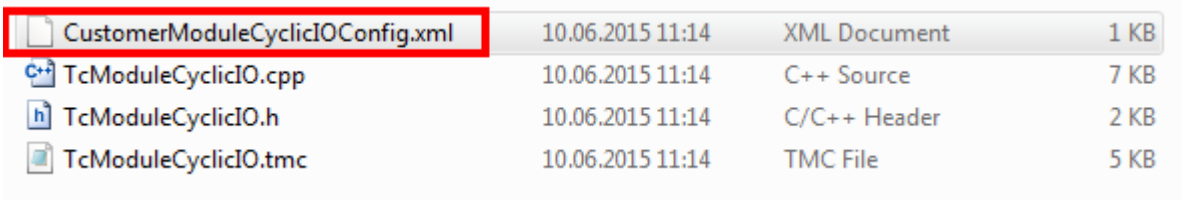
在 MSDN 中，向导环境的一般介绍是入口点：https://msdn.microsoft.com/de-de/library/7k3w6w59%28v=VS.120%29.aspx。

本文介绍如何借助 TwinCAT C++ 模块向导使用模板创建客户特定模块。

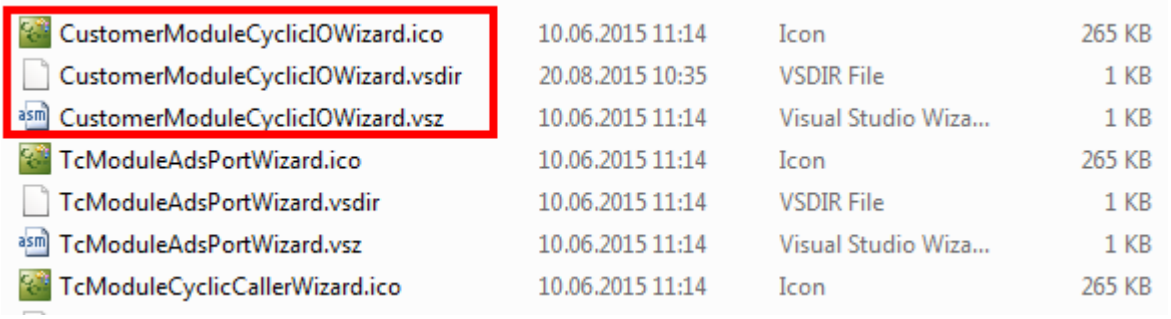
1. 将现有模块模板作为
   在 **C:\TwinCAT\3.1\Components\Base\CppTemplate\Templates** 中的复制模板



2. 重命名文件夹中的 .xml 文件

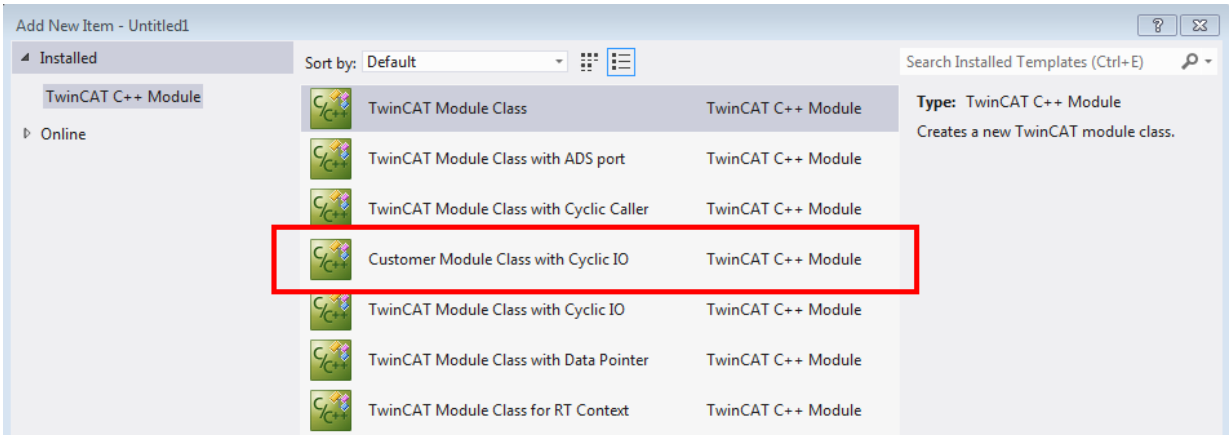


3. 将相应的 .ico/.vsdir/.vsz 文件同时复制到 Class/



4. 现在，在 .vsdir 文件中引用复制的 .vsz 文件并调整说明。
5. 在 .vsz 文件中输入第 2 步创建的 .xml 文件。
6. 现在，您可以更改 Template/CustomerModuleCyclicIO/ 目录中的源文件。使用该模板创建项目时，.xml 会负责替换。

⇨ TwinCAT 模块类向导现在会显示新项目供选择：



例如，如果 vsxproj 也要以更改后的形式提供，建议调整 TwinCAT C++ 项目向导副本。

如有必要，还应考虑使用 .props 文件中的设置，以便在根据模板创建的现有项目中也能更改设置，例如，由于版本管理系统更新 .props 文件。

在现有项目的基础上另行创建

可行的方法是创建一个完成的项目，然后将其转换为模板。

1. 将清理后的项目复制到 Templates\ 文件夹中。
2. 创建转换说明（XML 文件）。
3. 通过所述替换方法准备源文件和项目文件。
4. 提供 .ico/.vsdir/.vsz 文件。

1 编程参考手册

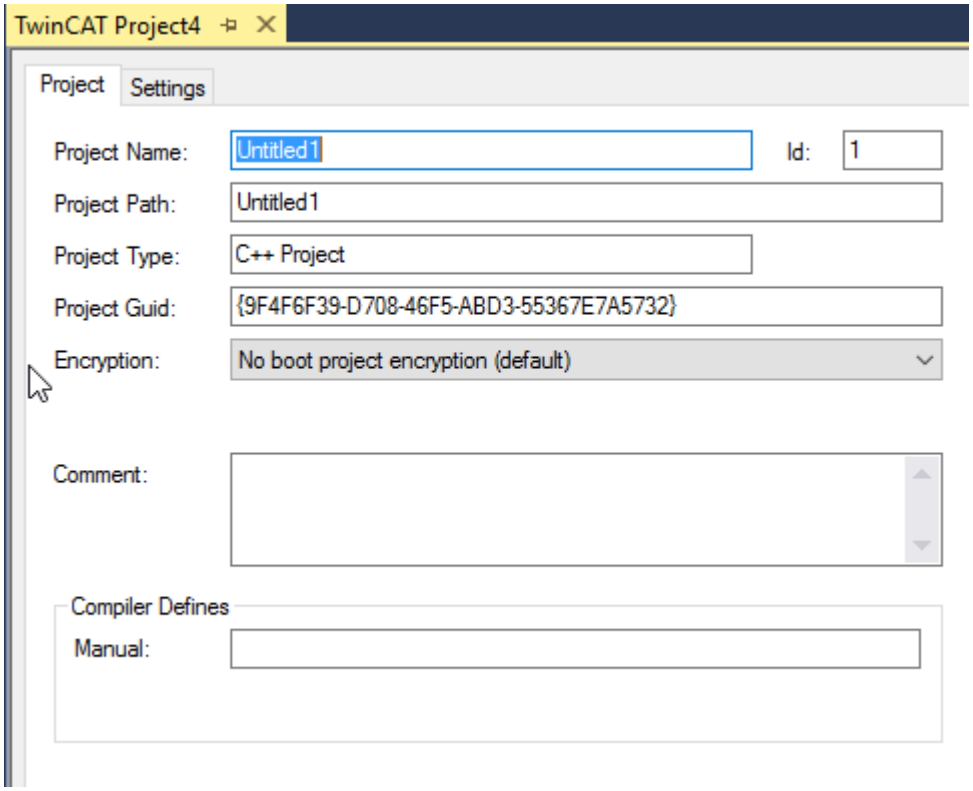
2

TwinCAT 可提供多种基本功能。这些功能对 TwinCAT C++ 程序员都非常有用，相关内容已在此处记录。

有大量 C++ 示例，其中包含处理模块和接口的重要信息。

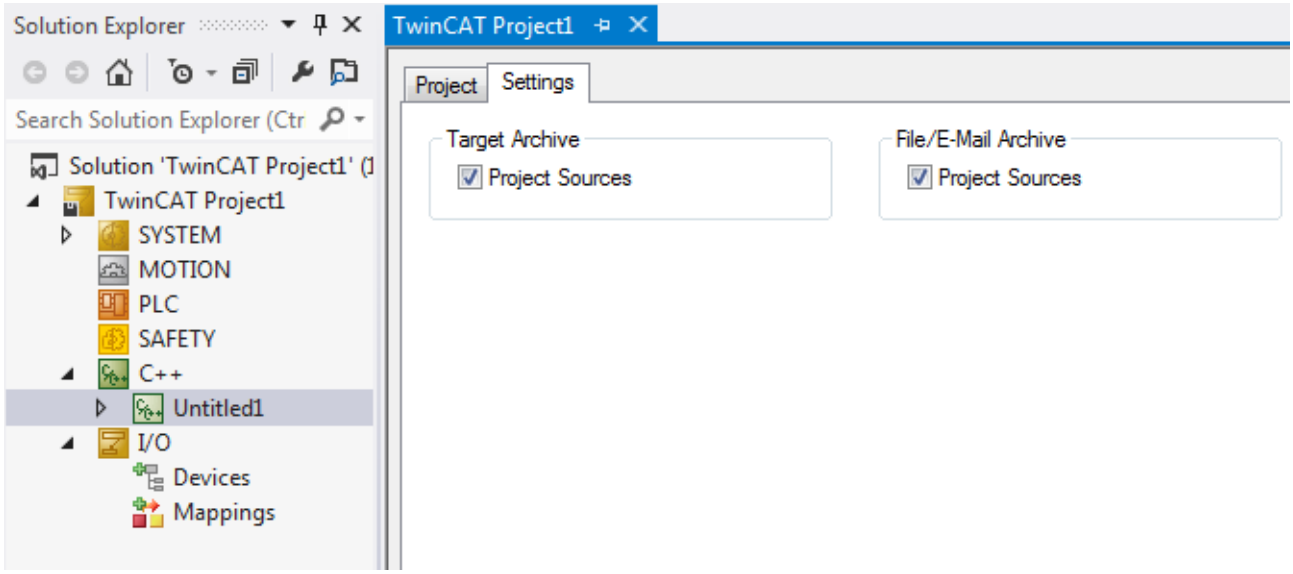
TwinCAT C++ 项目

TwinCAT C++ 项目有多个参数，双击 TwinCAT C++ 项目（此处项目名称为“Untitled1”）即可打开。



此处无法重命名（见 重命名 TwinCAT C++ 项目 [▶ 214]）

可在此处设置二进制模块加密，有关要求的详细说明请参见 加密模块 [▶ 51] 章。



对于传输到目标系统或通过电子邮件发送的两种存档类型，可在此处设置是否包含源的选项。

因此，取消选择时会创建空档案。

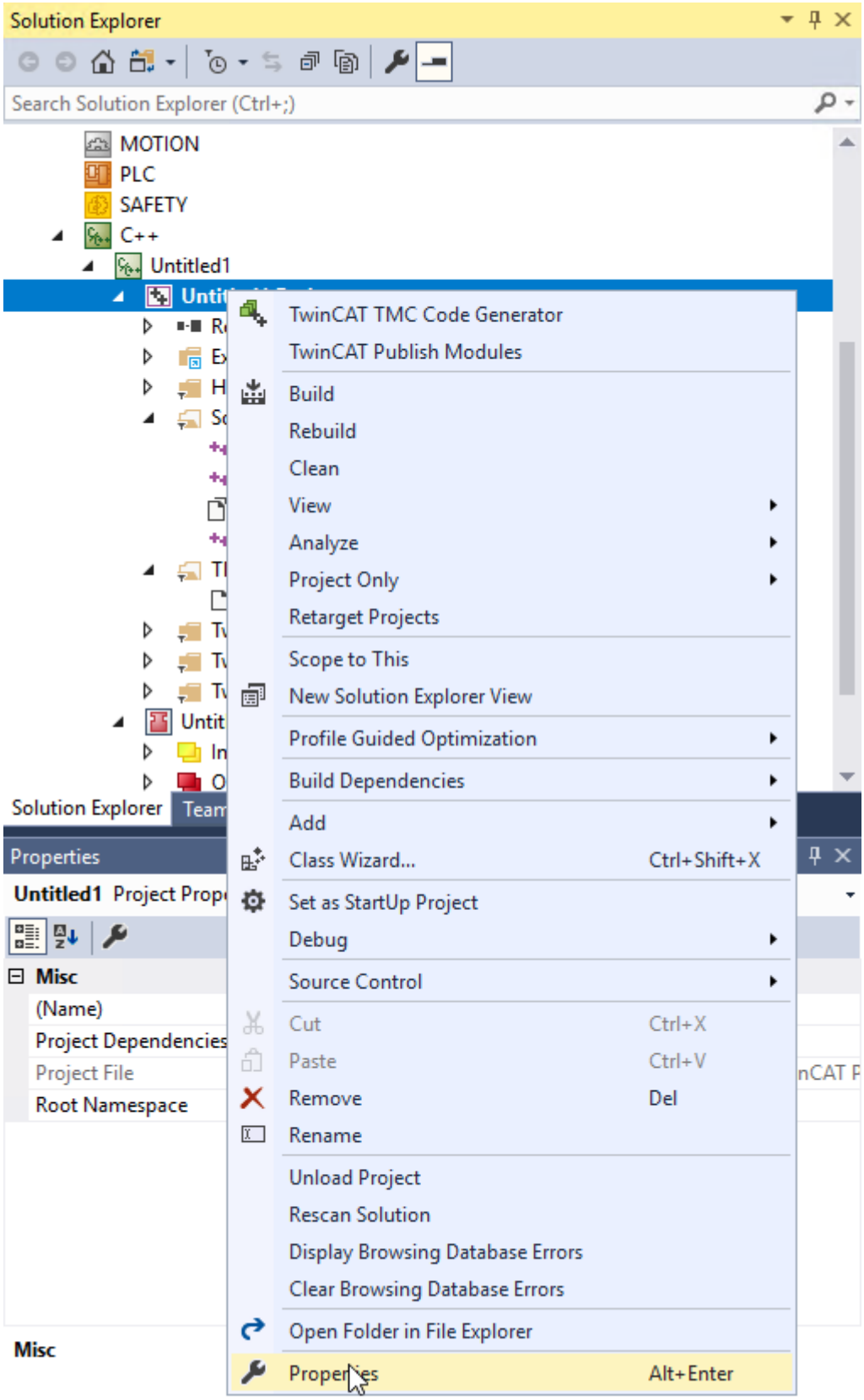
12.1 TwinCAT C++ 项目属性

TwinCAT 3.1 版本 4024.0 及以上

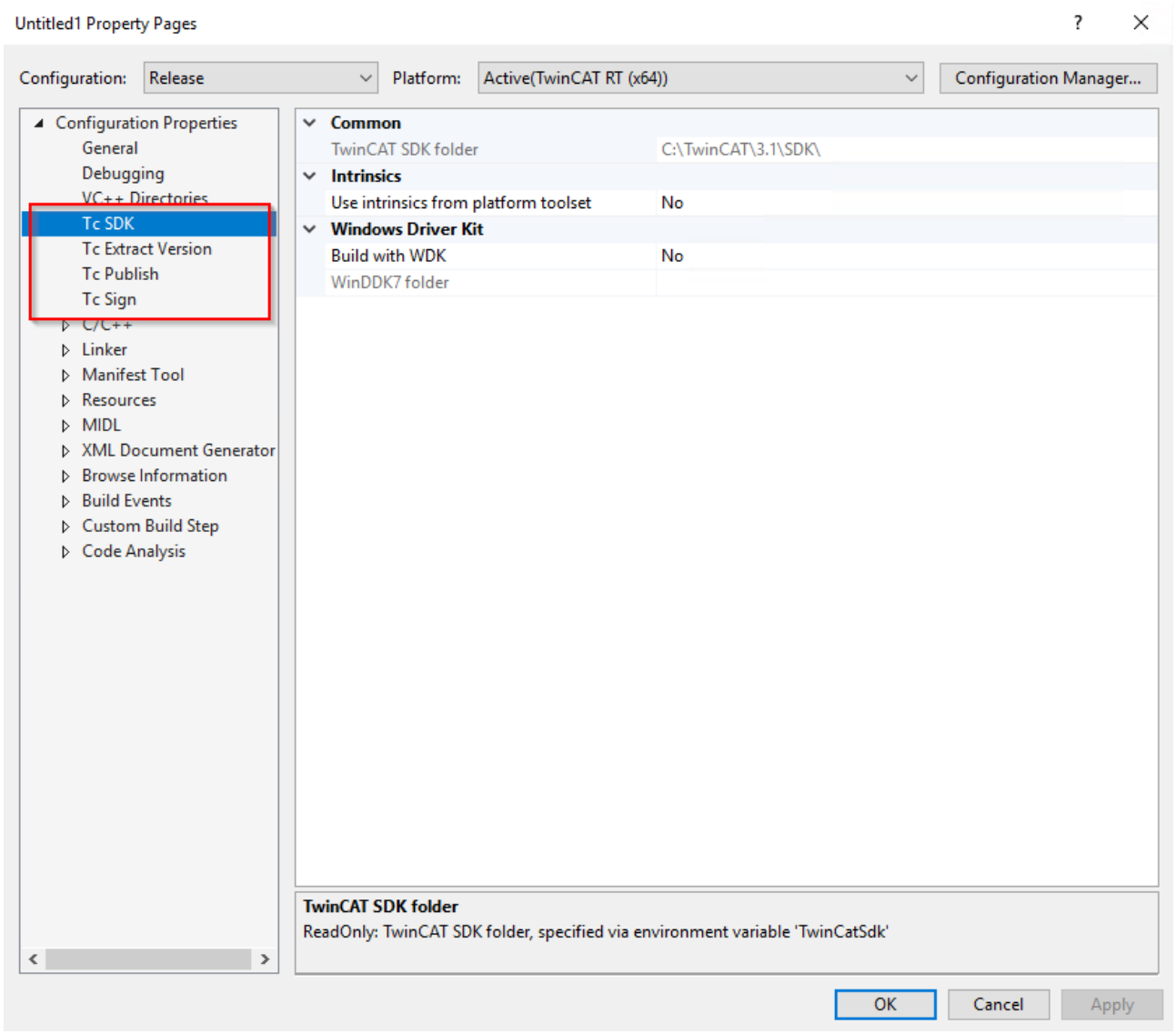
此处描述的功能从 TwinCAT 3.1 及以上版本可用。4024.0.

在 TwinCAT C++ 项目的项目属性中可以进行不同的设置。

右键单击“C++ 项目” -> **属性**，即可打开项目属性。

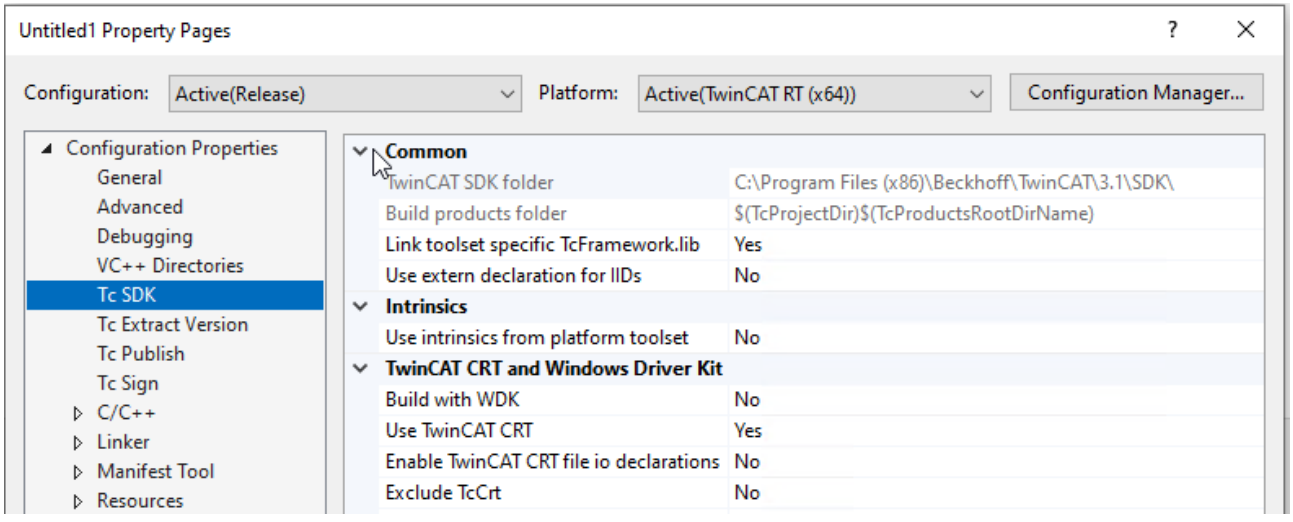


除了用于设置的 Visual Studio C++ 对话框外，还存在 TwinCAT 页面：



这些内容将在子页面中介绍。

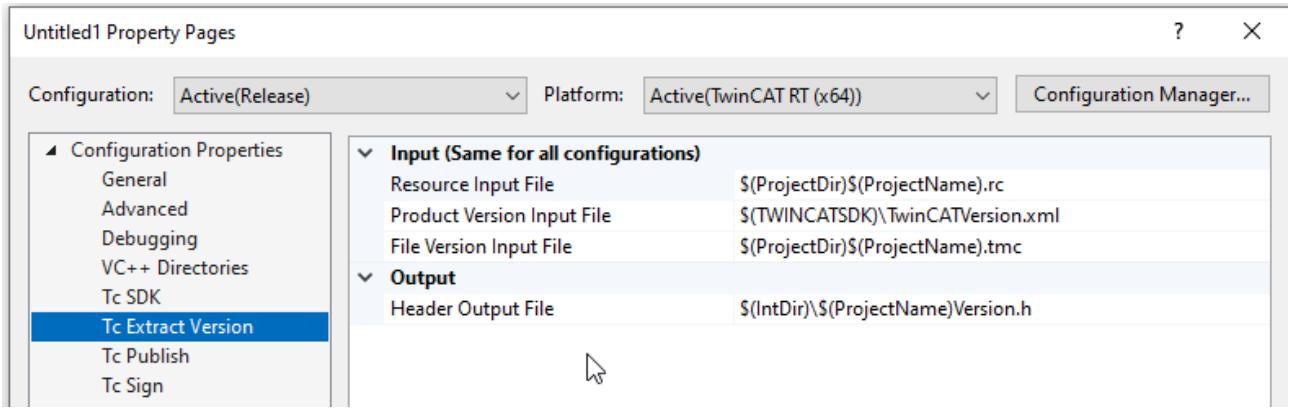
12.1.1 Tc SDK



TwinCAT SDK 设置

类别	字段	描述
通用	TwinCAT SDK 文件夹	提供 TwinCAT SDK 并显示环境变量 TWINCATSDK 值的文件文件夹。
内部函数	使用平台工具集的内部函数	应使用内部函数。

12.1.2 Tc 提取版本

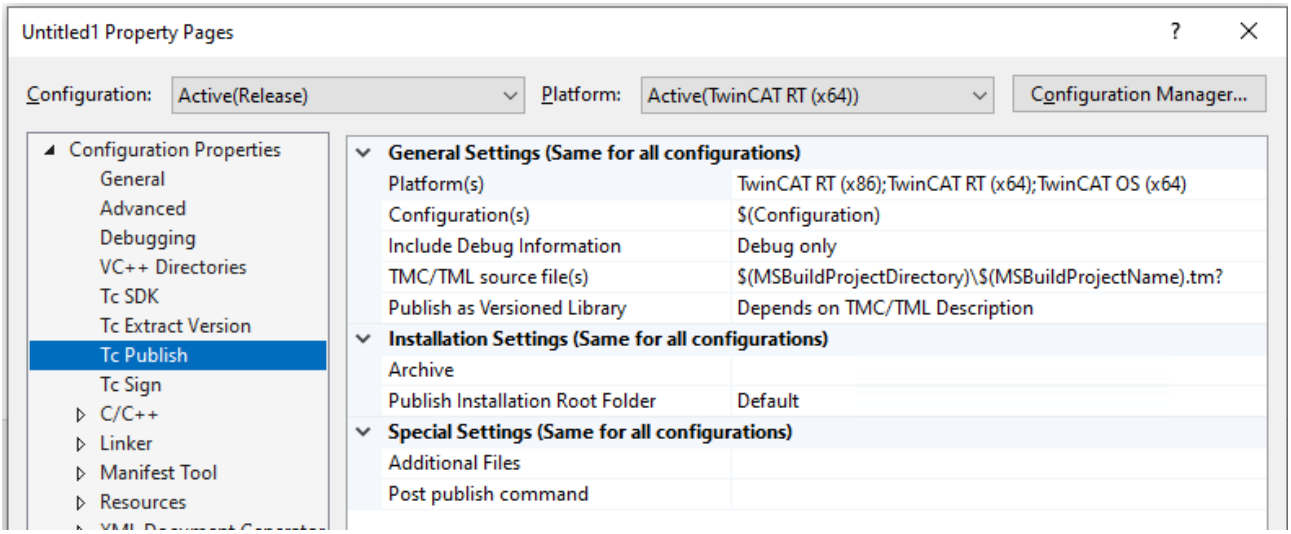


项目的版本信息可见报头文件，用于构建过程。

如果 .rc 文件包含版本信息，则会从中生成报头文件。在版本控制的 C++ 项目中，版本信息从 TMC 文件中读取，所生成报头文件中的宏用于 .rc 文件。

类别	字段	描述
提取版本	资源输入文件	要考虑的 .rc 文件。
	产品版本输入文件	TMC 文件，包含版本控制项目的产品版本。
	文件版本输入文件	TMC 文件，其中包含版本控制项目的文件版本。
	头文件输出文件	提供信息的报头文件。

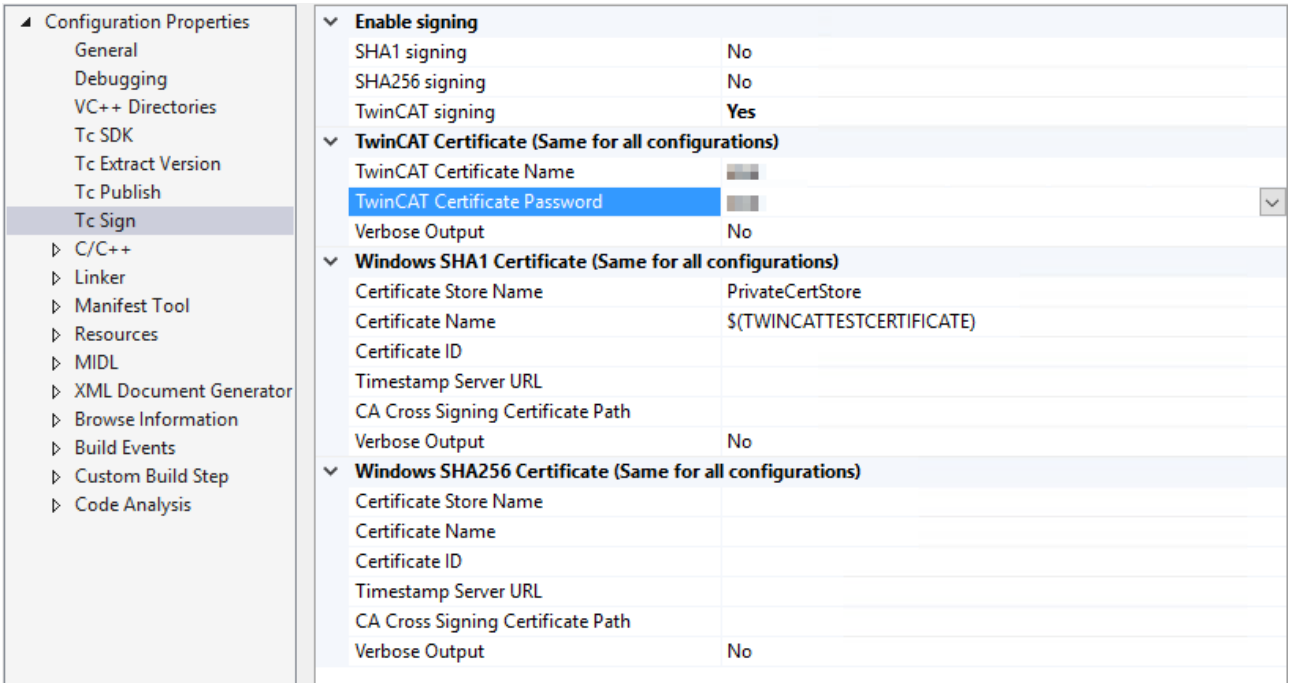
12.1.3 Tc 发布



有关发布模块的信息。

类别	字段	描述
通用设置	平台	哪些平台应在“发布”中构建?
	配置	选择是否构建调试/发布。
	包含调试信息	存储库中应提供哪些配置的调试符号 (PDB)?
	TMC/TML 源文件	项目中的 TMC/TML 文件，代表发布流程的起始文件。
	以版本控制存储库形式发布	是否应在存储库 [▶ 48]中发布?
安装设置	存档	存档的文件路径。允许使用扩展名 .zip（用于 ZIP 归档）和 .exe（用于自解压 ZIP 归档）。两者都包含另一个工程系统上的存储库（版本控制的 C++ 项目）或 CustomConfig/Modules（非版本控制的 C++ 驱动程序）的内容。
	发布安装根文件夹	如果选择我“none”，则不会在本地系统上进行安装。这些文件仅可见于 TWINCATSDK/_products/TcPublish。可以创建存档，手动将这些文件传输到另一个系统并安装到该系统。 如果选择“默认”，则会在本地系统上安装到存储库（版本控制的 C++ 项目）或 CustomConfig/Modules（非版本控制的 C++ 驱动程序）中。
特殊设置	附加文件	在“发布”流程中添加附加文件，这些文件在安装过程中存储在“部署”子目录中。
	发布后命令	在发布后执行命令，例如清理。

12.1.4 Tc 签名



TwinCAT 模块必须签名 [▶ 21]，可在此处进行配置。

类别	字段	描述
启用签名	SHA1 签名	是否应该进行操作系统签名（操作系统必要操作）？
	SHA256 签名	是否应该进行操作系统签名（操作系统必要操作）？
	TwinCAT 签名	是否应该使用 TwinCAT 用户证书进行签名？此操作对于 TwinCAT 加载器 [▶ 49]而言是必要的。
TwinCAT 证书 这些参数用于调试和发布等所有配置。	TwinCAT 证书名称	证书文件名称（目录：C:\ProgramData\Beckhoff\TwinCAT\3.1\CustomConfig\Certificates）。或者，也可将环境变量 TcSignTwinCatCertName 设置为证书文件的名称。
	TwinCAT 证书密码	用于保护 TwinCAT 用户证书的密码（以纯文本格式存储，必要时留空）。TcSignTool [▶ 54] 可用于不在项目中存储 TwinCAT 用户证书的密码，因为在项目中也会出现版本管理等问题。
	详细输出	签名时是否应输出扩展信息？
Windows 证书 (SHA1) 仅出于兼容性考虑而包含在内，请立即转用 TwinCAT 证书	证书存储区名称	操作系统证书管理器中证书存储区的名称。
	证书名称	证书存储区中的证书名称。
	证书 ID	证书的 ID。
	时间戳服务器 URL	签名时使用的时间戳服务器的 URL。由各种 CA 提供。
	CA 交叉签名证书路径	交叉签名证书的路径。
	详细输出	签名时是否应输出扩展信息？
Windows 证书 (SHA256) 仅出于兼容性考虑而包含在内，请立即转用 TwinCAT 证书	证书存储区名称	操作系统证书管理器中证书存储区的名称。
	证书名称	证书存储区中的证书名称。
	证书 ID	证书的 ID。
	时间戳服务器 URL	签名时使用的时间戳服务器的 URL，由 CA 提供。
	CA 交叉签名证书路径	交叉签名证书的路径。
	详细输出	签名时是否应输出扩展信息？

12.2 文件功能说明

在开发 TwinCAT C++ 模块期间，可以直接处理文件系统中的文件。这对了解系统如何运行或手动文件传输等特定使用场景都很有意义。

以下是与 C++ 模块相关的文件列表。

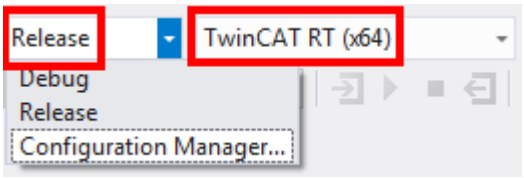
文件	描述	更多信息
工程/XAE		
*.sln	Visual Studio Solution 文件，承载 TwinCAT 和非 TwinCAT 项目	
*.tsproj	TwinCAT 项目，所有嵌套 TwinCAT 项目的集合，如 TwinCAT C++ 或 TwinCAT PLC 项目	
_Config/	文件夹包含属于 TwinCAT 项目的其他配置文件（*.xti）。	参见菜单“工具| 选项| TwinCAT| XAE-Environment| 文件设置”
_Deployment/	用于编译 TwinCAT C++ 驱动程序的文件夹	
*.tmc	TwinCAT 模块类文件（基于 XML）	请参见 TwinCAT 模块类编辑器 (TMC) [▶ 86]
*.rc	资源文件	请参见 设置版本/供应商信息 [▶ 213]
.vcxproj.	Visual Studio C++ 项目文件	
*ClassFactory.cpp/.h	该 TwinCAT 驱动程序的类工厂	
*Ctrl.cpp/.h	上传和删除 TwinCAT UM 平台的驱动程序	
*Driver.cpp/.h	上传和删除 TwinCAT RT 平台的驱动程序	
*Interfaces.cpp/.h	TwinCAT COM 接口类的声明	
*W32.cpp./.def/.idl		
*.cpp/.h	驱动程序中的每个 TwinCAT 模块都有一个 C++/报头文件。在此处插入用户代码。	
Resource.h	*.rc 文件必需	
TcPch.cpp/.h	用于创建预编译报头	
%TC_INSTALLPATH%\Repository\<Vendor>\<PrjName>\<Version>\<Platform>*.tmx	通过 TcLoader 加载的编译驱动程序。 TwinCAT 3.1.4024.x： C:\TwinCAT\3.1\Repository\C++ Module Vendor\Untitled1\0.0.0.1\TwinCAT RT *\Unititled1.tmx 从 TwinCAT 3.1.4026.x 起： C:\ProgramData\Beckhoff\TwinCAT\3.1\Repository\C++ Module Vendor\Untitled1\0.0.0.1\TwinCAT RT *\Unititled1.tmx	请参见 版本控制的 C++ 项目 [▶ 48]
%TC_INSTALLPATH%\CustomConfig\Modules*	已发布的 TwinCAT C++ 项目通常为 TwinCAT 3.1.4024.x： C:\TwinCAT\3.1\CustomConfig\Modules* 从 TwinCAT 3.1.4026.x 起： C:\ProgramFiles (x86)\Beckhoff\TwinCAT\3.1\Config\Modules*	请参见
运行时/XAR		
%TC_BOOTPRJPATH%\CurrentConfig*	当前配置设置 Windows： TwinCAT 3.1.4024.x： C:\TwinCAT\3.1\Boot 从 TwinCAT 3.1.4026.x 起： Windows：C:\ProgramData\Beckhoff\TwinCAT\3.1\Boot	

文件	描述	更多信息
	TwinCAT/BSD：/usr/local/etc/TwinCAT/3.1/Boot	
%TC_DRIVERAUTOINSTALLPATH% *.sys/pdb	经编译的平台特定驱动程序，通过操作系统加载。 Windows： TwinCAT 3.1.4024.x： C:\TwinCAT\3.1\Driver\AutoInstall（已由系统加载） 从 TwinCAT 3.1.4026.x 起： <not available> 请迁移至基于 TMX 的 C++ 的项目 TwinCAT/BSD®： <not available>	
%TC_INSTALLPATH% \Boot\Repository\<Vendor>\<PrjName>\<Version>*.tmx	经编译的平台特定驱动程序通过 TcLoader 加载。 Windows： TwinCAT 3.1.4024.x： C:\TwinCAT\3.1\Boot\Repository\C++ Module Vendor\Untitled1\0.0.0.1\Untitled1.tmx 从 TwinCAT 3.1.4026.x 起： C:\ProgramData\Beckhoff \TwinCAT\3.1\Boot\Repository\C++ Module Vendor\Untitled1\0.0.0.1\Untitled1.tmx TwinCAT/BSD：/usr/local/etc/TwinCAT/3.1/Boot\/Repository\C++ Module Vendor\Untitled1/0.0.0.1/Untitled1.tmx	
%TC_BOOTPRJPATH% \TM\OBJECTID.tmi	TwinCAT 模块实例文件 功能说明驱动程序的变量 文件名为 ObjectID.tmi Windows： TwinCAT 3.1.4024.x： C:\TwinCAT\3.1\Boot\TMI\OTCID.tmi 从 TwinCAT 3.1.4026.x 起： Windows：C:\ProgramData\Beckhoff \TwinCAT\3.1\Boot\TMI\OTCID.tmi TwinCAT/BSD：/usr/local/etc/TwinCAT/3.1/Boot/TMI/OTCID.tmi	
临时文件		
*.sdf	IntelliSense 数据库	
.suo/.v12.suo	用户特定文件和 Visual Studio 特定文件	
*.tsproj.bak	从 tsproj 自动生成备份文件	
ipch/	为预编译报头创建中间目录	

编译程序

此处说明的是在 TwinCAT 工程 XAE 中对 TwinCAT C++ 项目启动“构建”或“重新构建”的程序。例如，如果要整合公司特定的环境和构建流程，就必须考虑到这一点。

在“构建”或“重新构建”的情况下构建的配置取决于 Visual Studio 中的当前选择：



通过选择目标系统，可适当设置正确的目标架构（本例中为 TwinCAT RT (x64)）。

配置管理器允许对构建配置进行专用设置。

选择**生成**或**重新生成**（因此也是**激活配置**）时，会执行以下步骤：

1. 源代码位于相应的项目目录中。
2. 编译以架构特定方式在项目目录 _products\TwinCAT RT (x64)/Release\<ProjectName> 下生成。
3. 关联过程结束后，.tmx/.pdb 文件会存储在项目目录中。
4. 按下“Activate Configuration（激活配置）”按钮即可将 .tmx/.pdb 从工程存储库传输到目标系统（可能是本地副本）。

在项目上选择“发布”时，会为所选平台构建并将结果存储在工程存储库中：C:\ProgramData\Beckhoff\TwinCAT\3.1\Repository\<vendor>\<ProjectName>\<version>\TwinCAT RT (x64)\Release\（TMC 文件也会随之更新）。

12.3 在线更改

TwinCAT 3.1 版本 4024.0 及以上

此处描述的功能从 TwinCAT 3.1 及以上版本可用。4024.0.

TwinCAT 3.1 支持在运行时交换 C++ 模块，即无需中断实时程序。

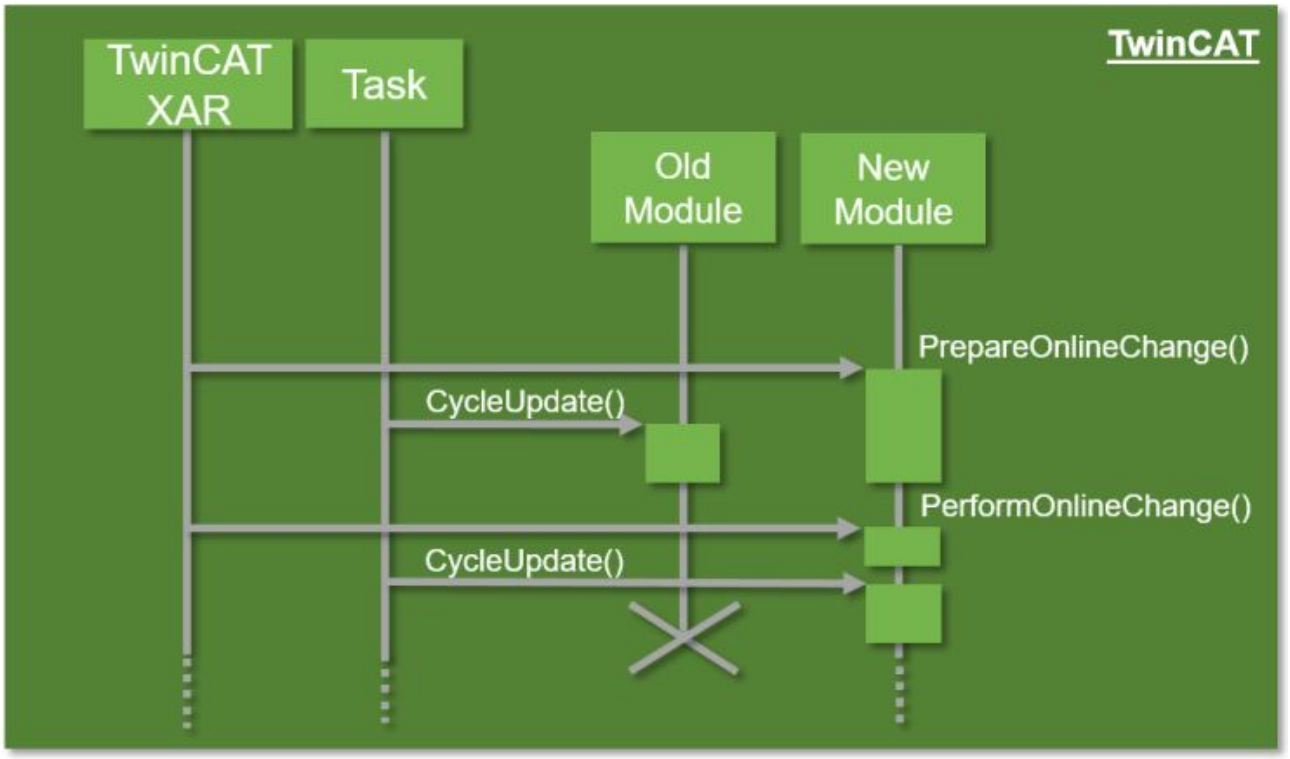
为此，已在目标系统中存储不同版本的 [▶ 48] TwinCAT 可执行文件 (TMX)。

对于 TMX 的所有模块实例，可通过工程发起版本之间的切换。

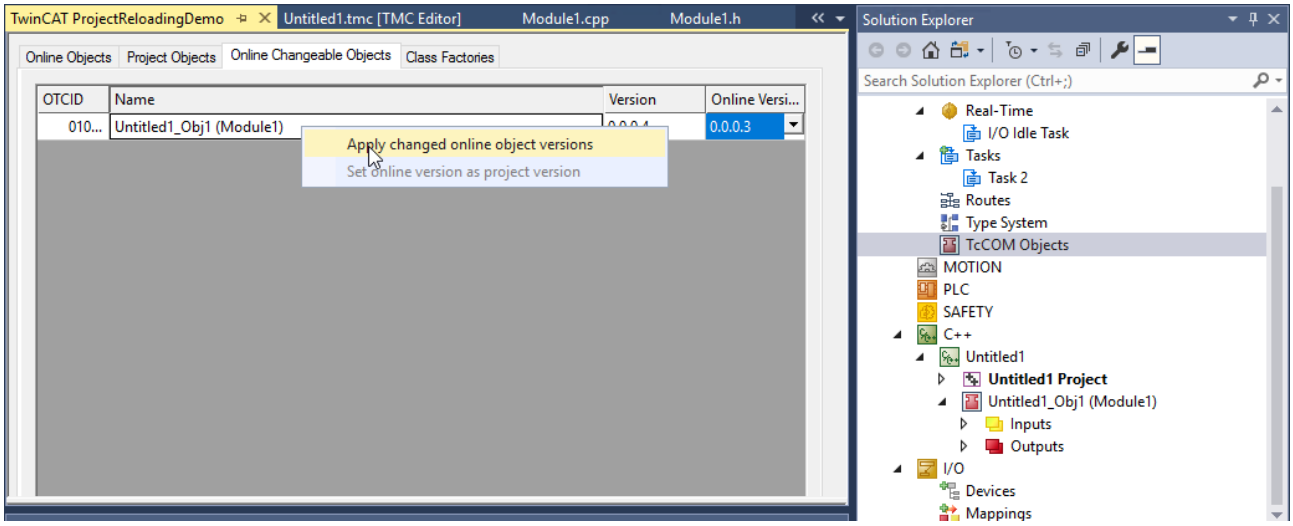
程序大致概述如下：

✓ TMX 中的“在线更改”功能模块

1. TwinCAT 将新模块实例化。任务仍在周期性调用旧模块。
2. TwinCAT 调用新模块的 ITcOnlineChange::PrepareOnlineChange()。
   该调用可以访问旧模块，并接受不会因模块循环调用而改变的数据，例如参数值。
3. TwinCAT 调用新模块的 ITcOnlineChange::PerformOnlineChange()。
   该调用可以访问旧模块，并接管此前经周期性调用已变更的数据。无周期性执行时触发。任务**不会**再次调用旧模块，但会调用新模块。PerformOnlineChange() 方法应使用尽可能少的计算时间，以便从一个任务循环切换为另一个任务循环。
4. 完成后，任务将循环调用新模块。



“在线更改”功能可通过工程中的此对话框执行。



因此，在处理“在线更改”时，需考虑以下几个方面：

- 项目必须提供版本控制功能。
- 这些模块的 DataArea 储存在 TcCOM 模块之外，可供模块使用。这意味着它们不需要考虑数据或 DataArea 中符号的映射。
- 模块的 DataArea 不得更改。
- 不得传递对内部数据结构的引用。务必通过 TcQueryInterface 检索的接口访问，因为这些引用会在“在线更改”功能运行时更新。

重新启动后，TwinCAT 将以模块的初始版本启动驱动程序。

12.4 限制条件

TwinCAT 3 C++ 模块 [▶ 35]会在 Windows 内核模式下执行。因此，开发人员必须注意某些限制：

- 内核模式下无法使用 Win32 API [▶ 150]

- 不得直接使用 Windows 内核模式 API。
  TwinCAT SDK 提供了受支持的功能。
- 不能使用用户模式库 (DLL)。（请参见 第三方库 [▶ 226]）
- 实时环境中的动态分配内存容量受路由器内存的限制（可在工程设计期间进行配置），请参见 内存分配 [▶ 150]。
- 支持 C++ 运行时库功能 (CRT) 的子集
- 不支持 C++ 异常机制。
- 不支持运行时类型信息 (RTTI [▶ 150])。
- 支持 STL 的子集（请参见 STL/容器 [▶ 207]）
- 支持通过 TwinCAT 实现 math.h 中的功能（请参见 数学函数 [▶ 204]）

TwinCAT 功能替代 Win32 API 功能

原 Win32 API 在 Windows 内核模式下不可用。因此，此处提供一份 Win32 API 的函数列表及其在 TwinCAT 中的对应函数：

Win32API	TwinCAT 功能
WinSock	TF6311 TCP/UDP 实时
消息框	跟踪 [▶ 208]
文件 I/O	请参见 接口 ITcFileAccess [▶ 163]、接口 ITcFileAccessAsync [▶ 170] 和 Sample19：同步文件访问 [▶ 288]、Sample20：FileIO-Write [▶ 289]、Sample20a：FileIO-Cyclic 读取/写入 [▶ 289]
同步	请参见 Sample11a：模块通信：C++ 模块调用 C++ 模块的方法 [▶ 283]
Visual C CRT	请参见 RtlR0.h

TwinCAT 中的 RTTI dynamic_cast 功能

TwinCAT 不支持 dynamic_cast<>。

相反，可以采用 TCOM 战略。定义一个继承自 ITcUnknown 的 ICustom 接口，包含从派生类调用的方法。基类 CMyBase 继承自ITcUnknown，并已实现该接口。类 CMyDerived 继承自 CMyBase 和 ICustom。该类重写 TcQueryInterface 方法，因此可以使用该方法来代替动态转换。

通过评估 TcQueryInterface 的返回值，还可实现 IsType() 函数的功能。

请参见 接口 ITcUnknown [▶ 186]。

12.5 内存分配

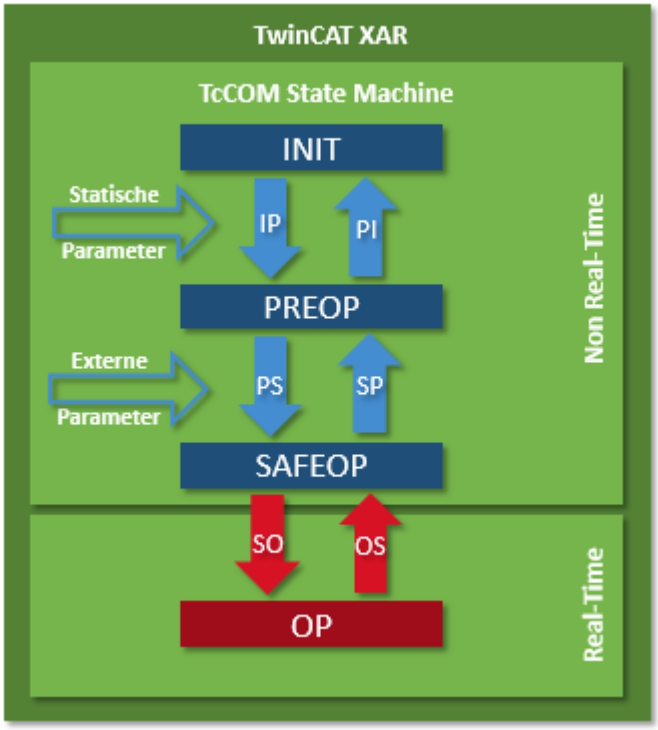
一般来说，我们建议借助模块类的成员变量来预留内存。对于在 TMC 编辑器中定义的数据区，将自动执行内存预留。
还可以动态分配和释放内存。

- 运算符新建/删除
- TcMemAllocate/TcMemFree

这种内存分配可用于状态机的转换 [▶ 43]或 OP 状态。

如果内存分配是在非实时环境中进行，则会在操作系统的非分页池中分配内存（图中蓝色部分）。在 TwinCAT 实时环境中，会在路由器内存中分配内存（图中红色部分）。

内存也可以在转换或 OP 状态中释放；我们建议务必在“对称”转换中释放内存，例如在 PS 中分配，在 SP 中释放内存。



在使用静态变量时，必须考虑一些特殊功能 [▶ 151]。

此外，还可以提供一些关于实现转换的说明：

- 转换 IP、PS 和 SP、PI 在非实时环境（如 Windows 内核模式）中执行。
- 然而，转换 SO、OS 是在 TwinCAT 实时任务环境（更准确地说：TcCOM 服务器任务）中执行的。
- 必须按照“全有或全无”的模式实现转换。这意味着，如果后续步骤失败，所有成功执行的步骤都必须撤销。
- 如果在 OP 状态下分配了内存，最迟必须在操作系统状态转换时释放。
- OS、SP、PI 转换应以确保成功的方式实现。不得返回错误代码。
- 转换应对称实现，即 OS 和 SO 互为逆操作，同理 PS <-> SP 和 IP <->PI 也应遵循此逻辑。
- 转换 SO 应在最后一步将模块登录到任务中（前提是存在 ITcCaller）
- 转换 OS 应在第一步从任务中注销该模块（前提是存在 ITcCaller）
- Semaphore
  - 可在非实时环境中创建和删除
  - 只允许在实时环境使用（挂起/发布）
- 多个模块可以在这些状态中相互等待。InitSeq（默认为“PSO”）用于指定模块实例应等待整个系统（以及所有其他模块）启动完成的状态节点。

12.6 静态变量

结合静态变量在 Windows 和实时环境之间分配内存

如果使用全局实例，请注意以下几点：

- 全局实例最多只能有 128 个。
  需要注意的是，全局实例也可由 TwinCAT 自身的模块使用。
- 如果超过该数量，则不会再调用析构函数。
- 从 TwinCAT 3.1 4026 开始，如果超过限值，工程“输出”中会显示消息：
  类工厂 <ProjectName>.tmx 关闭时检测到静态目标过多。超过 N 个项目的静态目标最大数量。可能会导致内存泄漏。

- 程序代码可以查询当前使用或可用的全局实例：

```
#include "CrtInit.h"
int overflow = cppGetNumberHandlerOverflow();
int free = TcCrtGetFreeNumberAtexitHandler();
```

- 例如，在实时环境中分配的内存必须在 OS 转换中释放，这样便不会通过析构函数发生这种情况。
- 当执行构造函数时，尚且无法访问 TwinCAT 系统资源，因为通过 Classfactory 与 TcCOM 对象服务器的连接尚未建立。下面您将了解如何处理这种情况。

本代码包含三个全局实例示例：

```
class MyClassA
{
public:
    MyClassA() {}
    ~MyClassA() {}
private:
    int v;
}

MyClassA A;

class MyClassB
{
    static MyClassA Value;
};

MyClassA& GetInstance()
{
    static MyClassA a;
    return a;
}
```

用于全局实例初始化的单例

为了延迟时间，也可以将全局或所谓的单例创建为本地静态对象。在使用全局类实例时，非常重要的一点是析构函数的调用时间要稍晚，此时与 TwinCAT 系统不再有连接，因此必须提前释放 TwinCAT 资源。

以下为代码示例，表示 TwinCAT 中的单例：

```
#include "TcInterfaces.h"
class VersionProvider : public ITcVersionProvider
{
public:
	DECLARE_IUNKNOWN_NODELETE()
	VersionProvider()
	{
		m_Version = { 1, 2, 3, 0};
	}
	virtual HRESULT TCOMAPI GetVersion(T_Version& version)
	{
		version = m_Version;
		return S_OK;
	};
private:
	T_Version m_Version;
};
BEGIN_INTERFACE_MAP(VersionProvider)
	INTERFACE_ENTRY(IID_ITcVersionProvider, ITcVersionProvider)
END_INTERFACE_MAP()
class MySingleton
{
public:
	static MySingleton& Instance(ITcVersionProvider* ip = NULL, bool bRelease=false)
	{
		static MySingleton g_instance(ip);
		if (bRelease) g_instance.Release();
		return g_instance;
	}
	T_Version GetVersion()
	{
	      T_Version v = {0, 0, 0, 0};
		if (m_spTcInst != NULL)
		{
			m_spTcInst->GetVersion(v);
		}
		return v;
	}
private:
	MySingleton(ITcVersionProvider* ip = NULL)
		: m_spTcInst(ip)
	{
	}
```

```
        void Release()
        {
              m_spTcInst = NULL;
        }
        ITcVersionProviderPtr m_spTcInst;
};
```

使用方法如下：

```
// actual instance which is an example for a TwinCAT resource
VersionProvider vp;
// Initialize singleton
MySingleton::Instance(&vp);
// use singleton to access information provided by TwinCAT resource
T_Version v1 = MySingleton::Instance().GetVersion();
Assert::IsTrue( v1.major == 1 && v1.minor == 2 && v1.build == 3 &&  v1.revision == 0);
// Deinitialize singleton and release reference to TwinCAT resource
MySingleton::Instance(NULL, true);
// use singleton after deinitialization which has to implement some default behaviour for this case
T_Version v2 = MySingleton::Instance().GetVersion();
Assert::IsTrue( v2.major == 0 && v2.minor == 0 && v2.build == 0 &&  v2.revision == 0);
```

12.7 多任务数据访问同步

当多个任务访问一组数据时，这些任务可能会同时访问相同的数据，具体取决于任务/实时配置。如果数据由至少一个任务写入，那么在更改期间或之后，数据可能会出现不一致的状态。为了防止出现这种情况，务必同步所有并发访问，以便每次只能有一个任务访问共享数据。

如果多个任务访问相同数据，且其中至少有一个访问会写入数据，则所有读取和写入访问必须同步。无论任务是在单核还是多核 CPU 上运行，这一点都适用。

⚠ 警告

由于数据访问不安全而导致的不一致和其他风险

如果不同步并发访问，则会存在数据记录不一致或无效的风险。根据在程序的后续阶段中使用数据的方式，这可能会导致错误的程序行为、意外的轴运动，甚至是程序突然进入停滞状态。根据不同的受控系统，可能会损坏设备和工件，或者危及人员健康和生命。

模块间通信 [▶ 45] 章已介绍 TcCOM 模块之间的一般同步选项。介绍的机构如下：

- 通过过程映像进行数据交换（IO 映射）。在此过程中，循环转换以及 TwinCAT 确保源和目标之间的数据同步。
- 通过可使用 CriticalSection 的接口的方法调用。
- ADS 以传输介质的形式传输数据，从而确保数据的同步性。
- 受 CriticalSection 保护的公用数据区。

然而，理想的情况是尽量避免同步需求。否则，最简单的方法通常是通过过程映像进行交换，将循环之间的数据从输出过程映像复制到输入过程映像，从而确保状态的一致性。
如果这样做还不够，并且需要从不同的环境访问数据，则可使用以下选项之一。

必须区分待访问数据的任务环境。在 TwinCAT Runtime 中，Windows 内核模式线程环境（简称 Windows 环境）与 TwinCAT 实时任务环境（简称 RT 环境）之间有区别。在 TwinCAT 模块初始化期间，IP、PS、SP 和 PI 转换在 Windows 环境中执行。SO 和 OS 转换在 RT 环境中执行。

SDK 可为这种情况提供适当的同步选项。然而，TcCOM 模块的设计应使其至少在 Windows 环境中是相互独立的，因此无需同步。如果需要，RT 环境中的转换可以使用 CriticalSection。

对于 CriticalSection 和 Semaphore，适用于在锁定的情况下，任务等待释放。在此期间，会释放核以执行其他任务。使用 CriticalSection 时，活动任务会继承等待任务的优先级（“优先级继承”）。

12.7.1 CriticalSection

CriticalSection 的实例是通过 TwinCAT 实时创建的。该实例既可在 Windows 环境中初始化，也可以在 RT 环境中初始化。CriticalSection 实例只能在 RT 环境中使用。

TwinCAT 实时会利用“优先级继承”来防止优先级较低的任务间接阻碍优先级较高的任务。

CCriticalSectionInstance

CCriticalSectionInstance 类提供了处理临界区（Critical Sections）的接口，并拥有所需内存。要创建临界区，该类需要 TwinCAT 实时实例的对象 ID（可通过 OID_TCRTIME_CTRL 获取），以及通过指向 ITComObjectServer 的指针对 TwinCAT 对象服务器的引用。

方法：

- CCriticalSectionInstance(OTCID oid=0, ITComObjectServer* ipSrv=NULL);
  默认构造函数，用于初始化“临界区提供程序”的对象 ID。如果指针指向对象服务器，那么也会对“临界区”进行初始化。
- ~CCriticalSectionInstance();
  析构函数会删除“临界区”实例。
- void SetOidCriticalSection(OTCID oid);
  设置“临界区提供程序”，该程序由 TwinCAT 实时系统提供，其对象 ID 可通过 OID_TCRTIME_CTRL 获取。
- bool HasOidCriticalSection();
  如果对象 ID 设置为非 0值，则返回 TRUE，否则返回 FALSE。此方法不会检查对象 ID 是否确实属于某个临界区提供程序。
- bool IsInitializedCriticalSection();
  如果“临界区”已成功初始化，则返回 TRUE。
- HRESULT CreateCriticalSection(ITComObjectServer* ipSrv);
  如果“临界区”已初始化，则会将其删除并初始化新的“临界区”。如果成功，则返回 S_OK。出错情况通过返回错误代码来表示：

ADS_E_INVALIDPARM	无效“临界区提供程序”
ADS_E_NOINTERFACE	对象 ID 设置为非“临界区提供程序”的引用，即 ITcRTime 未执行。
E_FAIL	“临界区提供程序”出现内部错误。

- HRESULT CreateCriticalSection(OTCID oid, ITComObjectServer* ipSrv);
  初始化“临界区”。如果再次使用此方法，则必须调用 DeleteCriticalSection()。返回值与 CreateCriticalSection (ITComObjectServer* ipSrv) 中的值相同；
- HRESULT DeleteCriticalSection(); “临界区”已删除。始终返回 S_OK。
- HRESULT EnterCriticalSection(); 阻塞，直到释放“临界区”。
- HRESULT LeaveCriticalSection();
  退出“临界区”，从而再次释放。
  如果调用者并非所有者，则返回 MAKE_RTOS_HRESULT(OS_CS_ERR)。

EnterCriticalSection() 和 LeaveCriticalSection() 必须在 RT 环境中调用，否则会返回以下返回值：

ADS_E_INVALIDCONTEXT	如果在 RT 环境之外输入“临界区”，则返回值。未输入“临界区”。

如果在使用这些方法时未将“临界区”初始化，则会返回 S_OK，而无需进一步操作。所有其他方法既可在 RT 环境中使用，也可在 Windows 环境中使用。

在 CCriticalSectionInstance 类中，同一项任务可以多次调用 EnterCriticalSection()方法。 然后，必须同样频繁地调用 Leave()。

CriticalSection 允许嵌套调用，每次 EnterCriticalSection() 调用都需要相关的 LeaveCriticalSection() 调用。最后一次调用 LeaveCritcalSection() 时，会释放“临界区”。

示例：

本示例 [▶ 283]展示的是如何利用 临界区（Critical Section）防止通过接口方法调用并发访问日期。

临界区并发

CCriticalSectionInstanceConcurrent 类允许并发访问。如果多项任务均可读取要保护的数据，但在写入访问时必须防止所有其他访问，则可以使用这种方法。

方法：

- CCriticalSectionInstanceConcurrent(UINT concurrent, OTCID oid=0, ITComObjectServer* ipSrv=NULL)；参数“并发”定义了可通过 CsEnterCriticalSectionConcurrent() 同时输入“临界区”的任务数。若“并（concurrent）”参数取值为 1，临界区并发（Concurrent）变体的工作方式与普通临界区相同。
  不允许使用 0 值。此值会导致临界区初始化失败。
- ~CCriticalSectionInstanceConcurrent()；析构函数隐式删除“临界区”
- void SetOidCriticalSection(OTCID oid)；设置“临界区提供程序”，该程序由 TwinCAT 实时系统提供，其对象 ID 可通过 OID_TCRTIME_CTRL 获取。
- bool HasOidCriticalSection()；如果对象 ID 设置为 非 0 值，则返回 TRUE，否则返回 FALSE。此方法不会检查对象 ID 是否确实属于某个临界区提供程序。
- bool IsInitializedCriticalSection()；如果“临界区”已成功初始化，则返回 TRUE。
- HRESULT CreateCriticalSection(ITComObjectServer* ipSrv)；如果“临界区”已初始化，则会将其删除并初始化新的“临界区”。如果成功，则返回 S_OK。出错情况通过返回错误代码来表示：

ADS_E_INVALIDPARM	无效“临界区提供程序”
ADS_E_NOINTERFACE	对象 ID 设置为非“临界区提供程序”的引用，即 ITcRTime 未执行。
E_FAIL	“临界区提供程序”出现内部错误。

- HRESULT CreateCriticalSection(OTCID oid, ITComObjectServer* ipSrv)；
  初始化“临界区”。如果再次使用此方法，则必须调用 DeleteCriticalSection()。返回值与 CreateCriticalSection (ITComObjectServer* ipSrv) 中的值相同；
- HRESULT DeleteCriticalSection()；“临界区”已删除。始终返回 S_OK。
- HRESULT EnterCriticalSection()；阻塞，直到释放“临界区”并可单独访问。
- HRESULT LeaveCriticalSection()；退出“临界区”，从而再次释放。如果调用者并非所有者，则返回 MAKE_RTOS_HRESULT(OS_CS_ERR)。
- HRESULT EnterCriticalSectionConcurrent()；与其他任务并行输入“临界区”。并行访问的次数由构造函数中的参数“concurrent”定义。当达到该最大值时，会阻止调用，直到其中一个并行访问结束。

必须在 RT 环境中调用 EnterCriticalSection()、EnterCriticalSectionConcurrent() 和 LeaveCriticalSection()，否则会返回以下返回值：

ADS_E_INVALIDCONTEXT	如果在 RT 环境之外输入“临界区”，则返回值。未输入“临界区”。

如果在使用这些方法时未将“临界区”初始化，则会返回 S_OK，而无需进一步操作。所有其他方法既可在实时（RT）环境中使用，也可在 Windows 环境中使用。

对于 CCriticalSectionInstanceConcurrent 类，如果初始化期间concurrent参数大于 1，则不允许同一任务多次调用 EnterCriticalSection() 方法。

12.7.2 Semaphore

信号量（Semaphore）用于同步对有限资源的访问。还可用于实现通知事件。Semaphore 通过 CSemaphoreInstance 类由 TwinCAT 实时提供。

CSemaphoreInstance

CSemaphoreInstance 类提供了处理 Semaphore 的接口。要创建 Semaphore，该实例需要 TwinCAT 实时实例的：对象 ID（可通过 OID_TCRTIME_CTRL 获取），以及指向 ITComObjectServer 的指针（用于关联 TwinCAT 对象服务器）。

方法：

- HRESULT SemCreate(WORD nCntInit, OTCID oid, ITComObjectServer* ipSrv)；以 nCntInit 作为可用资源的初始值，创建 Semaphore。如果成功，则返回 S_OK；如果无法生成 Semaphore，则返回 E_FAIL。
- HRESULT SemDelete(); SemDelete() 会删除 Semaphore。返回 S_OK。
- HRESULT SemPost() SemPost() 用于增加可用资源的数量。如果有任务在等待 Semaphore，则会释放优先级最高的任务，即可以继续执行。如果成功，则返回 S_OK。可能的错误代码：

MAKE_RTOS_HRESULT(51)	Semaphore 溢出。例如，如果内部内存不足。

- HRESULT SemPend(OSTICKS nTimeout)；减少可用资源的数量；如果没有可用资源，任务将在此处阻塞。
  SemPend() 等待一个 Semaphore，直到可用为止。可在 OSTICKS 中指定超时，因此该时间与处理器相关。可以使用 ITcRTime 接口来计算。在示例 TcSemaphoreSample 中，已通过 TimeoutMsToTicks 方法实现此功能。nTimeout 的特殊值：

RTIME_NOWAIT (-1)	如果 Semaphore 可用，则方法立即返回 S_OK。 如果没有可用资源，方法会返回超时。
RTIME_ENDLESSWAIT (0)	该方法将无限期等待信号量变为可用状态

如果成功获取 Semaphore，则该方法会返回 S_OK；否则：

MAKE_RTOS_HRESULT(10)	表示超时。如果 nTimeout 设置为 RTIME_NOWAIT，则此 Semaphore 不可用。

示例：

Sample24：信号量 [▶ 294]

12.7.3 FIFO 模板类

报头文件 Fifo.h 可提供不同的 FIFO 模板类。所有类都能识别内存何时已满，并在删除元素之前不接受新元素。元素类型可以是原始数据类型（“普通旧数据类型”，POD）。在 FIFO 缓冲区中无法存储带有虚基表的元素类型。允许使用指针（不论类型）。

CFiFoListBase

CFifoListBase 类是 FIFO 的基本实现。该模板类有一个 StorageType 实例，负责存储一种类型的元素。它使用 SyncType 实例来保持对所含存储和数据的同步访问。

存在多种保护机制：

- CNoSync/SyncObects.h：FIFO 内没有访问保护。可以使用外部 CriticalSection。
- CSpinLock/TcSpinlock.h：如上所述，通过自旋锁进行保护。

Add()、Remove() 和 Clear() 方法有相应的 _Unprotected_ 选项，可与 Lock() 和 Unlock() 方法一起使用，以实现 Add() 和 Delete() 操作的同步序列。

方法：

- CFiFoListBase()；构造函数。
- Lock() 和 Unlock()；传递给 SyncType 同步实例。
- Clear()、Clear_Unprotected()；传递给 StorageType 实例。
- Add()、Add_Unprotected()；添加元素。
- AddBlock()；添加多个元素。若待添加数据块大小超过剩余内存，仅添加与剩余内存匹配的元素数量。返回实际添加元素的数量。
- Remove()、Remove_Unprotected()；从 FIFO 中移除元素。如果没有元素，则返回 FALSE，否则返回 TRUE。

- RemoveBlock()；从 FIFO 中删除多个元素。如果存储的元素较少，则只会移除存储的元素。返回已移除元素的数量。
- GetEntryXBeforeHead()；从头部获取第 x 个元素之前的元素，而无需移除该元素。例如，传入参数 0 时读取第一个元素。否则，参数必须大于 0。如果元素不存在，则返回 FALSE，否则返回 TRUE。
- Count()；返回存储元素的数量。
- Size()；返回内存的容量，即内存可容纳元素的最大数量。
- FreeCount()；返回已添加到内存中的元素数量。
- GetNextEntry()；从表头开始遍历所有元素。

CFiFoList

CFiFoList 类基于 CFiFoListBase，使用 CFiFoListStorageStatic 作为存储类型，并与同步类型 CSyncDefault 结合使用。
该实现未新增额外方法，需要将元素类型和内存大小作为模板参数。

CFiFoListDyn

CFiFoListDyn 类使用动态存储类型 CFifoListStorageDynamic 和同步类型 CSyncDefault。
该实现未新增额外方法，需要将元素类型和内存大小作为模板参数。

Handover3Buffer

CHandover3Buffer 类用于在任务间传递元素。内存中包含三个给定类型的元素。写入任务始终可以写入元素。读取任务在有元素写入后可立即读取。且仅读取完整写入的元素。读取元素后不会将其删除。通过索引对内存进行读取和写入访问，由 InterlockedExchangeR0 函数实现同步，因此可从 Windows 和 实时（RT）环境进行访问。

方法：

- CHandover3Buffer()；构造函数。
- Write()；将元素写入内存。如果内存处于无效状态，则返回 FALSE。否则返回 TRUE。
- Read()；从内存中读取元素。如果内存为空或处于无效状态，则返回 FALSE。否则返回 TRUE。

CHandoverFifo

CHandoverFifo 类还用于在任务之间交换元素。内存的大小始终为 2 的幂次方。可以通过模板参数 DEPTH_EXP 进行设置。默认值为 3，因此对应内存容量为 8 个元素。

方法：

- Insert()/Add()；通过指针或引用将元素写入内存。如果没有可用内存，则返回 FALSE，否则返回 TRUE。
- Remove()；通过指针或引用从内存中删除并返回元素。如果没有可用元素，则返回 FALSE，否则返回 TRUE。

示例：
TcHandoverSample.tszip

12.8 接口

用户开发的模块与 TwinCAT 3 系统之间的交互有多种接口可供选择。在这些页面上有详细说明（API 层级）。

名称	描述
ITcUnknown [▶ 186]	ITcUnknown 定义引用计数以及获取更具体接口引用的查询方法。
ITComObject [▶ 175]	每个 TwinCAT 模块均实现 ITComObject 接口。
ITcCyclic [▶ 160]	接口由 TwinCAT 模块实现，每个任务周期调用一次。
ITcCyclicCaller [▶ 161]	用于将模块的 ItcCyclic 接口登录到 TwinCAT 任务中或从 TwinCAT 任务中注销的接口。
ITcFileAccess [▶ 163]	用于访问文件系统的接口
ITcFileAccessAsync [▶ 170]	异步访问文件操作。
ITcPostCyclic [▶ 180]	接口由 TwinCAT 模块实现，输出更新后每个任务周期调用一次。
ITcPostCyclicCaller [▶ 158]	用于将模块的 ITcPostCyclic 接口登录到 TwinCAT 任务或从 TwinCAT 任务中注销的接口。
ITcIoCyclic [▶ 171]	该接口由 TwinCAT 模块实现，这些模块会在任务周期内的输入更新和输出更新过程中调用。
ITcIoCyclicCaller [▶ 172]	用于将模块的 ITcIoCyclic 接口登录到 TwinCAT 任务中或从 TwinCAT 任务中注销的接口。
ITcRTimeTask [▶ 181]	查询 TwinCAT 任务扩展信息。
ITcTask [▶ 182]	查询 TwinCAT 任务的时间戳和任务特定信息。
ITcTaskNotification [▶ 185]	如果在上一个循环中已超过周期时间，则执行回调。

TwinCAT SDK

TwinCAT SDK 包含许多功能组件，可在 C:\TwinCAT\3.1\sdk\Include 中查阅。

- 此处提供的是 TcCOM 框架（特别是 TcInterfaces.h 和 TcServices.h）。
- 任务和数据区访问权限通过 TcIoInterfaces.h 提供。
- SDK 功能是数学函数 [▶ 204]。
- STL [▶ 207] 的子集。
- TwinCAT Runtime RtlR0.h
- ADS 通讯 [▶ 188]方式
- 名称以“Os”开头的类/功能不得用于实时环境。

12.8.1 接口 ITcPostCyclicCaller

用于在 TwinCAT 任务中登录或注销模块的 ITcPostCyclic 接口的接口。

语法

```
TCOM_DECL_INTERFACE("03000026-0000-0000-e000-000000000064", ITcCyclicCaller)
struct __declspec(novtable) ITcPostCyclicCaller : public ITcUnknown Ca
```

要求包括：`TcIoInterfaces.h`

方法

名称	描述
AddPostModule [▶ 159]	实现 ITcPostCyclic 接口的登录模块。
RemovePostModule [▶ 159]	注销先前登录的模块 ITcPostCyclic 接口。

ITcPostCyclicCaller 接口由 TwinCAT 任务实现。模块使用该接口将其 ITcPostCyclic 接口登录到任务中，通常作为 SafeOP 到 OP 转换的最后一个初始化步骤。登录后会调用模块实例的 PostCycleUpdate() 方法。该接口还用于注销模块，使其不再被任务调用。

12.8.1.1 方法 ITcPostCyclicCaller:AddPostModule

将模块的 ITcPostCyclic 接口登录到周期性调用者，如 TwinCAT 任务。

语法

```
virtual HRESULT TCOMAPI
AddPostModule(STcPostCyclicEntry* pEntry, ITcPostCyclic* ipMod, ULONG_PTR
context=0, ULONG sortOrder=0)=0;
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

如果循环任务调用器（即 TwinCAT 任务）未处于 OP 状态，则返回错误 ADSERR_DEVICE_INVALIDSTATE。

参数

名称	类型	描述
pEntry	STcPostCyclicEntry	[in] 指向插入循环任务调用器内部列表的列表项指针；另见说明。
ipMod	ITcPostCyclic	[in] 循环任务调用器使用的接口指针。
环境	ULONG_PTR	[optional] 每次调用时传输给 ITcPostCyclic::PostCyclicUpdate() 方法的环境值。
sortOrder	ULONG	[optional] 如果不同模块实例由同一个循环任务调用器执行，则排序顺序可用于控制执行顺序。

功能说明

TwinCAT 模块类使用“智能指针”来引用 ITcPostCyclicCallerPtr 类型的循环任务调用器。任务的对象 ID 存储在该“智能指针”中，可通过 TwinCAT 对象服务器获取任务的引用。此外，“智能指针”类已包含一个列表项。因此，“智能指针”可用作 AddPostModule 方法的第一个参数。

以下代码示例说明了如何登录 ITcPostCyclicCaller 接口。

```
RESULT hr =
S_OK;

if ( m_spPostCyclicCaller.HasOID() ) {

if ( SUCCEEDED_DBG(hr =
m_spSrv->TcQuerySmartObjectInterface(m_spPostCyclicCaller)) )
{

if ( FAILED(hr =
m_spPostCyclicCaller->AddPostModule(m_spPostCyclicCaller,
THIS_CAST(ITcPostCyclic))) ) {

m_spPostCyclicCaller = NULL;

}

}

}
```

12.8.1.2 方法 ITcPostCyclicCaller:RemovePostModule

删除模块实例，使其不再被循环任务调用器调用。

语法

```
virtual HRESULT TCOMAPI
RemovePostModule(STcPostCyclicEntry* pEntry)=0;
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

如果条目不在内部列表中，则该方法会返回 E_FAIL。

参数

名称	类型	描述
pEntry	STcPostCyclicEntry	指的是要从循环任务调用器内部列表中删除的列表项。

与 AddPostModule() 方法类似，如果要从循环任务调用器中删除模块实例，那么“智能指针”会作为列表项用于循环任务调用器。

“智能指针”的声明和使用情况：

ITcPostCyclicCallerInfoPtr m_spPostCyclicCaller;

```
if (
m_spPostCyclicCaller ) {
m_spPostCyclicCaller->RemovePostModule(m_spPostCyclicCaller);

}

m_spPostCyclicCaller = NULL;
```

12.8.2 返回值

ITc 接口方法通常会返回 HRESULT。

在 ITc 接口的情况下，可以返回以下返回值：

名称	HRESULT
S_OK	0x0000 0000
S_FALSE	0x0000 0001
E_NOTIMPL	0x8000 4001
E_NOINTERFACE	0x8000 4002
E_POINTER	0x8000 4003
E_ABORT	0x8000 4004
E_FAIL	0x8000 4005
E_UNEXPECTED	0x8000 FFFF
E_ACCESSDENIED	0x8007 0005
E_HANDLE	0x8007 0006
E_OUTOFMEMORY	0x8007 000E
E_INVALIDARG	0x8007 0057

此外，ADS 返回代码 [▶ 321]有可能作为 HRESULT 返回。这些宏也可在 SDK 中使用，例如 ADS 错误代码 ADSERR_DEVICE_BUSY 的宏称为 ADS_E_BUSY。

12.8.3 接口 ITcCyclic

ITcCyclic 接口由 TwinCAT 模块实现，每个任务周期调用一次。

语法

```
TCOM_DECL_INTERFACE("03000010-0000-0000-e000-000000000064", ITcCyclic)
struct __declspec(novtable) ITcCyclic : public ITcUnknown
```

要求包括：TcIoInterfaces.h

方法

名称	描述
CycleUpdate [▶ 161]	在每个任务周期调用一次，前提是接口已登录至循环任务调用器。

ITcCyclic 接口由 TwinCAT 模块实现。当模块自行登录到任务中时，通常作为从 SafeOP 转换到 OP 的最后一个初始化步骤，该接口会传递至 ITcCyclicCaller::AddModule() 方法。登录后会调用模块实例的 CycleUpdate() 方法。

12.8.3.1 方法 ITcCyclic：CycleUpdate

CycleUpdate 方法通常在接口登录后由 TwinCAT 任务调用。

语法

```
HRESULT TCOMAPI CycleUpdate(ITcTask* ipTask, ITcUnknown* ipCaller, ULONG_PTR context)
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

目前，该返回值已被 TwinCAT 任务忽略。

参数

名称	类型	描述
ipTask	ITcTask	指的是当前任务环境。
ipCaller	ITcUnknown	指的是调用实例。
环境	ULONG_PTR	环境包含传递至 ITcCyclicCaller::AddModule() 方法的值。

描述

在任务周期内，调用所有已注册模块实例的 InputUpdate() 方法后，才会调用 CycleUpdate() 方法。因此，必须使用这种方法来实现循环处理。

12.8.4 接口 ITcCyclicCaller

用于在 TwinCAT 任务中登录或注销模块的 ITcCyclic 接口。

语法

```
TCOM_DECL_INTERFACE("0300001E-0000-0000-e000-000000000064", ITcCyclicCaller)
struct __declspec(novtable) ITcCyclicCaller : public ITcUnknown
```

要求包括：TcIoInterfaces.h

方法

名称	描述
AddModule [▶ 162]	实现 ITcCyclic 接口的登录模块。
RemoveModule [▶ 162]	注销先前登录的模块的 ITcCyclic 接口。

ITcCyclicCaller 接口由 TwinCAT 任务实现。模块使用该接口将其 ITcCyclic 接口登录到任务中，通常作为从 SafeOP 转换到 OP 的最后一个初始化步骤。登录后会调用模块实例的 CycleUpdate() 方法。该接口还用于注销模块，使其不再被任务调用。

12.8.4.1 方法 ITcCyclicCaller:AddModule

将模块的 ITcCyclic 接口登录到循环任务调用器，如 TwinCAT 任务。

语法

```
virtual HRESULT TCOMAPI
AddModule(STcCyclicEntry* pEntry, ITcCyclic* ipMod, ULONG_PTR
context=0, ULONG sortOrder=0)=0;
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

如果循环任务调用器（即 TwinCAT 任务）未处于 OP 状态，则返回错误 ADSERR_DEVICE_INVALIDSTATE。

参数

名称	类型	描述
pEntry	STcCyclicEntry	[in] 指向插入循环任务调用器内部列表的列表项指针；另见说明 [▶ 162]。
ipMod	ITcCyclic	[in] 循环任务调用器使用的接口指针。
环境	ULONG_PTR	[optional] 每次调用 ITcCyclic::CycleUpdate() 方法时传递至该方法的环境值。
sortOrder	ULONG	[optional] 如果不同模块实例由同一个循环任务调用器执行，则排序顺序可用于控制执行顺序。

功能说明

TwinCAT 模块类通常使用“智能指针”来引用循环任务调用器类型 ITcCyclicCallerPtr。任务的对象 ID 存储在该“智能指针”中，可通过 TwinCAT 对象服务器获取任务的引用。此外，“智能指针”类已包含一个列表项。因此，“智能指针”可用作 AddModule 方法的第一个参数。

以下示例代码显示的是 ITcCyclicCaller 接口的登录信息。

```
RESULT hr =
S_OK;

if ( m_spCyclicCaller.HasOID() ) {

if ( SUCCEEDED_DBG(hr =
m_spSrv->TcQuerySmartObjectInterface(m_spCyclicCaller)) )
{

if ( FAILED(hr =
m_spCyclicCaller->AddModule(m_spCyclicCaller,
THIS_CAST(ITcCyclic))) ) {

m_spCyclicCaller = NULL;

}

}

}
```

12.8.4.2 方法 ITcCyclicCaller:RemoveModule

删除模块实例，使其不再被循环任务调用器调用。

语法

```
virtual HRESULT TCOMAPI
RemoveModule(STcCyclicEntry* pEntry)=0;
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

参数

名称	类型	描述
pEntry	STcCyclicEntry	指的是要从循环任务调用器内部列表中删除的列表项。

如果条目不在内部列表中，则该方法会返回 E_FAIL。

与 AddModule() 方法类似，如果要从循环任务调用器中删除模块实例，那么“智能指针”会作为列表项用于循环任务调用器。

“智能指针”的声明和使用情况：

ITcCyclicCallerInfoPtr m_spCyclicCaller;

```
if (
m_spCyclicCaller ) {
m_spCyclicCaller->RemoveModule(m_spCyclicCaller);
}
m_spCyclicCaller = NULL;
```

12.8.5 接口 ITcFileAccess

用于从 TwinCAT C++ 模块访问文件系统的接口

语法

```
TCOM_DECL_INTERFACE("742A7429-DA6D-4C1D-80D8-398D8C1F1747", ITcFileAccess) __declspec(novtable)
ITcFileAccess: public ITcUnknown
```

要求包括：`TcFileAccessInterfaces.h`

方法

名称	描述
FileOpen [▶ 164]	打开文件。
FileClose [▶ 164]	关闭文件。
FileRead [▶ 165]	从文件中读取。
FileWrite [▶ 165]	写入文件。
FileSeek [▶ 165]	指定文件中的位置，以便进一步操作。
FileTell [▶ 166]	查询文件中当前设定的位置。
FileRename [▶ 166]	重命名文件。
FileDelete [▶ 167]	删除文件。
FileGetStatus [▶ 167]	获取文件的状态。
FileFindFirst [▶ 168]	搜索文件，第一次迭代。
FileFindNext [▶ 168]	搜索文件，下一次迭代。
FileFindClose [▶ 169]	关闭文件搜索。
MkDir [▶ 169]	创建目录。
RmDir [▶ 169]	删除目录。

ITcFileAccess 接口用于访问文件系统中的文件。
由于所提供的方法会导致阻塞，因此该接口不得在 CycleUpdate()/实时环境中使用。派生接口 ITcFileAccessAsync [▶ 170] 增加一个 Check() 方法，可用于替代该方法。

请参见 Sample20a：FileIO-Cyclic 读取/写入 [▶ 289]。

该接口通过 CID_TcFileAccess 模块类实现。

12.8.5.1 方法 ITcFileAccess:FileOpen

打开文件。

语法

```
virtual HRESULT TCOMAPI FileOpen(PCCH szFileName, TcFileAccessMode AccessMode, PTcFileHandle
phFile);
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

如果超时（5 秒）已过，便会出现一个特别值得关注的错误代码 ADSERR_DEVICE_TIMEOUT。

参数

名称	类型	描述
szFileName	PCCH	[in] 要打开的文件名。
AccessMode	TcFileAccessMode	[in] 用于访问文件的方法；请参见 TcFileAccessServices.h。
phFile	TcFileHandle	[out] 返回的文件句柄。

AccessModes 的使用方法如下：

```
typedef enum TcFileAccessMode
{
amRead = 0x00000001,
amWrite = 0x00000002,
amAppend = 0x00000004,
amPlus = 0x00000008,
amBinary = 0x00000010,
amReadBinary = 0x00000011,
amWriteBinary = 0x00000012,
amText = 0x00000020,
amReadText = 0x00000021,
amWriteText = 0x00000022,
amEnsureDirectory = 0x00000040,
amReadBinaryED = 0x00000051,
amWriteBinaryED = 0x00000052,
amReadTextED = 0x00000061,
amWriteTextED = 0x00000062,
amEncryption = 0x00000080,
amReadBinEnc = 0x00000091,
amWriteBinEnc = 0x00000092,
amReadBinEncED = 0x000000d1,
amWriteBinEncED = 0x000000d2,
} TcFileAccessMode, *PTcFileAccessMode;
```

12.8.5.2 方法 ITcFileAccess:FileClose

关闭文件。

语法

```
virtual HRESULT TCOMAPI FileClose(PTcFileHandle phFile);
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

如果超时（5 秒）已过，便会出现一个特别值得关注的错误代码 ADSERR_DEVICE_TIMEOUT。

参数

名称	类型	描述
phFile	TcFileHandle	[out] 要关闭的文件的返回文件句柄

12.8.5.3 方法 ITcFileAccess:FileRead

从文件中读取数据。

语法

```
virtual HRESULT TCOMAPI
FileRead(TcFileHandle hFile, PVOID pData, UINT cbData, PUINT pcbRead);
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

如果超时（5 秒）已过，便会出现一个特别值得关注的错误代码 ADSERR_DEVICE_TIMEOUT。

参数

名称	类型	描述
hFile	TcFileHandle	[in] 指的是之前打开的文件。
pData	PVOID	[out] 要读取数据的存储位置。
cbData	PVOID	[in] 要读取数据的最大大小（pData 后面的内存大小）。
pcbRead	PUINT	[out] 读取数据的大小。

12.8.5.4 方法 ITcFileAccess:FileWrite

将数据写入文件。

语法

```
virtual HRESULT TCOMAPI
FileWrite(TcFileHandle hFile, PCVOID pData, UINT cbData, PUINT pcbWrite);
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

如果超时（5 秒）已过，便会出现一个特别值得关注的错误代码 ADSERR_DEVICE_TIMEOUT。

参数

名称	类型	描述
hFile	TcFileHandle	[in] 指的是之前打开的文件。
pData	PVOID	[in] 要写入数据的存储位置。
cbData	PVOID	[in] 要写入数据的大小（pData 后面的内存大小）。
pcbRead	PUINT	[out] 写入数据的大小。

12.8.5.5 方法 ITcFileAccess:FileSeek

指定文件中的位置，以便进一步操作。

语法

```
virtual HRESULT TCOMAPI FileSeek(TcFileHandle hFile, UINT uiPos);
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

如果超时（5 秒）已过，便会出现一个特别值得关注的错误代码 ADSERR_DEVICE_TIMEOUT。

参数

名称	类型	描述
hFile	TcFileHandle	[in] 指的是之前打开的文件。
uiPos	UINT	[in] 要进行设置的位置。

12.8.5.6 方法 ITcFileAccess:FileTell

查询文件中当前设定的位置。

语法

```
virtual HRESULT TCOMAPI FileTell(TcFileHandle hFile, PUINT puiPos);
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

如果超时（5 秒）已过，便会出现一个特别值得关注的错误代码 ADSERR_DEVICE_TIMEOUT。

参数

名称	类型	描述
hFile	TcFileHandle	[in] 指的是之前打开的文件。
puiPos	PUINT	[out] 要返回位置的存储位置。

12.8.5.7 方法 ITcFileAccess:FileRename

重命名文件。

语法

```
virtual HRESULT TCOMAPI FileRename(PCCH szOldName, PCCH szNewName);
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

如果超时（5 秒）已过，便会出现一个特别值得关注的错误代码 ADSERR_DEVICE_TIMEOUT。

参数

szOldName	PCCH	[in] 要更改的文件名。
szNewName	PCCH	[in] 新文件名。

12.8.5.8 方法 ITcFileAccess:FileDelete

删除文件。

语法

```
virtual HRESULT TCOMAPI FileDelete(PCCH szFileName);
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

如果超时（5 秒）已过，便会出现一个特别值得关注的错误代码 ADSERR_DEVICE_TIMEOUT。

参数

名称	类型	描述
szFileName	PCCH	[in] 要删除的文件的名称。

12.8.5.9 方法 ITcFileAccess:FileGetStatus

查询文件状态。

语法

```
virtual HRESULT TCOMAPI FileGetStatus(PCCH szFileName, PTcFileStatus pFileStatus));
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

如果超时（5 秒）已过，便会出现一个特别值得关注的错误代码 ADSERR_DEVICE_TIMEOUT。

参数

名称	类型	描述
szFileName	PCCH	[in] 有关文件的名称。
pFileStatus	PTcFileStatus	[out] 文件状态，参见 TcFileAccessServices.h 。

功能说明

此方法可查询给定文件名的状态信息。

其中包括以下信息：

```
typedef struct TcFileStatus
{
union
{
ULONGLONG ulFileSize;
struct
{
ULONG ulFileSizeLow;
ULONG ulFileSizeHigh;
};
};
ULONGLONG ulCreateTime;
ULONGLONG ulModifiedTime;
ULONGLONG ulReadTime;
DWORD dwAttribute;
DWORD wReserved0;
} TcFileStatus, *PTcFileStatus;
```

12.8.5.10 方法 ITcFileAccess:FileFindFirst

用于浏览目录中文件的选项。

语法

```
virtual HRESULT TCOMAPI FileFindFirst (PCCH szFileName, PTcFileFindData pFileFindData ,
PTcFileFindHandle phFileFind);
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

如果超时（5 秒）已过，便会出现一个特别值得关注的错误代码 ADSERR_DEVICE_TIMEOUT。

参数

名称	类型	描述
szFileName	PCCH	[in] 所查找文件的目录或路径和名称。文件名可以包含星号 (*) 或问号 (?) 等占位符。
pFileFindData	PTcFileFindData	[out] 第一个文件的说明，参见 TcFileAccessServices.h
phFileFind	PTcFileFindHandle	[out] 用于使用 FileFindNext 进一步搜索的句柄。

功能说明

该方法首先搜索指定目录中的文件。该方法允许访问找到的第一个文件的 PTcFileFindData，其中包含以下信息：

```
typedef struct TcFileFindData
{
TcFileHandle hFile;
DWORD dwFileAttributes;
ULONGLONG ui64CreationTime;
ULONGLONG ui64LastAccessTime;
ULONGLONG ui64LastWriteTime;
DWORD dwFileSizeHigh;
DWORD dwFileSizeLow;
DWORD dwReserved1;
DWORD dwReserved2;
CHAR cFileName[260];
CHAR cAlternateFileName[14];
WORD wReserved0;
} TcFileFindData, *PTcFileFindData;
```

12.8.5.11 方法 ITcFileAccess:FileFindNext

继续扫描目录中的文件。

语法

```
virtual HRESULT TCOMAPI FileFindNext (TcFileFindHandle hFileFind, PTcFileFindData pFileFindData);
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

如果超时（5 秒）已过，便会出现一个特别值得关注的错误代码 ADSERR_DEVICE_TIMEOUT。

参数

名称	类型	描述
hFileFind	PTcFileFindHandle	[in] 用于使用 FileFindNext 进一步搜索的句柄。
pFileFindData	PTcFileFindData	[out] 下一个文件的说明。与 TcFileAccessServices.h 比较

功能说明

该方法会搜索目录中的下一个文件。该方法允许访问所找到文件的 PTcFileFindData，其中包含以下信息：

```
typedef struct TcFileFindData
{
TcFileHandle hFile;
DWORD dwFileAttributes;
ULONGLONG ui64CreationTime;
ULONGLONG ui64LastAccessTime;
ULONGLONG ui64LastWriteTime;
DWORD dwFileSizeHigh;
DWORD dwFileSizeLow;
DWORD dwReserved1;
DWORD dwReserved2;
CHAR cFileName[260];
CHAR cAlternateFileName[14];
WORD wReserved0;
} TcFileFindData, *PTcFileFindData;
```

12.8.5.12 方法 ITcFileAccess:FileFindClose

关闭目录中的文件搜索。

语法

```
virtual HRESULT TCOMAPI FileFindClose (TcFileFindHandle hFileFind);
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

如果超时（5 秒）已过，便会出现一个特别值得关注的错误代码 ADSERR_DEVICE_TIMEOUT。

参数

名称	类型	描述
hFileFind	PTcFileFindHandle	[in] 用于退出搜索的句柄。

12.8.5.13 方法 ITcFileAccess:MkDir

在文件系统中创建目录。

语法

```
virtual HRESULT TCOMAPI MkDir(PCCH szDir);
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

如果超时（5 秒）已过，便会出现一个特别值得关注的错误代码 ADSERR_DEVICE_TIMEOUT。

参数

名称	类型	描述
szDir	PCCH	[in] 要创建的目录。

12.8.5.14 方法 ITcFileAccess:RmDir

从文件系统中删除目录。

语法

```
virtual HRESULT TCOMAPI RmDir(PCCH szDir);
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

如果超时（5 秒）已过，便会出现一个特别值得关注的错误代码 ADSERR_DEVICE_TIMEOUT。

参数

名称	类型	描述
szDir	PCCH	[in] 要删除的目录。

12.8.6 接口 ITcFileAccessAsync

异步访问文件操作。该接口扩展了 ITcFileAccess [▶ 163]。

语法

```
TCOM_DECL_INTERFACE("C04AC244-C126-466E-982E-93EC571F2277", ITcFileAccessAsync) struct
__declspec(novtable) ITcFileAccessAsync: public ITcFileAccess
```

要求包括：`TcFileAccessInterfaces.h`

属性

名称	描述
PID_TcFileAccessAsyncSegmentSize	传输到系统服务的网段大小。
PID_TcFileAccessAsyncTimeoutMs	以 [ms] 为单位设置超时。
PID_TcFileAccessAsyncNetId(Str)	待通信的系统服务的 NetId。

方法

名称	描述
检查 [▶ 170]	检查先前调用的文件操作的状态。

可以从类 ID 为 CID_TcFileAccessAsync 的模块实例中获取接口。
使用异步接口时，如果查询已成功提交但尚未完成，则从同步变量继承的接口方法会返回 ADS_E_PENDING。
如果仍在处理前一个请求的同时收到调用，则返回错误代码 ADS_E_BUSY。

模块参数说明：

- PID_TcFileAccessAsyncAdsProvider：提供 ADS 接口的任务的对象 ID。
- PID_TcFileAccessAsyncNetId/PID_TcFileAccessAsyncNetIdStr：用于文件访问的系统服务的 AmsNetId。“Str”变量将 AmsNetId 作为字符串。请使用一个。
- PID_TcFileAccessAsyncTimeoutMs：文件访问超时。
- PID_TcFileAccessAsyncSegmentSize：对文件的读写访问使用此段大小进行分片处理。

请参见 Sample20a：FileIO-Cyclic 读取/写入 [▶ 289]。

12.8.6.1 方法 ITcFileAccessAsync::Check()

检查先前调用的文件操作的状态。

语法

```
virtual HRESULT TCOMAPI Check();
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

特别有趣的错误代码为

- ADSERR_DEVICE_TIMEOUT，前提是超时（5 秒）已过。
- ADSERR_DEVICE_PENDING，前提是文件操作未完成。

参数

无

12.8.7 接口 ITcIoCyclic

该接口由 TwinCAT 模块实现，这些模块会在任务周期内的输入更新和输出更新过程中调用。

语法

```
TCOM_DECL_INTERFACE("03000011-0000-0000-e000-000000000064", ITcIoCyclic)
struct __declspec(novtable) ITcIoCyclic : public ITcUnknown
```

要求包括：TcIoInterfaces.h

方法

名称	描述
InputUpdate [▶ 171]	如果接口登录周期性 I/O 调用器，则在任务周期开始时调用。
OutputUpdate [▶ 172]	如果接口登录周期性 I/O 调用器，则在任务周期结束时调用。

ITcIoCyclic 可用于实现作为现场总线驱动程序或 I/O 滤波模块的 TwinCAT 模块。

当模块登录任务时，通常作为从 SafeOP 转换到 OP 的最后一个初始化步骤，该接口会被传递给方法 ITcIoCyclicCaller::AddIoDriver。登录后，模块实例的 InputUpdate() 和 OutputUpdate() 方法会在每个任务周期分别被调用一次。

12.8.7.1 方法 ITcIoCyclic:InputUpdate

如果接口登录周期性 I/O 调用器，则在任务周期开始时调用。

语法

```
virtual HRESULT TCOMAPI InputUpdate(ITcTask* ipTask,
ITcUnknown* ipCaller, DWORD dwStateIn, ULONG_PTR context = 0)=0;
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

参数

名称	类型	描述
ipTask	ITcTask	指的是当前任务环境。
ipCaller	ITcUnknown	指的是调用实例。
dwStateIn	DWORD	预留供将来扩展；目前该值始终为 0。
环境	ULONG_PTR	环境包含传递至 ITcCyclicCaller::AddIoDriver() 方法的值。

功能说明

在任务循环中，首先会调用所有已注册模块实例的 InputUpdate() 方法。因此，必须使用此方法更新模块输入源类型的数据区。

12.8.7.2 方法 ITcIoCyclic:OutputUpdate

如果接口登录周期性 I/O 调用器，则在任务周期结束时调用。

语法

```
virtual HRESULT TCOMAPI OutputUpdate(ITcTask* ipTask, ITcUnknown* ipCaller,
PDWORD pdwStateOut = NULL, ULONG_PTR context = 0)=0;
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

参数

名称	类型	描述
ipTask	ITcTask	指的是当前任务环境。
ipCaller	ITcUnknown	指的是调用实例。
pdwStateOut	DWORD	[out] 预留供将来扩展，目前忽略返回值。
环境	ULONG_PTR	环境包含传递至 ITcCyclicCaller::AddIoDriver() 方法的值。

功能说明

在任务循环中，所有已注册的模块实例都会调用 OutputUpdate() 方法。因此，必须使用此方法更新模块输出目标类型的数据区。

12.8.8 接口 ITcIoCyclicCaller

用于在 TwinCAT 任务中登录或注销模块的 ITcIoCyclic 接口的接口。

语法

```
TCOM_DECL_INTERFACE("0300001F-0000-0000-e000-000000000064", ITcIoCyclicCaller)
struct __declspec(novtable) ITcIoCyclicCaller : public ITcUnknown
```

要求包括：`TcIoInterfaces.h`

方法

名称	描述
AddIoDriver [▶ 173]	实现 ITcIoCyclic 接口的登录模块。
RemoveIoDriver [▶ 173]	注销模块先前登录的 ITcIoCyclic 接口。

ITcIoCyclicCaller 接口由 TwinCAT 任务实现。模块使用该接口将其 ITcIoCyclic 接口登录到任务中，通常作为 SafeOP 到 OP 转换的最后一个初始化步骤。登录后会调用模块实例的 CycleUpdate() 方法。该接口还用于注销模块，使其不再被任务调用。

12.8.8.1 方法 ITcIoCyclicCaller:AddIoDriver

将模块的 ITcIoCyclic 接口登录到周期 I/O 调用器中，如 TwinCAT 任务。

语法

```
virtual HRESULT TCOMAPI AddIoDriver(STcIoCyclicEntry*
pEntry, ITcIoCyclic* ipDrv, ULONG_PTR context=0, ULONG sortOrder=0)=0;
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

参数

名称	类型	描述
pEntry	STcIoCyclicEntry	指向插入周期 I/O 调用器内部列表的列表项的指针；另见说明 [▶ 162]。
ipDrv	ITcIoCyclic	[in] 周期 I/O 调用器使用的接口指针。
环境	ULONG_PTR	[optional] 每次调用时传输给 ITcIoCyclic::InputUpdate() 和 ITcIoCyclic::OutputUpdate 方法的环境值。
sortOrder	ULONG	[optional] 如果不同模块实例由同一个循环任务调用器执行，则排序顺序可用于控制执行顺序。

功能说明

TwinCAT 模块类通常使用“智能指针”来引用 ITcIoCyclicCallerPtr 类型的周期 I/O 调用器。周期 I/O 调用器的对象 ID 存储在该“智能指针”中，可通过 TwinCAT 对象服务器获取引用。此外，“智能指针”类已包含一个列表项。因此，“智能指针”可用作 AddIoDriver 方法的第一个参数。

以下代码示例说明了如何登录 ITcIoCyclicCaller 接口。

```
HRESULT hr = S_OK;
if ( m_spIoCyclicCaller.HasOID() )
{
if ( SUCCEEDED_DBG(hr = m_spSrv->TcQuerySmartObjectInterface(m_spIoCyclicCaller))
)
{
if ( FAILED(hr = m_spIoCyclicCaller->AddIoDriver(m_spIoCyclicCaller,
THIS_CAST(ITcIoCyclic))) )
{
m_spIoCyclicCaller = NULL;
}
}
}
```

12.8.8.2 方法 ITcIoCyclicCaller:RemoveIoDriver

删除模块实例，使其不再被周期 I/O 调用器调用。

语法

```
virtual HRESULT TCOMAPI
RemoveIoDriver(STcIoCyclicEntry* pEntry)=0;
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

如果条目不在内部列表中，则该方法会返回 E_FAIL。

参数

名称	类型	描述
pEntry	STcIoCyclicEntry	指的是要从周期 I/O 调用器内部列表中删除的列表项。

与 AddIoDriver() 方法类似，如果要将模块实例从周期 I/O 调用器中删除，那么“智能指针”会作为列表项用于周期 I/O 调用器。

“智能指针”的声明和使用情况：

```
ITcIoCyclicCallerInfoPtr
m_spIoCyclicCaller;
if ( m_spIoCyclicCaller )
{
m_spIoCyclicCaller->RemoveIoDriver(m_spIoCyclicCaller);
}
m_spCyclicCaller = NULL;
```

12.8.9 ITComOnlineChange 接口

ITComOnlineChange 接口用于执行模块的“在线更改”功能。

语法

```
TCOM_DECL_INTERFACE ("D28A8CD2-5477-4B75-AF0F-998841AF9E44", ITComOnlineChange)
```

方法

名称	描述
PrepareOnlineChange [▶ 174]	TwinCAT 调用此方法为“在线更改”做准备。
PerformOnlineChange [▶ 175]	TwinCAT 调用此方法来执行“在线更改”。

要使模块能够进行“在线更改”，就必须实现该接口。此外，必须在版本控制的 C++ 项目中创建此类模块。

- 以下 [▶ 148]是该程序的一般说明。
- 现有模块也可遵循此程序：在线更改 [▶ 148]。

12.8.9.1 方法 ITComOnlineChange:PrepareOnlineChange

TwinCAT 调用此方法为“在线更改”做准备。

在后台异步运行，访问现有对象时必须考虑到这一点。
准备工作应包括所有已经可以执行的操作。

语法

```
virtual HRESULT TCOMAPI PrepareOnlineChange(ITComObject* ipOldObj, TmcInstData* pOldInfo) = 0;
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

参数

名称	类型	描述
ipOldObj	ITComObject*	要交换的现有对象的引用。
pOldInfo	TmcInstData*	对现有对象信息的引用。

通过 ipOldObj，可以传输现有对象的数据，以便应用。

例如：

```
ULONG nData = sizeof(m_Parameter);
PVOID pData = &m_Parameter;
ipOldObj->TcGetObjPara(PID_Module1Parameter, nData, pData);
```

12.8.9.2 方法 ITComOnlineChange:PerformOnlineChange

TwinCAT 调用此方法来执行“在线更改”。

这被称为阻塞。因此，只需要很短的时间。

语法

```
virtual HRESULT TCOMAPI PerformOnlineChange(ITComObject* ipOldObj, TmcInstData* pOldInfo) = 0;
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

参数

名称	类型	描述
ipOldObj	ITComObject*	要交换的现有对象的引用。
pOldInfo	TmcInstData*	对现有对象信息的引用。

通过 ipOldObj，可以传输现有对象的数据，以便应用。

例如：

```
ULONG nData = sizeof(m_Parameter);
PVOID pData = &m_Parameter;
ipOldObj->TcGetObjPara(PID_Module1Parameter, nData, pData);
```

12.8.10 ITComObject 接口

每个 TwinCAT 模块均实现 ITComObject 接口。它提供了基本功能。

语法

```
TCOM_DECL_INTERFACE("00000012-0000-0000-e000-000000000064", ITComObject)
struct __declspec(novtable) ITComObject: public ITcUnknown
```

方法

名称	描述
TcGetObjectId(OTCID& objId) [▶ 176]	使用给定的 OTCID 引用保存对象 ID。
TcSetObjectId [▶ 176]	将对象的对象 ID 设置为给定的 OTCID。
TcGetObjectName [▶ 176]	将对象名称保存至指定长度的缓冲区中。
TcSetObjectName [▶ 177]	将对象的对象名称设置为给定的 CHAR*。
TcSetObjState [▶ 177]	初始化转换到预定义的状态。
TcGetObjState [▶ 177]	查询对象的当前状态。
TcGetObjPara [▶ 178]	查询以 PTCID 标识的对象参数。
TcSetObjPara [▶ 178]	设置用 PTCID 标识的对象参数。
TcGetParentObjId [▶ 178]	借助给定的 OTCID 引用保存父对象 ID。
TcSetParentObjId [▶ 179]	将父对象 ID 设置为给定的 OTCID。

每个 TwinCAT 模块均实现 ITComObject 接口。它提供与状态机有关的功能，以及与 TwinCAT 系统之间信息传输的功能。

12.8.10.1 ITcComObject:TcGetObjectId 方法

该方法借助给定的 OTCID 引用保存对象 ID。

语法

```
HRESULT TcGetObjectId( OTCID& objId )
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

参数

名称	类型	描述
objId	OTCID&	引用 OTCID 值。

12.8.10.2 ITcComObject:TcSetObjectId 方法

TcSetObjectId 方法将对象的对象 ID 设置为给定的 OTCID。

语法

```
HRESULT TcSetObjectId( OTCID objId )
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

目前，该返回值已被 TwinCAT 任务忽略。

参数

名称	类型	描述
objId	OTCID	要设置的 OTCID，表示 ID 更改成功。

12.8.10.3 ITcComObject:TcGetObjectName 方法

TcGetObjectName 方法以给定的长度将对象名称保存在缓冲区中。

语法

```
HRESULT TcGetObjectName( CHAR* objName, ULONG nameLen );
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

参数

名称	类型	描述
objName	CHAR*	要设置的名称。
nameLen	ULONG	要写入的最大长度。

12.8.10.4 ITcComObject:TcSetObjectName 方法

TcSetObjectName 方法将对象的对象名称设置为给定的 CHAR*。

语法

```
HRESULT TcSetObjectName( CHAR* objName )
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

参数

名称	类型	描述
objName	CHAR*	要设置的对象名称。

12.8.10.5 ITcComObject:TcSetObjState 方法

TcSetObjState 方法会将转换初始化到给定的状态。

语法

```
HRESULT TcSetObjState(TCOM_STATE state, ITComObjectServer* ipSrv, PTComInitDataHdr pInitData);
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

参数

名称	类型	描述
state	TCOM_STATE	显示新状态。
ipSrv	ITComObjectServer*	处理对象的 ObjServer。
pInitData	PTComInitDataHdr	指向参数列表（可选）；关于遍历该列表，请参见 IMPLEMENT_ITCOMOBJECT_EVALUATE_INITDATA 宏。

12.8.10.6 ITcComObject:TcGetObjState 方法

TcGetObjState 方法可查询对象的当前状态。

语法

```
HRESULT TcGetObjState(TCOM_STATE* pState)
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

参数

名称	类型	描述
pState	TCOM_STATE*	状态指针。

12.8.10.7 ITcComObject:TcGetObjPara 方法

TcGetObjPara 方法查询通过 PTCID 标识的对象参数。

语法

```
HRESULT TcGetObjPara(PTCID pid, ULONG& nData, PVOID& pData, PTCGP pgp=0)
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

参数

名称	类型	描述
pid	PTCID	对象参数的参数 ID。
nData	ULONG&	最大数据长度。
pData	PVOID&	指向数据的指针。
pgp	PTCGP	预留作将来扩展，传递 NULL。

12.8.10.8 ITcComObject:TcSetObjPara 方法

TcSetObjPara 方法会设置一个通过 PTCID 识别的对象参数。

语法

```
HRESULT TcSetObjPara(PTCID pid, ULONG nData, PVOID pData, PTCGP pgp=0)
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

参数

名称	类型	描述
pid	PTCID	对象参数的参数 ID。
nData	ULONG	最大数据长度。
pData	PVOID	指向数据的指针。
pgp	PTCGP	预留作将来扩展，传递 NULL。

12.8.10.9 ITcComObject:TcGetParentObjId 方法

TcGetParentObjId 方法借助给定的 OTCID 引用保存父对象 ID。

语法

```
HRESULT TcGetParentObjId( OTCID& objId )
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

参数

名称	类型	描述
objId	OTCID&	引用 OTCID 值。

12.8.10.10 ITcComObject:TcSetParentObjId 方法

TcSetParentObjId 方法借助给定的 OTCID 引用设置父对象 ID。

语法

```
HRESULT TcSetParentObjId( OTCID objId )
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

目前，该返回值已被 TwinCAT 任务忽略。

参数

名称	类型	描述
objId	OTCID	引用 OTCID 值。

12.8.11 ITComObject 接口（C++ 便捷封装版）

每个 TwinCAT 模块均实现 ITComObject 接口。它提供了基本功能。

TwinCAT C++ 可提供附加功能，这些功能并非通过接口直接定义。

语法

要求包括：`TcInterfaces.h`

方法

名称	描述
OTCID TcGetObjectId [▶ 179]	查询对象 ID。
TcTryToReleaseOpState [▶ 180]	释放资源；必须执行。

还有其他方法，在此不一一列举。

该功能由模块向导作为标准配置提供。

12.8.11.1 TcGetObjectId 方法

该方法可查询对象 ID。

语法

```
OTCID TcGetObjectId(void)
```

返回值

名称	描述
OTCID	返回对象的 OTCID。

12.8.11.2 TcTryToReleaseOpState 方法

方法 TcTryToReleaseOpState 释放资源（如数据指针），以便为退出 OP 状态做准备。

语法

```
BOOL TcTryToReleaseOpState(void)
```

返回值

返回 TRUE 或 FALSE。

功能说明

必须执行 TcTryToReleaseOpState 方法，以消除模块实例之间可能存在的相互依赖关系。

只有当其他模块持有对该模块的引用时，才会对其调用。

12.8.12 接口 ITcPostCyclic

该接口由 TwinCAT 模块实现，在输出更新后的每个任务周期调用一次（相当于 PLC 的属性 TcCallAfterOutputUpdate）。

语法

```
TCOM_DECL_INTERFACE("03000025-0000-0000-e000-000000000064", ITcPostCyclic)
struct __declspec(novtable) ITcPostCyclic : public ITcUnknown
```

要求包括：`TcIoInterfaces.h`

方法

名称	描述
PostCycleUpdate [▶ 180]	在输出更新后的每个任务周期调用一次（前提是已将接口登录到循环任务调用器）。

ITcPostCyclic 接口由 TwinCAT 模块实现。当模块自主登录任务时，通常作为从 SafeOP 转换到 OP 的最后一步初始化，会将该接口传递至 ITcCyclicCaller::AddPostModule() 方法。登录后会调用模块实例的 PostCycleUpdate() 方法。

12.8.12.1 方法 ITcPostCyclic:PostCyclicUpdate

PostCyclicUpdate 方法通常由输出更新后的 TwinCAT 任务在接口登录后调用。

语法

```
HRESULT TCOMAPI PostCycleUpdate(ITcTask* ipTask, ITcUnknown* ipCaller, ULONG_PTR context)
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

目前，该返回值已被 TwinCAT 任务忽略。

参数

名称	类型	描述
ipTask	ITcTask	指的是当前任务环境。
ipCaller	ITcUnknown	指的是调用实例。
环境	ULONG_PTR	Context 包含传递至 ITcPostCyclicCaller::AddPostModule() 方法的值。

功能说明

在一个任务周期内，所有已注册模块实例调用 OutputUpdate() 方法后，会调用 PostCycleUpdate() 方法。因此，必须使用此方法来实现这种周期处理。

12.8.13 接口 ITcRTimeTask

查询 TwinCAT 任务扩展信息。

语法

TCOM_DECL_INTERFACE("02000003-0000-0000-e000-000000000064", ITcRTimeTask)
struct __declspec(novtable) ITcRTimeTask : public ITcTask

要求包括：`TcRtInterfaces.h`

方法

名称	描述
GetCpuAccount [▶ 181]	查询 TwinCAT 任务的 CPU 占用率。

通过此接口可以查询和使用 TwinCAT 任务信息。

请参见 Sample30：时序测量 [▶ 298]

12.8.13.1 方法 ITcRTimeTask::GetCpuAccount()

查询 TwinCAT 任务的 CPU 占用率。

语法

```
virtual HRESULT TCOMAPI GetCpuAccount(PULONG pAccount)=0;
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

如果参数 pAccount = NULL，则为 E_POINTER。

参数

名称	类型	描述
pAccount	PULONG	[out] TwinCAT Task CPU占用率存储在该参数中。

GetCpuAccount() 方法可用于查询任务当前所用的计算时间。

显示 GetCpuAccount() 使用情况的代码片段，例如在 ITcCyclic::CycleUpdate() 方法中的使用情况：

```
// CPU account in 100 ns interval
ITcRTimeTaskPtr spRTimeTask = ipTask;
ULONG nCpuAccountForComputeSomething = 0;
if (spRTimeTask != NULL)
```

```
{
ULONG nStart = 0;
hr = FAILED(hr) ? hr : spRTimeTask->GetCpuAccount(&nStart);

ComputeSomething();

ULONG nStop = 0;
hr = FAILED(hr) ? hr : spRTimeTask->GetCpuAccount(&nStop);

nCpuAccountForComputeSomething = nStop - nStart;
}
```

12.8.14 接口 ITcTask

查询 TwinCAT 任务的时间戳和任务特定信息。

语法

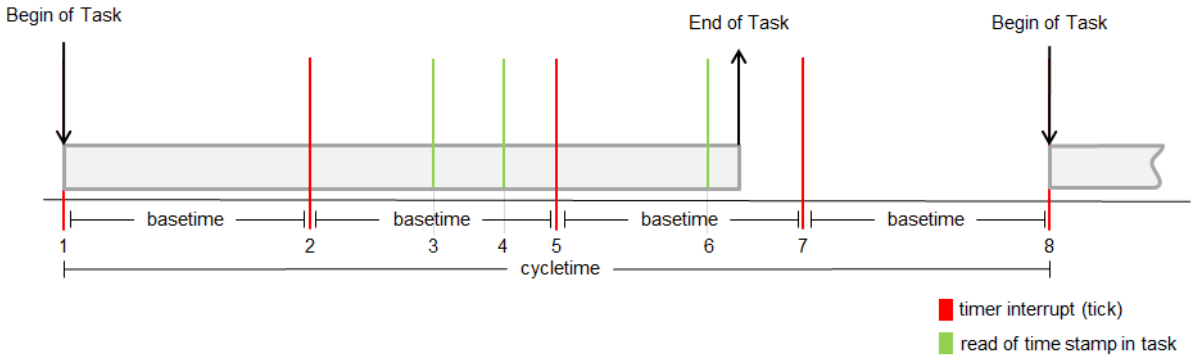
```
TCOM_DECL_INTERFACE("02000002-0000-0000-e000-000000000064", ITcTask)
struct __declspec(novtable) ITcTask : public ITcUnknown
```

要求包括：TcRtInterfaces.h

方法

名称	描述
GetCycleCounter [▶ 184]	查询任务开始后的任务周期数。
GetCycleTime [▶ 185]	以纳秒为单位查询任务循环时间，即“任务开始”到下一次“任务开始”之间的时间间隔。
GetPriority [▶ 183]	查询任务优先级。
GetCurrentSysTime [▶ 183]	查询自 1601 年 1 月 1 日 (UTC) 起以 100 纳秒为间隔的任务循环开始时间。
GetCurrentDcTime [▶ 184]	查询自 2000 年 1 月 1 日起以纳秒为单位的任务循环开始时的分布式时钟时间。
GetCurPentiumTime [▶ 184]	查询自 1601 年 1 月 1 日 (UTC) 起以 100 纳秒为间隔调用方法的时间。

通过 ITcTask 接口，可在实时环境中测量时间。



Reference	Function (ITcTask)	Unit	Zerotime	read time stamps at		
				3	4	6
Distributed Clock master (EtherCAT, Sercos,...)	GetCurrentSysTime	100ns	01.01.1601	1	1	1
	GetCurrentDcTime	1ns	01.01.2000	1	1	1
Processor Clock	GetCurPentiumTime	100ns	01.01.1601	3	4	6

12.8.14.1 方法 ITcTask:GetPriority

查询任务优先级。

语法

```
virtual HRESULT TCOMAPI GetPriority(PULONG
pPriority)=0;
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

如果参数 pPriority = NULL，则为 E_POINTER。

参数

名称	类型	描述
pPriority	PULONG	[out] 任务的优先级值存储在该参数中。

Sample30：时序测量 [▶ 298] 说明了该方法的使用情况。

12.8.14.2 方法 ITcTask:GetCurrentSysTime

查询自 1601 年 1 月 1 日 (UTC) 起以 100 纳秒为间隔的任务循环开始时间。

语法

```
virtual HRESULT TCOMAPI GetCurrentSysTime(PLONGLONG
pSysTime)=0;
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

如果参数 pSysTime = NULL，则为 E_POINTER。

参数

名称	类型	描述
pSysTime	PLONGLONG	[out] 该参数会在任务周期开始时存储当前系统时间。

Sample30：时序测量 [▶ 298] 说明了该方法的使用情况。

12.8.14.3 方法 ITcTask:GetCurrentDcTime

查询自 2000 年 1 月 1 日起以纳秒为单位的任务循环开始时的分布式时钟时间。

语法

```
virtual HRESULT TCOMAPI GetCurrentDcTime(PLONGLONG
pDcTime)=0;
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

如果参数 pDcTime = NULL，则为 E_POINTER。

参数

名称	类型	描述
pDcTime	PLONGLONG	[out] 该参数会在任务周期开始时存储分布式时钟时间。

Sample30：时序测量 [▶ 298] 说明了该方法的使用情况。

12.8.14.4 方法 ITcTask:GetCurPentiumTime

查询自 1601 年 1 月 1 日 (UTC) 起以 100 纳秒为间隔调用方法的时间。

语法

```
virtual HRESULT TCOMAPI GetCurPentiumTime(PLONGLONG
pCurTime)=0;
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

如果参数 pCurTime = NULL，则为 E_POINTER。

参数

名称	类型	描述
pCurTime	PLONGLONG	[out] 该参数会存储自 1601 年 1 月 1 日起以 100 纳秒为间隔的当前时间 (UTC)。

Sample30：时序测量 [▶ 298] 说明了该方法的使用情况。

12.8.14.5 方法 ITcTask:GetCycleCounter

查询任务开始后的任务周期数。

语法

```
virtual HRESULT TCOMAPI GetCycleCounter(PULONGLONG
pCnt)=0;
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

如果参数 pCnt = NULL，则为 E_POINTER。

参数

名称	类型	描述
pCnt	PULONGLONG	[out] 此参数存储自任务启动以来的任务周期数。

Sample30：时序测量 [▶ 298] 说明了该方法的使用情况。

12.8.14.6 方法 ITcTask:GetCycleTime

以纳秒为单位查询任务循环时间，即“任务开始”到下一次“任务开始”之间的时间间隔。

语法

```
virtual HRESULT TCOMAPI GetCycleTime(PULONG
pCycleTimeNS)=0;
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

如果参数 pCnt = NULL，则为 E_POINTER。

参数

名称	类型	描述
pCycleTimeNS	PULONG	[out] 配置的任务循环时间以纳秒为单位存储在此参数中。

Sample30：时序测量 [▶ 298] 说明了该方法的使用情况。

12.8.15 接口 ITcTaskNotification

如果在上一个循环中已超过周期时间，则执行回调。
该接口会提供 PLC PlcTaskSystemInfo->CycleTimeExceeded 等类似功能。

语法

```
TCOM_DECL_INTERFACE("9CDE7C78-32A0-4375-827E-924B31021FCD", ITcTaskNotification) struct
__declspec(novtable) ITcTaskNotification: public ITcUnknown
```

要求包括：TcRtInterfaces.h

方法

名称	描述
NotifyCycleTimeExceeded [▶ 186]	如果超过周期时间则调用。

请注意，回调不会出现在计算过程中，而是在周期结束时。因此该方法无法用于立即中止计算。

12.8.15.1 方法 ITcTaskNotification::NotifyCycleTimeExceeded()

当上一周期已超时后被调用——并非在超时瞬间，而是在超时后触发。

语法

```
virtual HRESULT TCOMAPI NotifyCycleTimeExceeded ();
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

参数

名称	类型	描述
ipTask	ITcTask	指的是当前任务环境。
环境	ULONG_PTR	环境

12.8.16 接口 ITcUnknown

ITcUnknown 定义引用计数以及对更具体接口的引用查询。

语法

```
TCOM_DECL_INTERFACE("00000001-0000-0000-e000-000000000064", ITcUnknown)
```

声明形式：TcInterfaces.h

要求包括：

方法

名称	描述
TcAddRef [▶ 186]	增加引用计数器。
TcQueryInterface [▶ 187]	通过 IID 查询已实施接口的引用。
TcRelease [▶ 188]	减少引用计数器。

每个 TcCOM 界面都直接或间接从 ITcUnknown 派生 因此，每个 TcCOM 模块类均执行 ITcUnknown，因为它是从 ITComObject 派生。

ITcUnknown 的标准执行可确保在释放最后一个引用后删除对象。因此，在调用 TcRelease() 后，不得取消引用接口指针。

12.8.16.1 方法 ITcUnknown:TcAddRef

增加引用计数器并返回新值。

语法

```
ULONG TcAddRef( )
```

返回值

产生引用计数值。

12.8.16.2 方法 ITcUnknown:TcQueryInterface

根据接口 ID (IID) 给出的接口查询接口指针。

语法

```
HRESULT TcQueryInterface(RITCID iid, PPVOID pipItf )
```

返回值

如果成功，将返回 S_OK（“0”）或其他正值，请参见返回值 [▶ 160]。关于扩展信息，请特别参见 ADS 返回代码 [▶ 321]中的 HRESULT 栏。

如果所要求的接口不可用，该方法将返回 ADSERR_DEVICE_NOINTERFACE。

参数

名称	类型	描述
iid	RITCID	接口 IID。
pipItf	PPVOID	指向接口指针。当所请求的接口类型在相应实例中可用时设置。

描述

通过 IID 查询已实施接口的引用。建议使用智能指针来初始化和保存接口指针。

变量 1：

```
HRESULT GetTraceLevel(ITcUnkown* ip, TcTraceLevel& tl)
{
HRESULT hr = S_OK;
if (ip != NULL)
{
ITComObjectPtr spObj;
hr = ip->TcQueryInterface(spObj.GetIID(), &spObj);
if (SUCCEEDED(hr))
{
hr = spObj->TcGetObjPara(PID_TcTraceLevel, &tl, sizeof(tl));
}
return hr;
}
}
```

与智能指针相关的接口 ID 可用作 TcQueryInterface 中的参数。“&”操作符将返回指向智能指针内部接口指针成员变量的指针。变量 1 假定当 TcQueryInterface 指示成功时，接口指针被初始化。如果范围仍然存在，则智能指针 spObj 的析构函数会释放引用。

变量 2：

```
HRESULT GetTraceLevel(ITcUnkown* ip, TcTraceLevel& tl)
{
HRESULT hr = S_OK;
ITComObjectPtr spObj = ip;
if (spObj != NULL)
{
spObj->TcGetObjParam(PID_TcTraceLevel, &tl);
}
else
{
hr = ADS_E_NOINTERFACE;
}
return hr;
}
```

如果接口指针 ip 被分配给智能指针 spObj，则将在 ip 引用的实例上使用 IID_ITComObject 隐式调用 TcQueryInterface 方法。这样会缩短代码，但会丢失 TcQueryInterface 的原始返回代码。

12.8.16.3 方法 ITcUnknown:TcRelease

该方法减少引用计数器。

语法

```
ULONG TcRelease( )
```

返回值

产生引用计数值。

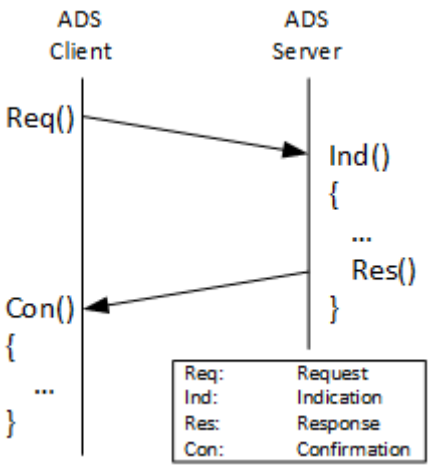
描述

减少引用计数器并返回新值。

如果引用计数器为 0，则对象会自行删除。

12.9 ADS 通信

ADS基于客户端 - 服务器架构实现。ADS 查询会调用服务器端相应的指示方法。ADS 响应会调用客户端相应的确认方法。



ADS客户端 - 服务器通信序列

本节将介绍 TwinCAT 3 C++ 模块的输出和输入 ADS 通信。

ADS 命令集	描述
AdsReadDeviceInfo [▶ 189]	通过这条命令可以读取通用设备的信息。
AdsRead [▶ 190]	ADS 读取命令，用于从 ADS 设备中获取数据。
AdsWrite [▶ 192]	ADS 写入命令，用于向 ADS 设备传输数据。
AdsReadState [▶ 196]	用于查询 ADS 设备状态的 ADS 命令。
AdsWriteControl [▶ 197]	用于改变 ADS 设备状态的 ADS 控制命令。
AdsAddDeviceNotification [▶ 199]	观察变量。如果发生事件，会通知客户端。
AdsDelDeviceNotification [▶ 201]	删除之前关联的变量。
AdsDeviceNotification [▶ 202]	用于传输设备通知事件。
AdsReadWrite [▶ 194]	ADS 读取/写入命令。通过一次调用，即可将数据传输至 ADS 设备（写入操作）并读取其响应数据。

ADS 返回代码 [▶ 321] 适用于整个 ADS 通信。

首先请查看 Sample07：接收 ADS 通知 [▶ 252]。

12.9.1 AdsReadDeviceInfo

12.9.1.1 AdsReadDeviceInfoReq

方法 AdsDeviceInfoReq 可传输 ADS 设备信息指令，用于读取 ADS 服务器的标识和版本号。
收到响应时会调用 AdsReadDeviceInfoCon。

语法

```
int AdsReadDeviceInfoReq( AmsAddr& rAddr, ULONG invokeId );
```

返回值

类型：int

错误代码 – 请参见 AdsStatuscodes [▶ 321]。

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。

12.9.1.2 AdsReadDeviceInfoInd

AdsDeviceInfoInd 方法指代 ADS DeviceInfo 指令，用于读取 ADS 服务器的标识和版本号。然后必须调用 AdsReadDeviceInfoRes [▶ 189]。

语法

```
void AdsReadDeviceInfoInd( AmsAddr& rAddr, ULONG invokeId );
```

返回值

void

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。

12.9.1.3 AdsReadDeviceInfoRes

方法 AdsReadDeviceInfoRes 发送 ADS 读取设备信息。AdsReadDeviceInfoCon [▶ 190] 构成对应程序，随后会被调用。

语法

```
int AdsReadDeviceInfoRes( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, CHAR
name[ADS_FIXEDNAMESIZE], AdsVersion version );
```

返回值

类型：int

错误代码 – 请参见 AdsStatuscodes [▶ 321]。

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含响应 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
nResult	ULONG	[in] 包含 ADS 写入命令的结果；请参见 AdsStatuscodes [▶ 321]。
名称	char[ADS_FIXEDNAMESIZE	[in] 包含设备名称。
版本	AdsVersion	[in] 设备的构建 (int)、修订 (byte) 和版本 (byte) 结构。

12.9.1.4 AdsReadDeviceInfoCon

通过 AdsReadDeviceInfoCon 方法，可以接收 ADS 读取设备信息的确认。接收模块必须实现此方法。且需预先调用对应的 AdsReadDeviceInfoReq [▶ 189]。

语法

```
void AdsReadDeviceInfoCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult,
CHAR name[ADS_FIXEDNAMESIZE], AdsVersion version );
```

返回值

void

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含响应 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
nResult	ULONG	[in] 包含 ADS 写入命令的结果；请参见 AdsStatuscodes [▶ 321]。
名称	char[ADS_FIXEDNAMESIZE]	[in] 包含设备名称。
版本	AdsVersion	[in] 设备的构建 (int)、修订 (byte) 和版本 (byte) 结构。

12.9.2 AdsRead

12.9.2.1 AdsReadReq

通过 AdsReadReq 方法，可以发送 ADS 读取命令，以便从 ADS 设备传输数据。
AdsReadCon [▶ 192] 在收到响应后被调用。

语法

```
int AdsReadReq( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup, ULONG indexOffset, ULONG
cbLength );
```

返回值

类型：int

错误代码 – 请参见 AdsStatuscodes [▶ 321]。

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
indexGroup	ULONG	[in] 包含所请求 ADS 服务的索引组号（32 位，无符号）。
indexOffset	ULONG	[in] 包含所请求 ADS 服务的索引偏移编号（32 位，无符号）。
cbLength	ULONG	[in] 包含要读取数据 (pData) 的长度（以字节为单位）。

12.9.2.2 AdsReadInd

通过 AdsReadInd 方法可以接收 ADS 读取请求。必须调用 AdsReadRes [▶ 191] 才能发送结果。

语法

```
void AdsReadInd( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup, ULONG indexOffset, ULONG
cbLength );
```

返回值

类型：int

ADS 返回代码 – 请参见 AdsStatuscodes [▶ 321]。

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含响应 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
indexGroup	ULONG	[in] 包含所请求 ADS 服务的索引组号（32 位，无符号）。
indexOffset	ULONG	[in] 包含所请求 ADS 服务的索引偏移编号（32 位，无符号）。
cbLength	ULONG	[in] 包含要读取数据 (pData) 的长度（以字节为单位）。

12.9.2.3 AdsReadRes

通过 AdsReadRes 方法可以发送 ADS 读取响应。AdsReadCon [▶ 192] 构成对应的方法，随后被调用。

语法

```
int AdsReadRes( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, ULONG cbLength, PVOID pData );
```

返回值

类型：int

ADS 返回代码 – 请参见 AdsStatuscodes [▶ 321]。

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含响应 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
nResult	ULONG	[in] 包含 ADS 读取命令的结果；请参见 AdsStatuscodes [▶ 321]。
cbLength	ULONG	[in] 包含以字节为单位的读取数据 (pData) 长度。
pData	PVOID	[in] 指向数据所在数据缓冲区的指针。

12.9.2.4 AdsReadCon

通过 AdsReadCon 方法可以接收 ADS 读取操作的确认响应。接收模块必须提供此方法。
对应的方法是 AdsReadReq [▶ 190]，必须已提前调用。

语法

```
void AdsReadCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, ULONG cbLength, PVOID pData );
```

返回值

void

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含响应 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
nResult	ULONG	[in] 包含 ADS 读取命令的结果；请参见 AdsStatuscodes [▶ 321]。
cbLength	ULONG	[in] 包含以字节为单位的读取数据 (pData) 长度。
pData	PVOID	[in] 指向数据所在数据缓冲区的指针。

12.9.3 AdsWrite

12.9.3.1 AdsWriteReq

通过 AdsWriteReq 方法可以发送 ADS 写入命令，将数据传输到 ADS 设备。
AdsWriteCon [▶ 193] 在收到响应后被调用。

语法

```
int AdsWriteReq( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup, ULONG indexOffset, ULONG
cbLength, PVOID pData );
```

返回值

类型：int

错误代码 – 请参见 AdsStatuscodes [▶ 321]。

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
indexGroup	ULONG	[in] 包含所请求 ADS 服务的索引组号（32 位，无符号）。
indexOffset	ULONG	[in] 包含所请求 ADS 服务的索引偏移编号（32 位，无符号）。
cbLength	ULONG	[in] 包含要写入数据 (pData) 的长度（以字节为单位）。
pData	PVOID	[in] 指向写入数据所在数据缓冲区的指针。

12.9.3.2 AdsWriteInd

AdsWriteInd 方法指定了一条 ADS 写入命令，用于向 ADS 设备传输数据。
必须调用 AdsWriteRes [▶ 193] 来确认操作完成。

语法

```
void AdsWriteInd( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup, ULONG indexOffset, ULONG
cbLength, PVOID pData );
```

返回值

void

错误代码 – 请参见 AdsStatuscodes [▶ 321]。

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
indexGroup	ULONG	[in] 包含所请求 ADS 服务的索引组号（32 位，无符号）。
indexOffset	ULONG	[in] 包含所请求 ADS 服务的索引偏移编号（32 位，无符号）。
cbLength	ULONG	[in] 包含要写入数据 (pData) 的长度（以字节为单位）。
pData	PVOID	[in] 指向写入数据所在数据缓冲区的指针。

12.9.3.3 AdsWriteRes

AdsWriteRes 方法可发送 ADS 写入响应。AdsWriteCon [▶ 193] 构成对应的方法，随后被调用。

语法

```
int AdsWriteRes( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

返回值

类型：int

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含响应 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
nResult	ULONG	[in] 包含 ADS 写入命令的结果；请参见 AdsStatuscodes [▶ 321]。

ADS 返回代码 – 请参见 AdsStatuscodes [▶ 321]。

12.9.3.4 AdsWriteCon

通过 AdsWriteCon 方法可以接收 ADS 写入确认响应。接收模块必须提供此方法。
AdsWriteReq [▶ 192] 是对应的方法，必须已提前调用。

语法

```
void AdsWriteCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

返回值

void

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含响应 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
nResult	ULONG	[in] 包含 ADS 写入命令的结果；请参见 AdsStatuscodes [▶ 321]。

12.9.4 AdsReadWrite

12.9.4.1 AdsReadWriteReq

通过 AdsReadWriteReq 方法可以发送 ADS 读取/写入命令，以便与 ADS 设备之间进行数据传输。收到响应后，AdsReadWriteCon [▶ 195] 将被调用。

语法

```
int AdsReadWriteReq( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup, ULONG indexOffset, ULONG
cbReadLength, ULONG cbWriteLength, PVOID pData );
```

返回值

类型：int

错误代码 – 请参见 AdsStatuscodes [▶ 321]。

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
indexGroup	ULONG	[in] 包含所请求 ADS 服务的索引组号（32 位，无符号）。
indexOffset	ULONG	[in] 包含所请求 ADS 服务的索引偏移编号（32 位，无符号）。
cbReadLength	ULONG	[in] 包含要读取数据 (pData) 的长度（以字节为单位）。
cbWriteLength	ULONG	[in] 包含要写入数据 (pData) 的长度（以字节为单位）。
pData	PVOID	[in] 指向写入数据所在数据缓冲区的指针。

12.9.4.2 AdsReadWriteInd

AdsReadWriteInd 方法可指定向 ADS 设备传输数据的 ADS 读取/写入命令。必须调用 AdsReadWriteRes [▶ 196] 才能发送结果。

语法

```
void AdsReadWriteInd( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup,
ULONG indexOffset, ULONG cbReadLength, ULONG cbWriteLength, PVOID pData );
```

返回值

void

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
indexGroup	ULONG	[in] 包含所请求 ADS 服务的索引组号（32 位，无符号）。
indexOffset	ULONG	[in] 包含所请求 ADS 服务的索引偏移编号（32 位，无符号）。
cbReadLength	ULONG	[in] 包含要读取数据 (pData) 的长度（以字节为单位）。
cbWriteLength	ULONG	[in] 包含要写入数据 (pData) 的长度（以字节为单位）。
pData	PVOID	[in] 指向写入数据所在数据缓冲区的指针。

12.9.4.3 AdsReadWriteRes

通过 AdsReadWriteRes 方法可以接收 ADS 读取/写入确认。AdsReadWriteCon [▶ 195] 构成对应的方法，随后被调用。

语法

```
int AdsReadWriteRes( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, ULONG cbLength, PVOID pData );
```

返回值

类型：int

ADS 返回代码 – 请参见 AdsStatuscodes [▶ 321]。

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含响应 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
nResult	ULONG	[in] 包含 ADS 写入命令的结果；请参见 AdsStatuscodes [▶ 321]。
cbLength	ULONG	[in] 包含以字节为单位的读取数据 (pData) 长度。
pData	PVOID	[in] 指向数据所在数据缓冲区的指针。

12.9.4.4 AdsReadWriteCon

通过 AdsReadWriteCon 方法可以接收 ADS 读取/写入确认。接收模块必须提供此方法。
AdsReadWriteReq [▶ 194] 是对应的方法，必须提前调用。

语法

```
void AdsReadWriteCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, ULONG cbLength, PVOID pData );
```

返回值

void

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含响应 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
nResult	ULONG	[in] 包含 ADS 写入命令的结果；请参见 AdsStatuscodes [▶ 321]。
cbLength	ULONG	[in] 包含以字节为单位的读取数据 (pData) 长度。
pData	PVOID	[in] 指向数据所在数据缓冲区的指针。

12.9.5 AdsReadState

12.9.5.1 AdsReadStateReq

通过 AdsReadStateReq 方法可以传输 ADS 状态读取命令，以便从 ADS 服务器读取 ADS 和设备状态。收到响应后，AdsReadStateCon [▶ 197] 将被调用。

语法

```
int AdsReadStateReq(AmsAddr& rAddr, ULONG invokeId);
```

返回值

类型：int

错误代码 – 请参见 AdsStatuscodes [▶ 321]。

参数

名称	类型	描述
rAddr	AmsAddr	[in] 包含 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。

12.9.5.2 AdsReadStateInd

AdsReadStateInd 方法显示 ADS 状态读取命令，用于从 ADS 设备读取 ADS 状态和设备状态。必须调用 AdsReadStateRes [▶ 196] 才能发送结果。

语法

```
void AdsReadStateInd( AmsAddr& rAddr, ULONG invokeId );
```

返回值

void

参数

名称	类型	描述
rAddr	AmsAddr	[in] 包含 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。

12.9.5.3 AdsReadStateRes

通过方法 AdsWriteRes 可以发送 ADS 状态读取响应。AdsReadStateCon [▶ 197] 构成对应的方法，随后被调用。

语法

```
int AdsReadStateRes( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, USHORT adsState, USHORT
deviceState );
```

返回值

类型：int

错误代码 – 请参见 AdsStatuscodes [▶ 321]。

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含响应 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
nResult	ULONG	[in] 包含 ADS 写入命令的结果；请参见 AdsStatuscodes [▶ 321]。
adsState	USHORT	[in] 包含设备的 ADS 状态。
deviceState	USHORT	[in] 包含设备的设备状态。

12.9.5.4 AdsReadStateCon

通过 AdsWriteCon 方法可以接收 ADS 状态读取确认。接收模块必须提供此方法。
AdsReadStateReq [▶ 196] 是对应的方法，必须提前调用。

语法

```
void AdsReadStateCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, USHORT adsState, USHORT
deviceState );
```

返回值

void

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含响应 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
nResult	ULONG	[in] 包含 ADS 写入命令的结果；请参见 AdsStatuscodes [▶ 321]。
adsState	USHORT	[in] 包含设备的 ADS 状态。
deviceState	USHORT	[in] 包含设备的设备状态。

12.9.6 AdsWriteControl

12.9.6.1 AdsWriteControlReq

AdsWriteControlReq 方法可用于发送 ADS 写入控制命令，以改变 ADS 服务器的 ADS 和设备状态。收到响应后，AdsWriteControlCon [▶ 199] 将被调用。

语法

```
int AdsWriteControlReq( AmsAddr& rAddr, ULONG invokeId, USHORT adsState,
USHORT deviceState, ULONG cbLength, PVOID pData );
```

返回值

类型：int

错误代码 – 请参见 AdsStatuscodes [▶ 321]。

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
adsState	USHORT	[in] 新的 ADS 状态（请参见 Ads.h 中的枚举 nAdsState）。
deviceState	USHORT	[in] 新的设备状态。
cbLength	ULONG	[in] 包含数据 (pData) 的长度（以字节为单位）。
pData	PVOID	[in] 指向写入数据所在数据缓冲区的指针。

12.9.6.2 AdsWriteControlInd

AdsWriteControlInd 方法可用于发送 ADS 写入控制命令，以改变 ADS 设备的 ADS 状态及设备状态。并且需预先调用 AdsWriteControlRes [▶ 198] 来确认流程。

语法

```
void AdsWriteControlInd( AmsAddr& rAddr, ULONG invokeId, USHORT adsState, USHORT deviceState, ULONG cbLength, PVOID pDeviceData );
```

返回值

void

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
adsState	USHORT	[in] 包含所请求 ADS 服务的索引组号（32 位，无符号）。
deviceState	USHORT	[in] 包含所请求 ADS 服务的索引偏移编号（32 位，无符号）。
cbLength	ULONG	[in] 包含数据 (pData) 的长度（以字节为单位）。
pData	PVOID	[in] 指向写入数据所在数据缓冲区的指针。

12.9.6.3 AdsWriteControlRes

通过 AdsWriteControlRes 方法可以发送 ADS 写入控制响应。AdsWriteControlCon [▶ 199] 构成对应的方法，随后被调用。

语法

```
int AdsWriteControlRes( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

返回值

类型：int

ADS 返回代码 – 请参见 AdsStatuscodes [▶ 321]。

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含响应 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
nResult	ULONG	[in] 包含 ADS 写入命令的结果；请参见 AdsStatuscodes [▶ 321]。

12.9.6.4 AdsWriteControlCon

通过 AdsWriteCon 方法可以接收 ADS 写入控制确认响应。接收模块必须提供该方法。
AdsWriteControlReq [▶ 197] 是对应的方法，必须提前调用。

语法

```
void AdsWriteControlCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

返回值

void

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含响应 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
nResult	ULONG	[in] 包含 ADS 写入命令的结果；请参见 AdsStatuscodes [▶ 321]。

12.9.7 AdsAddDeviceNotification

12.9.7.1 AdsAddDeviceNotificationReq

通过 AdsAddDeviceNotificationReq 方法，可以发送 ADS 添加设备通知命令，为 ADS 设备添加设备通知。收到响应后，AdsAddDeviceNotificationCon [▶ 200] 将被调用。

语法

```
int AdsAddDeviceNotificationReq( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup, ULONG
indexOffset,
AdsNotificationAttrib noteAttrib);
```

返回值

类型：int

错误代码 – 请参见 AdsStatuscodes [▶ 321]。

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
indexGroup	ULONG	[in] 包含所请求 ADS 服务的索引组号（32 位，无符号）。
indexOffset	ULONG	[in] 包含所请求 ADS 服务的索引偏移编号（32 位，无符号）。
noteAttrib	AdsNotificationAttrib	[in] 包含通知参数（cbLength、TransMode、MaxDelay）的规范。

12.9.7.2 AdsAddDeviceNotificationInd

通过 AdsAddDeviceNotificationInd 方法应能发送 AdsDeviceNotification [▶ 202]。必须调用 AdsAddDeviceNotificationRes [▶ 200] 来确认流程。

语法

```
void AdsAddDeviceNotificationInd( AmsAddr& rAddr, ULONG invokeId, ULONG indexGroup, ULONG
indexOffset, AdsNotificationAttrib noteAttrib );
```

返回值

void

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含响应 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
indexGroup	ULONG	[in] 包含所请求 ADS 服务的索引组号（32 位，无符号）。
indexOffset	ULONG	[in] 包含所请求 ADS 服务的索引偏移编号（32 位，无符号）。
noteAttrib	AdsNotificationAttrib	[in] 包含通知参数（cbLength、TransMode、MaxDelay）的规范。

12.9.7.3 AdsAddDeviceNotificationRes

通过 AdsAddDeviceNotificationRes 方法，可以发送 ADS 设备添加通知响应。
AdsAddDeviceNotificationCon [▶ 200] 构成对应的方法，随后被调用。

语法

```
void AdsAddDeviceNotificationCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, ULONG handle );
```

返回值

void

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含响应 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
nResult	ULONG	[in] 包含 ADS 写入命令的结果；请参见 AdsStatuscodes。
Handle	ULONG	[in] 生成设备通知的句柄。

12.9.7.4 AdsAddDeviceNotificationCon

AdsAddDeviceNotificationCon 方法确认 ADS 设备添加通知请求。AdsAddDeviceNotificationReq [▶ 199] 构成对应的方法，必须提前调用。

语法

```
void AdsAddDeviceNotificationCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult, ULONG handle );
```

返回值

void

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含响应 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
nResult	ULONG	[in] 包含 ADS 写入命令的结果；请参见 AdsStatuscodes。
Handle	ULONG	[in] 生成设备通知的句柄。

12.9.8 AdsDelDeviceNotification

12.9.8.1 AdsDelDeviceNotificationReq

通过 AdsDelDeviceNotificationReq 方法，可以发送 ADS 设备删除通知命令，以从 ADS 设备上删除设备通知。收到响应后，AdsDelDeviceNotificationCon [▶ 202] 将被调用。

语法

```
int AdsDelDeviceNotificationReq( AmsAddr& rAddr, ULONG invokeId, ULONG hNotification );
```

返回值

类型：int

错误代码 – 请参见 AdsStatuscodes [▶ 321]。

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
hNotification	ULONG	[in] 包含要删除的通知句柄。

12.9.8.2 AdsDelDeviceNotificationInd

通过 AdsAddDeviceNotificationCon 方法，可以接收 ADS 设备删除通知确认。接收模块必须实现此方法。并且需预先调用 AdsDelDeviceNotificationRes [▶ 202] 来确认流程。

语法

```
void AdsDelDeviceNotificationCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

返回值

void

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含响应 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
nResult	ULONG	[in] 包含 ADS 写入命令的结果；请参见 AdsStatuscodes [▶ 321]。

12.9.8.3 AdsDelDeviceNotificationRes

通过 AdsAddDeviceNotificationRes 方法，可以接收 ADS 设备删除通知。AdsDelDeviceNotificationCon [▶ 202] 构成对应的方法，随后被调用。

语法

```
int AdsDelDeviceNotificationRes( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

返回值

Int

返回 ADS 命令的结果，请参见 AdsStatuscodes [▶ 321]。

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含响应 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
nResult	ULONG	[in] 包含 ADS 命令的结果；请参见 AdsStatuscodes [▶ 321]。

12.9.8.4 AdsDelDeviceNotificationCon

通过 AdsAddDeviceNotificationCon 方法，可以接收 ADS 设备删除通知确认。接收模块必须实现此方法。AdsDelDeviceNotificationReq [▶ 201] 构成对应的方法，必须提前调用。

语法

```
void AdsDelDeviceNotificationCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

返回值

void

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含响应 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 传输命令的句柄；InvokeId 由源设备指定，用于标识指令。
nResult	ULONG	[in] 包含 ADS 写入命令的结果；请参见 AdsStatuscodes [▶ 321]。

12.9.9 AdsDeviceNotification

12.9.9.1 AdsDeviceNotificationReq

通过 AdsAddDeviceNotificationReq 方法，可以发送 ADS 设备通知，告知 ADS 设备。对应的 AdsDeviceNotificationInd [▶ 203] 会被调用。

语法

```
int AdsDeviceNotificationReq( AmsAddr& rAddr, ULONG invokeId, ULONG cbLength, AdsNotificationStream
notifications[] );
```

返回值

类型：int

ADS 返回代码 – 请参见 AdsStatuscodes [▶ 321]。

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
nResult	ULONG	[in] 包含设备通知显示的结果。
notifications[]	AdsNotificationStream	[in] 包含设备通知的相关信息。

12.9.9.2 AdsDeviceNotificationInd

通过 AdsDeviceNotificationInd 方法，可以接收来自 ADS 设备通知显示的信息。接收模块必须实现此方法。未确认收到。
必须在 AdsDeviceNotificationReq [▶ 203] 端调用 AdsDeviceNotificationCon [▶ 202]，以检查传输情况。

语法

```
void AdsDeviceNotificationInd( AmsAddr& rAddr, ULONG invokeId, ULONG cbLength,
AdsNotificationStream* pNotifications );
```

返回值

void

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含响应 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
cbLength	ULONG	[in] 包含 pNotifications 的长度。
pNotifications	AdsNotificationStream*	[in] 指向通知的指针。该数组由带有通知句柄的 AdsStampHeader 和通过 AdsNotificationSample 传输的数据组成。

12.9.9.3 AdsDeviceNotificationCon

发送方可使用 AdsAddDeviceNotificationCon 方法检查 ADS 设备通知的传输情况。
AdsDeviceNotificationReq [▶ 202] 必须提前调用。

语法

```
void AdsDeviceNotificationCon( AmsAddr& rAddr, ULONG invokeId, ULONG nResult );
```

返回值

void

参数

名称	类型	描述
rAddr	AmsAddr&	[in] 包含 ADS 服务器的 NetId 和端口号的结构。
invokeId	ULONG	[in] 已发送命令的句柄。InvokeId 由源设备指定，用于指令的标识。
nResult	ULONG	[in] 包含设备通知显示的结果。

12.1 数学函数

0

由于 Microsoft 的 math.h 实现不具备实时性，因此 TwinCAT 已实现专属数学函数。
这些函数在 TwinCAT SDK 的一部分 TcMath.h 中有所声明。在 x64 系统中，操作通过 SSE 执行；在 x86 系统中，则使用 FPU。

TwinCAT 3.1 4018 或更早版本

TwinCAT 3.1 4018 可提供具有相同方法的 fpu87.h。此版本继续存在，并重定向到 TcMath.h。

提供的方法

名称	描述
sqr_	计算平方。
sqrt_	计算平方根。
sin_	计算正弦值。
cos_	计算余弦值。
tan_	计算正切值。
atan_	计算指定数值的反正切角度。
atan2_	计算两个指定数值之商对应的反正切角度。
asin_	计算正弦值为指定值的角度。
acos_	计算指定数值对应的反正弦角度。
exp_	计算 自然常数 e 的指定次幂。
log_	计算指定值的对数。
log10_	计算指定值的 10 进制对数。
fabs_	计算绝对值。
fmod_	计算余数。
ceil_	计算大于或等于指定数字的最小整数。
floor_	计算小于或等于指定数字的最大整数。
pow_	计算指定数字的指定次幂。
sincos_	计算 x 的正弦值和余弦值。
fmodabs_	计算符合模运算欧几里得定义的绝对值。
round_	计算数值并四舍五入为最接近的整数。
round_digits_	计算带有指定小数位数的四舍五入值。
coubic_	计算立方值。
ldexp_	根据尾数和指数计算实数（双精度）。
ldexpf_	根据尾数和指数计算实数（浮点数）。
sinh_	计算指定角度的双曲正弦值。
cosh_	计算指定角度的双曲余弦值。
tanh_	计算指定角度的双曲正切值。
finite_	确定指定值是否为有限值。
isnan_	确定指定值是否并非数字 (NAN)。
rands_	计算介于 0 和 32767 之间的伪随机数。参数 holdrand 是随机设置的，每次调用都会改变。

- 这些函数均带有后缀下划线“_”，以此标识其为TwinCAT 专属实现函数。
- 大多数都是由 Microsoft 设计的模拟 math.h，仅用于双精度数据类型。

另请参见

模拟 math.h 功能的 MSDN 文档。

12.1 时间函数

1

TwinCAT 提供用于时间转换的函数，这些函数在 TwinCAT SDK 的一部分 TcTimeConversion.h 中有所声明。

提供的方法

名称	描述
TcDayOfWeek（WORD 日，WORD 月，WORD 年）	确定星期几。 输入：日 (0..30) 和月 (1..12) 返回：0 表示星期日，6 表示星期六
TcIsLeapYear	确定给定年份是否为闰年。
TcDaysInYear	确定给定年份中的天数。
TcDaysInMonth	确定给定月份中的天数。
TcSystemTimeToFileTime(const SYSTEMTIME* lpSystemTime, FILETIME *lpFileTime);	将给定系统时间转换为文件时间。
TcFileTimeToSystemTime(const FILETIME *lpFileTime, SYSTEMTIME* lpSystemTime);	将给定文件时间转换为系统时间。
TcSystemTimeToFileTime(const SYSTEMTIME* lpSystemTime, ULONGLONG& ul64FileTime);	将给定系统时间转换为文件时间（ULONGLONG 格式）。
TcFileTimeToSystemTime(const ULONGLONG& ul64FileTime, SYSTEMTIME* lpSystemTime);	将给定文件时间（ULONGLONG 格式）转换为系统时间。
TcIsISO8601TimeFormat（PCCH sDT）	检查 PCCH 是否遵循 ISO8601 时间格式。
TcDecodeDateTime(PCCH sDT)	将作为 DateTime 的 ULONG 从 PCCH 转换为 ISO8601 格式。
TcDecodeDcTime(PCCH sDT)	将作为 DcTime 的 LONGLONG 从 PCCH 转换为 ISO8601 格式。
TcDecodeFileTime(PCCH sFT)	将作为 FileTime 的 LONGLONG 从 PCCH 转换为 ISO8601 格式。
TcEncodeDateTime(ULONG value, PCHAR p, UINT len)	以 DateTime 格式的 ULONG 值为基础，将字符串 (p, len) 转换为 ISO8601 格式。 p 的最小长度为 24 字节。
TcEncodeDcTime(LONGLONG value, PCHAR p, UINT len)	以 DcTime 格式的 LONGLONG 为基础，将字符串 (p, len) 转换为 ISO8601 格式。 p 的最小长度为 32 字节。
TcEncodeFileTime(LONGLONG value, PCHAR p, UINT len)	以 FileTime 格式的 LONGLONG 为基础，将字符串 (p, len) 转换为 ISO8601 格式。 p 的最小长度为 32 字节。
TcDcTimeToFileTime(LONGLONG dcTime)	将作为 FileTime 的 LONGLONG 转换为 DcTime。
TcFileTimeToDcTime(LONGLONG fileTime);	将作为 DcTime 的 LONGLONG 转换为 FileTime。
TcDcTimeToDateTime(LONGLONG dcTime)	将作为 DateTime 的 ULONG，从 LONGLONG 转换为 DcTime。
TcDateTimeToDcTime(ULONG dateTime)	将作为 DcTime 的 ULONG，从 LONGLONG 转换为 DateTime。
TcFileTimeToDateTime(LONGLONG fileTime)	将作为 DateTime 的 ULONG，从 LONGLONG 转换为 FileTime。
TcDateTimeToFileTime(ULONG dateTime)	将作为 FileTime 的 LONGLONG 从 ULONG 转换为 DateTime。

- 有关不同时间源的更多信息，请参见：
  https://infosys.beckhoff.com/content/1031/ethercatsystem/2469114379.html

12.1 STL/容器

2

TwinCAT 3 C++ 在以下方面支持 STL

- 列表
- 映射
- 设置
- 堆栈
- 字符串
- 向量
- WString
- 算法（如 binary_search）
  - 请参见 %TWINCAT3DIR%\Sdk\Include\Stl\Stl\algorithm，了解支持算法的特定列表。

限制条件

- 并非所有数据类型都有类模板。
- 不得直接使用某些报头文件。

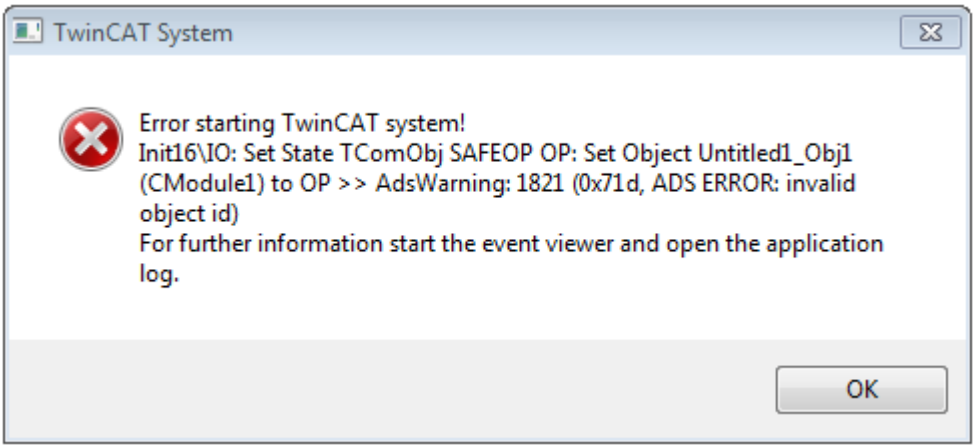
有关使用 STL 的内存管理的详细文档，请参见 内存分配 [▶ 150] 章。

12.1 错误信息的解读

3

在 TwinCAT 中，您会收到有关所发生错误的详尽信息。

例如，这条错误消息的意思是：



- 错误发生在从 SAFE OP 转变到 OP 的过程中。
- 受影响的对象是“Untitled1_Obj1” (CModule1)。
- 错误代码 1821/0x71d 表示对象 ID 无效。

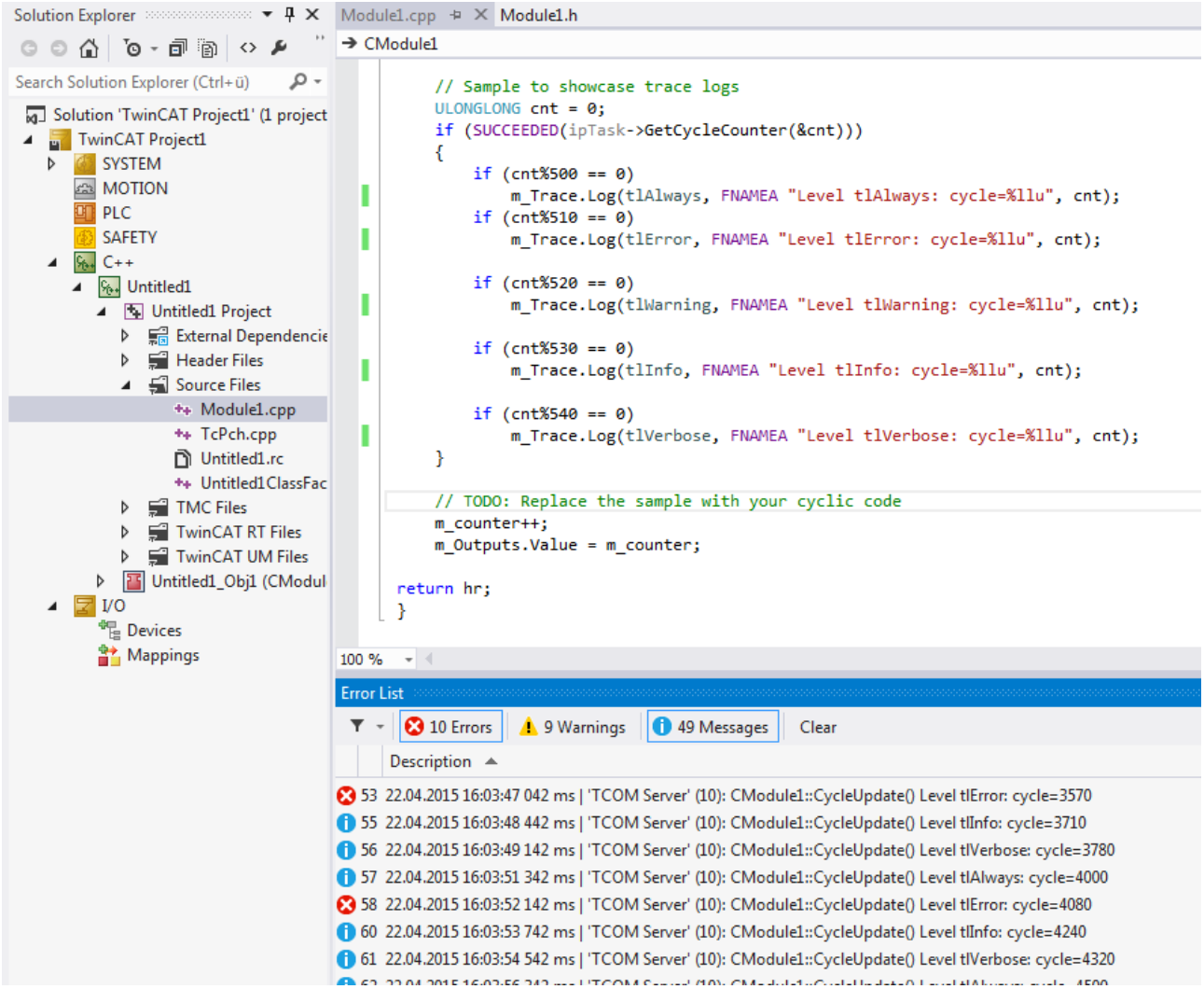
因此，您应该更细致地研究负责这一转变的方法“SetObjStateSP()”。在生成的标准代码中，可以看到模块的添加操作在此处完成。

出现此错误的原因是尚未将任务分配至该模块，因此该模块不可能有执行的任务。

12.1 工程的模块消息（日志记录/追溯）4

概述

TwinCAT 3 C++ 提供从 C++ 模块向 TwinCAT 3 工程环境发送追踪或日志记录消息的选项。



语法

记录消息的语法如下：

```
m_Trace.Log(TLEVEL, FNMACRO"A message", …);
```

通过这些属性：

- `TLEVEL` 可将消息分类为五个级别之一。
  高级别的记录始终包括低级别的记录：例如，被归类为“tlWarning”级别的消息会以“tlAlways”、“tlError”和“tlWarning”级别下被记录，不会记录“tlInfo”和“tlVerbose”消息。

0 级	tlAlways
1 级	tlError
2 级	tlWarning
3 级	tlInfo
4 级	tlVerbose

- `FNMACRO` 可用于将函数名置于要打印的消息之前

- FENTERA：输入函数时使用；打印函数名，后跟“>>>”。
- FNAMEA：在函数内部使用；打印函数名。
- FLEAVEA：在退出函数时使用；打印函数名，后跟“<<<”。

• %q 格式说明符用于输出指针和其他具有特定平台大小的变量。

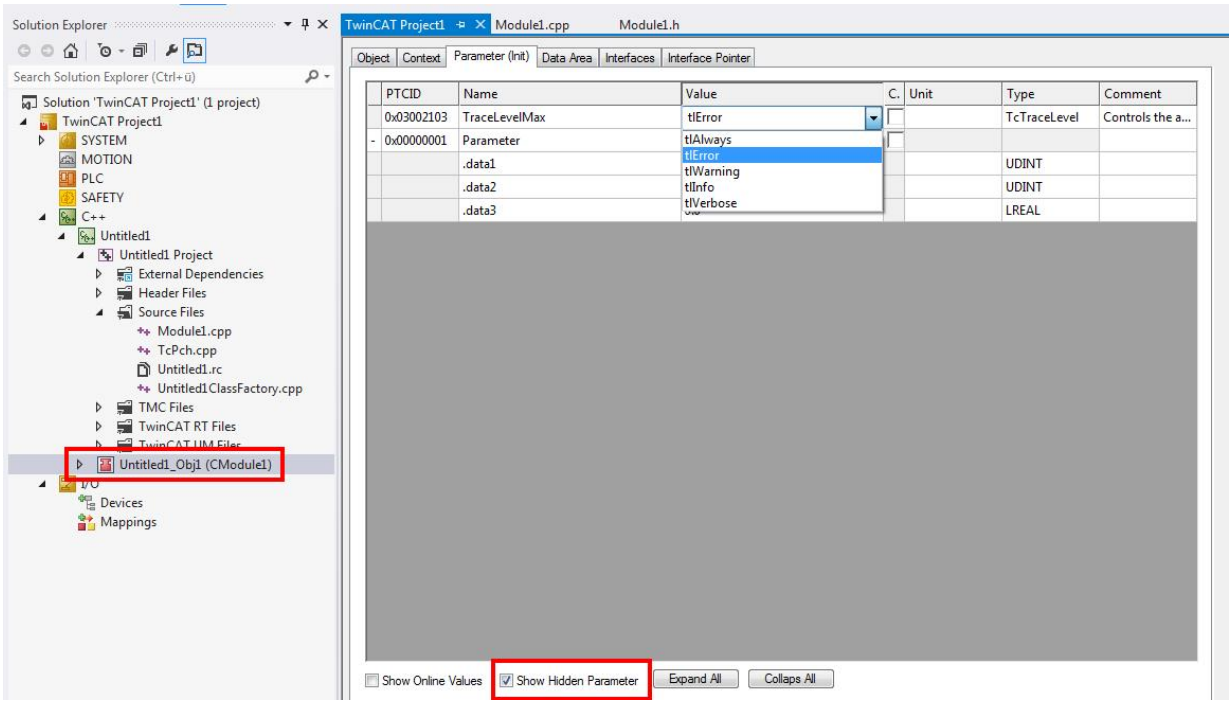
示例

```
HRESULT CModule1::CycleUpdate(ITcTask* ipTask, ITcUnknown* ipCaller, ULONG_PTR context)
{
    HRESULT hr = S_OK;

    // Sample to showcase trace logs
    ULONGLONG cnt = 0;
    if (SUCCEEDED(ipTask->GetCycleCounter(&cnt)))
    {
        if (cnt%500 == 0)
            m_Trace.Log(tlAlways, FENTERA "Level tlAlways: cycle= %llu", cnt);

        if (cnt%510 == 0)
            m_Trace.Log(tlError, FENTERA "Level tlError: cycle=%llu", cnt);

        if (cnt%520 == 0)
            m_Trace.Log(tlWarning, FENTERA "Level tlWarning: cycle=%lld", cnt);

        if (cnt%530 == 0)
            m_Trace.Log(tlInfo, FENTERA "Level tlInfo: cycle=%llu", cnt);

        if (cnt%540 == 0)
            m_Trace.Log(tlVerbose, FENTERA "Level tlVerbose: cycle=%llu", cnt);
    }

    // TODO: Replace the sample with your cyclic code
    m_counter++;
    m_Outputs.Value = m_counter;

    return hr;
}
```

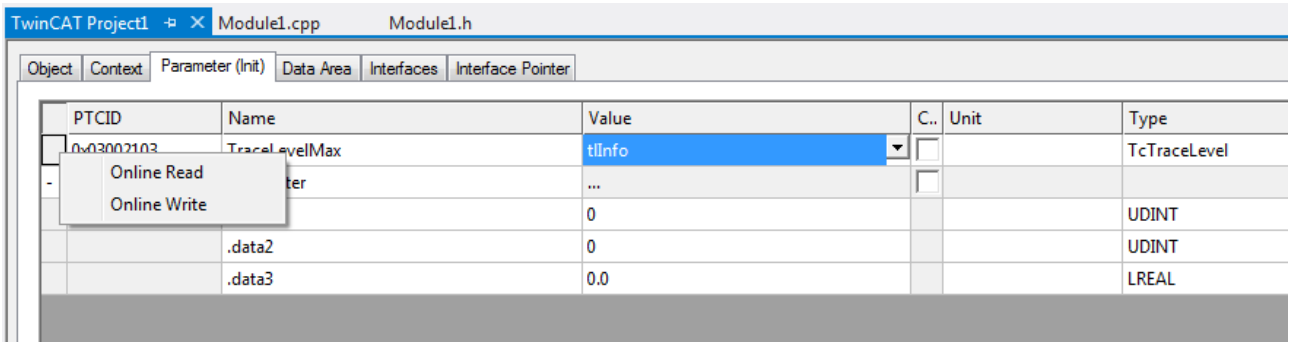
使用跟踪级别

跟踪级别可在模块实例级别预先配置。

1. 导航至解决方案树中的模块实例。
2. 选择右侧的**参数 (Init)** 选项卡。
3. 确保激活 **Show Hidden Parameter**（显示隐藏参数）。
4. 选择跟踪级别。
5. 要测试所有内容，请选择最高级别 **tlVerbose**。



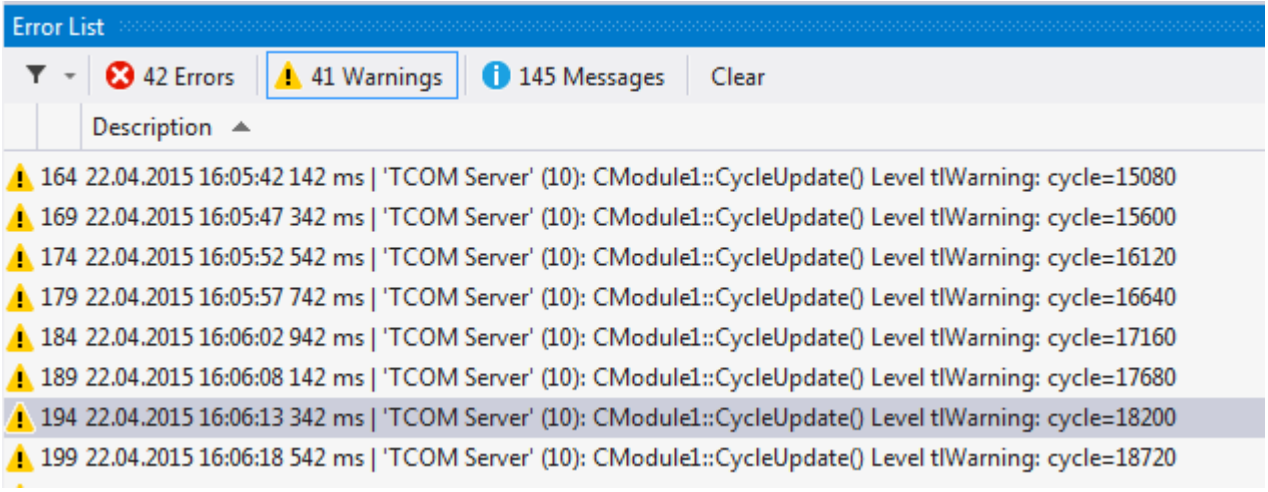
另外，您也可以在运行时更改跟踪级别，方法是转到实例，在 TraceLevelMax 参数的**值**处选择一个级别，右键单击第一列前面，然后选择**在线写入**。



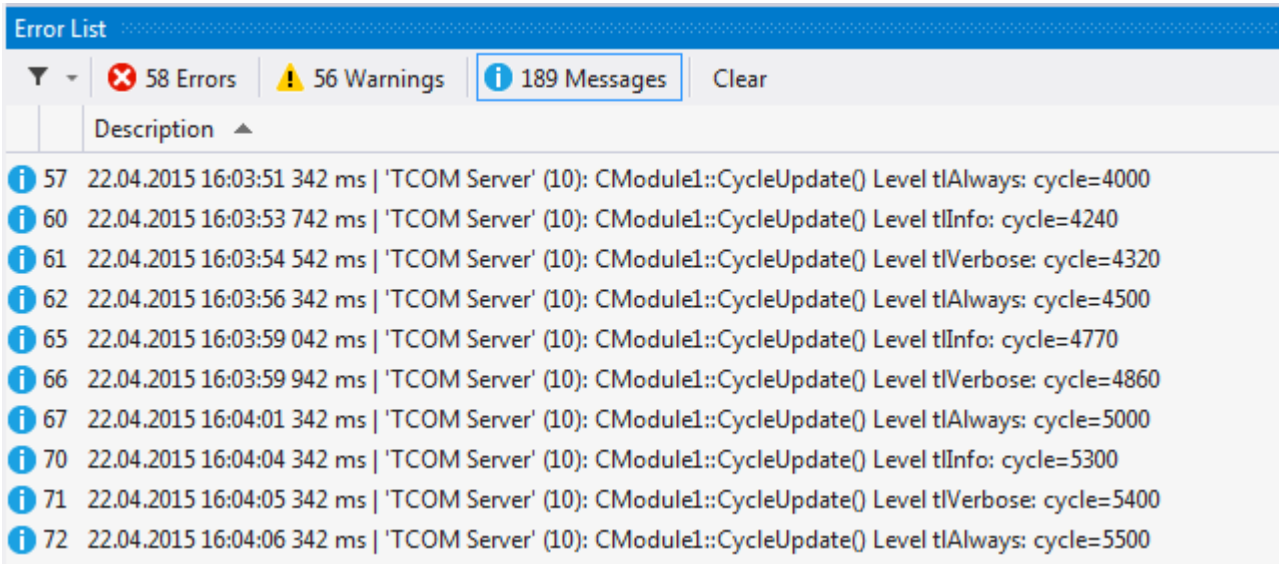
筛选消息类别

Visual Studio 错误列表可让您按类别对条目进行筛选。只需切换按键，便可独立启用或禁用**错误**、**警告**和**消息**三个类别。

在该截图中，仅启用**警告**，而禁用**错误**和**消息**：



在该截图中，仅启用**消息**显示，而禁用**错误**和**警告**的显示：



1 如何操作？

3

以下部分包含一组有关通用编程示例和 TwinCAT C++ 模块处理的常见问题。

13.1 使用自动化接口

自动化接口可用于 C++ 项目。

其中包括创建项目 [▶ 83]和使用向导来创建模块类 [▶ 83]。
此外，还可以设置项目属性，调用 TMC Code Generator 和发布模块。相应文档 [▶ 328]是自动化接口的一部分。

无论使用哪种编程语言，都可以访问、创建和处理 TcCOM 模块 [▶ 331]。

在此基础上，可以执行常见的系统管理器任务，如变量链接。

13.2 目标系统为 Windows 10，最大版本为 TwinCAT 3.1 Build 4022.2

对于 Windows 10 目标系统，传输的文件不能被覆盖，必须先重命名。

最大版本为 TwinCAT 3.1 Build 4022.2，必须在 **TMC 编辑器部署**中为此启用重命名目标 [▶ 127]选项。在之后的版本中，当目标系统将 Windows 10 用作操作系统时，此操作将自动完成。

13.3 在命令行上发布模块

通过以下调用，也可以从命令行启动 TwinCAT Engineering (XAE) 中的模块发布流程：

```
msbuild CppProject.vcxproj /t:TcPublishTMX
```

`CppProject.vcxproj` 参数必须根据现有项目文件进行调整。

输出结果将存入 C:\ProgramData\Beckhoff\TwinCAT\3.1\Repository 下的存储库（TwinCAT 3.1 Build 4024：C:\TwinCAT\3.1\Repository）。

13.4 克隆

运行时数据可以通过文件拷贝的方式从一台机器传输到另一台，前提是这两台机器来自同一平台，并与同等硬件设备连接。

以下步骤功能说明了将二进制配置从一台机器（“源”）传输到另一台机器（“目标”）的简单程序。

1. 清空源设备上的启动文件夹。
   ⇨ < TC3.1.4026.0: C:\TwinCAT\3.1\Boot
   ⇨ >=TC3.1.4026.0: C:\ProgramData\Beckhoff\TwinCAT\3.1\Boot
2. 在源机器上创建（或启用）模块。
3. 将启动文件夹从源机器转移到目标设备。
   该文件夹还包含含有必要 TMX 文件的存储库。该文件夹在源机器和目标设备上的存放路径如下所示。
   ⇨ < TC3.1.4026.0: C:\TwinCAT\3.1\Boot
   ⇨ >=TC3.1.4026.0: C:\ProgramData\Beckhoff\TwinCAT\3.1\Boot
4. 对于 TwinCAT 驱动程序项目 (.sys)：传输 MYDRIVER.sys 驱动程序，必要时也传输 PDB 文件。
   ⇨ < TC3.1.4026.0: C:\TwinCAT\3.1\Driver\AutoInstall\MYDRIVER.sys

⇨ >=TC3.1.4026.0: C:\ProgramData\Beckhoff\TwinCAT\3.1\Driver\AutoInstall\MYDRIVER.sys

5. 对于 TwinCAT 驱动程序项目 (.sys)，如果驱动程序是新安装在设备上的：
   TwinCAT 必须执行一次注册。通过 SysTray（右键单击 -> **系统** -> **启动/重启**）将 TwinCAT 切换到运行模式。
   或者也可以使用此指令（将“%1”替换为驱动程序名称）：

   ⇨ < TC3.1.4026.0：
   ```
   sc create %1 binPath= c:\TwinCAT\3.1\Driver\AutoInstall\%1.sys type=
   kernel start= auto group= "file system" DisplayName= %1 error= normal
   ```

   ⇨ >=TC3.1.4026.0：
   ```
   sc create %1 binPath= C:
   \ProgramData\Beckhoff\TwinCAT\3.1\Driver\AutoInstall\%1.sys type= kernel
   start= auto group= "file system" DisplayName= %1 error= normal
   ```

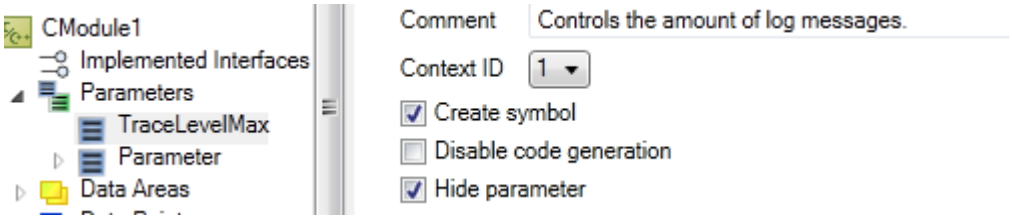
⇨ 现在可以启动目标机器。

处理授权

请注意，授权不能以这种方式转让。请使用预装授权、批量授权或其他方法提供授权。

13.5 通过 ADS 访问变量

如果 C++ 模块的变量在 TMC 编辑器中标记为“创建符号”，则可通过 ADS 访问这些变量：



供 ADS 访问的变量名称源自实例名称。

对于 TraceLevelMax 参数，可以是：

```
Untitled1_Obj1 (CModule1).TraceLevelMax
```

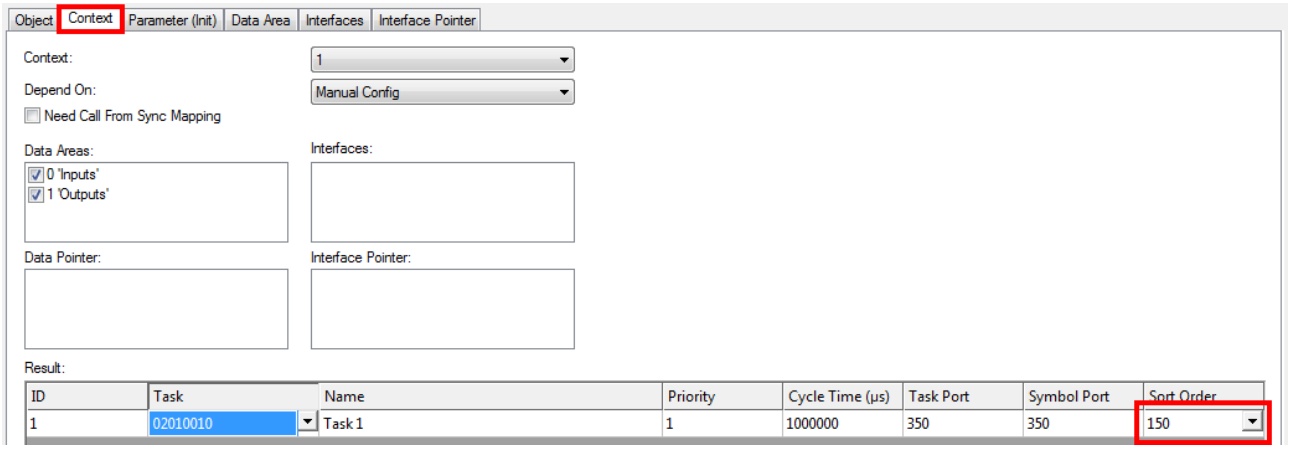
13.6 用于 C++ 模块的 TcCallAfterOutputUpdate

与 PLC 属性 TcCallAfterOutputUpdate 类似，C++ 模块可在输出更新后调用。
ITcPostCyclic [▶ 180] 接口的使用方法与 ITcCyclic [▶ 160] 接口相同。

13.7 确定任务的执行顺序

可以为一个任务分配不同的模块实例，因此用户需要一种机制来确定任务的执行顺序。

在 TwinCAT 模块实例配置器 [▶ 129] 的 **环境**中排序顺序 [▶ 131]下配置。



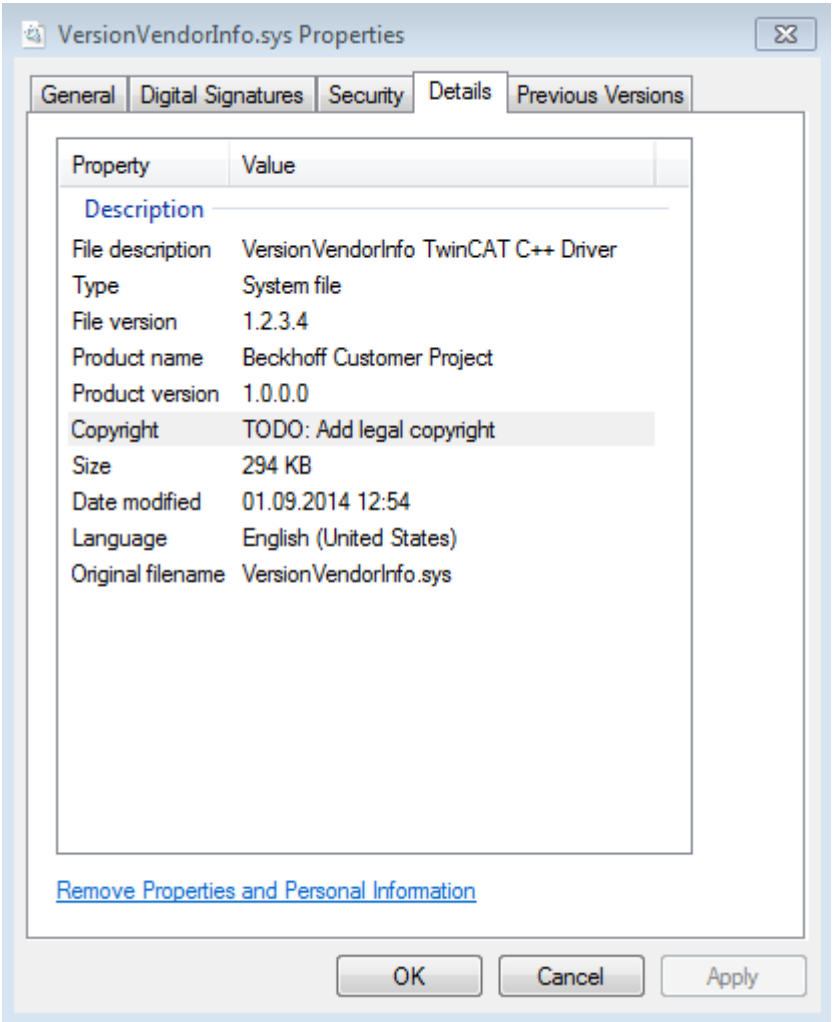
请参见 Sample26：任务的执行顺序 [▶ 296]，了解如何实现。

13.8 设置版本/供应商信息

Windows 提供了一种机构，用于查询在 .rc 文件编译过程中定义的供应商和版本资源。

Windows

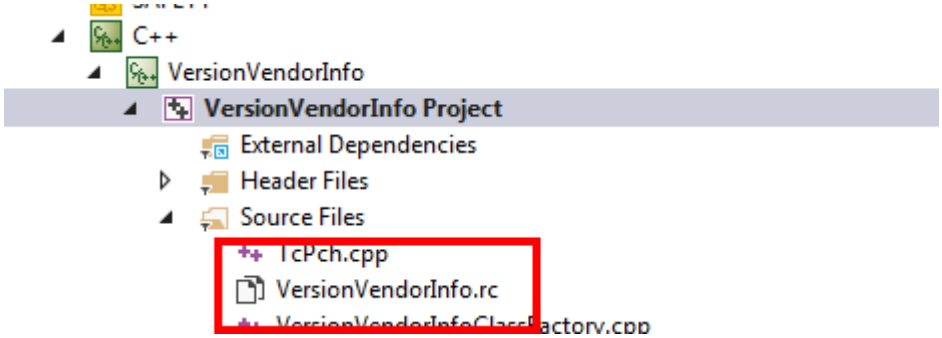
例如，可以通过每个属性文件的**详细信息**选项卡访问这些信息。



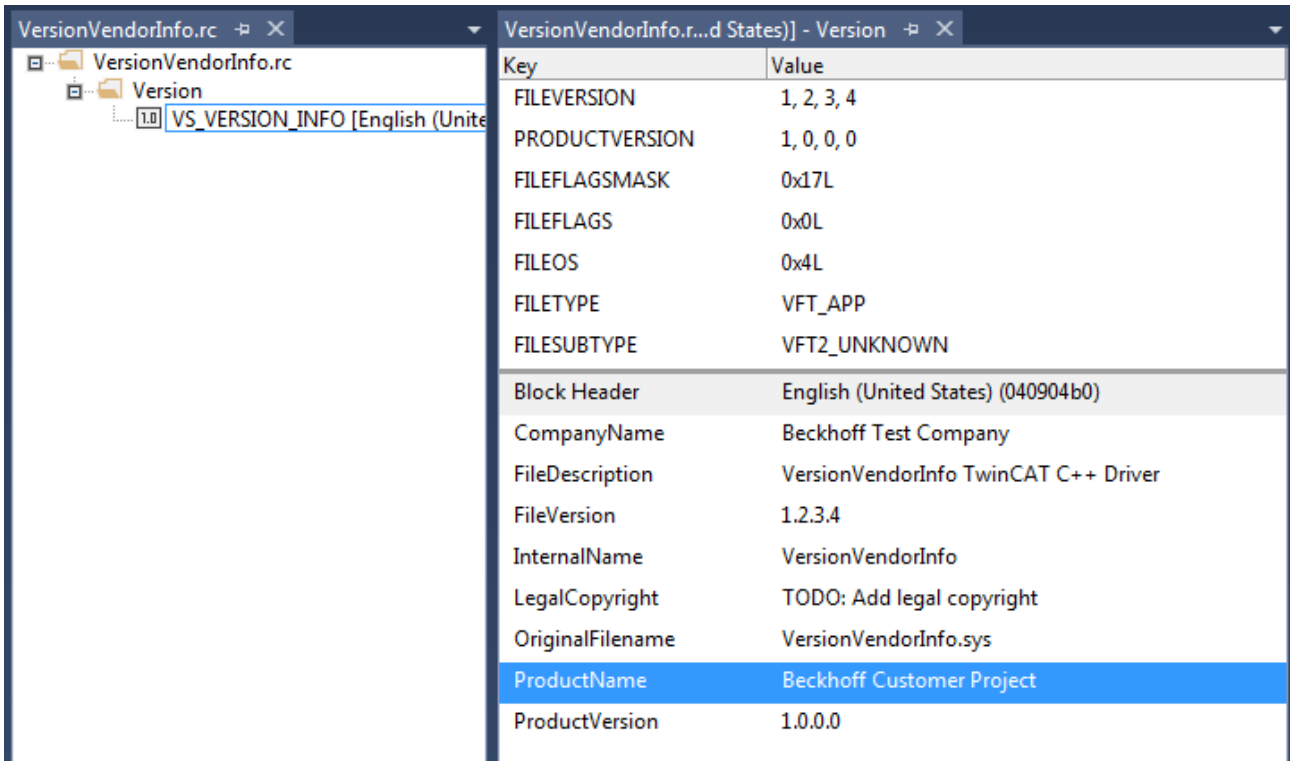
TwinCAT/BSD

文件中没有这些信息。

TwinCAT 通过常见的 Windows .rc 文件机构提供这种行为，这些文件是在创建 TwinCAT C++ 项目的过程中生成的。



使用资源编辑器编辑源文件文件夹中的 .rc 文件，以便定义这些属性：



13.9 重命名 TwinCAT C++ 项目

无法对 TwinCAT C++ 项目进行自动化重命名。

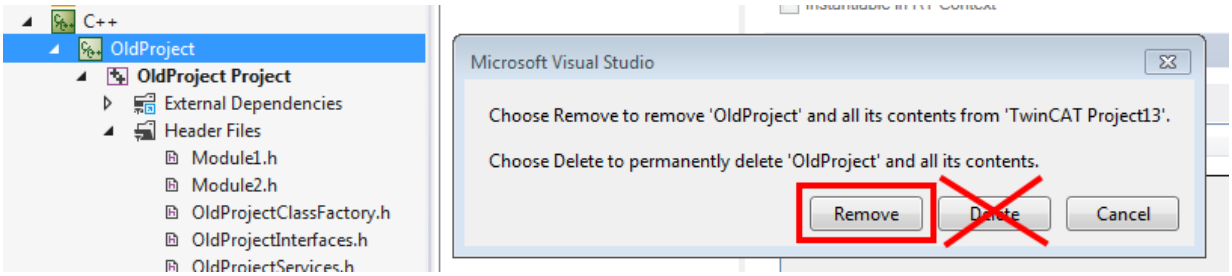
此时，将提供手动重命名项目的说明。

总之，C++ 项目会与其对应的文件一并完成重命名。

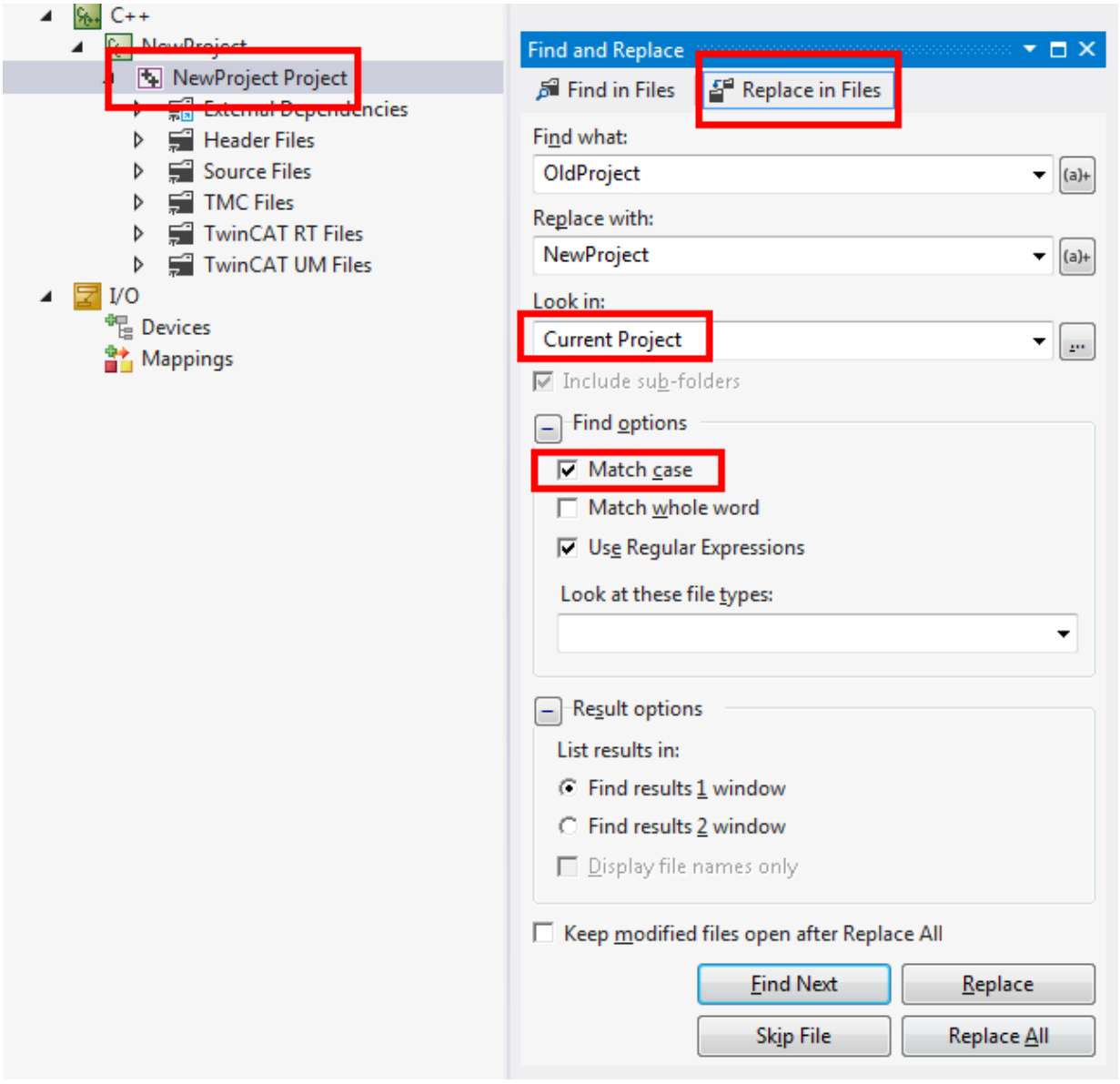
✓ 项目“OldProject”已经存在，并将更名为“NewProject”。

1. 如果项目中存在 TcCOM 实例，且需要保留其链接，则首先将其从项目中拖放到**系统** -> **TcCOM 对象**中。

2. 使用**删除**功能从 TwinCAT Solution 中删除旧项目。



3. 可以删除“OldProject”的汇编。为此，请删除“_Deployment”中相应的 .sys/.pdb 文件。
同时也可以删除现有的 .aps 文件。
4. 重命名 C++ 项目目录和项目文件（.vcxproj、.vcxproj.filters）。
如果使用版本管理，则必须通过版本管理系统进行重命名。
5. 如果存在 .vcvproj.user 文件，请检查其内容；该文件用于存储用户设置。如有必要，也可重命名该文件。
6. 打开 TwinCAT Solution。使用**添加现有项目**功能将重命名后的项目重新链接到 C++ 节点：导航至重命名后的子目录并选择其中的 .vcxproj 文件。
7. 将 ClassFactory、服务和接口以及头文件/源代码文件重命名为新项目名称。此外，还需重命名 TMC 文件以及项目文件夹“TwinCAT RT 文件”和“TwinCAT UM 文件”中的相应文件。
该重命名操作也应在版本管理系统中进行映射；如果版本管理系统未集成在 Visual Studio 中，则同时必须在版本管理系统中执行此步骤。替换源代码中出现的所有内容（区分大小写）：
“OLDPROJECT”变为“NEWPROJECT”，“OldProject”变为“NewProject”。
为此，请使用 Visual Studio 中的**查找和替换**对话框；注意必须选择解决方案资源管理器中的“NewProject 项目”。



⇨

注意

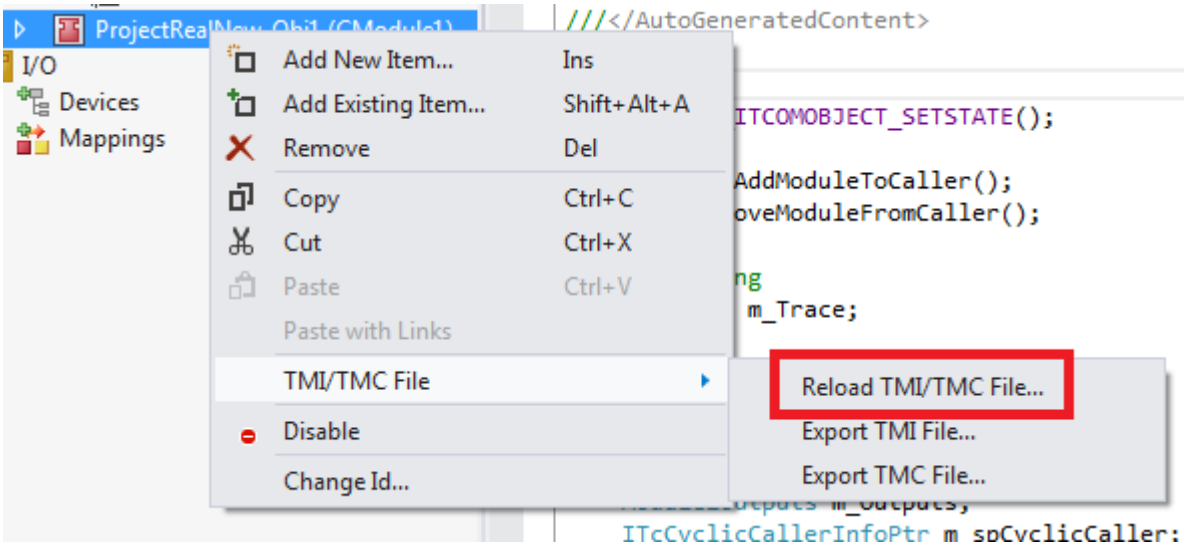
源代码不正确

对所有出现的字符串进行简单重命名可能会导致源代码不正确，例如在方法名称中使用项目名称。

- 如果可能出现这种情况，请单独进行重命名（**替换**而不是**全部替换**）。

如何构建项目：

1. A) 如果项目中的实例应该存在，请进行更新。为此，请右键单击实例，选择 **TTMI/TMC 文件** -> **重新加载 TMI/TMC 文件.....**，然后选择重命名的新 TMC 文件。



   B) 或者，右键单击 OTCID，通过**系统** -> **TcCOM 对象**和**项目对象**选项卡进行操作。

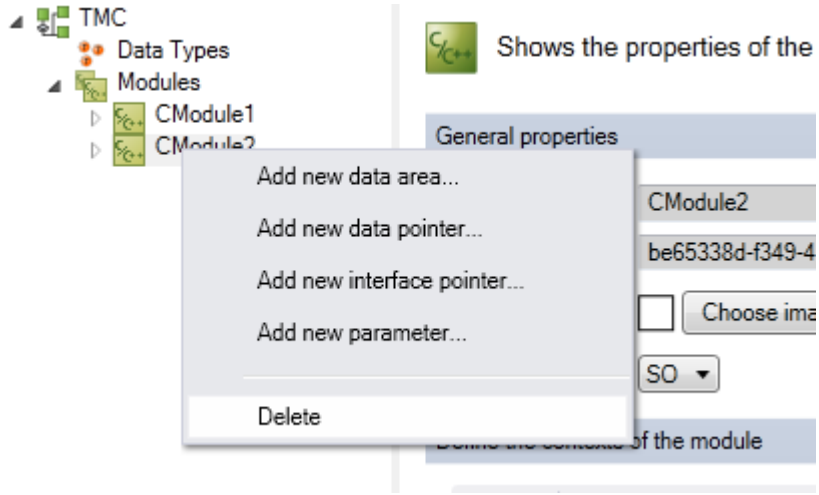
2. 将**系统** -> **TcCOM** 添加至项目中。
3. 清理目标系统。
   对于 TwinCAT C++ 驱动程序：删除 C:\TwinCAT\3.1\Driver\AutoInstall 中的“OldProject.sys/.pdb”文件。
   对于 TwinCAT Versioned C++ 项目：可以在 C:\TwinCAT\3.1\Repository 下清理存储库
4. 测试项目。

13.1 删除模块

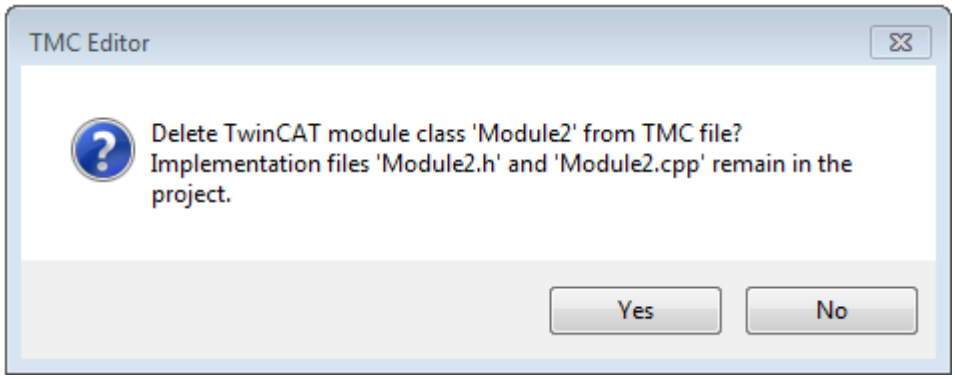
0

借助 TMC 编辑器，可以从 C++ 项目中删除 TwinCAT C++ 模块。

1. 右键单击模块（此处为 CModule2）
2. 选择**删除**。



3. 通过 TMC 确认删除。



4. 请注意，.cpp 和 .h 文件将被保留，如有必要，请手动删除。
删除其他相关组件（如报头文件、结构）。有关更多信息，请参见编译器错误信息。

13.1 随后添加版本控制和在线更改功能 1

带有 C++ TcCOM 模块的现有 C++ 驱动程序项目随后可迁移到版本控制的 C++ 项目中。

转换分几个步骤进行：

1. 点击环境菜单中的 **C++ 项目 TwinCAT Upgrade C+ 项目**，这是最便捷的选项。
或者，您也可以在此处找到手动说明：
将 C++ 项目转换为版本控制的 C++ 项目 [▶ 218]。
2. 然后，您便可以添加在线更改功能；必须手动执行操作：
使 C++ 模块类具备在线更改功能 [▶ 221]。

或者，也可以自行创建新的版本控制 C++ 项目，然后迁移差异内容。

13.11.1 C++ 项目 -> 版本控制

这些说明介绍了随后如何在 TwinCAT C++ 项目中添加版本控制。

要更改的代码以**粗体**格式存储在源代码中。

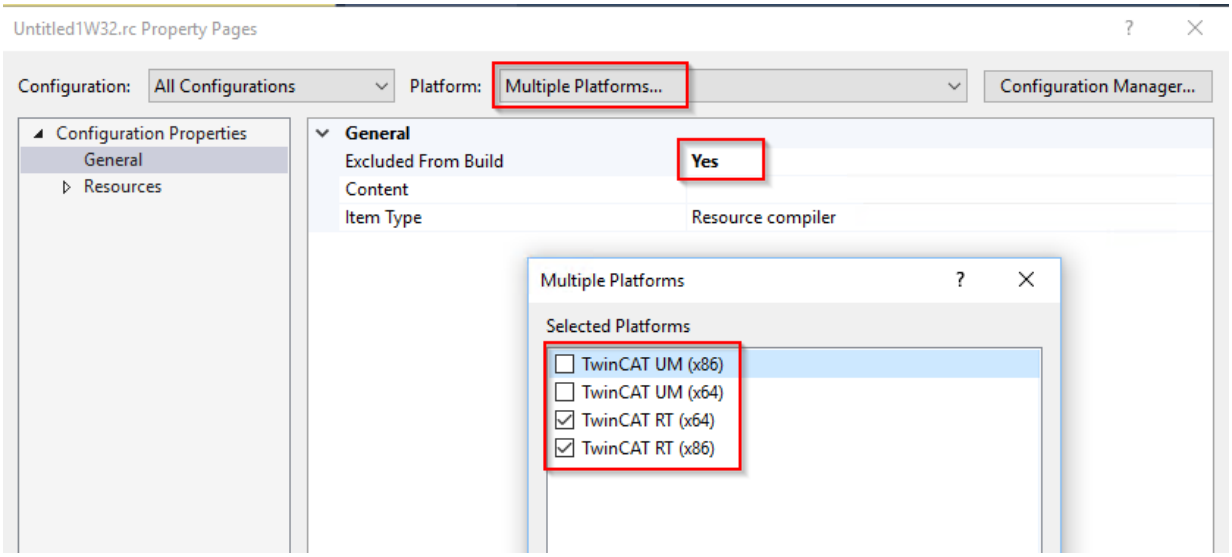
✓ C++ 项目。示例中使用“Untitled1”作为 C++ 项目名称。
此外，还要创建一个带有版本控制 C++ 项目的新空白项目。此为副本模板。

1. 用编辑器打开文件 Untitled1.vcxproj。
2. 添加以下内容：

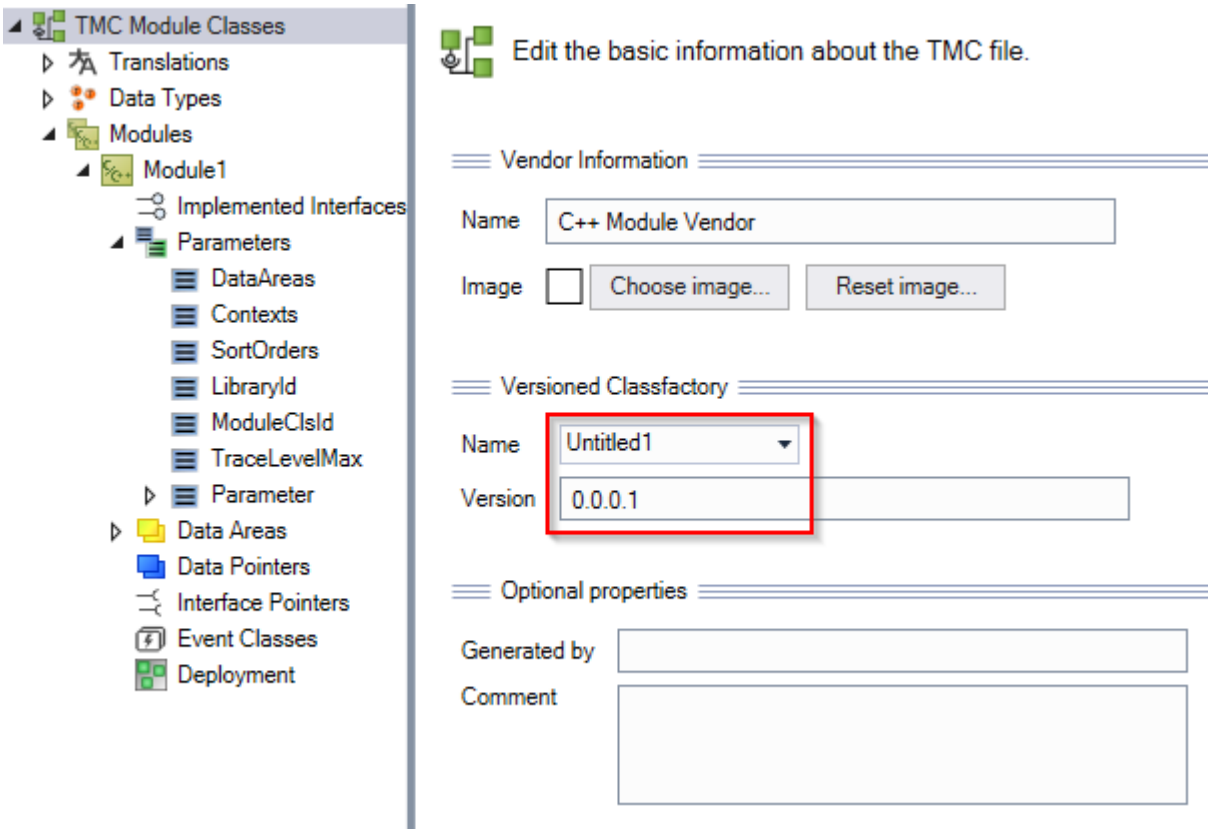
```
<PropertyGroup Label="Globals">
<ProjectGuid>{….}</ProjectGuid>
<RootNamespace>Untitled1</RootNamespace>
<Keyword>Win32Proj</Keyword>
<AutomaticRetargetPlatformVersion>true</AutomaticRetargetPlatformVersion>
</PropertyGroup>
  <PropertyGroup Label="TcGeneral">
<TcGeneralUseTmx>true</TcGeneralUseTmx>
</PropertyGroup>
<Import Project="$(VCTargetsPath)\Microsoft.Cpp.Default.props" />
```

3. 将新项目中的 Untitled1.rc 和 **Untitled1W32.rc** 文件转移到要迁移的项目中，覆盖 Untitled1.rc 文件。
4. 打开项目。
5. 在 **TwinCAT UM 文件** -> **环境菜单/添加现有项目**下，选择文件 Unititled1W32.rc，并将其添加到项目中。

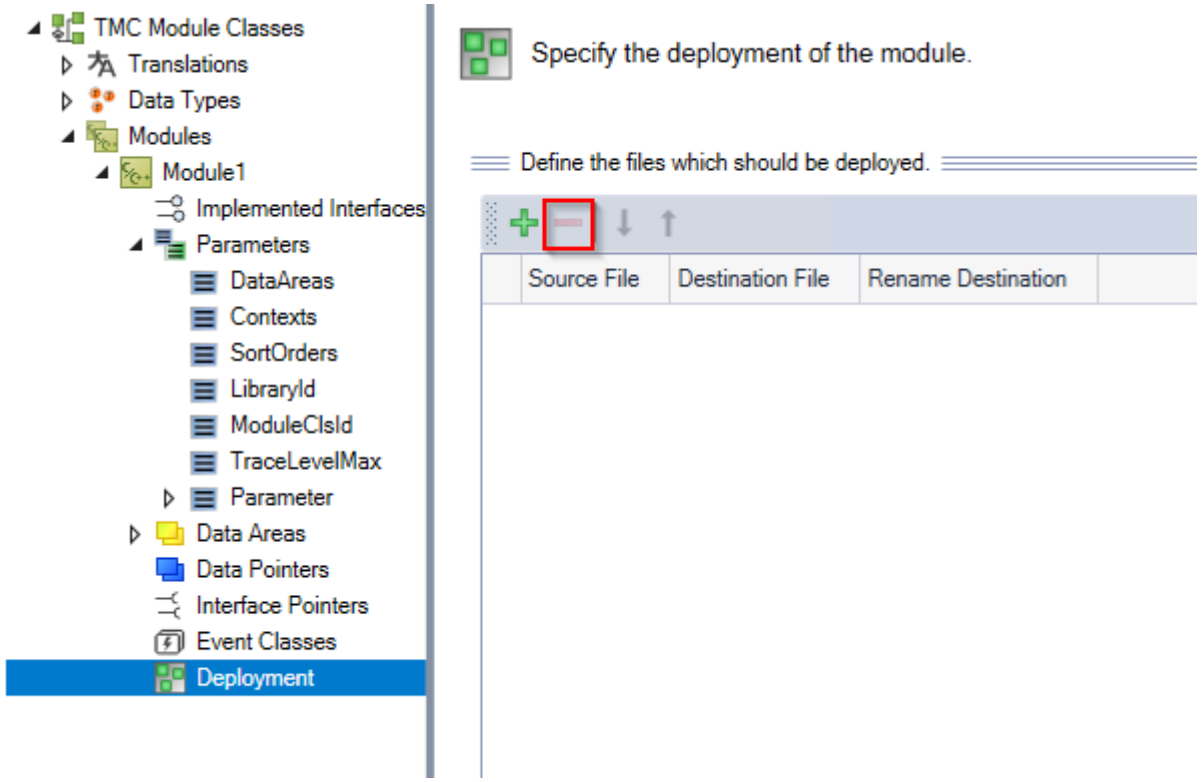
6. 只有在 TwinCAT UM 平台上才需要构建该文件，因此在构建过程中应将其排除在外。可以通过右键单击和属性来执行操作：



7. 打开 TMC 编辑器。
8. 将类工厂的名称和版本设为“0.0.0.1”。



9. 在**部署**下，删除所有条目。



10. 更改报头 Modul1.h DECLARE_IPERSIST_**LIB**()：

```
public:
DECLARE_IUNKNOWN()
DECLARE_IPERSIST_LIB()
DECLARE_ITCOMOBJECT_LOCKOP()
```

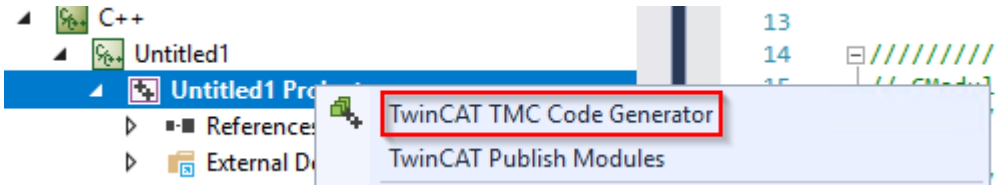
11. 在源代码中添加 Modul1.cpp：

```
#pragma hdrstop
#include "Module1.h"
#include "Untitled1Version.h"

#ifdef _DEBUG
```

12. 在源代码中添加 Modul1.cpp：

```
END_INTERFACE_MAP()

IMPLEMENT_IPERSIST_LIB(CModule1, VID_Untitled1,
CID_Untitled1CModule1)
IMPLEMENT_ITCOMOBJECT(CModule1)
IMPLEMENT_ITCOMOBJECT_SETSTATE_LOCKOP2(CModule1)
```

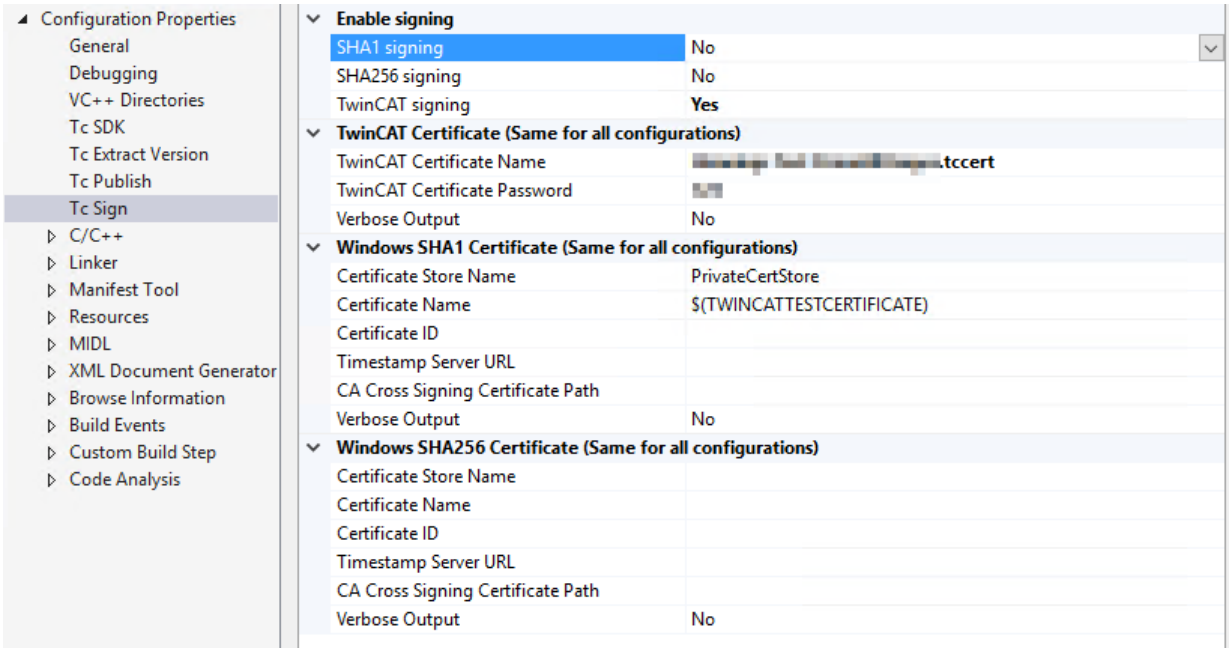
13. 调用代码生成器。



14. 更改文件 Untitled1Classfactory.cpp：

```
CUntitled1ClassFactory::CUntitled1ClassFactory() : CObjClassFactory()
{
TcDbgUnitSetImageName(TCDBG_UNIT_IMAGE_NAME_TMX(SRVNAME_UNTITLED1));
#if defined(TCDBG_UNIT_VERSION)
TcDbgUnitSetVersion(TCDBG_UNIT_VERSION(Untitled1));
#endif //defined(TCDBG_UNIT_VERSION)
}
```

15. 在 Tc 签名 [▶ 144]选项卡的项目属性中更改签名。为此，请关闭 **SHA1 签名**，并打开 **TwinCAT 签名**；同时提供 TwinCAT 用户证书和密码。



16. 触发项目**重建**。

⇨ 结果为支持版本控制的 C++ 项目。

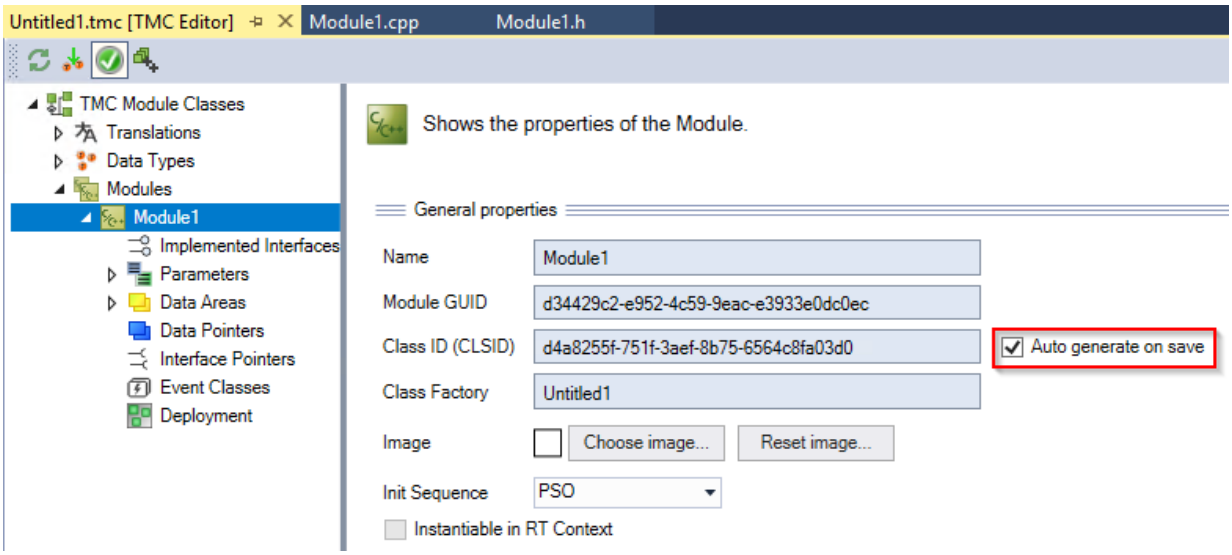
如果要使模块具备“在线更改”功能，可通过以下说明 [▶ 221]实现。

13.11.2 C++ 模块 -> OnlineChange

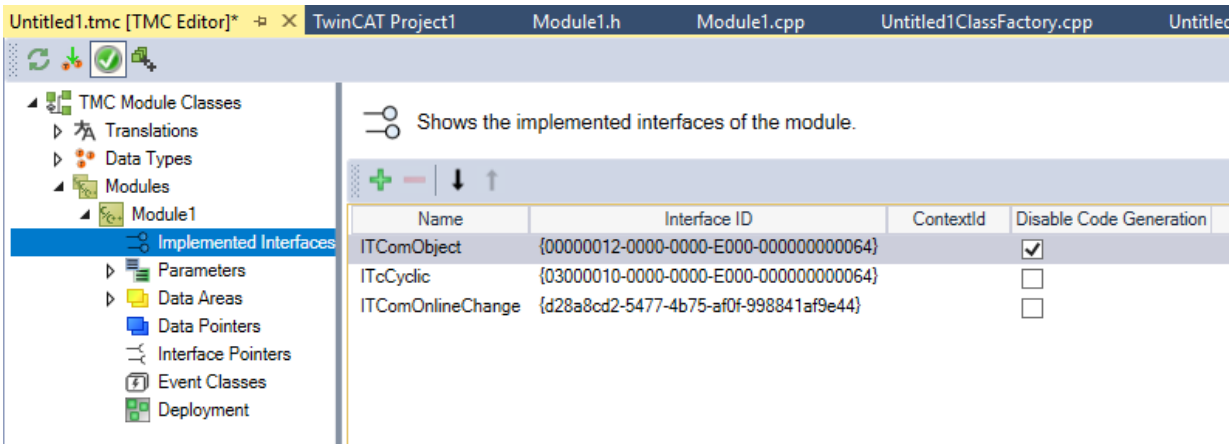
本说明介绍了如何在版本控制的 TwinCAT C++ 项目中使模块具备 OnlineChange 功能。

✓ 带有 C++ 模块的版本控制 C++ 项目，尚未具备“在线更改”功能。在本示例中，假定模块为“Module1”。此外，还可以创建一个带有“在线更改”功能模块的新空白项目，从而更容易地采用更改。

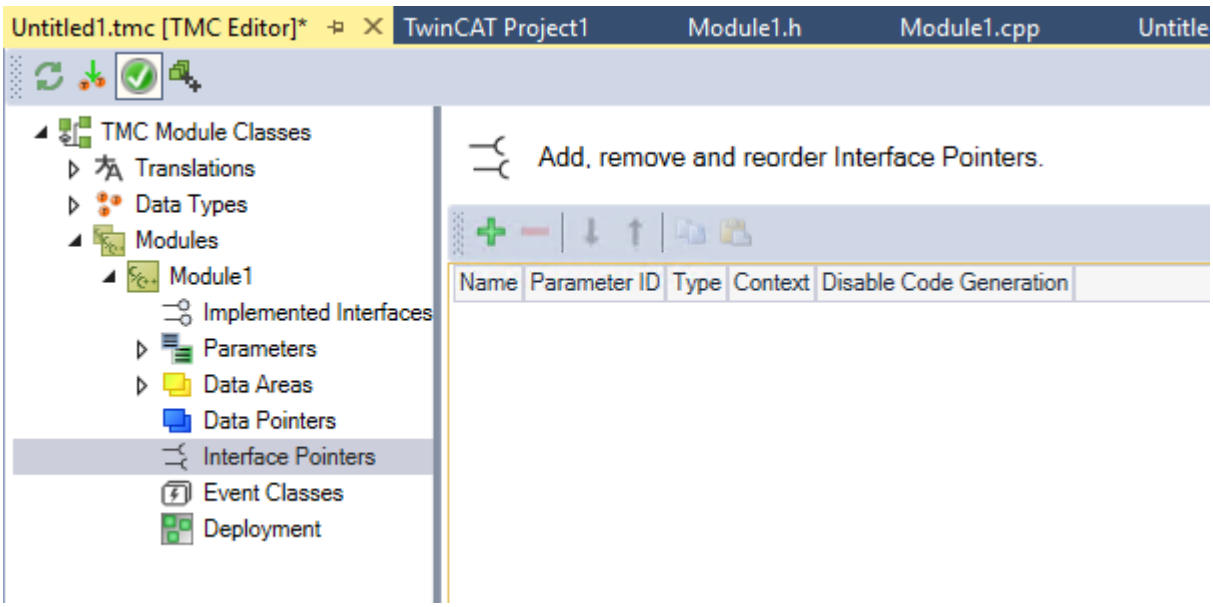
1. 打开项目和 TMC 编辑器。
2. 为模块设置**保存时自动生成**选项，以便自动更改 ClassID。



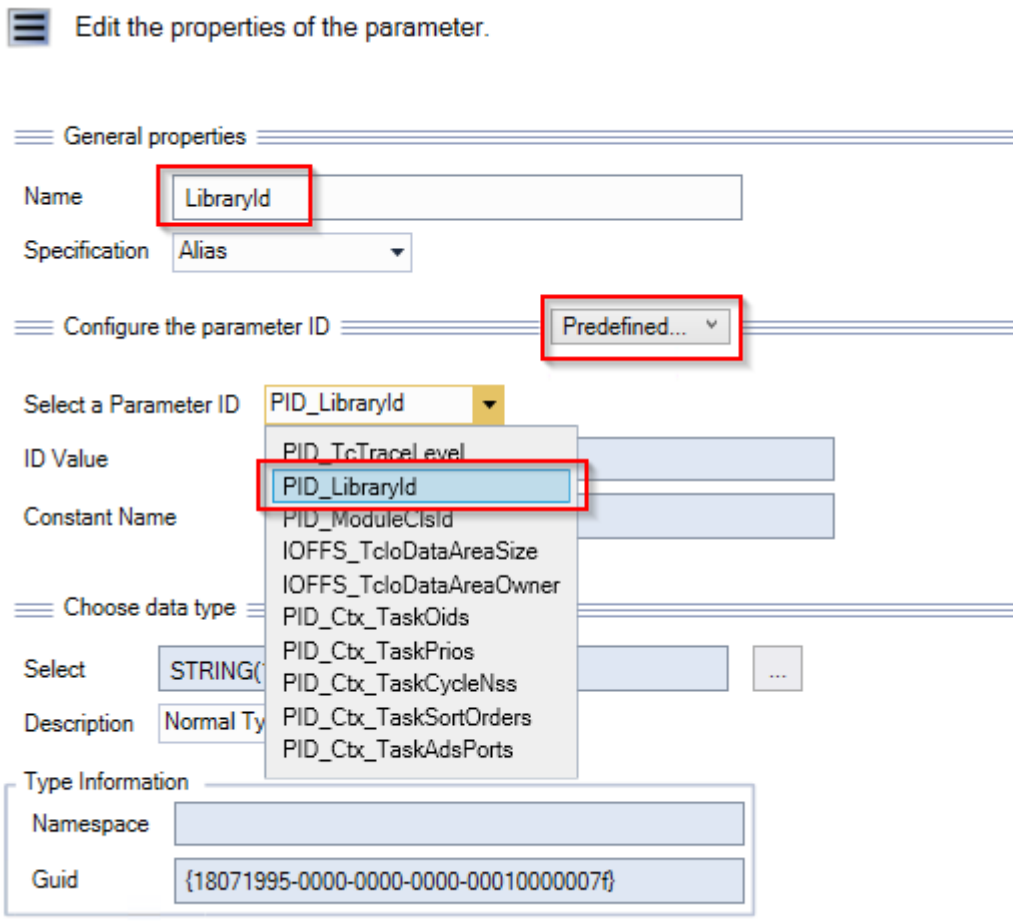
3. 在**已实现接口**下，删除接口 ITcADI 和 ITcWatchsource，并添加 ITComOnlineChange。



4. 删除**接口指针**下的 CyclicCaller。

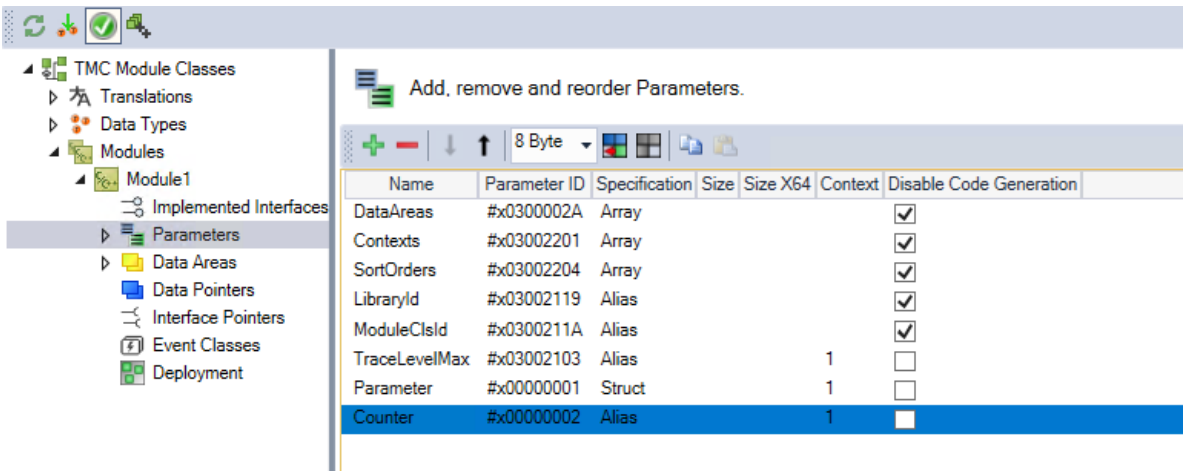


5. 在**参数**下，必须添加某些预定义参数。
   按下**参数**下的 **+**，并选择**预定义**以选择预定义的 ParameterID：

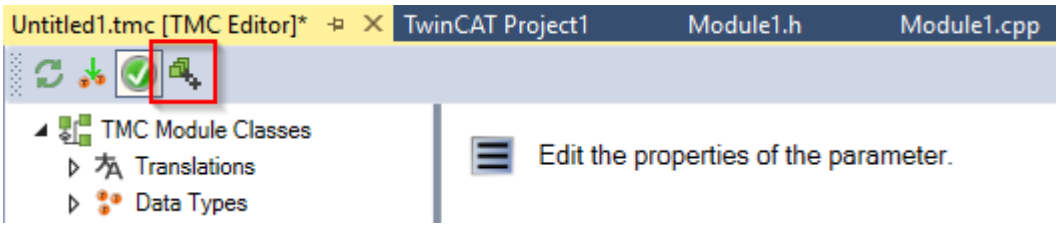


您可以用给定的名称创建下列参数，每个参数都无需生成代码：
“PID_LibraryID”，名称为“LibraryID”
“PID_ModuleClsId”，名称为“ModuleClsId”
“PID_Ctx_TaskSortOrders”，名称为“SortOrders”
“PID_Ctx_TaskOids”，名称为“Contexts”
“IOFFS_TcIoDataAreaSize”，名称为“DataAreas”

⇨ 结果见概述：



6. 启动代码生成。



7. 必须对模块标题“Module1.h”进行一些修改。首先，删除两处不再需要的接口声明和相应的映射。

```
class CModule1
: public ITComObject
, publicITcADI
, publicITcWatchSource
///<AutoGeneratedContent id="InheritanceList">
, public ITcCyclic
///</AutoGeneratedContent>
{
public:
DECLARE_IUNKNOWN()
DECLARE_IPERSIST(CID_Untitled1CModule1)
DECLARE_ITCOMOBJECT_LOCKOP()
DECLARE_ITCADI()
DECLARE_ITCWATCHSOURCE()
DECLARE_OBJPARAWATCH_MAP()
DECLARE_OBJDATAAREA_MAP()
```

8. 在头文件中创建新的成员变量：

```
///</AutoGeneratedContent>
ITcADIPtr m_spADI;
// TODO: Custom variable
```

9. 需要对模块“Module1.cpp”的源代码进行一些修改。首先，删除两处不再需要的接口实现。

```
BEGIN_INTERFACE_MAP(CModule1)
INTERFACE_ENTRY_ITCOMOBJECT()
INTERFACE_ENTRY(IID_ITcADI, ITcADI)
INTERFACE_ENTRY(IID_ITcWatchSource, ITcWatchSource)
///<AutoGeneratedContent id="InterfaceMap">
INTERFACE_ENTRY(IID_ITcCyclic, ITcCyclic)
///</AutoGeneratedContent>
END_INTERFACE_MAP()
IMPLEMENT_ITCOMOBJECT(CModule1)
IMPLEMENT_ITCOMOBJECT_SETSTATE_LOCKOP2(CModule1)
IMPLEMENT_ITCADI(CModule1)
IMPLEMENT_ITCWATCHSOURCE(CModule1)
```

10. 删除属于已删除接口的映射的实现：

```
BEGIN_SETOBJPARA_MAP(CModule1)
SETOBJPARA_DATAAREA_MAP()
///<AutoGeneratedContent id="SetObjectParameterMap">
SETOBJPARA_VALUE(PID_TcTraceLevel, m_TraceLevelMax)
SETOBJPARA_VALUE(PID_Module1Parameter, m_Parameter)
SETOBJPARA_ITFPTR(PID_Ctx_TaskOid, m_spCyclicCaller)
///</AutoGeneratedContent>
END_SETOBJPARA_MAP()
///////////////////////////////////////////////////////////////////////////////
// Get parameters of CModule1
BEGIN_GETOBJPARA_MAP(CModule1)
GETOBJPARA_DATAAREA_MAP()
///<AutoGeneratedContent id="GetObjectParameterMap">
GETOBJPARA_VALUE(PID_TcTraceLevel, m_TraceLevelMax)
GETOBJPARA_VALUE(PID_Module1Parameter, m_Parameter)
GETOBJPARA_ITFPTR(PID_Ctx_TaskOid, m_spCyclicCaller)
///</AutoGeneratedContent>
END_GETOBJPARA_MAP()
///////////////////////////////////////////////////////////////////////////////
// Get watch entries of CModule1
BEGIN_OBJPARAWATCH_MAP(CModule1)
OBJPARAWATCH_DATAAREA_MAP()
///<AutoGeneratedContent id="ObjectParameterWatchMap">
///</AutoGeneratedContent>
END_OBJPARAWATCH_MAP()
///////////////////////////////////////////////////////////////////////////////
// Get data area members of CModule1
BEGIN_OBJDATAAREA_MAP(CModule1)
///<AutoGeneratedContent id="ObjectDataAreaMap">
OBJDATAAREA_VALUE(ADI_Module1Inputs, m_Inputs)
```

```
OBJDATAAREA_VALUE(ADI_Module1Outputs, m_Outputs)
///</AutoGeneratedContent>
END_OBJDATAAREA_MAP()
```

11. 必须在状态机中的 P->S 转换中获取 m_spAPI 指针：

```
HRESULT CModule1::SetObjStatePS(PTComInitDataHdr pInitData)
{
m_Trace.Log(tlVerbose, FENTERA);
HRESULT hr = S_OK;
IMPLEMENT_ITCOMOBJECT_EVALUATE_INITDATA(pInitData);
// query TcCOM object server for ITcADI interface with own object id,
// which retrieves a reference to the TMC module instance handler
m_spADI.SetOID(m_objId);
hr = FAILED(hr) ? hr : m_spSrv->TcQuerySmartObjectInterface(m_spADI);
// TODO: Add initialization code
```

12. 在状态机中的 S->O 转换中添加以下内容，省略对 AddModuleToCaller 或 RemoveModuleFromCaller 的调用。

```
HRESULT hr = S_OK;
// Retrieve pointer to data areas via ITcADI interface from TMC module handler
///<AutoGeneratedContent id="DataAreaPointerInitialization">
///</AutoGeneratedContent>
// TODO: Add any additional initialization
// Cleanup if transition failed at some stage
if ( FAILED(hr) )
{
    SetObjStateOS();
}
```

13. 在状态机中的 O->S 转换中添加以下内容，省略 RemoveModuleFromCaller 调用：

```
HRESULT hr = S_OK;
// Release pointer to data areas via ITcADI interface from TMC module handler
///<AutoGeneratedContent id="DataAreaPointerRelease">
///</AutoGeneratedContent>
// TODO: Add any additional deinitialization
```

14. AddModuleToCaller 和 RemoveModuleFromCaller 方法不是必需的，可以删除。
15. 更改对数据区的访问权限：

```
HRESULT CModule1::CycleUpdate(ITcTask* ipTask, ITcUnknown* ipCaller, ULONG_PTR context)
{
  HRESULT hr = S_OK;
  // TODO: Replace the sample with your cyclic code
  m_Counter+=m_pInputs->Value;
  m_pOutputs->Value=m_Counter;
  return hr;
}
```

16. 实现 ITcOnlineChange。这些功能由上一代的代码生成所创建，但不得返回 NOTIMPL。

```
///<AutoGeneratedContent id="ImplementationOf_ITComOnlineChange">
//////////////////////////////////////////////////////////////////////////////
// PrepareOnlineChange is called after this instance has been set to PREOP in non RT context.
// Parameter ipOldObj refers to the currently active instance which is still in OP.
// Retrieve parameter values that are not changed during OP via ipOldObj here.
//
// Parameter pOldInfo refers to instance data which includes the libraryId and
// the module class id. This information can be used to implement switch from one
// specific version to another.
HRESULT CModule1::PrepareOnlineChange(ITComObject* ipOldObj, TmcInstData* pOldInfo)
{
HRESULT hr = S_OK;

ULONG nData = sizeof(m_Parameter);
PVOID pData = &m_Parameter;
ipOldObj->TcGetObjPara(PID_Module1Parameter, nData, pData);
return hr;
}
//////////////////////////////////////////////////////////////////////////////
// PerformOnlineChange is called after this instance has been set to SAFEOP in RT context.
// Parameter ipOldObj refers to old instance which is now in SAFEOP.
// Allows to retrieve data after the last cyclic update of the old instance and
// before the first cyclic update of this instance.
HRESULT CModule1::PerformOnlineChange(ITComObject* ipOldObj, TmcInstData* pOldInfo)
{
HRESULT hr = S_OK;

ULONG nData = sizeof(m_Counter);
PVOID pData = &m_Counter;
ipOldObj->TcGetObjPara(PID_Module1Counter, nData, pData);
return hr;
}
///</AutoGeneratedContent>
```

17. 再次启动 TMC Code Generator，生成用于数据区指针初始化的代码。

13.1 TMC 成员变量初始化 2

TcCOM 模块的所有成员变量都必须进行初始化。TMC Code Generator 支持以下功能：

```
///<AutoGeneratedContent id="MemberInitialization">
```

TMC Code Generator 将其替换为：

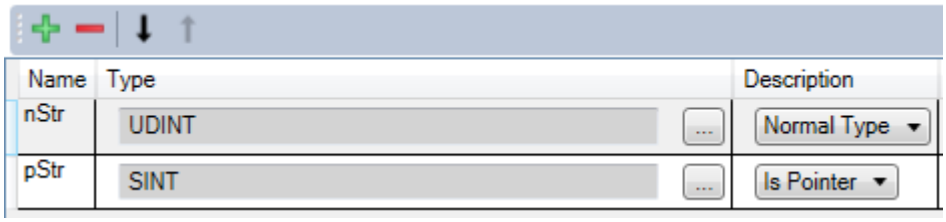
```
 ///<AutoGeneratedContent id="MemberInitialization">
m_TraceLevelMax = tlAlways;
memset(&m_Parameter, 0, sizeof(m_Parameter));
memset(&m_Inputs, 0, sizeof(m_Inputs));
memset(&m_Outputs, 0, sizeof(m_Outputs));
///</AutoGeneratedContent>
```

在 TwinCAT 3.1 Build 4018 之前使用 TwinCAT C++ Wizard 生成的项目不使用此属性，但可以通过在相应代码（如构造函数）中插入此行来轻松调整：

```
///<AutoGeneratedContent id="MemberInitialization">
```

13.1 使用 PLC 字符串作为方法参数 3

要将字符串作为方法参数从 PLC 传输到 C++，在 TMC 中声明方法时应使用带有长度信息的指针：



这种方法可以通过在封装功能块中实现一个方法来调用。

```
1  METHOD SetString : HRESULT
2  VAR_INPUT
3      sSent : STRING(80);
4  END_VAR
```

```
1  IF (ipStateMachine <> 0) THEN
2      SetString := ipStateMachine.SetString(SIZEOF(sSent),ADR(sSent));
3  END_IF
```

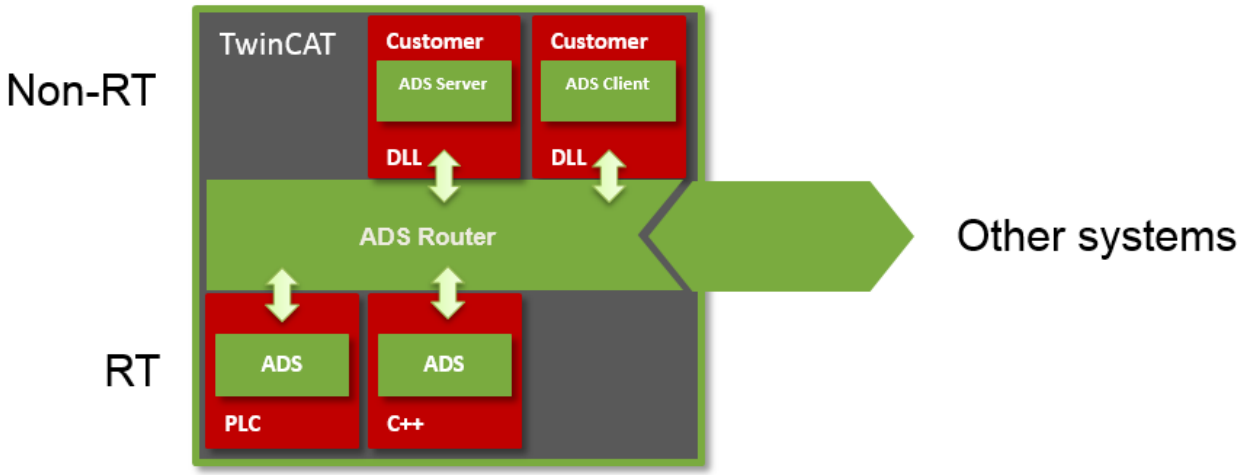
原因在于这两个领域对方法参数的处理方式不同：

- PLC：对 STRING(nn) 数据类型采用值传递方式。
- TwinCAT C++ (TMC)：使用引用传递。

13.1 第三方库 4

内核模式下的 C/C++ 代码无法关联或执行第三方开发的、专为用户模式（User Mode）设计的库。因此，无法在 TwinCAT C++ 模块中直接使用 DLL。

TwinCAT 3 实时环境的连接可以通过 ADS 通信实现。您可以利用通过 ADS 提供 TwinCAT 功能的第三方库来执行 Usermode 应用程序。



Usermode 中 ADS 组件的这一操作既可以作为客户端进行（即该动态链接库会在需要时向 TwinCAT 实时系统传输数据），也可以作为服务器进行（即 TwinCAT 实时在必要时从 Usermode 中获取数据）。

Usermode 下的此类 ADS 组件也可以通过 PLC 以同样的方式使用。此外，ADS 还能突破设备限制进行通信。

以下示例说明了 ADS 在 C++ 模块中的使用情况：

Sample03：C++ 用作 ADS 服务器 [▶ 238]

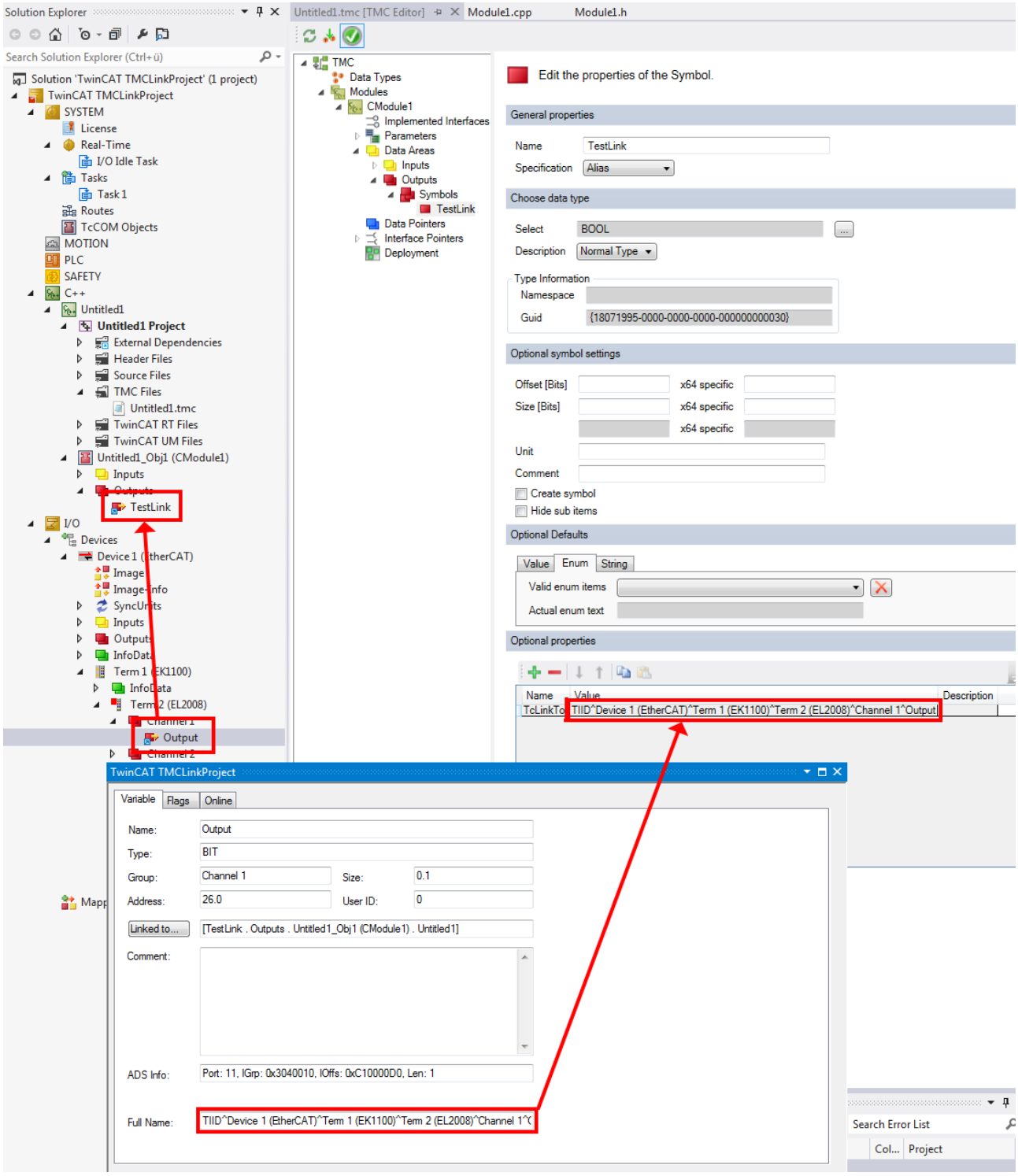
Sample07：接收 ADS 通知 [▶ 252]

Sample08：提供 ADS-RPC [▶ 253]

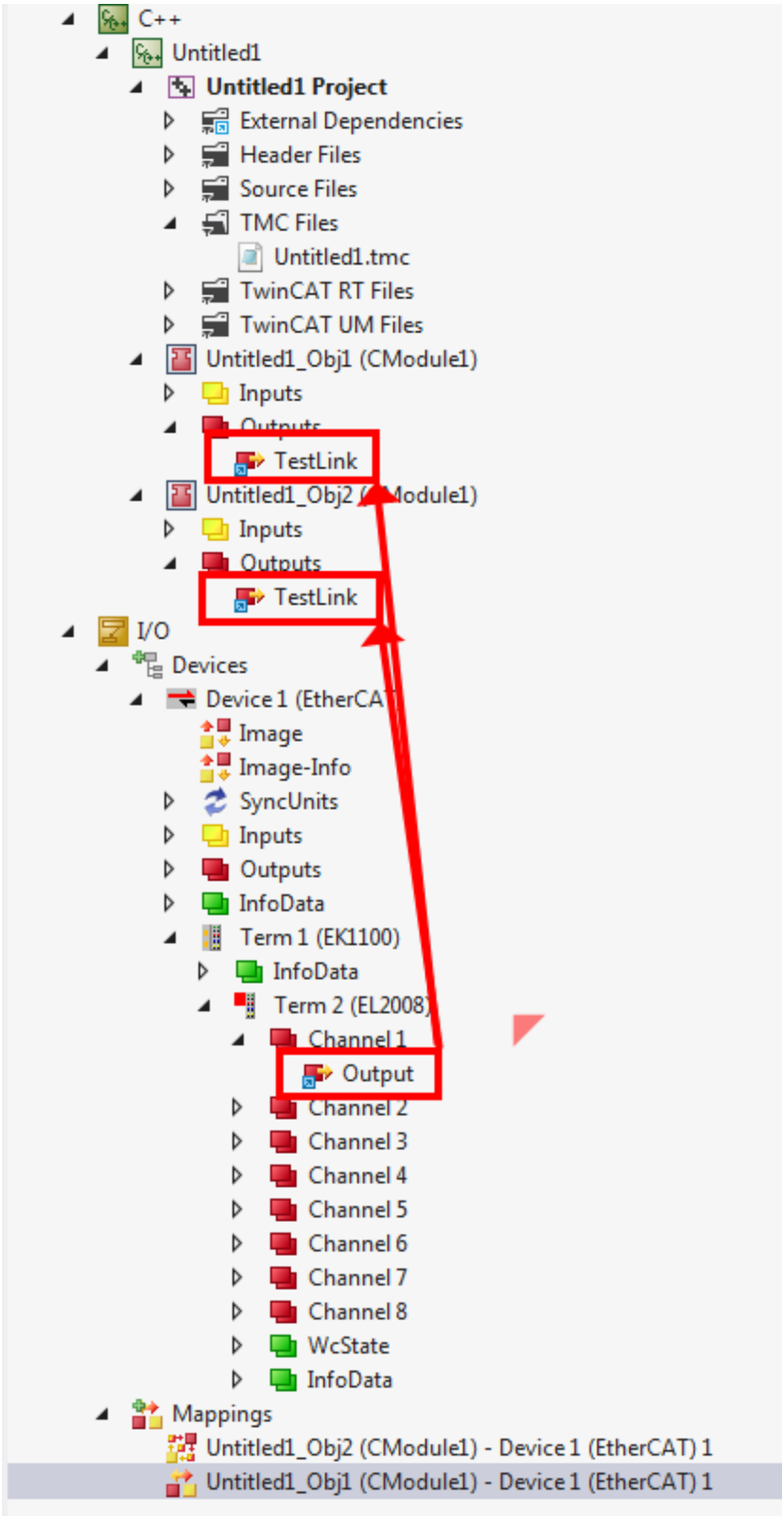
13.1 通过 TMC 编辑器进行关联 (TcLinkTo)

5

与 PLC 类似，在 TwinCAT C++ 中，例如与硬件的链接可以在编码时预先定义。
可以在要链接的符号处的 TMC 编辑器中完成操作。指定一个具有目标值的 **TcLinkTo** 属性。下方截图说明了这一点：



请注意，这样的指令适用于模块的所有实例：



13.1 根据 ADS 进行在线更改

6

在线更改 [▶ 148]通常由 XAE Engineering 执行。然而，也可以通过 ADS 执行“在线更改”，从而在自身程序中进行更改：

- ADS WRITE
- AmsPort：11

- IdxGrp：ObjectID
- IdxOffset：16#03002119 (PID_LibraryId)
- Daten：Library Id (“<vendor>|<library>|<version>”)

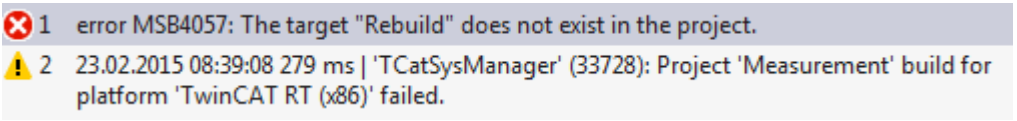
1 故障排除

4

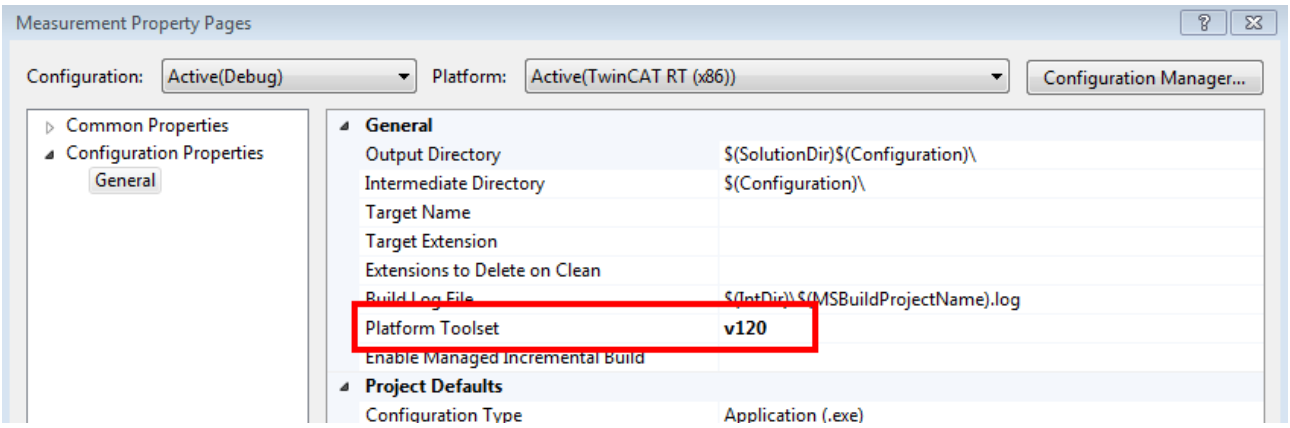
此为在处理 TwinCAT C++ 模块过程中的陷阱和故障列表。

14.1 构建 – “项目中不存在目标.....”

特别是在将 TwinCAT Solution 从一台机器传输到另一台机器时，Visual Studio 可能会显示错误信息，大意是项目中不存在所有目标（如“构建”、“重建”、“清理”）。

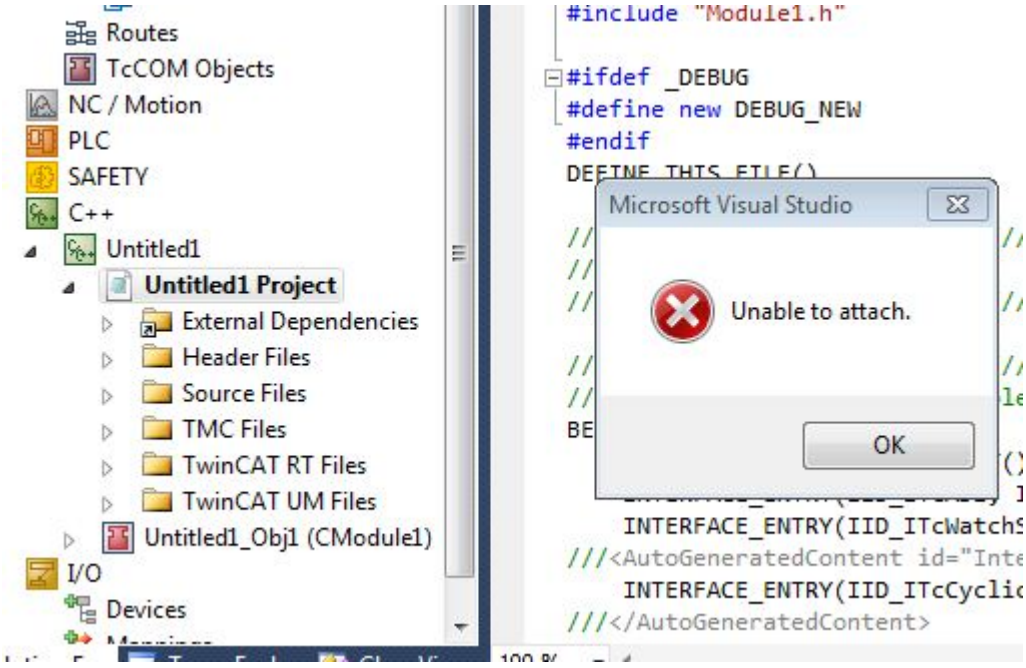


检查 C++ 项目“平台工具集”的配置。如果解决方案从一个 Visual Studio 版本迁移到另一个版本，可能需要重新配置：

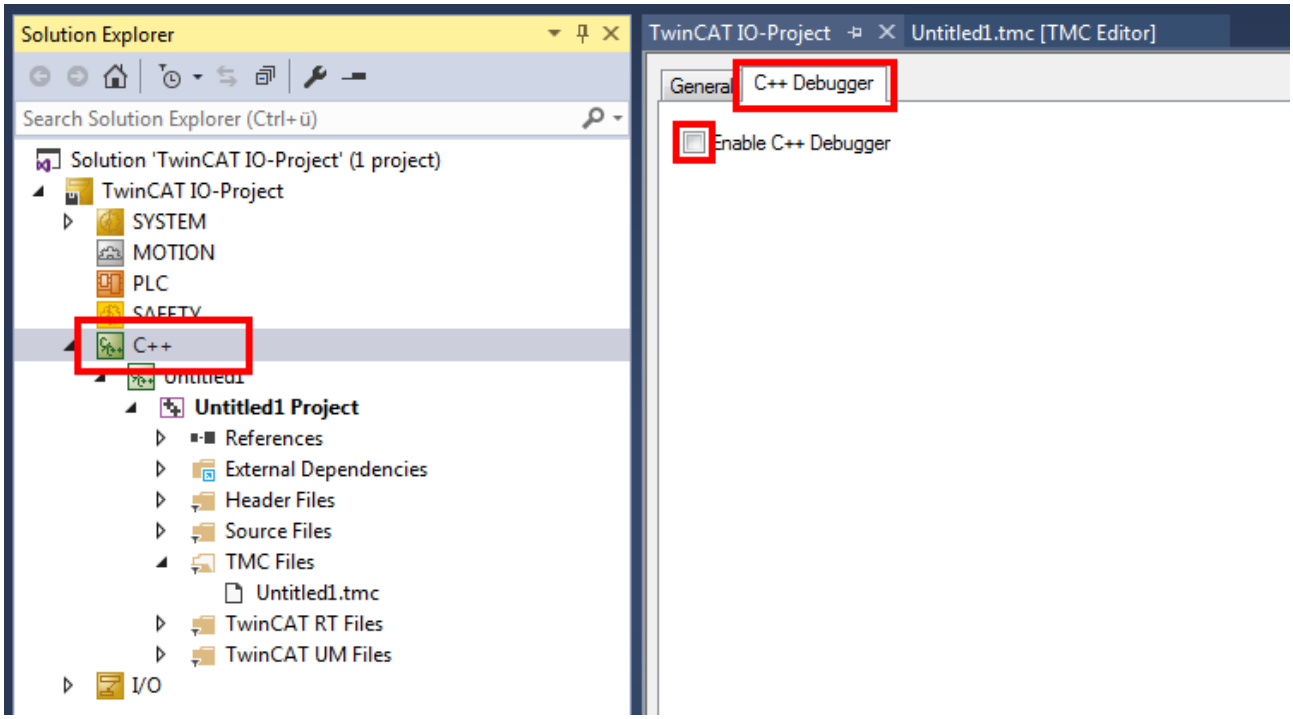


14.2 调试 – “无法附加”

如果在启动调试器调试 TwinCAT C++ 项目时出现此错误信息，则说明缺少一个配置步骤：

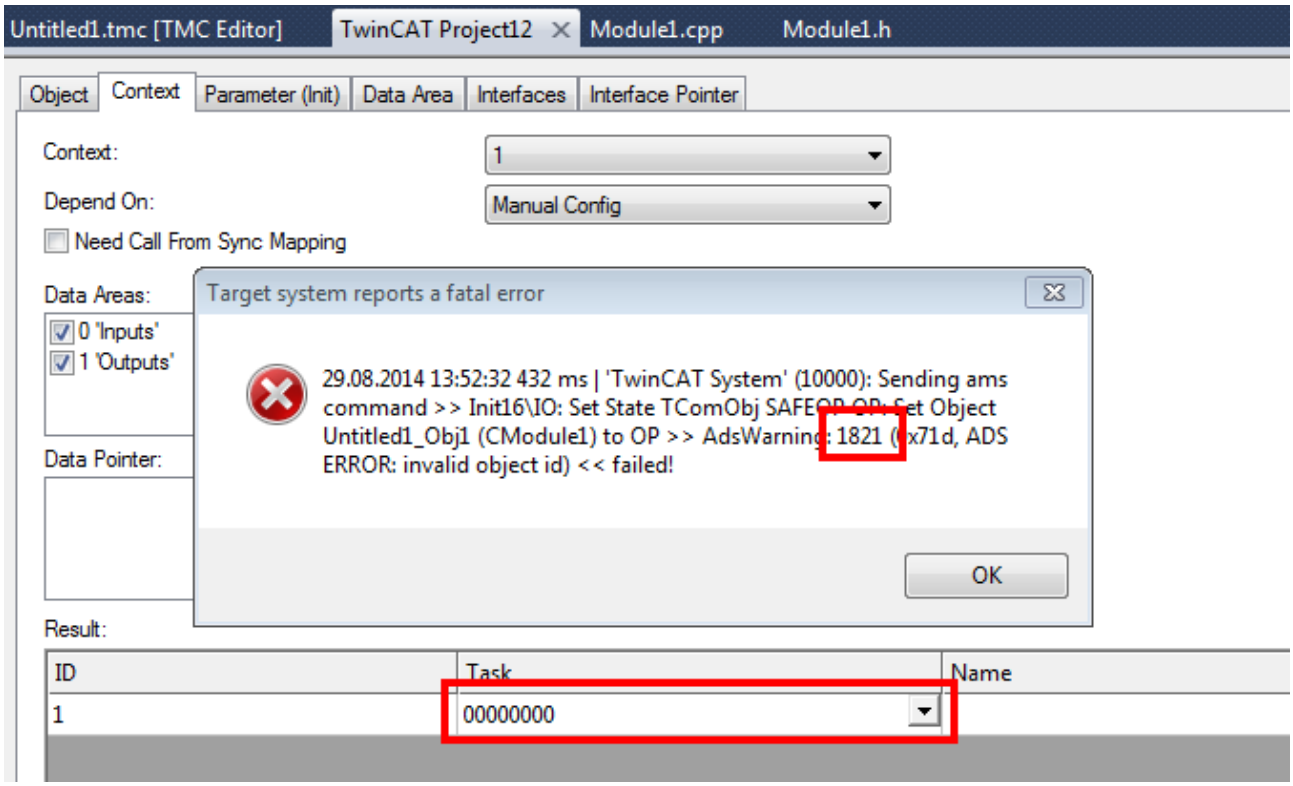


在这种情况下，请导航至**系统** -> **实时**，选择 **C++ 调试器**选项卡并激活**启用 C++ 调试器**选项。



14.3 激活 – “无效对象 ID” (1821/0x71d)

如果在启动过程中报告 ADS 返回代码 1821/0x71d，请按照快速入门 [▶ 70]中的说明检查模块实例的环境。



14.4 在 TwinCAT C++ 模块中使用 C++ 类

当使用 Visual Studio 环境菜单**添加** -> **类.....** 添加（非 TwinCAT）C++ 类时，编译器/链接器会报告：

```
Error 4 error C1010: unexpected end of file while looking for precompiled
header. Did you forget to add '#include ""' to your source?
```

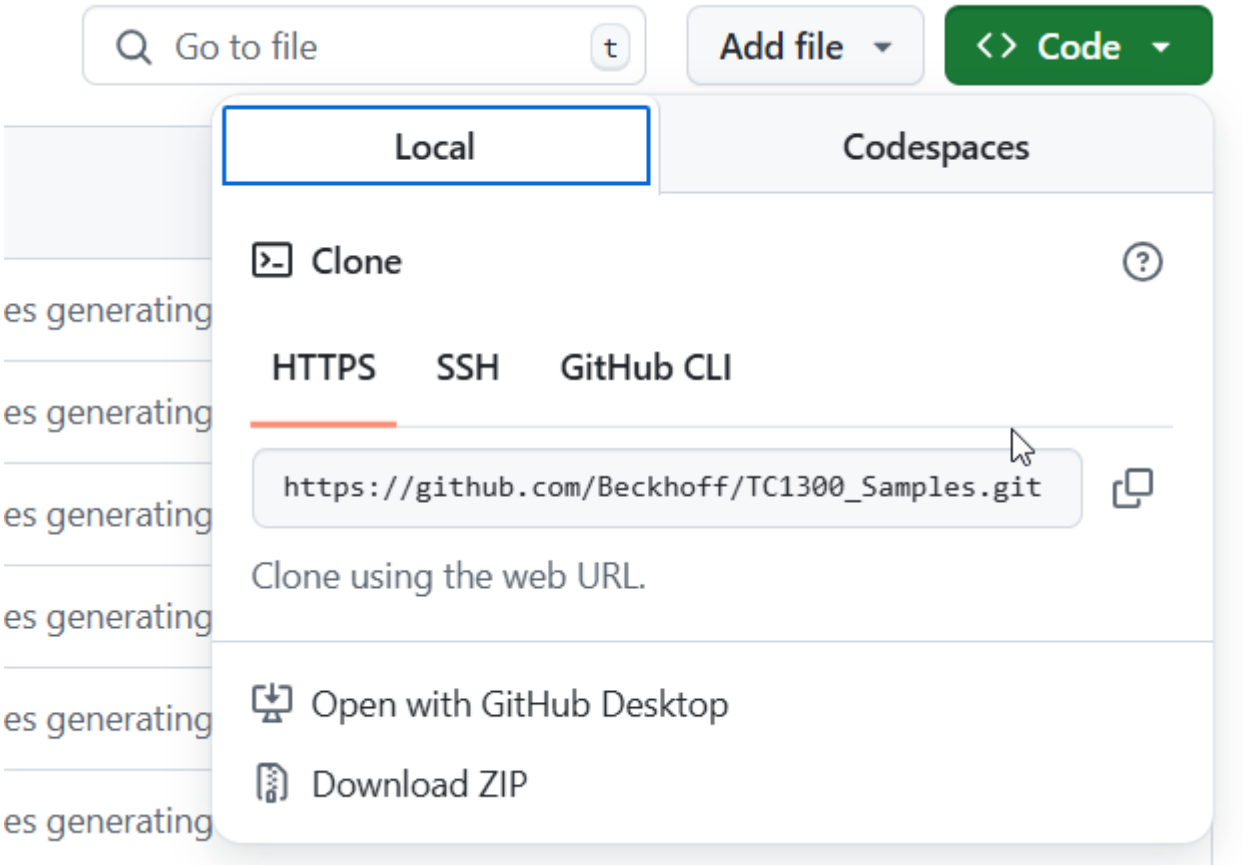
在生成的类文件开头插入以下行：

```
#include "TcPch.h"
#pragma hdrstop
```

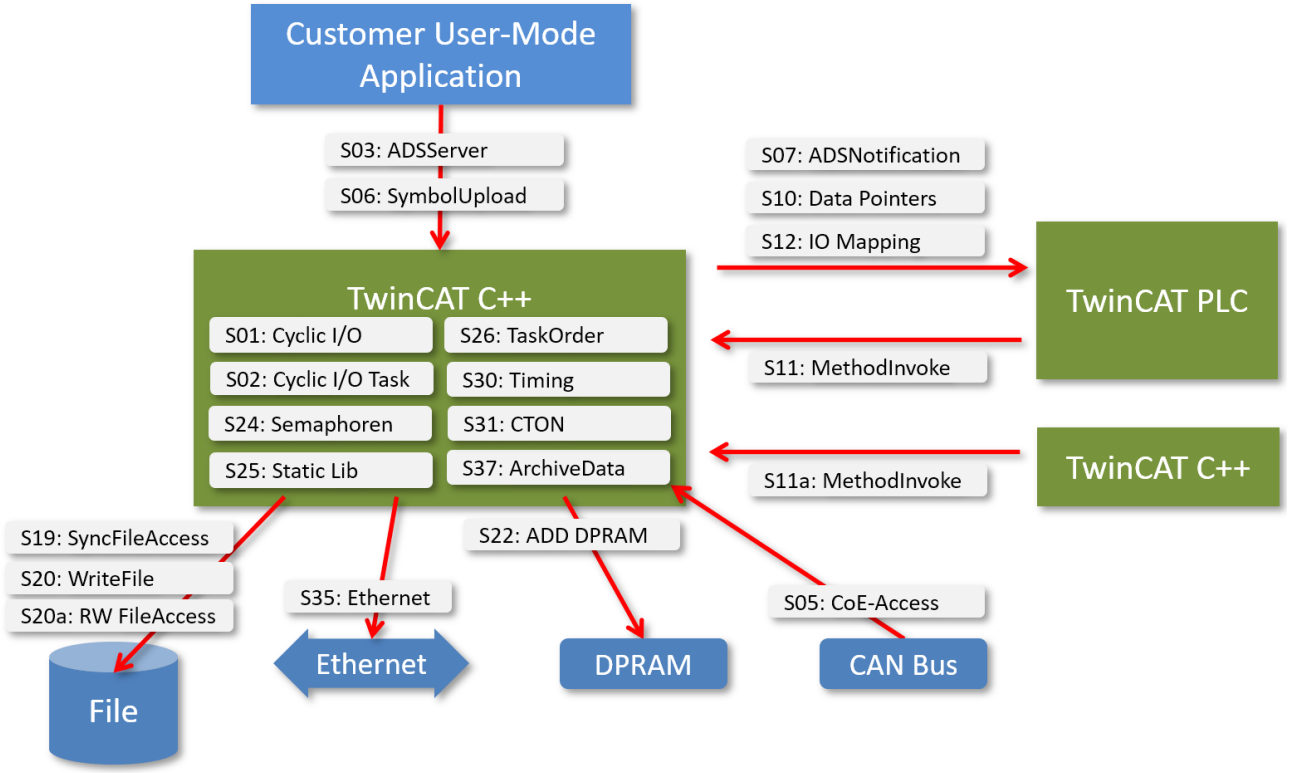
1 C++-示例

5

本产品的示例代码和配置可从 GitHub 上的相应存储库获取：https://github.com/Beckhoff/TC1300_Samples。您可以选择克隆存储库或下载包含示例的 ZIP 文件。



此图以图形的形式提供概览，并强调 C++ 模块的交互能力。



除此以外，此为包含各示例程序简要说明的对照表。

编号	标题	描述
01	Sample01：带 IO 的周期性模块 [▶ 237]	本文介绍使用与物理 IO 映射的 IO 模块的 TwinCAT 3 C++ 模块实现。本示例介绍的是快速入门，目的是创建一个 C++ 模块，在每个周期递增一个计数器，并将计数器分配给数据区的逻辑输出Value。 可以将数据区分配给物理 IO 或其他模块实例的其他逻辑输入。
02	Sample02：带 IO 任务的周期性任务 [▶ 238]	介绍 C++ 代码在处理任务中配置的 IO 时的灵活性。由于采用了这种方法，最终编译完成的 C++ 模块可以更灵活地影响与 IO 任务连接的各种 IO。其中一个应用是检查周期性模拟量输入通道，不同项目的输入通道数量可能不同。
03	Sample03：ADS 服务器客户端 [▶ 238]	介绍在 C++ 模块中自定义 ADS 接口的设计与实现。 示例包含两个部分： • 基于 TwinCAT 3 C++ 实现、搭载自定义 ADS 接口的 ADS 服务器。 ADS 客户端 UI 在 C# 中实现，可将用户特定 ADS 消息传送至 ADS 服务器。
05	Sample05：通过 ADS 访问 CoE [▶ 246]	展示如何通过 ADS 访问 EtherCAT 设备的 CoE 寄存器。
06	Sample06：ADS C# 客户端上传 ADS 符号 [▶ 248]	展示如何通过 ADS 接口访问 ADS 服务器中的符号。C# ADS 客户端与 PLC/C++/Matlab® 中实现的模块连接。上传可用符号信息，并读取/写入过程值的订阅。
07	Sample07：接收 ADS 通知 [▶ 252]	介绍 TwinCAT 3 C++ 模块的实现，该模块可接收有关其他模块数据变化的 ADS 通知。
08	Sample08：提供 ADS-RPC [▶ 253]	介绍 ADS 可通过任务调用的方法的实现。
10	Sample10：模块通信：数据指针的应用 [▶ 255]	介绍两个使用直接数据指针的 C++ 模块之间的交互。这两个模块必须在同一实时环境中的同一 CPU 内核上实现。
11	Sample11：模块通信：PLC 模块到 C++ 模块的方法调用 [▶ 256]	该示例包含两个部分： • 一个 C++ 模块，其功能是用作状态机，提供带有启动/停止方法，以及设置/维护状态机方法的接口。 第二个是 PLC 模块，通过调用 C++ 模块的方法与第一个模块交互。
11a	Sample11a：模块通信：C++ 模块调用 C++ 模块的方法 [▶ 283]	该示例包含一个驱动程序中的两个类（也可以在两个驱动程序之间执行）。 • 一个可提供计算方法的模块。通过 CriticalSection 保护的访问权限。 第二个模块作为调用者使用另一个模块中的方法。
12	Sample12：模块通信：IO 映射的应用 [▶ 284]	介绍两个模块如何通过不同模块数据区的符号映射进行交互。这两个模块可以在相同或不同的 CPU 内核上执行。
13	Sample13：模块通信：C++ 模块调用 PLC 模块的方法 [▶ 285]	介绍 TwinCAT 3 C++ 模块如何使用 TcCOM 接口方法调用 PLC 功能块。
19	Sample19：同步文件访问 [▶ 288]	介绍如何在 C++ 模块中以同步方式使用文件 IO 功能。 该示例将过程值写入文件。写入文件由确定性周期触发，文件 IO 的执行已解耦（异步），即确定性周期继续进行，不会受到向文件写入的阻碍。可以检查解耦写入文件例程的状态。
20	Sample20：FileIO-Write [▶ 289]	介绍如何将文件 IO 功能与 C++ 模块结合使用。 该示例将过程值写入文件。写入文件由确定性周期触发，文件 IO 的执行已解耦（异步），即确定性周期继续进行，不会受到向文件写入的阻碍。可以检查解耦写入文件例程的状态。
20a	Sample20a：FileIO-Cyclic 读取/写入 [▶ 289]	比 S20 和 S19 更广泛的示例。介绍从 C++ 模块对文件的循环读取和/或写入访问。

编号	标题	描述
22	Sample22：自动化设备驱动程序 (ADD)：访问 DPRAM [▶ 291]	介绍如何写入 TwinCAT Automation Device Driver (ADD)，以便访问 DPRAM。
23	Sample23：结构化异常处理 (SEH) [▶ 292]	介绍基于五种变体的结构化异常处理 (SEH) 的使用情况。
24	Sample24：信号量 [▶ 294]	介绍 Semaphore 的使用情况。
25	Sample25：静态库 [▶ 295]	介绍如何使用另一个 C++ 模块中包含的 C++ 静态库。
26	Sample26：任务的执行顺序 [▶ 296]	介绍在将一项任务分配给多个模块的情况下，如何确定任务执行顺序。
30	Sample30：时序测量 [▶ 298]	介绍对 C++ 周期或执行时间的测量。
31	Sample31：TwinCAT3 C++ 中的功能块 TON [▶ 299]	介绍在 C++ 中实现与 PLC/61131 的 TON 功能块类似的行为。
37	Sample37：归档数据 [▶ 301]	介绍在初始化和反初始化过程中加载和保存对象状态的过程。
TcCOM	TcCOM 示例 [▶ 302]	有若干示例用以说明 PLC 和 C++ 之间的模块通信。

15.1 Sample01：带 IO 的周期性模块

本文介绍如何实现使用模块 IO 映射到物理 IO 的 TwinCAT 3 C++ 模块。

下载

您可在此处获取该示例的源代码：
https://github.com/Beckhoff/TC1300_Samples/tree/main/S01-CyclicIO

1. 利用已安装 TwinCAT 的 Visual Studio，通过**打开项目.....**来打开项目
2. 通过右键单击**项目 -> 属性 -> Tc 签名**打开 TwinCAT 签名，为该项目配置签名，必要时配置证书和密码。有关签署 C++ 项目的更多信息，请参见 TwinCAT [▶ 22] 章节。
3. 选择目标系统。
4. 构建示例（例如**构建 -> 构建解决方案**）。
5. 点击  激活配置。

⇨ 示例已做好操作准备。

功能说明

本示例介绍的是快速入门，目的是创建一个 C++ 模块，在每个周期递增一个计数器，并将计数器分配给数据区中的逻辑输出Value。
可将数据区分配给物理 IO 或其他逻辑输入或其他模块实例。

简短说明 [▶ 57]中对示例进行了分步说明。

15.2 Sample02：循环 C++ 逻辑，使用来自 IO 任务的 IO

本文介绍了使用 IO 任务镜像的 TwinCAT 3 C++ 模块的实现。

下载

您可以在此访问该示例的源代码：

https://github.com/Beckhoff/TC1300_Samples/tree/main/S02-CyclicIOTask

1. 利用已安装 TwinCAT 的 Visual Studio，通过**打开项目……**来打开项目
2. 通过右键单击**项目 -> 属性 -> Tc 签名**打开 TwinCAT 签名，为该项目配置签名，必要时配置证书和密码。有关签署 C++ 项目的更多信息，请参见 TwinCAT [▶ 22] 章节。
3. 选择目标系统。
4. 构建示例（例如**构建 -> 构建解决方案**）。
5. 点击  激活配置。

⇨ 示例已做好操作准备。

并非由向导自动生成的源代码会标记为起始标志“//示例代码”和结束标志“//示例代码结束”。
这样便可在文件中搜索这些字符串，以了解详情。

功能说明

该示例介绍了 C++ 代码在处理任务中配置的 IO 时的灵活性。如果 IO 任务关联不同数量的 IO，那么这种方法可让编译完成的 C++ 模块更灵活地做出响应。一种应用方案是根据项目的需要，用不同数量的通道对模拟输入通道进行循环测试。

示例包含

- 带有模块实例 TcIoTaskIMageAccessDrv_Obj1 的 C++ 模块 TcIoTaskIMageAccessDrv
- 一个带有镜像、10 个输入变量 (Var1..Var10) 和 10 个输出变量 (Var11..Var20) 的“Task1”。
- 彼此相互关联：上述模块实例由 Task1 调用，并使用该任务的数据镜像。

C++ 代码通过数据镜像访问这些值，数据镜像在从 SAFEOP 转换到 OP (SO) 的过程中进行初始化。

在周期性执行的 CycleUpdate 方法中，每个输入变量的值都会通过调用辅助方法 CheckValue 进行检查。如果小于 0，则相应的输出变量设置为 1；如果大于 0，则设置为 2；如果为 0，则输出设置为 3。

激活配置后，您可以通过解决方案资源管理器访问变量并对其进行设置。
双击系统的 Task1 镜像以查看概览。
输入变量可以打开，然后通过**在线**选项卡进行设置。

15.3 Sample03：C++ 用作 ADS 服务器

本文介绍：

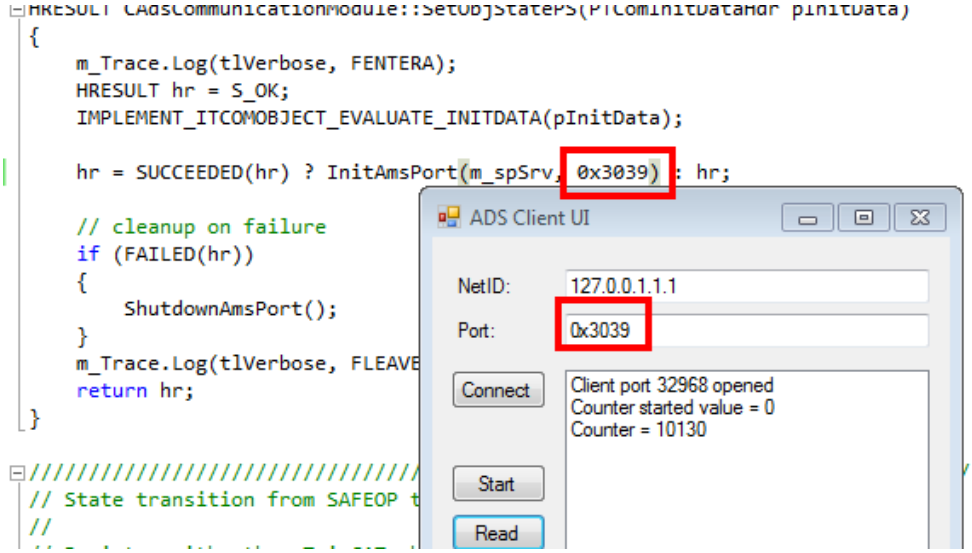
- 创建一个作为 ADS 服务器的 TwinCAT 3 C++ 模块。
  服务器提供一个 ADS 接口，用于启动/停止/重置 C++ 模块中的一个计数器变量。计数器可作为模块输出，并可分配给输出端子模块（支持模拟量或若干数字IO通道）。
  如何实现用 C++ 写入的 TC3 ADS 服务器功能。 [▶ 239]

- 创建一个 C# ADS 客户端，与 C++ ADS 服务器进行交互。
  客户端提供一个 UI，用于在本地或通过网络与 ADS 服务器进行连接，并提供 ADS 接口进行计数。UI 可启动/停止/读取/覆盖和重置计数器。
  示例代码：用 C# 写入的 ADS 客户端 UI [▶ 243]。

示例说明

示例中使用的是自动识别 ADS 端口的实现方案。这样做的缺点是，必须在每次启动时配置客户端，才能访问正确的 ADS 端口。

或者，也可以在模块中对 ADS 端口进行硬编码，如下所示。
缺点是：由于无法共享 ADS 端口，C++ 模块仅能实例化一次（无法多次实例化）。



15.3.1 Sample03：基于 C++ 编写的 TC3 ADS 服务器

本文介绍如何创建用作 ADS 服务器的 TC3-C++ 模块。
该服务器将提供一个 ADS 接口，用于启动/停止/重置洞察 C++ 模块的计数器变量。

下载

您可以在此访问该示例的源代码：

https://github.com/Beckhoff/TC1300_Samples/tree/main/S03-ADSServer

.

1. 利用已安装 TwinCAT 的 Visual Studio，通过**打开项目……**来打开项目
2. 通过右键单击**项目 -> 属性 -> Tc 签名**打开 TwinCAT 签名，为该项目配置签名，必要时配置证书和密码。有关签署 C++ 项目的更多信息，请参见 TwinCAT [▶ 22] 章节。
3. 选择目标系统。
4. 构建示例（例如**构建 -> 构建解决方案**）。
5. 点击 激活配置。

⇨ 示例已做好操作准备。

功能说明

该示例包含一个用作 ADS 服务器的 C++ 模块。服务器允许访问可以启动、停止和读取的计数器。

模块的报头文件定义了计数器变量 m_bCount，相应的 .cpp 文件则在构造函数中对该值进行初始化，并在 CycleUpdate 方法中实现逻辑。

.cpp 文件中的 AdsReadWriteInd 方法会分析传入的消息并返回返回值。在报头文件中增加一个宏定义，用于进一步添加消息类型。

有关 ADS 消息类型的定义等详细信息，请参阅下方操作指南，您可以在其中手动编译示例。

操作指南

下面将分步介绍如何创建 C++ 模块。

1. 创建新的 TwinCAT 3 项目解决方案

按照创建新的 TwinCAT 3 项目 [▶ 57]的相关步骤操作。

2. 创建带有 ADS 端口的 C++ 项目

按照创建新的 TwinCAT 3 C++ 项目 [▶ 58]的相关步骤操作。

在**类模板**对话框中选择**带 ADS 端口的 TwinCAT 模块类**。

3. 将示例逻辑添加到项目中

1. 打开报头文件 <MyClass>.h（本例中为 Module1.h），将计数器 m_bCount 作为新的成员变量添加到保护区：

```
class CModule1
    : public ITComObject
    , public ITcCyclic
    ,...
{
public:
    DECLARE_IUNKNOWN()
    ....
protected:
    DECLARE_ITCOMOBJECT_SETSTATE();
///<AutoGeneratedContent id="Members">
    ITcCyclicCallerInfoPtr m_spCyclicCaller;
    .....
///</AutoGeneratedContent>
    ULONG m_ReadByOidAndPid;
    BOOL m_bCount;
};
```

2. 打开类文件 <MyClass>.cpp（本例中为 Module1.cpp），在构造函数中初始化新值：

```
CModule1::CModule1()
    .....
{
    memset(&m_Counter, 0, sizeof(m_Counter));
    memset(&m_Inputs, 0, sizeof(m_Inputs));
    memset(&m_Outputs, 0, sizeof(m_Outputs));
    m_bCount = FALSE; // by default the counter should not increment
    m_Counter = 0;    // we also initialize this existing counter
}
```

⇨ 已添加示例代码。

3.a. 将示例逻辑添加至 ADS 服务器接口。

通常，ADS 服务器会收到一条 ADS 消息，其中包含两个参数（indexGroup 和 indexOffset），或许还有 pData。

设计 ADS 接口
需要对我们的计数器执行启动、停止、重置、改写值或按要求向 ADS 客户端发送值：

indexGroup	indexOffset	描述
0x01	0x01	m_bCount = TRUE，计数器递增。
0x01	0x02	计数器值传输到 ADS 客户端。
0x02	0x01	m_bCount = FALSE，计数器不再递增。
0x02	0x02	重置计数器。
0x03	0x01	用 ADS 客户端传输的值覆盖计数器。

这些参数会在 modules1Ads.h 中定义，更改源代码以添加 IG_RESET 新命令。

```
#include "TcDef.h"
enum Module1IndexGroups : ULONG
{
Module1IndexGroup1 = 0x00000001,
Module1IndexGroup2 = 0x00000002, // add command
IG_OVERWRITE = 0x00000003 // and new command
};

enum Module1IndexOffsets : ULONG
{
Module1IndexOffset1 = 0x00000001,
Module1IndexOffset2 = 0x00000002
};
```

更改 <MyClass>::AdsReadWriteInd() 方法中的源代码（本例中为 Module1.cpp）。

```
switch(indexGroup)
{
case Module1IndexGroup1:
switch(indexOffset)
{
case Module1IndexOffset1:
...
// TODO: add custom code here
m_bCount = TRUE; // receivedIG=1 IO=1, start counter
AdsReadWriteRes(rAddr, invokeId, ADSERR_NOERR, 0,NULL);
break;
case Module1IndexOffset2:
...
// TODO: add custom code here
// map counter to data pointer
pData = &m_Counter; // received IG=1 IO=2, provide counter value via ADS
AdsReadWriteRes(rAddr, invokeId, ADSERR_NOERR, 4 ,pData);
//comment this: AdsReadWriteRes(rAddr, invokeId,ADSERR_NOERR, 0, NULL);
break;
}
break;
case Module1IndexGroup2:
switch(indexOffset)
{
case Module1IndexOffset1:
...
// TODO: add custom code here
// Stop incrementing counter
m_bCount = FALSE;
// map counter to data pointer
pData = &m_Counter;
AdsReadWriteRes(rAddr, invokeId, ADSERR_NOERR, 4,pData);
break;
case Module1IndexOffset2:
...
// TODO: add custom code here
// Reset counter
m_Counter = 0;
// map counter to data pointer
pData = &m_Counter;
AdsReadWriteRes(rAddr, invokeId, ADSERR_NOERR, 4, pData);
break;
}
break;
case IG_OVERWRITE:
switch(indexOffset)
{
case Module1IndexOffset1:
...
// TODO: add custom code here // override counter with value provided by ADS-client
unsigned long *pCounter = (unsigned long*) pData;
m_Counter = *pCounter;
AdsReadWriteRes(rAddr, invokeId, ADSERR_NOERR, 4, pData);
break;
}
break;
}
break;
```

```
default:
__super::AdsReadWriteInd(rAddr, invokeId, indexGroup,indexOffset, cbReadLength, cbWriteLength,
pData;
break;
}
```

3.b. 将示例逻辑嵌入至周期性执行代码段中

<MyClass>::CycleUpdate() 方法会被循环调用，这是改变逻辑的要点。

```
// TODO: Replace the sample with your cyclic code
m_Counter+=m_Inputs.Value; // replace this line
m_Outputs.Value=m_Counter;
```

在这种情况下，如果布尔变量 **m_bCount**为 TRUE，计数器 **mCounter** 便会递增。

将该条件分支语句嵌入至周期性执行方法中。

```
HRESULT CModule1::CycleUpdate(ITcTask* ipTask,
ITcUnknown* ipCaller, ULONG context)
{
HRESULT hr = S_OK;
// handle pending ADS indications and confirmations
CheckOrders();
....
// TODO: Replace the sample with your cyclic code
if (m_bCount) // new part
{
m_Counter++;
}
m_Outputs.Value=m_Counter;
}
```

4. 运行服务器示例

1. 运行 TwinCAT TMC Code Generator，为该模块配置输入 / 输出接口。
2. 保存项目。
3. 编译项目。
4. 创建模块实例。
5. 创建循环任务并配置 C++ 模块，以便在此环境中执行。
6. 扫描硬件 IO 并将输出的符号值分配给指定的输出端子模块（此为可选项）。
7. 激活 [▶ 71] TwinCAT 项目。

⇨ 示例已做好操作准备。

5. 确定模块实例的 ADS 端口

一般来说，ADS 端口可以

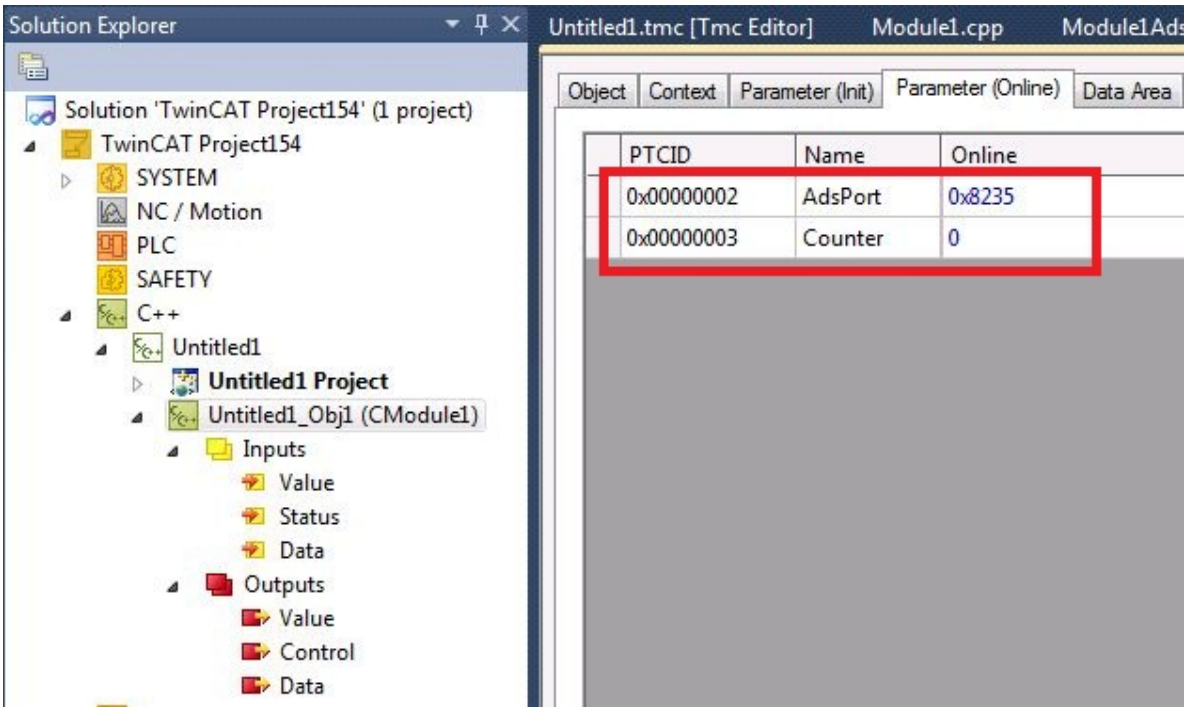
- 预先编号，以便该模块实例始终使用相同的端口。
- 保持可定制，以便在启动 TwinCAT 系统时为多个模块实例分配独立的 ADS 端口。

在本示例中，已选择默认设置（保持灵活模式）。首先，您必须确定分配给刚刚激活的模块的 ADS 端口。

1. 导航至模块实例。
2. 选择**参数在线**选项卡。

   ⇨ ADS 端口分配的编号为 0x8235（十进制表示为 33333），该数值在你的示例程序中可能会有所不同。如果创建的实例越来越多，每个实例都会分配到各自唯一的 AdsPort。

⇨ 计数器仍为“0”，因为尚未发送开始递增的 ADS 消息。



⇨ 服务器部分已完成，继续执行 ADS 客户端发送 ADS 消息 [▶ 243]。

15.3.2 Sample03：C# 中的 ADS 客户端 UI

本文介绍的是 ADS 客户端，它会将 ADS 消息发送到前面介绍的 ADS 服务器。

ADS 服务器的实现既不取决于语言 (C++/C#/PLC/.....)，也不取决于 TwinCAT 版本（TwinCAT 2 或 TwinCAT 3）。

下载

您可以在此访问该示例的源代码：

https://github.com/Beckhoff/TC1300_Samples/tree/main/S03-ADSClient

✓ 该代码需要使用 .NET Framework

1. 用 Visual Studio 打开其中包含的 sln 文件。
2. 在本地机器上创建示例（右键单击项目，然后点击**构建**）。
3. 右键单击**项目**、**调试** -> **启动新实例**，启动程序。

功能说明

该客户端执行两项任务：

- 测试之前介绍过的 ADS 服务器。
- 提供用于实现 ADS 客户端的示例代码

使用客户端

选择通信伙伴

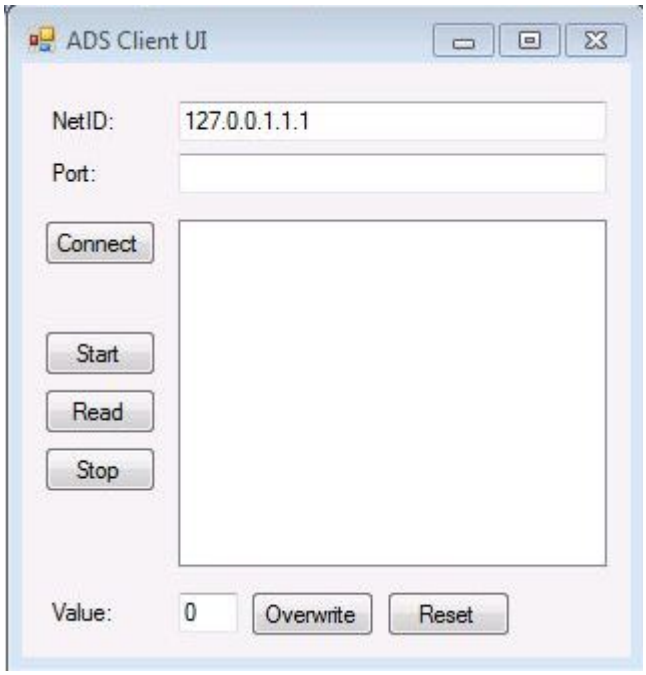
输入两个 ADS 参数，以指定 ADS 通信伙伴：

- NetID：
  127.0.0.1.1.1（适用于同样连接到本地 ADS 消息路由器的 ADS 合作伙伴）
  如果要通过网络与连接到另一个 ADS 路由器的 ADS 合作伙伴进行通信，请输入不同的 NetID。
  您必须先在您的设备和远程设备之间建立 ADS 路由。

- AdsPort
  输入通信伙伴的 AdsServerPort。
  不要将 ADS 服务器端口（已明确实现您自己的消息处理程序）与用于访问符号的常规 ADS 端口混淆（该端口由系统自动分配，无需用户手动干预）。
  查找分配的 AdsPort [▶ 239]，在本示例中，AdsPort 为 0x8235 (dec 33333)。

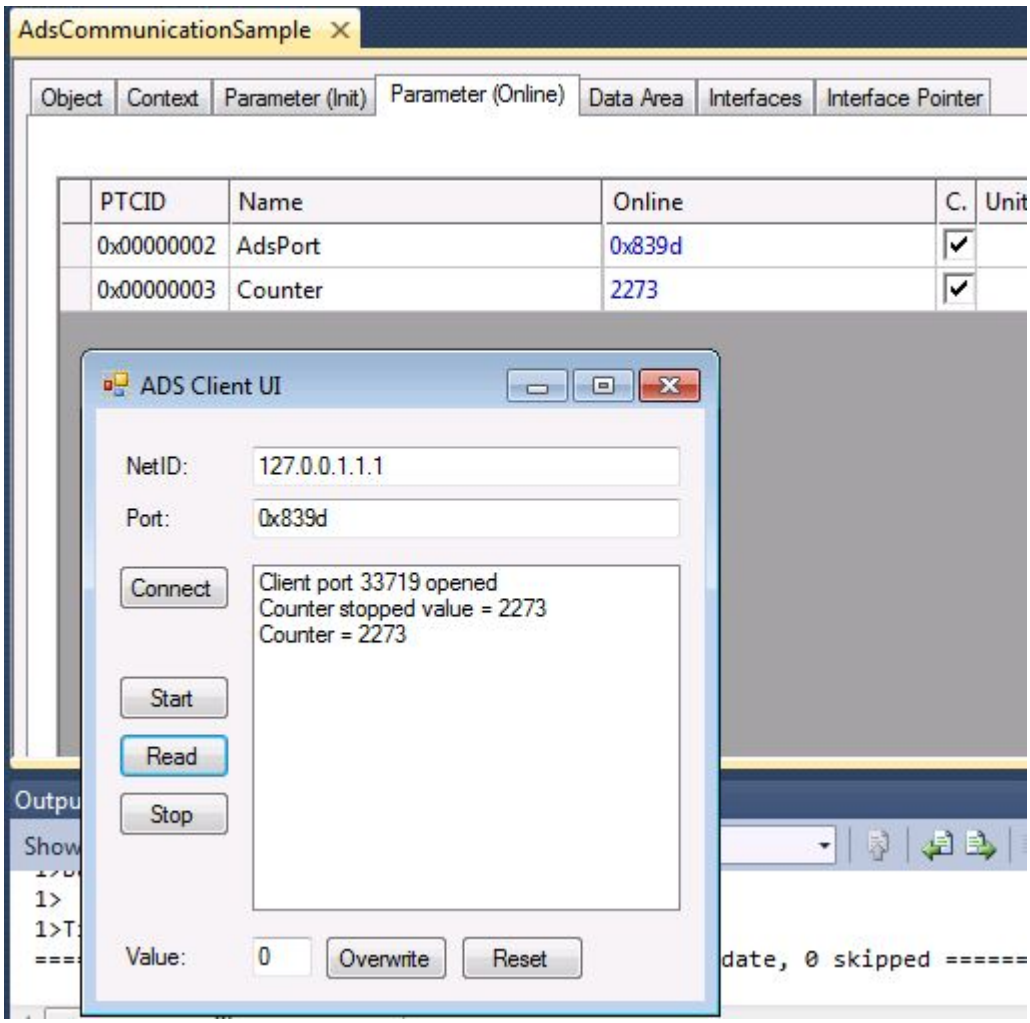
与通信伙伴建立联系

点击**连接**调用 TcAdsClient.Connect 方法，以创建与配置端口的链接。



启动/**读取**/**停止**/**覆盖**/**复位**按钮用于向 ADS 服务器发送 ADS 消息。
ADS 服务器的 ADS 接口中已设计 [▶ 239]特定的 indexGroup/indexOffset 指令。

点击命令按钮的结果也可以在模块实例的**参数（在线）**选项卡中看到。



C# 程序

以下为 ADS 客户端的“核心”代码，通过上方 GUI 或 ZIP 文件下载。

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using TwinCAT.Ads;

namespace adsClientVisu
{
public partial class form : Form
{
public form()
{
InitializeComponent();
}

private void Form1_Load(object sender, EventArgs e)
{
// create a new TcClient instance
_tcClient = new TcAdsClient();
adsReadStream = new AdsStream(4);
adsWriteStream = new AdsStream(4);
}

/*
* Connect the client to the local AMS router
*/

private void btConnect_Click(object sender, EventArgs e)
{
AmsAddress serverAddress = null;
try
{
serverAddress = new AmsAddress(tbNetId.Text,
Int32.Parse(tbPort.Text));
```

```
 }
catch
{
MessageBox.Show("Invalid AMS NetId or Ams port");
return;
}

try
{
_tcClient.Connect(serverAddress.NetId, serverAddress.Port);
lbOutput.Items.Add("Client port " + _tcClient.ClientPort + " opened");

}
catch
{
MessageBox.Show("Could not connect client");
}
}

private void btStart_Click(object sender, EventArgs e)
{
try
{
_tcClient.ReadWrite(0x1, 0x1, adsReadStream, adsWriteStream);
byte[] dataBuffer = adsReadStream.ToArray();
lbOutput.Items.Add("Counter started value = " + BitConverter.ToInt32(dataBuffer, 0));
}

catch (Exception err)
{
MessageBox.Show(err.Message);
}

}

private void btRead_Click(object sender, EventArgs e)
{
try
{
_tcClient.ReadWrite(0x1, 0x2, adsReadStream, adsWriteStream);
byte[] dataBuffer = adsReadStream.ToArray();
lbOutput.Items.Add("Counter = " + BitConverter.ToInt32(dataBuffer, 0));
}

catch (Exception err)
{
MessageBox.Show(err.Message);
}
}

private void btStop_Click(object sender, EventArgs e)
{
try
{
_tcClient.ReadWrite(0x2, 0x1, adsReadStream, adsWriteStream);
byte[] dataBuffer = adsReadStream.ToArray();
lbOutput.Items.Add("Counter stopped value = " + BitConverter.ToInt32(dataBuffer, 0));
}

catch (Exception err)
{
MessageBox.Show(err.Message);
}
}

private void btReset_Click(object sender, EventArgs e)
{
try
{
_tcClient.ReadWrite(0x2, 0x2, adsReadStream, adsWriteStream);
byte[] dataBuffer = adsReadStream.ToArray();
lbOutput.Items.Add("Counter reset Value = " + BitConverter.ToInt32(dataBuffer, 0));
}

catch (Exception err)
{
MessageBox.Show(err.Message);
}
}
}
}
```

15.4 Sample05：通过 ADS 访问 C++ CoE

本文介绍如何实现 TwinCAT 3 C++ 模块，该模块可以访问 EtherCAT 端子模块的 CoE（CANopen over EtherCAT）寄存器。

下载

您可以在此访问该示例的源代码：

https://github.com/Beckhoff/TC1300_Samples/tree/main/S05-CoEAccess

1. 点击**打开项目……**，打开 TwinCAT 3 中包含的该项目的 zip 文件。
2. 选择目标系统。
3. 在本地机器上构建示例（例如，**构建** -> **构建解决方案**）。
4. 请注意本页**配置**下所列操作。
5. 点击  激活配置。

⇨ 示例已做好操作准备。

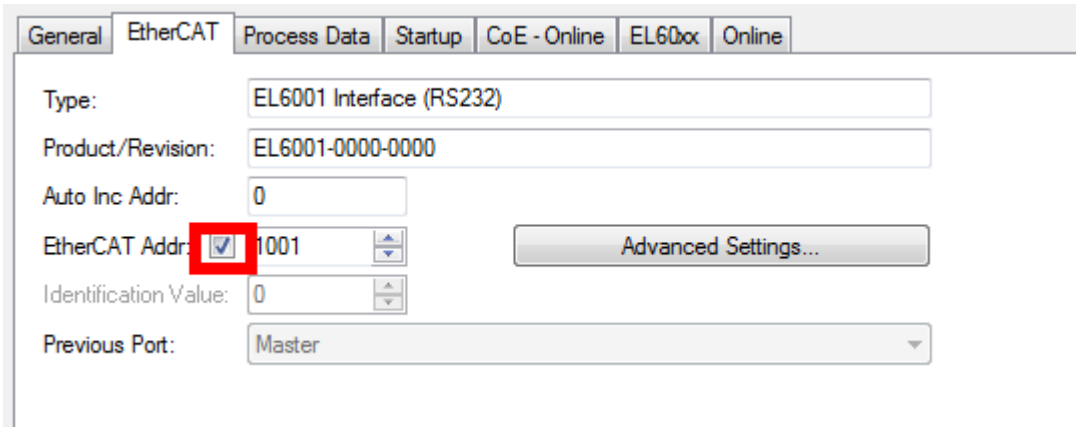
功能说明

本示例介绍如何访问 EtherCAT 端子模块，读取制造商 ID 并设置串行通信的波特率。

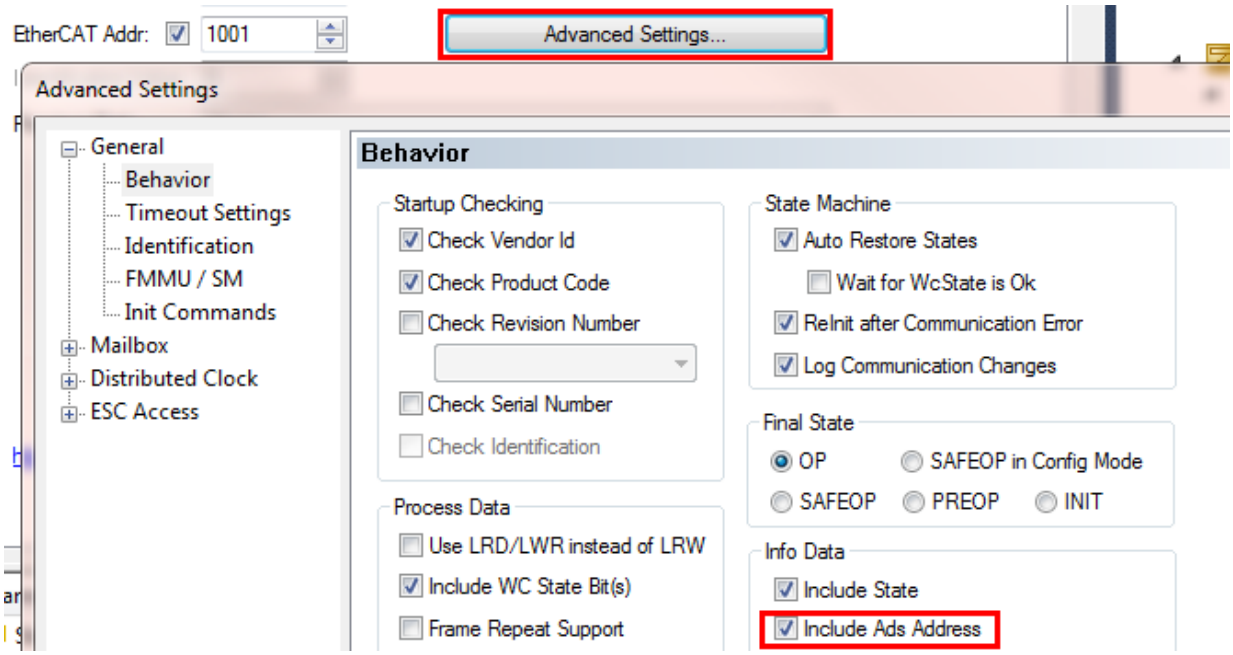
本示例介绍的是快速入门，目的是创建一个 C++ 模块，在每个周期递增一个计数器，并将计数器分配给数据区中的逻辑输出Value。

配置

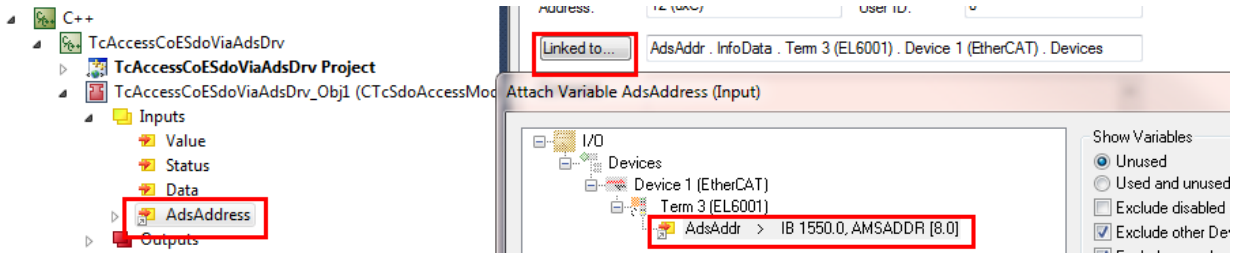
1. 激活相关端子模块的 EtherCAT 地址并进行分配。



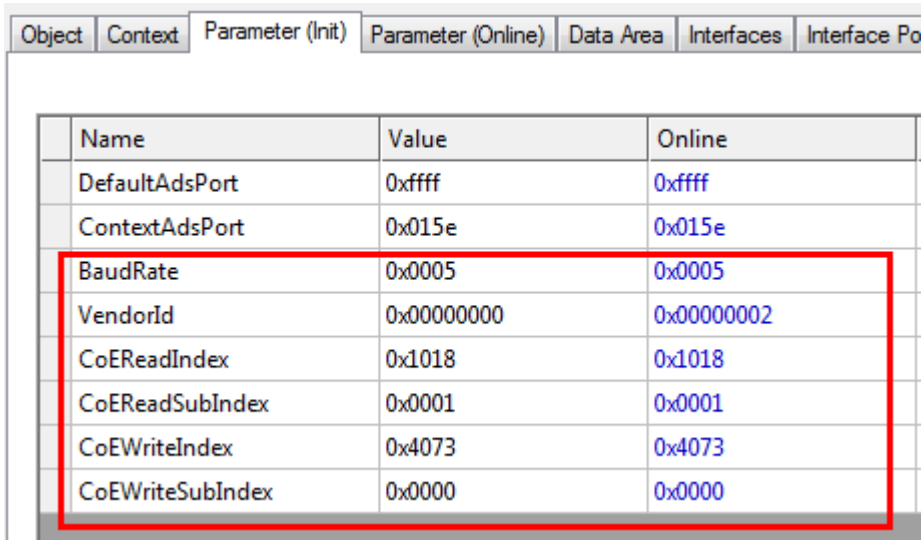
2. 激活 EtherCAT 端子模块高级设置中的 ADS 地址：



3. 将 ADS 地址（包括 netId 和端口）分配至模块输入 AdsAdress：



4. 在初始化过程中，示例代码会读出并显示模块参数：



Name	Value	Online
DefaultAdsPort	0xffff	0xffff
ContextAdsPort	0x015e	0x015e
BaudRate	0x0005	0x0005
VendorId	0x00000000	0x00000002
CoEReadIndex	0x1018	0x1018
CoEReadSubIndex	0x0001	0x0001
CoEWriteIndex	0x4073	0x4073
CoEWriteSubIndex	0x0000	0x0000

15.5 Sample06：UI-C#-ADS 客户端从模块上传符号

本文介绍了 ADS 客户端的实现，以

- 连接到提供过程映像（数据区）的 ADS 服务器；可以在本地建立连接，也可以通过网络远程建立连接，
- 上传符号信息，
- 同步读取/写入数据，

- 订阅符号，以便在变化时作为回调获取值。

下载

您可以在此访问该示例的源代码：

https://github.com/Beckhoff/TC1300_Samples/tree/main/S06-SymbolUploadClient

✓ 该代码需要使用 .NET Framework 3.5 或更高版本！

1. 用 Visual Studio 打开其中包含的 sln 文件。
2. 在本地机器上创建示例（右键单击项目，然后点击“构建”）。
3. 右键单击**项目**、**调试** -> **启动新实例**，启动程序。

客户端示例应与示例 03“C++ 作为 ADS 服务器”配合使用。

在开始使用此客户端示例之前，请先打开示例 03 [▶ 239]！

功能说明

根据本示例功能说明 ADS 的可能性。

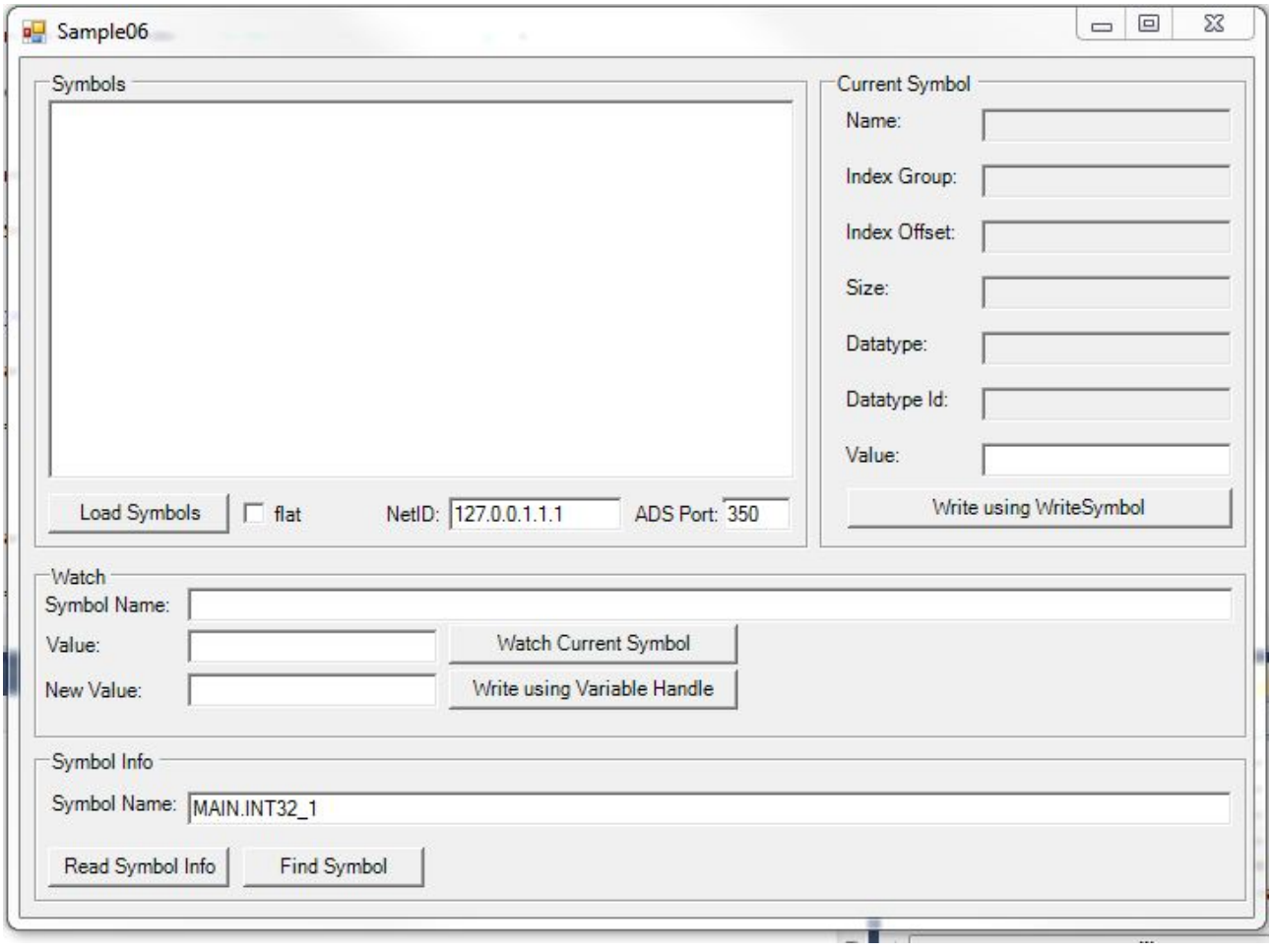
实现的细节在 Form1.cs 中有所说明，该文件包含在下载内容中。通过 ADS 与目标系统的连接是在 btnLoad_Click 方法中建立的，点击**加载符号**按钮时会调用该方法。在此，您可以探索不同的 GUI 功能。

背景信息：

对于该 ADS 客户端来说，ADS 服务器是基于 TwinCAT 2 还是 TwinCAT 3 并不重要，服务器是 C++ 模块、PLC 模块还是没有任何逻辑的 IO 任务也不重要。

ADS 客户端 UI

启动示例后，会显示用户界面 (UI)。



选择通信伙伴

启动客户端后，输入两组ADS 参数以确定 ADS 通信伙伴。

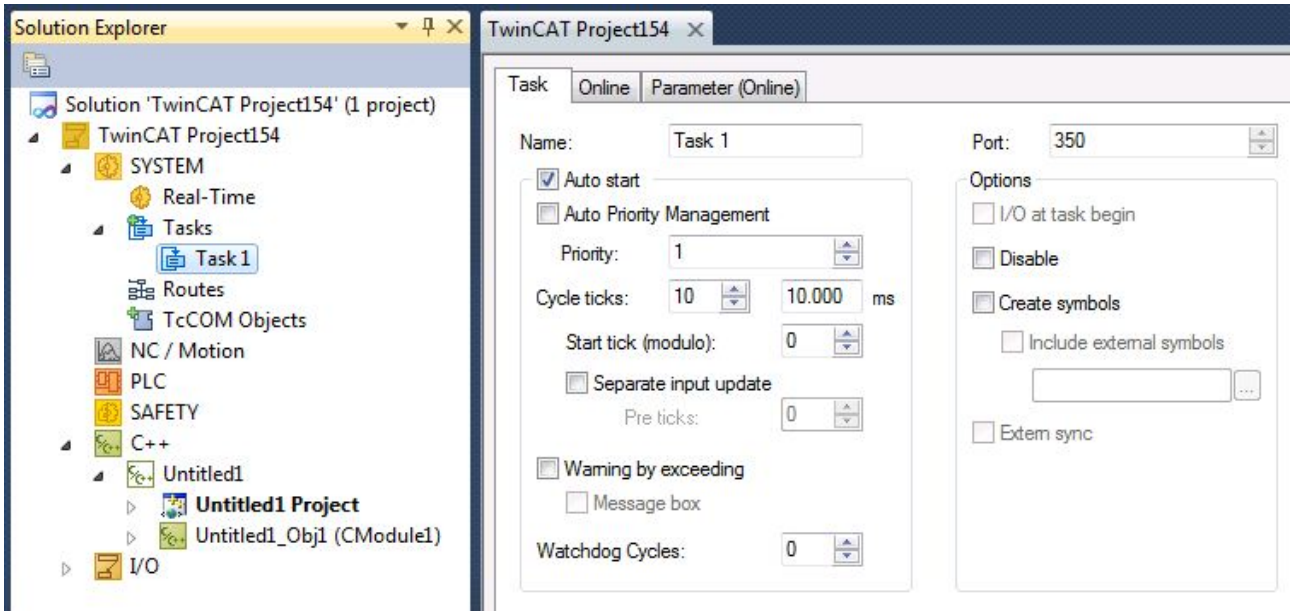
- NetID：
  127.0.0.1.1.1（ADS 合作伙伴同时与本地 ADS 消息路由器相连）。
  如果要通过网络与连接到另一个 ADS 路由器的 ADS 合作伙伴通信，请输入另一个 NetID。
  首先要在您的设备与远程设备之间创建 ADS 路由。
- AdsPort
  输入通信伙伴的 AdsPort：350（在本示例中）。

不要将 ADS 服务器端口与常规 ADS 端口混淆。

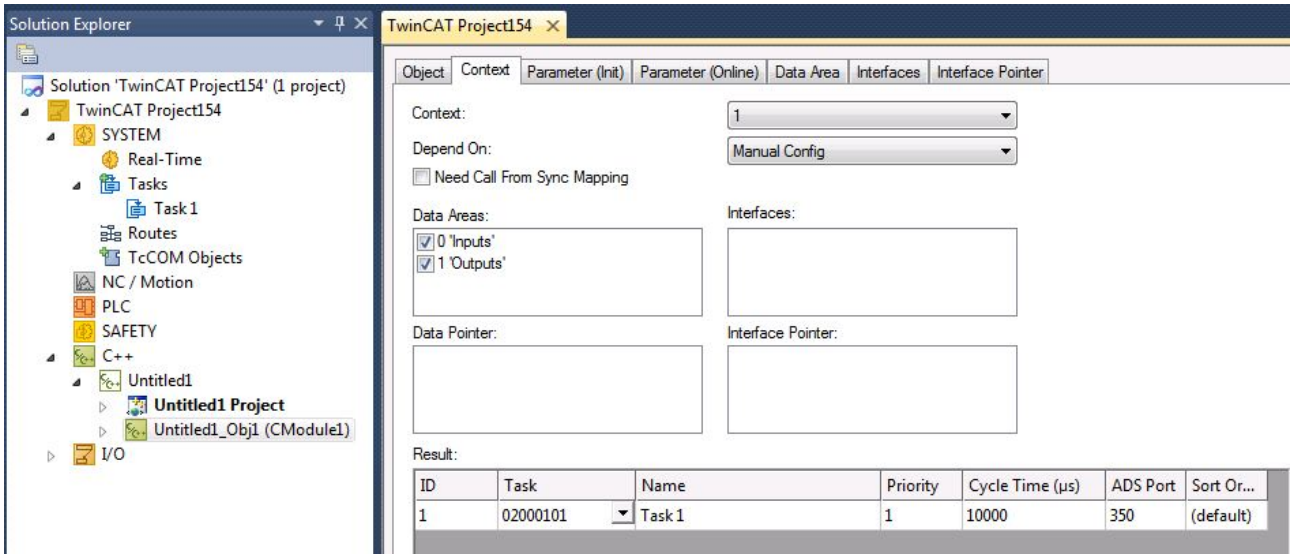
不要将 ADS 服务器端口（在示例 03 中已明确实现，用于提供您自身的消息处理程序）与用于访问符号的常规 ADS 端口（自动提供，无需用户干预）相混淆：

访问符号需要使用常规 ADS 端口。您可以为您的实例或模块实例的 IO 任务查找 AdsPort（因为模块是在 IO 任务的环境中执行的）。

导航至 IO 任务 Task1，并记下端口值：350。



由于 C++ 模块实例是在 Task1 环境中执行的，因此 ADS 端口也是 350。

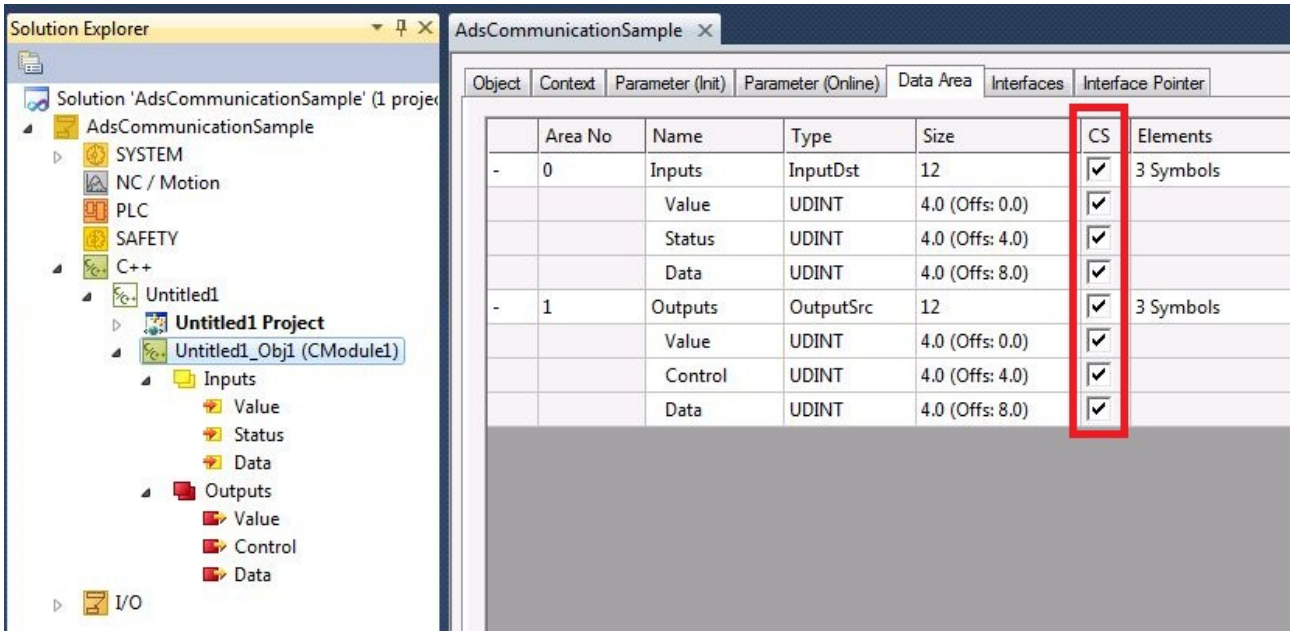


启用可通过 ADS 访问的符号信息

可以通过 ADS 提供或不提供单个符号（或者甚至完整的数据区）进行访问。
导航至实例的**数据区**选项卡，对 **C/S** 栏执行启用或禁用操作。

在本示例中，所有符号均已做标记，因此可供 ADS 访问。

更改后，点击**激活配置**。



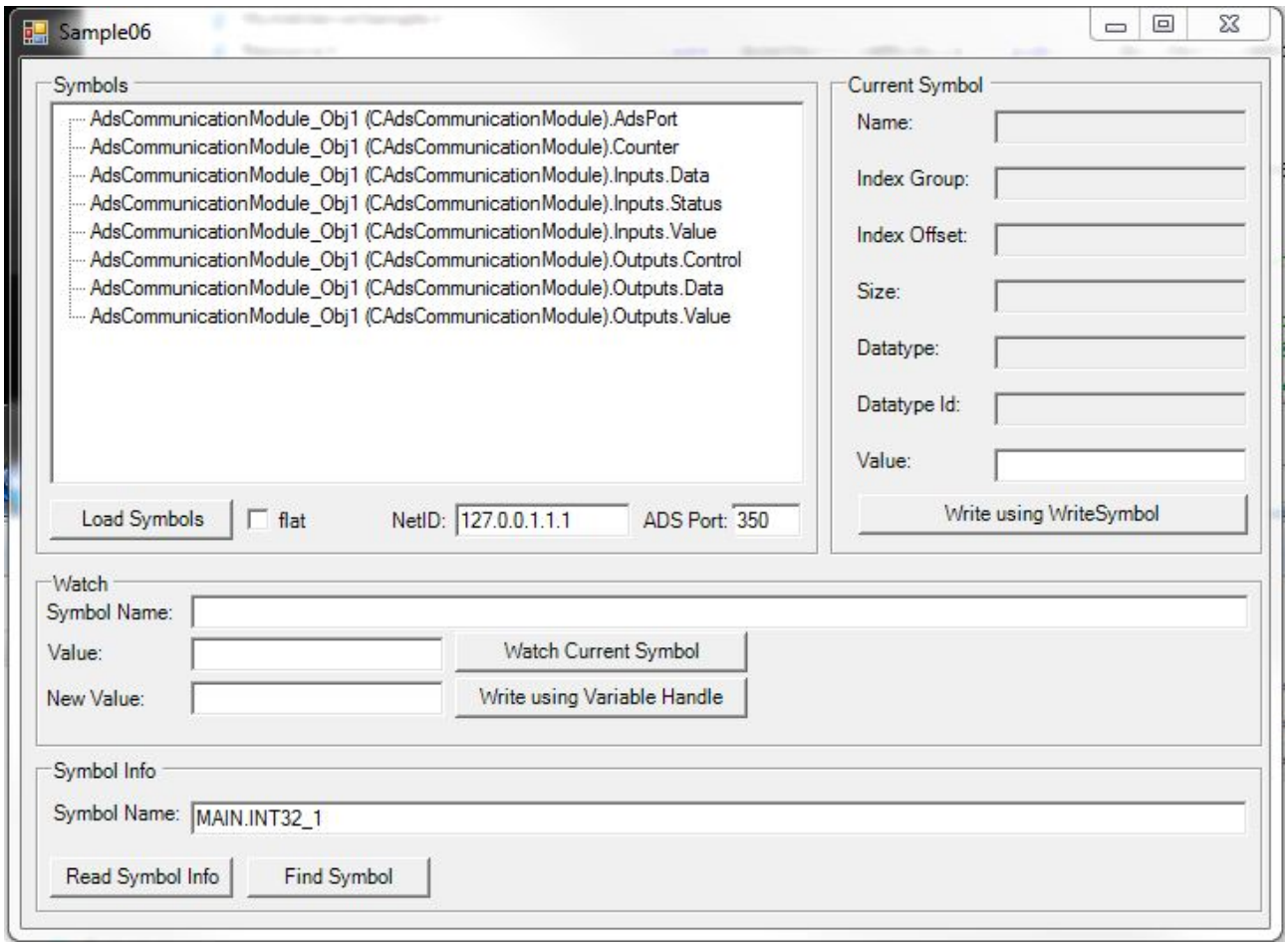
加载符号

NetID 和 ADS 端口设置完成后，点击**加载符号**按钮，与目标系统建立连接并加载符号。

然后所有可用符号都会显示出来。这样您便可以：

- 写入新值：
  在左侧树中选择一个符号变量，如**计数器**。
  在右侧的**值**编辑字段中输入一个新值，然后点击**使用 WriteSymbol 写入**。
  新值将写入 ADS 服务器。

  通过**使用 WriteSymbol 写入**功能写入新值后，C# 应用程序也会收到一个带有新值的回调。
- 订阅以在值发生变化时接收回调：
  在左侧树中选择一个图标，如**计数器**。
  点击**查看当前符号**。



15.6 Sample07：接收 ADS 通知

本文介绍如何实现一个 TwinCAT 3 C++ 模块，该模块可接收有关其他模块上数据变化的 ADS 通知。
由于所有其他 ADS 通信都必须以类似方式实现，因此本示例是对来自 TwinCAT 3 C++ 模块的 ADS 通通信的通用入口。

下载

您可以在此访问该示例的源代码：

https://github.com/Beckhoff/TC1300_Samples/tree/main/S07-AdsNotifications

1. 利用已安装 TwinCAT 的 Visual Studio，通过**打开项目.....**来打开项目
2. 通过右键单击**项目 -> 属性 -> Tc 签名**打开 TwinCAT 签名，为该项目配置签名，必要时配置证书和密码。有关签署 C++ 项目的更多信息，请参见 TwinCAT [▶ 22] 章节。
3. 选择目标系统。
4. 构建示例（例如**构建 -> 构建解决方案**）。
5. 点击 激活配置。

⇨ 示例已做好操作准备。

功能说明

本示例介绍了在 TwinCAT 3 C++ 模块中接收 ADS 通知的过程。

为此，解决方案包含 2 个模块。

- 一个 C++ 模块，用于查询变量的 ADS 通知。

- 便于理解的简单示例：一个提供变量 MAIN.PlcVar 的 PLC 程序。
  如果其值发生变化，则会向 C++ 模块发送 ADS 通知。
- C++ 模块使用消息记录选项。为了更好地理解代码，只需启动示例，注意更改 PLC 模块的 Main.PlcVar 值时的输出/错误日志记录。

该地址是在模块转换 PREOP -> SAFEOP (SetObjStatePS) 期间准备的。CycleUpdate 方法包含一个简单的状态机，可发送所需的 ADS 指令。相应的方法显示接收确认。

当收到通知时，会调用继承和重载方法 AdsDeviceNotificationInd。

在停机过程中，ADS 消息会在转换过程中发送，目的是注销 (SetObjStateOS)，而模块会等待接收确认，直至超时。

启动模块开发

借助 ADS 端口向导创建 TwinCAT 3 C++ 模块。这样便可设置建立 ADS 通信所需的一切操作。如示例所示，只需使用并覆盖 ADS.h 中所需的 ADS 方法即可。

另请参见

ADS 通信 [▶ 188]

15.7 Sample08：提供 ADS-RPC

本文介绍了 ADS 可通过任务调用的方法的实现。

下载

您可以在此访问该示例的源代码：

https://github.com/Beckhoff/TC1300_Samples/tree/main/S08-ADSRPC

1. 利用已安装 TwinCAT 的 Visual Studio，通过**打开项目.....**来打开项目
2. 通过右键单击**项目 -> 属性 -> Tc 签名**打开 TwinCAT 签名，为该项目配置签名，必要时配置证书和密码。
   有关签署 C++ 项目的更多信息，请参见 TwinCAT [▶ 22] 章节。
3. 选择目标系统。
4. 构建示例（例如**构建 -> 构建解决方案**）。
5. 点击  激活配置。

⇨ 示例已做好操作准备。

功能说明

下载内容包含 2 个项目：

- TwinCAT 项目，其中包含一个 C++ 模块。其中提供了一些可以由 ADS 调用的方法。
- 此外还包括一个 Visual C++ 项目，该项目作为客户端调用用户模式的方法。

提供并调用四种具有不同签名的方法。这些方法被安排在两个接口中，这样方法的 ADS 符号名称构成便清晰明了。

示例说明

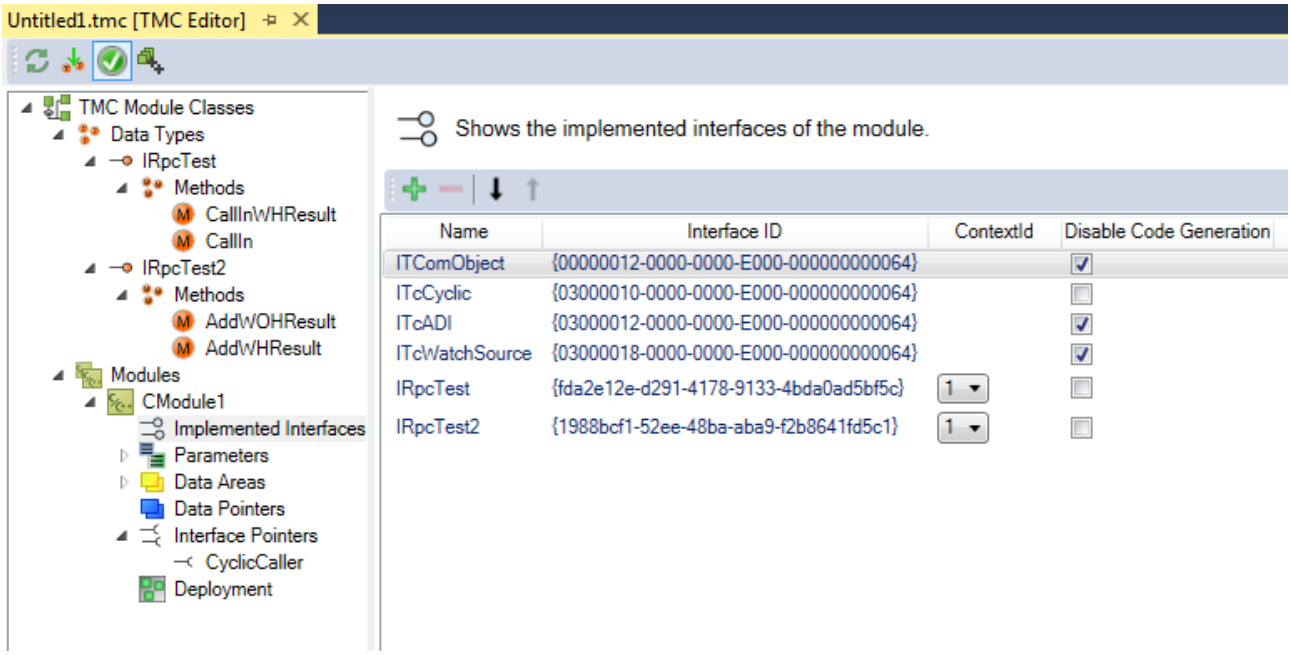
本示例由提供 RPC 方法的 TwinCAT C++ 模块和调用这些方法的 C++ 示例程序组成。

TwinCAT C++ 模块

TwinCAT C++ 项目包含一个模块和一个名为“foobar”的模块实例。

RPC 方法是普通方法，由 TMC 编辑器中的接口进行功能说明，并通过 **RPC 启用**复选框另外启用。TMC 编辑器说明 [▶ 86]中对这些选项有更详细的介绍。

本模块介绍并实现了两个接口，可以在 TMC 编辑器中看到：



这些方法共有四种，其调用值和返回值的签名各不相同。

ADS 符号名称根据模式形成：`ModulInstance.Interface#Methodenname`
ContextId 定义了执行的环境，在实现模块中尤为重要。

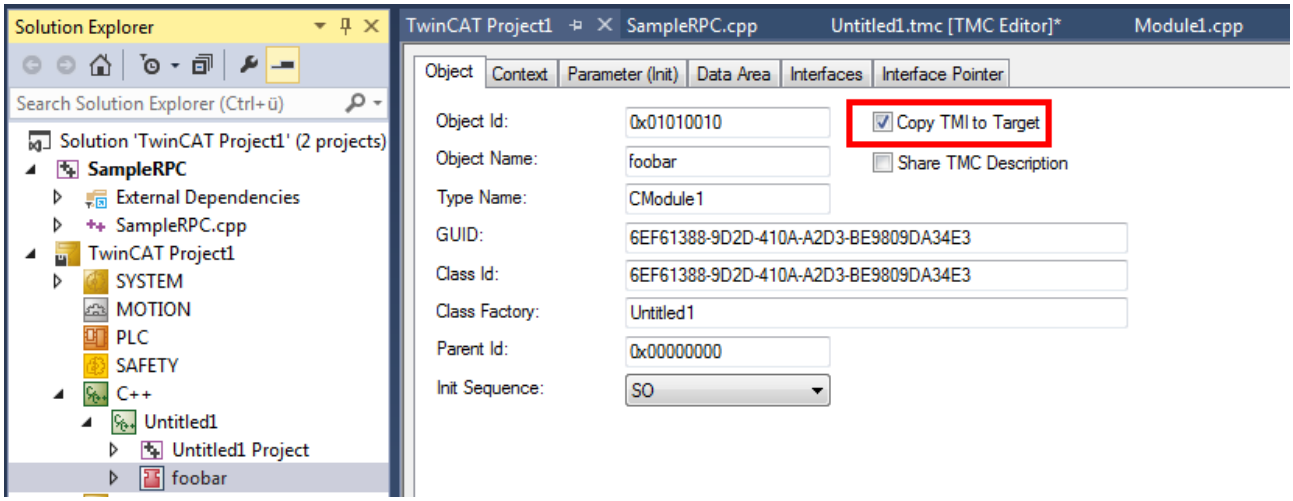
从 C++ 代码本身可以看出，这些方法由代码生成器生成，并像 TcCOM 模块的常规方法的形式实现。

Module1.cpp | Untitled1.tmc [TMC Editor]

Untitled1 → CModule1 → AddModuleToCaller()

```
///<AutoGeneratedContent id="ImplementationOf_IRpcTest">
HRESULT CModule1::CallInWHResult(LONG in)
{
    HRESULT hr = S_OK;
    return hr;
}

HRESULT CModule1::CallIn(LONG in)
{
    HRESULT hr = S_OK;
    return hr;
}
///</AutoGeneratedContent>

///<AutoGeneratedContent id="ImplementationOf_IRpcTest2">
HRESULT CModule1::AddWOHResult(LONG a, LONG b, LONG& sum)
{
        HRESULT hr = S_OK;
        sum = a+b;
        m_Trace.Log(tlAlways, FNAMEA "got called with %d %d -> %d", a, b, sum);
        return hr;
}

HRESULT CModule1::AddWHResult(LONG a, LONG b, LONG& sum)
{
    HRESULT hr = S_OK;
    sum = a + b;
    m_Trace.Log(tlAlways, FNAMEA "got called with %d %d -> HRESULT %d ", a, b, sum);
    return hr;
}
///</AutoGeneratedContent>
```

如果要在目标系统中获得方法的类型信息，可以将模块的 TMI 文件传输到目标系统。



TwinCAT OPC UA 服务器还可提供通过 OPC UA 调用这些方法的选项，为此需要在目标系统上安装 TMI 文件。

C++ 示例客户端

启动后，C++ 客户端将立即获取句柄，随后可以多次调用这些方法；然而，在各程序之间应该有一个 [RETURN]。其他每一个键都会导致句柄的启用和程序的终止。
输出结果说明了调用情况：

```
OK: AdsSyncReadWriteReq (getHdl foobar.IRpcTest#CallIn)
OK: AdsSyncReadWriteReq (getHdl foobar.IRpcTest#CallInWHResult)
OK: AdsSyncReadWriteReq (getHdl foobar.IRpcTest2#AddWOHResult)
OK: AdsSyncReadWriteReq (getHdl foobar.IRpcTest2#AddWHResult)

Press key to call all methods

Calling foobar.IRpcTest#CallIn
Send: 0

Calling foobar.IRpcTest#CallInWHResult
Value given: 1
ReturnCode: 0

Calling foobar.IRpcTest2#AddWOHResult
Value given A: 1
Value given B: 2
Value got (A+B): 3

Calling foobar.IRpcTest2#AddWHResult
Value given A: 1
Value given B: 2
ReturnCode: 0
Value got (A+B): 3
```

15.8 Sample10：模块通信：数据指针的应用

本文介绍两个 TwinCAT 3 C++ 模块的实现，这两个模块通过一个数据指针进行通信。

下载

您可以在此访问该示例的源代码：

https://github.com/Beckhoff/TC1300_Samples/tree/main/S10-Mod2ModDataPointer

1. 利用已安装 TwinCAT 的 Visual Studio，通过**打开项目……**来打开项目
2. 通过右键单击**项目 -> 属性 -> Tc 签名**打开 TwinCAT 签名，为该项目配置签名，必要时配置证书和密码。有关签署 C++ 项目的更多信息，请参见 TwinCAT [▶ 22] 章节。
3. 选择目标系统。

4. 构建示例（例如**构建 -> 构建解决方案**）。

5. 点击  激活配置。

⇨ 示例已做好操作准备。

功能说明

这种通信以“分割”数据区为基础：由一个模块提供，另一个模块通过指针访问。

在输出或输入数据区中，无法将两个不同的数据指针与同一个条目联系起来；如果没有这一限制，可能会出现同步问题。因此，ModuleDataProvider 模块会将输入和输出合并到一个标准数据区，不受此限制。

总之，本示例包括以下模块：

- ModuleDataProvider 提供一个数据区，其他模块可以访问该区域。
  数据区包含 4 位（2 个用于输入，2 个用于输出）和 2 个整数（1 个用于输入，1 个用于输出）。
- ModuleDataInOut 提供“常规”输入变量（写入 ModuleDataProvider 的数据区）和输出变量（从数据区
  读取）。这一 CModuleDataInOut 类的实例是对真实 IO 的模拟。
- ModuleDataAccessA 可访问 ModuleDataProvider 的数据区，并对 Bit1、BitOut1 以及整数数据进行周期性处理。
- ModuleDataAccessB 可访问 ModuleDataProvider 的数据区，并对 Bit2、BitOut2 以及整数数据进行周期性处理。

示例用户通过设置变量 ValueIn/Bit1/Bit2 触发 ModuleDataInOut：

- 设置输入 Bit1 时，也会相应设置输出 Switch1。
- 设置输入 Bit2 时，也会相应设置输出 Switch2。
- 设置输入 ValueIn 时，输出 ValueOut 会在每个周期内递增两次。

所有模块均配置为具有相同的任务环境，此为必要设置，因为通过指针访问不会提供同步跟踪机构。执行顺序与环境配置选项卡上指定的顺序一致。该值将作为参数 SortOrder 传递，并存储在循环任务调用器 (m_spCyclicCaller) 的智能指针中，其中还包含循环任务调用器的对象 ID。

示例说明

模块 ModuleDataInOut 有输入和输出变量。它们与数据提供程序的相应变量相关联。

模块 ModuleDataProvider 会提供输入和输出数据数组，并实现 ITcIoCylic 接口。InputUpdate 方法会将输入变量中的数据复制到标准数据区“数据”的 DataIn 符号中，而 OutputUpdate 方法则会将 DataOut 符号中的数据复制到输出变量中。

模块 ModuleDataAccessA 和 ModuleDataAccessB 包含通过链接指向数据提供程序数据区的指针。这些指针会在从 SAFEOP 到 OP 的转换过程中进行初始化。ModuleDataAccessA 根据 Bit1 循环设置 BitOut1。相应地，ModuleDataAccessB 根据 Bit2 循环设置 BitOut2。二者均通过内部计数器与 ValueIn 值相乘来递增 ValueOut。

由于没有通过数据指针的同步跟踪机构，所有模块都必须在同一环境中执行，这一点非常重要。执行顺序由各模块**环境**选项卡上的**排序顺序**确定。这将作为参数 **SortOrder** 提供给 SmartPointer (m_SpCyclicCaller)，其中还包括 ObjectID。

15.9 Sample11：模块通信：PLC 模块到 C++ 模块的方法调用

本文介绍了实现情况：

- 一个 C++ 模块 [▶ 257]，可提供用于控制状态机的方法。
  按照该分步介绍，实现可提供状态机接口的 C++ 模块。

- 用于调用 C++ 模块功能的 PLC 模块 [▶ 271]。
  PLC 与 C++ 模块之间不存在硬编码链接，这一点非常重要。相反，调用的 C++ 实例可以在系统管理器中进行配置。
  按照该分步介绍，了解如何实现从 C++ 模块调用方法的 PLC 项目。

下载

您可以在此访问该示例的源代码：

https://github.com/Beckhoff/TC1300_Samples/tree/main/S11-Mod2ModMethod

1. 利用已安装 TwinCAT 的 Visual Studio，通过**打开项目.....**来打开项目
2. 通过右键单击**项目 -> 属性 -> Tc 签名**打开 TwinCAT 签名，为该项目配置签名，必要时配置证书和密码。
   有关签署 C++ 项目的更多信息，请参见 TwinCAT [▶ 22] 章节。
3. 选择目标系统。
4. 构建示例（例如**构建 -> 构建解决方案**）。
5. 点击  激活配置。

⇨ 示例已做好操作准备。

15.9.1 提供方法的 TwinCAT 3 C++ 模块

本文介绍了 TwinCAT 3 C++ 模块的创建过程，该模块可提供一个带有多个方法的接口，可被 PLC 调用，也可被其他 C++ 模块调用。

我们的想法是用 C++ 创建一个简单的状态机，其他模块可以从外部启动和停止该状态机，但也可以设置或读取 C++ 状态机的特定状态。

还有两篇文章使用了该 C++ 状态机的结果。

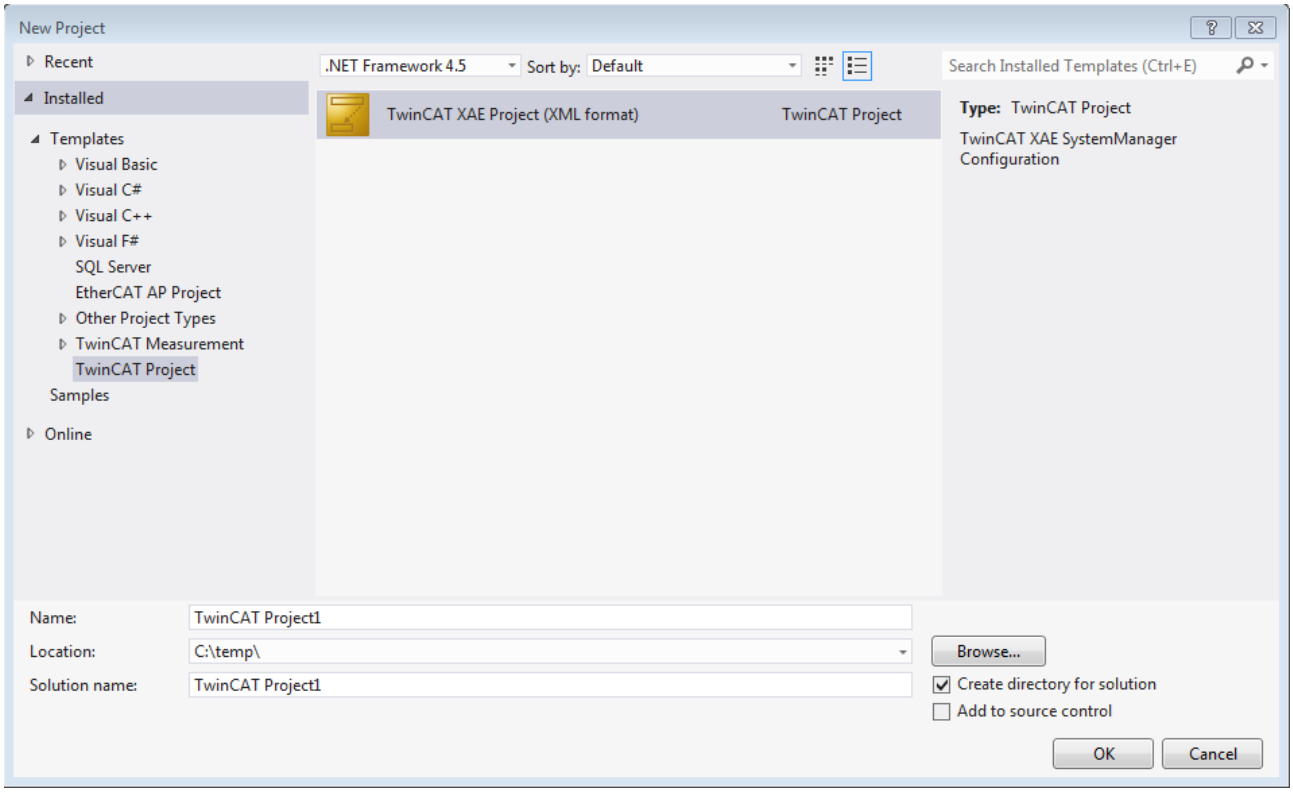
- 从 PLC 逻辑调用功能 [▶ 256]，即影响 PLC 的 C++ 代码。
- 从 C++ 逻辑调用功能 [▶ 283]，即两个 C++ 模块之间的交互。

本文介绍：

- 第 1 步：创建新的 TwinCAT 3 项目 [▶ 257]。
- 第 2 步：创建新的 TwinCAT 3 C++ 驱动程序 [▶ 258]。
- 第 3 步：创建新的 TwinCAT 3 接口。
- 第 4 步：向接口添加方法 [▶ 261]。
- 第 5 步：向模块添加新接口。
- 第 6 步：启动 TwinCAT TMC Code Generator，生成模块类功能说明的代码 [▶ 266]。
- 第 7 步：实现成员变量和构造函数。
- 第 8 步：实现方法。
- 第 9 步：实现周期性更新。
- 第 10 步：编译代码 [▶ 269]
- 第 11 步：创建 C++ 模块实例。
- 第 12 步：完成。检查结果。

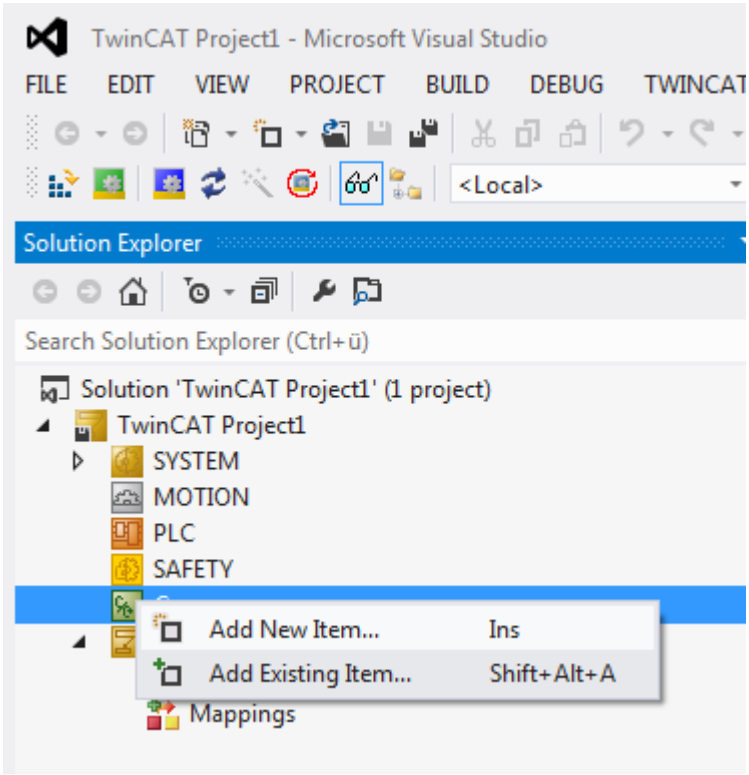
第 1 步：创建 TwinCAT 3 项目

首先，像常规流程创建一个 TwinCAT 项目。

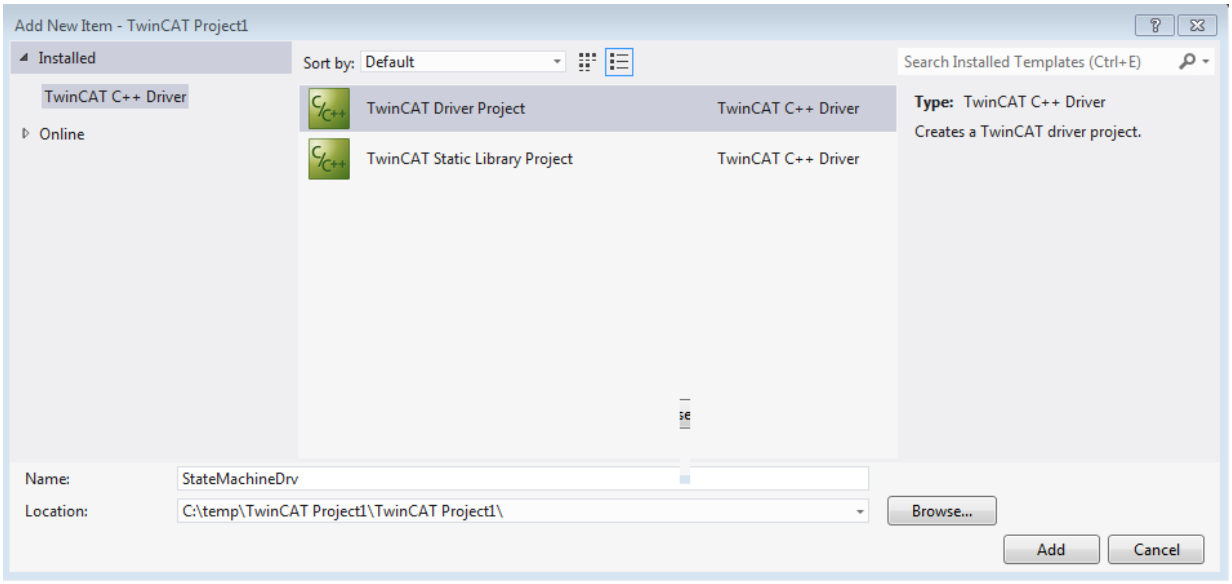


第 2 步：创建新的 TwinCAT 3 C++ 驱动程序

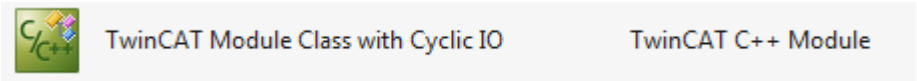
1. 右键单击 **C++** 和**添加新项目...**⋯



2. 选择 TwinCAT Driver Project 模板并输入驱动程序名称，在本示例中为“StateMachineDrv”。点击**添加**继续操作。

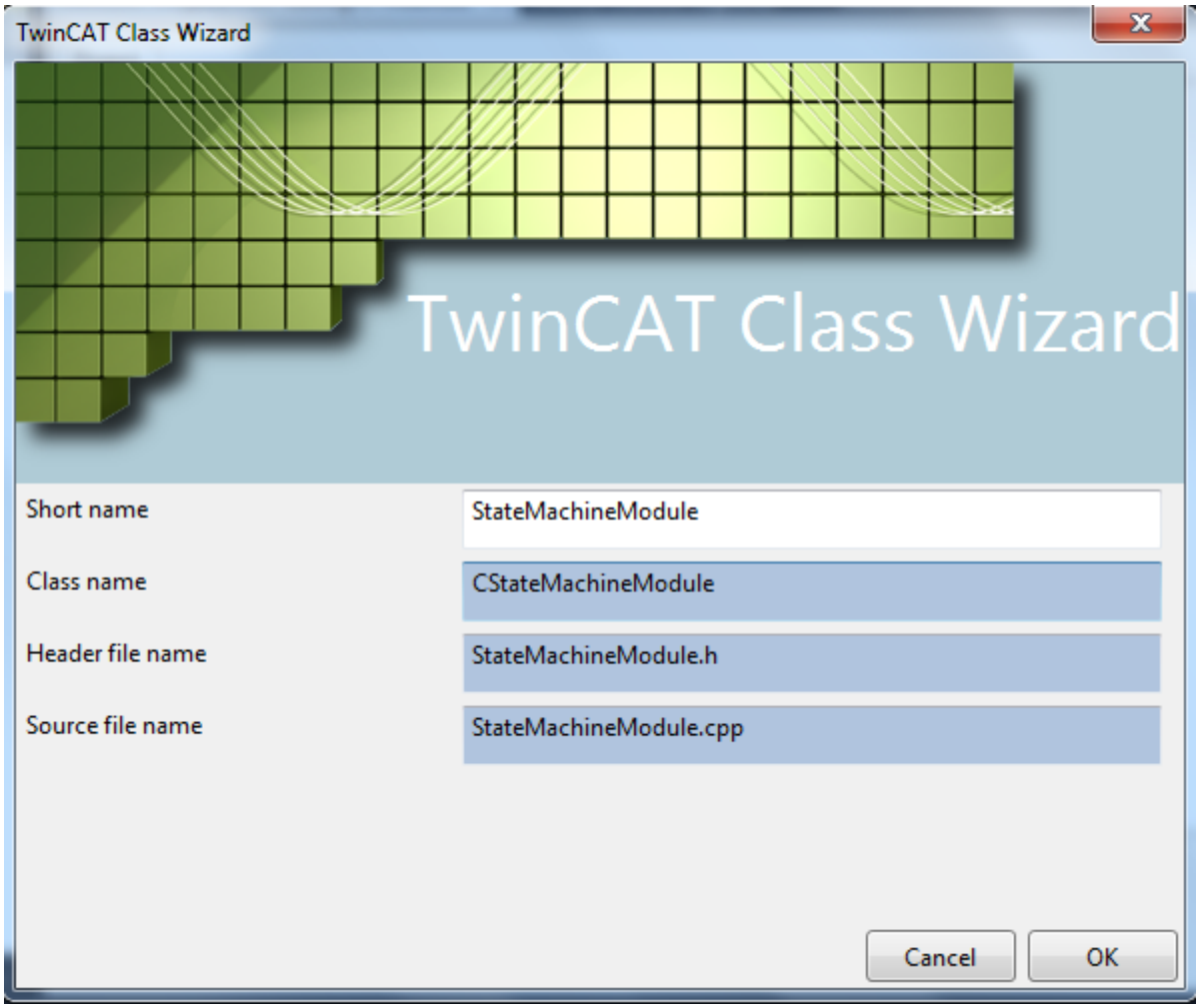


3. 为新驱动程序选择一个模板。在本示例中选择的是“带有周期性 IO 的 TwinCAT Module Class”，原因是状态机的内置计数器可用于 IO 映射配置。
4. 点击**添加**继续操作。



5. 为 C++ 驱动程序“StateMachineDrv”中的新类指定一个名称。
模块类、报头文件和源文件的名称均来自指定的“简称”。

6. 点击**确定**继续操作。



⇨ 然后，向导会创建一个 C++ 项目，该项目可以进行无差错编译。

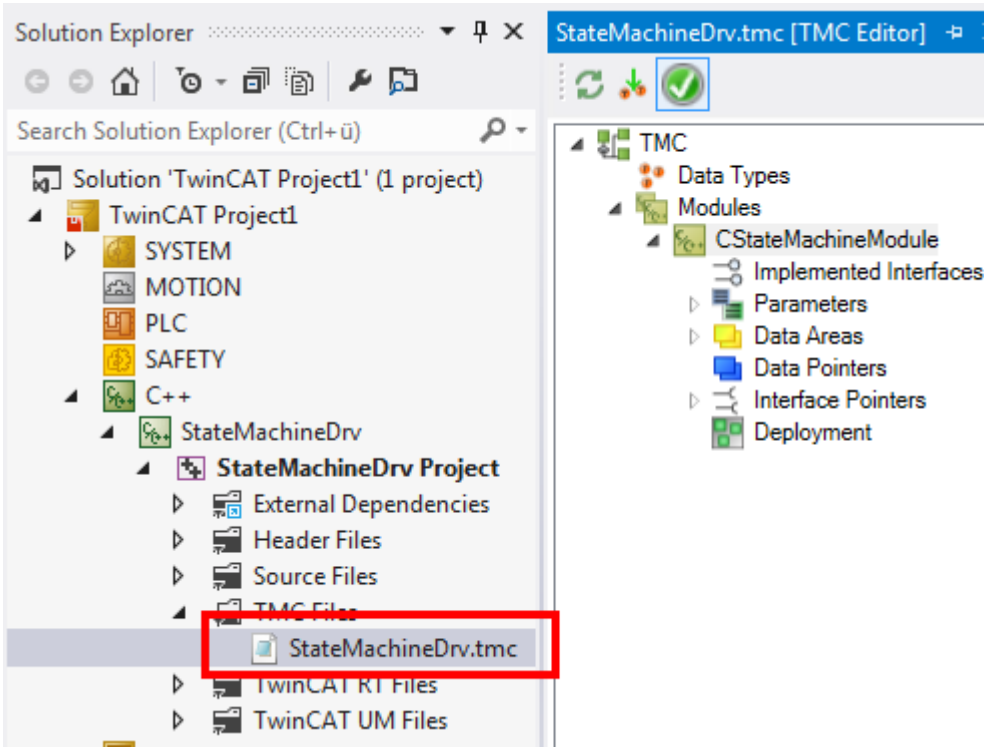
第 3 步：创建新的 TwinCAT 3 接口

注意

名称冲突

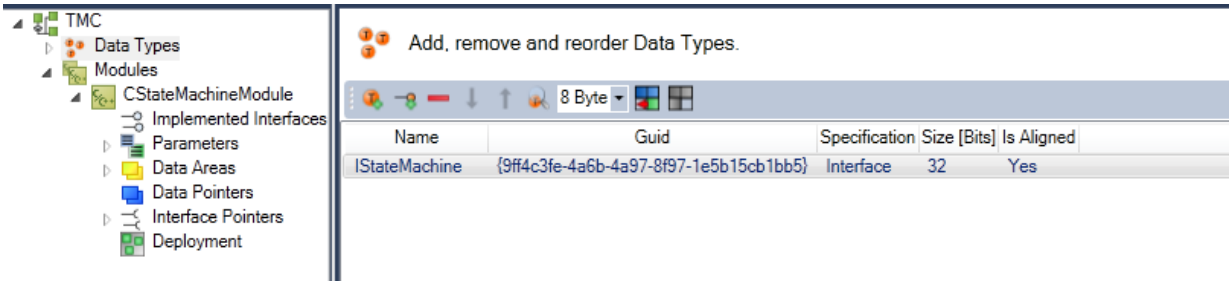
如果将驱动程序与 PLC 模块结合使用，可能会发生名称冲突。
不要将任何为 PLC 保留的关键字用作名称。

1. 双击 **StateMachineDrv.tmc**，启动 TMC 编辑器。



2. 在 TMC 编辑器中选择**数据类型**。

3. 点击**添加新接口**  ，添加新的接口。
   ⇨ 然后会列出一个新条目 **IInterface1**。
4. 双击打开 **IInterface1**，更改接口属性。
5. 输入一个有意义的名称，在本示例中为“IStateMachine”。

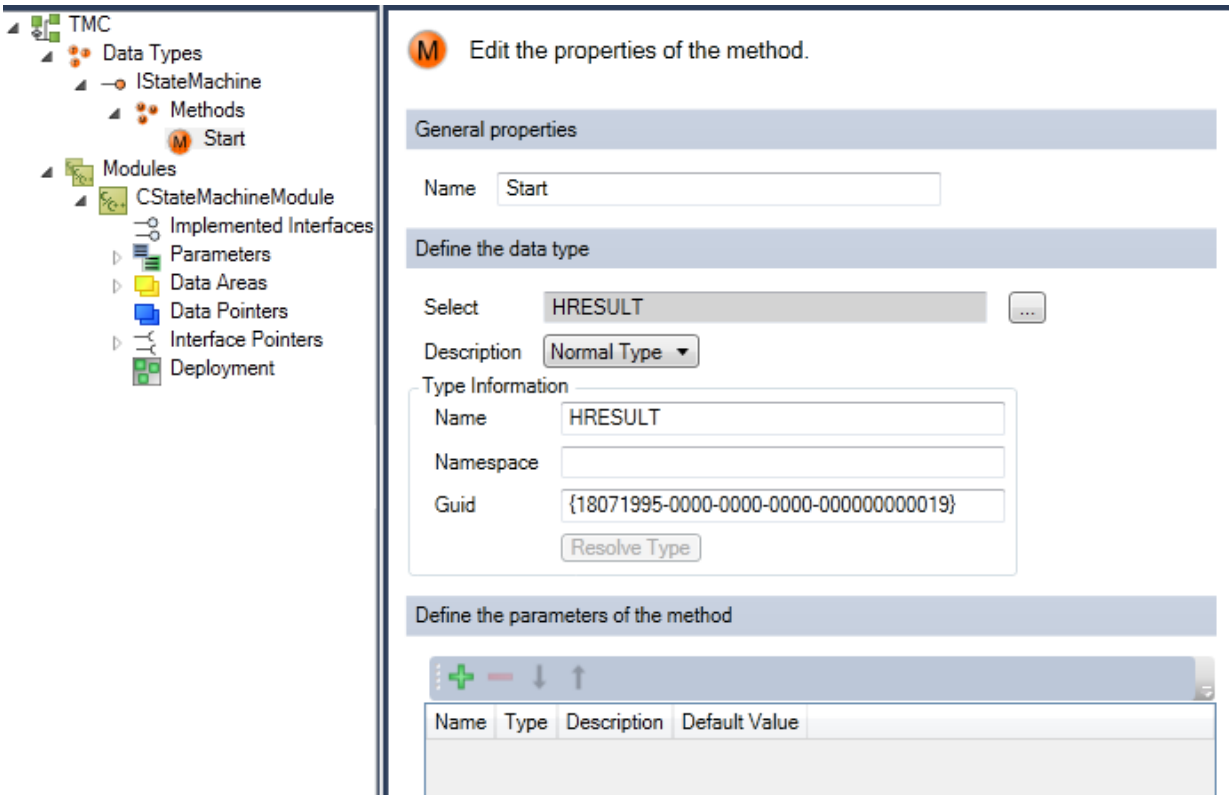


⇨ 接口已创建完成。

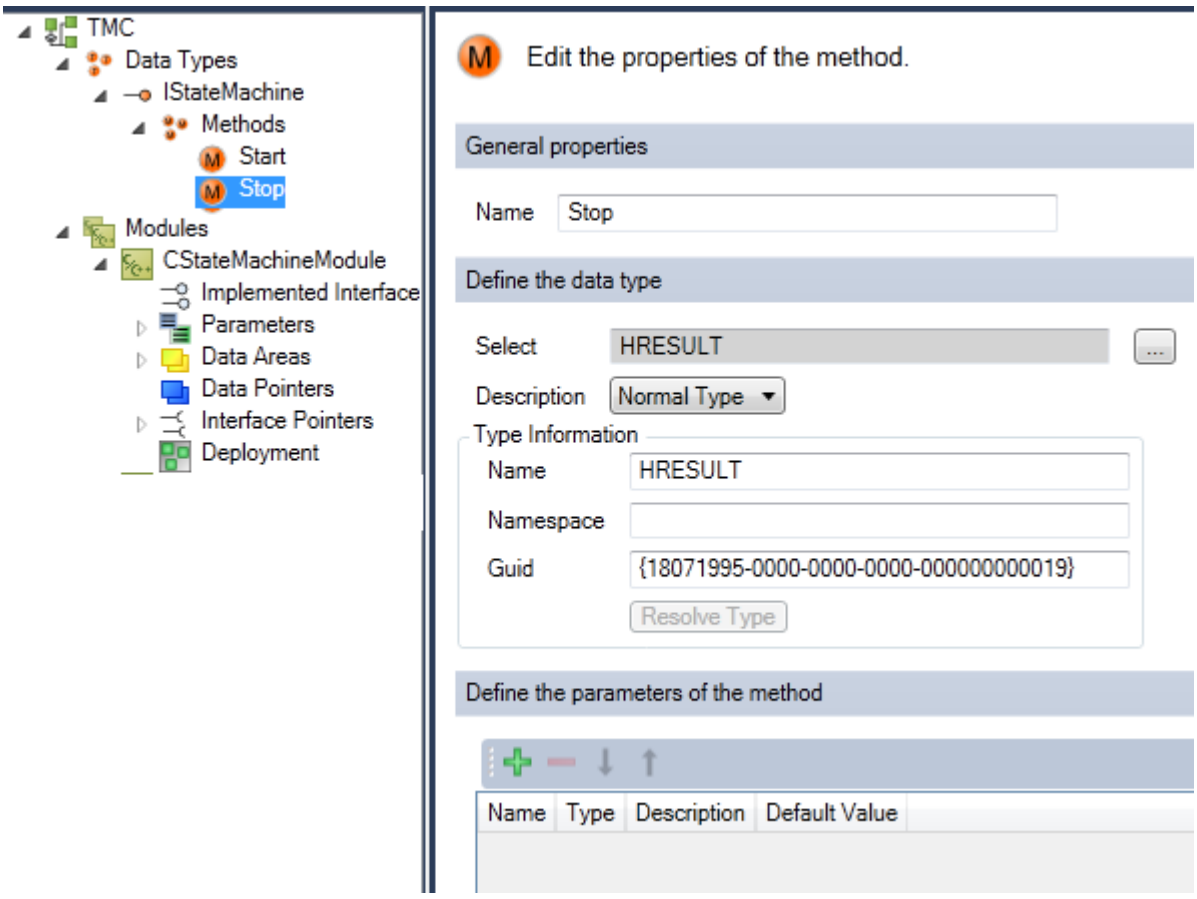
第 4 步：向接口添加方法

1. 点击**编辑方法.....**获取该接口的方法列表：
   点击 **+** 按钮创建新的默认方法 Method1。

2. 用一个更有意义的名称（本示例中为“Start”）替换默认名称 Method1。

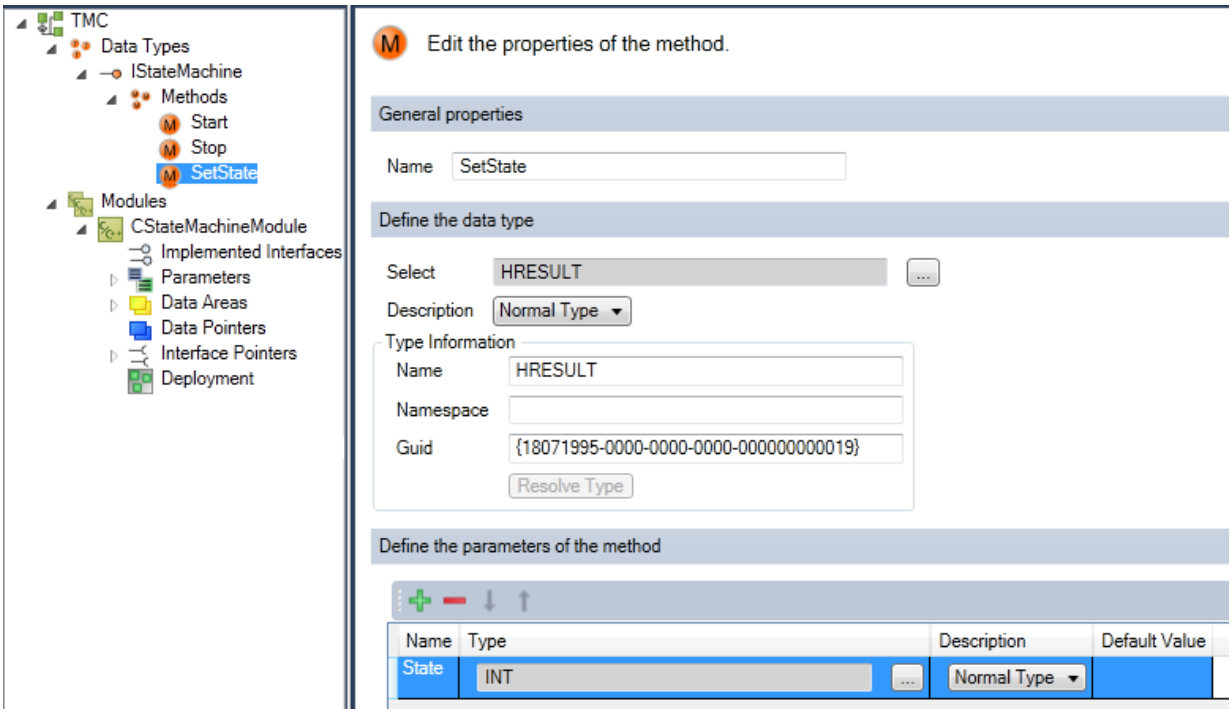


3. 添加第二个方法，命名为“Stop”。



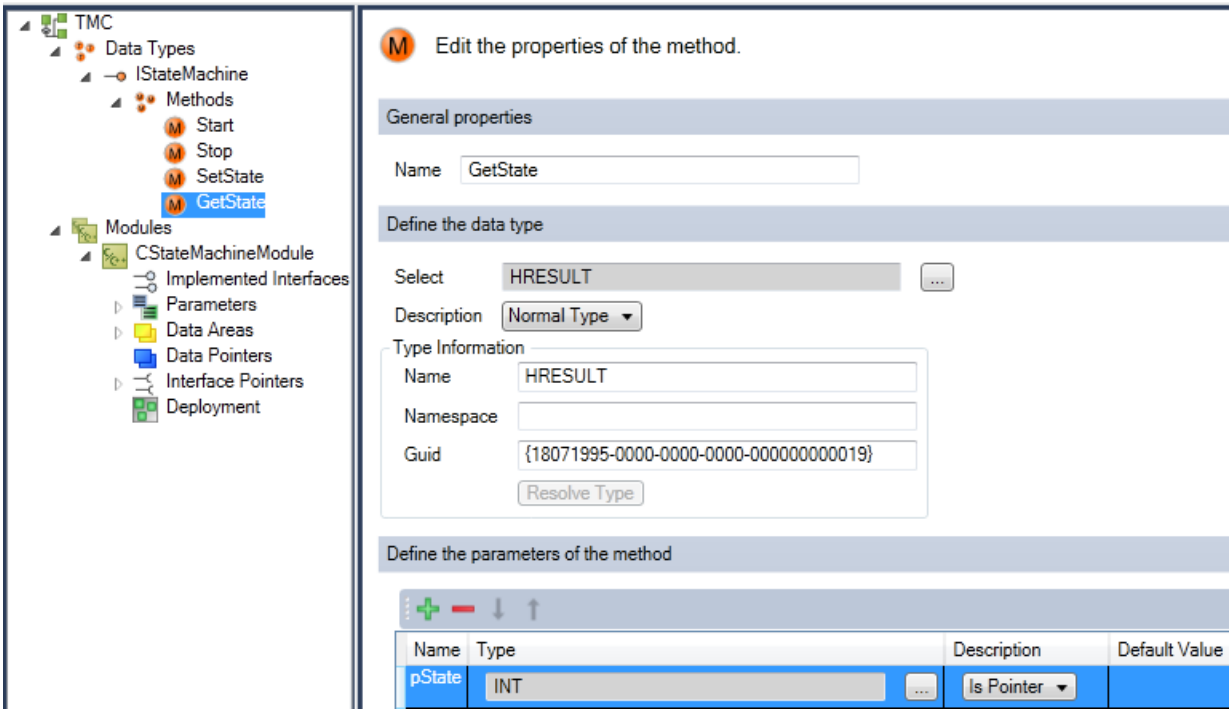
4. 添加第三个方法，命名为“SetState”。

5. 随后，您可以点击**添加新参数**来添加参数，或编辑 SetState 方法的参数。

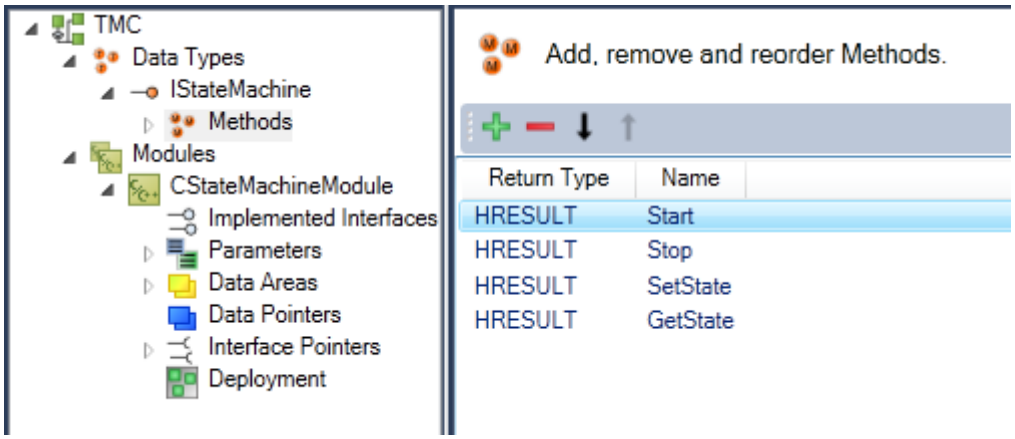


⇨ 新参数 Parameter1 默认生成为**正常类型 INT**。

6. 点击名称“Parameter1”，将编辑框中的名称改为“State”。
7. 在定义“Start”、“Stop”和“SetState”之后，再定义一个方法。
8. 将其重命名为“GetState”。
9. 添加一个参数，并将其命名为“pState”（设计为稍后作为指针使用）。
10. 将“Normal Type”改为“指针类型”。

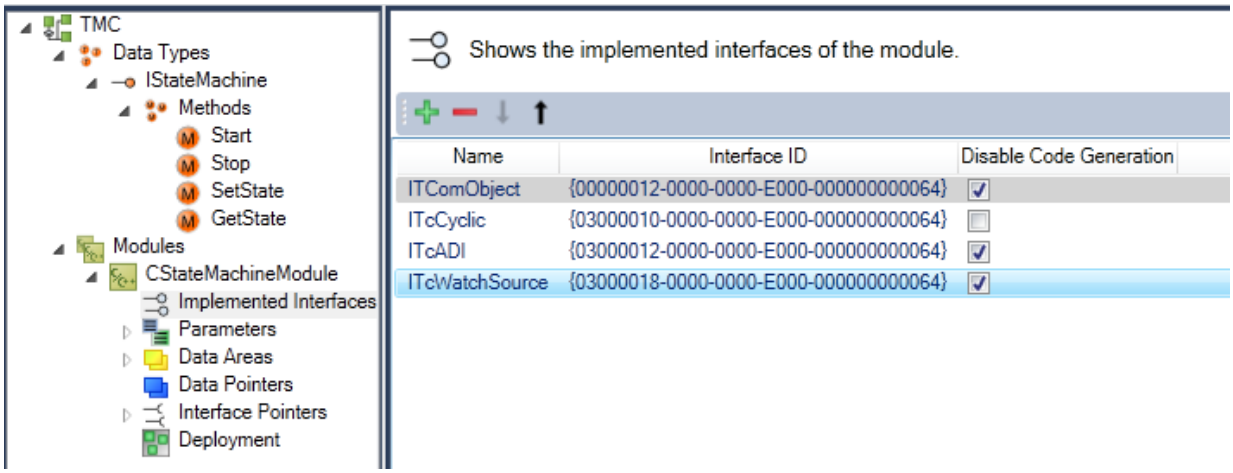


⇨ 然后，您将获得所有方法的列表。您可以使用  按钮更改方法的顺序。

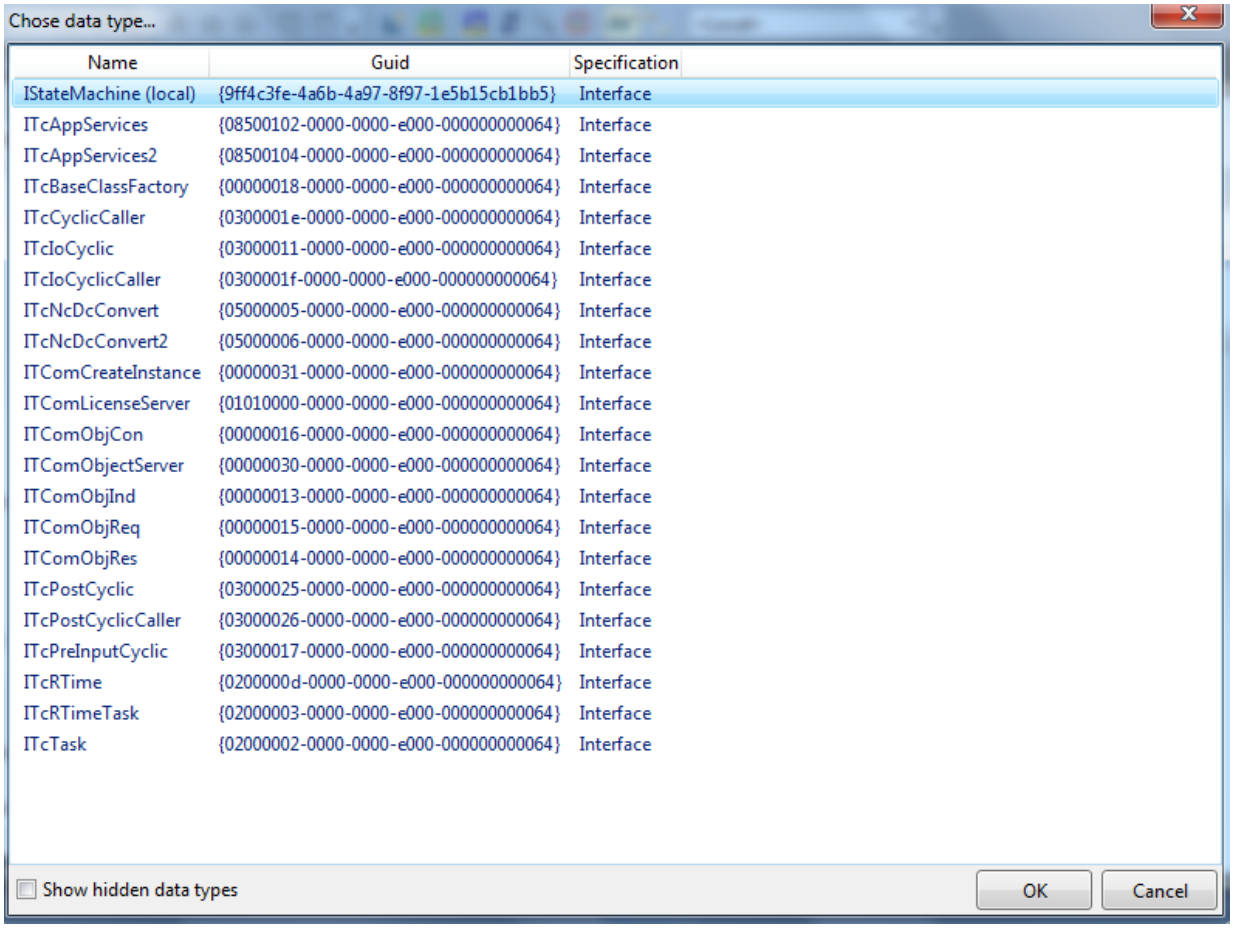


第 5 步：向模块添加新接口

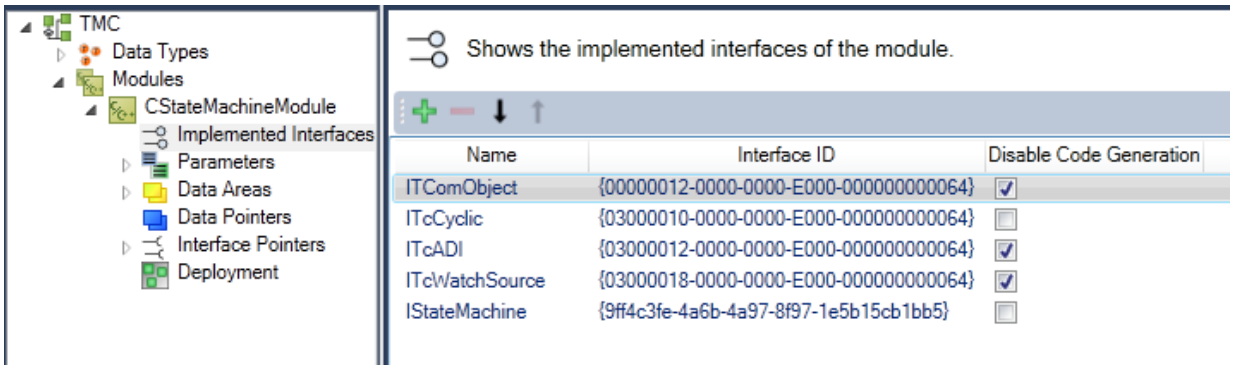
1. 选择要由新接口扩展的模块，在本例中，选择目标**模块 -> CStateMachineModule**。
2. 点击 **+** 按钮，使用**将新接口添加至模块**功能，在已实现接口的列表中添加一个新的接口。



3. 所有可用接口均已列出，选择新接口 IStateMachine，并点击**确定以结束**。

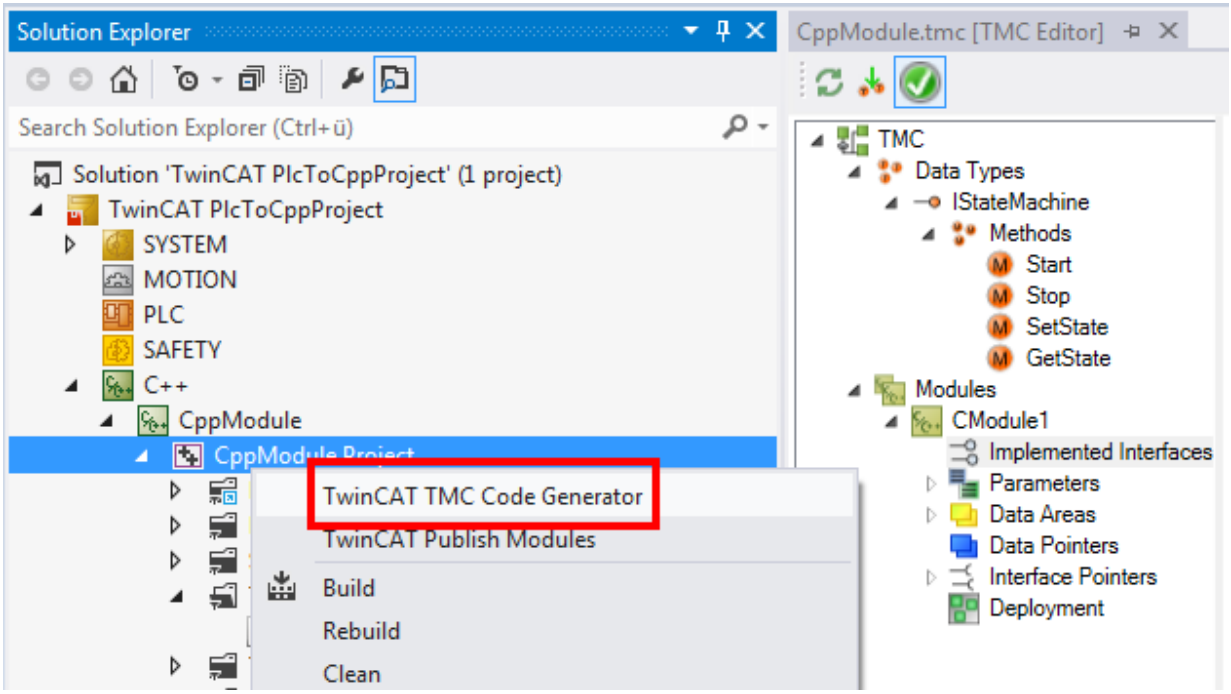


⇨ 新接口 IStateMachine 是模块说明的一部分。



第 6 步：启动 TwinCAT TMC Code Generator

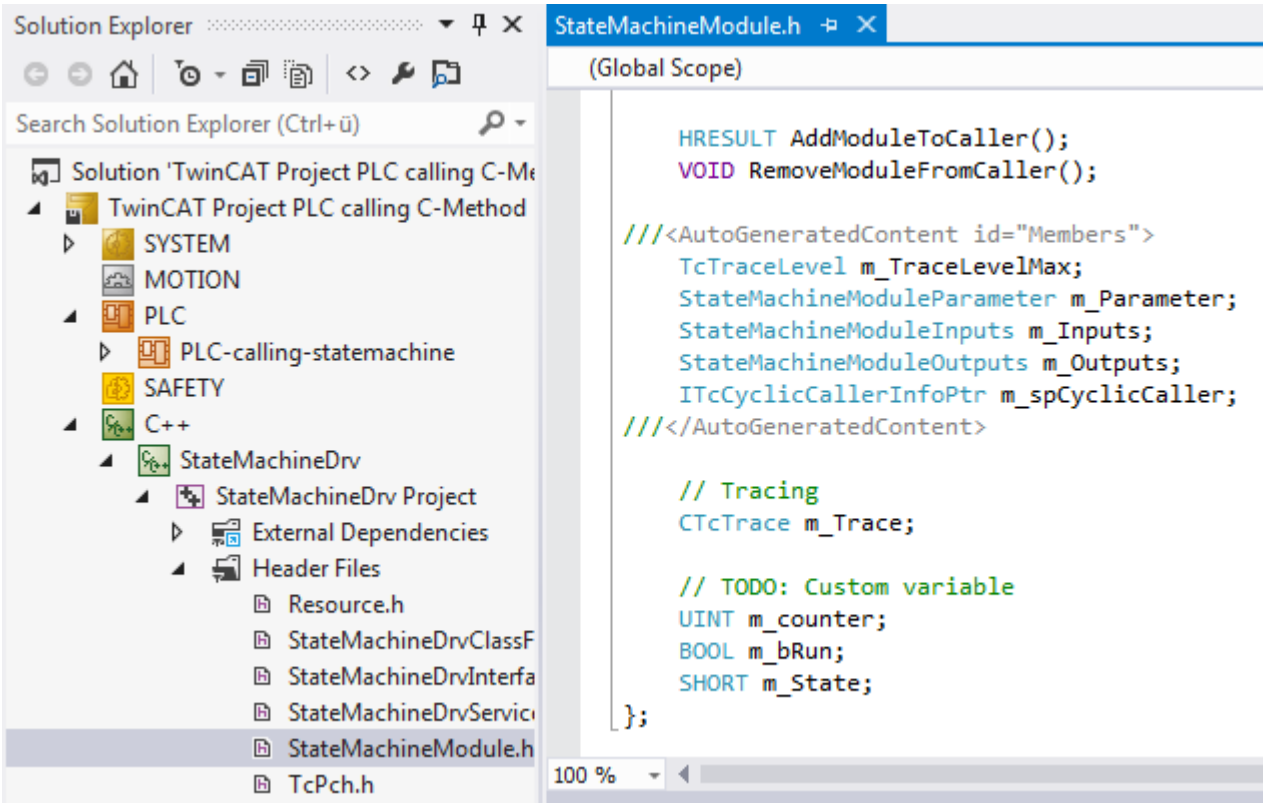
1. 为了在此模块的基础上生成 C/C++ 代码，请右键单击 C/C++ 项目，然后选择 **TwinCAT TMC Code Generator**。



⇨ 模块 StateMachineModule.cpp 现在包含新接口
CModule1：Start()
CModule1：Stop()
CModule1：SetState(SHORT State)
CModule1：GetState(SHORT* pState)。

第 7 步：实现成员变量和构造函数

将成员变量添加到报头文件 StateMachineModule.h 中。



第 8 步：实现方法

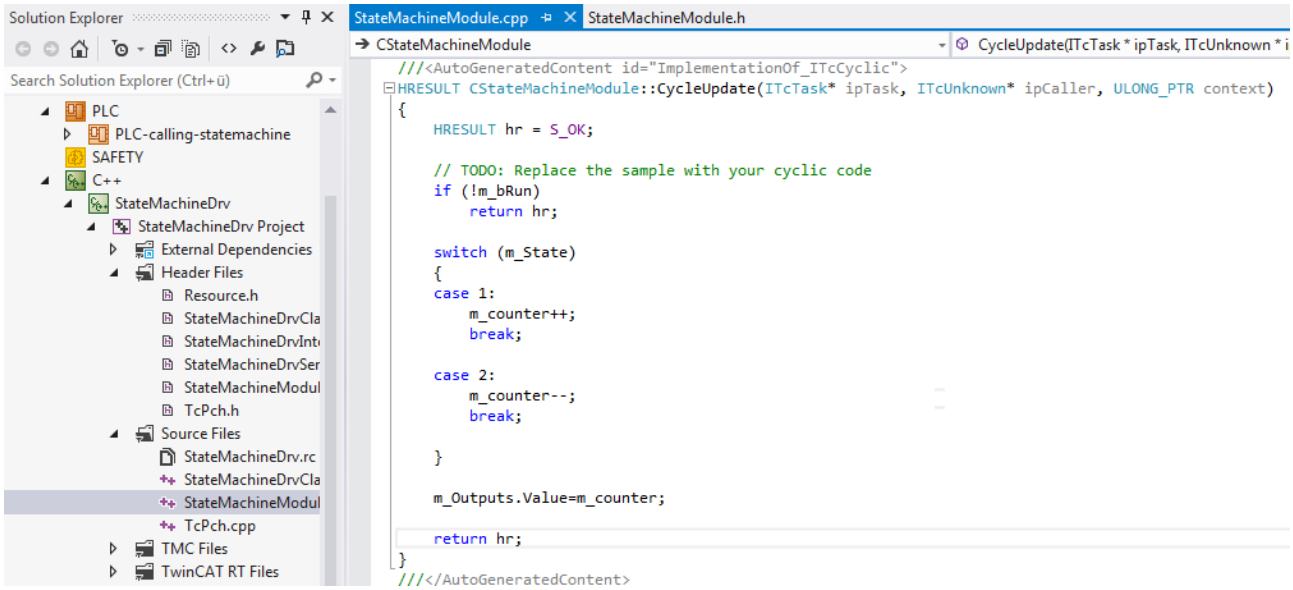
实现 StateMachineModule.cpp 中四个方法的代码：

```
///<AutoGeneratedContent id="ImplementationOf_IStateMachine">
HRESULT CModule1::Start()
{
    HRESULT hr = S_OK;
    m_bRun = TRUE;
    return hr;
}

HRESULT CModule1::Stop()
{
    HRESULT hr = S_OK;
    m_bRun = FALSE;
    return hr;
}

HRESULT CModule1::SetState(SHORT State)
{
    HRESULT hr = S_OK;
    m_State = State;
    return hr;
}

HRESULT CModule1::GetState(SHORT* pState)
{
    HRESULT hr = S_OK;
    *pState = m_State;
    return hr;
}
///</AutoGeneratedContent>
```

第 9 步：实现周期性更新

即使内部状态机处于“停止”模式，C++ 模块实例也会被周期性调用。

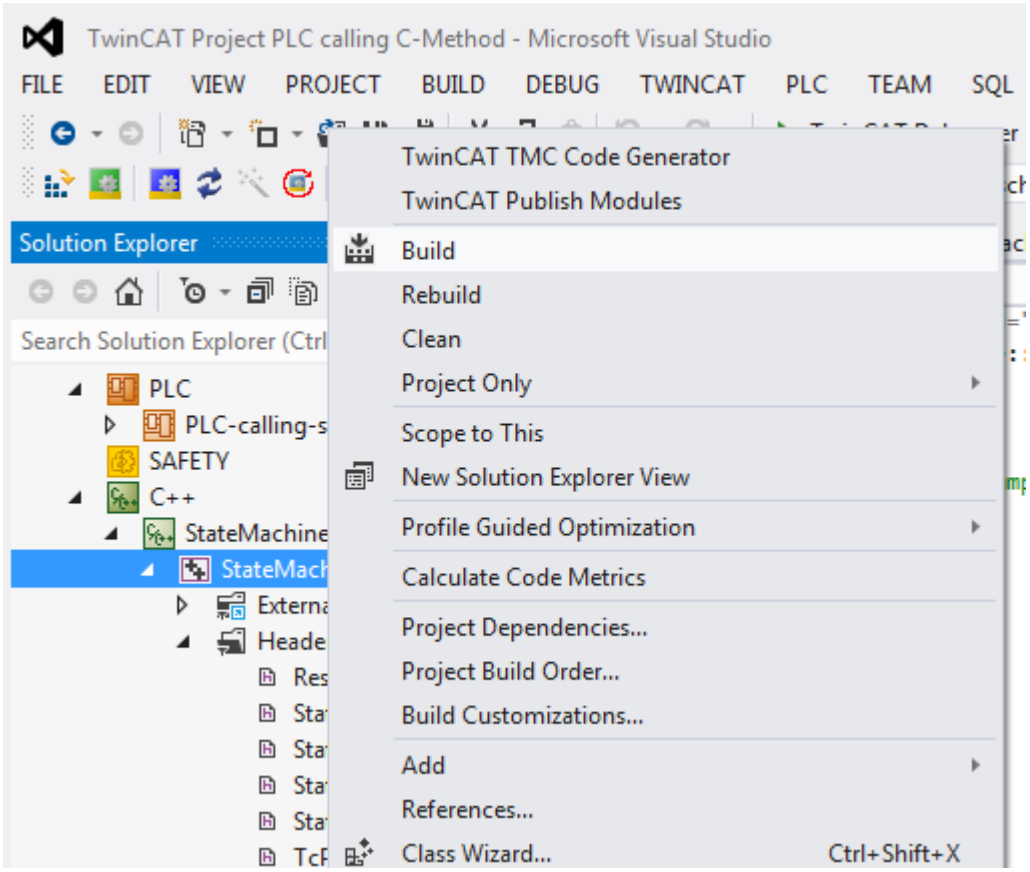
- 如果不执行状态机，则 m_bRun 标志会发出信号，表示要退出内部状态机的代码执行。
- 如果状态为“1”，则计数器必须递增。
- 如果状态为“2”，则计数器必须递减。
- 由此产生的计数器值会分配至Value，该值是数据区逻辑输出的成员变量。稍后可将其分配到物理 IO 层或其他模块的其他数据区。



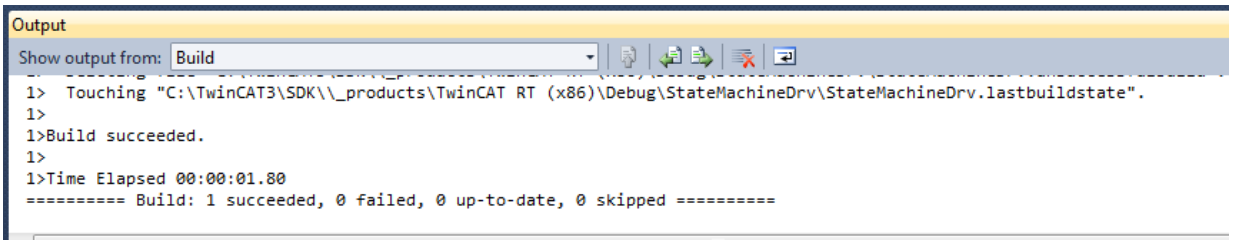
.

第 10 步：编译代码

1. 实现所有接口后，右键单击状态机并选择**构建**，即可编译代码。

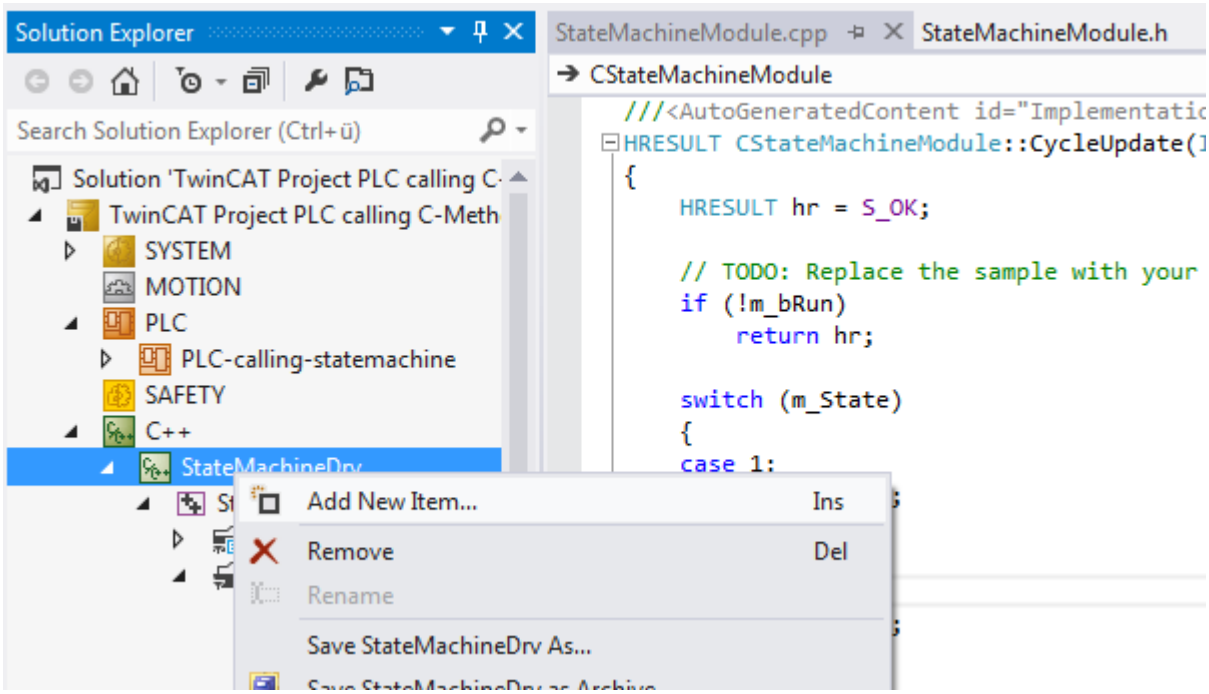


2. 重复编译并优化代码，直到结果如下所示：

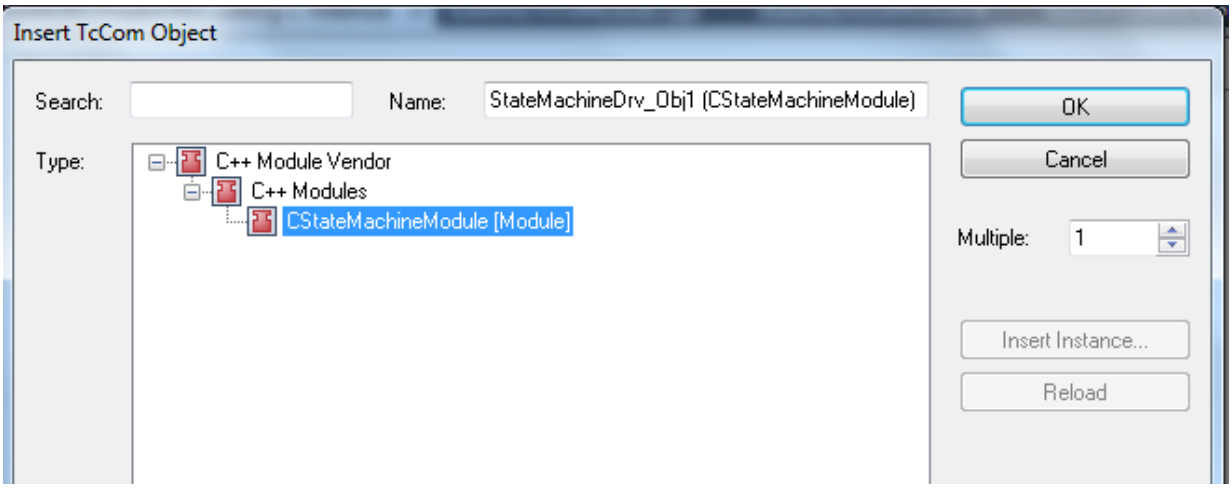


第 11 步：创建 C++ 模块实例

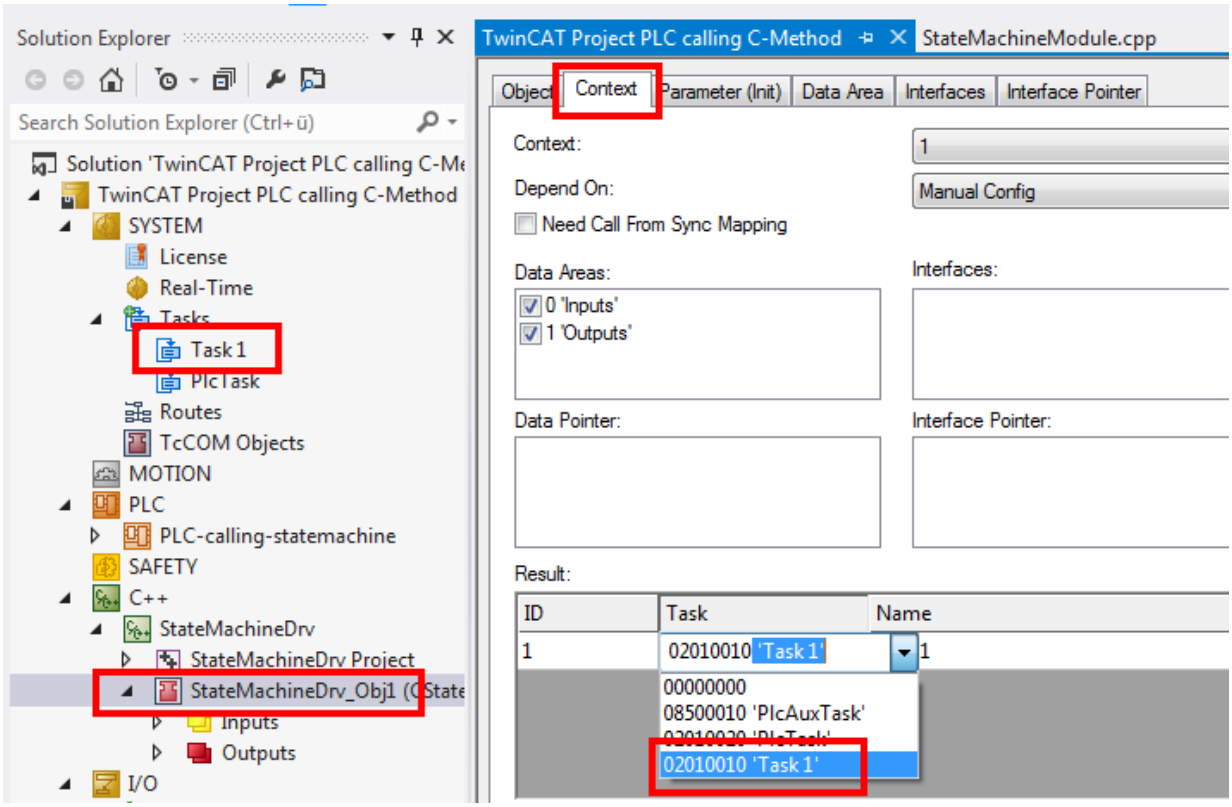
1. 右键单击 C++ 项目，选择**添加新项目……**，创建新的模块实例。



2. 选择要添加为新实例的模块，本例中为 CStateMachineModule。



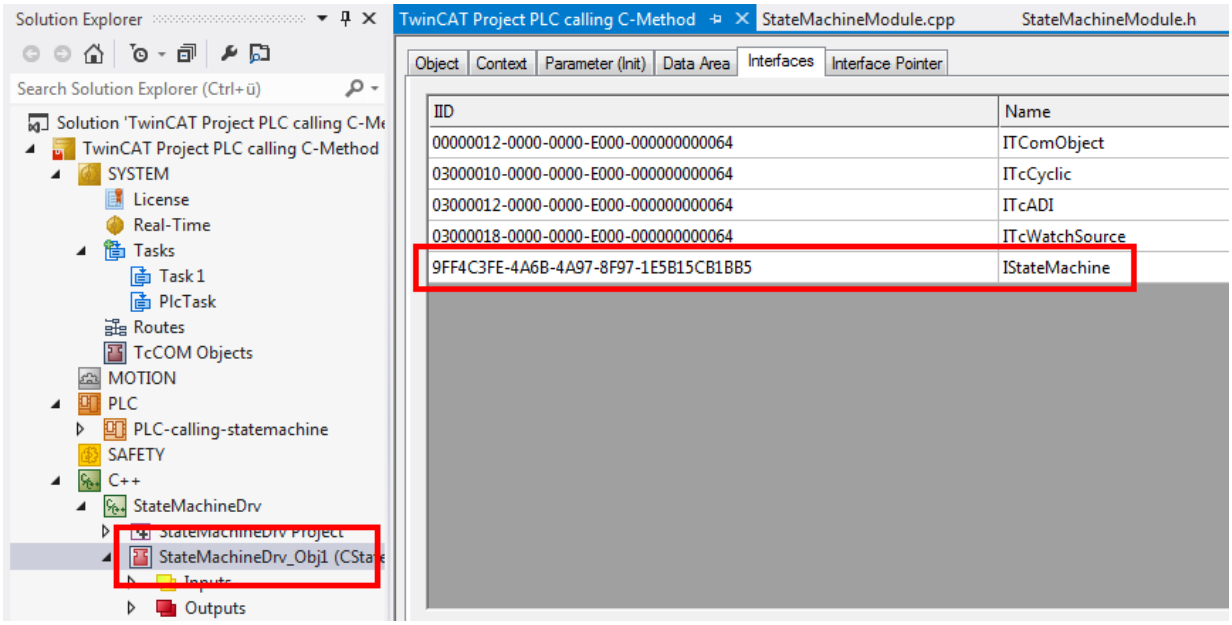
3. 将实例分配给任务：



第 12 步：完成，检查结果

1. 导航至解决方案树中列出的模块，然后选择右侧的**接口**选项卡。

⇨ 列出新接口 IStateMachine。



15.9.2 通过 PLC 调用另一个模块提供的方法

本文介绍了 PLC 如何调用另一个模块提供的方法；在本例中，调用的是先前定义的 C++ 模块。

- 第 1 步：检查可用接口。
- 第 2 步：创建新的 PLC 项目。
- 第 3 步：添加新的 FB 状态机 [▶ 273]（作为调用 C++ 模块方法的代理）。

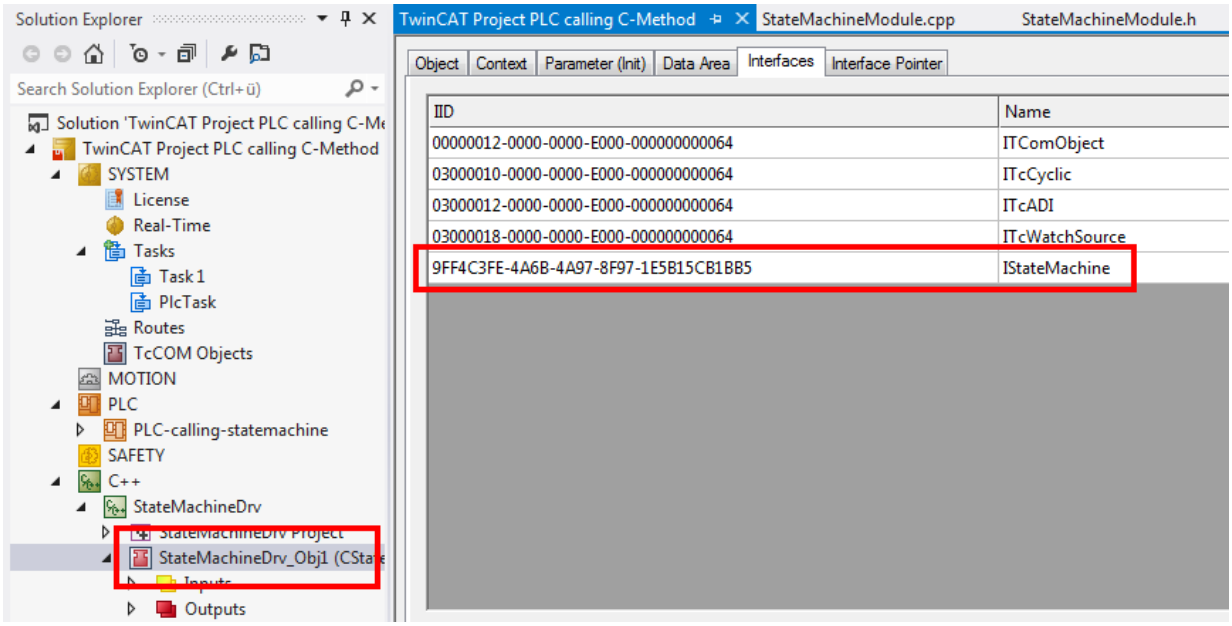
- 第 4 步：清理功能块接口。
- 第 5 步：添加 FB 方法“FB_init”和“exit”。 [▶ 277]
- 第 6 步：实现 FB 方法。
- 第 7 步：调用 PLC 中的 FB 状态机。 [▶ 281]
- 第 8 步：编译 PLC 代码。
- 第 9 步：将 PLC FB 与 C++ 实例相关联。
- 第 10 步：观察 PLC 和 C++ 两个模块的执行情况。

第 1 步：检查可用接口

选项 1：

1. 导航至 C++ 模块实例。
2. 选择**接口**选项卡。

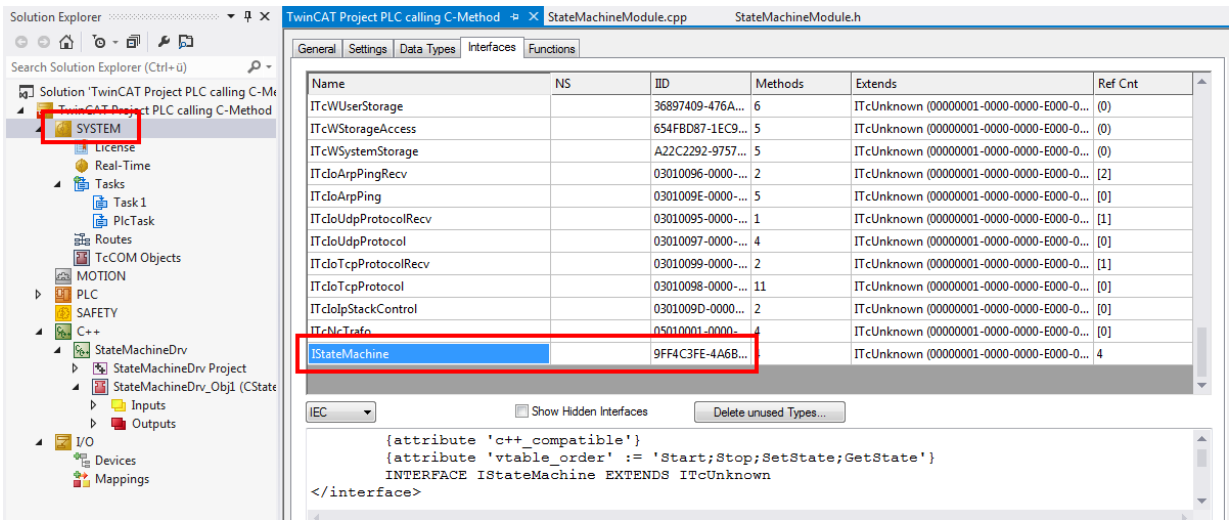
⇨ 列表中的 **IStateMachine** 接口有其特定的 IID（接口 ID）。



选项 2：

1. 导航至**系统**。
2. 选择**接口**选项卡。

⇨ 列表中的 **IStateMachine** 接口有其特定的 IID（接口 ID）。

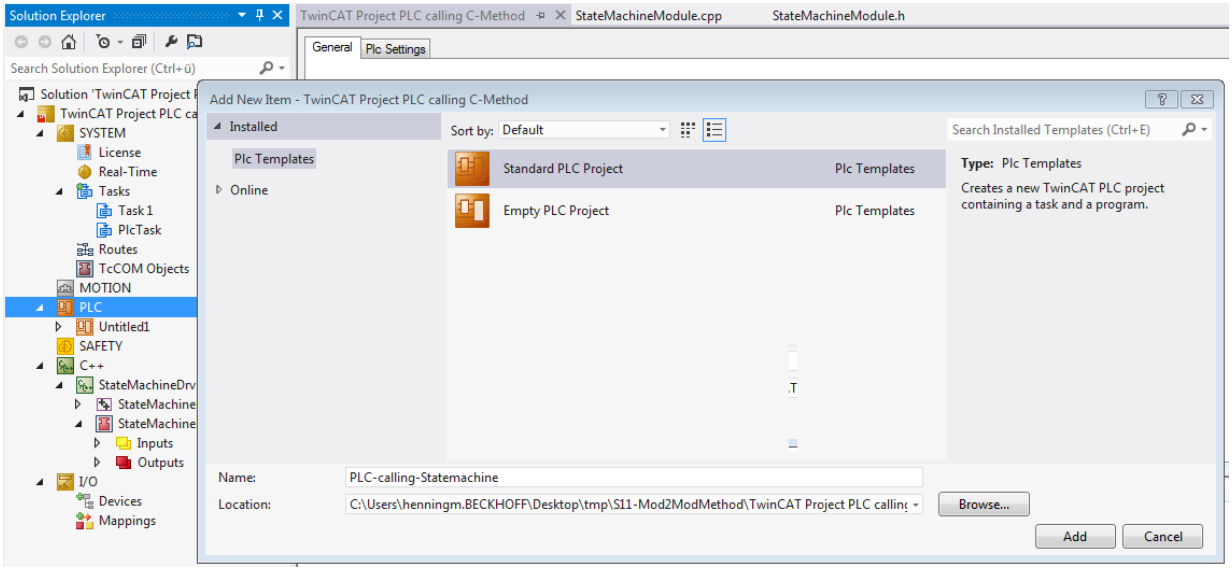


下部显示以不同编程语言存储的代码。

第 2 步：创建新的 PLC 项目

创建名为“PLC 调用状态机”的标准 PLC 项目。

1. 右键单击 PLC 节点。
2. 选择**标准 PLC 项目**。
3. 修改名称。

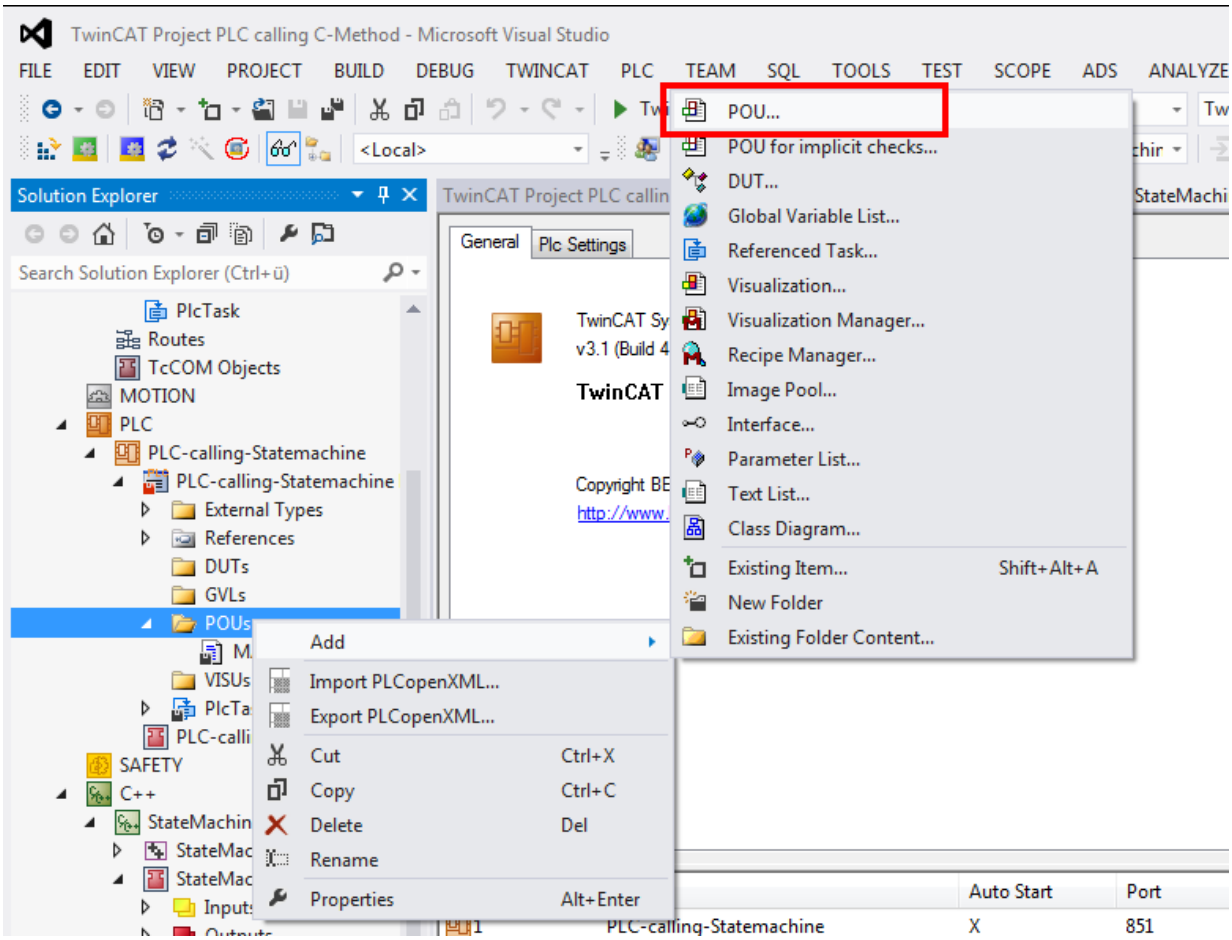


⇨ 项目已成功创建。

第 3 步：添加功能块 (FB) (作为调用 C++ 模块方法的代理)

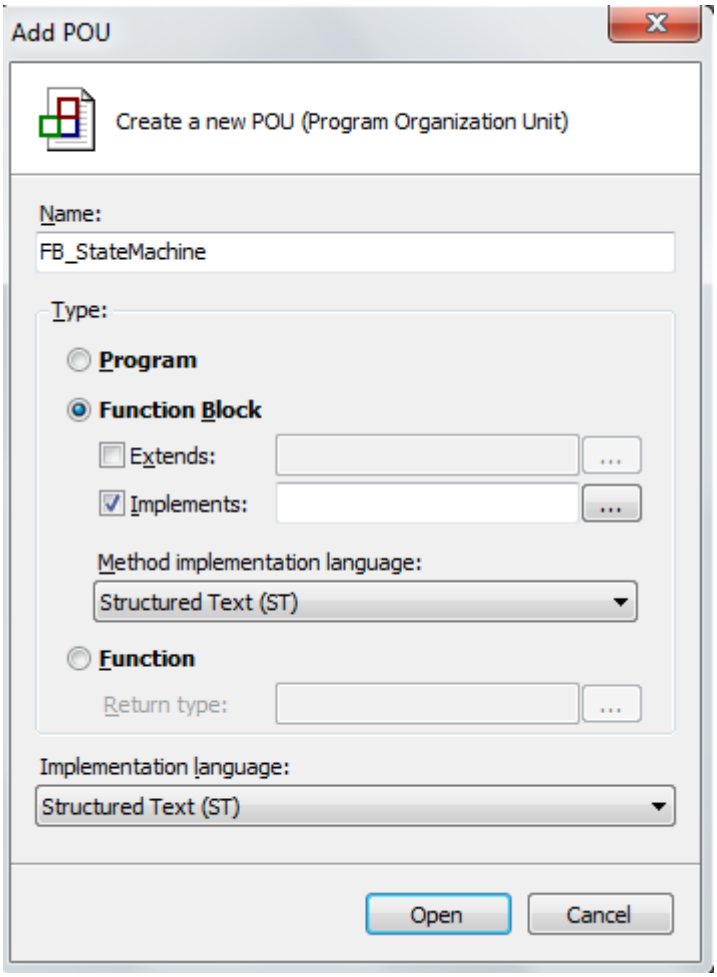
1. 右键单击 **POU**。

2. 选择**添加 -> POU... ...**

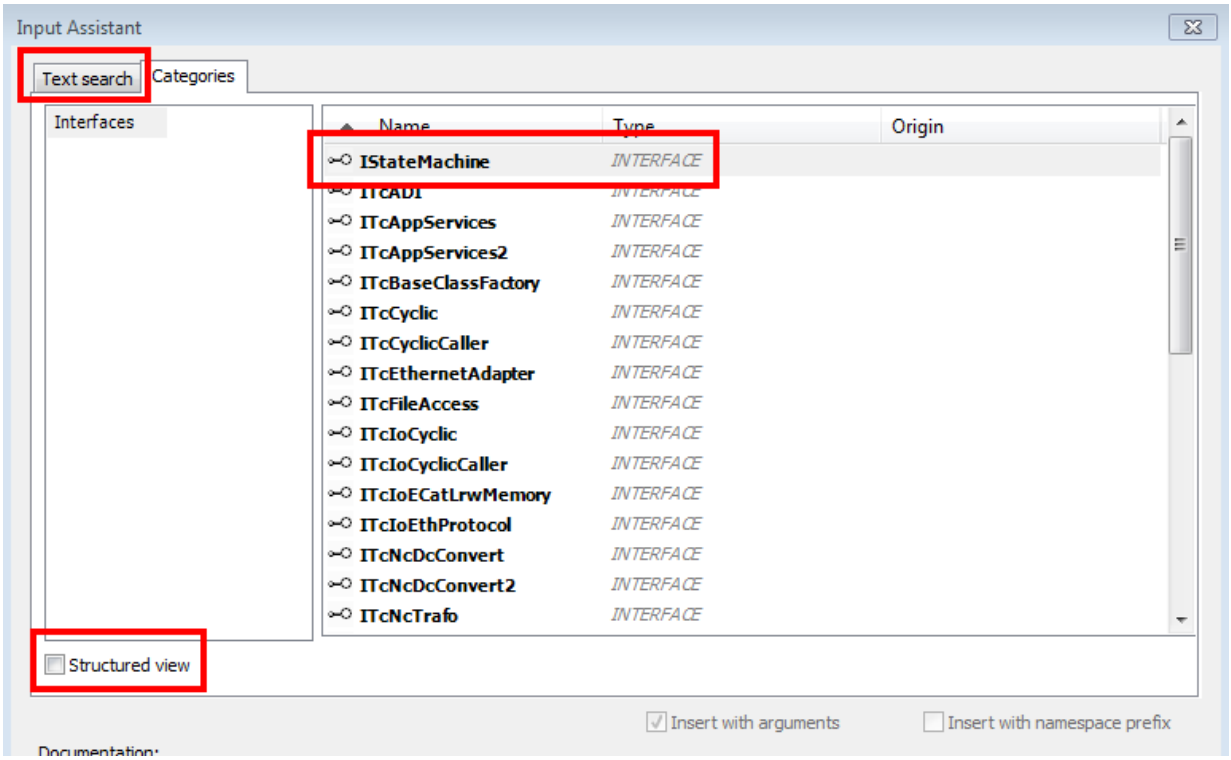


3. 定义要创建的新 FB，该 FB 稍后将用作调用 C++ 类的代理：输入新 FB 的名称：“FB_StateMachine”。

4. 选择**功能块**，再选择**执行**，然后点击 **...** 按钮。



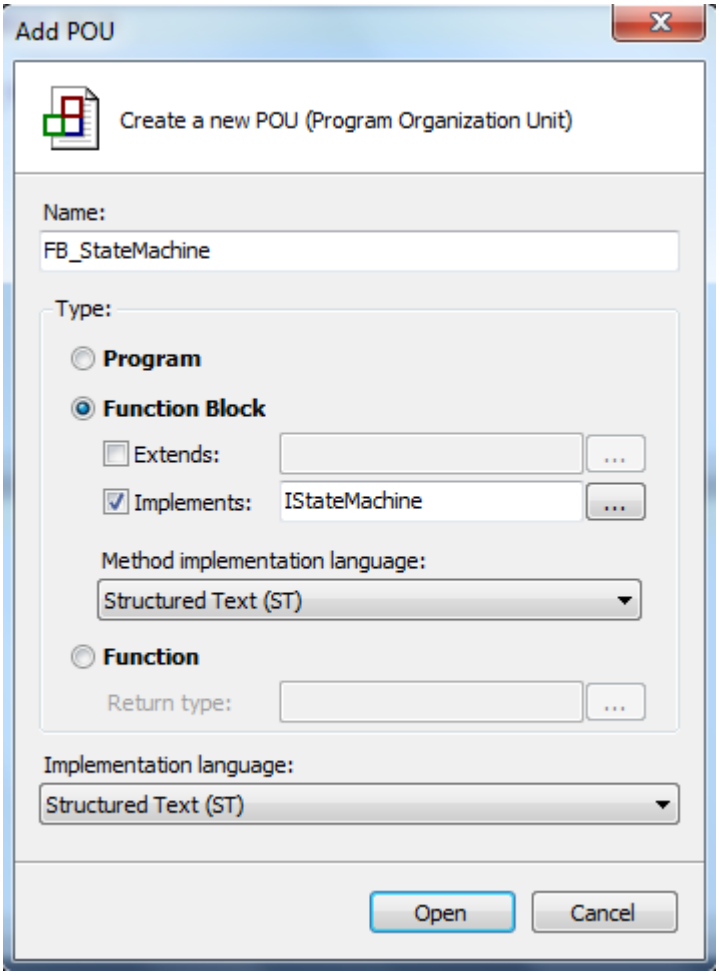
5. 通过**文本搜索**选项卡或**类别**选项卡选择接口，取消选择**结构化视图**。



6. 选择 **IStateMachine**，并点击**确定**。

   ⇨ 然后，IStateMachine 接口被列为要实现的接口。

7. 选择**结构化文本 (ST)** 作为**方法实现语言**。
8. 选择**结构化文本 (ST)** 作为实现语言。
9. 按**打开**结束该对话框。



⇨ 您已成功添加 FB。

第 4 步：调整功能块接口

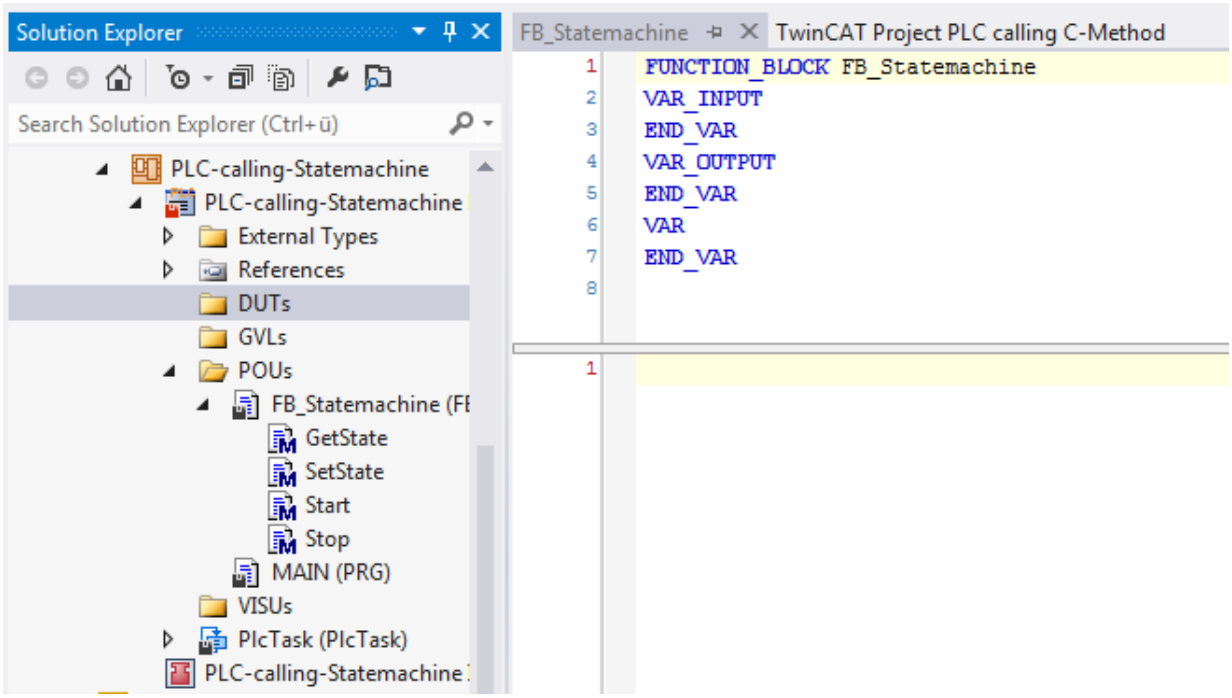
创建可实现 IStateMachine 接口的 FB 后，向导将创建具有相应方法的 FB。

FB_StateMachine 提供 4 种方法：

- GetState
- SetState
- 开始
- 停止

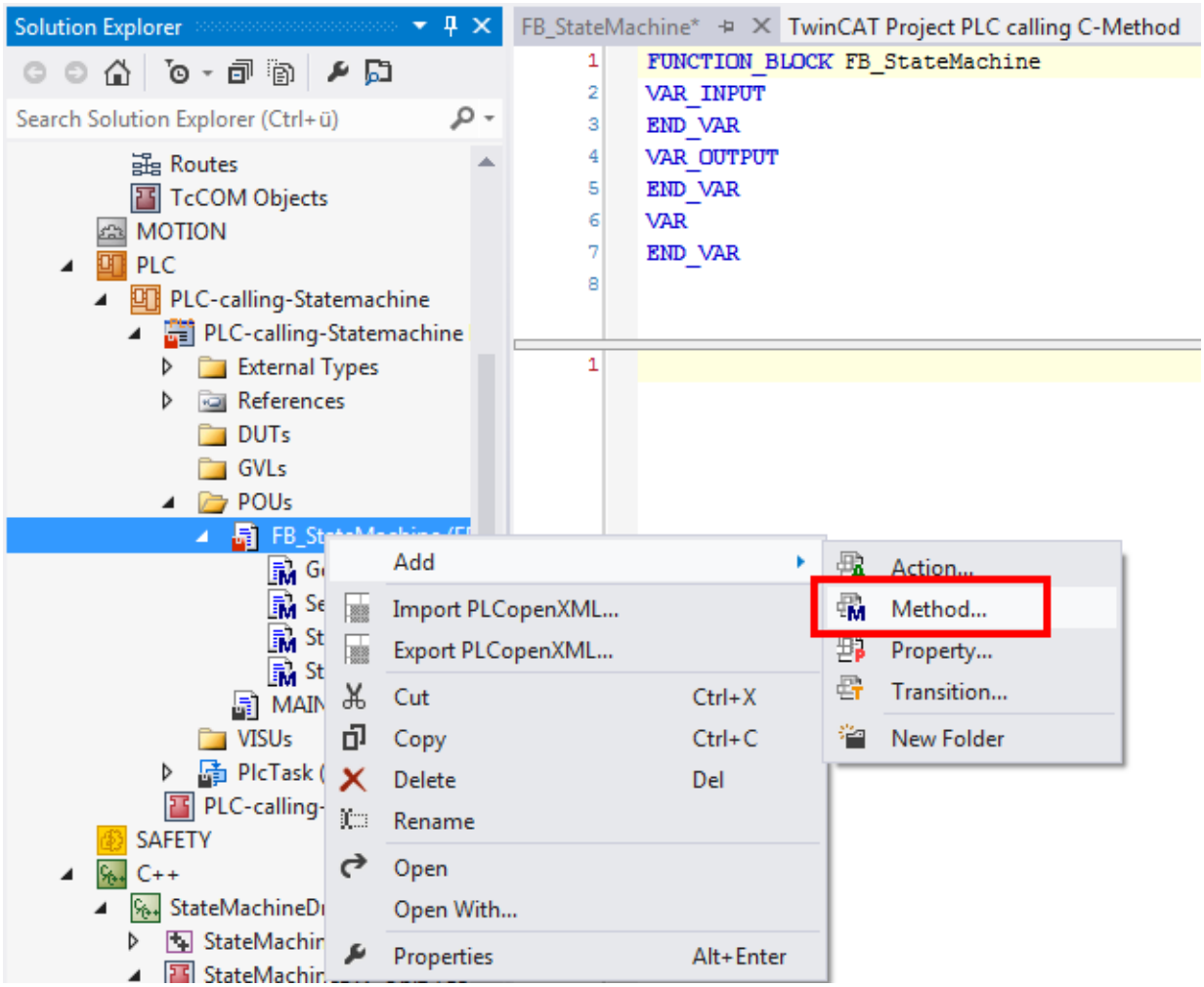
1. “删除”可实现 IStateMachine。由于功能块应充当代理，因此它本身并不会实现接口。因此，可以将其删除。
2. 删除 TcAddRef、TcQueryInterface 和 TcRelease 方法。代理功能块不需要这些方法。

⇨ 结果为：

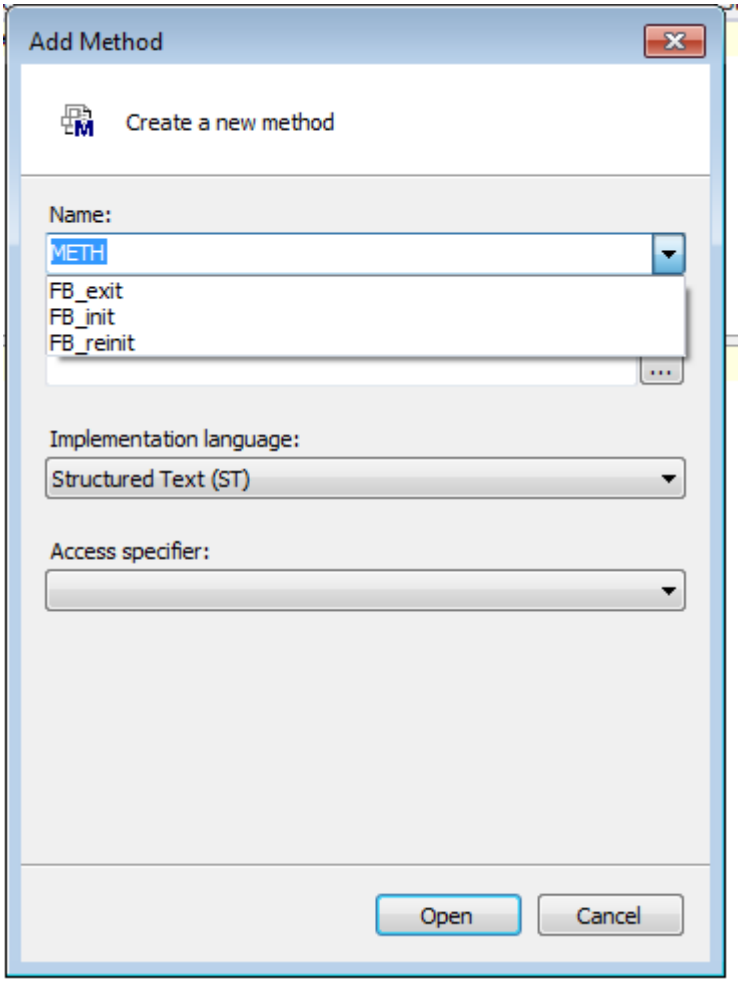


第 5 步：添加 FB 方法 FB_init（构造函数）和 FB_exit（析构函数）

1. 右键单击树中的 **FB_StateMachine**，并选择**添加/方法.....**

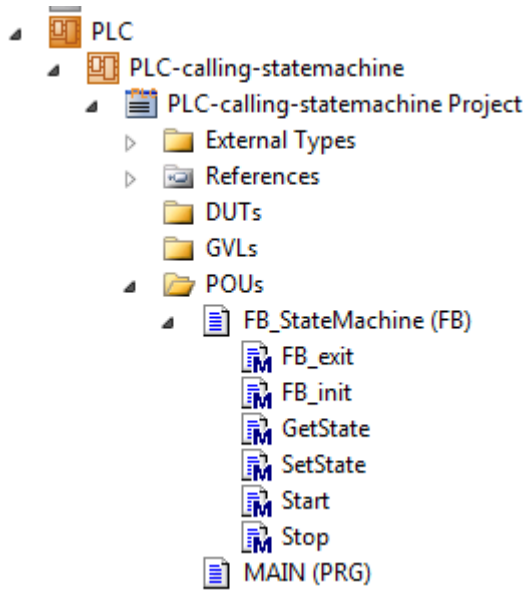


2. 添加 FB_exit 和 FB_init 方法，这两种方法均使用结构化文本 (ST) 作为实现语言。它们可以作为预定义名称使用。



3. 点击**打开**，退出对话框。

⇨ 最后，所有必需方法都可以使用：



第 6 步：实现 FB 方法

现在，所有方法都必须写入代码。

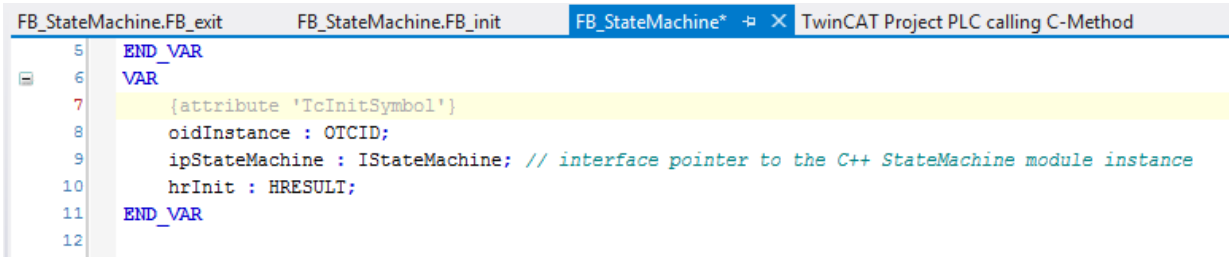
注意

属性缺失导致意外行为

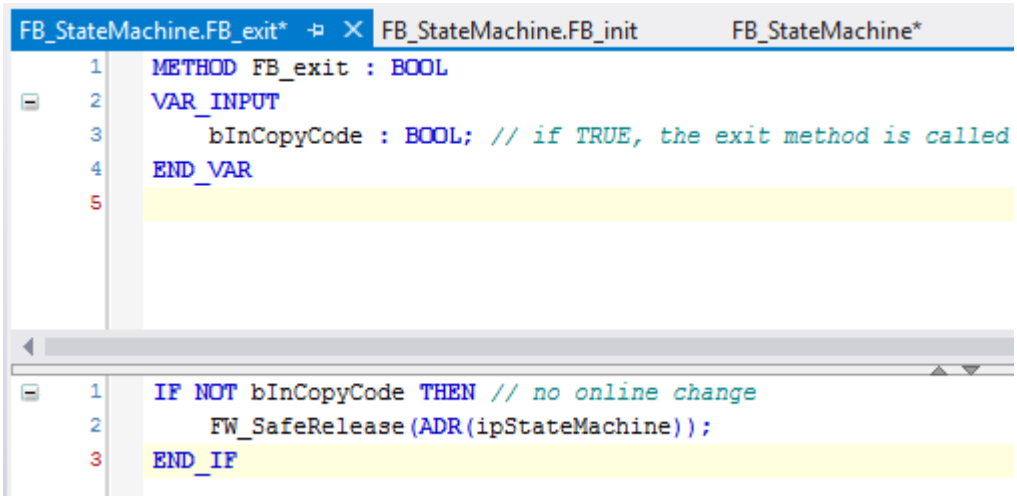
括号中的属性语句表示要添加的代码。

有关属性的更精确信息，请参阅 PLC 文档。

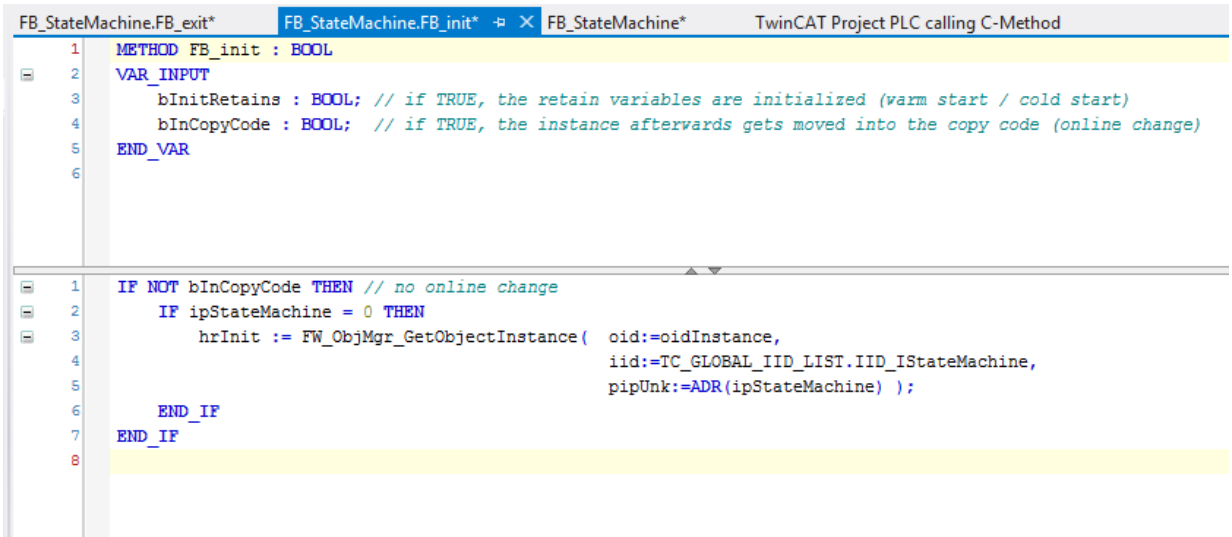
1. 实现 FB_Statemachine 的变量声明。FB 本身不需要循环执行代码。



2. 实现变量声明和方法 FB_exit 的代码区。



3. 实现变量声明和方法 FB_init 的代码区。



4. 实现变量声明和 GetState 方法的代码区（生成的编译指示可以删除，因为代理 FB 不需要它们）。

FB_StateMachine.GetState* | FB_StateMachine.FB_exit* | FB_StateMachine.FB_init*

```
METHOD GetState : HRESULT
VAR_INPUT
    pState  : POINTER TO INT;
END_VAR
```

```
IF (ipStateMachine <> 0) THEN
    GetState:= ipStateMachine.GetState(pState);
END_IF
```

5. 实现变量声明和 SetState 方法的代码区（生成的编译指示可以删除，因为代理 FB 不需要它们）。

FB_StateMachine.SetState* | FB_StateMachine.GetState* | FB_StateMachine.FB_exit*

```
METHOD SetState : HRESULT
VAR_INPUT
    State   : INT;
END_VAR
```

```
IF (ipStateMachine <> 0) THEN
    SetState:= ipStateMachine.SetState(State);
END_IF
```

6. 实现变量声明和“Start”方法的代码区
（对于代理功能块（FB）而言，生成的编译指令（pragmas）可予以删除）。

FB_StateMachine.Start | FB_StateMachine.SetState* | FB_StateMachine.GetState*

```
METHOD Start : HRESULT
```

```
IF (ipStateMachine <> 0) THEN
    Start:= ipStateMachine.Start();
END_IF
```

7. 实现变量声明和“Stop”方法的代码区
（生成的编译指示可以删除，因为代理 FB 不需要它们）。

FB_StateMachine.Stop | FB_StateMachine.Start | FB_StateMachine.SetState*

```
METHOD Stop : HRESULT
```

```
IF (ipStateMachine <> 0) THEN
    Stop:= ipStateMachine.Stop();
END_IF
```

⇨ 作为调用 C++ 模块实例代理的 FB_StateMachine 的实现已经完成。

第 7 步：在 PLC 中调用 FB

现在，FB_StateMachine 在 POU MAIN 中调用。

该简单示例作用如下：

- PLC 计数器循环递增 nCounter
- 如果 nCounter = 500，那么 C++ StateMachine 会以状态“1”启动，以递增其内部 C++ 计数器。然后使用 GetState() 读取 C++ 的状态。
- 如果 nCounter = 1000，那么 C++ 状态机会设置为状态“2”，以递减其内部的 C++ 计数器。然后使用 GetState() 读取 C++ 的状态。
- 如果 nCounter = 1500，那么 C++ StateMachine 停止运行。PLC nCounter 也设置为“0”，这样所有操作便从头开始。

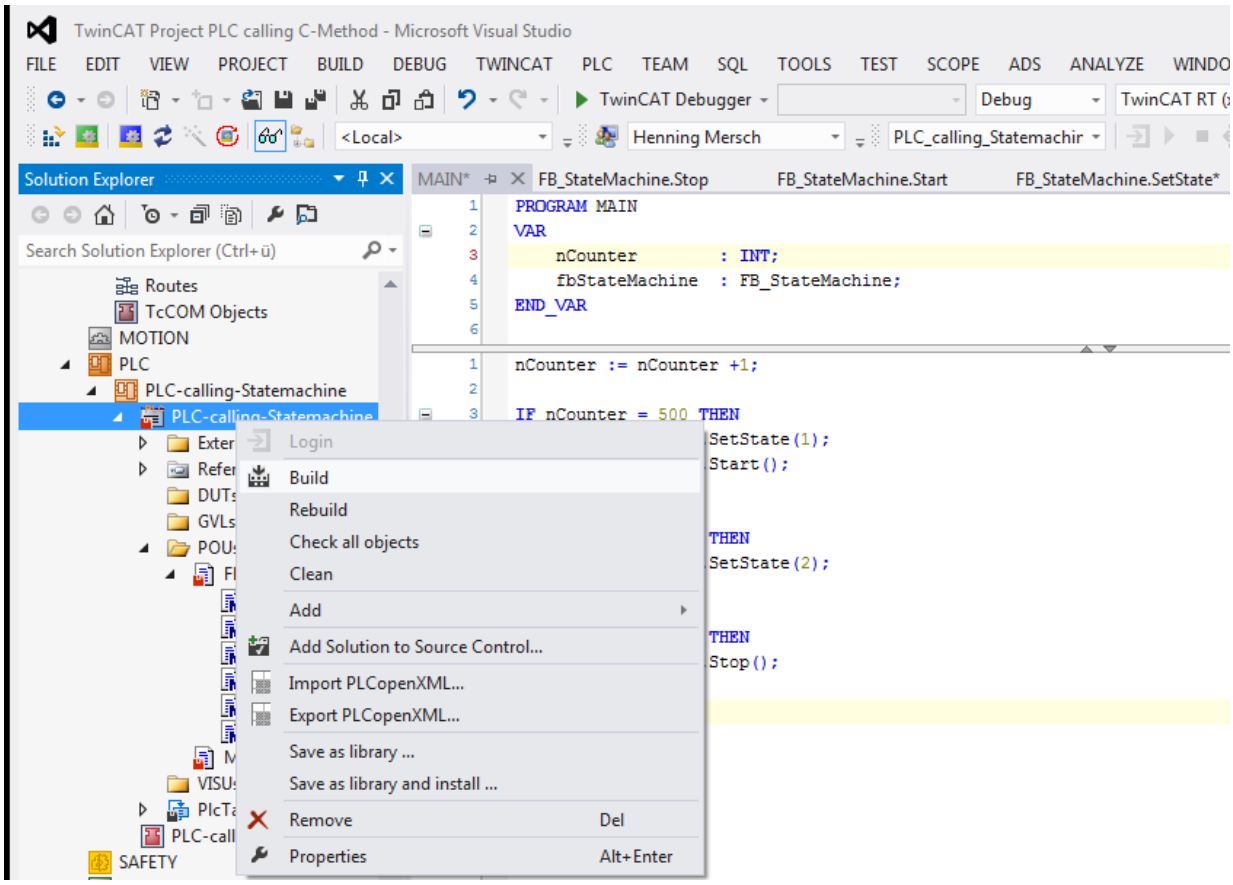
MAIN | TwinCAT Project PLC calling C-Method | FB_StateMachine.GetState

```
1  PROGRAM MAIN
2  VAR
3      nCounter         : INT;
4      nCurrentState    : INT;
5      fbStateMachine   : FB_StateMachine;
6  END_VAR
7
```

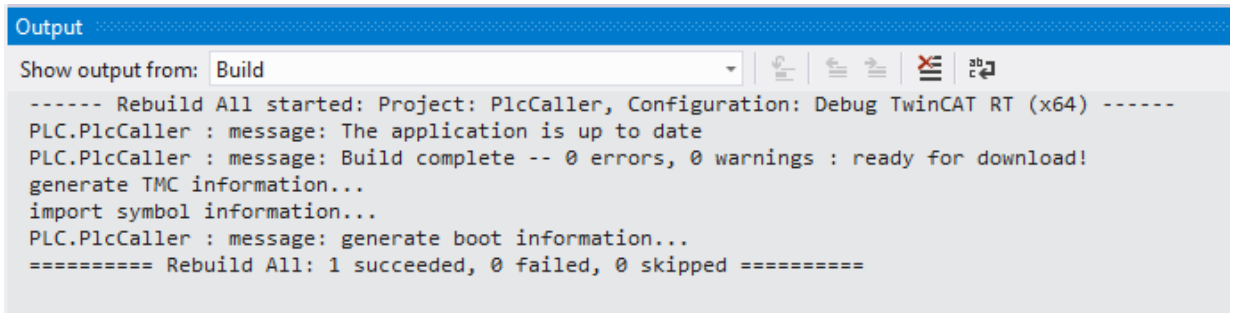
```
1  nCounter := nCounter +1;
2
3  IF nCounter = 500 THEN
4      fbStateMachine.SetState(1);
5      fbStateMachine.GetState(ADR(nCurrentState));
6      fbStateMachine.Start();
7  END_IF
8
9  IF nCounter = 1000 THEN
10     fbStateMachine.SetState(2);
11     fbStateMachine.GetState(ADR(nCurrentState));
12 END_IF
13
14 IF nCounter = 1500 THEN
15     fbStateMachine.Stop();
16     nCounter := 0;
17 END_IF
```

第 8 步：编译 PLC 代码

1. 右键单击 PLC 项目，然后点击**构建**。



⇨ 编译结果显示“1 成功 – 0 失败”。



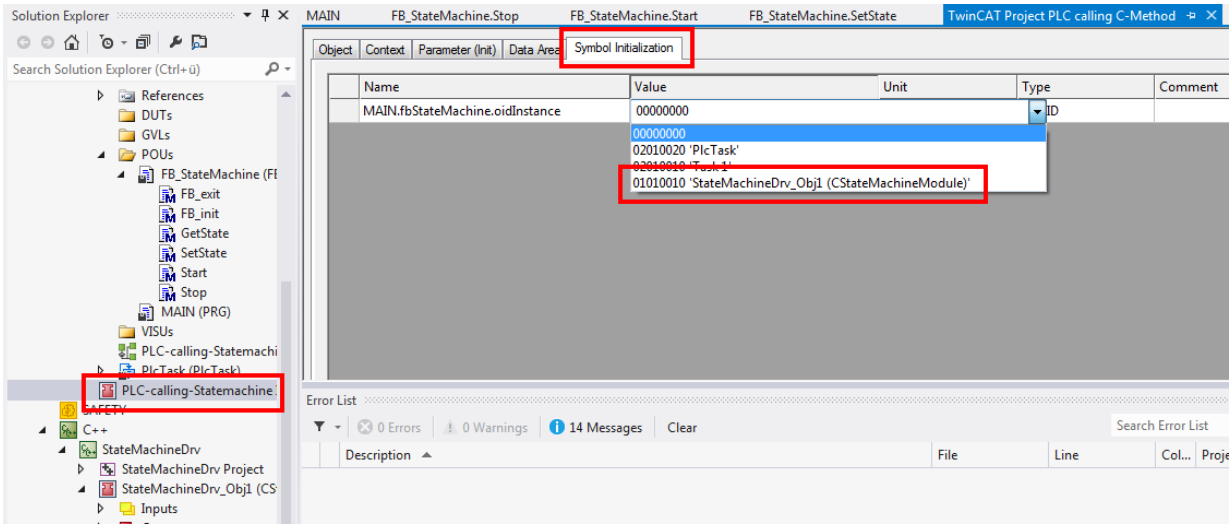
第 9 步：将 PLC FB 与 C++ 实例相关联

前面所有步骤的成效现在显而易见：

PLC FB FB_StateMachine 可配置为与 C++ 模块 StateMachine 的每个实例相关联。这是一种非常灵活且功能强大的方法，可将机器上的 PLC 模块和 C++ 模块相互连接。

1. 导航至左侧树中的 PLC 模块实例，然后选择右侧**Symbol Initialization**选项卡。
   ⇨ FB_StateMachine 的所有实例均已列出；在本示例中，我们仅在 POU MAIN 中定义一个 FB 实例。

2. 选择下拉字段**Value**，然后选择要关联到 FB 实例的 C++ 模块实例。

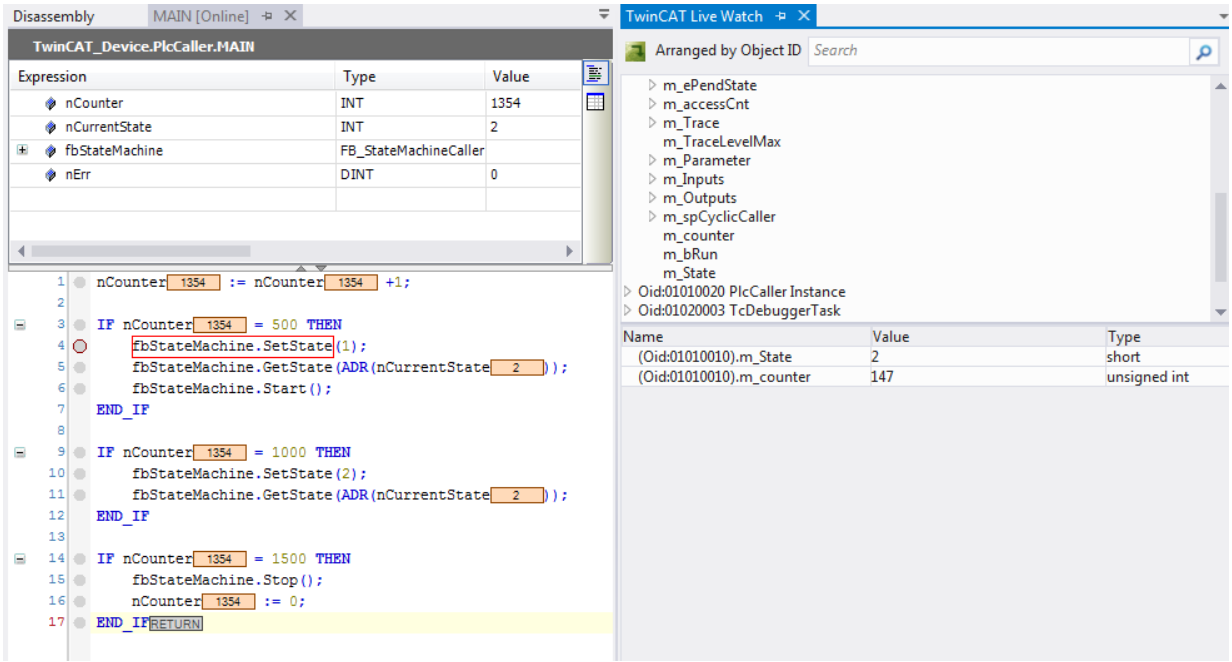


⇨ PLC 模块和 C++ 模块相互连接。

第 10 步：观察 PLC 和 C++ 两个模块的执行情况

在激活 TwinCAT 配置和下载并启动 PLC 代码之后，PLC 和 C++ 这两个代码的执行情况便很容易观察到：

1. 登录并启动 PLC 项目后，编辑器已处于在线模式（左侧 – 见下图）。
2. 为了能够访问 C++ 模块的在线变量，请激活 C++ 调试 [▶ 69]并按照快速入门 [▶ 57]中的步骤进行操作，以便开始调试（如下图右侧所示）。



15.1 Sample11a：模块通信：C++ 模块调用 C++ 模块的方法 0

本文介绍 TwinCAT 3 C++ 模块如何通过方法调用进行通信。该方法通过临界区保护数据，因此可以从不同的环境/任务中发起访问。

下载

您可以在此访问该示例的源代码：

https://github.com/Beckhoff/TC1300_Samples/tree/main/S11a-Mod2ModCS

1. 利用已安装 TwinCAT 的 Visual Studio，通过**打开项目.....**来打开项目
2. 通过右键单击**项目 -> 属性 -> Tc 签名**打开 TwinCAT 签名，为该项目配置签名，必要时配置证书和密码。有关签署 C++ 项目的更多信息，请参见 TwinCAT [▶ 22] 章节。
3. 选择目标系统。
4. 构建示例（例如**构建 -> 构建解决方案**）。
5. 点击  激活配置。

⇨ 示例已做好操作准备。

功能说明

该项目包含三个模块：

- CModuleDataProvider 类的实例可托管数据，并通过临界区防止通过“检索”和“存储”方法访问数据。
- 模块类 CModuleDataRead 的实例通过调用“检索”方法读取 DataProvider 的数据。
- 模块类 CModuleDataWrite 的实例通过调用“存储”方法写入 DataProvider 的数据。

读取/写入实例是为访问 DataProvider 实例而配置的，可在实例配置的**接口指针**菜单中查看。
还可在此配置执行实例的环境（任务）。在本示例中，使用两个任务：TaskRead 和 TaskWrite。
CModuleDataWrite 和 DataReadCounterModulo (CModuleDataRead) 的 DataWriteCounterModul 参数可确定模块实例发起访问的时刻。

在 SDK 中，CriticalSections 在 TcRtInterfaces.h 中有所功能说明，因此针对的是实时环境。

15.1 Sample12：模块通信：使用 IO 映射 1

本文介绍了两个 TwinCAT 3 C++ 模块如何通过 TwinCAT 3 的常用 IO 映射进行通信：两个实例通过 IO 映射相关联，并定期访问变量值。

下载

您可以在此访问该示例的源代码：

https://github.com/Beckhoff/TC1300_Samples/tree/main/S12-Mod2ModIOMapping

1. 利用已安装 TwinCAT 的 Visual Studio，通过**打开项目.....**来打开项目
2. 通过右键单击**项目 -> 属性 -> Tc 签名**打开 TwinCAT 签名，为该项目配置签名，必要时配置证书和密码。有关签署 C++ 项目的更多信息，请参见 TwinCAT [▶ 22] 章节。
3. 选择目标系统。
4. 构建示例（例如**构建 -> 构建解决方案**）。
5. 点击  激活配置。

⇨ 示例已做好操作准备。

功能说明

这两个实例都是通过模块类 ModuleInToOut 实现的：该类会将其输入数据区的Value循环复制到输出数据区的Value中
“前部”实例作为用户的前端。输入Value通过 cycleupdate() 方法传送到输出Value。该“Front”输出值被分配（关联）至“Back”实例的输入Value。
“Back”实例将输入Value复制到其输出Value，用户可以对输出Value进行监控（请参见以下快速入门步骤以开始调试：调试 TwinCAT 3 C++ 项目）

最终，用户可以定义“Front”实例的输入值，并观察 “Back”的输出值。

15.1 Sample13：模块通信：C++ 模块到 PLC 模块的方法调用 2

本文介绍了 TwinCAT C++ 模块如何通过 TcCOM 接口调用 PLC 功能块的方法。

下载

您可在此处获取该示例的源代码： https://github.com/Beckhoff/TC1300_Samples/tree/main/S13-CppToPLC

1. 利用已安装 TwinCAT 的 Visual Studio，通过**打开项目.....**来打开项目
2. 通过右键单击**项目 -> 属性 -> Tc 签名**打开 TwinCAT 签名，为该项目配置签名，必要时配置证书和密码。有关签署 C++ 项目的更多信息，请参见 TwinCAT [▶ 22] 章节。
3. 选择目标系统。
4. 构建示例（例如**构建 -> 构建解决方案**）。
5. 点击  激活配置。

⇨ 示例已做好操作准备。

功能说明

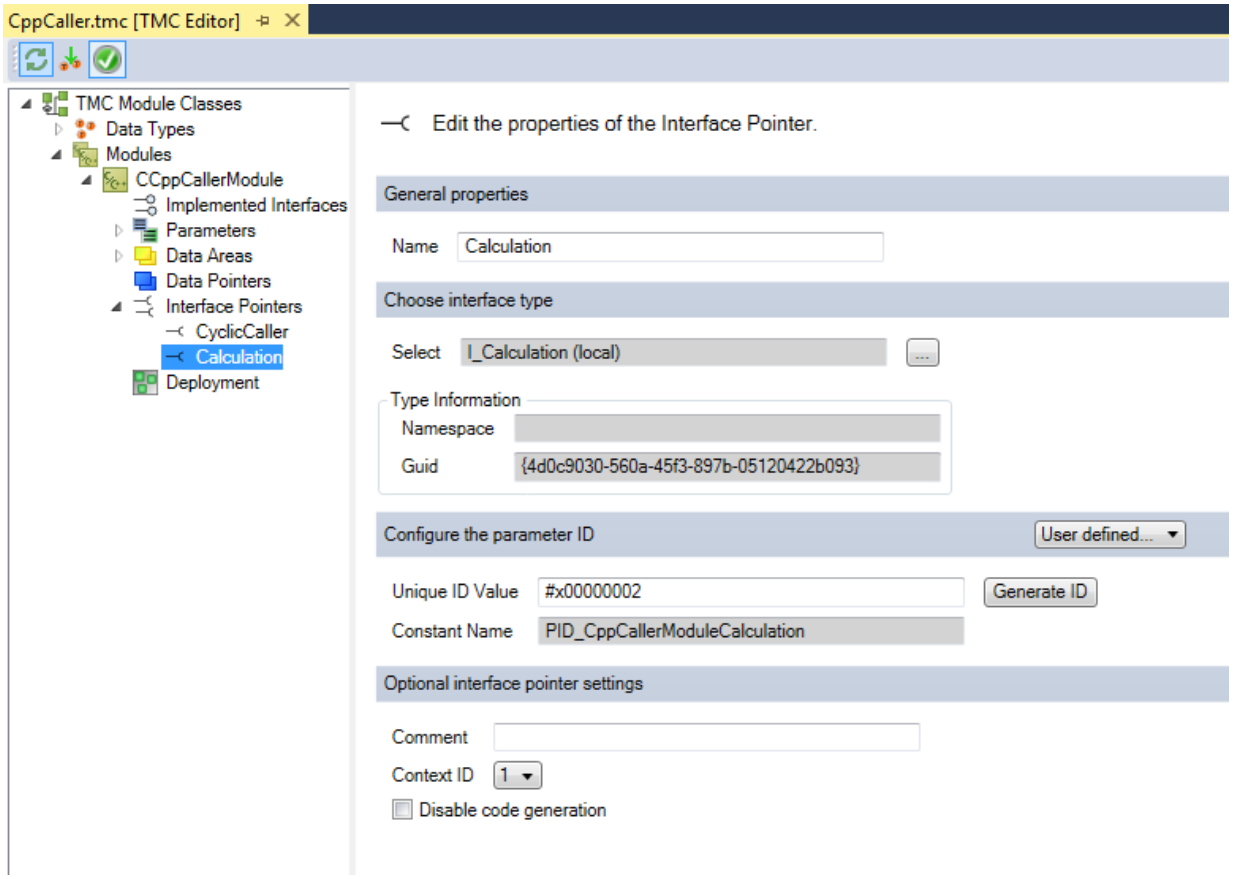
该示例通过方法调用实现了从 C++ 模块到 PLC 功能块的通信。为此，定义了一个由 PLC 提供并由 C++ 模块使用的 TcCOM 接口。

流程中作为提供者的 PLC 页面与 TcCOM Sample 01 [▶ 302] 的相应项目相对应，在该项目中考虑了PLC通信后的PLC情况。现在，C++ 中提供了一个调用者，它使用相同的接口。

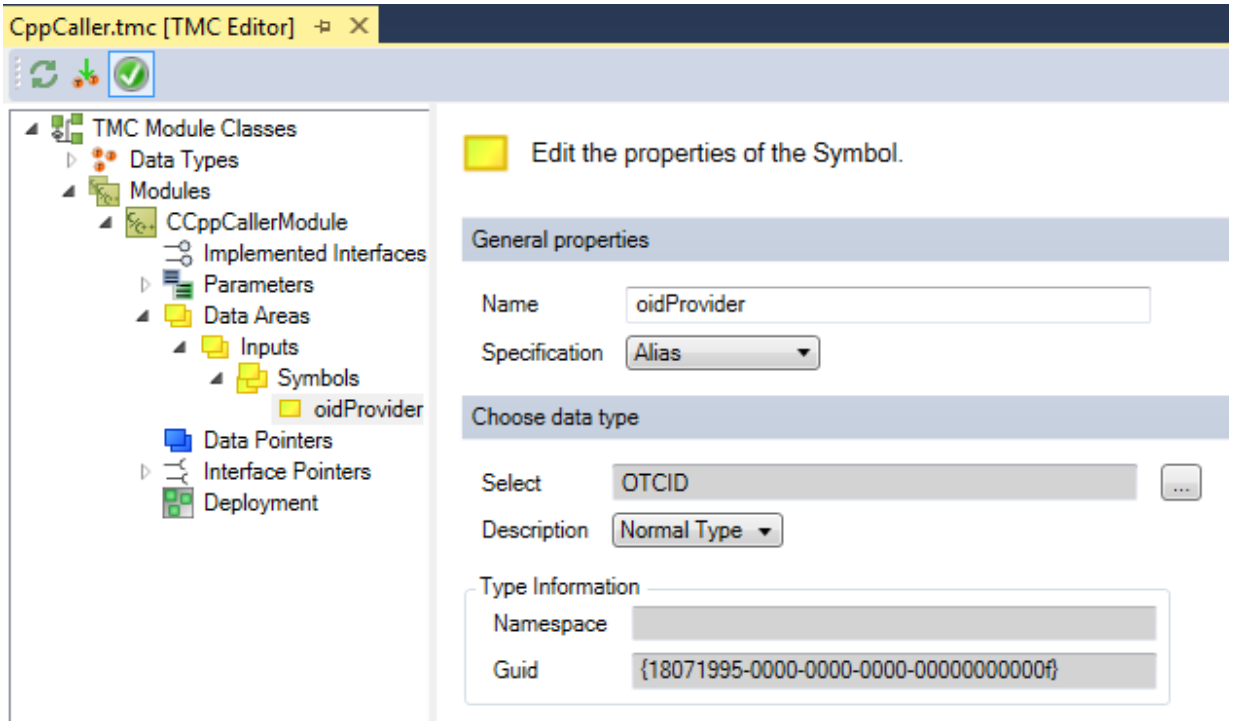
TcCOM Sample 01 [▶ 302] 采用 PLC 页面。作为 TcCOM 模块注册的功能块可将所分配的对象 ID 作为输出变量。
C++ 模块的任务是使该功能块提供的接口可供访问。

✓ 假设一个 C++ 项目有一个循环 IO 模块。

1. 在 TMC 编辑器中，创建一个 I_Calculation 类型的接口指针，名称为 Calculationn)。随后的访问也是通过该指针进行的。

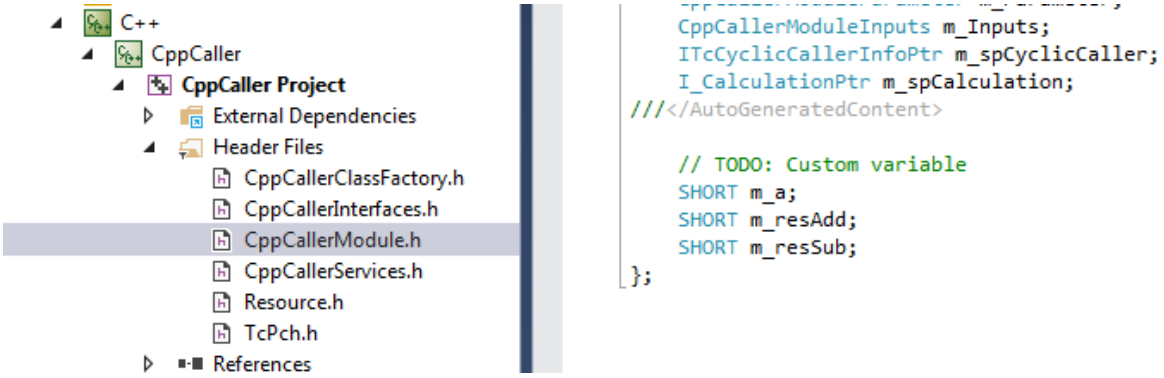


2. 模块向导已创建输入-目的类型的数据区输入。在 TMC 编辑器中创建一个 OTCID 类型的输入，名称为 oidProvider，稍后将通过该输入从 PLC 关联对象 ID。

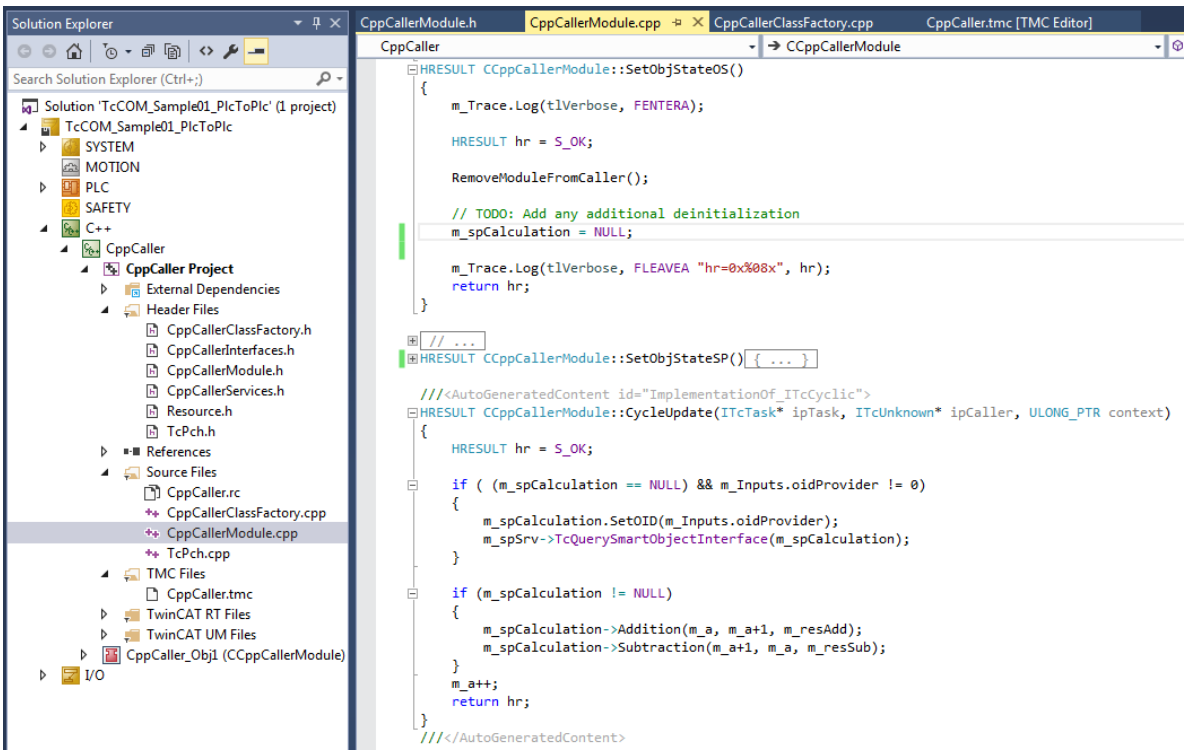


3. 所有其他符号均与示例无关，可以删除。

⇨ TMC-Code-Generator 据此生成代码。
在模块的标题中创建一些变量，以便稍后执行方法调用。



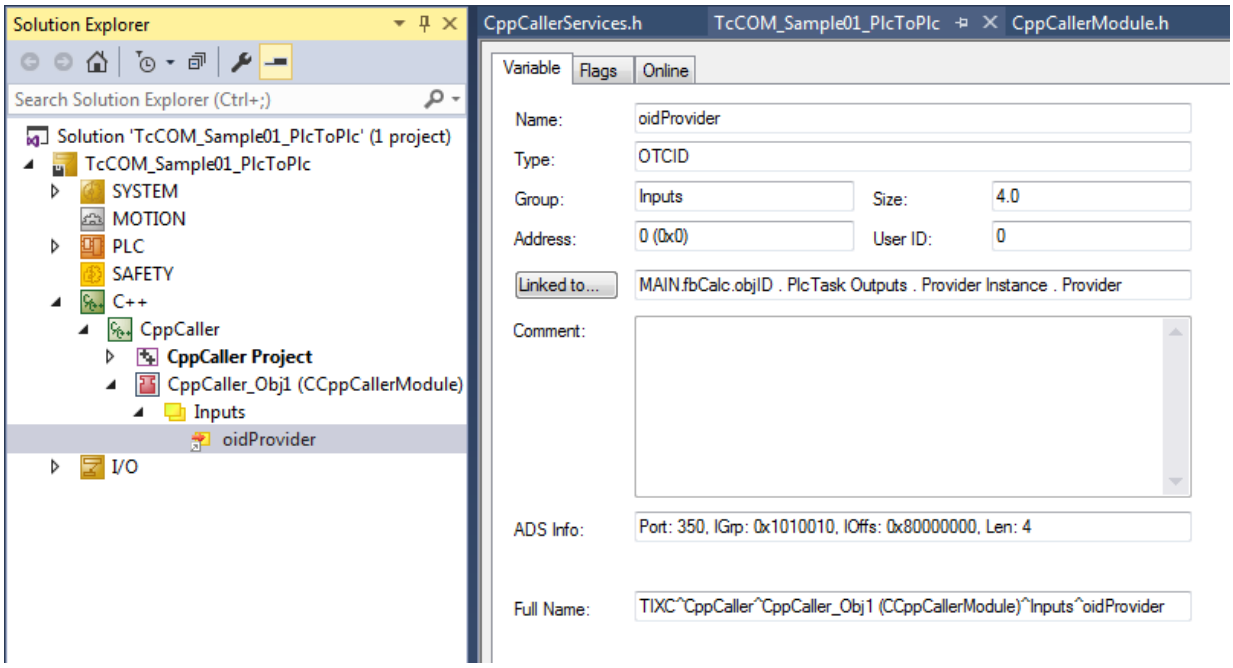
在 CycleUpdate() 模块的实际代码中，使用从 PLC 传输的对象 ID 设置接口指针。重要的是，由于 PLC 必须先提供功能块，因此必须在 CycleUpdate() 中执行操作，也就是在实时实时环境进行。完成一次操作后，便可调用这些方法。



此外，如上所述，当程序关闭时，接口指针也会被清除。SetObjStateOS 方法中会出现此情况。

4. 现在构建 C++ 项目。
5. 创建模块实例。

6. 将 C++ 模块的输入与 PLC 的输出连接。



⇨ 项目可以启动。当 PLC 运行时，对象标识符（OID）会通过映射传递给 C++ 实例。一旦出现这种情况，便可调用该方法。

15.1 Sample19：同步文件访问3

本文介绍了如何实现 TwinCAT 3 C++ 模块，该模块会在模块启动时访问硬盘上的文件，因此是在实时环境中执行的。

下载

您可在此处获取该示例的源代码： https://github.com/Beckhoff/TC1300_Samples/tree/main/S19-FileIOSync

1. 利用已安装 TwinCAT 的 Visual Studio，通过**打开项目.....**来打开项目
2. 通过右键单击**项目 -> 属性 -> Tc 签名**打开 TwinCAT 签名，为该项目配置签名，必要时配置证书和密码。有关签署 C++ 项目的更多信息，请参见 TwinCAT [▶ 22] 章节。
3. 选择目标系统。
4. 构建示例（例如**构建 -> 构建解决方案**）。
5. 点击  激活配置。

⇨ 示例已做好操作准备。

所有并非由向导自动生成的源代码都会标注注释开始标志“//示例代码”和注释结束标志“//示例代码结束”。
这样便可在文件中搜索这些字符串，以了解详情。

描述

本示例介绍如何通过 TwinCAT 接口 ITCFileAccess 访问文件。这种访问是同步进行的，可用于在模块启动时读取配置等。

该示例包含一个 C++ 模块 TcFileTestDrv，该模块的实例为 TcFileTestDrv_Obj1。
在该示例中，文件访问发生在从 PREOP 到 SAFEOP 的转换期间，即 SetObjStatePS() 方法中。

辅助方法简述了文件处理过程。
首先，一般文件信息和目录列表均会打印到 TwinCAT 3 的日志记录设备上。然后，会将文件 %TC_TARGETPATH%DefaultConfig.xml（通常为 C:\TwinCAT\3.1\Target\DefaultConfig.xml）复制到 %TC_TARGETPATH%DefaultConfig.xml.bak 中。

要访问日志记录条目，请参见 TwinCAT 3 输出窗口的**错误列表**选项卡。
可以通过更改**参数 (Init)**选项卡中 TcFileTestDrv_obj1 实例的 TraceLevelMax 变量来设置信息量。

15.1 Sample20：FileIO-Write 4

本文介绍 TwinCAT 3 C++ 模块的实现，这些模块会将（过程）值写入文件。

文件的写入由确定性循环触发，文件 IO 的执行已解耦（异步），即：确定性循环继续运行，不会受到写入文件的阻碍。

下载

您可在此处获取该示例的源代码： https://github.com/Beckhoff/TC1300_Samples/tree/main/S20-FileIOWrite

1. 利用已安装 TwinCAT 的 Visual Studio，通过**打开项目.....**来打开项目
2. 通过右键单击**项目 -> 属性 -> Tc 签名**打开 TwinCAT 签名，为该项目配置签名，必要时配置证书和密码。有关签署 C++ 项目的更多信息，请参见 TwinCAT [▶ 22] 章节。
3. 选择目标系统。
4. 构建示例（例如**构建 -> 构建解决方案**）。
5. 点击  激活配置。

⇨ 示例已做好操作准备。

功能说明

该示例包括 TcAsyncWritingModule 的一个实例，它会将数据写入 BOOTPRJPATH 目录 (Windows: C:\TwinCAT\3.1\Boot; TwinCAT/BSD: /usr/local/etc/TwinCAT/3.1/Boot) 中的 AsyncTest.txt 文件。

TcAsyncBufferWritingModule 有两个缓冲区（m_Buffer1、m_Buffer2），交替填充当前数据。成员变量 m_pBufferFill 指向当前要填充的缓冲区。缓冲区填满后，成员变量 m_pBufferWrite 将被设置为指向填满的缓冲区。

这些数据将借助 TcFsmFileWriter 写入文件。

请注意，文件中没有可读内容，如 ASCII 字符；在此示例中，会将二进制数据写入文件。

15.1 Sample20a：FileIO-Cyclic 读取/写入 5

本文是比 S20 和 S19 更全面的示例。其中演示从 TwinCAT 3 C++ 模块对文件的循环读取和/或写入访问。

下载

您可以在此访问该示例的源代码：

https://github.com/Beckhoff/TC1300_Samples/tree/main/S20a-FileIO-CyclicReadWrite

1. 利用已安装 TwinCAT 的 Visual Studio，通过**打开项目.....**来打开项目
2. 通过右键单击**项目 -> 属性 -> Tc 签名**打开 TwinCAT 签名，为该项目配置签名，必要时配置证书和密码。有关签署 C++ 项目的更多信息，请参见 TwinCAT [▶ 22] 章节。

3. 选择目标系统。
4. 构建示例（例如**构建 -> 构建解决方案**）。
5. 点击  激活配置。

⇨ 示例已做好操作准备。

功能说明

本示例介绍了如何使用 CycleUpdate 方法（即循环方式）访问文件进行读取和/或写入。

本示例包含以下项目和模块实例。

- 静态库 (TcAsyncFileIo) 可提供文件访问功能。
  文件访问的代码可以共享，因此该代码位于静态库中，供驱动程序项目使用。
- 一个驱动程序 (TcAsyncBufferReadingDrv) 可提供两个实例：
  - ReadingModule：使用静态库读取 AsyncTest.txt 文件。
  - WriteDetectModule：检测写入操作并启动读取操作。
- 一个驱动程序 (TcAsyncBufferWritingDrv) 可提供一个实例：
  - WriteModule：使用静态库写入 AsyncTest.txt 文件。

启动示例时，写入模块开始将数据写入位于启动项目路径中的文件 (Windows: C:\TwinCAT\3.1\Boot\AsyncTest.txt; TwinCAT/BSD: /usr/local/etc/TwinCAT/3.1/Boot/AsyncTest.txt)。输入变量 bDisableWriting 可用来禁止写入。

这些对象相互连接：写入完成后，WritingModule 会触发 TcAsyncBufferReadingDrv 的 DetectModule。因此，ReadingModule 会启动读取程序。

观察 WritingModule/ReadingModule 的 nBytesWritten/nBytesRead 输出变量。除此以外，协议消息还会以详细级别生成。这些消息依旧可以借助模块的 TraceLevelMax 参数进行配置。

- 一个驱动程序 (TcAsyncFileFindDrv) 可提供一个实例
  - FileFindModule：借助静态库列出目录中的文件。

借助输入变量 bExecute 启动操作。FilePath 参数包含要列出文件的目录 (Windows: c:\TwinCAT\3.1\Boot*; TwinCAT/BSD: /usr/local/etc/TwinCAT/3.1/Boot/*)。

观察与找到的文件列表有关的顺序跟踪（详细协议级别）。

理解示例

项目 TcAsyncFileIO 包含静态库中的各种类。该库用于驱动程序项目的读取和写入。

每个类都用于文件访问操作，如打开/读取/写入/列明/关闭/.....。由于执行是在循环实时环境中进行的，因此每个操作都有一个状态，该类简述了此状态机。

作为了解文件访问的切入点，可以从 TcFsmFileReader 和 TcFsmFileWriter 类开始。

如果历史跟踪消息过多，妨碍对示例的理解，可以禁用模块！

另请参见

示例 S19 [▶ 288]

示例 S20 [▶ 289]

示例 S25 [▶ 295]

接口 ITcFileAccess [▶ 163]/接口 ITcFileAccessAsync [▶ 170]

15.1 Sample22：自动化设备驱动程序 (ADD)：访问 DPRAM 6

本文介绍如何实现 TwinCAT 3 C++ 驱动程序，该驱动程序作为 TwinCAT 自动化设备驱动程序 (ADD) 访问 DPRAM。

下载

您可以在此访问该示例的源代码：

https://github.com/Beckhoff/TC1300_Samples/tree/main/S22-ADD

注意

配置详情

激活前请阅读以下配置详情。

1. 点击**打开项目……**，打开 TwinCAT 3 中包含的该项目的 zip 文件。
2. 选择目标系统。
3. 在本地机器上构建示例（例如，**构建** -> **构建解决方案**）。
4. 请注意本页**配置**下所列操作。
5. 点击  激活配置。

⇨ 示例已做好操作准备。

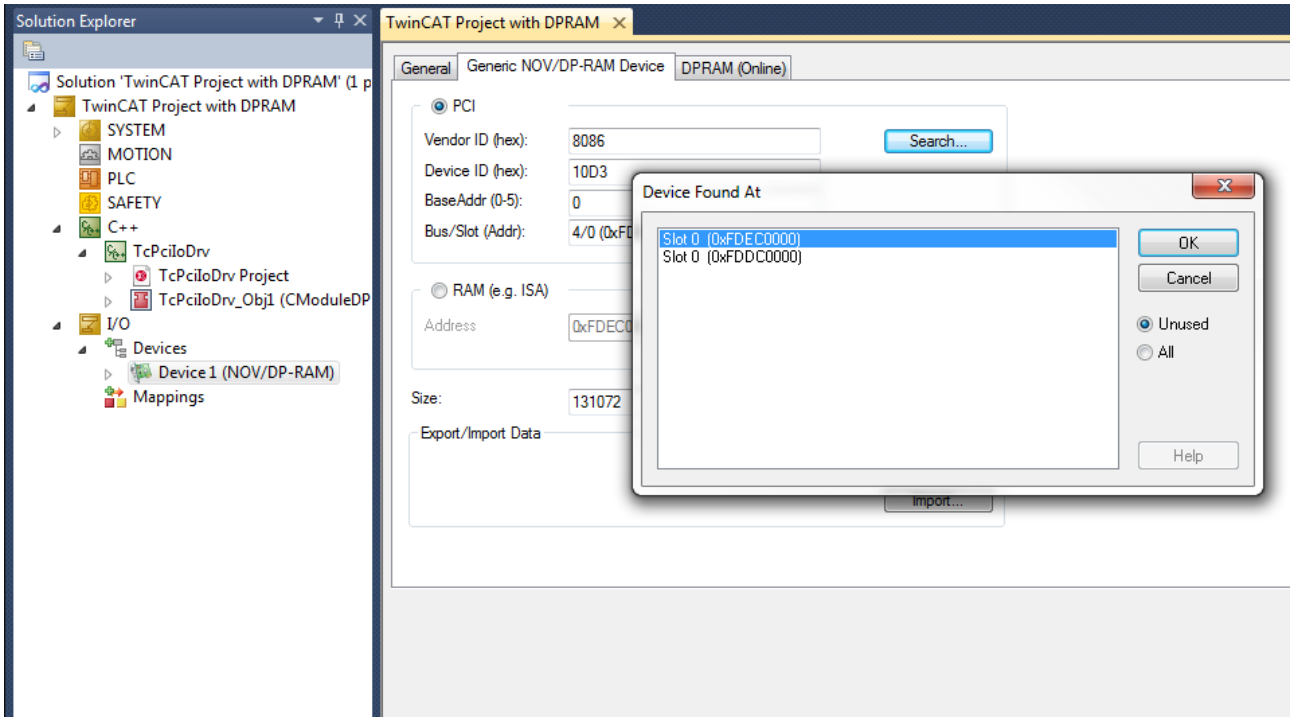
功能说明

本示例用于循环开关网络适配器（即 CX5010）的链路检测位。

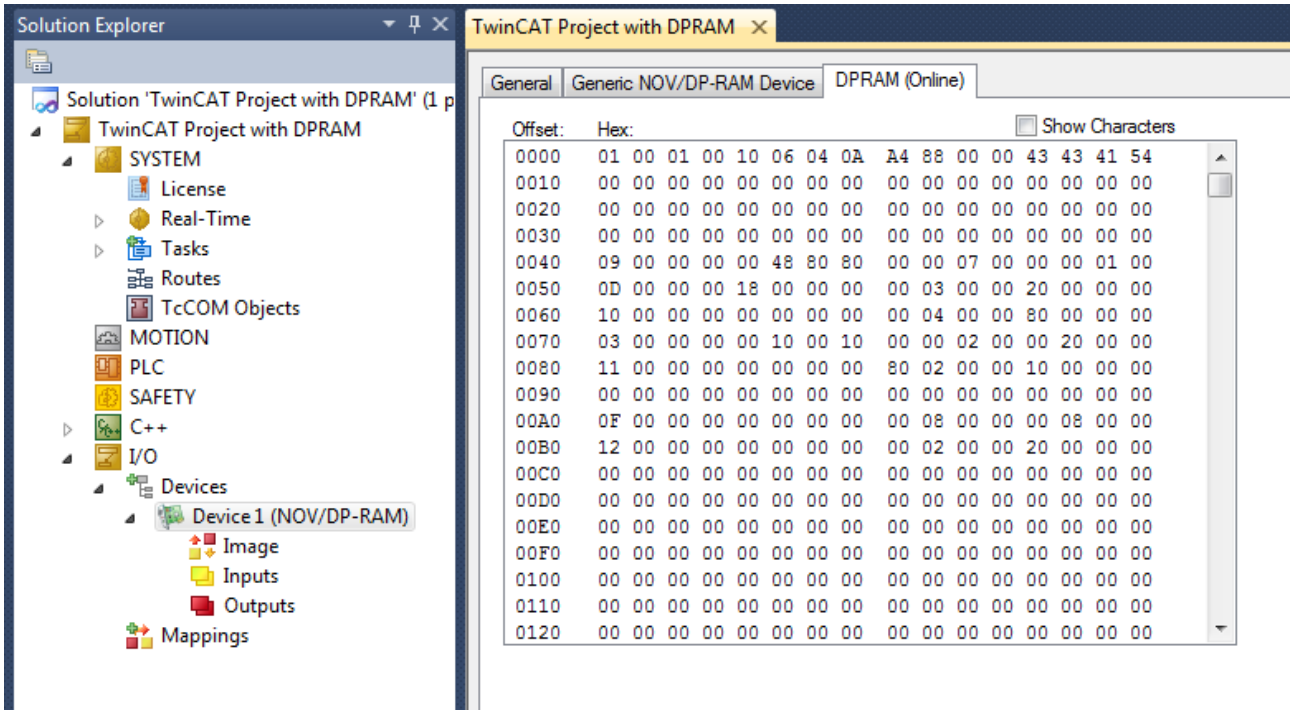
C++ 模块通过 C++ 模块的 PciDeviceAdi 接口指针与 NOV/DP-RAM 设备连接。

配置

为了使示例正常工作，必须将硬件地址配置为与您自己的硬件匹配。
检查 PCI 配置：



要检查与 NOV/DPRAM 的通信设置是否正确，请使用 DPRAM（在线）视图：



15.1 Sample23：结构化异常处理 (SEH)

7

本文在五个变体的基础上介绍了结构化异常处理 (SEH) 的使用情况。

下载

您可以在此访问该示例的源代码：

https://github.com/Beckhoff/TC1300_Samples/tree/main/S23-SEH

1. 利用已安装 TwinCAT 的 Visual Studio，通过**打开项目.....**来打开项目
2. 通过右键单击**项目 -> 属性 -> Tc 签名**打开 TwinCAT 签名，为该项目配置签名，必要时配置证书和密码。有关签署 C++ 项目的更多信息，请参见 TwinCAT [▶ 22] 章节。
3. 选择目标系统。
4. 构建示例（例如**构建 -> 构建解决方案**）。
5. 点击  激活配置。

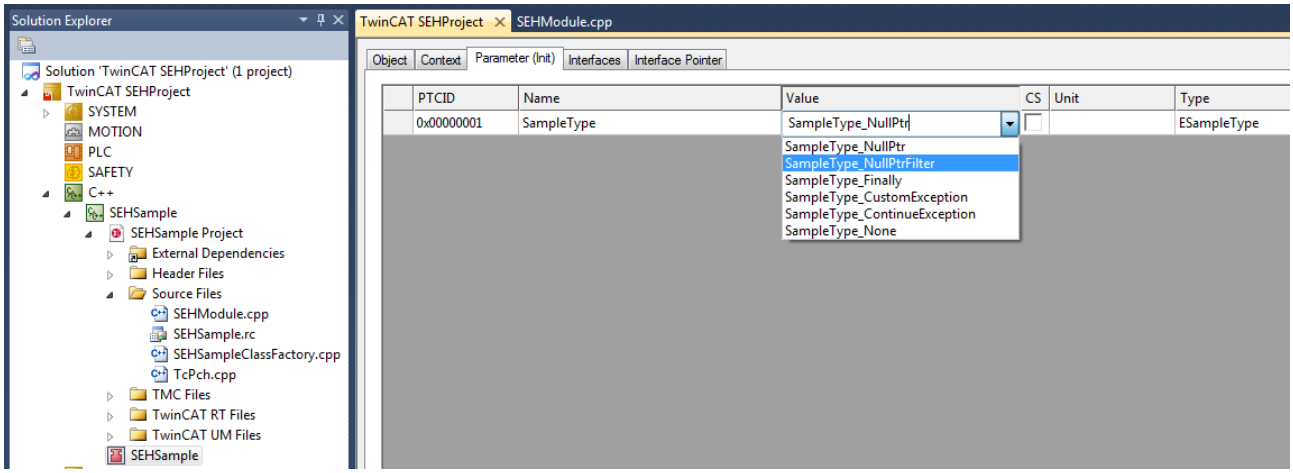
⇨ 示例已做好操作准备。

功能说明

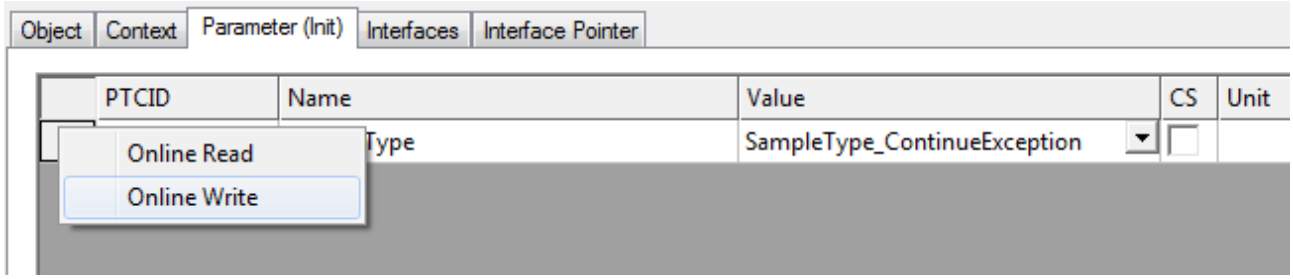
该示例包含五个变体，演示了 SEH 在 TwinCAT C++ 中的使用情况：

1. 访问空指针时出现异常
2. 在带有过滤器的空指针访问情况下的异常处理
3. 带有“Finally”的异常
4. 针对客户的结构化异常
5. “Continue”块出现异常

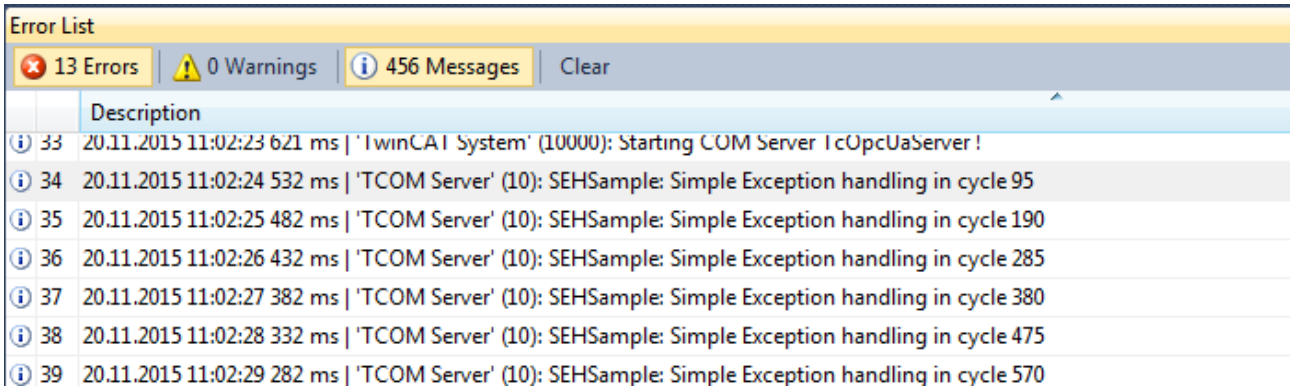
所有这些变体都可以通过 C++ 实例的下拉框进行选择：



选择变体后，您还可以右键单击第一列，在运行时处写入值：



所有变体都会写入跟踪消息来解释行为，因此消息会出现在 TwinCAT Engineering 中：



示例说明

下拉框中的选择操作是枚举，在模块的 CycleUpdate() 中用于选择案例（即 switch case 语句中的情况）。因此，在此可以独立考虑这些变体：

1. 访问空指针时出现异常
   在此，将 PBYTE 创建为 NULL 并在之后使用，进而导致异常。
   TcTry{} 模块会拦截这一过程并生成输出。
2. 带筛选器的空指针访问出现异常
   该变体也可访问空指针，但在 TcExcept{} 中，其使用的是方法 FilterException()，这一方法也在模块中有所定义。对方法中的不同异常做出反应；在这种情况下，仅输出一条消息。
3. 带有“Finally”的异常
   在此再次进行空指针访问，但这次每种情况都要执行 TcFinally{} 块。
4. 针对客户的结构化异常
   通过 TcRaiseException() 生成异常，由 FilterException() 方法拦截和处理。由于此为模块中定义的异常，因此 FilterException() 方法会另外输出一条（具体的）消息。
5. “继续”块出现异常
   在此，TcExcept{} 同样再次进行空指针访问；然而，这次异常是在 FilterException() 方法中处理后转发的，这样其他 TcExcept{} 也能处理异常。

15.1 Sample24：信号量 8

本文介绍了信号的使用情况。

下载

您可以在此访问该示例的源代码：

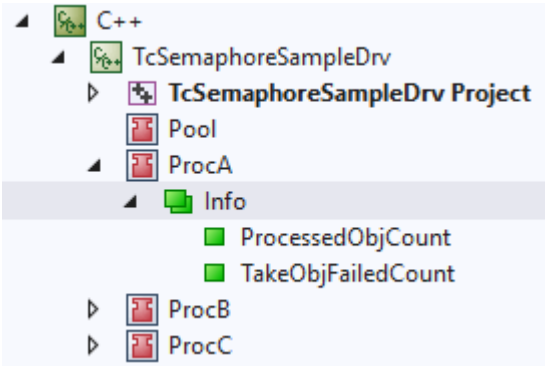
https://github.com/Beckhoff/TC1300_Samples/tree/main/S24-Semaphores

1. 点击**打开项目……**，打开 TwinCAT 3 中包含的该项目的 zip 文件。
2. 选择目标系统。
3. 在本地机器上构建示例（例如，**构建** -> **构建解决方案**）。
4. 请注意本页**配置**下所列操作。
5. 点击  激活配置。

⇨ 示例已做好操作准备。

功能说明

本示例包含一个“LargeObjPool”，其中有两个供“LargeObjProcessors”处理的“LargeObj”。



“LargeObjProcessors”由运行在不同 CPU 内核上的具有不同优先级的不同任务控制。在信息数据区，它们会统计处理工作可以执行的频率 ("Info->ProcessedObjCount") 以及在延迟时间内没有对象可供处理的频率 ("Info->TakeObjFailedCount")。

示例说明

在 PS 转换中，LargeObjPool 通过 `InitPool()` 方法进行初始化，其中包含两个 LargeObj（由参数“ObjCount”定义）。池主要提供两种方法，供“LargeObjProcessors”使用：

- `TakeObj()` 返回一个 LargeObj（如果有的话）。系统会根据信号等待超时，以查看是否有对象可用。对象本身存储在映射中。映射的访问受 CriticalSection 保护。
- `ReturnObj()` 会将处理过的对象提供给池，以便进一步处理。

其 `CycleUpdate()` LargeObjProcessor 使用 `TakeObj()` 从池中获取 LargeObj 进行处理。如果操作成功，则会通过辅助函数 `ConsumeTime()` 进行模拟处理，然后再通过 `ReturnObj` 将 LargeObj 返回池。

在关闭系统之前，必须完成所有 LargeObj 的处理。此操作通过 `TcTryLockOpState()` 和 `TcReleaseOpState()` 由 LargeObjPool 通知 TwinCAT 系统正在处理一个对象来实现。只要存储的计数器不为 0，TcCOM 对象“Pool”就会被重置，稍后才会通过 OP -> SafeOp 转换关闭。

15.1 Sample25：静态库

9

本文介绍了静态 TwinCAT 3 C++ 库的模块实现和使用。

下载

您可以在此访问该示例的源代码：

https://github.com/Beckhoff/TC1300_Samples/tree/main/S25-StaticLibrary

1. 利用已安装 TwinCAT 的 Visual Studio，通过**打开项目.....**来打开项目
2. 通过右键单击**项目 -> 属性 -> Tc 签名**打开 TwinCAT 签名，为该项目配置签名，必要时配置证书和密码。有关签署 C++ 项目的更多信息，请参见 TwinCAT [▶ 22] 章节。
3. 选择目标系统。
4. 构建示例（例如**构建 -> 构建解决方案**）。
5. 点击  激活配置。

⇨ 示例已做好操作准备。

功能说明

示例包含两个项目，DriverUsingStaticLib 项目使用 StaticLib 项目的静态内容。

StaticLib:
一方面，StaticLib 会在 StaticFunction.h/.cpp 中提供功能 ComputeSomething。
另一方面，会定义接口 ISampleInterface（见 TMCEditor），并在 MultiplicationClass 中实现。

DriverUsingStaticLib:
在使用 StaticLib 的模块的 CycleUpdate 方法中，同时使用了 StaticLib 的类和功能。

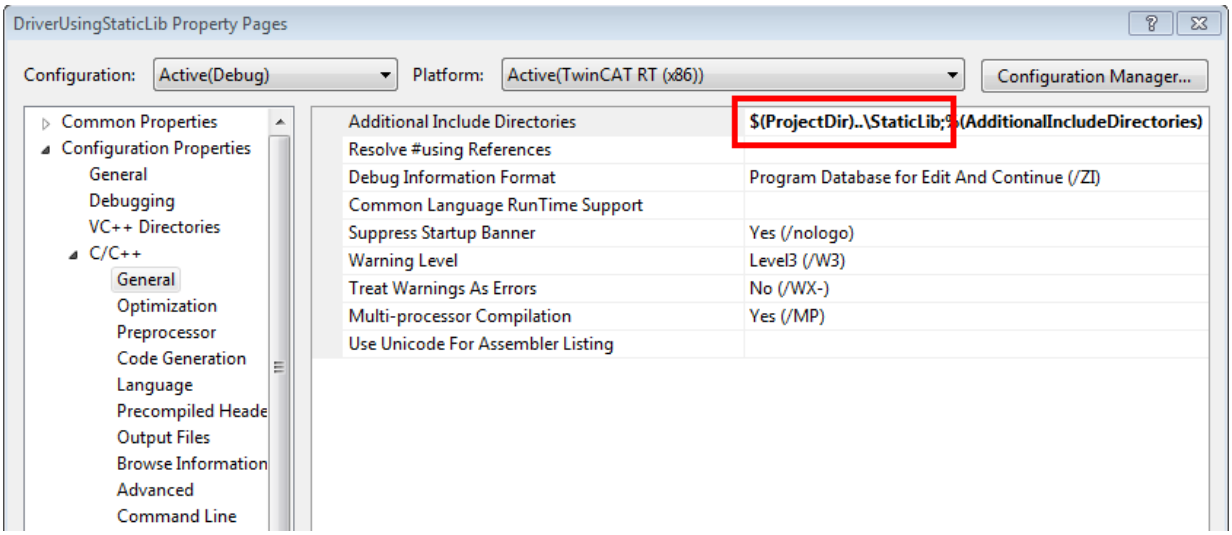
理解示例

请按照以下步骤创建和使用静态库。

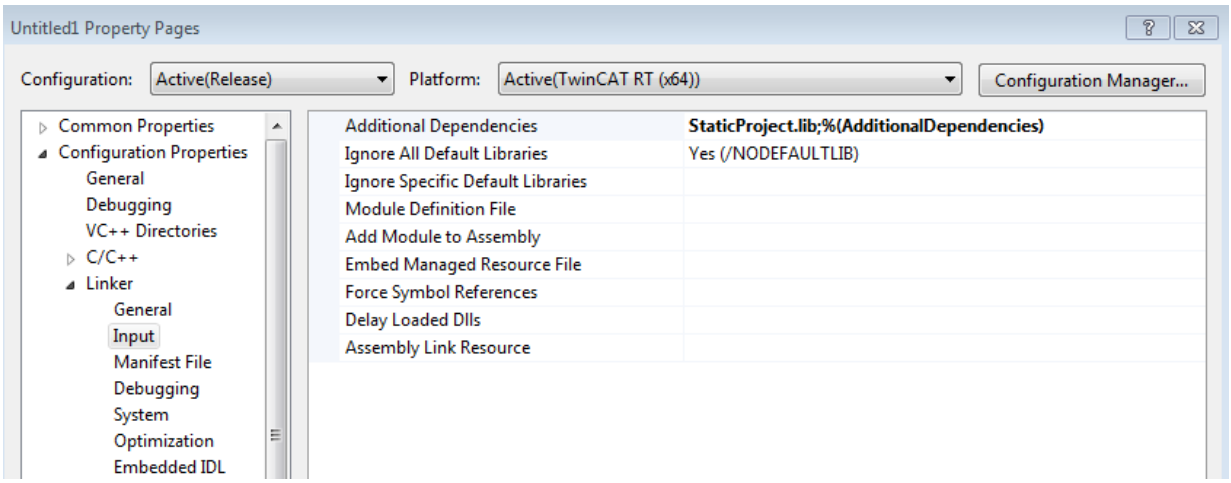
手动重新编译
请注意，在驱动程序开发过程中，Visual Studio 不会自动重新编译静态库。手动操作。

✓ 在开发 C++ 项目过程中，使用 TwinCAT Static Library Project 模板来创建静态库。
✓ 在以下步骤中，请使用 VisualStudio 的**编辑**对话框，然后使用 %(AddtitionalIncludeDirectories) 或 %(AdditionalDependencies)。

1. 在驱动程序中，在**附加包含目录**下向编译器添加静态库目录。



2. 在使用静态库的驱动程序中，将其添加为链接器的附加依赖项。打开驱动程序的项目属性，添加静态库：



15.2 Sample26：任务的执行顺序0

本文介绍了在为一个任务分配多个模块的情况下如何确定任务执行顺序。

下载

您可以在此访问该示例的源代码：

https://github.com/Beckhoff/TC1300_Samples/tree/main/S26-SortOrder

1. 利用已安装 TwinCAT 的 Visual Studio，通过**打开项目……**来打开项目
2. 通过右键单击**项目 -> 属性 -> Tc 签名**打开 TwinCAT 签名，为该项目配置签名，必要时配置证书和密码。有关签署 C++ 项目的更多信息，请参见 TwinCAT [▶ 22] 章节。
3. 选择目标系统。
4. 构建示例（例如**构建 -> 构建解决方案**）。
5. 点击 激活配置。

⇨ 示例已做好操作准备。

功能说明

示例包含 SortOrder 模块，该模块已进行两次实例化。排序顺序决定执行顺序，可通过 TwinCAT 模块实例配置器 [▶ 129] 进行配置。

例如，CycleUpdate 方法会跟踪对象名称和 ID，以及该模块的排序顺序。在控制台窗口中可以看到执行顺序：

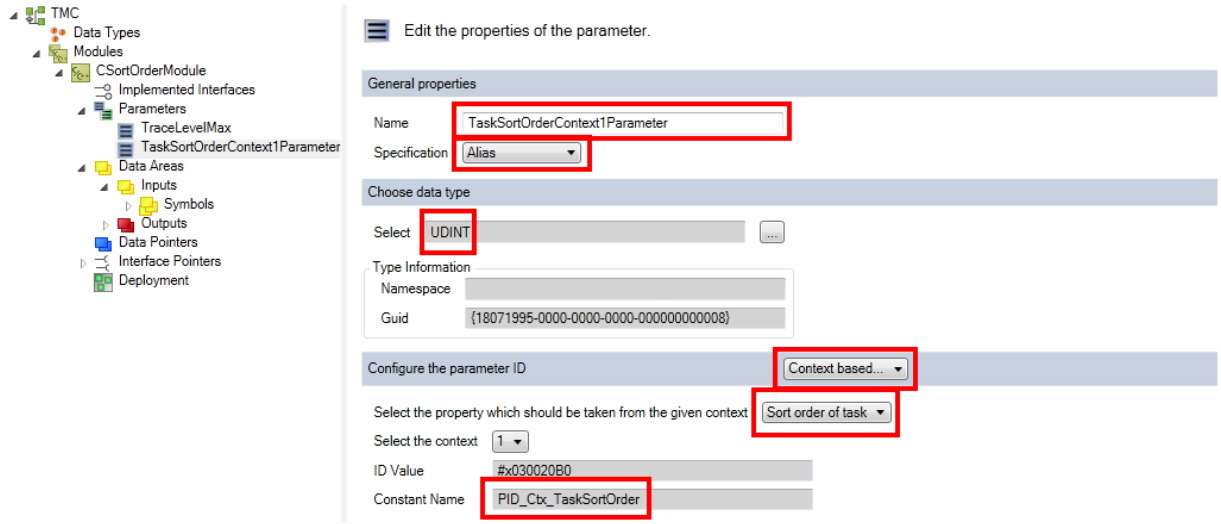
```
32 05.09.2014 13:01:14 343 ms | 'TCOM Server' (10): CSortOrderModule::CycleUpdate() I am 'SortOrder1' (0x01010010) w/ SortOrder 150
33 05.09.2014 13:01:14 343 ms | 'TCOM Server' (10): CSortOrderModule::CycleUpdate() I am 'SortOrder2' (0x01010020) w/ SortOrder 170
34 05.09.2014 13:01:15 343 ms | 'TCOM Server' (10): CSortOrderModule::CycleUpdate() I am 'SortOrder1' (0x01010010) w/ SortOrder 150
35 05.09.2014 13:01:15 343 ms | 'TCOM Server' (10): CSortOrderModule::CycleUpdate() I am 'SortOrder2' (0x01010020) w/ SortOrder 170
36 05.09.2014 13:01:16 343 ms | 'TCOM Server' (10): CSortOrderModule::CycleUpdate() I am 'SortOrder1' (0x01010010) w/ SortOrder 150
37 05.09.2014 13:01:16 343 ms | 'TCOM Server' (10): CSortOrderModule::CycleUpdate() I am 'SortOrder2' (0x01010020) w/ SortOrder 170
```

在示例中，一个实例配置为 Sort Order 150，一个配置为 Sort Order 170，而这两个实例均已分配给一个任务。

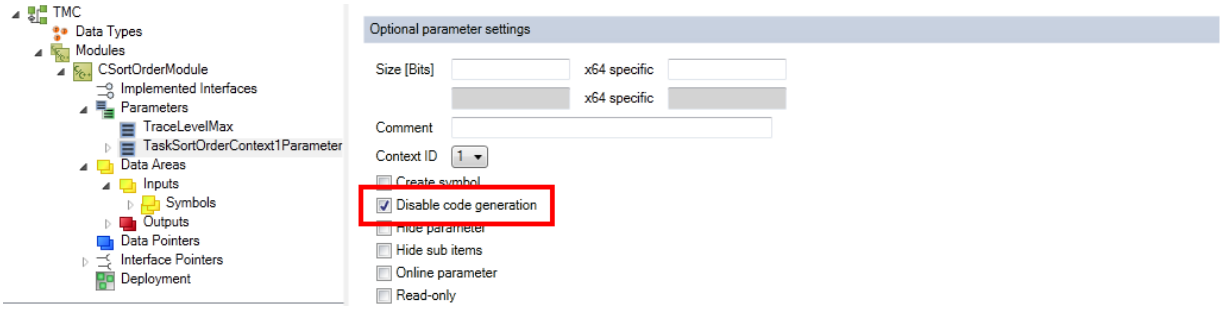
示例说明

✓ 带有循环 IO 的 TcCOM C++ 模块。

1. 该模块需要一个基于上下文的参数“任务排序顺序”，自动选择 PID_Ctx_TaskSortOrder 作为名称。请注意，参数必须是数据类型 UDINT 的别名（规范）：



2. 启动 TMC Code Generator，以获取标准执行。
3. 由于代码会在下一步中修改，因此现在禁用该参数的代码生成。



4. 确保在重启 TMC Code Generator 之前接受更改：
查看 CPP 模块（示例中的 SortOrderModule.cpp）。循环任务调用器的智能指针实例包含信息数据，包括排序顺序字段。参数值存储在该字段中。

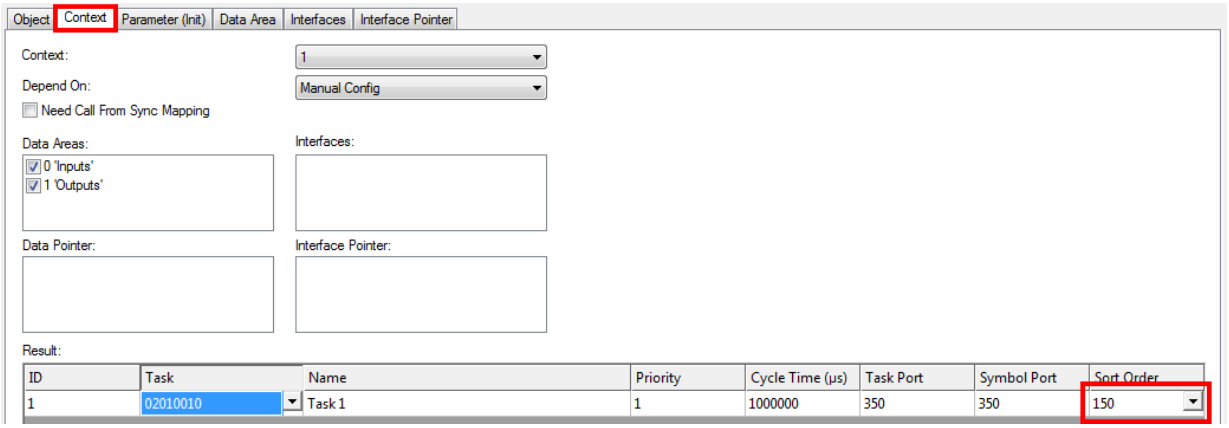
```
////////////////////////////////////////////////////////////////////////////////
// Set parameters of CSortOrderModule
BEGIN_SETOBJPARA_MAP(CSortOrderModule)
    SETOBJPARA_DATAAREA_MAP()
///<AutoGeneratedContent id="SetObjectParameterMap">
    SETOBJPARA_VALUE(PID_TcTraceLevel, m_TraceLevelMax)
```

```
    SETOBJPARA_ITFPTR(PID_Ctx_TaskOid, m_spCyclicCaller)
///</AutoGeneratedContent>
    SETOBJPARA_TYPE_CODE(PID_Ctx_TaskSortOrder, ULONG, m_spCyclicCaller.GetInfo()-
>sortOrder=*p) //ADDED
    //generated code: SETOBJPARA_VALUE(PID_Ctx_TaskSortOrder, m_TaskSortOrderContext1Parameter)
END_SETOBJPARA_MAP()

////////////////////////////////////////////////////////////////////////////////
// Get parameters of CSortOrderModule
BEGIN_GETOBJPARA_MAP(CSortOrderModule)
    GETOBJPARA_DATAAREA_MAP()
///<AutoGeneratedContent id="GetObjectParameterMap">
    GETOBJPARA_VALUE(PID_TcTraceLevel, m_TraceLevelMax)
    GETOBJPARA_ITFPTR(PID_Ctx_TaskOid, m_spCyclicCaller)
///</AutoGeneratedContent>
    GETOBJPARA_TYPE_CODE(PID_Ctx_TaskSortOrder, ULONG, *p=m_spCyclicCaller.GetInfo()-
>sortOrder) //ADDED
    //generated code: GETOBJPARA_VALUE(PID_Ctx_TaskSortOrder, m_TaskSortOrderContext1Parameter)
END_GETOBJPARA_MAP()
```

5. 在本示例中，对象名称、ID 和排序顺序会循环跟踪：

```
    // TODO: Add your cyclic code here
     m_counter+=m_Inputs.Value;
     m_Outputs.Value=m_counter;
    m_Trace.Log(tlAlways, FNAMEA "I am '%s' (0x%08x) w/ SortOrder %d ", this->TcGetObjectName(),
this->TcGetObjectId() , m_spCyclicCaller.GetInfo()->sortOrder); //ADDED
```

6. 排序顺序也可以作为 ITcCyclicCaller::AddModule() 方法的第四个参数传输，该方法在 CModuleA::AddModuleToCaller() 中使用。
7. 为该模块的实例分配**周期间隔较长**（例如 1000 毫秒）的任务，以限制发送到 TwinCAT Engineering 系统的跟踪信息。
8. 通过 TwinCAT 模块实例配置器 [▶ 129]：



为每个实例指定不同的排序顺序

15.2 Sample30：时序测量

1

本文介绍如何实现包含时间测量功能的 TwinCAT 3 C++ 模块。

下载

您可以在此访问该示例的源代码：

https://github.com/Beckhoff/TC1300_Samples/tree/main/S30-Timing

1. 利用已安装 TwinCAT 的 Visual Studio，通过**打开项目……**来打开项目
2. 通过右键单击**项目 -> 属性 -> Tc 签名**打开 TwinCAT 签名，为该项目配置签名，必要时配置证书和密码。有关签署 C++ 项目的更多信息，请参见 TwinCAT [▶ 22] 章节。
3. 选择目标系统。
4. 构建示例（例如**构建 -> 构建解决方案**）。
5. 点击  激活配置。

⇨ 示例已做好操作准备。

功能说明

本示例专门处理时间测量，例如

- 以纳秒为单位查询任务循环时间
- 查询任务优先级
- 查询自 1601 年 1 月 1 日 (UTC) 起以 100 纳秒为间隔的任务循环开始时间
- 查询自 2000 年 1 月 1 日起以纳秒为单位的任务循环开始时的分布式时钟时间
- 查询自 1601 年 1 月 1 日 (UTC) 起以 100 纳秒为间隔调用方法的时间

另请参见

ITcTask 接口 [▶ 182]

15.2 Sample31：TwinCAT3 C++ 中的功能块 TON 2

本文介绍用 C++ 实现行为，该行为与 PLC/IEC-61131-3 的 TON 功能块相当。

源

您可以在此访问该示例的源代码：

https://github.com/Beckhoff/TC1300_Samples/tree/main/S31-CTON

1. 利用已安装 TwinCAT 的 Visual Studio，通过**打开项目.....**来打开项目
2. 通过右键单击**项目 -> 属性 -> Tc 签名**打开 TwinCAT 签名，为该项目配置签名，必要时配置证书和密码。有关签署 C++ 项目的更多信息，请参见 TwinCAT [▶ 22] 章节。
3. 选择目标系统。
4. 构建示例（例如**构建 -> 构建解决方案**）。
5. 点击  激活配置。

⇨ 示例已做好操作准备。

功能说明

此模块的行为与使用 Cyclic IO 向导创建的模块相当。m_input.Value 已添加到 m_Output.Value 中。与循环 IO 模块不同的是，该模块只会在规定时间间隔（1000 毫秒）结束后将 m_input.Value 添加到 m_Output.Value。

借助 CTON 类实现操作，该类与 PLC/61131 的 TON 功能块相当。

示例说明

C++ 类 CTON (TON.h/.cpp) 可提供 PLC/61131 的 TON 功能块的行为。Update() 方法与功能块的残余部分相当，必须定期调用。

Update() 方法包含两个“in”参数：

- IN1：在上升沿启动定时器开关，在下降沿复位定时器开关。
- PT：功能说明 Q 设置前的等待时间。

以及两个“out”参数：

- Q：如果 PT 在 IN 后几秒出现上升沿，则为 TRUE。

- ET：指定已用时间。

除此之外，还必须提供 ITcTask 来查询时基。

另请参见

Sample30：时序测量 [▶ 298]

ITcTask 接口 [▶ 182]

15.2 Sample35：接入以太网 3

本文介绍通过以太网卡直接通信的 TwinCAT 3 C++ 模块的实现。示例代码通过 ARP 数据包的周期性传输和接收，向通信伙伴查询硬件地址 (MAC)。

本示例说明如何直接访问以太网卡。TF6311 TCP/UDP RT 功能可在 TCP 和 UDP 的基础上访问以太网卡，因此无需在此示例的基础上实现网络堆栈。

下载

您可以在此访问该示例的源代码：

https://github.com/Beckhoff/TC1300_Samples/tree/main/S35-AccessEthernet

1. 点击**打开项目……**，打开 TwinCAT 3 中包含的该项目的 zip 文件。
2. 选择目标系统。
3. 在本地机器上构建示例（例如，**构建** -> **构建解决方案**）。
4. 请注意本页**配置**下所列操作。
5. 点击  激活配置。

⇨ 示例已做好操作准备。

功能说明

本示例包含 TcEthernetSample 模块的实例，该模块会发送和接收 ARP 数据包，目的是确定远程硬件地址 (MAC)。

CycleUpdate 方法实现用于发送 ARP 数据包并等待超时响应的基本状态机。

本示例使用 TwinCAT 的两个以太网组件：

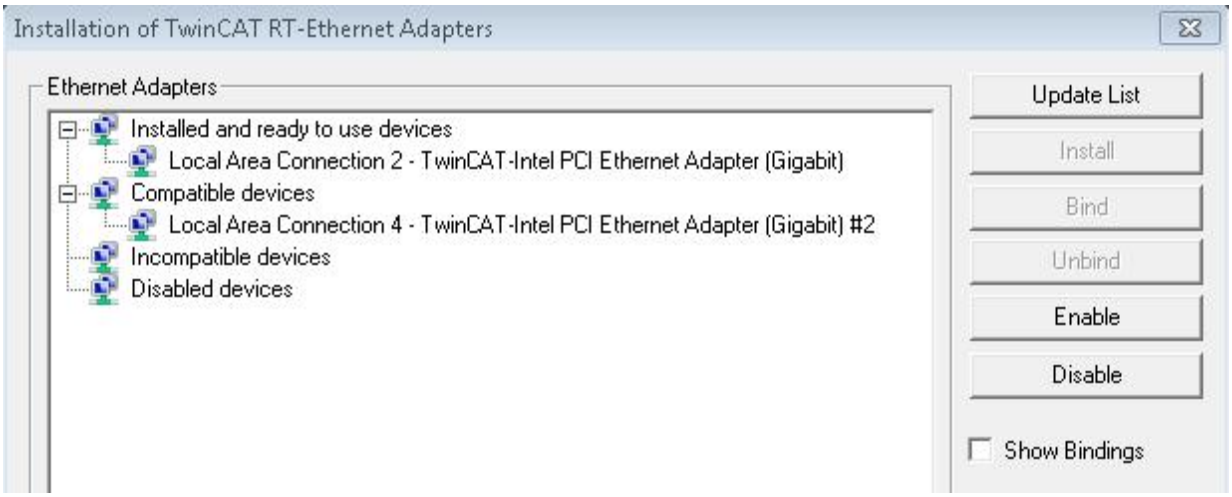
1. **ITcEthernetAdapter**（示例中的实例名称为 m_spEthernetAdapter）代表 RT 以太网适配器。能够访问适配器参数，如硬件 MAC 地址、链接速度和链接错误。可用于发送以太网帧，并使模块实例能够通过 registerProtocol 方法自行注册为 ItcIoEthProtocol。
2. **ITcIOoEthProtocol** 由采样模块扩展，可确保在发生以太网事件时通过 **ITcEthernetAdapter** 发出通知。

配置

下载的 TwinCAT 项目必须配置为在网络环境中执行。请执行以下步骤：

✓ 本示例要求以太网卡使用 TwinCAT 驱动程序。

1. 从 XAE 通过菜单 **TwinCAT -> 显示实时以太网兼容设备……** 或从 XAR 系统的硬盘启动 TcRteInstall.exe。



2. 您可能需要借助按钮来安装和激活驱动程序。
3. TwinCAT 必须确定要使用哪个以太网卡。在 **XAE** 中打开项目，点击**选择 I/O/设备/设备 1（RT 以太网适配器）**。
4. 点击**适配器**选项卡，通过**搜索**选择适配器。
5. 必须配置 TcEthernetSample_Obj1。打开实例窗口并设置以下值：
   参数 (Init)：SenderIpAddress（在第 2 步中配置的网络适配器的 IP）
   参数 (Init)：TargetIpAddress（目标主机的 IP）
   接口指针：EthernetAdapter 必须指向 I/O/设备/设备 1 (RT-Ethernet Adapter)。

15.2 Sample37：归档数据

4

TcCOM Object Archive 示例介绍了在初始化和反初始化过程中恢复和保存对象状态的过程。

TwinCAT 支持保留数据

TwinCAT 还支持保留数据，以便利用设备的 NOVRAM 使数据持久化。

下载

您可以在此访问该示例的源代码：

https://github.com/Beckhoff/TC1300_Samples/tree/main/S37-ArchiveData

1. 利用已安装 TwinCAT 的 Visual Studio，通过**打开项目……**来打开项目
2. 通过右键单击**项目 -> 属性 -> Tc 签名**打开 TwinCAT 签名，为该项目配置签名，必要时配置证书和密码。
   有关签署 C++ 项目的更多信息，请参见 TwinCAT [▶ 22] 章节。
3. 选择目标系统。
4. 构建示例（例如**构建 -> 构建解决方案**）。
5. 点击  激活配置。

⇨ 示例已做好操作准备。

功能说明

TcCOM 对象存档示例介绍了在初始化和反初始化过程中恢复和保存对象状态的过程。示例类 CmoduleArchive 的状态与计数器 CModuleArchive::m_counter 的值相对应。

在从 PREOP 向 SAFEOP 转换期间，即调用 CModuleArchive::SetObjStatePS() 方法时，对象归档服务器 (ITComObjArchiveServer) 用于创建供读取的对象归档，可通过接口 ITComArchiveOp 进行访问。该接口可提供 operator>>() 的重载，以便在存档中进行读取。

在从 SAFEOP 向 PREOP 转换期间，即调用 CModuleArchive::SetObjStateSP() 方法时，TCOM 对象归档服务器用于创建供写入的对象归档，可通过接口 ITComArchiveOp 进行访问。该接口可提供 operator<<() 的重载，以便在存档中进行写入。

此处使用的接口并不是为实时环境 [▶ 43]开发，因此该接口只能在非实时环境中使用。

15.2 TcCOM 示例 5

模块可在 PLC 和 C++ 之间进行通信。因此，该说明涵盖了在 PLC 侧对 C++ 模块的处理和在 C++ 侧对 PLC 的处理。
此处显示的是与 PLC 通信的 TcCOM 示例。

TcCOM_Sample01 示例 [▶ 302]显示了两个 PLC 之间如何进行 TcCOM 通信。在此过程中，一个 PLC 的功能可直接从另一个 PLC 调用。

TcCOM_Sample02 示例 [▶ 312]显示了 PLC 应用程序如何使用 TwinCAT C++ 类现有实例的功能。因此，用 C++（或 MATLAB®）写入的算法可在 PLC 中轻松使用。
虽然在使用现有 TwinCAT C++ 模块的情况下，需要在目标系统上获得 TwinCAT C++ 授权，但在目标系统或开发计算机上不需要 C++ 开发环境。

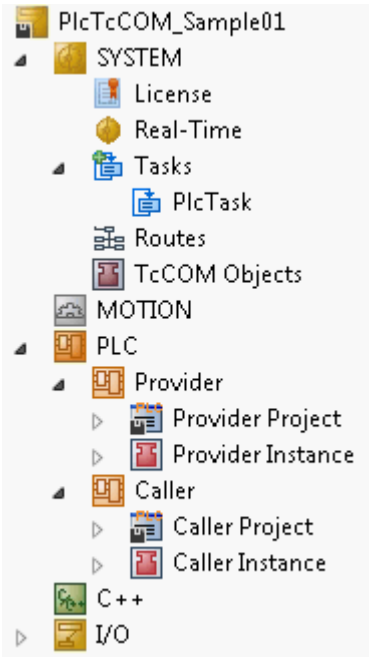
TcCOM_Sample03 示例 [▶ 316]显示了 PLC 应用程序如何通过同时生成一个 C++ 类实例来使用 TwinCAT C++ 类的功能。与之前的示例相比，可以提高灵活性。

15.25.1 TcCOM_Sample01_PlcToPlc

本示例介绍了两个 PLC 之间的 TcCOM 通信。

第一个 PLC（在示例中也称为“提供者”）中的功能块提供的功能被第二个 PLC（在示例中也称为“调用者”）调用。为此，无需复制功能块或其程序代码。 相反，程序会直接操作第一个 PLC 中的对象实例。

两个 PLC 都必须处于 TwinCAT Runtime 中。在这种连接中，功能块通过全局定义的接口在全系统范围内提供其方法，并表示自身为 TcCOM 对象。与每个 TcCOM 对象一样，此类功能块也会在运行时中的 **TcCOM 对象** 节点中列出。



以下子章节将对该程序进行说明：

- 在第一个 PLC 中创建一个 FB，在全局范围内提供其功能 [▶ 303]
- 在第二个 PLC 中创建一个 FB，作为一个简单的代理，该 FB 也能提供这种功能 [▶ 307]
- 执行示例项目 [▶ 310]

下载示例：https://infosys.beckhoff.com/content/1033/TC3_C/Resources/2343046667.zip

使用多任务（多线程）时的竞态条件

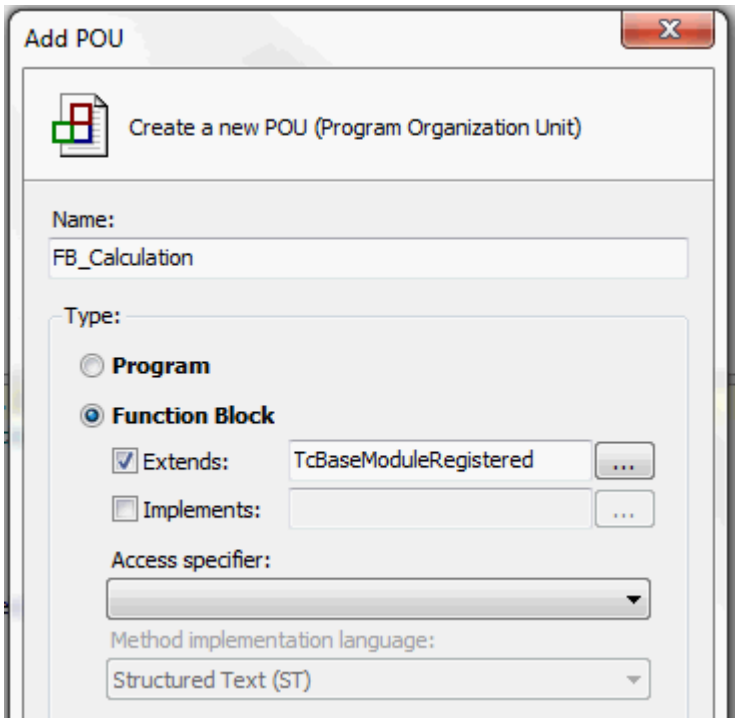
在全局范围内提供功能的功能块在第一个 PLC 中实例化。它可以像任何功能块一样使用。此外，如果从不同的 PLC（或例如从 C++ 模块）使用，请确保所提供的方法是线程安全的，因为根据系统配置的不同，各种调用可能从不同的任务环境同时进行或相互中断。在这种情况下，方法不得访问功能块的成员变量或第一个 PLC 的全局变量。如果确实有必要这么做，则必须防止同时访问。观察 Tc2_System 库中的 TestAndSet() 功能。

系统要求

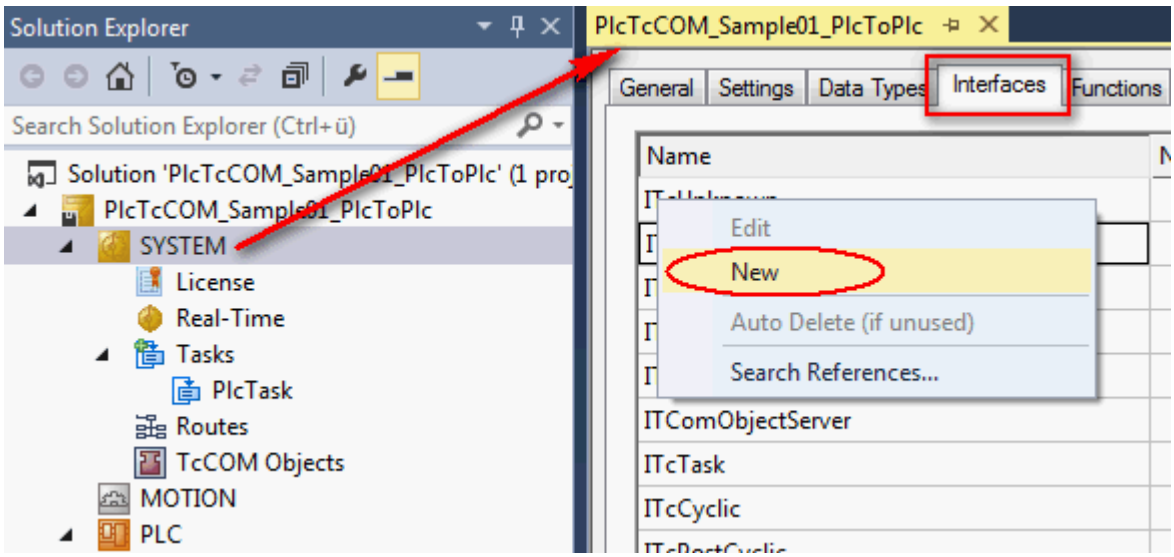
TwinCAT 版本	硬件	要集成的库
TwinCAT 3.1, Build 4020	x86, x64, ARM	Tc3_Module

15.25.1.1 在第一个 PLC 中创建一个 FB，在全局范围内提供其功能

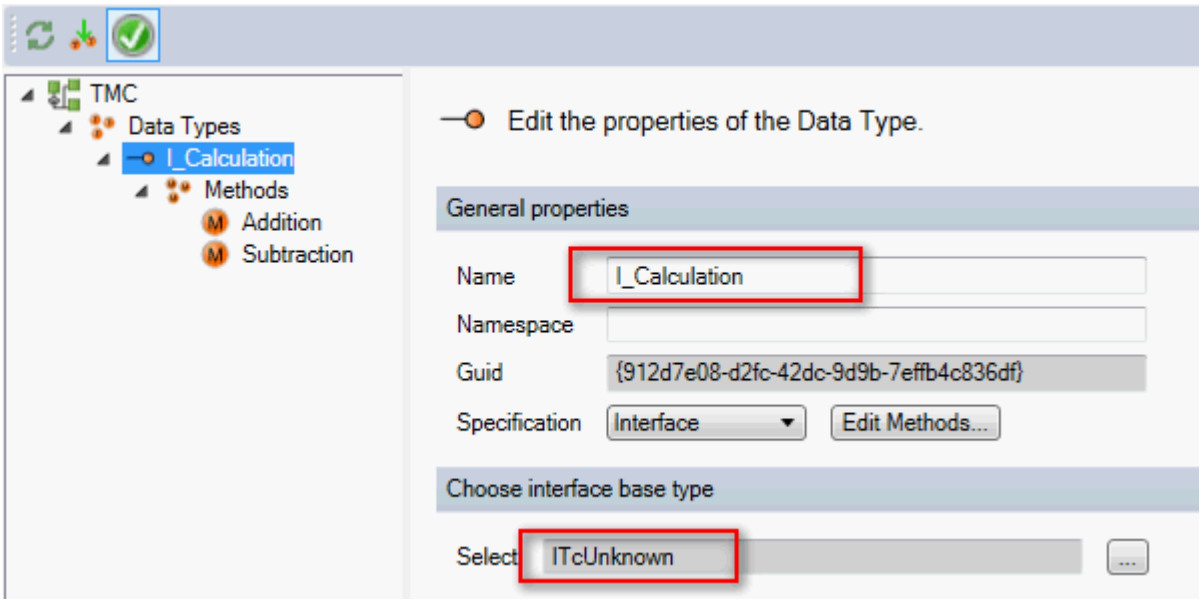
1. 创建一个 PLC 和一个新的功能块 (FB)（本例中为 FB_Calculation）。从 TcBaseModuleRegistered 类派生功能块，这样，功能块的实例不仅可以在同一 PLC 中使用，也可以在第二个 PLC 中使用。您也可以修改现有 PLC 中的 FB。



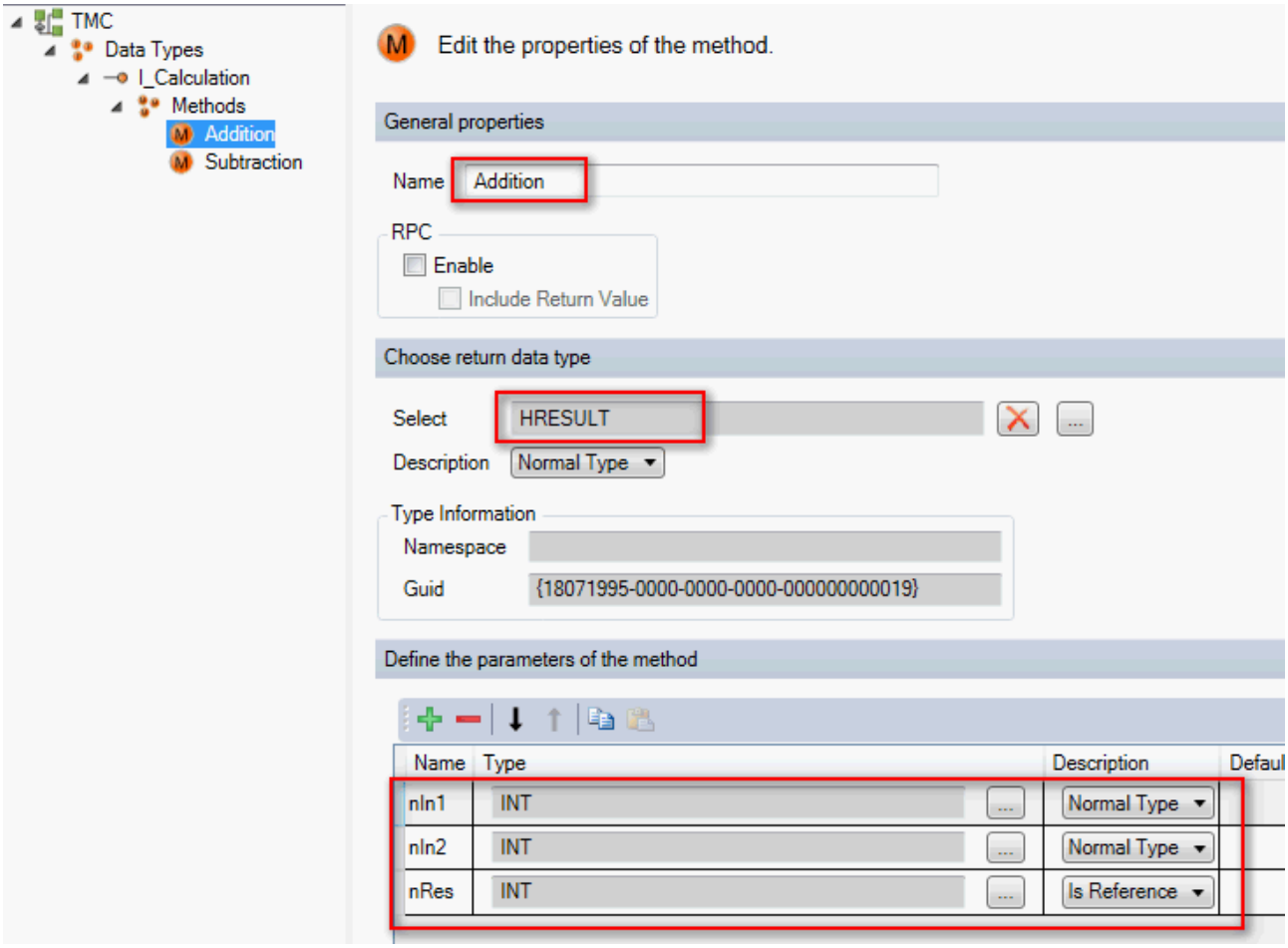
2. 功能块必须通过方法提供功能。方法在全局接口中定义，其类型在整个系统中都是已知的，与编程语言无关。要创建全局接口，请打开系统属性中“接口”选项卡的环境菜单，然后选择“新建”。
   ⇨ 打开 TMC Editor，帮助创建全局接口。



3. 输入名称（本例中为 I_Calculation），并附加所需方法。该接口自动从 ITcUnknown 派生，以符合 TwinCAT TcCOM 模块概念。



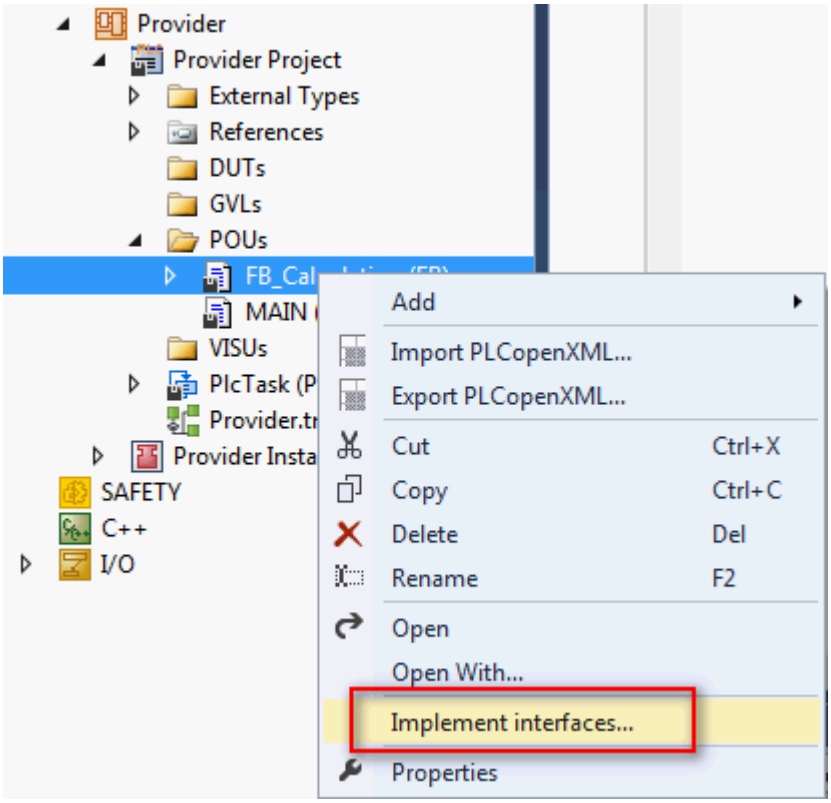
4. 输入方法名称（本例中为 Addition() 和 Subtraction()），并选择 HRESULT 作为返回数据类型。如果要实现这种 TcCOM 通信，则必须使用这种返回类型。
5. 最后，指定方法参数并关闭 TMC Editor。



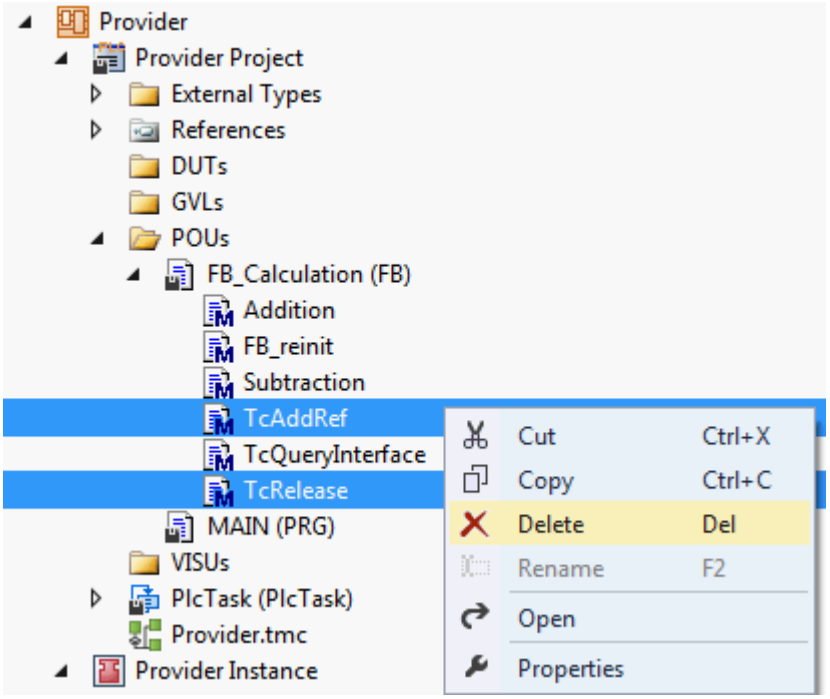
6. 现在，在 FB_Calculation 功能块中实现 I_Calculation 接口，并附加“c++_compatible”属性。

```
{attribute 'c++_compatible'}
FUNCTION_BLOCK FB_Calculation EXTENDS TcBaseModuleRegistered IMPLEMENTS I_Calculation

VAR
END_VAR

```

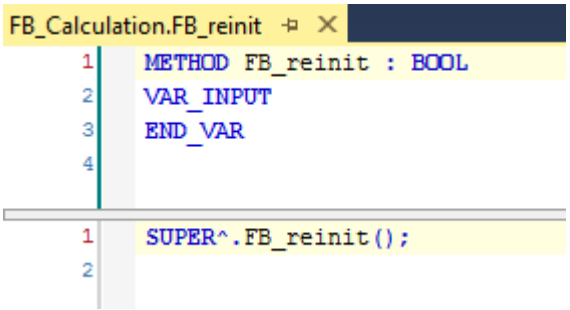
7. 选择功能块环境菜单中的“实现接口.....” 选项，以获取属于该接口的方法。



8. 删除两个方法 TcAddRef() 和 TcRelease()，因为要使用基类的现有实现。



9. 为 FB_Calculation 功能块创建 FB_reinit() 方法，并调用基本实现。这样可确保基类的 FB_reinit() 方法在“在线更改”期间运行。这是至关重要的。

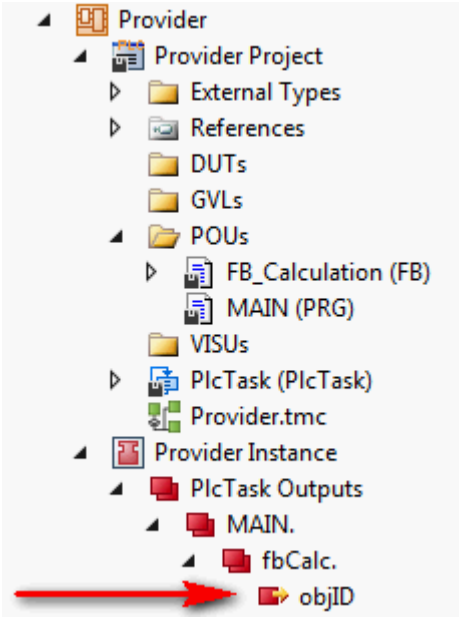


10. 实现 ITcUnknown 接口 [▶ 186]的 TcQueryInterface() 方法。通过该方法，其他 TwinCAT 组件可以获得指向该功能块实例的接口指针，从而执行方法调用。如果功能块或其基类提供通过 iid（接口 ID）请求的接口，则调用 TcQueryInterface 成功。在这种情况下，所传递的接口指针被分配至功能块的一个地址，并改变其类型，同时使用 TcAddRef() 递增引用计数器。
11. 向 Addition() 和 Subtraction() 方法添加相应代码，以提供以下功能：`nRes := nIn1 + nIn2` 和 `nRes := nIn1 - nIn2`
12. 将该功能块的一个或多个实例添加到 MAIN 程序块或全局变量列表中。

    ⇨ 第一个 PLC 的实现已完成。

```
MAIN*
1  PROGRAM MAIN
2  VAR
3      m : UDINT;
4
5      fbCalc : FB_Calculation('MAIN.fbCalc');
6  END_VAR
7
```

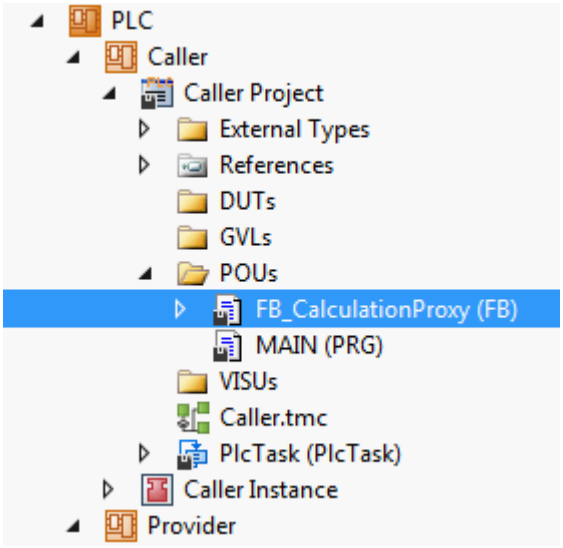
⇨ 编译 PLC 后，代表 FB_Calculation 实例的 TcCOM 对象的对象 ID 可作为过程映像中的出口。



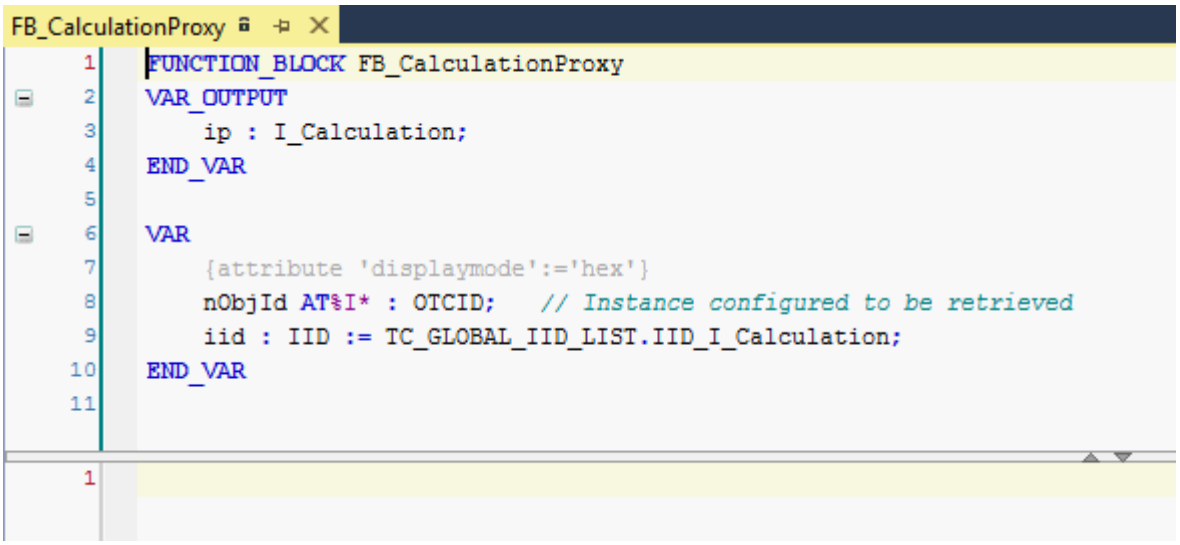
15.25.1.2 在第二个 PLC 中创建一个同样提供这种功能的 FB，作为一个简单的代理，

1. 创建 PLC 并在其中附加新的功能块。

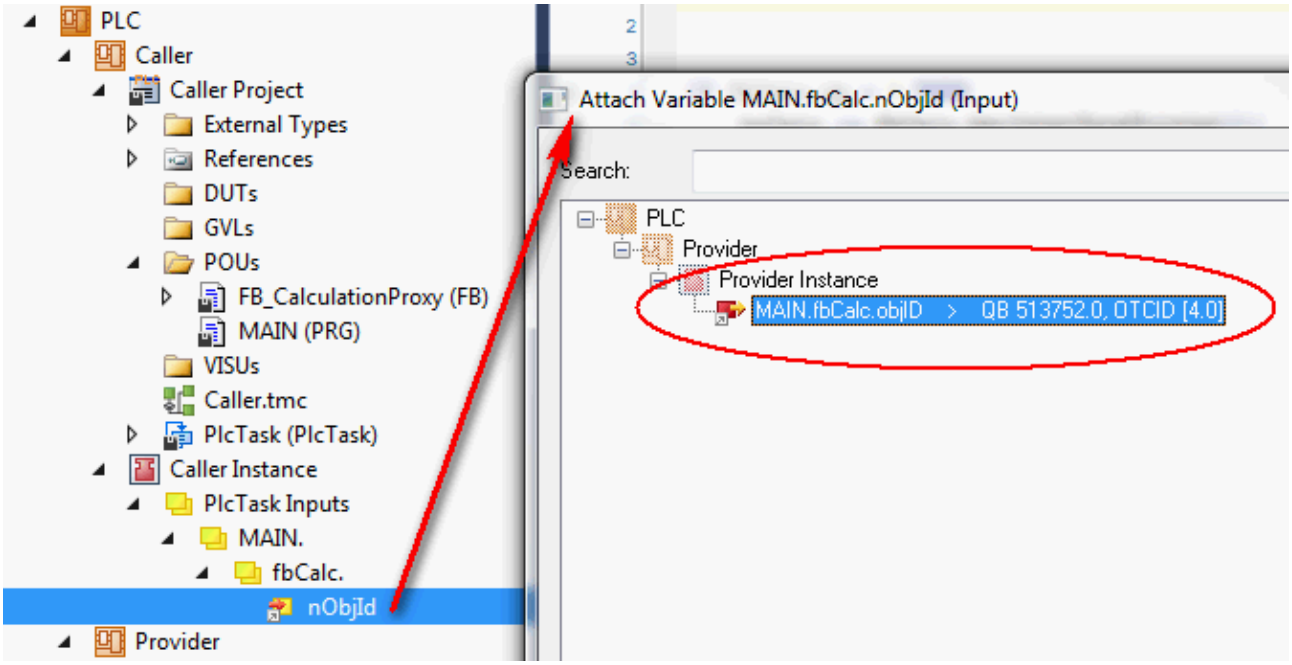
   ⇨ 该代理功能块应提供在第一个 PLC 中编程的功能。通过全局接口 I_Calculation 类型的接口指针来实现这一功能。



2. 在功能块的声明部分，声明一个指向全局接口的接口指针作为输出，全局接口随后向外提供功能。



3. 此外，将对象 ID 和接口 ID 创建为本地成员变量。
虽然接口 ID 可通过全局列表获得，但对象 ID 是通过过程映像中的链接分配的。



4. 执行 PLC 代理功能块。首先在功能块中添加 GetInterfacePointer() 方法。
借助 FW_ObjMgr_GetObjectInstance() 函数，接口指针被获取到指定 TcCOM 对象的指定接口。只有当对象 ID 有效且接口指针尚未分配时，才会执行此操作。对象本身会递增引用计数器。

FB_CalculationProxy.GetInterfacePointer

```
METHOD GetInterfacePointer : HRESULT
VAR
END_VAR

IF nObjID <> 0 THEN
    IF (ip = 0) THEN // only get interface pointer if it is not already existing
        GetInterfacePointer := FW_ObjMgr_GetObjectInstance(oid:=nObjID, iid:=iid, pipUnk:=ADR(ip));
    ELSE
        GetInterfacePointer := E_HRESULTAdsErr.EXISTS;
    END_IF
ELSE
    GetInterfacePointer := E_HRESULTAdsErr.INVALIDOBJID;
END_IF
```

5. 必须再次释放使用过的引用。为此，请在功能块的 FB_exit 析构函数中调用 FW_SafeRelease() 函数。

FB_CalculationProxy.FB_exit | FB_CalculationProxy.GetInterfacePointer

```
{attribute 'hide'}
METHOD FB_exit : BOOL
VAR_INPUT
    bInCopyCode : BOOL; // if TRUE, the exit method is c
END_VAR

IF NOT bInCopyCode THEN // if not online change
    FW_SafeRelease(ADR(ip));
END_IF
```

⇨ 至此，代理功能块的执行工作完成。

6. 在应用程序中实例化代理功能块 FB_CalculationProxy，并调用其方法 GetInterfacePointer() 获取有效的接口指针。
在应用程序中声明代理块的实例，以调用通过接口提供的方法。调用本身全部通过定义为功能块输出的接口指针执行。对于指针来说，必须事先进行空值检查。然后就可以直接调用这些方法，也可以通过 Intellisense 调用。

MAIN*

```
PROGRAM MAIN
VAR
    fbCalc : FB_CalculationProxy;
    hrCalc : HRESULT;
    a : INT := 10;
    b : INT := 7;
    nSum : INT; // a + b
    nDiff : INT; // a - b
END_VAR
```

```
IF fbCalc.ip = 0 THEN
    hrCalc := fbCalc.GetInterfacePointer();
END_IF
IF fbCalc.ip <> 0 THEN
    hrCalc := fbCalc.ip.Addition(a,b,nSum);
    hrCalc := fbCalc.ip.Subtraction(a,b,nDiff);
END_IF
```

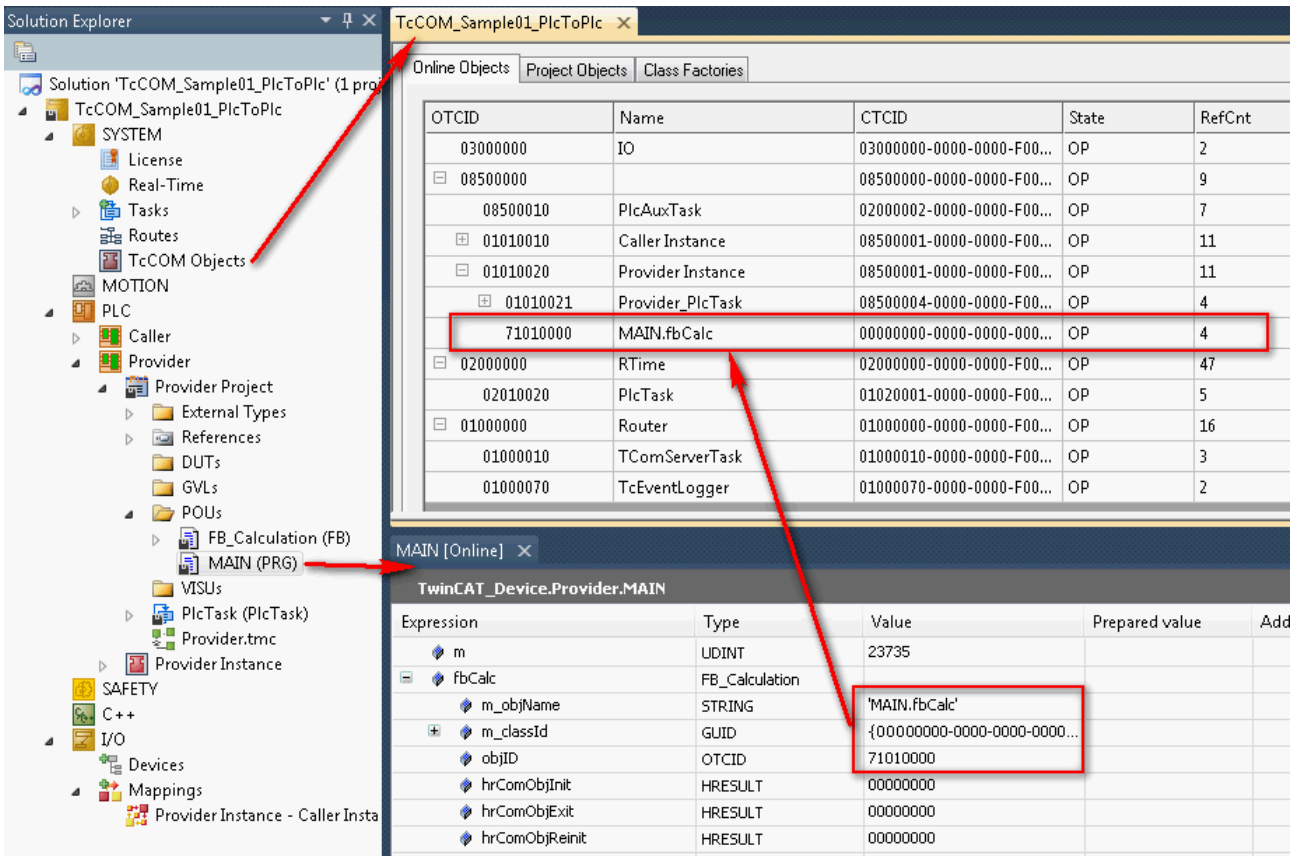
⇨ 示例已准备好进行测试。

顺序无关

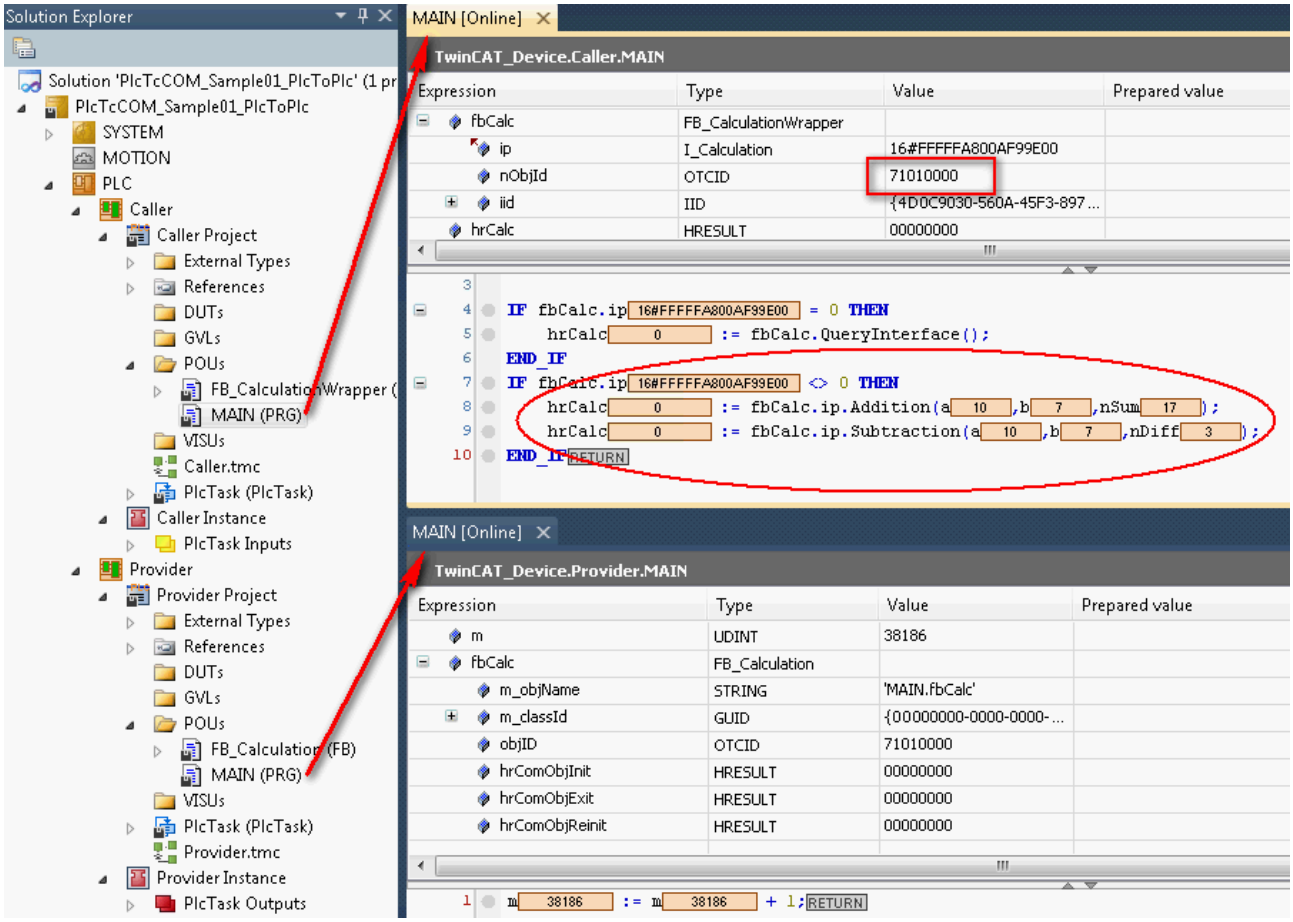
在本实现方案中，两个 PLC 的启动顺序无关紧要。

15.25.1.3 执行示例项目

1. 选择目标系统并编译项目。
2. 启用 TwinCAT 配置，执行登录并启动两个 PLC。
   ⇨ 在 PLC 应用程序“提供者”的在线视图中，可以在 PLC 功能块 FB_Calculation 中看到生成的 C++ 对象 ID。项目节点“TcCOM Objects”会将生成的对象及其对象 ID 和所选名称保存在其列表中。



⇨ 在 PLC 应用程序“调用者”的在线视图中，代理功能块已通过过程映像分配相同的对象 ID。接口指针的值有效，便可执行方法。



15.25.2 TcCOM_Sample02_PlcToCpp

本示例介绍 PLC 与 C++ 之间的 TcCOM 通信。在这种连接中，PLC 应用程序使用现有 TwinCAT C++ 类实例的功能。这样，用 C++ 写入的算法便可在 PLC 中轻松使用。
虽然在使用现有 TwinCAT C++ 模块的情况下，需要在目标系统上获得 TwinCAT C++ 授权，但在目标系统或开发计算机上不需要 C++ 开发环境。

已构建的 C++ 模块可提供一个或多个类，这些类的接口已存入 TMC 说明文件，因此在 PLC 中是已知的。

以下子章节将对该程序进行说明：

1. 将 TwinCAT C++ 类实例化为 TwinCAT TcCOM Object [▶ 312]
2. 在 PLC 中创建 FB，作为一个简单的封装器，提供 C++ 对象的功能 [▶ 313]
3. 执行示例项目 [▶ 315]

下载示例：https://infosys.beckhoff.com/content/1033/TC3_C/Resources/2343048971.zip

系统要求

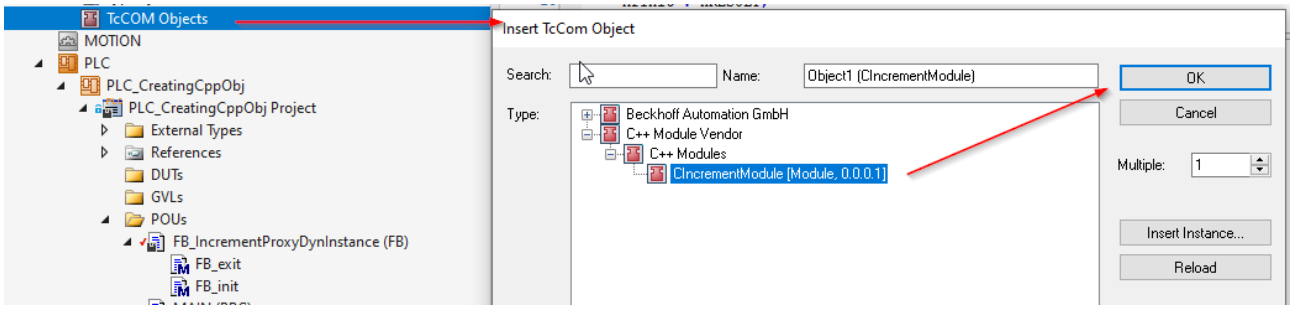
TwinCAT 版本	硬件	要集成的库
TwinCAT 3.1, Build 4020	x86, x64	Tc3_Module

15.25.2.1 将 TwinCAT++ 类实例化为 TwinCAT TcCOM 对象

工程系统中必须有二进制 TwinCAT C++ 项目，以便在激活过程中与 PLC 项目一起传输到目标系统。

TwinCAT 可为工程系统之间的分配提供部署机构，在版本控制的 C++ 项目 [▶ 48]中对其进行了介绍。
（本示例在下载时包含相应的 TMX，因为如果使用“类工厂”，TwinCAT 则会自动将其置于存档中。）

1. 打开一个 TwinCAT 项目或创建一个新项目。
2. 在 **TcCOM Objects** 节点下的解决方案中添加 CIncrementModule 类的实例。

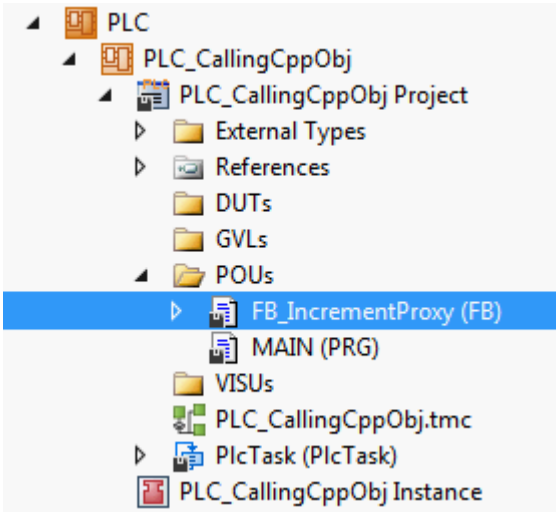


创建 C++ 模块

在 TwinCAT C++ 文档 [▶ 11]中，包含关于如何创建 TwinCAT C++ 模块的详细说明。

15.25.2.2 在 PLC 中创建 FB，作为简单的代理，提供 C++ 对象的功能

1. 创建 PLC 并在其中附加新的功能块。
   该代理功能块应提供在 C++ 中编程的功能。可以通过 C++ 类中定义的接口指针来实现此功能，而 PLC 中的 TMC 说明文件也会显示该接口指针。



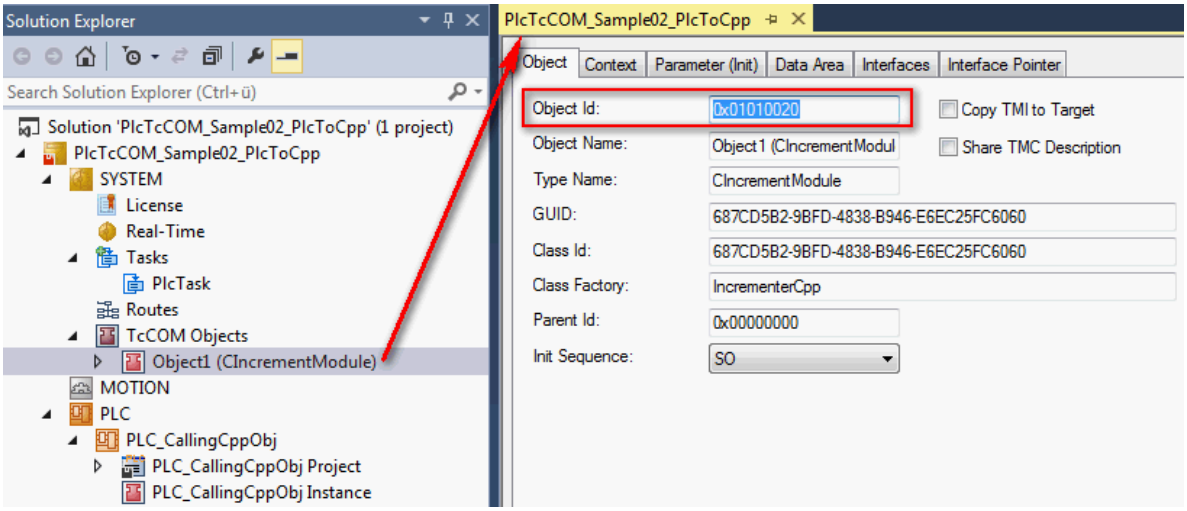
2. 在功能块的声明部分，声明一个接口指针作为输出，该指针指向随后向外提供功能的接口。
3. 将对象 ID 和接口 ID 创建为本地成员变量。
   虽然接口 ID 已通过全局列表提供，但对象 ID 是通过 TwinCAT Symbol Initialization分配的。
   TcInitSymbol 属性可确保变量出现在外部Symbol Initialization的列表中。应分配创建的 C++ 对象的对象 ID。

FB_IncrementProxy

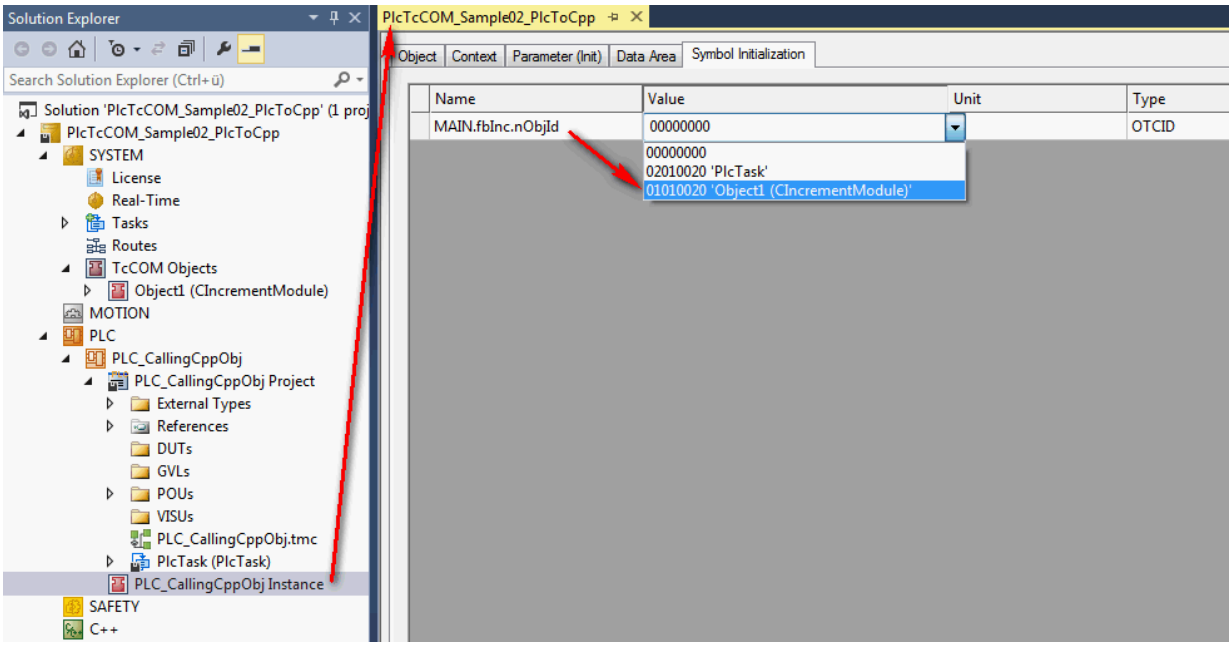
```
1  FUNCTION_BLOCK FB_IncrementProxy
2  VAR_OUTPUT
3      ip : IIncrement;
4  END_VAR
5
6  VAR
7      {attribute 'TcInitSymbol'}
8      {attribute 'displaymode':='hex'}
9      nObjId : OTCID;     // Instance configured to be retrieved
10     iid : IID := TC_GLOBAL_IID_LIST.IID_IIncrement;
11     hrInit : HRESULT;
12 END_VAR
13
```

```
1
```

⇨ 在选择 **TcCOM Objects** 节点下的对象时，会显示对象 ID。
如果使用的是 TcInitSymbol 属性，则Symbol Initialization列表位于**Symbol Initialization**选项卡的 PLC 实例节点中。



4. 在此，通过下拉方式将现有对象 ID 分配给变量的符号名称。该值会在下载 PLC 时分配，因此可在 PLC 运行前定义。随着 PLC 的新下载版本，相应地输入新的Symbol Initialization或更改。



作为替代方案，也可以通过过程映像链接来实现对象 ID 的传递，如第一个示例 (TcCOM_Sample01_PlcToPlc [▶ 302])。

5. 执行 PLC 代理功能块。
首先在功能块中添加 FB_init 构造函数方法。如果不再提供 OnlineChange，而是功能块的初始化，则借助函数 FW_ObjMgr_GetObjectInstance() 获取指向指定 TcCOM 对象的指定接口的接口指针。在这种连接中，对象本身会递增引用计数器。

FB_IncrementProxy.FB_init

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

```
IF NOT bInCopyCode THEN // if not online change
    IF nObjID <> 0 THEN
        hrInit := FW_ObjMgr_GetObjectInstance(oid:=nObjID, iid:=iid, pipUnk:=ADR(ip));
    ELSE
        hrInit := E_HRESULTAdsErr.INVALIDOBJID;
    END_IF
END_IF
```

6. 必须再次释放使用过的引用。为此，请在功能块的 FB_exit 析构函数中调用 FW_SafeRelease() 功能。

FB_IncrementProxy.FB_exit | FB_IncrementProxy.FB_init

```
{attribute 'hide'}
METHOD FB_exit : BOOL
VAR_INPUT
    bInCopyCode : BOOL; // if TRUE, the exit method is called for
END_VAR
```

```
IF NOT bInCopyCode THEN // if not online change
    FW_SafeRelease(ADR(ip));
END_IF
```

⇨ 至此，代理功能块的执行工作完成。

7. 声明代理功能块的实例，以调用应用程序中通过接口提供的方法。
调用本身全部通过定义为功能块输出的接口指针执行。对于指针来说，必须事先进行空值检查。然后就可以直接调用这些方法，也可以通过 Intellisense 调用。

MAIN*

```
PROGRAM MAIN
VAR
    fbInc : FB_IncrementProxy;
    nValue : UDINT;
END_VAR
```

```
IF fbInc.ip <> 0 THEN
    fbInc.ip.doIncrement(4, ADR(nValue));
END_IF
```

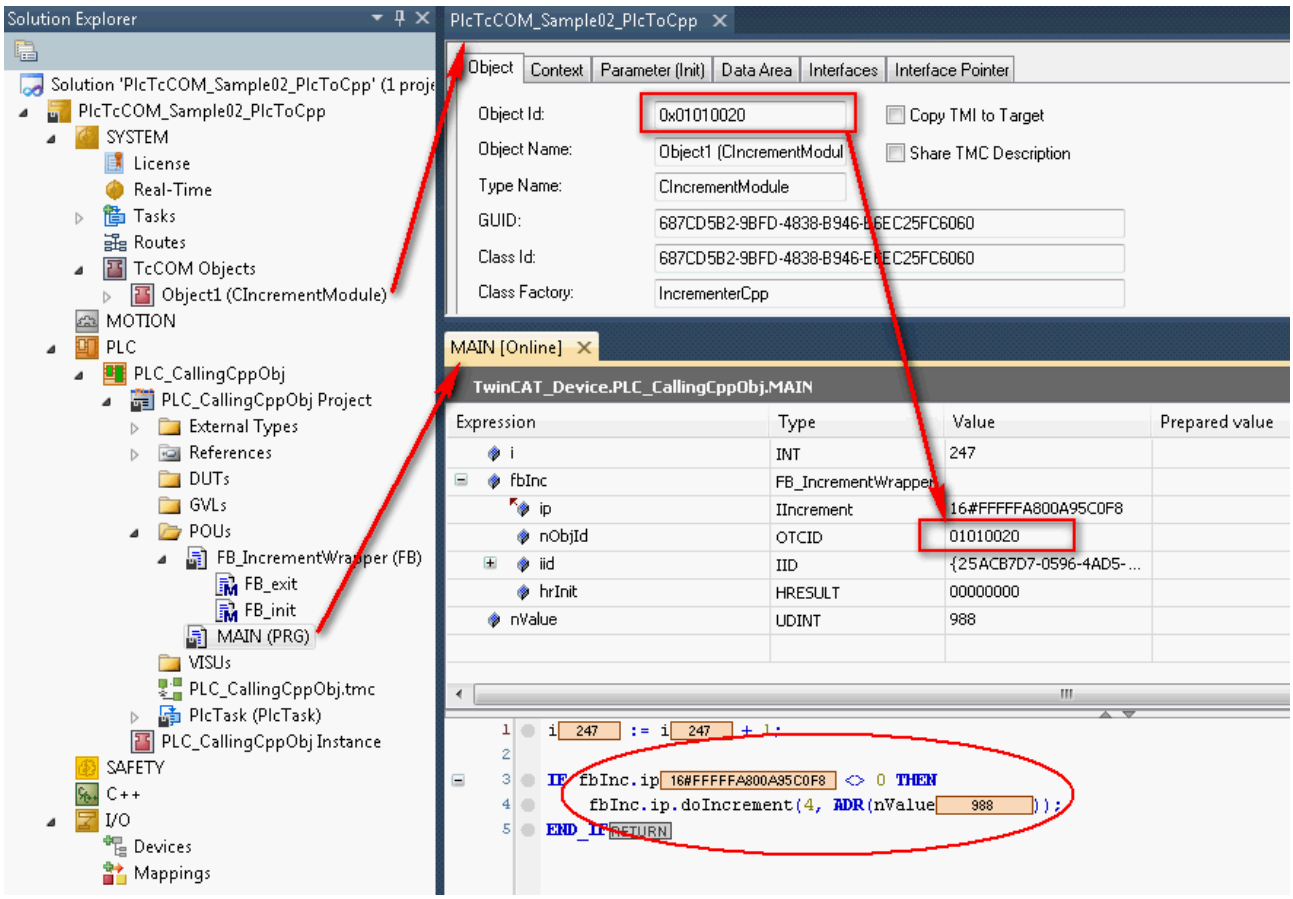
⇨ 示例已准备好进行测试。

15.25.2.3 执行示例项目

1. 选择目标系统并编译项目。
2. 启用 TwinCAT 配置，然后登录并启动 PLC。

3. 此 C++ TcCOM 模块需要使用 Windows 测试模式。激活后即可使用该模块，从而使用该示例。
4. 首次激活 TwinCAT 配置时，必须接受测试证书：

⇨ 在 PLC 应用程序的在线视图中，PLC 代理功能块会显示 C++ 对象的指定对象 ID。接口指针的值有效，便可执行方法。



15.25.3 TcCOM_Sample03_PlcCreatesCpp

与 Sample02 一样，该示例介绍了 PLC 与 C++ 之间的 TcCOM 通信。为此，PLC 应用程序使用 TwinCAT C++ 类的功能。在本示例中，所需 C++ 类实例由 PLC 自行创建。这样，用 C++ 写入的算法便可在 PLC 中轻松使用。
虽然在使用现有 TwinCAT C++ 驱动程序的情况下，需要在目标系统上获得 TwinCAT C++ 授权，但在目标系统或开发计算机上不需要 C++ 开发环境。

已构建的 C++ 驱动程序提供一个或多个类，这些类的接口已存入 TMC 说明文件，因此 PLC 已知晓这些接口。

以下子章节将对该程序进行说明：

1. 提供 TwinCAT C++ 驱动程序及其类 [▶ 317]

2. 在 PLC 中创建 FB，用于创建 C++ 对象并提供其功能 [▶ 318]

3. 执行示例项目 [▶ 319]

下载示例：https://infosys.beckhoff.com/content/1033/TC3_C/Resources/2343051531.zip

系统要求

TwinCAT 版本	硬件	要集成的库
TwinCAT 3.1, Build 4020	x86, x64	Tc3_Module

15.25.3.1 提供二进制 C++ 项目 (TMX) 及其类

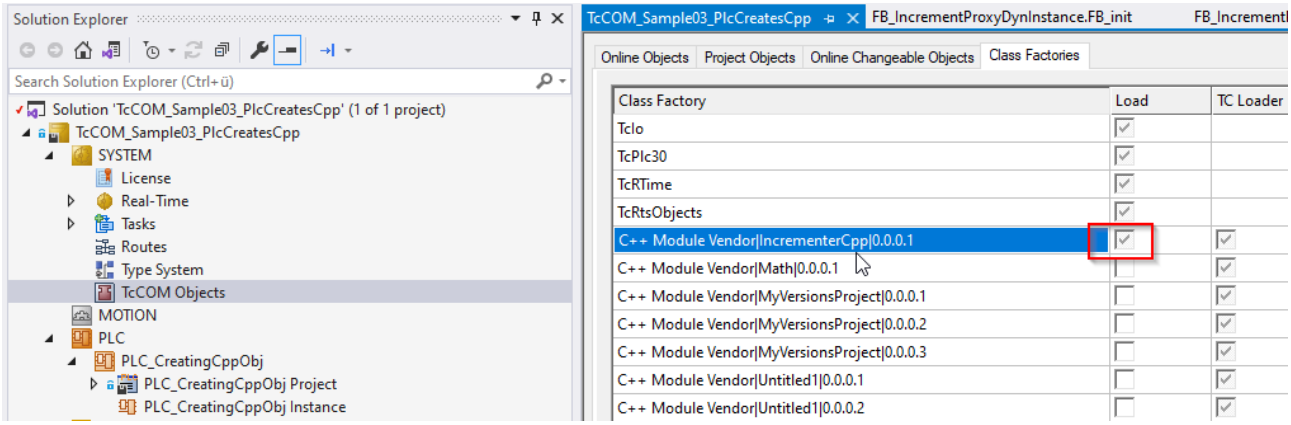
工程系统中必须有二进制 TwinCAT C++ 项目，以便在激活过程中与 PLC 项目一起传输到目标系统。

TwinCAT 提供一种在工程系统之间进行分配的部署机构，详情请访问 版本控制的 C++ 项目 [▶ 48]。
（本示例在下载时包含相应的 TMX，因为如果使用“类工厂”，TwinCAT 则会自动将其置于存档中）。

打开一个 TwinCAT 项目或创建一个新项目。

1. 在**类工厂**选项卡的 **TcCOM 对象**节点下的“解决方案”中选择所需 C++ 驱动程序。如果已相应实现这一功能（如此处示例所示），则复选框也可由 TwinCAT 自动设置。

⇨ 这样可以确保在启动 TwinCAT 时在目标系统上传输和加载驱动程序。

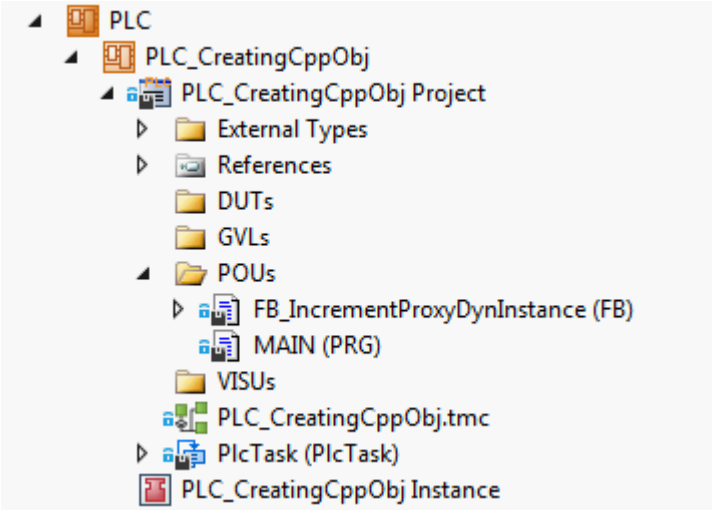


创建二进制 C++ 项目

TwinCAT C++ 的文档 [▶ 11]详细说明了如何创建 TMX。
对于 Sample03，需要注意的是，要对其类进行动态实例化的 TwinCAT C++ 模块必须定义为“用于 RT 环境的 TwinCAT 模块类”。C++ 向导为此提供一个特殊模板。
此外，该示例使用的 TwinCAT C++ 类没有 TcCOM 初始化数据，也没有 TcCOM 参数。

15.25.3.2 在 PLC 中创建 FB，用于创建 C++ 对象并提供其功能

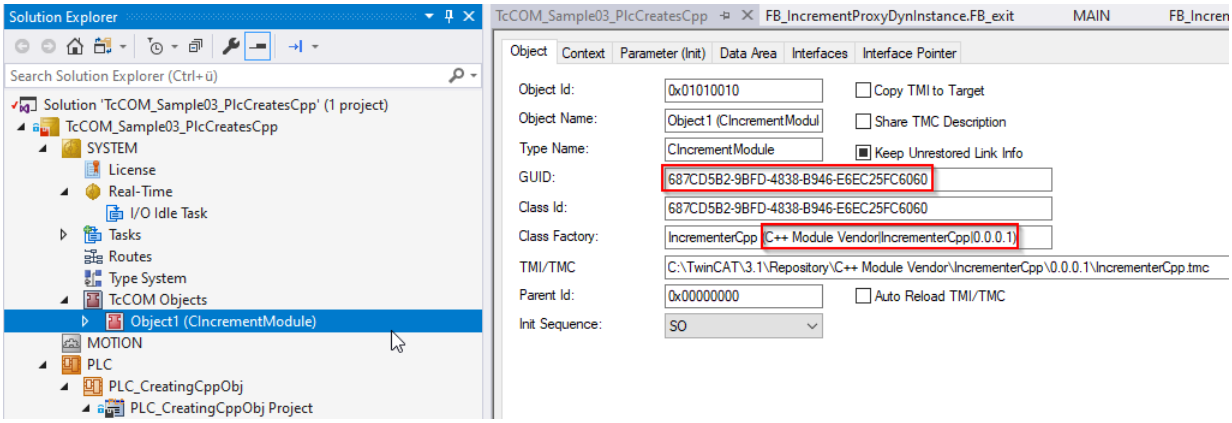
1. 创建 PLC 并在其中附加新的功能块。
   该代理功能块应提供在 C++ 中编程的功能。通过接口指针进行管理，该指针由 C++ 定义，PLC 通过 TMC 说明文件了解该指针。



2. 在功能块的声明部分，声明一个接口指针作为输出，该指针指向随后向外提供功能的接口 (IIncrement)。

```
{attribute 'TcDependOnClassFactory' := 'C++ Module Vendor|IncrementerCpp|0.0.0.1'}
FUNCTION_BLOCK FB_IncrementProxyDynInstance
VAR_OUTPUT
    ip : IIncrement;
END_VAR

VAR
    objName : STRING;
    classId : CLSID := STRING_TO_GUID('687cd5b2-9bfd-4838-b946-e6ec25fc6060');
    sLibraryId : STRING := 'C++ Module Vendor|IncrementerCpp|0.0.0.1';
    classIdVersioned : CLSID;
    iid : IID := TC_GLOBAL_IID_LIST.IID_IIncrement;
    hrInit : HRESULT;
END_VAR

```

3. 如上一步所示，创建库 ID、类 ID 和接口 ID 作为成员变量。
   接口 ID 可以通过全局列表获得，而库 ID 和类 ID（若尚未知）则需要通过其他方式确定。一种可行的方法是临时创建一个实例，并在再次删除之前从对话框中获取信息：



4. 向 PLC 代理功能块添加 FB_init 构造函数方法。
   如果不是在线更改，而是初始化功能块，则需创建一个新的 TcCOM 对象（指定类的类实例），并获取指定接口的接口指针。

首先，使用 F_GetClassIdVersioned() 方法从库 ID 和类 ID 计算版本控制的类 ID。然后，所用 FW_ObjMgr_CreateAndInitInstance() 功能也会给出 TcCOM 对象的名称和目标状态。这两个参数在此声明为 FB_init 方法的输入参数，因此在代理功能块的实例化过程中需要指定这两个参数。要进行实例化的 TwinCAT C++ 类不需要 TcCOM 初始化数据，也不需要 TcCOM 参数。
通过该功能调用，对象本身会计算引用计数器。

FB_IncrementProxyDynInstance.FB_init | FB_IncrementProxyDynInstance

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

    sObjName : STRING;      // object name to be set for this instance (optional)
    eObjState : TCOM_STATE; // target object state (usually TCOM_STATE.TCOM_STATE_OP)
END_VAR
```

```
IF NOT bInCopyCode THEN // if not online change
    objName := sObjName;
    F_GetClassIdVersioned(sLibraryId:=sLibraryId, clsId:=classId, clsIdVersioned:=classIdVersioned);
    hrInit := FW_ObjMgr_CreateAndInitInstance(  clsId       := classIdVersioned,
                                                iid         := iid,
                                                pipUnk      := ADR(ip),
                                                objId       := OTCID_CreateNewId,
                                                parentId    := TwinCAT_SystemInfoVarList._AppInfo.ObjId, // set PLC instance as parent
                                                name        := sObjName,
                                                state       := eObjState,
                                                pInitData   := 0 );
END_IF
```

5. 如果不再使用该对象，就必须再次释放已使用的引用并将其删除。为此，请在功能块的 FB_exit 析构函数中调用 FW_ObjMgr_DeleteInstance() 功能。

FB_IncrementProxyDynInstance.FB_exit | FB_IncrementProxyDynInstance.FB_init | FB_IncrementProxyDynInstance

```
{attribute 'hide'}
METHOD FB_exit : BOOL
VAR_INPUT
    bInCopyCode : BOOL; // if TRUE, the exit method is called for exiting an instance that is copied afterwards (online change).
END_VAR
```

```
IF NOT bInCopyCode THEN // if not online change
    FW_ObjMgr_DeleteInstance(ADR(ip));
END_IF
```

⇨ 至此，代理功能块的执行工作完成。

6. 声明一个代理功能块实例，以调用应用程序中通过接口提供的方法。调用本身全部通过定义为功能块输出的接口指针执行。对于指针来说，必须事先进行空值检查。然后就可以直接调用这些方法，也可以通过 Intellisense 调用。

MAIN | FB_IncrementProxyDynInstance.FB_exit | FB_IncrementProxyDynInstance.FB_init | FB_IncrementProxyDynInstance

```
PROGRAM MAIN
VAR
    fbInc : FB_IncrementProxyDynInstance(   sObjName:='CIncrementModule:fbInc',
                                            eObjState:=TCOM_STATE.TCOM_STATE_OP);
    nValue : UDINT;
END_VAR
```

```
IF fbInc.ip <> 0 THEN
    fbInc.ip.doIncrement(100, ADR(nValue));
END_IF
```

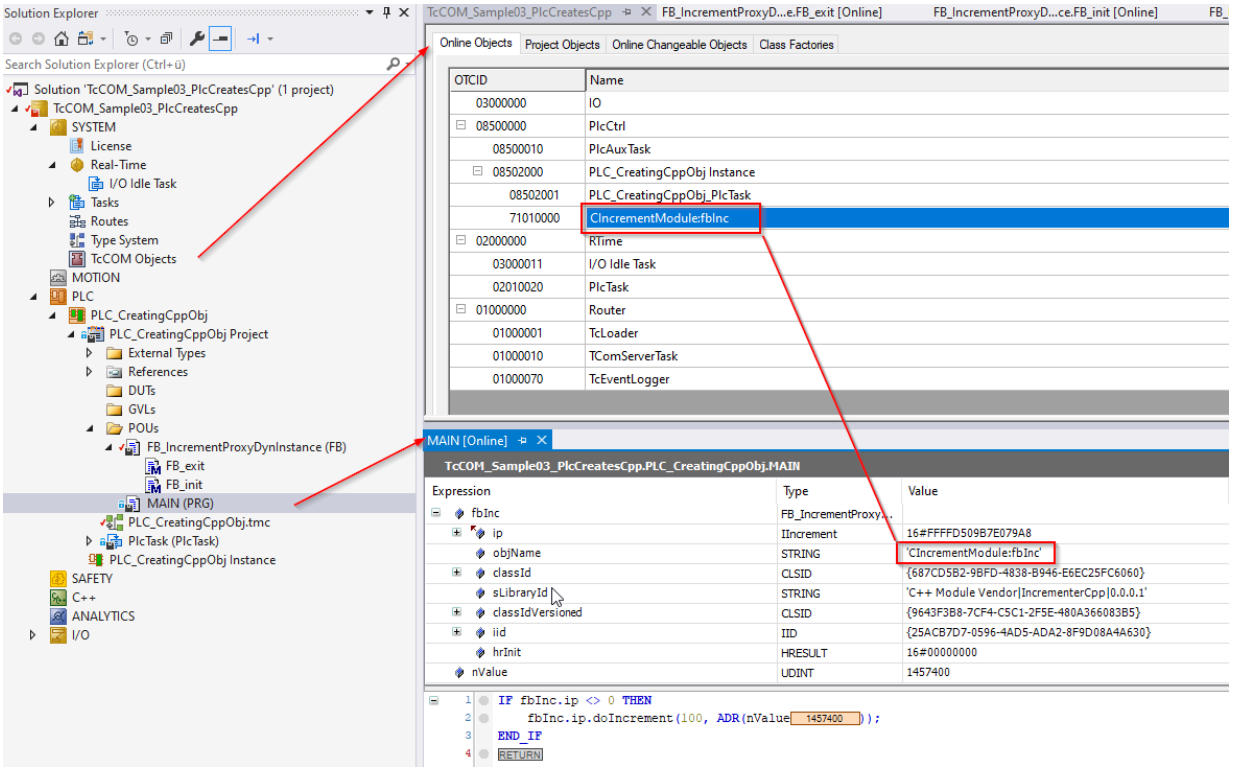
⇨ 示例已准备好进行测试。

15.25.3.3 执行示例项目

1. 选择目标系统并编译项目。
2. 启用 TwinCAT 配置，然后登录并启动 PLC。
3. 此 C++ TcCOM 模块需要使用 Windows 测试模式。激活后即可使用该模块，从而使用该示例。

4. 首次激活 TwinCAT 配置时，必须接受测试证书：

⇨ 在 PLC 应用程序的在线视图中，所需的 TcCOM 对象名称显示在 PLC 代理功能块中。项目节点 **TcCOM 对象**包含生成的对象，其列表中包含生成的 ID 和所需名称。接口指针的值有效，便可执行方法。



1 附录

6

- ADS 返回代码 [▶ 321]在整个 TwinCAT 3 中都很重要，尤其是在实现 ADS 通信 [▶ 188]的情况下。
- （在 NOVRAM 存储器中）保留型数据 [▶ 325]可通过 PLC 和 C++ 以类似方式使用。
- 除了 TcCOM 模块 [▶ 35]概念之外，TwinCAT 3 类型系统也是理解的重要基础。
- 以下页面源自自动化接口的文档。
  使用自动化接口时，请参阅专用文档。
  - 创建和处理 C++ 项目和模块 [▶ 328]
  - 创建和处理 TcCOM 模块 [▶ 331]

16.1 ADS 返回代码

错误代码分组：
全局错误代码：0x0000 [▶ 321]... （0x9811_0000 ...）
路由器错误代码：0x0500 [▶ 322]... （0x9811_0500 ...）
一般 ADS 错误：0x0700 [▶ 322]... （0x9811_0700 ...）
RTime 错误代码：0x1000 [▶ 324]... （0x9811_1000 ...）

全局错误代码

Hex	Dec	HRESULT	名称	描述
0x0	0	0x98110000	ERR_NOERROR	无错误。
0x1	1	0x98110001	ERR_INTERNAL	内部错误。
0x2	2	0x98110002	ERR_NORTIME	不具有实时性。
0x3	3	0x98110003	ERR_ALLOCLOCKEDMEM	分配已锁定 – 内存错误。
0x4	4	0x98110004	ERR_INSERTMAILBOX	邮箱已满 – 无法发送 ADS 消息。减少每个周期的 ADS 消息数量将有所帮助。
0x5	5	0x98110005	ERR_WRONGRECEIVEHMSG	HMSG 错误。
0x6	6	0x98110006	ERR_TARGETPORTNOTFOUND	未找到目标端口 – ADS 服务器未启动或无法访问。
0x7	7	0x98110007	ERR_TARGETMACHINENOTFOUND	未找到目标计算机 – 未找到 AMS 路由。
0x8	8	0x98110008	ERR_UNKNOWNCMDID	未知命令 ID。
0x9	9	0x98110009	ERR_BADTASKID	任务 ID 无效。
0xA	10	0x9811000A	ERR_NOIO	No IO。
0xB	11	0x9811000B	ERR_UNKNOWNAMSCMD	未知 AMS 命令。
0xC	12	0x9811000C	ERR_WIN32ERROR	Win32 错误。
0xD	13	0x9811000D	ERR_PORTNOTCONNECTED	端口未连接。
0xE	14	0x9811000E	ERR_INVALIDAMSLENGTH	AMS 长度无效。
0xF	15	0x9811000F	ERR_INVALIDAMSNETID	AMS Net ID 无效。
0x10	16	0x98110010	ERR_LOWINSTLEVEL	安装级别 – TwinCAT 2 授权错误。
0x11	17	0x98110011	ERR_NODEBUGINTAVAILABLE	无调试可用。
0x12	18	0x98110012	ERR_PORTDISABLED	端口已禁用 – TwinCAT 系统服务未启动。
0x13	19	0x98110013	ERR_PORTALREADYCONNECTED	端口已连接。
0x14	20	0x98110014	ERR_AMSSYNC_W32ERROR	AMS Sync Win32 错误。
0x15	21	0x98110015	ERR_AMSSYNC_TIMEOUT	AMS Sync 超时。
0x16	22	0x98110016	ERR_AMSSYNC_AMSERROR	AMS Sync 错误。
0x17	23	0x98110017	ERR_AMSSYNC_NOINDEXINMAP	不存在适用于 AMS Sync 的索引映射。
0x18	24	0x98110018	ERR_INVALIDAMSPORT	AMS 端口无效。
0x19	25	0x98110019	ERR_NOMEMORY	无内存。
0x1A	26	0x9811001A	ERR_TCPSEND	TCP 发送错误。
0x1B	27	0x9811001B	ERR_HOSTUNREACHABLE	主机无法访问。
0x1C	28	0x9811001C	ERR_INVALIDAMSFRAGMENT	AMS 片段无效。
0x1D	29	0x9811001D	ERR_TLSSEND	TLS 发送错误 – ADS 安全连接失败。
0x1E	30	0x9811001E	ERR_ACCESSDENIED	拒绝访问 – 拒绝 ADS 安全访问。

路由器错误代码

Hex	Dec	HRESULT	名称	描述
0x500	1280	0x98110500	ROUTERERR_NOLOCKEDMEMORY	无法分配锁定的内存。
0x501	1281	0x98110501	ROUTERERR_RESIZEMEMORY	路由器内存大小无法更改。
0x502	1282	0x98110502	ROUTERERR_MAILBOXFULL	邮箱已达到最大消息数。
0x503	1283	0x98110503	ROUTERERR_DEBUGBOXFULL	调试邮箱已达到最大消息数。
0x504	1284	0x98110504	ROUTERERR_UNKNOWNPORTTYPE	端口类型未知。
0x505	1285	0x98110505	ROUTERERR_NOTINITIALIZED	路由器未初始化。
0x506	1286	0x98110506	ROUTERERR_PORTALREADYINUSE	端口号已分配。
0x507	1287	0x98110507	ROUTERERR_NOTREGISTERED	端口未注册。
0x508	1288	0x98110508	ROUTERERR_NOMOREQUEUES	已达到最大端口数。
0x509	1289	0x98110509	ROUTERERR_INVALIDPORT	端口无效。
0x50A	1290	0x9811050A	ROUTERERR_NOTACTIVATED	路由未激活。
0x50B	1291	0x9811050B	ROUTERERR_FRAGMENTBOXFULL	邮箱已达到最大片段消息数。
0x50C	1292	0x9811050C	ROUTERERR_FRAGMENTTIMEOUT	发生片段超时。
0x50D	1293	0x9811050D	ROUTERERR_TOBEREMOVED	端口已移除。

一般性 ADS 错误代码

Hex	Dec	HRESULT	名称	描述
0x700	1792	0x98110700	ADSERR_DEVICE_ERROR	一般性设备错误。
0x701	1793	0x98110701	ADSERR_DEVICE_SRVNOTSUPP	服务器不支持该服务。
0x702	1794	0x98110702	ADSERR_DEVICE_INVALIDGRP	索引组无效。
0x703	1795	0x98110703	ADSERR_DEVICE_INVALIDOFFSET	索引偏移量无效。
0x704	1796	0x98110704	ADSERR_DEVICE_INVALIDACCESS	不允许读取或写入。 可能有几种原因。例如，创建路由时输入了错误的密码。
0x705	1797	0x98110705	ADSERR_DEVICE_INVALIDSIZE	参数大小不正确。
0x706	1798	0x98110706	ADSERR_DEVICE_INVALIDDATA	无效数据值。
0x707	1799	0x98110707	ADSERR_DEVICE_NOTREADY	设备尚未准备好运行。
0x708	1800	0x98110708	ADSERR_DEVICE_BUSY	设备正忙。
0x709	1801	0x98110709	ADSERR_DEVICE_INVALIDCONTEXT	操作系统上下文无效。这可能是由于在不同的任务中使用 ADS 函数块造成的。可以通过在 PLC 中进行多任务同步解决这个问题。
0x70A	1802	0x9811070A	ADSERR_DEVICE_NOMEMORY	内存不足。
0x70B	1803	0x9811070B	ADSERR_DEVICE_INVALIDPARM	参数值无效。
0x70C	1804	0x9811070C	ADSERR_DEVICE_NOTFOUND	未找到（文件…）。
0x70D	1805	0x9811070D	ADSERR_DEVICE_SYNTAX	文件或命令中存在语法错误。
0x70E	1806	0x9811070E	ADSERR_DEVICE_INCOMPATIBLE	对象不匹配。
0x70F	1807	0x9811070F	ADSERR_DEVICE_EXISTS	对象已存在。
0x710	1808	0x98110710	ADSERR_DEVICE_SYMBOLNOTFOUND	符号未找到。
0x711	1809	0x98110711	ADSERR_DEVICE_SYMBOLVERSIONINVALID	符号版本无效。这可能是由于联机更改造成的。创建新句柄。
0x712	1810	0x98110712	ADSERR_DEVICE_INVALIDSTATE	设备（服务器）处于无效状态。
0x713	1811	0x98110713	ADSERR_DEVICE_TRANSMODENOTSUPP	不支持 AdsTransMode。
0x714	1812	0x98110714	ADSERR_DEVICE_NOTIFYHNDINVALID	通知句柄无效。
0x715	1813	0x98110715	ADSERR_DEVICE_CLIENTUNKNOWN	通知客户端未注册。
0x716	1814	0x98110716	ADSERR_DEVICE_NOMOREHDLS	没有更多的句柄可用。
0x717	1815	0x98110717	ADSERR_DEVICE_INVALIDWATCHSIZE	通知大小过大。
0x718	1816	0x98110718	ADSERR_DEVICE_NOTINIT	设备未初始化。
0x719	1817	0x98110719	ADSERR_DEVICE_TIMEOUT	设备超时。
0x71A	1818	0x9811071A	ADSERR_DEVICE_NOINTERFACE	接口查询失败。
0x71B	1819	0x9811071B	ADSERR_DEVICE_INVALIDINTERFACE	请求的接口错误。
0x71C	1820	0x9811071C	ADSERR_DEVICE_INVALIDCLSID	Class ID 无效。
0x71D	1821	0x9811071D	ADSERR_DEVICE_INVALIDOBJID	Object ID 无效。
0x71E	1822	0x9811071E	ADSERR_DEVICE_PENDING	请求待定。
0x71F	1823	0x9811071F	ADSERR_DEVICE_ABORTED	请求已中止。
0x720	1824	0x98110720	ADSERR_DEVICE_WARNING	警告信号。
0x721	1825	0x98110721	ADSERR_DEVICE_INVALIDARRAYIDX	数组索引无效。
0x722	1826	0x98110722	ADSERR_DEVICE_SYMBOLNOTACTIVE	符号未激活。
0x723	1827	0x98110723	ADSERR_DEVICE_ACCESSDENIED	拒绝访问。 有几种可能的原因。例如，单向 ADS 路由用于相反方向。
0x724	1828	0x98110724	ADSERR_DEVICE_LICENSENOTFOUND	缺少授权。
0x725	1829	0x98110725	ADSERR_DEVICE_LICENSEEXPIRED	授权已过期。
0x726	1830	0x98110726	ADSERR_DEVICE_LICENSEEXCEEDED	超出授权。
0x727	1831	0x98110727	ADSERR_DEVICE_LICENSEINVALID	授权无效。
0x728	1832	0x98110728	ADSERR_DEVICE_LICENSESYSTEMID	授权问题：系统 ID 无效。
0x729	1833	0x98110729	ADSERR_DEVICE_LICENSENOTIMELIMIT	授权不受时间限制。
0x72A	1834	0x9811072A	ADSERR_DEVICE_LICENSEFUTUREISSUE	授权问题：未来的时间。
0x72B	1835	0x9811072B	ADSERR_DEVICE_LICENSETIMETOLONG	授权期限太长。
0x72C	1836	0x9811072C	ADSERR_DEVICE_EXCEPTION	系统启动异常。
0x72D	1837	0x9811072D	ADSERR_DEVICE_LICENSEDUPLICATED	授权文件读取了两次。
0x72E	1838	0x9811072E	ADSERR_DEVICE_SIGNATUREINVALID	签名无效。
0x72F	1839	0x9811072F	ADSERR_DEVICE_CERTIFICATEINVALID	证书无效。
0x730	1840	0x98110730	ADSERR_DEVICE_LICENSEOEMNOTFOUND	OEM未知公钥。
0x731	1841	0x98110731	ADSERR_DEVICE_LICENSERESTRICTED	此系统 ID 授权无效。

Hex	Dec	HRESULT	名称	描述
0x732	1842	0x98110732	ADSERR_DEVICE_LICENSEDEMODENIED	演示授权已禁止。
0x733	1843	0x98110733	ADSERR_DEVICE_INVALIDFNCID	无效函数 ID。
0x734	1844	0x98110734	ADSERR_DEVICE_OUTOFRANGE	超出有效范围。
0x735	1845	0x98110735	ADSERR_DEVICE_INVALIDALIGNMENT	无效对齐。
0x736	1846	0x98110736	ADSERR_DEVICE_LICENSEPLATFORM	无效平台级别。
0x737	1847	0x98110737	ADSERR_DEVICE_FORWARD_PL	上下文 – 转到被动级别。
0x738	1848	0x98110738	ADSERR_DEVICE_FORWARD_DL	上下文 – 转到调度级别。
0x739	1849	0x98110739	ADSERR_DEVICE_FORWARD_RT	上下文 – 转到实时。
0x740	1856	0x98110740	ADSERR_CLIENT_ERROR	客户端错误。
0x741	1857	0x98110741	ADSERR_CLIENT_INVALIDPARM	服务包含无效参数。
0x742	1858	0x98110742	ADSERR_CLIENT_LISTEMPTY	轮询列表为空。
0x743	1859	0x98110743	ADSERR_CLIENT_VARUSED	Var 连接已在使用中。
0x744	1860	0x98110744	ADSERR_CLIENT_DUPLINVOKEID	调用的 ID 已在使用中。
0x745	1861	0x98110745	ADSERR_CLIENT_SYNCTIMEOUT	已发生超时 – 远程终端在指定的 ADS 超时中未响应。远程终端的路由设置可能配置不正确。
0x746	1862	0x98110746	ADSERR_CLIENT_W32ERROR	Win32 子系统中发生错误。
0x747	1863	0x98110747	ADSERR_CLIENT_TIMEOUTINVALID	客户端超时值无效。
0x748	1864	0x98110748	ADSERR_CLIENT_PORTNOTOPEN	端口未打开。
0x749	1865	0x98110749	ADSERR_CLIENT_NOAMSADDR	无 AMS 地址。
0x750	1872	0x98110750	ADSERR_CLIENT_SYNCINTERNAL	Ads sync 中发生内部错误。
0x751	1873	0x98110751	ADSERR_CLIENT_ADDHASH	哈希表溢出。
0x752	1874	0x98110752	ADSERR_CLIENT_REMOVEHASH	未在表格中找到密钥。
0x753	1875	0x98110753	ADSERR_CLIENT_NOMORESYM	缓存中没有符号。
0x754	1876	0x98110754	ADSERR_CLIENT_SYNCRESINVALID	收到无效响应。
0x755	1877	0x98110755	ADSERR_CLIENT_SYNCPORTLOCKED	同步端口已锁定。
0x756	1878	0x98110756	ADSERR_CLIENT_REQUESTCANCELLED	请求被取消。

RTime 错误代码

Hex	Dec	HRESULT	名称	描述
0x1000	4096	0x98111000	RTERR_INTERNAL	实时系统中发生内部错误。
0x1001	4097	0x98111001	RTERR_BADTIMERPERIODS	计时器值无效。
0x1002	4098	0x98111002	RTERR_INVALIDTASKPTR	任务指针提供了无效值 0（零）。
0x1003	4099	0x98111003	RTERR_INVALIDSTACKPTR	堆栈指针提供了无效值 0（零）。
0x1004	4100	0x98111004	RTERR_PRIOEXISTS	请求任务优先级已分配。
0x1005	4101	0x98111005	RTERR_NOMORETCB	没有可用的空闲 TCB（任务控制块）。TCB 的最大数量为 64。
0x1006	4102	0x98111006	RTERR_NOMORESEMAS	没有可用的空闲信号。信号的最大数量为 64。
0x1007	4103	0x98111007	RTERR_NOMOREQUEUES	队列中没有可用的空闲空间。队列中的最大位置数为 64。
0x100D	4109	0x9811100D	RTERR_EXTIRQALREADYDEF	已应用外部同步中断。
0x100E	4110	0x9811100E	RTERR_EXTIRQNOTDEF	未应用外部同步中断。
0x100F	4111	0x9811100F	RTERR_EXTIRQINSTALLFAILED	外部同步中断应用失败。
0x1010	4112	0x98111010	RTERR_IRQLNOTLESSOREQUAL	在错误的上下文中调用服务函数
0x1017	4119	0x98111017	RTERR_VMXNOTSUPPORTED	不支持 Intel® VT-x 扩展。
0x1018	4120	0x98111018	RTERR_VMXDISABLED	BIOS 中未启用 Intel® VT-x 扩展。
0x1019	4121	0x98111019	RTERR_VMXCONTROLSMISSING	Intel® VT-x 扩展中缺少函数。
0x101A	4122	0x9811101A	RTERR_VMXENABLEFAILS	Intel® VT-x 激活失败。

具体的正向 HRESULT 返回代码：

HRESULT	名称	描述
0x0000_0000	S_OK	无错误。
0x0000_0001	S_FALSE	无错误。 示例：处理成功，但有一个负面或不完整的结果。
0x0000_0203	S_PENDING	无错误。 示例：处理成功，但还没有结果。
0x0000_0256	S_WATCHDOG_TIMEOUT	无错误。 示例：处理成功，但发生了超时。

TCP Winsock 错误代码

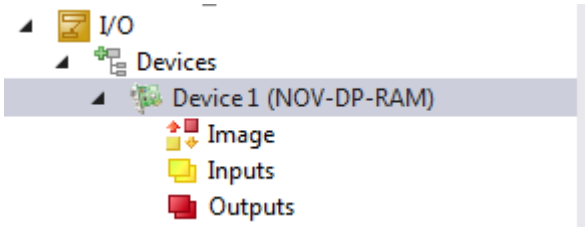
Hex	Dec	名称	描述
0x274C	10060	WSAETIMEDOUT	发生连接超时 - 在创建连接时发生错误，因为远程终端在特定时间后未正确响应，或者所建立连接因所连接主机未响应而无法维持。
0x274D	10061	WSAECONNREFUSED	连接遭到拒绝 - 无法建立连接，因为目标计算机明确拒绝了该连接。此错误通常由尝试连接到外部主机上处于非活动状态的服务（即没有运行服务器应用程序的服务）导致。
0x2751	10065	WSAEHOSTUNREACH	不存在至主机的路由 - 套接字操作引用了不可用的主机。
更多 Winsock 错误代码：Win32 错误代码			

16.2 保留型数据

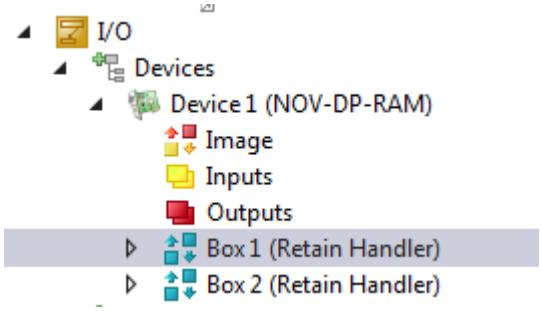
本节介绍即使在命令性或自发性重启系统后仍可使用数据的选项。设备的 NOV-RAM 可用于此目的。EL6080 不能用于这些保留型数据，因为必须首先传输相应的数据，从而导致相应的运行时。
下一部分将介绍保留型处理程序（用于存储数据并使其再次可用）以及不同 TwinCAT 3 编程语言的应用。

配置保留型设备

1. 保留型数据由保留型处理程序存储和提供，该处理程序是 TwinCAT 解决方案 IO 部分 NOV-DP-RAM 设备的一部分。在“解决方案”的 IO 区域创建 NOV-RAM DP 设备。

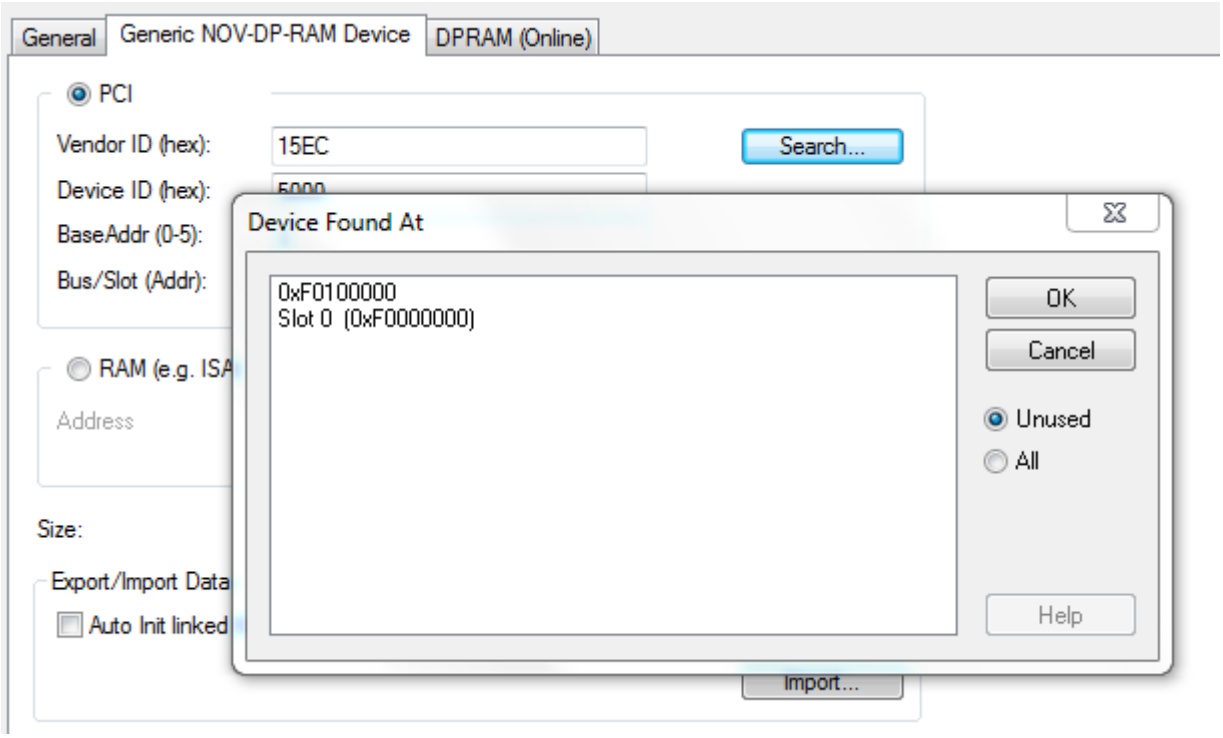


2. 在该设备下方创建一个或多个“保留处理程序”。



存储位置：NOVRAM

3. 配置 NOV-DP RAM 设备。在**通用型 NOV-DP-RAM 设备**选项卡中，使用**搜索.....**功能来确定要使用的区域。



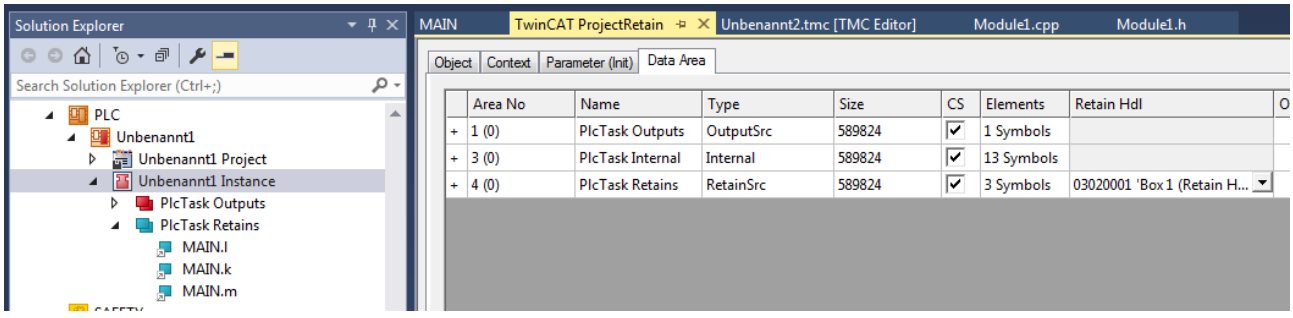
4. 在 TwinCAT 启动目录中为符号创建一个额外的保留目录。

在 PLC 项目中使用保留处理程序

在 PLC 项目中，这些变量要么在 VAR RETAIN 区域创建，要么用属性 TcRetain 标识。

```
PROGRAM MAIN
VAR RETAIN
  l: UINT;
  k: UINT;
END_VAR
VAR
  {attribute 'TcRetain':='1'}
  m: UINT;
  x: UINT;
END_VAR
```

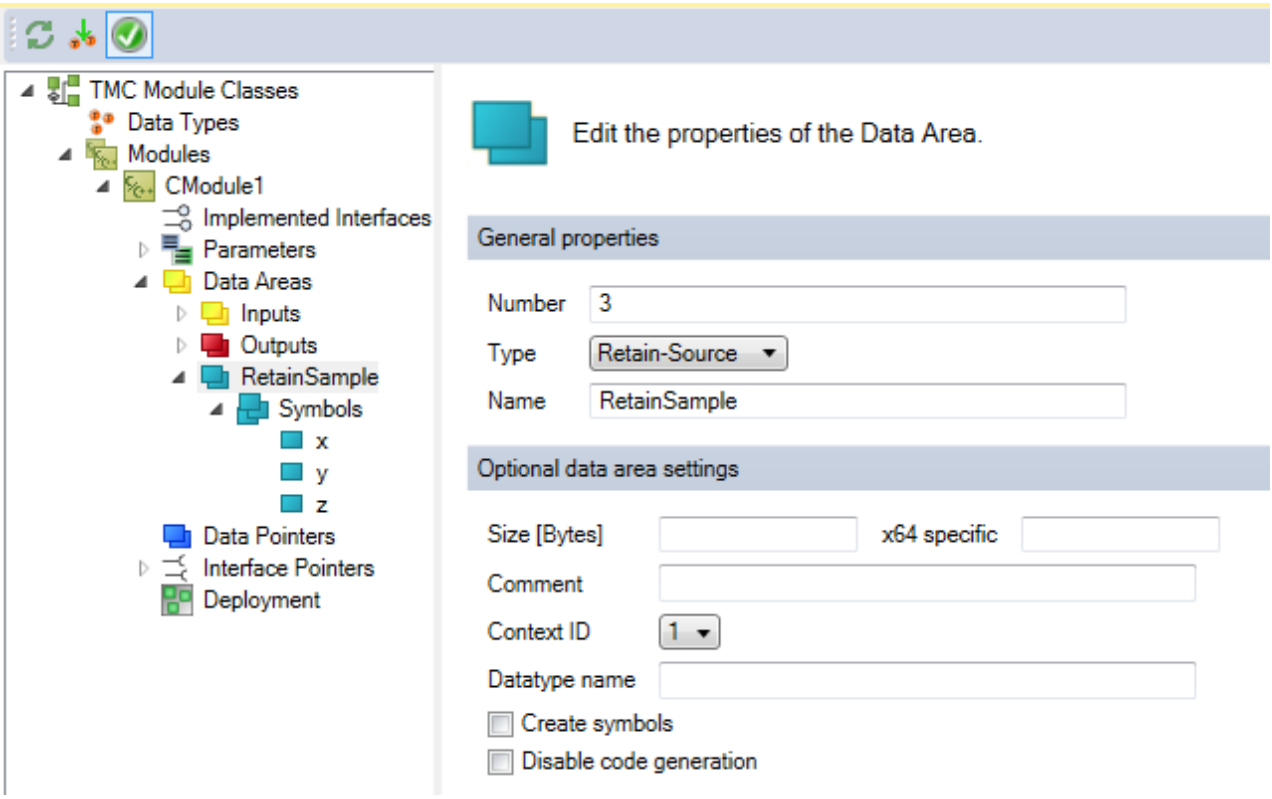
执行“构建”后会创建相应的符号。
对 NOV-DP-RAM 设备的保留型处理程序的分配任务在**Retain Hdl** 栏中完成。



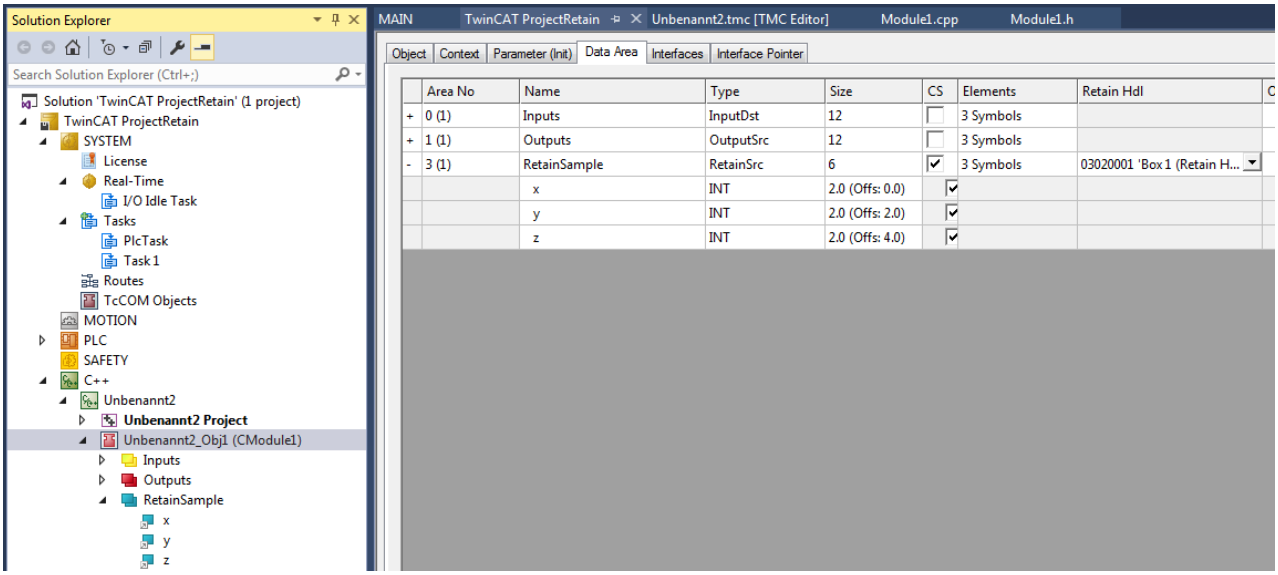
如果要将自定义的数据类型 (DUT) 用作保留型数据，则数据类型必须在 TwinCAT 类型系统中可用。您可以使用选项**转换为全局类型**，也可以直接创建 `STRUCT RETAIN` 结构。然而，此后保留数据处理程序会处理该结构的所有实例。
保留型数据不能用于整个 POU（功能块）。不过，可以使用 POU 的单个元素。

在 C++ 模块中使用保留型处理程序

在 C++ 模块中，会创建一个“保留源”类型的数据区，其中包含相应的符号。

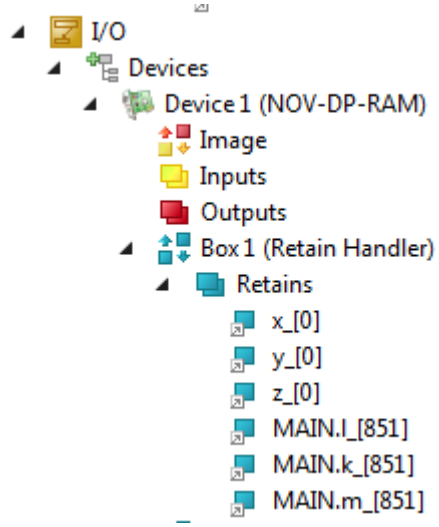


在 C++ 模块的实例中，**Retain Hdl** 栏会定义用于该数据区的 NOV-DP-RAM 设备的保留型处理程序。



结论

在相应项目中选择保留型处理程序作为目标时，执行“构建”后会自动创建保留型处理程序下的符号和一个映射。



16.3 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 [▶ 83]。

客户可以自行定义模板，请参见 C++ /向导的相应小节 [▶ 132]。

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，使用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)
```

要实例化驱动程序项目，请使用“TcVersionedDriverWizard”作为 `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`。

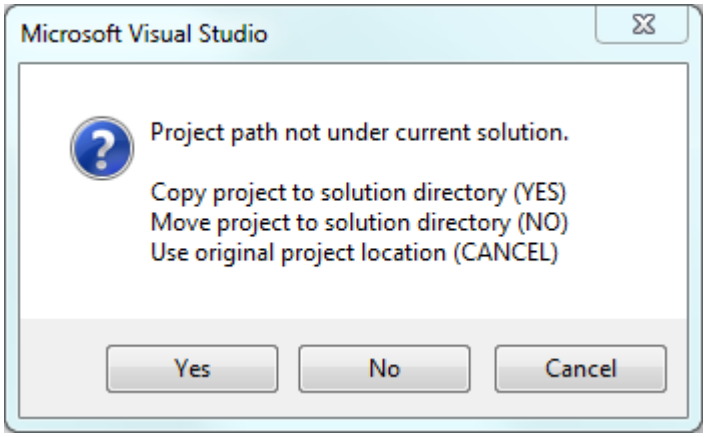
打开现有 C++ 项目

要通过自动化接口打开现有的 C++ 项目，必须导航至 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）：



您必须使用要添加的 C++ 项目（或其 vcxproj 文件）路径，而非模板文件。或者，也可以使用 C++ 项目存档文件（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，因此必须指定原始项目名。（请参见 重命名 TwinCAT C++ 项目 [▶ 214]）

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 [▶ 331]。

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

可以调用 TMC 代码生成器，根据 TMC 文件或 C++ 项目的变更生成 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

本次发布涵盖了所有平台的项目开发情况。按照 C++ 模块处理部分所述，编译后的模块可用于导出。

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。
这些选项可通过自动化接口进行设置。

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。

设置项目

要构建项目或解决方案，可以使用 Visual Studio API 的相应类和方法。

16.4 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

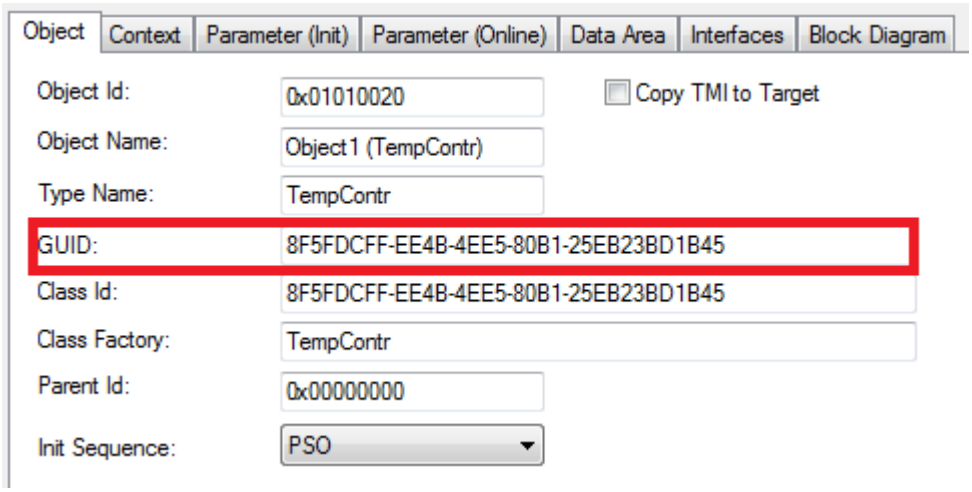
要将现有的 TcCOM 模块添加到 TwinCAT 组态中，需要由 TwinCAT 检测到这些模块。使用如下方法之一可达此目的：

- 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 模块可以由其 GUID或者名称 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 模块的属性页面确定。



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gai该TcCOM 模块已经在 TwinCAT 中注册并可被检测到 。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#)：

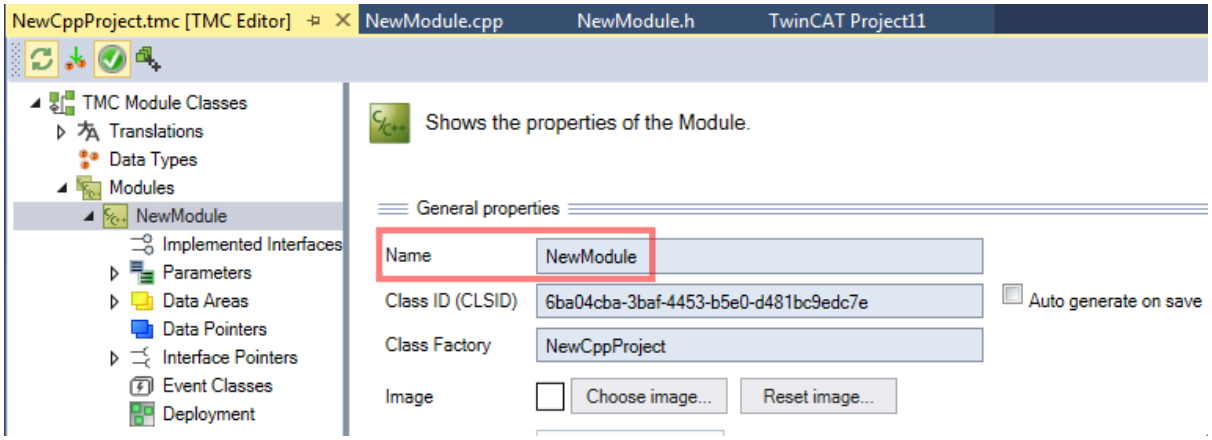
```
Dictionary<string,Guid> tcomModuleTable = new Dictionary<string,Guid>();
tcomModuleTable.Add("TempContr",Guid.Parse("{8f5fdcff-ee4b-4ee5-80b1-25eb23bd1b45}"));
ITcSmTreeItem tempController = tcComObjects.CreateChild("Test", 0, "",
tcomModuleTable["TempContr"]);
```

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。

- 此名称可用于将 TcCOM 模块通过 ITcSmTreeItem::CreateChild() 方法添加到 TwinCAT 组态中。名称可在 TwinCAT XAE 中通过 TMC 编辑器确定。



- 使用如下方法可达此目的：

代码片段 (C#)：

```
ITcSmTreeItem tempController = tcComObjects.CreateChild("Test", 1, "", "NewModule");
```

代码片段 (Powershell)：

```
$tempController = $tcComObjects.CreateChild("Test", 0, "", "NewModule")
```

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#)：

```
ITcSysManager3 sysManager3 = sysManager as ITcSysManager3;

ITcModuleManager3 moduleManager = sysManager3.GetModuleManager() as ITcModuleManager3;
foreach (ITcModuleInstance2 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，因此以下类型转换也将有效：

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。以下代码片段展示如何激活参数“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。以下代码片段展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#)：

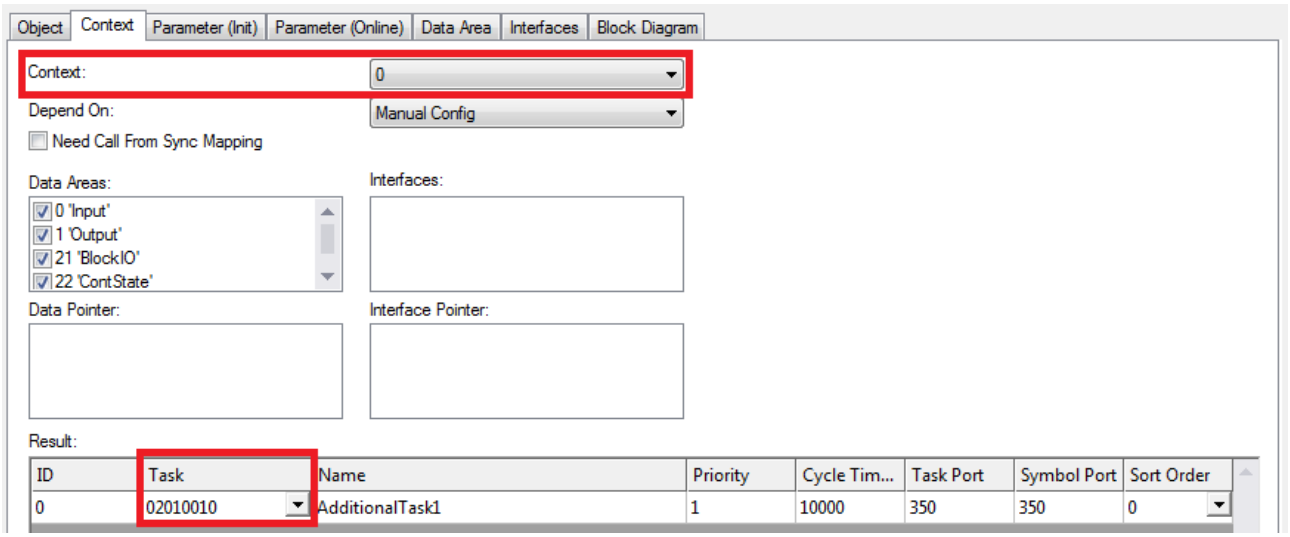
```
bool activateCS = true;
// First step: Read all Data Areas of a TcCOM module instance
string tempControllerXml = tempController.ProduceXml();
XmlDocument tempControllerDoc = new XmlDocument();
tempControllerDoc.LoadXml(tempControllerXml);
XmlNode sourceDataAreas = tempControllerDoc.SelectSingleNode("TreeItem/TcModuleInstance/Module/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

通过使用常规的Automation Interface 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。

16.5 Third-party components

This software contains third-party components.
Please refer to the license file provided in the following folder for further information:
C:\Program Files(x86)\Beckhoff\LegalTwinCAT-XAE-CppPlatformWizard

C:\Program Files(x86)\Beckhoff\LegalTwinCAT-XAE-PublicSDK

Trademark statements

Beckhoff®, ATRO®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, MX-System®, Safety over EtherCAT®, TC/BSD®, TwinCAT®, TwinCAT/BSD®, TwinSAFE®, XFC®, XPlanar® and XTS® are registered and licensed trademarks of Beckhoff Automation GmbH.

Third-party trademark statements

CANopen and CANopen FD are registered trademarks of CAN in AUTOMATION - International Users and Manufacturers Group e.V.

DALI, DALI-2, D4i, DALI+, and DiiA are trademarks in various countries in the exclusive use of the Digital Illumination Interface Alliance.

DSP System Toolbox, Embedded Coder, MATLAB, MATLAB Coder, MATLAB Compiler, MathWorks, Predictive Maintenance Toolbox, Simscape, Simscape™ Multibody™, Simulink, Simulink Coder, Stateflow and ThingSpeak are registered trademarks of The MathWorks, Inc.

Excel, IntelliSense, Microsoft, Microsoft Azure, Microsoft Edge, PowerShell, Visual Studio, Windows and Xbox are trademarks of the Microsoft group of companies.

Intel, the Intel logo, Intel Core, Xeon, Intel Atom, Celeron and Pentium are trademarks of Intel Corporation or its subsidiaries.

更多信息:
www.beckhoff.com/te1000

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Germany
电话号码: +49 5246 9630
info@beckhoff.com
www.beckhoff.com